CAD-based CFD shape optimisation using discrete adjoint solvers

Shenren Xu
School of Engineering and Materials Science
Queen Mary, University of London

A thesis submitted for the degree of
Doctor of Philosophy

September 22, 2015

Acknowledgements

This thesis would not have been possible without the support of my research supervisor Dr. Jens-Dominik Müller, who gave me sufficient freedom in exploring different research directions throughout my doctoral study. I thank Prof. Mike Giles and Prof. Shahrokh Shahpar for the constructive suggestions during the examination which has helped improve the quality of this thesis substantially. I am thankful for the financial support from both Queen Mary University of London and Rolls Royce. I would particularly like to thank Dr. Marcus Meyer and David Radford from Rolls Royce for the technical advice and support for the use of HYDRA code and for providing the test cases. I am grateful for the love and support of my family, colleagues and friends.

Statement of Originality

I, Shenren Xu, confirm that the research included within this thesis is my own work or that where it has been carried out in collaboration with, or supported by others, that this is duly acknowledged below and my contribution indicated. Previously published material is also acknowledged below.

I attest that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge break any UK law, infringe any third party's copyright or other Intellectual Property Right, or contain any confidential material.

I accept that the College has the right to use plagiarism detection software to check the electronic version of the thesis.

I confirm that this thesis has not been previously submitted for the award of a degree by this or any other university.

Signature: Shenren Xu

Date: September 22, 2015

Abstract

Computational fluid dynamics is reaching a level of maturity that it can be used as a predictive tool. Consequently, simulation-driven product design and optimisation is starting to be deployed for industrial applications. When performing gradient-based aerodynamic shape optimisation for industrial applications, adjoint method is preferable as it can compute the design gradient of a small number of objective functions with respect to a large number of design variables efficiently. However, for certain industrial cases, the iterative calculation of steady state nonlinear flow solver based on the Reynolds-averaged Navier–Stokes equations tends to fail to converge asymptotically. For such cases, the adjoint solver usually diverges exponentially, due to the inherited linear instability from the non-converged nonlinear flow. A method for stabilising both the nonlinear flow and the adjoint solutions via an improved time-stepping method is developed and applied successfully to industrial relevant test cases. Another challenge in shape optimisation is the shape parametrisation method. A good parametrisation should represent a rich design space to be explored and at the same time be flexible to take into account the various geometric constraints. In addition, it is preferable to be able to transform from the parametrisation to a format readable by most CAD software, such as the STEP file. A novel NURBS-based parametrisation method is developed that uses the control points of the NURBS patches as design variables. In addition, a test-point approach is used to impose various geometric constraints. The parametrisation is fully compatible with most CAD software. The NURBS-based parametrisation is applied to several industrial cases.

Contents

List of Figures

Nomenclature

Flow equation

U	conservative flow variable vector
Q	source term for Navier–Stokes equations
t	time
Ω	control volume for integration
$\partial\Omega$	boundary of control volume for integration
x_i	coordinates of the center of control volumes
$\vec{n}$	normal vector of control volume boundary surfaces
ρ	density
$\vec{v}$	velocity vector
p	pressure
T	temperature
H	total enthalpy
E	total energy
$\vec{\Theta}$	heat flux
$\bar{\bar{\tau}}$	stress tensor
F_c	convective flux
F_v	viscous flux
κ	thermal diffusivity
μ	dynamic viscosity
δ_{ij}	Kronecker delta
$\tilde{\nu}$	Spalart–Allmaras turbulence variable
R_i	flow residual of the i-th control volume
V_i	volume of the i-th control volume
$\mathcal{A}$	time stepping operator
U_L	reconstructed left state
U_R	reconstructed right state
$\Delta\vec{S}$	flux face area with orientation
A	flow Jacobian matrix
A_{Roe}	flow Jacobian matrix with Roe-averaged variables
$\tilde{\rho}$	Roe-averaged density
$\tilde{u}, \tilde{v}, \tilde{w}$	Roe-averaged x, y and z velocity
$\tilde{H}$	Roe-averaged total enthalpy
$\lambda_1, \lambda_2, \lambda_{3,4,5}$	eigenvalues of the flow Jacobian matrix
c	speed of sound

δ	entropy fix threshold
$\vec{r}_{i,c}, \vec{r}_{j,c}$	vector from i/j-th node to the flux face center
∇U	gradient of flow variables
$\mathbf{B}$	correction matrix for boundary nodes with strong boundary condition
$\vec{v}_{ghost}$	velocity of ghost cell for boundary nodes with weak boundary condition

Time stepping

β	implicit under-relaxation factor
σ	Courant number
Δt_i^{max}	maximal local time step of the i-th control volume according to the spectrum of the local flow Jacobian matrices
ΔU_i^n	flow variable update for the i-th control volume at the n-th pseudo time step
$\mathbf{P}_{i,i}$	preconditioning matrix for the i-th control volume
α_1, α_m	Runge-Kutta stage coefficients

Multigrid

h	fine grid
H	coarse grid
R^h, R^H	residual on fine/coarse grid
f^h, f^H	source term on fine/coarse grid
U^h, U^H	flow solution on fine/coarse grid
E^h, E^H	solution error on fine/coarse grid
I_h^H	flow solution transfer operator from fine to coarse grid (restriction)
$\hat{I}_h^H$	residual transfer operator from fine to coarse grid (prolongation)
I_H^h	flow solution transfer operator from coarse to fine grid (prolongation

Adjoint solver

$R^{(I)}$	nonlinear flow residual based on first order spatial discretisation
$R^{(II)}$	nonlinear flow residual based on second order spatial discretisation
$J(U, \alpha)$	cost function
α_i	i-th design variable
$\mathcal{I}$	operator for taking the imaginary part of a complex number
X, X_s	volume and surface node coordinates
$\mathbf{L}$	linearised flow equation operator (the flow Jacobian matrix)
$\mathbf{L}^T$	adjoint flow equation operator (the transposed flow Jacobian matrix)
u	linear perturbation variable
v	adjoint variable
$\tilde{U}$	exact solution to the discretised nonlinear flow solution
$\epsilon_{flow}^n, \epsilon_{adj}^n$	error of the nonlinear flow and adjoint solutions at the n-th pseudo time step
$\mathbf{L}, \mathbf{D}, \mathbf{U}$	lower, upper triangular and diagonal matrices from the LU decomposition
$\mathbf{L}, \mathbf{U}$	lower and upper triangular matrices from the incomplete LU-factorisation
$\mathcal{K}_m$	Krylov subspace of dimension m

Mesh deformation

$[K_{ij}]_{3\times 3}$	stiffness matrix connecting nodes i and j
l	distance between nodes i and j
δX_j	the displacement of node j to be solved
$\delta \tilde{X}_j$	the known displacement of node j
σ, ϵ	stress/strain tensor for linear elasticity
μ, λ	Lamé constants
E, ν	Young's modulus and Poisson's ratio
$d(x)$	wall distance

CAD-based parametrisation

P	coordinates of NURBS control points
u, v	parametric variables of the surface mesh points
$B(u, v)$	NURBS rational basis functions
G_0, G_1	G_0 and G_1 constraint functions
δP	perturbation of NURBS control points
$\mathbf{C}$	derivative of continuity constraint function with respect to control points
P_{tot}	total pressure
C_T^m, C_R^m	trailing edge thickness/radius constraint function at m-th design iteration
η	turbine stage efficiency
ϕ	turbine capacity
χ	turbine stage reaction ratio

Chapter 1

Introduction

Computational fluid dynamics (CFD) has reached a level of maturity that simulation-driven‘ product design and optimisation is becoming a reality. With the realisation of predictive CFD simulations, CFD shape optimisation, is starting to be widely deployed for industrial applications such as in turbomachinery [46, 53] and in automotive industries [70]. One way of performing CFD shape optimisation is to use non-gradient based algorithms where only the value of the objective function is needed at each optimisation step. Therefore, many simulations are performed until the global minimum is gradually reached [96]. This approach is easy to set up and a large number of different candidate designs could be evaluated simultaneously, therefore it is quite popular for industrial applications. Alternatively, gradient-based optimisation, using both the function value and its gradient to improve the design, can be used to find local optima more rapidly. One major drawback of the gradient-based approach is it is likely to be trapped in local optima and thus fail to find the global. Therefore, it is best when combined with a non-gradient based approach in order to efficiently search for the global optimum [54, 25].

This work focuses on the gradient based optimisation method only, since for many industrial applications, such as shape optimisation of turbomachinery components, aeroplane wings and fuselages where the baseline shapes are believed to be near optimal, a large shape change is neither expected nor desirable. For these applications, it would be more efficient to use gradient-based optimisation methods. Nevertheless, it does not imply that gradient-based approach is restricted to produce a very small design deviation. In fact, the constantly seen small design change in literature using gradient-based method could be attributed to either the limitation of the mesh deformation capability or the tight constraints that the improved design has to meet.

1.1 Gradient calculation using adjoint methods

The key to successfully performing a gradient-based optimisation is the efficient and robust calculation of the gradient. This is particularly difficult for CFD optimisation based on Reynolds averaged Navier-Stokes (RANS) equations since a nonlinear equation with tens of millions of unknowns need to be iteratively solved. To reduce the computational cost, one could either reduce the order of RANS equation on both continuous and discrete level [7], or lower the fidelity of the flow model, e.g., use potential flow instead of RANS [74]. However, to better harness the advantage of high-fidelity simulation tools and the even increasing computational resource, it is probably wise to base the optimisation on the full order RANS model and aim to develop efficient methods for computing the corresponding gradient. One simple way to calculate the gradient is to use finite differences. There are two issues with this approach. The first difficulty is in choosing a suitable step size when strongly nonlinear terms are present such as shock and turbulence source terms. Tangent linear solver or complex variable method could both alleviate the accuracy issue associated with the difficulty in finding a suitable step size. The second issue, preventing all three methods (finite difference, tangent linear and complex variable method) to be widely used for industrial applications is that the computational cost all scales linearly with the number of design variables which is routinely in the order of a few thousands. For typical industrial applications, the number of objective functions are usually only a few, while the number of design variables is usually much larger. For these cases, the most efficient way of calculating the gradient is to use the adjoint method. The computational cost of computing the gradient scales linearly with number of the objective functions and is almost independent of the number of the design variables. This is thus an enabling feature for gradient-based high-fidelity CFD shape optimisation.

1.2 Robustness issues of adjoint solvers

A major obstacle in using the adjoint solver for industrial applications is the lack of robustness of the adjoint solver for flows of practical interest, due to the complexity of either the geometry or the flow physics itself. The steady flow solver may not always be able to converge asymptotically, which is usually not a problem for flow analysis, as long as the objective function has converged to engineering accuracy. However, the adjoint solver is very sensitive to the nonlinear flow solver convergence. The adjoint equation, derived by linearising the nonlinear flow equation about an equilibrium point, inherits the linear stability/instability of the nonlinear flow. The adjoint solver using the same time discretisation as the nonlinear flow solver, would either asymptotically converge or diverge. The linear stability of the resulting system depends on both the spatial and temporal discretisations, which, together, is reflected mathematically in the spectrum of the system matrix of the linearised adjoint equation. To this end, it is not distinguished whether the instability is of physical or numerical origin, as it is a result of the interplay of both. The linear instability is usually qualitatively associated with certain flow phenomena such as an oscillating shock wave and an unstable flow separation point while it also depends on many numerical aspects such as the mesh and the discretisation. For example, flow around an aerofoil with a blunt trailing edge may well be expected to have vortices shed periodically from the trailing edge. However, this could be easily suppressed by deliberately coarsening the mesh around the trailing edge or by using a large CFL number to average out the physical oscillation.

A few solutions have been proposed to stabilise the linearised equation, when the nonlinear flow solver fails to asymptotically convergence. In [13], generalized minimal residual (GMRES) method is proposed to solve the linearised equation, instead of using the same time-marching scheme as the nonlinear flow. GMRES does not diverge even in the presence of outliers. However, its convergence does significantly slow down for stiff system. Convergence difficulty of GMRES is constantly encountered in industrial applications, although rarely reported in literature. Another method that has been proposed for stabilisation is recursive projection method (RPM) [15]. RPM uses power method to identify the most unstable modes (ones that after a large number of iterations start to exponentially diverge in distinct rate) and applies Newton step to those unstable modes while the existing time-marching is applied to the remaining stable system. The difficulty in applying RPM to large industrial cases is that it takes many iterations to identify the unstable modes, the number of which is unknown a priori, leading to large memory requirement and long CPU time. Similar to RPM, proper-orthogonal-decomposition (POD) technique [26] has been proposed as an alternative method. The stabilisation method using POD was originally proposed to stabilise the flow solver, but it is straightforward to use the method for the stabilisation of the corresponding linear and adjoint solvers. In [26], POD stabilisation method is demonstrated on a small two-dimensional aerofoil case where the oscillation is believed by the author to be too small to be representative of any oscillation seen in realistic three dimensional industrial cases.

In this thesis, a new stabilisation method for stabilising the adjoint solution for large industrial cases are developed. The stabilisation method is based on an improved time-stepping scheme that not only improves the convergence of the steady state nonlinear flow solution, but also consequently improves the adjoint convergence. Compared to the existing methods, because it eliminates the linear instability by improving numerical scheme of the nonlinear flow solver, rather then use a better linear solver that tolerate the linear instability.

1.3 Shape parametrisation methods

In addition to a robust adjoint solver, another key aspect of industrial shape optimisation is the parametrisation. A versatile parametrisation should be able to parametrise complex shapes and be able to accommodate various constraints, i.e., geometric, manufacturing, etc. Parametrisation determines the design space and thus the optimal shapes. The shape parametrisation methods can be broadly categorised into mesh-based and geometry-based. The mesh-based parametrisation uses the coordinates of CFD mesh points as the design variable, while the geometry-based one uses the underlying geometric parameters as design variables. It is usually not up to the flow analyst to decide which parametrisation to use as in practice, parametrisation is chosen at the design stage. Since one parametrisation in general cannot be switched to another without loss of accuracy, all the engineers working on the same geometry would need to use the same parametrisation whenever possible. Therefore, an important criterion in choosing a parametrisation method is its compatibility with the industrial work flow. Computer-Aided-Design (CAD) software is now routinely used for industrial shape design in aeronautical and automotive industries, thus ideally, the parametrisation method should be compatible with the CAD software to streamline the shape optimisation process. Here 'compatible' means that the parametrisation chosen for design optimisation should be able to be interchangeable with CAD-based parameters.

The optimisation result strongly depends on the constraints imposed. A good parametrisation method should either have the constraint implicitly built into the parametrisation itself or allow the various constraints to be imposed relatively easily.

1.4 Flow and adjoint solvers used in this work

In this thesis, three different nonlinear flow solvers and their accompanying discrete adjoint solvers are used for various parts of the work.

HYDRA

HYDRA [51] is a suite of finite volume method based CFD codes that solve the nonlinear compressible RANS equations [63] and its discrete adjoint [31] using block-Jacobi preconditioned multi-stage Runge–Kutta time stepping on unstructured meshes. It is an industrial CFD solver that has been extensively validated, particularly for turbomachinery applications. HYDRA uses automatic differentiation tool Tapenade, developed at INRIA, to generate the linear and adjoint versions of the key nonlinear functions within the code. The code is used for the results in chapter 4 for robust adjoint solver development and in chapter 6 for the CAD-based optimisation of a one-stage turbine.

MGOPT

MGOPT is an in-house compressible RANS flow and discrete adjoint solver developed at Queen Mary University of London [17]. It is similar to HYDRA, but is implemented in Fortran 90 which then allows modules and derived data types to simply the code. The discrete adjoint also uses Tapenade and thus the adjoint code is automatically generated at compiling time to inherit all the updates to the nonlinear flow solver. MGOPT has the same core algorithm as HYDRA regarding the spatial and temporal discretisations, but is much simpler, as a research code. Therefore, it is used as a test bed for developing new algorithms. The JT-KIRK implicit time stepping algorithm, to be explained in chapter 4, is first developed in MGOPT, validated, and then reimplemented in HYDRA. The flow and adjoint solver theories, implementation notes and the validation results in chapters 2 and 3 are based on MGOPT.

GPDE

GPDE is an in-house CFD solver for solving incompressible RANS flows and its discrete adjoint [44]. Different from HYDRA and MGOPT, GPDE uses SIMPLE algorithm to iterate the solution. Tapenade is also used for automatically generating the differentiated subroutines for the discrete adjoint solver. This code is used for computing the nonlinear flow, the adjoint as the surface sensitivity for the duct optimisation in chapter 7 since the flow condition is low speed for which an incompressible solver is more suitable.

1.5 Summary

In this thesis, the two major challenges in applying gradient-based shape optimisation to industrial applications are addressed: (i) the efficient and robust computation of the gradient and (ii) the efficient and versatile shape optimisation method. The remainder of the thesis is structured as follows. In chapter 2, the flow solver is explained in terms of the mathematical formulation, algorithmic development and some implementation details. In chapter 3, the adjoint method is explained along with the development of an adjoint solver for compressible RANS equations. The lack of robustness of an adjoint solver that is currently widely used for turbomachinery applications is then explained in chapter 4 with the proposed method to stabilise the adjoint solver. The proposed method is applied to four industrial test cases relevant to turbomachinery industry to demonstrate the enhanced robustness and the efficiency of both the nonlinear flow and adjoint solvers. Mesh deformation technique is explained in chapter 5 and CAD-based shape parametrisation method is discussed in chapter 6. The adjoint-based shape optimisation using the CAD-based parametrisation is then performed for two industrial cases, an air duct from automotive industry in chapter 7 and a high pressure turbine stage from turbomachinery industry in chapter 8. Finally, chapter 9 summarises the main findings of this work and some directions for future work are proposed.

Chapter 2

The RANS flow solver

This chapter covers the mathematical formulation of the Reynolds-Averaged Navier-Stokes (RANS) equations and the numerical method to iteratively solve them, including both the spatial discretisation and temporal integration methods.

2.1 The RANS flow equations

The governing equations for the compressible flow are based on the conservation law of mass, momentum and energy. For a stationary control volume Ω with boundary $\partial\Omega$, its integral formulation is

$$\frac{\partial}{\partial t}\int_{\Omega} U dv + \oint_{\partial\Omega} (F_c - F_v) \cdot n ds = \int_{\Omega} Q dv \tag{2.1}$$

where Q is the source term. The conservative variables U, convective flux F_c and viscous flux F_v are defined as

$$U = \rho \begin{bmatrix} 1 \\ \vec{v} \\ E \end{bmatrix}, F_c = \rho\vec{v}\cdot\vec{n} \begin{bmatrix} 1 \\ \vec{v} \\ H \end{bmatrix} + \begin{bmatrix} 0 \\ p\vec{n} \\ 0 \end{bmatrix}, F_v = \begin{bmatrix} 0 \\ \bar{\bar{\tau}}\cdot\vec{n} \\ \vec{\Theta}\cdot\vec{n} \end{bmatrix}$$

and

$$\vec{\Theta} = \bar{\bar{\tau}}\cdot\vec{v} + \kappa\nabla T$$

where ρ is density, $\vec{v}$ is the velocity, p is the static pressure, T is the temperature, κ is thermal conductivity and and $\vec{n}$ is the unit normal vector on one face of the control volume. The stress tensor $\bar{\bar{\tau}}$ for Newtonian fluid under Stokes' hypothesis is

$$\tau_{ij} = \mu(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}) - \frac{2\mu}{3}\frac{\partial U_k}{\partial x_k}\delta_{ij}$$

with δ_{ij} being the Kronecker delta, μ the dynamic viscosity.

2.2 Spalart-Allmaras turbulence model

Turbulence is modelled with the Spalart–Allmaras (SA) one-equation turbulence model [89] in this thesis. The SA model employs a transport equation for the eddy-viscosity variable $\tilde{\nu}$:

$$\begin{aligned}\frac{\partial \tilde{\nu}}{\partial t} + \frac{\partial}{\partial x_j}(\tilde{\nu} v_j) &= C_{b1}(1 - f_{t2})\tilde{S}\tilde{\nu} \\ &+\frac{1}{\sigma}\left\{\frac{\partial}{\partial x_j}\left[(\nu_L + \tilde{\nu})\frac{\partial \tilde{\nu}}{\partial x_j}\right] + C_{b2}\frac{\partial \tilde{\nu}}{\partial x_j}\frac{\partial \tilde{\nu}}{\partial x_j}\right\} \\ &-\left[C_{w1}f_w - \frac{C_{b1}}{k^2}f_{t2}\right]\left(\frac{\tilde{\nu}}{d}\right)^2 \\ &+f_{t1}||\Delta\vec{v}||_2^2\end{aligned}$$

The four terms on the right hand sie are eddy-viscosity production, diffusion, near-wall turbulence destruction and transition source of turbulence. $\nu_L = \mu_L/\rho$ is laminar kinematic viscosity and d denotes the distance to wall. As a simplified implementation, the transition term $f_{t1}||\Delta\vec{v}||_2^2$ can be ignored, assuming the flow is fully turbulent.

The integral form of the eddy-viscosity transport equation is

$$\frac{\partial}{\partial t}\int_{\Omega}\tilde{\nu}d\Omega + \oint_{\partial\Omega}(F_{c,T} - F_{v,T})dS = \int_{\Omega}Q_T d\Omega$$

where the convective flux $F_{c,T}$, the viscous flux $F_{v,T}$ and the source term $Q_T$are

$$F_{c,T} = \tilde{\nu}\vec{v}$$

$$F_{v,T} = \frac{1}{\sigma}(\nu_T + \tilde{\nu})\frac{\partial \tilde{\nu}}{\partial x_j}n_j$$

$$Q_T = C_{b1}(1 - f_{t2})\tilde{S}\tilde{\nu} + \frac{C_{b2}}{\sigma}\frac{\partial \tilde{\nu}}{\partial x_j}\frac{\partial \tilde{\nu}}{\partial x_j} - \left[C_{w1}f_w - \frac{C_{b1}}{k^2}f_{t2}\right]\left(\frac{\tilde{\nu}}{d}\right)^2$$

Initial and Boundary condition for eddy-viscosity $\tilde{\nu}$: initial value of $\tilde{\nu}$ is set as $\tilde{\nu} = 0.1\nu_L$. The same value is also specified at inflow boundaries. At the outflow boundary, $\tilde{\nu}$ is extrapolated from the interior domain. At non-slip solid walls, $\tilde{\nu} = 0$. At slip walls (such as symmetric plane), $\frac{\partial \tilde{\nu}}{\partial n} = 0$

2.3 Discretisation of the flow equations

When the finite volume method (FVM) is used on unstructured meshes to solve the fluid equation, each control volume is assigned a vector of flow variable to represent the average of the continuous distribution of the flow variable over the whole control volume

$$U_i := \frac{\int_{\Omega_i} U d\sigma}{\int_{\Omega_i} d\sigma} = \frac{\int_{\Omega_i} U d\sigma}{V_i}$$

with V_i being the volume of control volume i. Using the method of lines, the integral form Eq. (2.1) can be written as

$$V_i \frac{dU_i}{dt} + R_i(U) = 0 \tag{2.2}$$

where $R_i(U)$ is the residual term that represents the spatial discretisation of the convective and viscous fluxes. The steady state solution is one state of U that satisfies the nonlinear flow equation

$$R_i(U) = 0 \tag{2.3}$$

for every control volume and it can be reached via

$$U^{n+1} = U^n - \mathcal{A}(R(U))$$

where the operator $\mathcal{A}$ represents the time stepping method used.

The formulation of residual $R(U)$ depends on the spatial discretisation and thus determines the accuracy of the converged solution, while the time stepping operator $\mathcal{A}$ represents the temporal discretisation which determines how the intermediate solution evolves to the final converged solution. When the method of lines is used, the spatial and temporal discretisations can be devised separately.

2.3.1 Spatial discretisation

For a node-centred finite volume discretisation, the residual for each vertex (the center of one control volume) is computed by looping over all the edges from the vertex. Hence the essential step is to determine how to calculate the flux for each face of a control volume.

2.3.1.1 Convective flux

For a flux face with left and right states U_L and U_R and the oriented face $\Delta\vec{S}(=\vec{n}\Delta S)$, the flux is formally

$$F = F(U_L, U_R, \Delta\vec{S}) \tag{2.4}$$

Numerous flux schemes exist, each of which has pros and cons. One of the most robust shock-capturing scheme, Roe's flux scheme [80], is used in this work. Roe's flux follows the thinking of a characteristic solver and use the absolute-valued Jacobian matrix to provide unwinding effect:

$$F = \frac{1}{2}(F(U_L) + F(U_R)) - \frac{1}{2}|A_{Roe}|(U_R - U_L) \tag{2.5}$$

where

$$|A_{Roe}| = R_\Lambda |\Lambda| R_\Lambda^{-1} \tag{2.6}$$

with Λ and R_Λ being the eigenvalues and right eigenvectors of the Jacobian matrix A_{Roe} using Roe-averaged interface state. The robustness and the shock-capturing capability of the Roe flux is attributed to the specially designed Roe-average interface state defined as

$$\begin{aligned}\tilde{\rho} &= \sqrt{\rho_L \rho_R}\\ \tilde{u} &= \frac{u_L\sqrt{\rho_L} + u_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}}\\ \tilde{v} &= \frac{v_L\sqrt{\rho_L} + v_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}}\\ \tilde{w} &= \frac{w_L\sqrt{\rho_L} + w_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}}\\ \tilde{H} &= \frac{H_L\sqrt{\rho_L} + H_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}}\end{aligned}$$

which are rigorously devised such that it recognises when a jump (not necessarily an infinitesimal one) of the state $U_R - U_L$ is a pure jump in one characteristic family only and thus produces the exact propagation velocity [52].

The Jacobian matrix using Roe-averaged values has five eigenvalues

$$\lambda_1 = V_n - c, \quad \lambda_2 = V_n + c, \quad \lambda_{3,4,5} = V_n \tag{2.7}$$

where

$$V_n = \vec{v} \cdot \vec{n} \tag{2.8}$$

and c is the local sound speed. For stagnation point and sonic point, the vanishing eigenvalue needs to be modified to be bounded away from zero. One of such fixes is Harten's entropy fix

$$\tilde{\lambda} = \begin{matrix} \dfrac{\lambda^2 + \delta^2}{2\delta} \quad \text{if} \quad \lambda < \delta \\ \\ \lambda \quad \text{if} \quad \lambda \geq \delta \end{matrix} \tag{2.9}$$

and the threshold δ can be set to some fraction of the local sound speed, e.g.,

$$\delta = \frac{c}{10} \tag{2.10}$$

For a first order accurate scheme, the left and right states U_L and U_R are the flow variables of the left and right nodes respectively. Second order accuracy can be achieved by assuming a linear distribution of the flow variable within each control volume, and in that case, the left and right states can be calculated by extrapolating from the node value to the face center using the gradient, i.e.,

$$U_L = U_i + (\nabla U)_i \cdot \vec{r}_{i,c} \qquad U_R = U_j + (\nabla U)_j \cdot \vec{r}_{j,c}$$

where i and j are the indices of the left and right nodes, and $\vec{r}_{i,c}$ and $\vec{r}_{j,c}$ are the vectors pointing from node i and j to the flux face center.

2.3.1.2 Viscous flux

Both the flow variables and their gradient are needed on the flux face in order to calculate the viscous flux. The face value is simply the average of the flow variables at the left and right nodes

$$U_f = \frac{1}{2}(U_i + U_j)$$

The gradient is first calculated for each node using either Green-Gauss or least-square and then averaged at the flux face

$$\overline{\nabla U}_{ij} = \frac{1}{2}(\nabla U_i + \nabla U_j)$$

To avoid the checker-board pattern oscillation which cannot be suppressed by the above gradient calculation, the gradient component along the face normal direction is replaced by the directional derivative

$$\left(\frac{\partial U}{\partial l}\right)_{ij} = \frac{U_j - U_i}{|x_i - x_j|} \tag{2.11}$$

and the unit vector $\vec{t}_{ij}$ along the line connection node i and j is

$$\vec{t}_{ij} = \frac{x_j - x_i}{|x_j - x_i|} \tag{2.12}$$

The modified gradient average on the face can then be written as

$$(\nabla U)_f = \overline{\nabla U}_{ij} - \left[\overline{\nabla U}_{ij} \cdot \vec{t}_{ij} - \left(\frac{\partial U}{\partial l}\right)_{ij}\right] \vec{t}_{ij} \tag{2.13}$$

2.3.2 Wall boundary conditions

The correct formulation and implementation of the wall boundary condition is critical for a robust flow solver. Two types of wall boundary treatments are considered: weak boundary condition and strong boundary condition.

2.3.2.1 Weak boundary condition

When a weak boundary condition is applied on wall boundaries, no explicit treatment for the wall node residual is needed because the wall effect is taken into account by constructing the flow variable at the ghost cell such that zero mass flux across the wall boundary is satisfied. For a boundary node with boundary face normal $\vec{n}$ and velocity $\vec{v}$, the ghost cell velocity is then set to

$$\vec{v}_{ghost} = \vec{v} - 2\vec{v} \cdot \vec{n}\vec{n}$$

for a slip wall, and set to

$$\vec{v}_{ghost} = -\vec{v}$$

for a no-slip wall. The density and pressure of the ghost cell will be the same as the wall node. For a more accurate result, wall curvature should be taken into account to obtain a more accurate pressure in the ghost cell, but it is beyond the scope of this work.

2.3.2.2 Strong boundary condition

The strong boundary condition explicitly enforces zero mass flux on the wall by setting the velocity component normal to the wall (for slip wall) or the whole velocity (for no-slip wall) to zero, and also set the corresponding residual component to zero, in order to avoid an update for those zeroed components. In both cases, the correction for residual is

$$R_{hardbc} = (I - \mathbf{B})R$$

where B is the correction matrix defined as

$$\mathbf{B}_{5\times5} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \vec{n} \otimes \vec{n} & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

for slip walls and

$$\mathbf{B}_{5\times5} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & I_{3\times3} & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

for no-slip walls The resulting nonlinear flow equations to solve become

$$\begin{aligned} (I - \mathbf{B})R(U) &= 0 \\ \mathbf{B}U &= 0 \end{aligned}$$

When a hard boundary condition is used, the force asserted by the fluid on the wall boundary is formulated as

$$\vec{F} = (p\vec{n} - \mathbf{B}R(U)) \cdot \vec{n}\Delta S$$

where p is the static pressure and ΔS is the area of the boundary surface associated with a particular boundary node.

HYDRA code used strong boundary condition for mesh points on both inviscid walls and viscous wall. However, if a sharp corner is present on inviscid walls, the surface normal of those points is set to zero, which essentially means weak boundary condition is applied.

2.3.3 Temporal discretisation

Once the residual is calculated for each control volume, it needs to be integrated in time in order to drive the solution to steady state. A generic time-marching scheme can be written for node i as

$$\frac{U_i^{n+1} - U_i^n}{\Delta t_i} V_i = -(1-\beta)R_i(U^n) - \beta R_i(U^{n+1})$$

where the coefficient $\beta > 0$ provides a blending between explicit and implicit time stepping. If an implicit scheme is used, the residual at time level $(n+1)$ is linearised using the Jacobian matrix at time level n to be

$$R_i(U^{n+1}) \approx R_i(U^n) + \left.\frac{\partial R_i}{\partial U_j}\right|_{U=U^n} (U_j^{n+1} - U_j^n)$$

and by replacing Δt_i with $\sigma \Delta t_i^{max}$, Eq. (2.14) becomes

$$\left(\frac{V_i}{\sigma \Delta t_i^{max}} + \beta \left.\frac{\partial R_i}{\partial U_j}\right|_{U=U^n}\right) \Delta U_j^n = -R_i(U^n) \tag{2.14}$$

where Δt_{max} is the maximum allowable time step that can be estimated by the wave speeds of the characteristics for all flux faces of each control volume, and σ is the CFL number.

When $\sigma = +\infty$, Eq. (2.14) becomes a Newton step, which gives quadratic convergence. However, this approach is rarely taken for large cases for the following reasons:

- Newton step is stable only in the vicinity of the stationary point
- The exact Jacobian is difficult, if not impossible, to compute
- The storage overhead for the exact Jacobian is prohibitive
- The resulting linear system is stiff and difficult to solve

Therefore, for practical cases, various modifications can be done to make the system less stiff, more robust and more memory efficient, such as

- Introduce a finite CFL number
- Use an approximate Jacobian matrix
- Solve the linear system approximately

One widely used algorithm is block-Jacobi preconditioned point-implicit solver, which is also used in the baseline solver in this work. For a block-Jacobi solver, the full exact Jacobian matrix is approximated by only keeping the diagonal blocks and basing the derivative on the 1st order nonlinear residual, and the time stepping becomes

$$\left(\frac{V_i}{\sigma \Delta t_i^{max}} + \beta \left.\frac{\partial R_i}{\partial U_i}\right|_{U=U^n}\right) \Delta U_i^n = -R_i(U^n) \tag{2.15}$$

which is usually stable even with CFL set to infinity, but the finite time step is kept for some cases with strong transient behaviour due to poor initialisation. For more details of the block-Jacobi algorithm implemented in the solver, refer to [64, 21, 63]. To facilitate the discussion of time integration, the LHS of both Eq. (2.14) and Eq. (2.15) is denoted by a preconditioning matrix $\mathbf{P}$, which is a function of CFL number, flow solution and blending coefficient, and it is a sparse matrix consisting of 5×5 blocks, each of which is defined as

$$\mathbf{P}_{i,i} = \frac{V_i}{\sigma \Delta t_i^{max}} + \beta \left.\frac{\partial R_i}{\partial U_i}\right|_{U=U^n}$$

and consequently the time stepping for node i becomes

$$\mathbf{P}_{i,i} \Delta U_i^n = -R(U^n)$$

The time marching of the flow equations is often combined with Runge–Kutta methods, with two major motivations. Traditionally, RK is used to provide additional stability and thus to allow a CFL number larger than unity to be used for explicit solvers [40]. The other is to efficiently damp the high frequency error [95], in order to increase the efficiency when combined with multigrid.

When an m-stage Runge–Kutta scheme is used, the solution U^n is updated as follows

$$\begin{aligned} U^{(0)} &= U^n \\ U^{(1)} &= U^{(0)} - \alpha_1 \mathbf{P}^{-1} R(U^{(0)}) \\ &\vdots \\ U^{(m)} &= U^{(0)} - \alpha_m \mathbf{P}^{-1} R(U^{(m-1)}) \\ U^{n+1} &= U^{(m)} \end{aligned}$$

For block-Jacobi time stepping, the preconditioning matrix is inverted by directly inverting each of the diagonal block matrices before the 1^{st} stage and the inverted matrix $\mathbf{P}^{-1}$

is stored to multiply with the residual vector at each stage to update the flow solution. The associated memory for storing the block-Jacobian matrices for all control volumes is only 5 times that of the flow solution and thus is deemed affordable.

2.4 Multigrid

Multigrid (MG) methods are a group of algorithms for solving partial differential equations using a hierarchy of grid levels. There are mainly two types of multigrid methods: geometric multigrid and algebraic multigrid:

- **Geometric multigrid (GMG)** formulates the hierarchy of discretisation on a series of successively coarsened meshes
- **Algebraic multigrid(AMG)** constructs the hierarchy of discretisation directly from the system matrix itself

AMG has the advantage of being used as a black-box preconditioner without much input from the user and thus has been widely used. GMG on the other hand, needs not only mesh coarsening algorithms to generate a hierarchy of meshes, but also properly constructed transfer operators between meshes. This causes some inconvenience in applying geometric multigrid, but also offers great flexibility: a good coarsening algorithm can coarsen the mesh by taking into account the features of the flow, such as boundary layers, and this will in turn make the multigrid much more efficient than the black-box approach of AMG. In this work, we use the geometric multigrid method and use software call 'hip' for mesh coarsening [64]. As an example, the unstructured mesh for NACA0012 airfoil is coarsened and shown in fig. 2.1

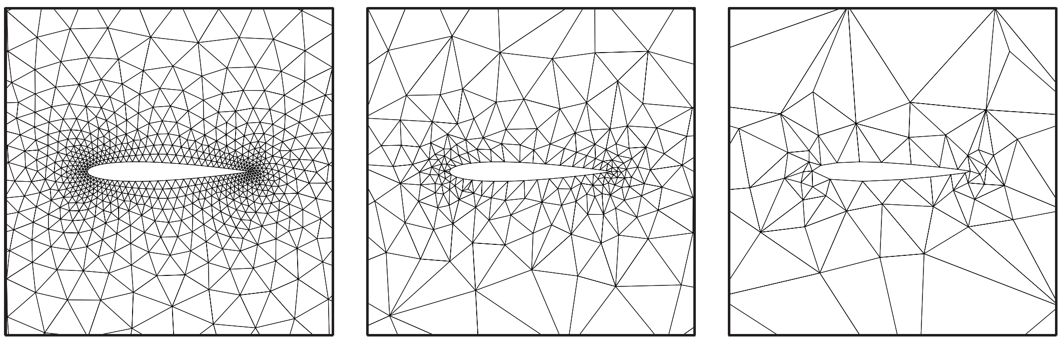

Figure 2.1: Three consecutive levels of multigrid meshes.

To explain the mathematical aspects of the multigrid, consider a nonlinear problem

$$R(U) = f$$

where R stands for a general differential operator. A fixed-point iterative solver updates the intermediate solution U^n iteratively until the residual, $f - R(U)$, is driven to zero:

$$U^{n+1} \leftarrow U^n + \mathcal{A}(f - R(U^n)) \tag{2.16}$$

where $\mathcal{A}$ denotes a time-stepping operator. For numerical computation, R and f need to be discretised for a particular computational mesh denoted by h, and we denote this discrete operator, i.e., the system matrix, and the source term by R^h and f^h. The exact solution $\hat{U}^h$ of the discrete system satisfies

$$R^h(\hat{U}^h) = f^h. \tag{2.17}$$

The multigrid methods solve the discrete system in the following procedure:

- Pre-smooth the solution on fine grid h:
  The intermediate solution U^h is updated as

  $$U^h \leftarrow U^h + \mathcal{A}(f^h - R^h(U^h)) \tag{2.18}$$

  An important feature of the explicit smoothing operator $\mathcal{A} \circ R^h$ is that it damps the high frequency error modes efficiently but is much less effective for low frequency ones. Therefore, the convergence using (2.18) quickly degenerates before reaching full convergence. Multigrid is designed exactly to remedy this. Denote the error on the fine grid as E^h

  $$E^h = \hat{U}^h - U^h \tag{2.19}$$

  which by definition satisfies

  $$R^h(U^h + E^h) = f^h \tag{2.20}$$

  Subtracting $R^h(U^h)$ from both sides yields

  $$R^h(U^h + E^h) - R^h(U^h) = f^h - R^h(U^h) = r^h \tag{2.21}$$

- Transfer of both the residual (r^h) and the solution U^h to a coarser grid H:
  Transferring both the residual r^h and the solution U^h to the coarse mesh H using transfer operators I_h^H and $\hat{I}_h^H$ (not necessarily the same) yields

  $$R^H(I_h^H U^h + E^H) - R^H(I_h^H U^h) = \hat{I}_h^H r^h = \hat{I}_h^H (f^h - R^h(U^h)) \tag{2.22}$$

  Note that the source term on the coarse grid, instead of simply $\hat{I}_h^H f^h$, becomes

  $$f^H = \hat{I}_h^H f^h - \hat{I}_h^H R^h(U^h) + R^H(I_h^H U^h) \tag{2.23}$$

Therefore, on the coarse mesh H, the discrete system to solve is

$$R^H(U^H) = R^H(I_h^H U^h + E^H) = \hat{I}_h^H f^h - \hat{I}_h^H R^h(U^h) + R^H(I_h^H U^h) \tag{2.24}$$

where U^H is initialized using the solution from the fine mesh $I_h^H U^h$. The same transfer is recursively carried out to an even coarser mesh until the coarsest mesh is reached with smoothing applied on each level:

$$U^H \leftarrow U^H + \mathcal{A}(f^H - R^H(U^H)) \tag{2.25}$$

- Prolong the correction on coarse mesh H to fine mesh h

$$U^h \leftarrow U^h + I_H^h(U^H - I_h^H U^h) \tag{2.26}$$

where I_H^h is the prolongation operator, which is usually the transpose of the solution restriction operator I_h^H.

The description above explains the procedure for a V-cycle multigrid process on multigrid meshes. For GMG, the time-stepping operator $\mathcal{A}$ on the coarse grids is constructed by applying the same spatial discretisation on different meshes. Usually, even when the 2nd order accurate spatial discretisation is used on the finest grid, only 1st order accurate discretisation is applied to all coarser levels, due to the compromised grid quality during the grid coarsening.

2.5 Flow solver validation

The in-house RANS flow solver MGOPT, for unstructured hybrid meshes accelerated by geometric multigrid, is applied to the following test cases for validation.

2.5.1 Turbulent flow over a 2D flat plate

The flow parameters and meshes are identical to [2]. Some key parameters are Reynolds number 5 million and Mach number 0.2. The domain is [-1/3m, 2m]×[0, 1m], with structured meshes refined near the wall and the leading edge located at (0m, 0m). The results for velocity profile at the end of the plate (x=2m) is compared with the NASA CFL3D code [1] shown in fig 2.2 with excellent agreement. 7 levels of multigrid meshes are used to accelerate the convergence. The SA variable contour plot for the converged flow solution is shown in fig.2.3.

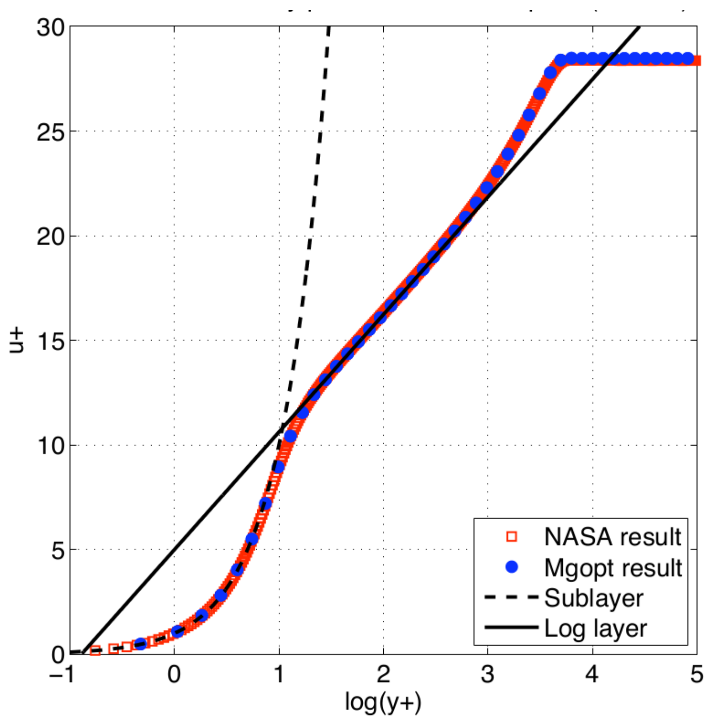

Figure 2.2: The non-dimensionalised stream-wise velocity profiles at location x=2m, from the in-house code MGOPT compared with NASA CFL3D code and analytic inner law and log law (kappa=0.41, B=5.0) [97].

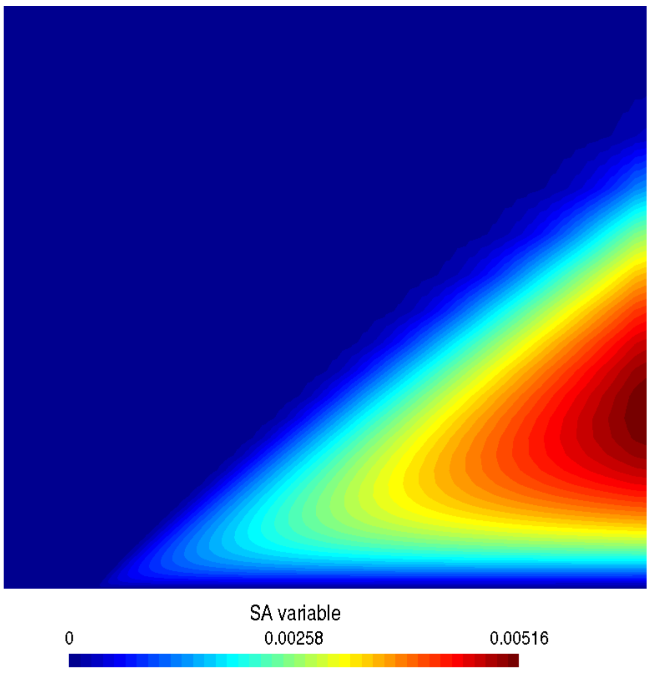

Figure 2.3: SA variable contour (stretched 50× in y direction).

2.5.2 Transonic turbulent flow around RAE2822 airfoil

Euler flow around a RAE2822 airfoil is computed with angle of attack 2.72 degrees at Mach number 0.75. The Reynolds number is 6.2 million. The fine grid mesh is shown in fig. 2.4. Five levels of multigrid meshes are used to accelerate the convergence. The results from MGOPT are compared with the results from ANSYS Fluent density based solver using similar setting and experimental result taken from [4] in fig. 2.5 with good agreement.

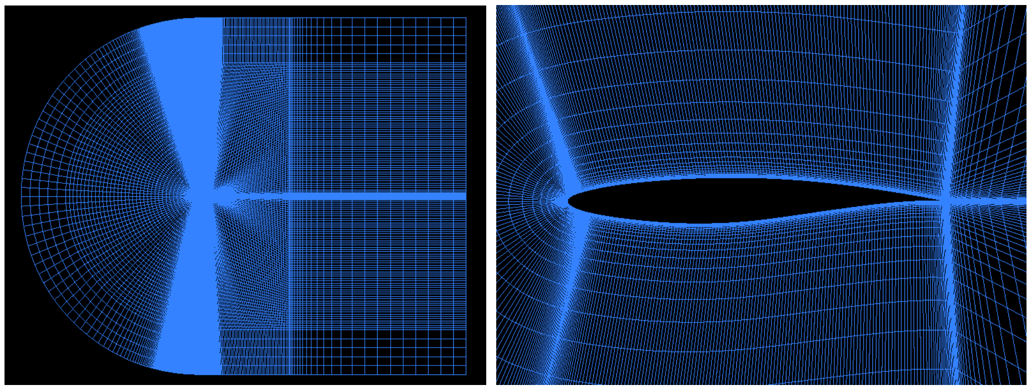

Figure 2.4: Fine grid mesh for RAE2822.

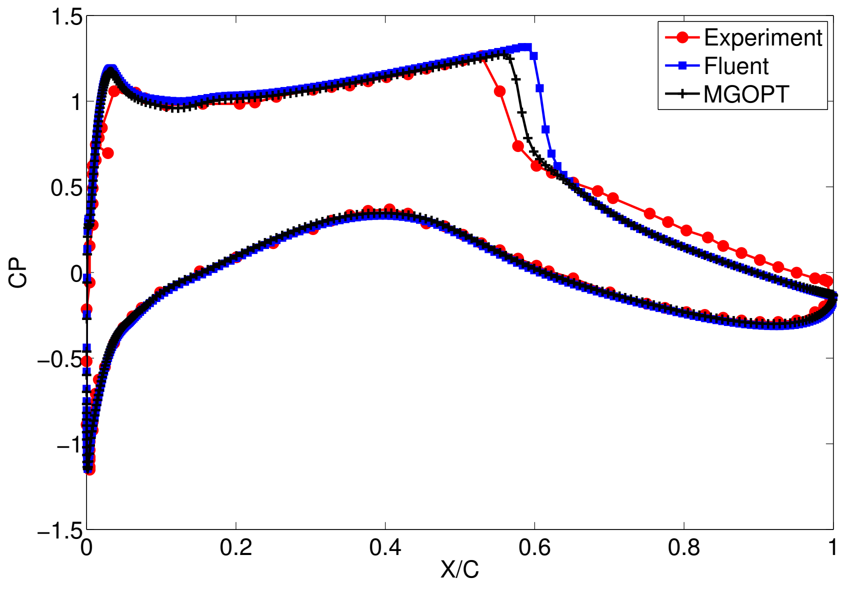

Figure 2.5: Pressure coefficient comparison for MGOPT, ANSYS Fluent and experimental data.

2.5.3 Transonic turbulent flow around an M6 wing

The transonic turbulent flow around a three dimensional Onera M6 wing at an angle of attack 3.06 degrees with Mach number 0.84 and Reynolds number 20 million is computed using MGOPT with Spalart–Allmaras turbulence model. The overall geometry is shown in fig. 2.6 along with the Mach contour illustrating the shock. For quantitative compar-

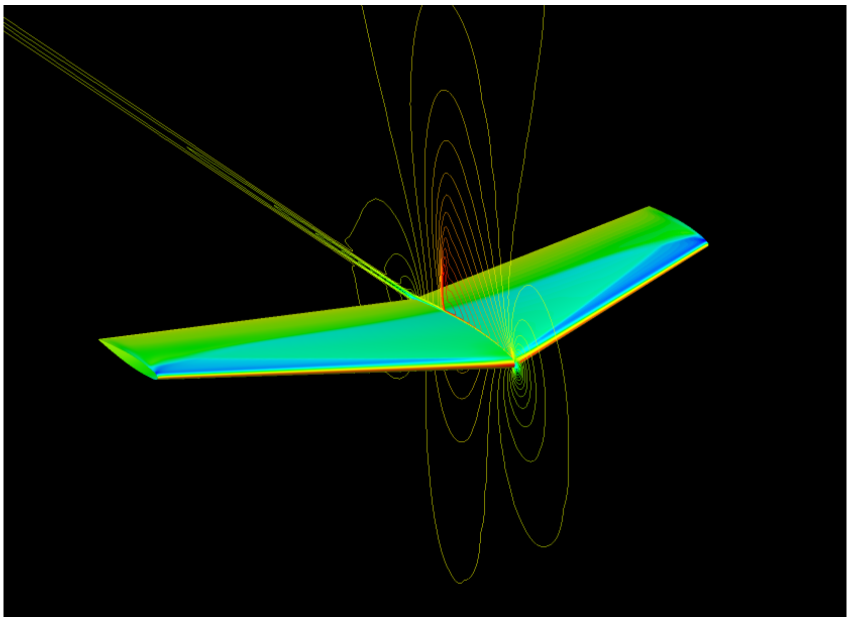

Figure 2.6: Onera M6 wing with pressure contour on surface and isolines for Mach number contour. The wing is mirror about the symmetric plane for illustrative purpose.

ison, the pressure coefficient distribution at different spanwise locations are compared with both the experimental results [4] and the ANSYS Fluent solver with similar settings in fig. 2.7.

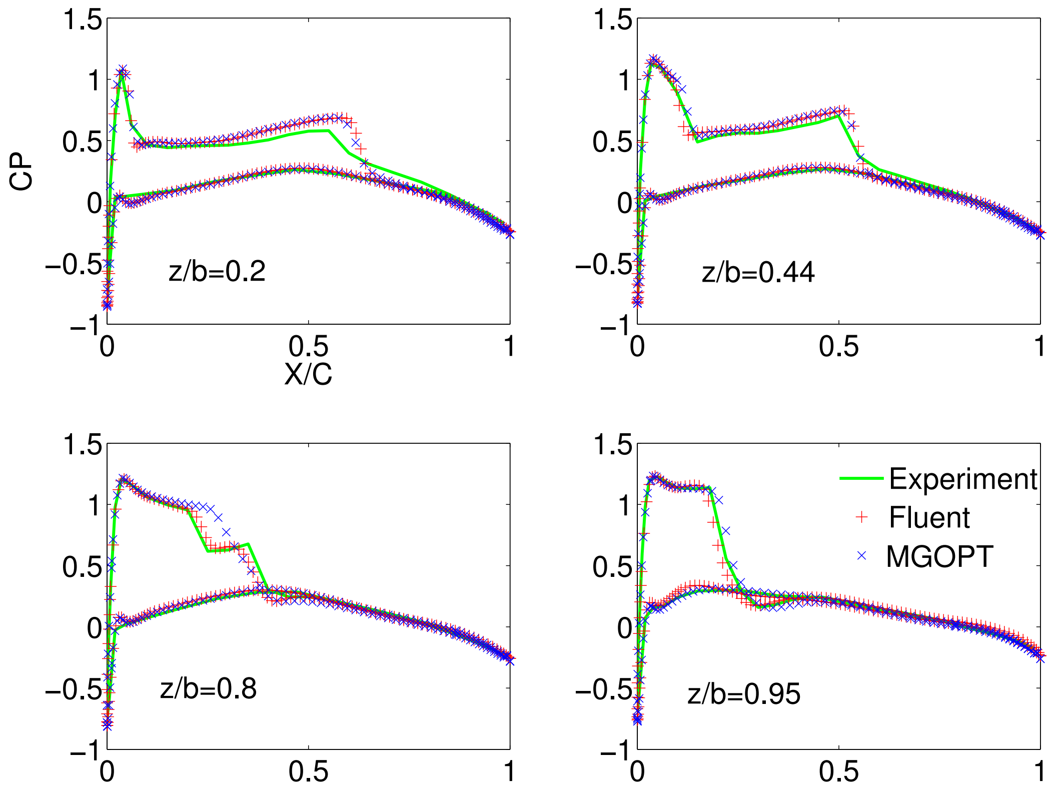

Figure 2.7: The pressure coefficient computed using ANSYS Fluent and MGOPT, compared with experimental data.

Chapter 3

The adjoint solver

3.1 Introduction

The adjoint solver is an essential part of the gradient-based CFD shape optimisation as it allows the efficient computation of the gradient of a cost function with respect to many design variables. There are two different approaches to compute the adjoint, namely, the discrete and the continous adjoint [32, 5, 71, 69]. The continuous adjoint approach first derives the adjoint partial differential equation (PDE) from the perspective of Lagrangian multiplier and then discretises it. The discrete adjoint approach directly transforms the discretised flow equations to form the discrete adjoint equation. The continuous approach offers the possibility to optimise and stabilise the adjoint solver by tailoring the discretisation according to the adjoint PDE, while the discrete approach ensures the exactness of the final design sensitivity with respect to the output of the flow solver. In this thesis, the discrete adjoint is used. Its derivation is discussed in sec. 3.2

To develop a discrete adjoint solver, one starts with an existing nonlinear solver. A straightforward way is to compute and store the exact flow Jacobian and solve the resulting transposed linear system. This is possible and suitable for simple two dimensional case [16]. However, for practical three dimensional cases, both the memory and runtime for this approach could be prohibitively expensive [24]. Alternatively, one can solve the adjoint equation using the same time-marching scheme of the nonlinear solver [28, 17], which is explained in sec.3.3. This approach does not require storing the exact flow Jacobian and solving the resulting stiff linear system. The advantage for the adjoint solver to mimic the time stepping of the nonlinear flow solver is that the convergence of the adjoint solver can be guaranteed when reusing the symmetric elements of the time-stepping of the nonlinear solver such as multigrid prolongation and restriction and correctly transposing any non-symmetric elements in the iterative operator such as preconditioners [17], as is shown in sec. 3.4.

3.2 The discrete adjoint

Let the cost function J be a function of both the flow solution U and the design variable α, and the goal is to minimise or maximise the cost function with constraint that the flow solution having to satisfy the flow equation with a particular boundary shape defined by any perturbed design $\alpha + \delta\alpha$. That is

$$\begin{aligned} &\underset{\alpha}{\text{minimize}} && J(U(\alpha), \alpha) \\ &\text{subject to} && R(U, \alpha) = 0 \\ &\text{and other constraints} && \end{aligned}$$

To evaluate the gradient of the cost functions with respect to a large number of design variables, one straightforward way is to use finite difference (FD) to perturb one design variable α_i using a small step size $\Delta\alpha_i$, and compute the gradient with respect to that particular design variable

$$\frac{dJ}{d\alpha_i} = \frac{J(U(\alpha + \delta\alpha), \alpha + \delta\alpha) - J(U(\alpha - \delta\alpha), \alpha - \delta\alpha)}{2\Delta\alpha_i} \tag{3.1}$$

where $\delta\alpha = (0, ..., \delta\alpha_i, ..., 0)$ and repeat the same for all other design variables. The drawback of FD are the gradient is extremely dependent on the step size. Moreover, the computational cost is prohibitive for large problems with many design variables.

The complex variable method [28] can circumvent the step size dependency issue. By replacing the real type variable with complex variable, and perturbing the imaginary part, the gradient can be computed as

$$\frac{dJ}{d\alpha_i} = \frac{\mathcal{I}(J(\alpha_i + j\epsilon))}{\epsilon} \tag{3.2}$$

where $\mathcal{I}$ means taking the imaginary part, j denotes the imaginary unit and ϵ is the step size. Unlike finite difference, the complex variable method does not suffer from the huge numerical error of subtracting two real numbers of small difference when the perturbation size is too small. The complex variable method provides a good tool for validating the sensitivity obtained with other approaches. However, the computational cost still scales with the number of variables and thus is not practical for computing the sensitivity of large scale problems.

In order to evaluate the final design sensitivity, one can linearise the cost function as

$$\frac{\partial J}{\partial \alpha} = \frac{\partial J}{\partial X}\frac{\partial X}{\partial X_s}\frac{\partial X_s}{\partial \alpha} = \frac{\partial J}{\partial \alpha} + g^T u \tag{3.3}$$

where

$$g^T = \frac{\partial J}{\partial U} \quad \text{and} \quad u = \frac{\partial U}{\partial \alpha} = \frac{\partial U}{\partial X}\frac{\partial X}{\partial X_s}\frac{\partial X_s}{\partial \alpha} \tag{3.4}$$

The two derivatives $\frac{dX}{dX_s}$ and $\frac{dX_s}{d\alpha}$ are the derivatives for volume mesh smoothing and surface mesh parametrisation respectively, which will be explained in more details in following chapters. The terms $\frac{\partial J}{\partial \alpha}$ and g^T denote the derivative of the cost function w.r.t. the design variable and the flow, which can be easily computed. The most difficult part is to compute u, which is the perturbation field from design variable perturbation. This term is implicitly determined by the flow equation constraint

$$R(U, \alpha) = 0 \tag{3.5}$$

which when linearised, yields

$$\mathbf{L}u = f \tag{3.6}$$

where

$$\mathbf{L} = \frac{\partial R}{\partial U} \quad \text{and} \quad f = -\frac{\partial R}{\partial \alpha} \tag{3.7}$$

Denote the i^{th} column of the perturbation field as u_i

$$u_i = \frac{\partial U}{\partial \alpha_i} \tag{3.8}$$

which can be calculated by setting the RHS of the linearised equation to residual perturbation caused by a unit perturbation of α_i, i.e,

$$\mathbf{L}u_i = f_i = \frac{\partial R}{\partial \alpha}\delta\alpha \tag{3.9}$$

where the i^{th} component of $\delta\alpha$ is set to unity and all the rest is set to zero. This procedure, involving solving a linear system with different RHS, needs to be carried out for each component of α to obtain the perturbation matrix u, which is prohibitively expensive for large scale problem with a large number of design variables.

The adjoint approach rewrites the perturbation of the cost function w.r.t. the design variable as

$$\frac{dJ}{d\alpha} = \frac{\partial J}{\partial \alpha} + v^T f \tag{3.10}$$

with the adjoint variable v being the solution of the adjoint equation

$$\mathbf{L}^T v = g \tag{3.11}$$

. The equivalence used in Eq. (3.10) is

$$g^T u = g^T \mathbf{L}^{-1} f = (\mathbf{L}^{-T} g)^T f = v^T f \tag{3.12}$$

Note that the RHS of Eq. (3.11) is the perturbation of the cost function w.r.t. the flow variable, and the design variable does not explicitly appear in the adjoint equation. This equation needs to be solved as many times as the number of the cost functions. Therefore, when the number of the design variables is much greater than that of the cost functions, the adjoint solver is much more superior than any other competing methods for computing the gradient.

3.3 Time-marching the adjoint equation

To solve the adjoint equation, the full Jacobian matrix can be explicitly computed and stored in memory and the resulting linear system can be solved using either Newton method or other iterative linear solver. The major issues are the prohibitive memory and runtime cost. First, the full Jacobian is usually of 2nd order, and thus the sparsity of the matrix is based on the 2nd order stencil, equivalent to around 20 neighbouring nodes for each node, if an all-hex mesh is used. Second, the full Jacobian usually has a very high condition number and would result in an extremely stiff linear system which requires a very expensive (in terms of mainly memory, but also runtime) preconditioner to solve efficiently. The memory overhead can be avoided if a linear solver is used that only needs matrix vector multiplication, such as GMRES and thus the full Jacobian does not have to be stored in memory, thus the so-called Jacobian Free methods. However, in order to effectively precondition the system, one would still need either the 2nd-order or 1st-order approximate Jacobian matrix to be stored in memory.

In this work, the same time stepping for the flow solver is used for solving the adjoint method. The main advantages include the low memory requirement and the fact that many techniques for convergence acceleration of the nonlinear flow solver, such as RK, MG, block-Jacobi preconditioner, can be directly applied to the adjoint solution. In addition, the same asymptotic convergence rate guaranteed by using the identical times stepping is a powerful debugging tool for the development of an adjoint solver.

The adjoint solution can be solved for by driving the adjoint residual

$$R_a(v) = \mathbf{L}^T v - g$$

to zero. The time stepping is the same as the flow solver, only replace the flow residual $R(U)$ by the adjoint residual $R_a(v)$:

$$\mathbf{P}^T \Delta v^n = -R_a(v^n)$$

or

$$v^{n+1} = v^n - \mathbf{P}^{-T} R_a(v^n)$$

3.4 Eigenvalues and asymptotic convergence

Note that the preconditioner for the adjoint equation is the transpose of that of the flow equation $\mathbf{P}$. The resulting time stepping thus has the same asymptotic convergence as the flow solver. Assume $\tilde{U}$ is the exact solution of the flow equation, and the error of the intermediate flow solution U^n is defined as

$$\epsilon^n_{flow} = U^n - \tilde{U}$$

which satisfies

$$\epsilon_{flow}^{n+1} = (I - \mathbf{P}^{-1}\mathbf{L})\epsilon_{flow}^{n}$$

and thus the convergence rate for flow is

$$\frac{||\epsilon_{flow}^{n+1}||}{||\epsilon_{flow}^{n}||} = \rho(I - \mathbf{P}^{-1}\mathbf{L})$$

where the operator $\rho(\cdot)$ means the spectral radius of a matrix, which determines the asymptotic convergence rate of a fixed-point iteration (FPI). Similarly, assuming $\tilde{v}$ is the exact solution of the adjoint equation, then for error $\epsilon_{adj}^{n} = v^n - \tilde{v}$ we have

$$\epsilon_{adj}^{n+1} = (I - \mathbf{P}^{-T}\mathbf{L}^{T})\epsilon_{adj}^{n}$$

and the convergence rate for adjoint is

$$\frac{||\epsilon_{adj}^{n+1}||}{||\epsilon_{adj}^{n}||} = \rho(I - \mathbf{P}^{-T}\mathbf{L}^{T})$$

The asymptotic convergence rate of the primal and the adjoint are guaranteed to be the same because

$$\rho(I - \mathbf{P}^{-1}\mathbf{L}) = \rho(I - \mathbf{P}^{-T}\mathbf{L}^{T}) \tag{3.13}$$

To prove Eq. (3.13), suppose there are non-singular matrices A and B and matrix AB has eigenvalue λ and eigenvector v, i.e.,

$$\mathbf{AB}v = \lambda v$$

multiplying each side by matrix $\mathbf{B}$ yields

$$\mathbf{BAB}v = \lambda \mathbf{B}v$$

that is, the matrix $\mathbf{BA}$ has eigenvalue λ and eigenvector $\mathbf{B}v$. Therefore, matrices $\mathbf{AB}$ and $\mathbf{BA}$ have the same eigenvalues. Since $\mathbf{BA}$ and $\mathbf{A}^T\mathbf{B}^T$, being the transpose of each other, also have the same eigenvalues, matrices $\mathbf{AB}$ and $\mathbf{A}^T\mathbf{B}^T$ have the same eigenvalues.

Furthermore, if λ is the eigenvalue of matrix $\mathbf{AB}$, then

$$\det(\mathbf{AB} - \lambda I) = 0$$

and thus

$$\det(I - \mathbf{AB} - (\lambda + 1)I) = 0$$

Similarly, we have

$$\det(I - \mathbf{A}^T\mathbf{B}^T - (\lambda + 1)I) = 0$$

Therefore, both $I - \mathbf{AB}$ and $I - \mathbf{A}^T\mathbf{B}^T$ have eigenvalue $\lambda + 1$. Thus identity 3.13 is proved.

The conclusion on the identity of the asymptotic convergence rates of flow and adjoint can be straightforwardly extended to include Runge-Kutta and multigrid [28] (or the transpose of them if they are not self-adjoint) which have the equivalent effect as a preconditioner.

3.5 Adjoint code implementation using AD tool

The main technical challenge of implementing the adjoint code is how to compute matrix vector product on the LHS of

$$\mathbf{L}^T v = g$$

without explicitly computing and storing the transposed Jacobian matrix $\mathbf{L}^T$. This is possible using reverse mode Automatic Differentiation with an AD tool.Suppose the residual subroutine has the interface as follows

`subroutuine res(`U`, `R`)`

then the forward differentiated residual subroutine is

`subroutuine res_d(`U`, `$\dot{U}$`, `R`, `$\dot{R}$`)`

and the reverse differentiated residual subroutine is

`subroutuine res_b(`U`, `$\overline{U}$`, `R`, `$\overline{R}$`)`

The primal code subroutine 'res' takes the flow variable U as the input and computes nonlinear residual R as the output. The forward differentiated subroutine 'res_d' takes both the flow variable U and the perturbation $\dot{U}$ as an input and compute both the nonlinear residual R and the linear residual $\dot{R}$ as outputs. The linear residual computed is

$$\dot{R} = \frac{\partial R}{\partial U}\dot{U} = \mathbf{L}\dot{U} \tag{3.14}$$

i.e., the linear residual is essentially the matrix vector product of the Jacobian matrix and the flow perturbation.

The reverse differentiated subroutine 'res_b' takes the flow variable U and $\overline{R}$ as inputs, and computes the nonlinear residual R and $\overline{U}$ as outputs. According to the definition of reverse differentiation

$$\overline{U} = \left(\frac{\partial R}{\partial U}\right)^T \overline{R} = \mathbf{L}^T\overline{R} \tag{3.15}$$

i.e., if the flow variable U and the adjoint variable v are passed into the subroutine 'res_b' as input arguments U and $\overline{R}$, the output argument $\overline{U}$ is essentially the matrix vector product of the transposed Jacobian matrix and the adjoint variable, that is, the adjoint residual that we would like to compute

$$R_a \leftarrow \overline{U} - g = \mathbf{L}^T v - g \tag{3.16}$$

To reverse differentiate an algorithm by hand is not impossible. In fact, it is quite simple as long as one strictly follows the call graph and differentiate the code statements one by one backward. In practice, reverse differentiation by hand is too tedious and error-prone to be applied to large codes. This is the main reason AD tools are used.

Mentioned above is the basic methodology of developing a discrete adjoint solver using AD tool. The automation of this process can be done by using the advanced features of the AD tools through additional pre- and post-processing scripting through the Makefile, as is explained in [17, 44, 29]. The obvious advantage of automatically generating the adjoint code at compilation time is to ensure the consistency between the discrete adjoint code and the baseline nonlinear flow solver which could be constantly modified over time.

3.6 Summary

In this section, the mathematical formulation of the discrete adjoint approach is explained in detail. The preferred iterative method for solving the adjoint equation is to mimic the time-stepping of the original nonlinear flow solver, which avoids the excessive memory overhead that would incur if a direct solve approach were used. Reverse differentiation is used to compute the matrix vector product for the adjoint equation without computing and storing the transposed Jacobian matrix. The use of the AD tool Tapenade in developing a discrete adjoint without the tedious hand differentiation, is also explained.

Chapter 4

Stabilisation of discrete adjoint

4.1 Introduction

The adjoint method is an essential ingredient of gradient-based steady-state CFD shape optimisation as it allows the computation of the gradient of an objective function with respect to a large number of design variables at near constant computational cost comparable to that of the flow solution.

As discussion in the previous chapter, discrete adjoint approach is chosen in this work because it exactly preserves the spectrum of the Jacobian matrix of the asymptotically converged steady state flow solution, which provides a powerful debugging tool and completely removes the uncertainty due to the different discretisation as seen in continuous adjoint. This also ensures that the computed gradients are the exact gradients of the discrete model, which is a very desirable property: the discrete gradient is then exactly zero where the flow solution has an unconstrained minimum. This also means that the properties of the flow discretisation, including any preconditioners, are inherited and govern the spectral behaviour of the adjoint, which ensures that if the primal solution (flow) converges (i.e. the system Jacobian is contractive with the magnitude of all eigenvalues $|\lambda| \leq 1$), so will the discrete adjoint as the exact transposition of the system Jacobian preserves the eigenvalues [28].

However, in many industrial cases the flow does not converge in this sense, but enters limit cycle oscillations (LCO) instead [13]. This can be caused e.g. by the mesh being fine enough in some area to resolve certain localised flow unsteadiness due to vortex shedding, or e.g. by separation bubbles in the flow which have only very loose physical coupling with the bulk flow. This is typically not perceived as a problem for the analysis of the flow field as the value of the objective function is often steady enough and considered accurate enough to be used. On the other hand, applying the 'steady-state' discrete approach in this situation often fails, as a Jacobian taken from a snapshot during the LCO is likely to be not contractive but exhibit a number of eigenvalues with magnitude larger than one, and hence the iterative scheme for the adjoint will fail once the error modes associated with those eigenvalues have grown sufficiently.

To stabilise the linear solver, the Recursive Projection Method (RPM) and the Generalized Minimal Residual Method (GMRES) have been proposed [15, 25, 26]. GMRES[82] is an iterative method that is guaranteed not to diverge even in the presence of outliers. Therefore the existing linear fixed-point iteration (FPI) can be used as a preconditioner for the GMRES solver, and the adjoint GMRES solver, provided a large enough number of Krylov vectors are used, is bound to converge fully. This approach can be implemented with minimal amount of change to the existing code and should be capable of stabilising the adjoint. The main drawback is that for large industrial cases, a large number of Krylov vectors are typically needed in order to converge, e.g., 100 Krylov vectors with no restarts were used in [55]. Although to the author's better knowledge, this limitation is not reported in literature, the large memory overhead of GMRES for stabilisation is the major limiting factor when trying to converge the adjoint for those unstable nonlinear flows. RPM stabilises the adjoint with a different mechanism. The unstable FPI is iterated for many iterations to allow the dominant error modes to grow and to be identified. Then a direct solver is applied to those particular modes while the existing FPI is applied to the remaining, presumably stable, modes. If the remaining modes still contain unstable modes, then the unstable FPI will identify again the unstable modes and the procedure above is repeated until all the unstable modes are identified. The RPM method needs to run the FPI adjoint solver for a large number iterations in order to identify the dominant unstable modes, and a typical value can be $k = 100 - 10000$, or even larger for practical problems. At the same time, m vectors need to be stored for potential unstable modes. The long runtime and large memory overhead is also preventing RPM from being used widely for stabilising the adjoint for three dimensional flows for practical industrial applications. An additional problem with both RPM and GMRES is that the linearisation is then based on an unstable saddle-point of the flow solution which is arbitrarily picked from the LCO, hence the adjoint sensitivity may be inconsistent with the average of the primal over the LCO [48]. The objective function the designer is interested in is actually the sensitivity of the averaged unsteady flow solution, rather than the one of an arbitrary snapshot. To approximate this average, one could take the average of the flow field in the pseudo-time interval and base the Jacobian on this averaged state. Irrespective of the question whether the objective function is linear enough with respect to the state to justify this approximation, in general the Jacobian will fail to be contractive as this averaged solution does not satisfy the steady state flow equations. Still, this approach has been shown to be practicable for some limited cases of industrial relevance when used with continuous adjoint methods, where the re-discretisation of the adjoint equation can provide additional stabilisation terms [70]. For a rigorous application of this approach, regularisation in time would be required that gives rise to stabilising contributions to the Jacobian, but the knowledge in the field on this is in its early stages [73].

An approach guaranteed to work would be to treat the limit cycles as an unsteady flow and trace backward in time the adjoint characteristics [61]. Storing checkpoints and recomputing intermediate solutions would result in a significant increase in runtime and memory requirements, which would only be warranted if the objective function is non-linearly affected by the instability.

So far we have not distinguished whether the instability of the iterative method is due to the flow physics, such as vortex shedding and separation bubbles decoupled from the mean flow, or whether it is due to the discretisation. There cannot be a clear distinction between the two as e.g. vortex shedding behind rounded trailing edges of turbine blades can be suppressed with coarse meshing and large time steps. Putting numerical stability into focus suggests another approach to achieve convergence of the adjoint solver, namely to improve the stability of the flow discretisation such that flow can be converged to a level that the Jacobian is contractive. Clearly, such an approach may fail for flows with strong physical unsteadiness or will be inaccurate where the unsteady phenomena have a significant effect on the average of the objective function. But for cases with minor unsteadiness, such an approach will be stable and accurate, as well as being significantly less expensive than the computation of the unsteady flow and adjoint. In the author's view, apart from [48], where the physically unstable flow solution is converged to steady state using a strong implicit solver, the approach of obtaining stable discrete adjoints through focusing on the primal stability has not been explored adequately in the literature.

Before presenting the proposed iterative scheme, a brief overview of the various techniques for accelerating convergence of nonlinear flow solvers, with particular focus on RANS, is given in the next section.

4.1.1 Convergence acceleration techniques for nonlinear flow solvers

A fixed-point iteration can generally be presented as

$$\mathbf{P}\Delta U^n = -R(U^n) \tag{4.1}$$

where the right hand side represents the residual of the discretisation which determines the accuracy of the converged solution, while the left hand side matrix $\mathbf{P}$ is a non-singular preconditioning matrix that controls the transient behaviour of the intermediate solution U^n over the iterations. At each iteration, the flow update is computed by inverting $\mathbf{P}$ directly or approximately.

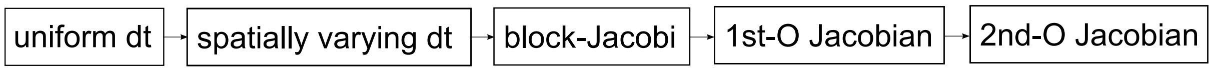

Figure 4.1: A spectrum of various time-stepping methods based on different approximations of the Jacobian.

A hierarchy of time-marching methods can be derived by using for the preconditioning matrix $\mathbf{P}$ different approximations of the flow Jacobian matrix $\partial R/\partial U$ [50], as illustrated in fig. 4.1. Scalar time stepping, with either a uniform or a spatially varying time step, is too inefficient for practical steady RANS cases due to the highly stretched mesh in the boundary layer regions and the numerical decoupling between the equations. The physical coupling can be included by retaining the diagonal blocks of the Jacobian in the preconditioning matrix, thus termed a point-implicit or Block-Jacobi (B-J) solver. A B-J solver accelerated with multigrid [64] is in general efficient enough to compute viscous flows when combined with semi-coarsened geometric multigrid [66]. In addition, the B-J solver is easy to implement, to parallelise and has very low memory requirements. The drawback is that the convergence tends to degenerate for practical cases, especially for cases where the viscous effect dominates. Furthermore, as stated in the introduction, for large industrial cases with complex geometries, the B-J flow solver may converge only to LCO after an initial residual drop, corresponding to pseudo unsteadiness of either numerical or physical origins.

A better approximation of the Jacobian can be constructed by including the off-diagonal blocks in the Jacobian matrix, coupling the neighbouring nodes. This leads to an implicit solver with faster convergence by damping the transient modes more effectively. The highest level of approximation is obviously the exact 2nd-order Jacobian, i.e., the exact linearisation of the residual w.r.t. the flow variables, which could e.g. be used with Newton's method. Such a Newton solver is applied to a 2D viscous RAE2822 case in [24]. In addition to a start-up difficulty, which requires such stringent control of the step-width that it renders the Newton solver practically useless for this application, it is reported that the resulting linear system at each nonlinear iteration is very stiff and the GMRES(50) solver needs to be preconditioned by ILU(4) in order to prevent stalling. The combined memory requirements of the exact Jacobian and ILU(4) combined is roughly 18 times that of B-J (6x for the Jacobian and 12x for ILU(4) and 50 Krylov vectors). This memory increase will be even more substantial for 3D cases due to the increased number of neighbouring nodes.

Jacobian-Free Newton-Krylov (JFNK) methods [47] can avoid the storing Jacobians but instead work out the Jacobian-vector product needed in the Krylov solvers on the fly, typically using finite differences,

$$\frac{\partial R}{\partial U} \cdot \delta U = \frac{R(U + \epsilon \delta U) - R(U - \epsilon \delta U)}{2\epsilon}. \tag{4.2}$$

However, for the Krylov solver to converge, effective preconditioners are needed, with ILU factorisation being recognised as on of the most effective ones. In order to limit memory overhead, the Jacobian that is ILU-factorised could be approximated [47], but practical preconditioners require storage equivalent to the Jacobian.

Major storage savings arise from dropping second-order neighbours and computing the Jacobian on nearest-neighbour contributions only. For a standard MUSCL scheme [94] this would correspond to frozen spatial gradients. In our implementation we use zero spatial gradient for MUSCL scheme when computing the inviscid flux and non-zero but frozen gradient for computing the viscous flux. The resulting memory savings of course are paid for with an impaired convergence rate. While an exact 2nd-order Jacobian typically does not lead to a robust scheme [24], at the least requiring a sophisticated solver steering strategy [88], a heuristic blend of first and second-order Jacobians can provide very good results [62], but again at the cost of very large memory use.

Multigrid (MG) is a very effective preconditioner, especially in its full approximation storage form (FAS) [12] that includes non-linear effects on the coarser grids. On each grid level FAS needs to solve the non-linear equations using an h-elliptic discretisation [12] that has good high-frequency smoothing. Unfortunately Krylov solvers such as GMRES do not provide adequate high-frequency damping to be used as a smoother within MG, but it is possible to use MG as a preconditioner for Jacobian-Free Newton-Krylov method (JFNK) [47], which however then reduces to the linear Correction Scheme (CS) MG which is much less effective [60].

Algebraic Multigrid (AMG) does not require to build a coarse grid, but the coarsening is applied to the system matrix of the linear problem, rather than the grid [67]. It is hence a linear CS MG scheme, which avoids re-discretisation on the coarse grid, but hence loses the convergence advantage of the non-linear coarse-grid re-discretisation. This may be compensated by efficient directional coarsening in all areas of the flowfield, not just the viscous layer.

Swanson et al. [93] show that an implicit discretisation with symmetric Gauss-Seidel (SGS), preconditioned with the 1st-order Jacobian is a good smoother for multigrid [77] if the linear system is wrapped inside a standard Runge-Kutta (RK) multistage scheme, with RK providing the desired damping for the high-frequency error modes. Furthermore, the RK coefficients could be fine-tuned for better robustness and performance [81]. This approach of solving the 1st-order linear system using SGS at each RK stage, accelerated by MG at the outer iteration, has proven to be robust and is now generally accepted as the benchmark solver performance. SGS is susceptible to a lack of diagonal dominance, and thus when SGS is used to solve the linear system, the CFL number is lowered not for the nonlinear instability of the outer iteration, but for the linear instability of the inner SGS iteration, reducing the convergence rate of the scheme.

The performance of the MG method does strongly depend on the method that is used to generate the coarse grids and how the problem is discretised on them. Agglomeration MG fuses fine grid cells into coarse grid cells which then are no longer convex [49]. The scheme can be implemented as an efficient non-linear FAS scheme if the equations are re-discretised on the coarser meshes, however an accurate discretisation of the viscous operator is not straightforward on the non-convex cells. Directional agglomeration with additional line-relaxation is needed for high-Reynolds number flows [60].

Geometric coarsening through edge-collapsing is very effective for simplex grids, but much more difficult to achieve for hybrid grids. The element-collapsing method [66, 64] collapses sets of edges to remove elements and produces coarse grids with the standard element types, possibly with degenerate edges. Semi-coarsened grids for high-Reynolds flows can be produced. However the mesh quality and coarsening ratio degenerate with repeated application on the coarser levels.

The authors acknowledge that the performance of a particular smoother is very closely linked to the way the coarse grids are generated, the results presented here use the geometric element-collapsing scheme of Müller [66]. Assessing the performance of the proposed iterative method in the context of e.g. agglomeration multigrid or AMG will be reported in future work.

4.1.2 Proposed new algorithm

The survey of the existing approaches in Sec. 4.1.1 shows that an effective RANS method should make use of a) non-linear FAS multigrid which offers improved convergence, and b) robust GMRES linear solvers which allow large CFL numbers. To the author's best knowledge there is no published algorithm using multigrid as a RANS solver ('outside') and GMRES as a smoother ('inside'), due to the poor smoothing properties of the Krylov solvers. Presenting a solution to this problem and successful application of such an algorithm is the main novelty proposed in this thesis.

To achieve adequate high-frequency damping of the GMRES linear solver, in the proposed scheme it is embedded in a Runge-Kutta multistage time-stepping scheme, preconditioned with an ILU(0)-factorisation of the first-order Jacobian. The proposed method is hence referred to as the Jacobian-Trained Krylov-Implicit-Runge-Kutta or JT-KIRK algorithm.

This chapter first explains in Sec. 4.2 the mathematical background of the nonlinear flow solver and the development of the JT-KIRK algorithm with emphasis on the temporal rather than spatial discretisation. Sec. 4.3 describes the discrete adjoint equation and how its time-marching relates to the one of the flow solver. Sec. 4.4 compares the ILU preconditioned GMRES and SGS solution strategies for the linear system arising from the implicit scheme and presents parameter studies on how to optimise the solver efficiency. Sec. 4.6 shows the results for four test cases, two stable cases which the B-J solver can fully converge, and two unstable cases that cannot be converged by the B-J solver, resulting in divergence of the discrete B-J adjoint. Eigenvalue analysis is shown for the second unstable Case 4 to highlight the eigenvalue clustering using the implicit algorithm. Conclusions are presented in Sec. 4.7.

4.2 Flow solver

4.2.1 Typical temporal discretisation for RANS flow solvers

The flow solver used here is an industrial compressible RANS flow solver using a vertex-centred finite volume method on unstructured grids. A multistage Runge-Kutta time integration scheme is used to time-march the solution to steady state, and geometric multigrid with semi-coarsening as well as B-J preconditioning are used to accelerate the convergence [21, 63, 64]. The steady state solution is reached when the residual of each control volume reaches zero, i.e.,

$$R(U) = 0,$$

where the flow variables U and residual R are both column vectors of dimension $5\times N$ for 3-D Euler and laminar Navier-Stokes and $6\times N$ when a one-equation turbulence model is used, on a mesh with N nodes.

A generic semi-discrete time-marching scheme for a conservation equation can be written as

$$\frac{U^{n+1}-U^{n}}{\sigma\Delta t}V + (1-\beta)R(U^{n}) + \beta R(U^{n+1}) = 0 \tag{4.3}$$

where Δt is the time-step, σ denotes the CFL number for the linear solver of the implicit scheme and V is the size of the control volume. The coefficient β provides a blending between explicit and implicit residuals. The residual at time level $n+1$ can be linearised using the Jacobian at time level n to be

$$R(U^{n+1}) \approx R(U^{n}) + \left.\frac{\partial R}{\partial U}\right|_{U=U^{n}} (U^{n+1}-U^{n})$$

and Eq. (4.3) becomes

$$\left(\frac{V}{\sigma\beta\Delta t} + \left.\frac{\partial R}{\partial U}\right|_{U=U^n}\right)\Delta U^n = \frac{-1}{\beta}R(U^n). \tag{4.4}$$

For an implicit scheme with $\beta > 0$, the CFL number usually can be very large which is important for fast convergence to the steady state solution. Note that although β is usually chosen between 0 and 1, it is possible to allow β to be greater than 1, equivalent to under-relaxation.

4.2.2 Block-Jacobi solver

To simplify solving the linear system at each time step and to reduce the memory overhead, the Jacobian matrix $\frac{\partial R}{\partial U}$ on the LHS of Eq. 4.4 can be approximated by the Block-Jacobian for node i, while the higher-order accurate residual operator $R(U^n)$ is used on the RHS which determines the accuracy of the converged steady state solution. The B-J matrix can be computed using hand differentiation [63] or using AD tools such as Tapenade [35].

The resulting B-J or point-implicit scheme is

$$\left(\beta \left.\frac{\partial R_i^{(I)}}{\partial U_i}\right|_{U=U^n}\right)\Delta U_i^n = -R_i^{(II)}(U^n) \tag{4.5}$$

where the Roman superscript for the residual denotes that the spatial operator is of either 1st-order or 2nd-order, and the subscript denotes the i-th node.

To simplify the notation, the LHS matrix of Eq. 4.5 can be combined into one preconditioning matrix $\mathbf{P}$, with the i-th block on the diagonal defined as

$$\mathbf{P}_i = \frac{V_i}{\sigma\Delta t_i} + \beta\frac{\partial R_i^{(I)}}{\partial U_i}$$

and the discrete governing equation equation can be simplified as

$$\mathbf{P}\Delta U^n = -R^{(II)}(U^n). \tag{4.6}$$

The B-J time-stepping is combined with RK to provide additional damping for high-frequency error modes which makes the smoother suitable for MG [40].

The Block-Jacobian $\frac{\partial R_i^{(I)}}{\partial U_i}$ is approximated per block with the one-equation turbulence model treated as a passive scalar. Including the derivatives of the turbulence model variable in the Block-Jacobian in the author's experience increased the computational cost, did not improve convergence rate but in cases affected the robustness adversely. The resulting structure in 3-D is hence a full 5×5 block for the conservative variables and a unit diagonal entry in the 6,6 position for the Spalart-Allmaras variable. To keep the implementation simple we retain the zeros in the 6th row and column in the Block-Jacobian. The associated memory cost for storing the B-J matrices for all control volumes in the case of the single equation Spalart-Allmaras (SA) turbulence [89] model is 6 times for the storage of the flow solution, which is typically deemed affordable. Alternatively, the the non-zero blocks of the B-J matrices can be formed on the fly during the update step, further reducing the memory at the cost of a limited increase of CPU time.

When an m-stage RK scheme is used, the solution is updated as

$$\begin{aligned}
U^{(0)} &= U^{n}, \\
U^{(1)} &= U^{(0)} - \alpha_1 \mathbf{P}^{-1} R^{(II)}(U^{(0)}), \\
&\vdots \\
U^{(m)} &= U^{(0)} - \alpha_m \mathbf{P}^{-1} R^{(II)}(U^{(m-1)}), \\
U^{n+1} &= U^{(m)}.
\end{aligned}$$

At each RK stage the preconditioned matrix $\mathbf{P}$ is inverted by directly inverting each of the diagonal block matrice to update the flow solution. The B-J solver combined with MG can produce grid-independent convergence for inviscid flows and works reasonably well for viscous cases if semi-coarsening is used [64]. However, grid-independent convergence is usually not observed for general three dimensional RANS simulations.

4.2.3 1st-order Jacobian implicit solver

To obtain a stronger coupling between neighbouring nodes and to further accelerate convergence, the 1st-order Jacobian can be used to improve over the B-J matrix. The preconditioner $\mathbf{P}$ based on the 1st-order Jacobian can be written as

$$\mathbf{P}_{i,j} = \frac{V_i}{\sigma \Delta t_i}\delta_{ij} + \beta \frac{\partial R_i^{(I)}}{\partial U_j},$$

where nodes i and j either are identical or are immediate neighbours of each other. Each node i and each edge ij then give rise to a 6×6 block (in the case of the Spalart one equ. model) with the same structure as discussed for B-J in Sec. 4.2.2. To save on storage, the full 5×5 blocks for the Navier-Stokes variables are stored in a block-wise compressed sparse row format (BCSR) separately from the SA-variable which is stored in point-wise compressed sparse row (CSR) format. Both matrices have the same sparsity pattern so the row and column index vectors are shared. This approximation to the Jacobian implies full coupling of the Navier-Stokes variables, but an implicit treatment of a passive scalar similar to B-J for the turbulence variable.

In this case each $\mathbf{P}_{i,j}$ is a 5×5 or 6×6 block matrix at location (i,j) of the whole matrix $\mathbf{P}$ which can be computed by differentiating the residual subroutine either by hand or using Automatic Differentiation. For the presented results it was computed in vector-forward mode with the AD-tool Tapenade [35].

The restriction to first-order neighbour contributions in the Jacobian precludes a differentiation of the gradients. As typical in literature, the inviscid flux contributions are evaluated with zero gradients (1st-order accuracy), which will adversely affect convergence rate, but increase robustness. The viscous terms are evaluated here using a correction for the normal component of the Green-Gauss face gradient $\overline{\nabla U}_{ij}$ [63] to improve high frequency damping formulated as

$$\nabla U_{ij} = \overline{\nabla U}_{ij} - \left(\overline{\nabla U}_{ij} \delta s_{ij} - \frac{U_i - U_j}{|x_i - x_j|} \right) \delta s_{ij}$$

where

$$\delta s_{ij} = \frac{x_i - x_j}{|x_i - x_j|}$$

with the nodal coordinates x_i for node i. The Jacobian of the viscous flux then differentiates w.r.t. the term $U_i - U_j$ but considers $\overline{\nabla U}_{ij}$ fixed.

The cost of computing the approximate Jacobian with AD is equivalent to evaluating the nonlinear residual 5 times (or 6 times for SA turbulence), but is not substantially increased compared to evaluating the B-J matrix. The cost of computing the approximate Jacobian can be further reduced by optimising the implementation of the nonlinear flux subroutine [65] and by selectively eliminating parts of the flux calculation from the AD tool using scripts or pragmas. Strategies for efficiently solving the linear system for both the nonlinear flow and the corresponding adjoint solver are explained in detail in Sec. 4.4.

4.3 Adjoint solver

To develop a discrete adjoint solver using an existing flow solver, one could explicitly compute and store the exact Jacobian corresponding to the second-order accurate discretisation and solve the resulting linear system. However, for practical cases, both the memory and runtime for this approach could be prohibitively expensive [24], moreover efficient preconditioners would need to be implemented. Alternatively, one can solve the adjoint equations using the same time-marching scheme as the nonlinear flow solver [18, 20, 28, 17]. The discrete adjoint equation uses the transposed exact Jacobian matrix of the nonlinear flow equation

$$\mathbf{L}^T v = g$$

with

$$\mathbf{L} = \frac{\partial R}{\partial U} \quad \text{and} \quad g = \frac{\partial J}{\partial U}^T,$$

where v is the adjoint variable and J is the objective function, which is a function of both the mesh and the flow field. The residual of the adjoint equations

$$R_v(v) = \mathbf{L}^T v - g$$

can computed by application of AD tools [35, 17] and then be used to update the adjoint solution,

$$\mathbf{P}^T \delta v^n = -R_v(v^n), \tag{4.7}$$

where the preconditioner $\mathbf{P}^T$ is the transpose of the preconditioner for the flow solver. Assume $\tilde{v}$ is the exact solution of the adjoint equation, then for an error $e^{n+1} = v^{n+1} - \tilde{v}$ at $n+1$-th iteration, we have

$$e^{n+1} = (I - \mathbf{P}^{-T}\mathbf{L}^T)e^n.$$

The asymptotic convergence rate of the flow and the adjoint are guaranteed to be the same [30] since

$$\rho(I - \mathbf{P}^{-1}\mathbf{L}) = \rho(I - \mathbf{P}^{-T}\mathbf{L}^T),$$

where the operator $\rho(\cdot)$ means the spectral radius of a matrix, which determines the asymptotic convergence rate of a FPI. The conclusion on the identity of the asymptotic convergence rates of flow and adjoint can be straightforwardly extended to include Runge-Kutta multistage integration as well as multigrid. If all operators contributing to $\mathbf{P}$ are exactly transposed for $\mathbf{P}^T$ in (4.7), not only is full convergence of the adjoint solver guaranteed as long as the nonlinear flow solver asymptotically converges, but also the transient of the sensitivity is exactly the same when comparing tangent-linear and adjoint approaches [30], which is a very powerful validation tool [14]. This is the approach taken for the results presented in this chapter. However to achieve stability typically this condition can be relaxed, with only transposing non-symmetric parts of $\mathbf{P}$ such as low-Mach preconditioners and not reversing the sequence of the operators [17].

4.4 Solving the linear system

For both the flow and adjoint equations, a linear system needs to be solved at each RK stage on each level of MG grids. Two approaches to solve this linear equation are considered here, SGS and ILU(0)-preconditioned GMRES.

4.4.1 SGS as linear solver

A symmetric Gauss-Seidel (SGS) iteration is a forward followed by a backward sweep of Gauss-Seidel (GS). The system matrix $\mathbf{A}$ is decomposed as $\mathbf{A} = \mathbf{D} + \mathbf{L} + \mathbf{U}$, where $\mathbf{D}$, $\mathbf{L}$ and $\mathbf{U}$ are the block-wise diagonal, lower and upper triangular matrices and iterate as follows

$$\begin{aligned}(\mathbf{D} + \mathbf{L})x^* &= -\mathbf{U}x^n - R \\ (\mathbf{D} + \mathbf{U})x^{n+1} &= -\mathbf{L}x^* - R.\end{aligned}$$

Since $\mathbf{D} + \mathbf{L}$ and $\mathbf{D} + \mathbf{U}$ are both block-wise triangular matrices, both equations can be solved with block-wise backward/forward substitutions. In order for the SGS to converge, diagonal dominance is required which can be achieved by decreasing the CFL number σ from infinity to a smaller finite value.

It is not necessary to fully converge the linear system at each RK stage as the system matrix is only a first-order accurate approximation. In practice, 3 sweeps of SGS are used for each linear system solve, hence the label SGS(3). Using more than 3 iterations of SGS will increase the overall runtime which is proportional to the number of SGS sweeps, but does not further improve convergence rate or stability as reported by Swanson et al. [92] and also independently verified through numerical experiments.

4.4.2 ILU-preconditioned GMRES as linear solver

The proposed JT-KIRK scheme uses GMRES [82] to solve the linear system for its proven robustness for non-symmetric systems. GMRES stores m Krylov vectors and hence requires additional memory equivalent to that of m nonlinear flow solutions. The memory requirement becomes prohibitive for large m, hence in practice, a limited m is used. To then avoid stalling convergence one uses restarted GMRES: whenever the number of basis vectors reaches m, they are discarded and GMRES is restarted using the partly converged solution. For all the cases used in this chapter, the stopping criterion for the GMRES solver is either using three GMRES vectors with zero restart or a one order of magnitude drop of the relative residual of the linear system, whichever criteria is met first, denoted by GMRES(1,3,0.1).

GMRES still needs an appropriate preconditioner for good convergence, most widely used are Jacobi, Lower-Upper-Symmetric Gauss-Seidel (LU-SGS) and Incomplete LU factorisation (ILU) of the approximate Jacobian [82]. The first two are easy to calculate but our results show that ILU(0), i.e. ILU with 0 level of fill-in, is a more robust and efficient preconditioner. The ILU(0) preconditioner, for simplicity denoted as ILU, is used in all the results shown here for GMRES. The preconditioned linear system (4.1) then becomes

$$\mathbf{U}^{-1}\mathbf{L}^{-1}\mathbf{P}\Delta U^n = -\mathbf{U}^{-1}\mathbf{L}^{-1}R, \tag{4.8}$$

where the matrices $\mathbf{U}$ and $\mathbf{L}$ are from the ILU decomposition

$$\mathbf{P} \approx \mathbf{L} \cdot \mathbf{U}. \tag{4.9}$$

GMRES uses the linear combination of orthogonal basis vectors of the Krylov subspace to approximate the exact solution. The Krylov subspace is constructed for the system matrix $\mathbf{A}$ of a linear system of Eq. $\mathbf{A}x{=}b$ as follows

$$\mathbf{K}_m(\mathbf{A}, b) = \mathrm{span}\{b, \mathbf{A}b, \mathbf{A}^2b, \mathbf{A}^3b, ..., \mathbf{A}^{m-1}b\} \tag{4.10}$$

where $\mathbf{A} = \mathbf{U}^{-1}\mathbf{L}^{-1}\mathbf{P}$ and $b = -\mathbf{U}^{-1}\mathbf{L}^{-1}R$. In (4.8), both the RHS and LHS are computed via several sweeps of matrix-vector multiplication and no matrix-matrix multiplication is done even though all the matrices are stored.

In the JT-KIRK scheme, the 1st-order approximate Jacobian and its ILU factorisation are computed only at the first RK stage at each nonlinear iteration of the flow solver. In the adjoint JT-KIRK solver, the approximate Jacobian and its ILU are constant, and hence only computed once at the first iteration for all grid levels. In the parallel implementation, ILU is performed for each sub-domain, and due to this decoupled implementation, the convergence deteriorates with increased number of partitions, as explained in Sec. 4.6.1.4.

4.5 Implementation

The JT-KIRK algorithm is implemented both in MGOPT and HYDRA codes. The time stepper is first implemented in MGOPT and tested on various two dimensional cases. The knowledge and experience gained is then used to re-implement it into the HYDRA nonlinear flow solver, which mainly involves replacing the existing Block-Jacobi preconditioner with the 1st-order Jacobian and the associated GMRES linear solver preconditioned by ILU(0). Additional implementation into HYDRA involves the correction to the 1st-order Jacobian due to the rotational periodic boundary condition. All the results shown in this chapter are produced using the HYDRA code.

4.6 Results

In this section, the proposed JT-KIRK flow solver and its discrete adjoint are applied to four different test cases with different flow features to assess both the solver efficiency and robustness improvement. The four test cases from turbomachinery applications are first briefly explained in terms their geometries, meshes and the flow characteristics, followed by detailed analysis of the solver performance on each case.

The Block-Jacobi discretisation manages to converge cases 1 and 2 fully, hence labelled as 'stable cases'. These cases demonstrate the solver efficiency improvement of the JT-KIRK algorithm. Cases 3 and 4 can only be converged by B-J to LCO with the adjoint diverging, hence these are labelled as 'unstable cases'. Case 3 exhibits a mild instability with insignificant effect on the objective function, an example of a case where a steady-state stabilisation is entirely justified. Case 4 is an artificially created case with severe instability where objective functions obtained by steady-state simulation will markedly deviate from the full unsteady ones. This case is presented to demonstrate stabilisation of the adjoint solver when JT-KIRK is used to converge fully both the flow and adjoint solutions.

4.6.1 Case 1, nozzle guide vane (NGV)

Case 1 is a nozzle guide vane (NGV) with subsonic inlet and outlet. The domain is meshed with 0.5 million hexahedral elements. The B-J solver converges fully with the residual dropping by 12 orders of magnitude. The NGV geometry and various contour plots (Mach number, static pressure and SA variable) are shown in fig. 4.2 for the surface at midspan. The flow is fully attached to the blade surface.

As typical for this type of RANS computation, the mesh around the trailing edge shown in fig. 4.3 is chosen coarse enough not to resolve the vortex shedding, but fine enough to limit the truncation error. The maximum cell aspect ratio for the boundary layer cells is 170 and y^+ for the first node is around unity.

Geometric multigrid is used to accelerate the convergence. The hierarchy of multigrid meshes has been generated using an element-collapsing algorithm with semi-coarsening [64], with a maximum allowable isotropic coarsening factor of 2 per dimension and an element aspect-ratio threshold of 3 for applying semi-coarsening, a set of parameters optimised for the smoothing characteristics of the B-J scheme. The sizes of the mesh levels and their effective coarsening ratios are listed in Tab. 4.1. It can be seen that the overall coarsening ratio remains below 2 and gradually deteriorates with coarser meshes.

4.6.1.1 Effect of parameter choices

Both the original B-J solver and the implicit SGS and JT-KIRK solvers are applied to Case 1 to explore the effect of the parameter choices on the stability and efficiency. The three solvers will be labelled B-J, SGS and JT-KIRK in the following, their adjoint

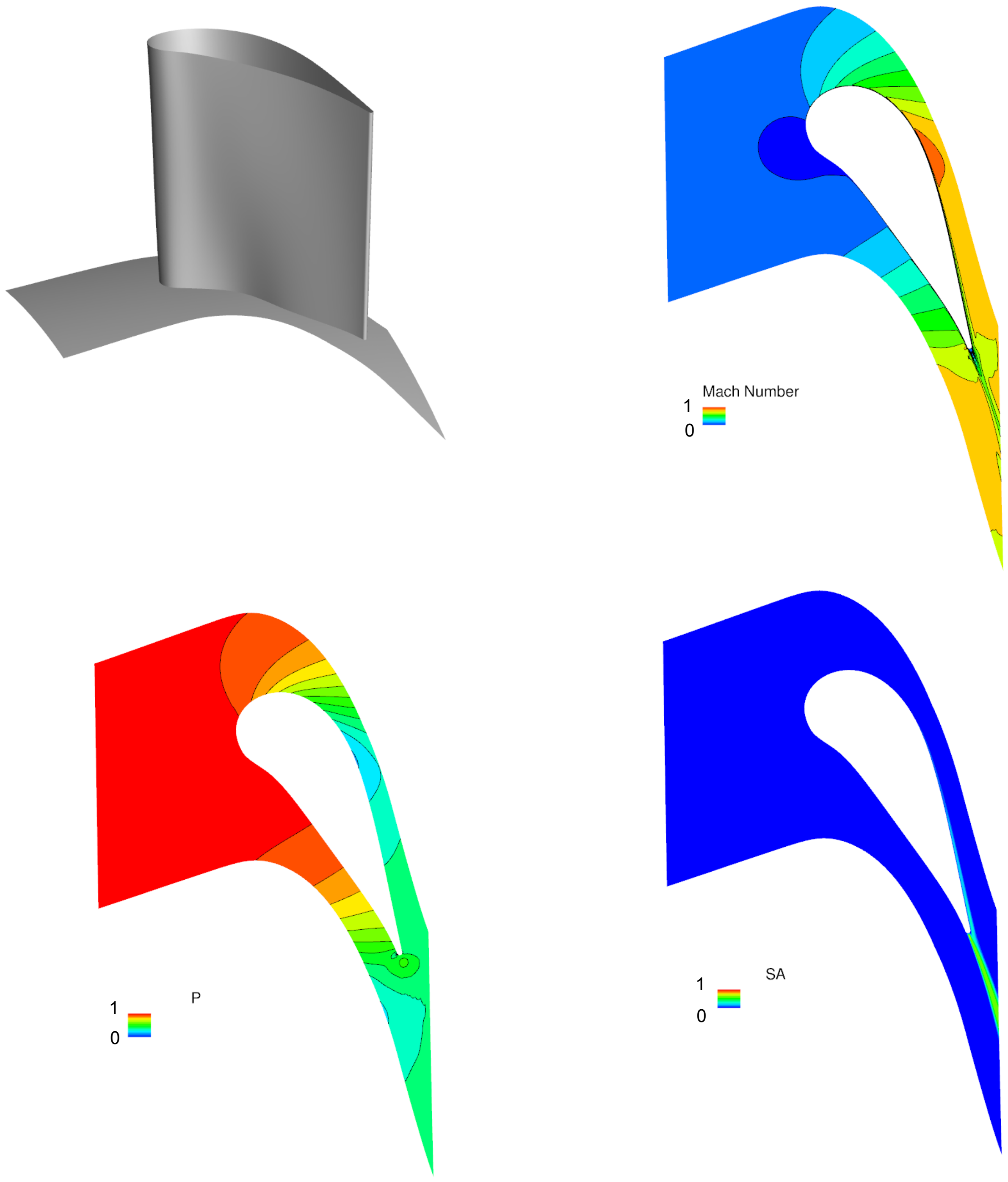

Figure 4.2: Case 1, Nozzle Guide Vane (NGV). Upper left: NGV geometry; upper right: Mach number contours at midspan; lower left: static pressure contours at midspan; lower right: Spalart-Allmaras turbulence variable contours at midspan. All legends are non-dimensionalised using their corresponding minimum and maximum values.

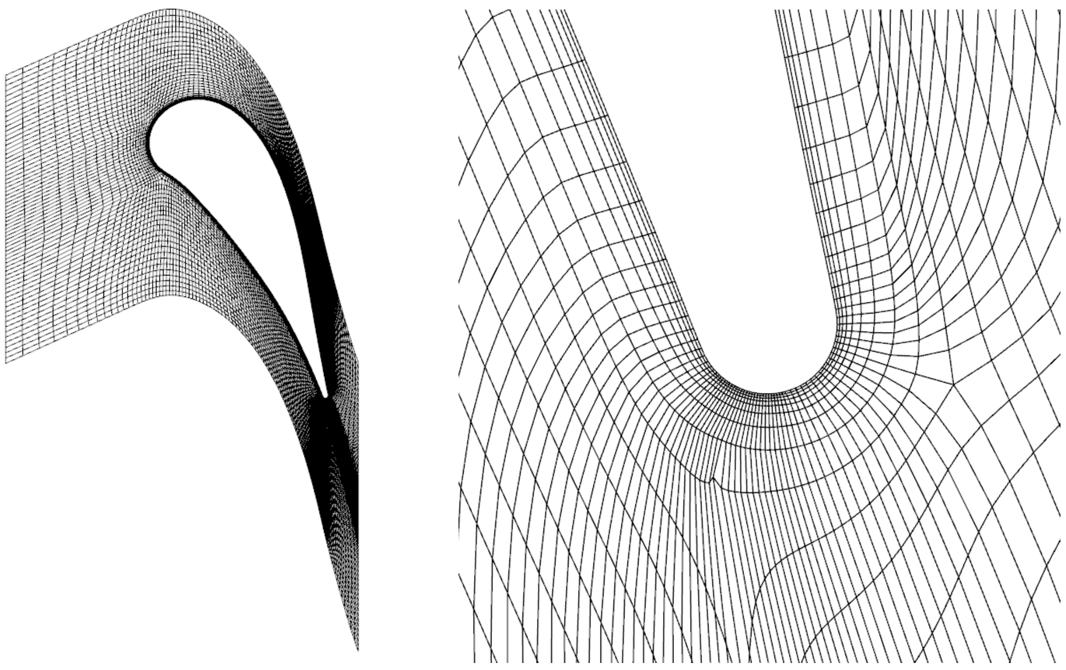

Figure 4.3: Case 1, NGV. Left: mesh for the whole domain, right: close-up view at the trailing edge. The quasi-2D mesh is taken at midspan.

Table 4.1: multigrid mesh statistics for Case 1

level	# of nodes	node ratio	# of edges	edge ratio
1	499150		1471161	
2	276565	1.80	894771	1.64
3	162065	1.71	565082	1.58
4	126743	1.28	451674	1.25
5	115514	1.10	414550	1.09

variants are developed following exactly the same time-stepping algorithm as the flow solver. The nomenclature used for all the solvers is explained in Table 4.2.

Table 4.2: Summary of different solvers

solver	preconditioner	linear equation solve
B-J	Block-Jacobian	block-wise direct inversion
SGS	1st order Jacobian	SGS(3)
JT-KIRK	1st order Jacobian	ILU(0)+GMRES(1,3,0.1)

4.6.1.2 Number of RK stages

Numerical experiments were performed for the B-J and JT-KIRK schemes to find the optimal number of RK stages. The coefficients for each stage are used as optimised for viscous flow [57] and proposed for B-J [63], both authors advocating the use of RK5. The linear solver settings for both B-J and JT-KIRK are based on the optimal parameters found in Sec. 4.6.1.3. Convergence results for a drop in residual of 12 orders of magnitude are shown in fig. 4.4.

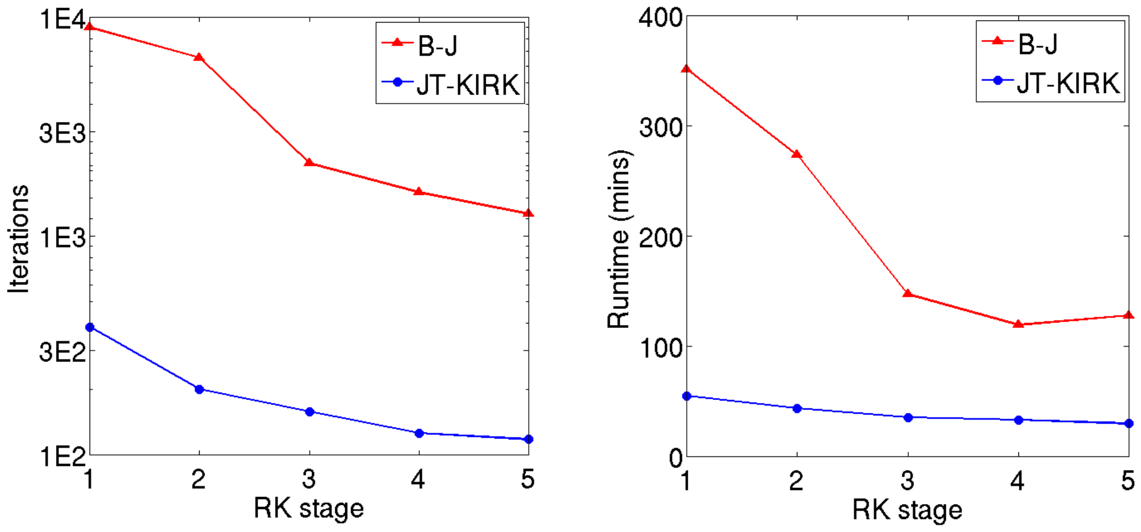

Figure 4.4: Effect of RK stage number on iteration number (left) and runtime (right) for full convergence of case1.

In all our numerical experiments RK5 outperforms RK with fewer stages in terms of iterations. B-J shows an optimum in runtime for RK4, with a slight but insignificant increase for RK5. The optimum between RK4 and RK5 for B-J is as expected, since the coefficients used have been optimised to achieve optimal high-frequency damping. For JT-KIRK, smoothing also improves as shown by the decrease in number of iterations needed. In this case runtime further decreases with the number of RK stages since the most expensive part of the non-linear step is the computation of the Jacobian and its ILU decomposition, which is only performed once for all RK stages.

A further improvement might be achieved for the JT-KIRK scheme by adding more RK stages, but as fig. 4.4 suggests, this gain will be slight. A more significant effect may be the optimisation of the stage coefficients, which however is difficult to analyse due to the nonlinear nature of the GMRES solver. As a base for comparison all results in the chapter are produced using a 5-stage Runge-Kutta multi-stage scheme with standard coefficients.

4.6.1.3 Number of multigrid levels, β, and σ

The two parameters in Eq. 4.4, β and σ, are important for stability and efficiency. To determine best values we perform a parameter study for all three solvers on Case 1.

Tab. 4.3 varies β from 0 to 1 for each scheme. All the data corresponds to converging Case 1 fully with a residual drop of 12 orders of magnitude; cases are run on a HPC cluster with 24 cores. For each tested value of β, initially using 5 levels of MG, σ is varied as $10 \leq \sigma \leq \infty$. Tab. 4.3 lists the σ for each β that achieves the lowest runtime. For the best-performing 5-level MG version of each of the three schemes, we then decrease the levels of MG (data highlighted in grey) to determine the optimal number of levels. Best overall performance is reported in bold font. The values β and σ have then been varied for these optimal combinations (results not reported here) to confirm that the combination of all three parameters is indeed optimal.

In summary, we find that

- For B-J, the best run-time performance is achieved with $\beta = 0.6$, and 4 or 5 levels of MG. Reducing the MG levels below 4 prevents convergence.
- For SGS, β=0.8 with σ=1000 and 2 levels of MG is most efficient.
- For JT-KIRK, β=0.6 with σ=∞ and 2 levels of MG is most efficient.
- For both SGS and JT-KIRK, the smallest acceptable β is around $0.4 - 0.7$, below which the scheme quickly becomes either too slow or unstable. A similar conclusion has been reported in [93] and [91].

Table 4.3: Case 1: parameter study for β, σ and multigrid level

B-J					**SGS**					**JT-KIRK**				
$1/\beta$	σ	MG level	# of iter.	time (min)	$1/\beta$	σ	MG level	# of iter.	time (min)	$1/\beta$	σ	MG level	# of iter.	time (min)
1.00	∞	5	1927	158	1.00	1k	5	212	94	1.00	∞	5	183	98
1.11	∞	5	1789	142	1.11	∞	5	197	88	1.11	∞	5	170	91
1.25	∞	5	1648	135	1.25	1k	5	182	79	1.25	1k	5	167	89
1.43	∞	5	1503	122	1.25	1k	4	182	65	1.43	∞	5	143	76
1.67	∞	5	1369	109	1.25	1k	3	184	50	1.67	∞	5	129	69
1.67	∞	4	1362	**109**	1.25	1k	2	228	**38**	1.67	∞	4	192	57
1.67	∞	1-3	>10k	>1k	1.25	1k	1	735	46	1.67	∞	3	129	44
2.00	∞	5	1580	128	1.43	10	5	2262	977	1.67	∞	2	147	**30**
										1.67	∞	1	443	34
										2.00	1k	5	144	79
										2.50	30	5	1132	584

- Overall, the best runtime is achieved with the JT-KIRK solver, which is 20% more efficient than SGS and 70% more efficient than B-J.

Both implicit smoothers are shown to work best in a 2-level multigrid when using the B-J-optimised coarsening parameters (cf. Tab 4.1). The implicit solvers offer much better smoothing than the point-implicit B-J, which should allow more aggressive coarsening. In particular the ILU preconditioner, although very effective, is expensive to compute. This cost does not reduce adequately on the coarser levels which are quite dense due to the need for semi-coarsening and the requirement of adequate mesh quality.

The B-J scheme requires semi-coarsened grids. An alternative method to alleviate the stiffness that arises from the boundary layer mesh is to construct linelets and apply a line-implicit smoother to couple the nodes in the direction normal to the wall [59]. The implicit scheme using 1st-order Jacobian, either SGS or JT-KIRK, may provide sufficient coupling in the boundary layer to either allow the threshold for semi-coarsening to be raised significantly, or to even allow isotropic coarsening, which will increase the coarsening ration significantly. To allow a comparison we have kept the B-J coarsening parameters for all cases.

In the steady-state discrete adjoint approach the computation of the approximate Jacobian matrix needs to be performed only once for each MG level throughout the entire adjoint run. Thus the ratio of runtime of both SGS and JT-KIRK compared to B-J will be even lower for the adjoint solver than for the flow solver. We use the optimal flow solver parameters of Table 4.3 also for each adjoint solver. The runtime of the flow and adjoint are shown in Table 4.4, runtimes are presented for a residual convergence of 12 orders of magnitude using 24 cores, same as for the flow calculation. For B-J, the adjoint solver requires around 3 times the runtime of the flow solver, which is typical for a discrete adjoint solver using source-transformation AD. As expected, the SGS and JT-KIRK adjoint solvers require less runtime than their respective flow solver. For JT-KIRK, the adjoint needs only 10% of the runtime compared to B-J.

Table 4.4: Case 1: runtime of different adjoint solvers, 24 cores

solver	parameter setting			runtime (min)			
	MG lvl	β	σ	flow	norm.	adjoint	norm.
B-J	5	0.6	1.67	109	1.00	355	1.00
SGS	2	0.8	1000	38	0.35	31	0.09
JT-KIRK	2	0.6	∞	30	0.28	22	0.06

4.6.1.4 Parallelism

The baseline Block-Jacobi solver is straightforward to parallelise as communication is only needed for the residual calculation of the nodes shared by multiple partitions. For implicit solvers, the communication due to solving the resulting linear system using SGS or GMRES is also not substantial as both SGS and GMRES only need the matrix-vector products which can be computed for each sub-domain and assembled in a reduction step. To avoid this reduction step, we choose to compute the Jacobian and perform the SGS or GMRES for each partition. For JT-KIRK, the ILU preconditioner is also computed for each partition separately. The GMRES solver also computed per partition only.

Fig. 4.5 shows how this simplification affects the convergence rate on Case 1 with 500k nodes. The results are produced with the optimal settings for each solver, i.e. 5 MG levels for B-J, 2-level MG for JT-KIRK. The number of iterations is barely affected by the partition size for JT-KIRK using up to 12 CPUs, or around 40k nodes per partition, but convergence does degenerate beyond 12 CPUs, While a global ILU factorisation would be prohibitively expensive to compute, using a global GMRES solve is possible and is likely to reduce the minimal partition size needed for effective convergence. However, strong scaling in iteration numbers down to a partition size of 40k nodes will be acceptable for typical industrial applications.

4.6.2 Case 2: cavity

Case 2 is a cavity between the Nozzle Guide Vane (NGV) and the rotor disks. Its geometry with labelling for the various boundaries is shown in the leftmost panel of fig. 4.6. The 'stator' boundary is attached to the NGV and thus stationary while the 'rotor' boundary is attached to the shaft and is spinning with the same speed as the rotor. A subsonic outflow boundary condition is applied to both 'seal' and 'bore' boundaries while a subsonic

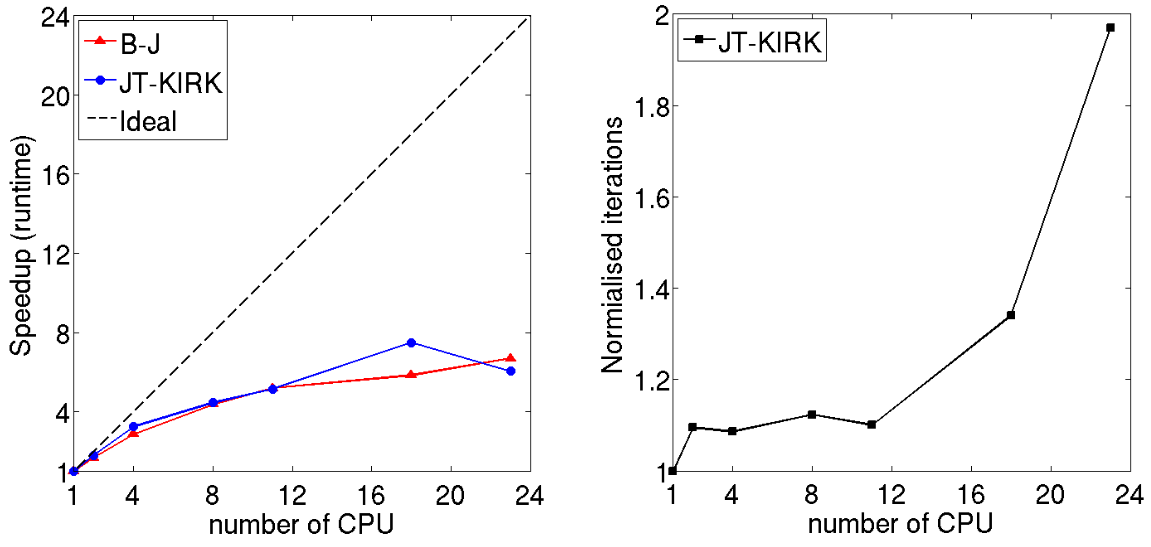

Figure 4.5: Scalability of Block-Jacobi (B-J) and JT-KIRK solvers on test Case 1 (left) and the iteration number w.r.t. the number of processors for JT-KIRK solver showing the degeneration of ILU+GMRES per subdomain (right).

inflow boundary condition is applied to the 'inlet'. The mesh has approximately 1 million hexahedral cells and is refined near the viscous wall with the maximum cell aspect ratio around 90 and a y+ of the first interior node around unity. Different from Case 1 where the main flow is driven by the axial flow, the flow in this case is dominated by wall-rotation-induced swirling flow. In addition, the majority of the flow field is low speed, a further challenge to convergence. The baseline B-J solver, although converging very slowly, does fully converge to steady state, shown in fig. 4.6. The streamline plot superimposed on the Mach number contour plot illustrates the complex flow pattern due to the strong shearing.

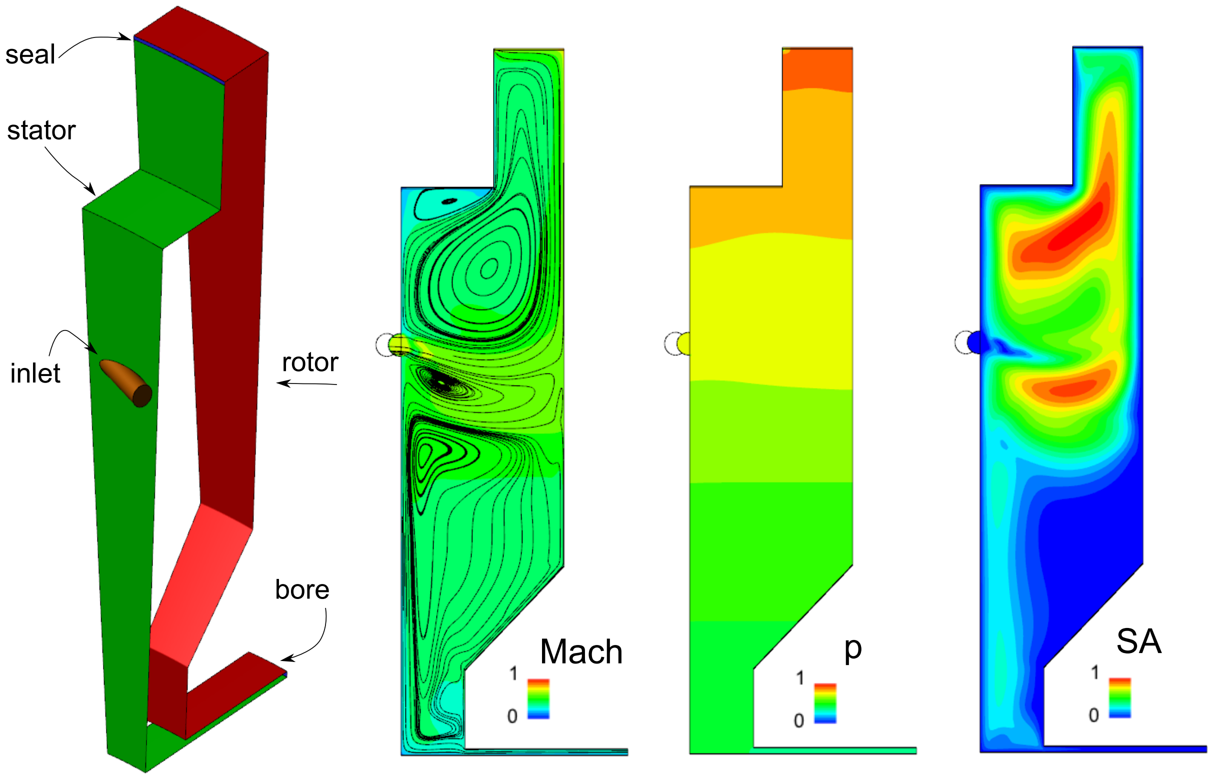

Figure 4.6: Case 2, cavity. From left to right: (1) cavity geometry, (2) Mach contours with 2D streamline, (3) static pressure contours and (4) SA variable contours. Contour plots are taken at the medium azimuthal angle. All legends are non-dimensionalised using their corresponding minimum and maximum values.

4.6.2.1 Convergence speedup

The purpose of presenting Case 2 is to assess the performance improvement of the proposed JT-KIRK algorithm for cases where viscous dissipation dominates and a stronger coupling is expected to have a significant impact on the convergence. Using the same parameters optimised for Case 1, the JT-KIRK flow and adjoint solvers are run for Case

Table 4.5: Case 2: runtime of different adjoint solvers, 12 cores

solver	parameter setting			runtime (h)			
	MG levels	β	σ	flow	normalised	adjoint	normalised
B-J	5	0.6	1.67	13.75	1.00	45.4	1.00
JT-KIRK	2	0.6	2000	6.2	0.45	5.5	0.12

2, except that the CFL number has to be lowered to 2000 for this case for startup. To permit a fair comparison, possible improvements with more advanced initialisation or ramping techniques are not investigated here and the results for a constant CFL number for the entire run are reported in Tab. 4.5. The runtime for Case 2 is significantly longer than Case 1, due to the shearing-dominating nature of this type of flows. The JT-KIRK solvers speed the flow computation by over half and the adjoint by 88%.

4.6.3 Case 3: turbine stage with rotor tip gap

Having demonstrated the efficiency gains of the JT-KIRK scheme for stable flows, Case 3 presents a mild instability that has small influence on the objective function.

Case 3 is a one-stage turbine, the rotor has a tip gap of constant clearance along the axial direction. The geometry of the stage is shown in the upper left of fig. 4.7; periodic boundaries and the casing surfaces for both the NGV and the rotor are not shown for better illustration. Total pressure and temperature are prescribed at the NGV inlet and subsonic outflow is applied at the rotor exit with prescribed static pressure. A mixing plane is used to model the NGV and rotor interaction for steady state flow. The case is meshed with 2.4 million hexahedral cells with the maximum cell aspect ratio 230. The tip gap is discretised by 25 layers of cells with a growth ratio around 1.3.

The B-J flow solver is run first with a typical setting of $\beta = 1$ and $\sigma = +\infty$ and then with alternative settings with reduced σ and increased β, but the B-J solver was never able to fully converge, at best converged to LCO, at worst diverged. Shown in fig. 4.8 is the convergence history for both the flow and adjoint with the typical setting $\beta = 1$ and $\sigma = +\infty$.

Using any snapshot of the LCO-converged flow solution will lead to divergence of the discrete B-J adjoint solver shown on the right of fig. 4.8.

To investigate the flow physics, fig. 4.7 shows various contour plots at midspan for an arbitrary snapshot within the LCO. The streamline plot in fig. 4.9 shows the flow to be mainly attached, except near the tip where the flow around the rotor reveals the complex vortex structure due to the tip gap. Although not shown here, it is confirmed that the LCO convergence corresponds to a small-scale oscillation of the stagnation zone near the

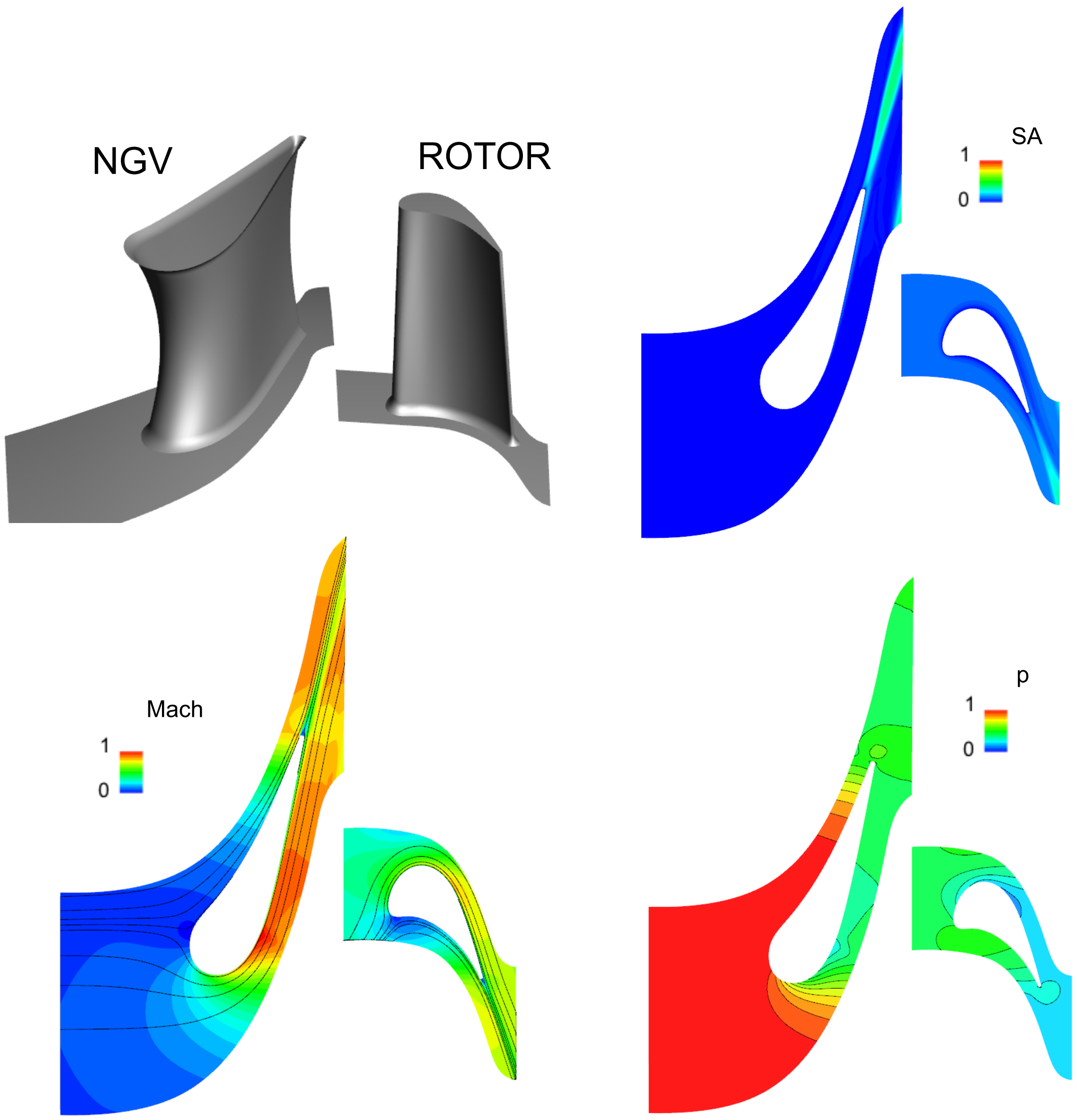

Figure 4.7: Case 3, NGV + rotor with tip gap. Upper left: NGV and rotor geometry; upper right: SA variable contour; lower left: Mach contour with 2D streamline; lower right: static pressure contour. The contour plots are taken at midspan. All legends are non-dimensionalised using their corresponding minima and maxima.

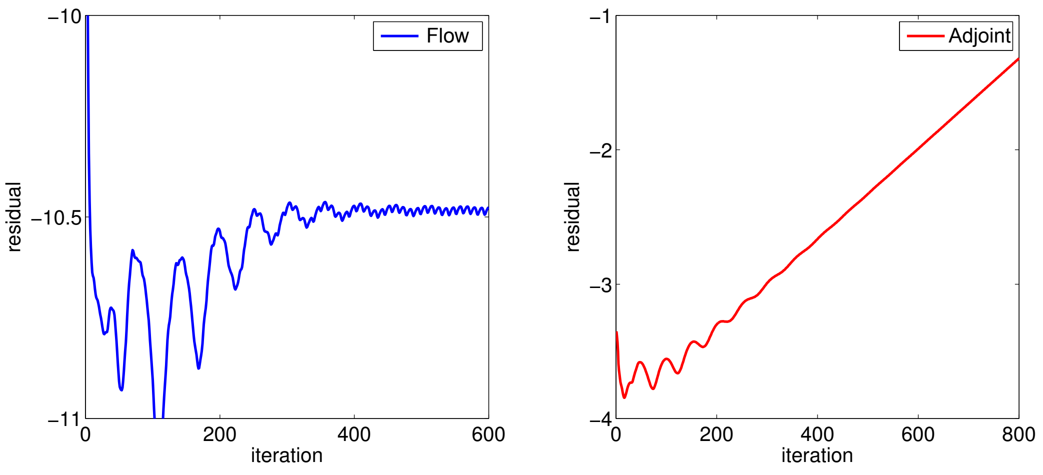

Figure 4.8: Case 3: convergence history of both Block-Jacobi flow and adjoint solvers.

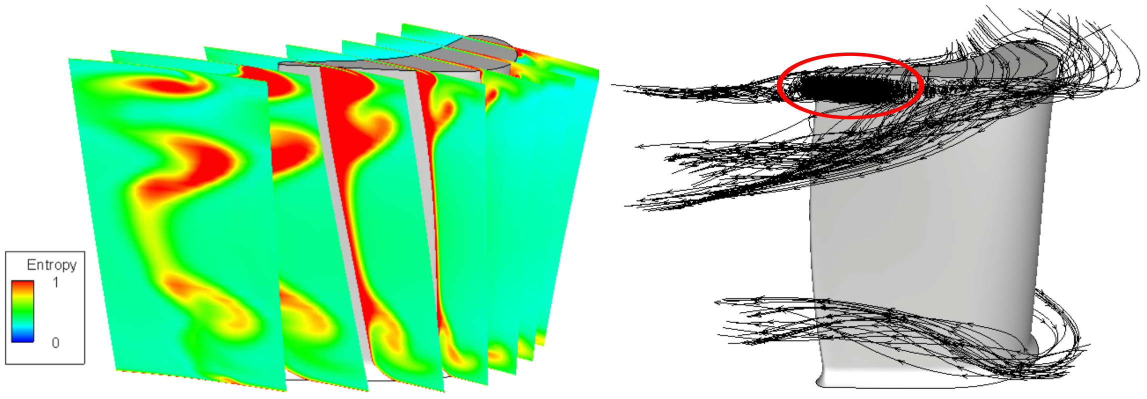

Figure 4.9: Case 3, flow visualisation around the rotor. Left: entropy contour plot at various axial locations; right: streamline illustrating the tip vortex and the two passage vortices.

rotor tip on the suction side towards the trailing edge (highlighted with the ellipse in fig. 4.9).

4.6.3.1 Stabilisation with the JT-KIRK scheme

The flow fully converges to steady state with a residual reduction of 10 orders of magnitude. using the JT-KIRK flow solver with $\beta = 1$ and $\sigma = 1000$. The fully converged flow solution is then linearised for the adjoint which also fully converges using the JT-KIRK scheme with the same parameters. The convergence history of both the flow and adjoint are shown in fig. 4.10

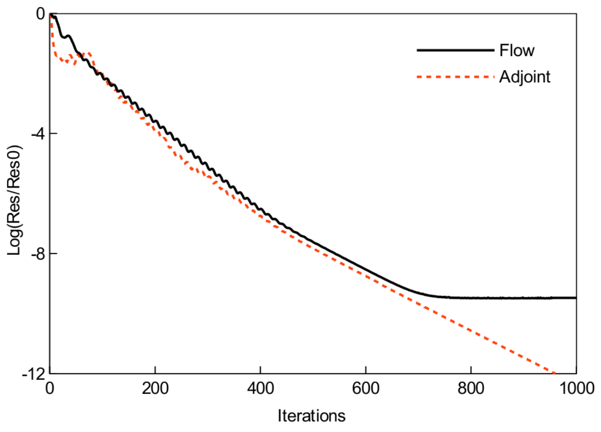

Figure 4.10: Case 3: convergence history of flow and adjoint solvers using JT-KIRK algorithm.

In the presence of possible unsteadiness both the B-J and JT-KIRK results may be questioned. While B-J does reproduce some unsteady behaviour, it is not a time-accurate scheme and the LCO may be entirely numerical. On the other hand, JT-KIRK adds a large first-order artificial viscosity in time to stabilise the discretisation. If the flow does contain unstable modes that have a nonlinear effect, this may result in a seemingly steady solution that is inaccurate.

To assess the accuracy of both the B-J and JT-KIRK results, an unsteady analysis is performed for this case using dual time-stepping with a 2-step backward differentiation formula (BDF2). The Block-Jacobi solver is used for converging fully the inner system at each physical time step. Convergence studies on both the convergence level of the inner system and time step size of the outer system have been performed to ensure that the unsteady flow simulation has converged for this particular mesh. For this comparison the focus is on resolving the unsteadiness of the blade flow and comparing that to a fully steady treatment. The mixing-plane treatment between rotor and stator has hence been maintained as in the steady case.

Table 4.6: Case 3: relative flow output deviation between fully and LCO-converged flow solution.

Quantity	JT-KIRK error %	B-J error %
Efficiency	+0.00246	-0.01208
Capacity	+0.00352	+0.00092
Reaction	-0.00385	-0.03651

Tab. 4.6 shows the errors of the objective functions for the flow solutions using the B-J and JT-KIRK solvers relative to the time-averaged objective functions of the unsteady flow simulation. The relative deviations for all the three objective functions are very small (all below 0.05%), justifying in this case the use of the JT-KIRK solver to suppress the instability and converge the flow to steady state. However, the deviations in sensitivities may be more significant than for the objective functions.

4.6.3.2 Effect of flow convergence level on adjoint stability

In the stabilisation results shown in the previous subsection the JT-KIRK adjoint solver was applied to the fully converged solution while the B-J adjoint solver was applied to the LCO-converged flow The stabilisation could be due to the more contractive Jacobian of JT-KIRK or the better convergence level of the flow that the Jacobian is based on. To decouple the effect of flow convergence level and adjoint time-stepping algorithm, two additional runs are performed: (i) application of the JT-KIRK adjoint solver to LCO-converged flow and (ii) application of the B-J adjoint solver to fully-converged flow. The results are summarised in Tab. 4.7.

The B-J adjoint based on the fully converged flow diverges, while on the other hand, for this flow JT-KIRK is able to converge the adjoint solution based on the LCO-converged flow solution. The latter is not guaranteed as the preconditioned Jacobian $\mathbf{A} := I - \mathbf{P}^{-T}(U)\mathbf{L}^{T}(U)$ depends on the nonlinear flow solution U about which the adjoint is linearised. The fact that the JT-KIRK adjoint is also stable for the LCO-converged flow in this case can probably be attributed to the fact the flow deviation is small, implied by the small deviation of the objective functions shown in Table 4.6. It will be shown in Sec. 4.6.4 that this is not the case for cases with stronger unsteadiness.

4.6.4 Case 4: turbine stage with skewed rotor

Case 4 is a one-stage turbine with a NGV and a rotor. Different from Case 3, the rotor does not have a tip gap (geometry shown in fig. 4.11 upper left). Instead, in order to investigate the effect of large unsteadiness on the convergence of both the flow and adjoint

Table 4.7: Case 3: effect of flow convergence level and adjoint scheme on the adjoint convergence. The solver setting for all the JT-KIRK flow and adjoint runs are the same.

Set	Flow solver	Flow convergence	Adj. solver	Adj. convergence
A	JT-KIRK	Full convergence	JT-KIRK	Full convergence
B	JT-KIRK	Full convergence	B-J	Divergence
C	B-J	LCO	JT-KIRK	Full convergence
D	B-J	LCO	B-J	Divergence

solver, the angle of attack of the rotor blade has been deliberately changed significantly away from the design condition to create a massive flow separation on the rotor suction side. The case consists of 0.6 million hexahedral cells with maximal cell aspect ratio of 75.

Similar to Case 3, the baseline B-J solver only converges to LCO, stalling after an initial drop of the residual by 2 orders of magnitude. The convergence curve in fig. 4.12 is presented for $\beta=0.6$ and $\sigma=+\infty$. The discrete adjoint based on the flow solution at the 400-th iteration diverges after around 100 adjoint iterations. Using a different snapshot of the flow solution, a smaller σ and/or different β does not achieve convergence; the lack of convergence is due to outlying eigenvalues as demonstrated by the eigen-analysis in Sec. 4.6.4.2.

An arbitrary snapshot of the flow solution taken during the LCO is shown in fig. 4.11. The streamlines around the rotor at midspan clearly show the separation zone with a separation point close to the leading edge (marked with arrow in lower left of fig. 4.11).

4.6.4.1 Stabilisation using JT-KIRK solvers

Case 4 has a much stronger instability, hence for this case we assess the robustness of the stabilisation effect of the JT-KIRK. Differently from Case 3, the strong instability may result in significant differences of objective functions predicted by an unsteady or the steady-state B-J and JT-KIRK approaches.

When applying JT-KIRK to Case 1, fastest convergence was obtained with β=0.6. This choice is actually not stable for Case 4. A more accurate solve of the linear system using more Krylov vectors or a lower convergence threshold does not help to achieve full convergence, however increasing the under-relaxation to $\beta = 2.0$ does, while the CFL number can remain at σ=$+\infty$. Optimal runtime for this combination is achieved with 2 levels of MG, solver settings and convergence results are shown in Tab. 4.8

Similar to Case 3, a time-accurate unsteady simulation is performed to assess the accuracy of the values of the objective functions of the LCO-converged flow solution from

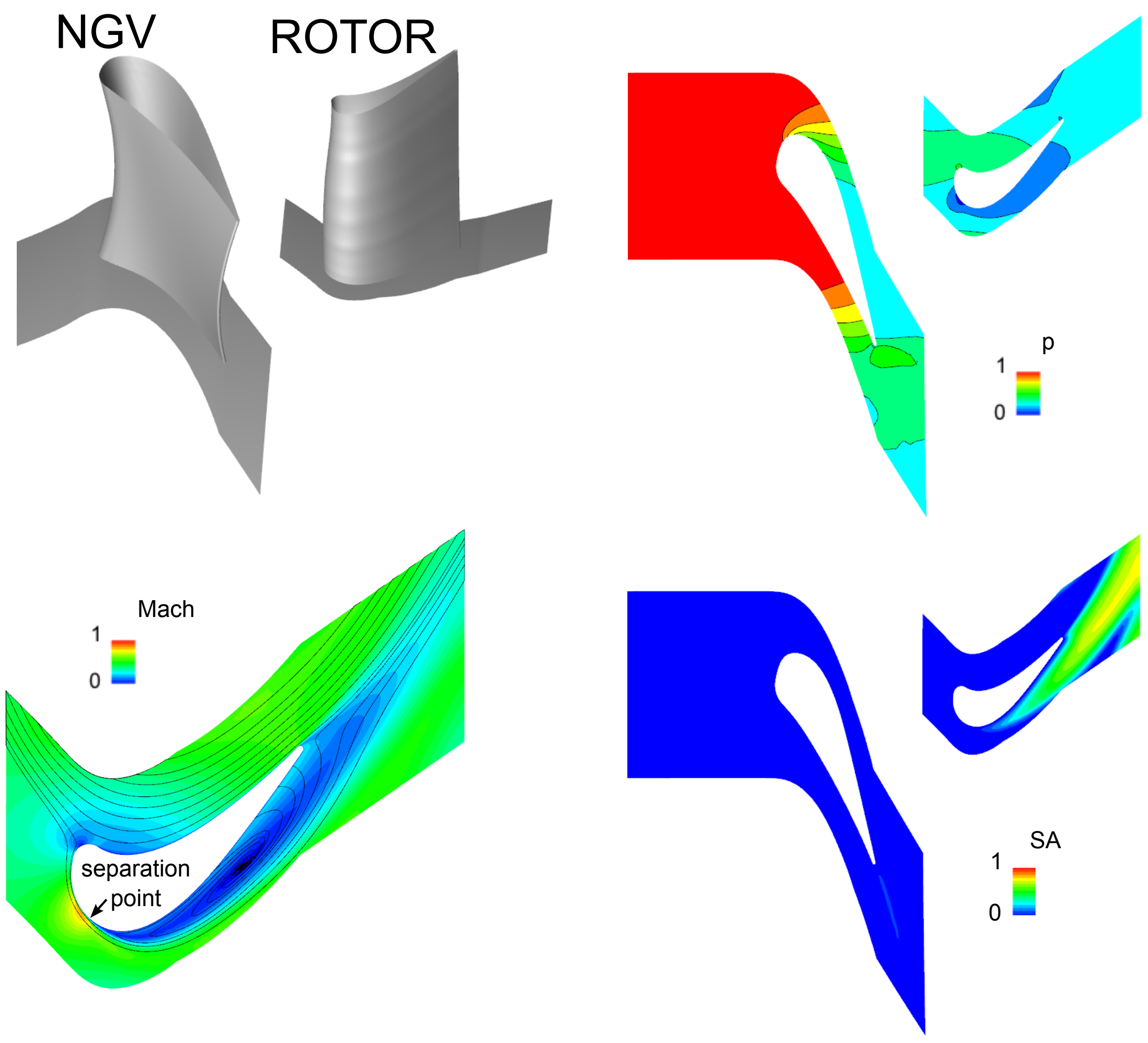

Figure 4.11: Case 4, NGV and skewed rotor. Upper left: NGV and rotor geometry; upper right: static pressure contour; lower left: Mach contour with 2D streamline for the rotor only; lower right: SA variable contour. The contour plots are taken at midspan. All legends are non-dimensionalised using their corresponding minima and maxima.

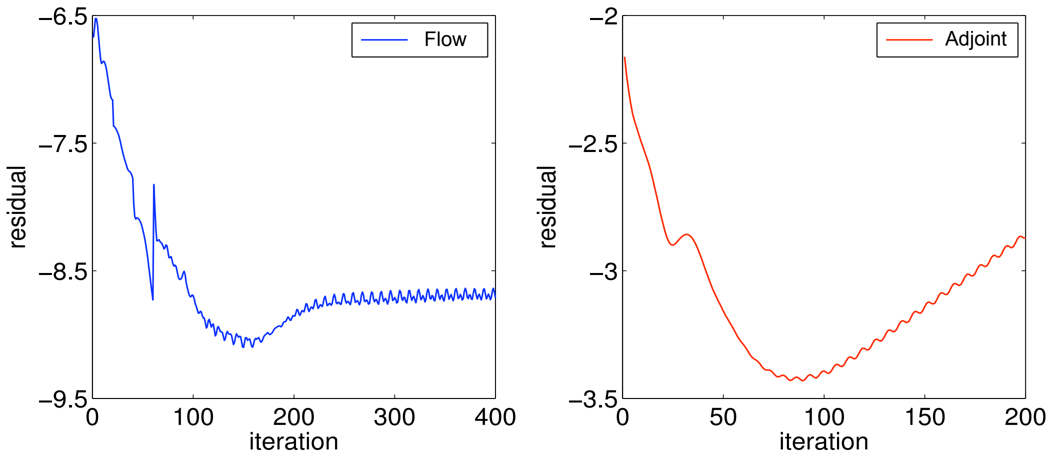

Figure 4.12: Case 4, LCO convergence of the flow solver and the divergence of the adjoint solver.

Table 4.8: Case 4: solver setting B-J and JT-KIRK, (results from settings highlighted in bold are shown in fig. 4.11 and table 4.9 (DNC=Did Not Converge).

solver	β	σ	MG level	iteration	runtime(min)
flow (B-J)	0.6	1.67	4	DNC	DNC
adjoint (B-J)	0.6	1.67	4	DNC	DNC
flow (JT-KIRK)	0.6	$+\infty$	4	DNC	DNC
flow (JT-KIRK)	1.0	$+\infty$	4	DNC	DNC
flow (JT-KIRK)	1.5	$+\infty$	4	DNC	DNC
flow (JT-KIRK)	2.0	$+\infty$	4	1042	1750
flow (JT-KIRK)	2.0	$+\infty$	2	1058	801
adjoint (JT-KIRK)	2.0	$+\infty$	2	1096	507

Table 4.9: Case 4: relative flow output deviation between fully and LCO-converged flow solution.

Quantity	JT-KIRK error %	B-J error %
Efficiency	-0.03124	+0.10064
Capacity	+0.00052	-0.00035
Reaction	-17.24848	+23.12120

B-J and the fully converged flow solution from JT-KIRK. Convergence studies have been performed for the unsteady solver to ensure that the prescribed level of convergence for solving the inner system is low enough and that the time step is chosen small enough. The errors of the objective functions from LCO-converged B-J and fully-converged JT-KIRK relative to the unsteady simulation are summarised in Tab. 4.9. The deviations of both the efficiency and capacity of the steady-state approaches are small (below 0.1%), but the reaction ratio of both B-J and JT-KIRK solutions differs significantly from the unsteady average, with the deviation having comparable magnitude for both schemes at around 20%, but with opposite signs.

On one hand, the stabilisation effect of the JT-KIRK solver is very strong thus managing to fully converge a case with such strong unsteadiness. Such strong flow separations or other severe instabilities may occur for initial or intermediate designs, and it is important that the sequence of design iterations is not broken by a diverging adjoint. Reduced accuracy of objective functions and sensitivities may be perfectly acceptable in this phase.

On the other hand, it implies that extra care needs to be taken when such a strong stabilisation is used. The user needs to carefully assess when objective functions obtained from steady approaches are acceptable, and when unsteady simulations are required. Providing guidance on this goes beyond the scope of this thesis, however two observations can be made. The comparison of cases 3 and 4 suggests that if the JT-KIRK adjoint is able to converge based on the flow solution converged to LCO by the B-J flow solver, this may indicate small unsteady deviations resulting in negligible changes in objective function. However, this is a rather arbitrary threshold and the second observation is that the deviation of the objective function is strongly linked to its non-linearity: two of the objectives in Case 4 only show small errors, while reaction ratio strongly deviates.

4.6.4.2 Eigen-analysis of the stabilisation mechanism

To better understand the effect of the JT-KIRK scheme on the convergence of both the flow and the adjoint solvers, eigen-analysis is performed on the Jacobian of both the B-J and JT-KIRK adjoint solvers. For the B-J solver, we take as before the limit-cycle

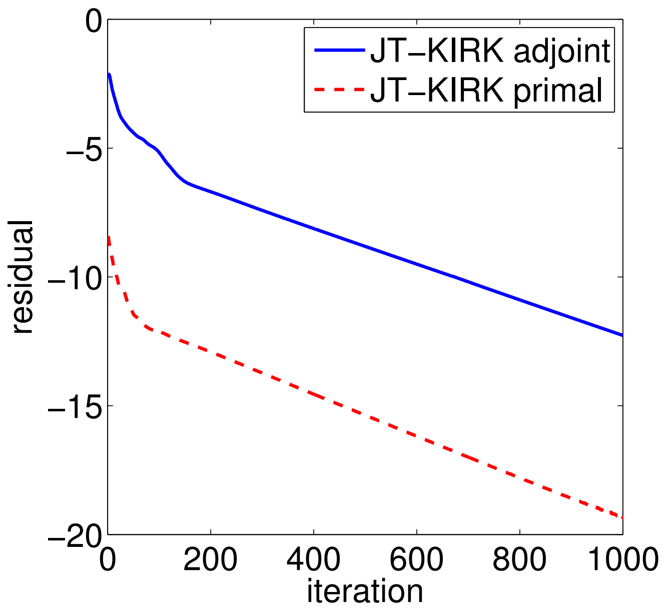

Figure 4.13: Case 4: convergence history of both JT-KIRK flow and adjoint solvers.

flow solution at the 400-th iterations of the B-J solver. For JT-KIRK, we take the fully converged flow solution.

To calculate the dominant eigenvalues we use Arnoldi iterations as described by Campobasso et al. [13]. Strictly speaking, the Arnoldi iteration can only be applied to a linear operator, such as a linear multigrid smoother. The smoother used in the JT-KIRK algorithm has a GMRES solver wrapped inside a RK multi-stage scheme. Due to the nonlinear nature of the GMRES solver, the overall smoother is not a linear operator, and thus the Arnoldi iterations may fail to correctly compute the eigen-spectrum. However, the specific novelty of the JT-KIRK scheme is to wrap the GMRES into a RK multi-stage scheme to achieve high-frequency damping, it is this attribute that makes the solver suitable to be used as a smoother within multigrid. Similarly, this diminishes the non-linear character of GMRES sufficiently that the effects of the linear RK multi-stage scheme and the linear multigrid scheme dominate. As a confirmation, the magnitude of the largest eigenvalue is found to be in excellent agreement with the asymptotic convergence/divergence rate of the adjoint solver, demonstrating the validity of the eigen-analysis.

150 dominant eigenvalues are computed for both the B-J and JT-KIRK solvers, as shown in fig. 4.14. The eigenvalues for B-J include two outliers which are responsible for the lack of full convergence, while the JT-KIRK solver shows all eigenvalues well within the linear stability boundary and thus the JT-KIRK adjoint can converge fully.

Similar to the experiment with Case 3, it is also attempted to converge the adjoint

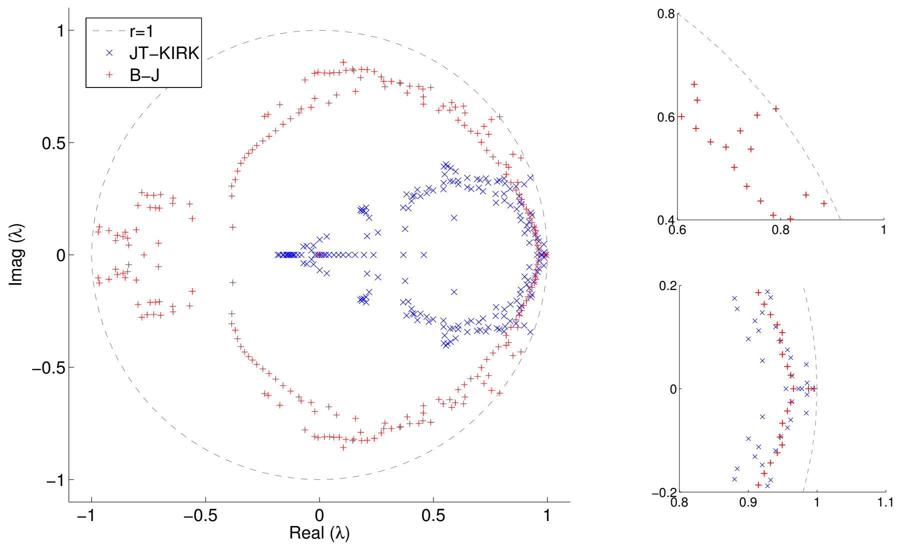

Figure 4.14: The dominant eigenvalues values for both the B-J and JT-KIRK solvers produced using Arnoldi iterations. The zoomed region on the top right show the unstable mode.

using another two time-stepping combinations for the adjoint: (i) JT-KIRK for the adjoint based on the LCO-converged flow solution using B-J and (ii) B-J adjoint solver on the fully converged JT-KIRK flow solution, results are summarised in Table 4.10.

Table 4.10: Case 4: effect of flow convergence level and adjoint scheme on the adjoint convergence. The solver setting for all the JT-KIRK flow and adjoint runs are the same.

Set	Flow solver	Flow Convergence	Adjoint solver	Adjoint Convergence
A	JT-KIRK	Full convergence	JT-KIRK	Full convergence
B	JT-KIRK	Full convergence	B-J	Divergence
C	B-J	LCO	JT-KIRK	Divergence
D	B-J	LCO	B-J	Divergence

Differently from Case 3, the JT-KIRK adjoint solver using the same setting does not manage to stabilise the adjoint for LCO-converged flow for Case 4. Eigen-analysis is performed for all the four combinations in Table 4.10 and spectra are shown in fig. 4.15. The magnitude of the largest eigenvalue in every plot is confirmed to be in agreement with the convergence/divergence rate of the respective adjoint solves. The spectra of the two JT-KIRK adjoint solutions (A and C in fig. 4.15) are very similar, both showing significant clustering of the eigenvalues compared with their counterparts using the B-J adjoint solver on the right. However, the spectrum of the adjoint solver for LCO-converged flow (C in fig. 4.15) has two outliers that cause the adjoint to diverge. This is different from the results for Case 3, where the JT-KIRK adjoint solver is stable for both the fully converged and the LCO-converged flows. As stated in the discussion of Case 3, there is no guarantee for discrete adjoint convergence based on a non-contractive Jacobian even when using a stronger solver for the adjoint. In Case 4 the instability of the flow is large enough to prevent convergence of discrete adjoints based on the LCO state, even with JT-KIRK. Some fine tuning of the JT-KIRK adjoint solver parameters might make the adjoint solver stable also for the B-J LCO linearisation, but using JT-KIRK for both flow and adjoint is guaranteed to converge since JT-KIRK does converge the flow.

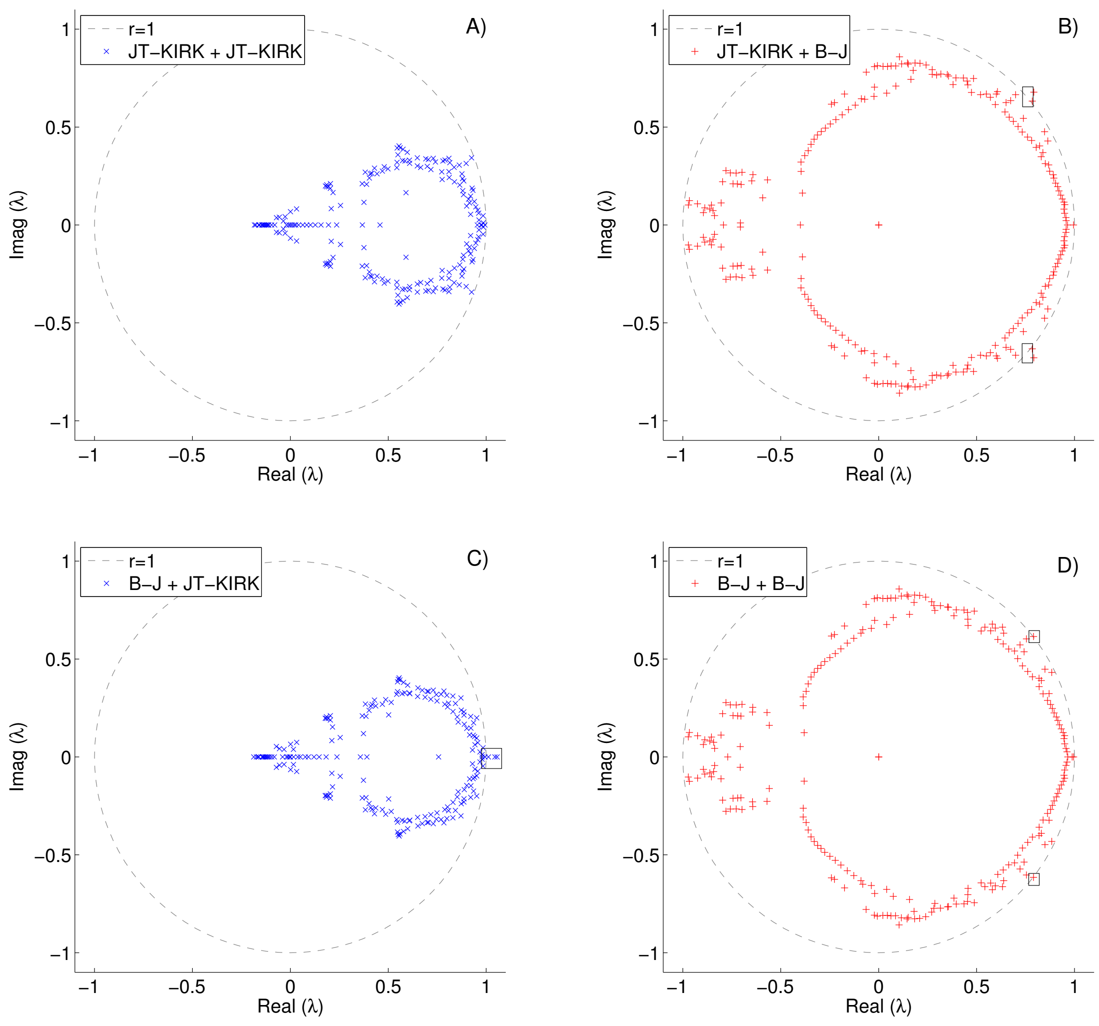

Figure 4.15: The eigenvalues for the solver combinations of Tab. 4.10. Outlying eigenvalues with $|\lambda| > 1$ are highlighted in boxes.

4.7 Summary

A robust implicit time-stepping scheme for the steady flow and adjoint RANS equations has been presented to address the lack of robustness of the typical Block-Jacobi (point-implicit) multigrid solvers when calculating steady state discrete adjoint solutions for cases with mild physical or numerical instabilities. The JT-KIRK algorithm has two main features. First, as opposed to Newton-Krylov methods that use exact Jacobian-vector products, a 1st-order-approximate Jacobian is formed, stored and approximately inverted using ILU-preconditioned GMRES. The Jacobian is computed using the Automatic Differentiation tool Tapenade [35] in vector-forward mode. Secondly, the algorithm uses the Krylov solver as a smoother within a FAS-cycle multigrid method. This is achieved by embedding the linear solver inside a Runge-Kutta multi-stage time-stepping scheme, which provides high-frequency damping similarly to explicit smoothers.

For a typical explicit or point-implicit compressible RANS solver using Runge-Kutta time-stepping and multigrid, only moderate change to the iterative scheme is needed to upgrade existing flow and adjoint solvers to JT-KIRK. The memory required by the implicit solver is approximately 7 times larger than the total memory required by the the reference Block-Jacobi solver to store the approximate Jacobian and its ILU decomposition. The use of GMRES inside multigrid produces optimal performance using only three Krylov vectors which does not affect the overall memory requirement significantly.

The implicit solver is applied to four representative test cases from turbo-machinery applications. For cases 1 and 2 where the baseline Block-Jacobi solver can fully converge, the JT-KIRK solver is much more efficient than the Block-Jacobi solver with 28% of the runtime for the flow solver, and slightly more efficient than the implicit solver using Symmetric Gauss-Seidel (SGS) with 79%. The performance advantage is much more significant for the adjoint, since the expensive preconditioners have to be built only once. The adjoint JT-KIRK uses only 6% of the runtime of the adjoint Block-Jacobi, and 70% of that of SGS. Further improvements in runtime can be expected by adapting the multigrid coarsening strategy inherited from the Block-Jacobi scheme to the improved smoothing of the implicit solver, and by selectively applying Automatic Differentiation to the fluxes omitting costly but less relevant elements of the Jacobian.

For the unstable test cases the Block-Jacobi flow solver converges only to limit cycle oscillations and consequently the discrete adjoint solver diverges due to a non-contractive system Jacobian. The JT-KIRK algorithm fully converges flow and adjoint, even if the unsteadiness is severe. The improvement in robustness is analysed by examining the eigenvalues of the JT-KIRK scheme. The spectral analysis conducted for a strongly unstable case confirms that the stabilisation is brought about by JT-KIRK moving the outliers of the Block-Jacobi time-stepping inside the stability boundary.

Numerical experiments were conducted with different solver combinations for flow and adjoint to determine whether the instability is transient, and a strong and expensive solver is only needed for the primal, or whether it is still present in the fully converged solution. It is found that, as the theory suggests, the instability is persistent, i.e. if the flow requires a strong solver to converge, convergence of its discrete adjoint is only guaranteed if the same solver is used for the adjoint. However, the results also show that in the case of mild unsteadiness a stronger adjoint solver may be able to converge the adjoint based on a flow converged only to limit cycle oscillations.

For the unstable cases the choice of iterative solver does affect the values of typical objective functions for turbo-machinery applications. In the case of mild unsteadiness, even though the Block-Jacobi solver does not converge and its discrete adjoint diverges, the steady-state approaches can provide objective functions with good accuracy compared to an averaged objective functions evaluated by fully unsteady calculations. For cases with pronounced instability, the objective functions of both the Block-Jacobi and the JT-KIRK steady-state approaches deviate strongly from the unsteady average. While stable with JT-KIRK, the use of the steady-state approach is not justifiable in this case, moreover the gradient sensitivity will vary much more dramatically. The JT-KIRK algorithm therefore extends the applicability of the steady-state discrete adjoint approach to marginally stable flows without compromising the efficiency for stable cases.

Despite the success of stabilising the discrete adjoint using the JT-KIRK algorithm, the robustness improvement may converge to a physically incorrect steady state solution when the flow has significant or strongly non-linear unsteadiness. In those cases, unsteady flow and adjoint solvers should be used to accurately reflect the flow physics.

Chapter 5

Mesh deformation techniques

5.1 Introduction

For shape optimisation, the computational mesh has to be updated at every design step. This includes both the surface and volume meshes. Once the surface mesh is perturbed following the updated boundary geometry, the volume mesh needs to be updated as well. This can be done either by re-meshing or by volume mesh perturbation. To perform re-meshing, one needs a mesher that can automatically mesh the computational domain using the perturbed surface geometry or mesh. The advantage is its robustness in generating a valid mesh for the perturbed shape in the presence of a large perturbation of the surface. The main disadvantage is that re-meshing is very time-consuming for large industrial cases. Most importantly for gradient based optimisation, the re-meshing varies the truncation error, introducing noise into the sequence of gradients, hence severely impairing application of Quasi-Newton optimisation algorithms. If the surface displacement is small enough not to result in a topology change in the multi-block unstructured mesh, then using an iterative mesh deformation algorithm is more appropriate and cost-effective. A mesh deformation algorithm updates the existing volume mesh by computing a finite perturbation field according to the known boundary mesh displacement, and superimposing the volume mesh perturbation field to the existing volume mesh onto obtain the perturbed volume mesh. The mesh deformation algorithm is usually PDE-based solver which can be easily included in the existing solver framework.

Broadly speaking, for unstructured meshes, the widely used methods for mesh deformation are (1) analytic, (2) spring-analogy, (3) linear elasticity and (4) radial basis function. The analytic approach is a general approach that first maps the volume mesh onto a background object that has an analytic description and can be deformed smoothly. The smooth deformation of the analytic object is then mapped back onto the matched volume mesh nodes. Two typical examples of this approach include the free-form deformation [85] and volumetric B-Splines [56]. The spring-analogy approach sees the volume mesh as a mass-spring system. The simplest approach is to model the edge as a spring with stiffness inversely proportional to its length. This basis version spring-analogy approach is called tension-spring. An improved version that includes the torsion into the spring model, i.e., the torsional-spring approach, is more robust in preventing the cells from deforming to negative volume [3]. The linear elasticity approach views the domain as an elastic body and solves for the 'physical' displacement of the elasticity body under prescribed displacement certain boundary mesh nodes. Although the structural analysis is traditionally discretised using finite element approach, it can also be discretised in finite volume method, to better fit into an FVM nonlinear flow solver. One efficient implementation is described in [41]. Radial basis functions (RBFs) based mesh deformation is becoming a well-established method due to its robustness [22]. RBFs uses a linear combination of various radial basis functions to interpolate the volume mesh deformation, and the coefficients for each basis function are determined using the known displacement of the design surface nodes. One of the disadvantages is that it's also quite expensive to solve with the number of degrees of freedom equal to the number of surface nodes.

In terms of robustness, linear elasticity and radial basis function approaches are more favourable, although more expensive. It is the author's view that the linear elasticity approaches offers more flexibility with the freedom of tuning the material properties in order to localise the deformation and has better physical mechanism of preventing cells of negative volume from appearing. In addition, the linear elasticity equation can be solved easily with the same FVM discretisation requiring minimal coding effort. Therefore, spring-based and linear-elasticity based deformation algorithms are implemented in this work for comparison. In this chapter, both algorithms are explained in terms of theory and implementation, and are both applied to a few test problems to demonstrate their respective capabilities.

5.2 Spring analogy

The spring-based mesh deformation algorithm assumes the volume mesh nodes are connected through springs (illustrated in fig. 5.1), represented by the edges, whose stiffness matrix is defined as

$$[K_{ij}]_{3\times 3} = \frac{1}{l}\hat{n}_{ij} \otimes \hat{n}_{ij}$$

where the distance l and the unit normal vector $\hat{n}_{ij}$ are defined as

$$l = \|X_i - X_j\|_2 \qquad \text{and} \qquad \hat{n}_{ij} = \frac{X_i - X_j}{l}$$

with X_i and X_j being the coordinates of the two end nodes of the edge i-j. The system

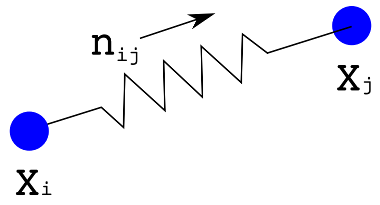

Figure 5.1: Illustration of the node-edge spring system.

of equations for the perturbation δX_j is then

$$K_{ij}\delta X_j = \delta \tilde{X}_i$$

or

$$\mathbf{K}\delta X = \delta \tilde{X}$$

In order to form a square system matrix $\mathbf{K}$, the right hand side term of the spring system uses a forcing term formulated on the whole volume mesh $\delta\tilde{X}$, which is defined as

$$[\delta\tilde{X}]_{N\times 1} = [K_{s2v}]_{N\times N_b}[\delta X_s]_{N_b\times 1}$$

where N and N_b denotes the total number of volume nodes and mesh nodes and matrix K_{s2v} is a permutation matrix mapping the known perturbation of each surface node to its corresponding volume node. K_{s2v} is defined as: $K_{s2v}(i,j) = 1$ if i-th volume node is j-th surface node and $K_{s2v}(i,j) = 0$ if i-th volume node does not correspond to any surface node. Furthermore, the system matrix K_{ij} is modified such that if i-th or j-th volume node is also a surface node, then the diagonal 3-by-3 matrix is an identify matrix, since the grid displacement is known. This can be done by setting K_{ij} to a 3-by-3 identity matrix if the i-th volume node is the j-th surface node.

The forward mapping from δX_s to δX, i.e., compute the volume node displacement due to the known displacement of the surface nodes, is then represented by the rectangular matrix

$$\hat{\mathbf{K}} = \mathbf{K}^{-1}K_{s2v}$$

The volume node displacement can be computed using prescribed surface mesh displacement as

$$\delta X = \hat{\mathbf{K}}\delta X_s.$$

To clarify the notation, $\mathbf{K}$ is the square block-diagonal matrix for the entire domain and the hatted matrix $\hat{\mathbf{K}}$ is a rectangular matrix mapping the surface node displacement to the volume node displacement.

The mesh deformation algorithm above is for the ‘forward’ computation, i.e., given a known surface displacement, compute the volume displacement. For adjoint-based optimisation, one would like to compute the surface-based sensitivity from the known volume-based sensitivity, the so-called 'reverse' computation of the mesh deformation, i.e.,

$$\frac{dJ}{dX} \quad \Longrightarrow \quad \frac{dJ}{dX_s}$$

Formally, since

$$\frac{dJ}{dX_s} = \frac{dJ}{dX}\frac{dX}{dX_s} = \frac{dJ}{dX}\mathbf{K}^{-1}K_{s2v}$$

we have

$$\left(\frac{dJ}{dX_s}\right)^T = K_{s2v}^T\mathbf{K}^{-T}\left(\frac{dJ}{dX}\right)^T$$

Left multiplying with the matrix $\mathbf{K}^{-T}$ is equivalent to solving the spring system with the transposed stiffness matrix, using the volume based sensitivity $(dJ/dX)^T$ as the driving term. In addition, recognizing that the effect of left multiplying with the matrix K_{s2v}^T is to map a volume node to its corresponding surface node if any, the projection of the volume sensitivity to the surface sensitivity consists of three steps: (i) compute and store the transposed stiffness matrix $\mathbf{K}^T$, (ii) compute $\mathbf{K}^{-T}(dJ/dX)^T$ through solving the transposed spring system and (iii) zero the sensitivity of those volume nodes that are not surface nodes.

For both the 'forward' and 'reverse' computations of the mesh deformation, the linear system to be solved involves a block-wise sparse matrix with 3-by-3 block matrices. This well-conditioned system is solved relatively easily with Jacobi preconditioned BiCG solver. The residual is reduced by ten orders of magnitude, and the computational cost is negligible compared to solving the flow and adjoint.

Despite its simplicity in implementation, spring-analogy based mesh deformation is usually effective only for deforming isotropic meshes with small displacement. The main drawback is that this approach does not have any mechanism to prevent cells of negative volumes before the deformation is solely dependent on the distance between nodes. An improved version, the so-called torsional spring model [23], has been proposed to demonstrate better robustness. However, according to the author's experience, even a torsional spring model still is not very useful in dealing with anisotropic mesh with very high aspect ratio cells in the boundary layer. Therefore, a much more robust mesh deformation algorithm based on linear elasticity, explained in the following section, is implemented for the shape optimisation results in this thesis.

5.3 Linear elasticity

The linear-elasticity-based mesh deformation treats the whole CFD computational domain as a volume of elastic material. The design surface is prescribed with a known displacement and the displacement of the volume nodes are solved for using the static linear elasticity governing equation without external forcing term:

$$\nabla \cdot \sigma = 0 \tag{5.1}$$

where σ is the stress tensor, which for isotropic material can be defined as

$$\sigma = 2\mu\epsilon + \lambda \mathbf{Tr}(\epsilon) I$$

where μ and λ are the Lamé constants related to Young's modulus E and Poisson's ratio ν through

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}, \qquad \mu = \frac{E}{2(1+\nu)}$$

The strain tensor ϵ is defined as

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

where u is the displacement vector field. Dirichlet boundary conditions, i.e., u=g on $\partial\Omega$, are used on design surfaces where the displacement is known.

5.3.1 Basic implementation

In order to solve the problem in an FVM framework Eq. (5.1) is reformulated to integral form

$$\oint_{\partial\Omega} \left(\frac{E}{2(1+\nu)}(\nabla u + \nabla u^T) \cdot \hat{n} + \frac{\nu E}{(1+\nu)(1-2\nu)} \nabla \cdot u\hat{n} \right) dS = 0$$

which is simplified to

$$\oint_{\partial\Omega} \frac{E}{2(1+\nu)} \left(\mu(\nabla u + \nabla u^T) \cdot \hat{n} + \lambda \nabla \cdot u\hat{n} \right) dS = 0$$

where ν is between 0.0 and 0.5, and consequently $\lambda = \dfrac{2\nu}{1-2\nu}$ is between 0.0 (for λ=0.0) and $+\infty$ (for λ=0.5).

The solution method is based on [41] and [11]. The successful implementation relies on the setting of Young's modulus and Poisson's ratio. Recall that a large Young's modulus value makes the element stiff and require large stress to deform, while a large Poisson's ratio, e.g., 0.5 for 3D, makes the element incompressible. They are two distinctive properties and can cause confusion if not distinguished. A simple and robust approach, according to [11], is to set the Young's modulus inversely proportional to wall distance and set Poisson's ratio everywhere to zero. The first criterion is to ensure the boundary layer meshes are deformed much less than the isotropic mesh away from the boundary layer meshes.The second one is extremely important to make sure that the material can tolerate arbitrary volume change without building up excessive internal stress.

The linear elasticity module is applied to a few test cases to demonstrate its capability. In fig. 5.2 and fig. 5.3, a circular cylinder undergoes a change in diameter, rotational and translational movements. In fig. 5.4, a cantilever (rectangular wing with rectangular cross section) meshed with high AR meshes near the wing surface undergoes bending and torsion.

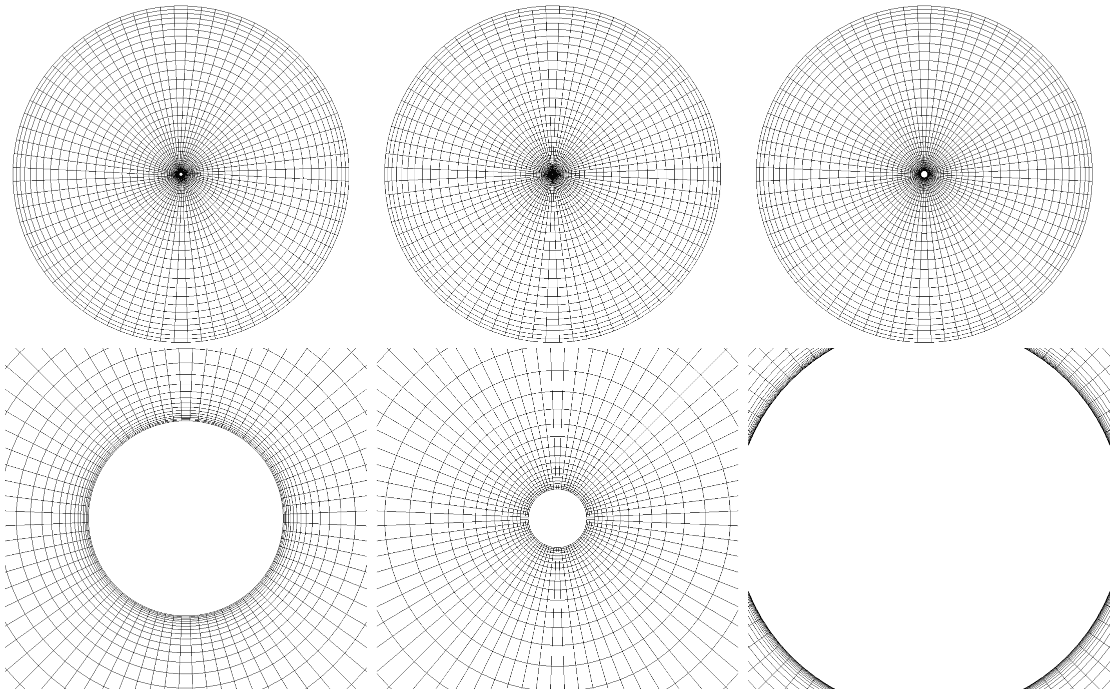
Figure 5.2: Left to right: original mesh, cylinder scaled ×0.3, scaled ×2. Top: overview, bottom: zoomed-view.

5.3.2 Improved implementation

The quality of the deformed mesh can be further improved by raising the order of the Young's modulus. Instead of setting Young's modulus to be inversely proportional to wall distance

$$E(x) = \frac{1}{d(x)}$$

a higher order can be used, such as

$$E(x) = \frac{1}{d(x)^n},\ n > 1$$

The larger n is, the stiffer the elements in the viscous boundary layer. A comparison of using n=1, 2, 3 and 4 for the twisted cantilever is shown in fig. 5.5. A near rigid body movement of the element near wall and a better smoother transition of the mesh

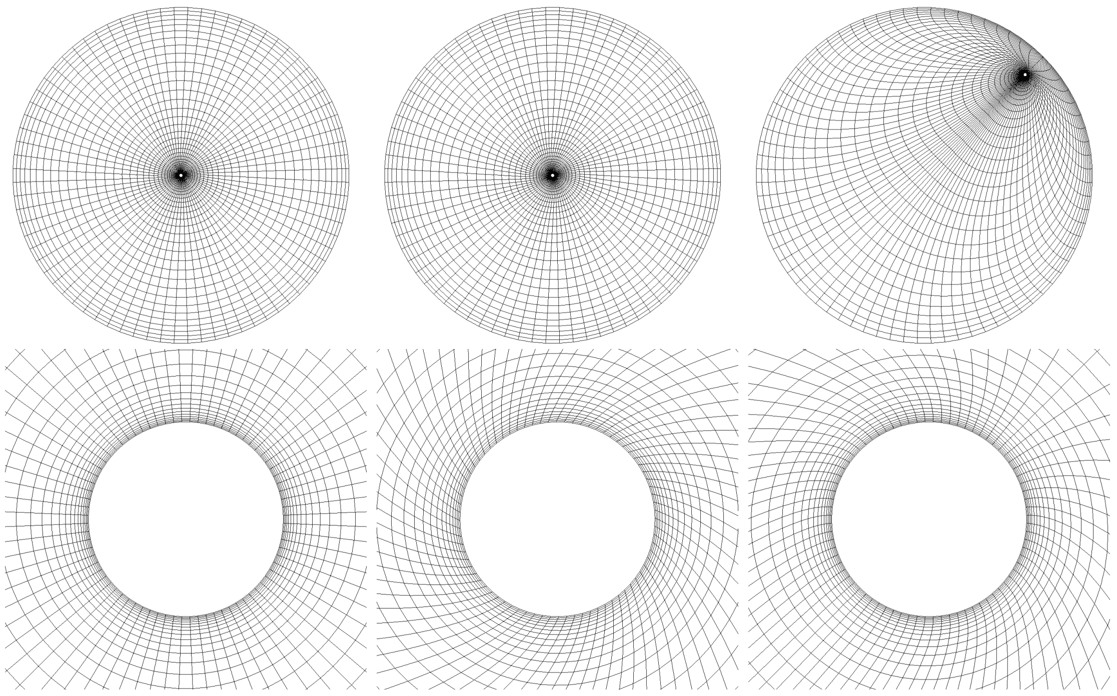
Figure 5.3: Left to right: original mesh, cylinder counter clock-wise rotate 60°, cylinder translated 60D in both x and y directions. Top: overview, bottom: zoomed-view.

deformation field is observed for n=2, 3 and 4. Therefore, for all the mesh deformation results from here on, n=2 is the default setting.

Due to the spatially varying distribution of Young's modulus, the resulting linear system is no longer symmetric, thus the ILU(0) preconditioned GMRES, rather than a CG solver, is used for solving the linear elasticity equation.

5.3.3 Approximate wall distance calculation

In the linear elasticity based mesh deformation algorithm, Young's modulus is scaled according to the distance to viscous wall. Therefore a fast algorithm for calculating the distance to wall is needed. The ray tracing algorithm [19] is accurate but does not scale (complexity $\mathcal{O}$ (n^2)). On the other hand, PDE based methods, such as Spalding's Poisson equation based method [90] is approximate but scales well(complexity $\mathcal{O}$(n)). Spalding's method first solves the Poisson equation of variable ϕ with a volume source term -1

$$\nabla^2\phi = -1$$

with $\phi = 0$ on wall, or in the integral form,

$$\oint_{\partial\Omega} \nabla\phi \cdot d\vec{A} + \int_{\Omega} 1 dv = 0$$

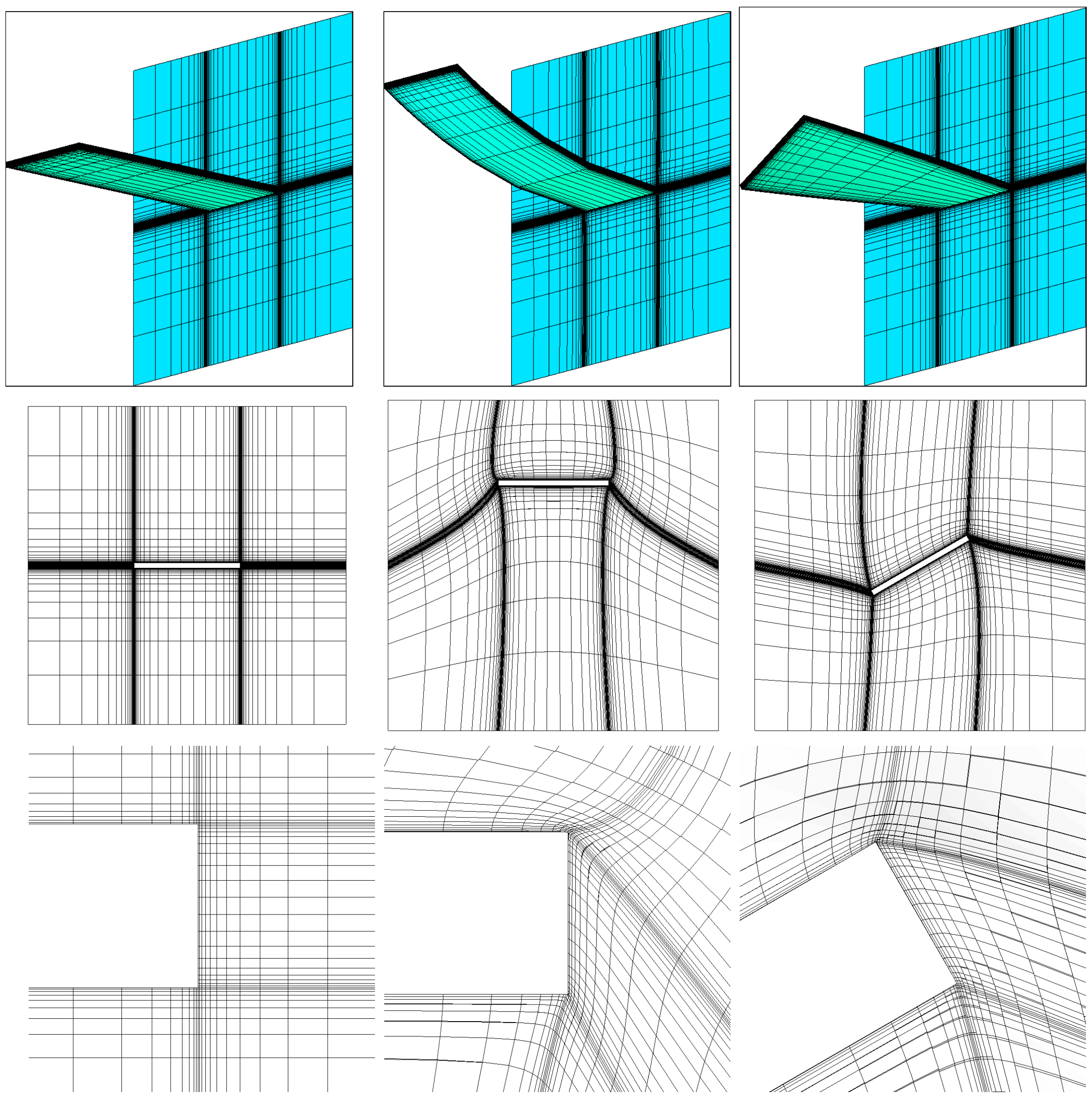

Figure 5.4: Original mesh (left) bending with span-wise parabolic deformation with wingtip displacement 30% of wingspan (middle) and twist of 30° (right).

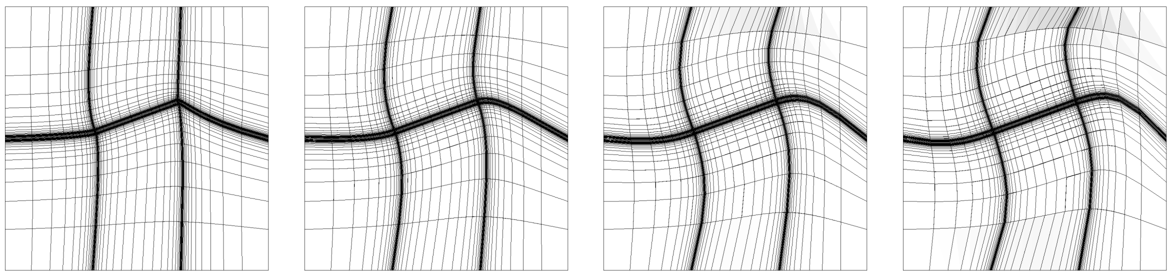

Figure 5.5: From left to right: n=1-4. Cross section at wing tip.

Once ϕ is computed, the distance to wall is approximated as

$$d = \sqrt{\nabla\phi \cdot \nabla\phi + 2\phi} - \sqrt{\nabla\phi \cdot \nabla\phi} \tag{5.2}$$

Eq. (5.2) produces exact wall distance for an infinite planar wall and is inaccurate in the presence of curved walls. To check the error of Spalding's method, the exact ray-tracing algorithm and Spalding's method are compared in fig. 5.6 for the domain around a rectangular body. As shown in fig. 5.6, Spalding's method over predicts the overall wall

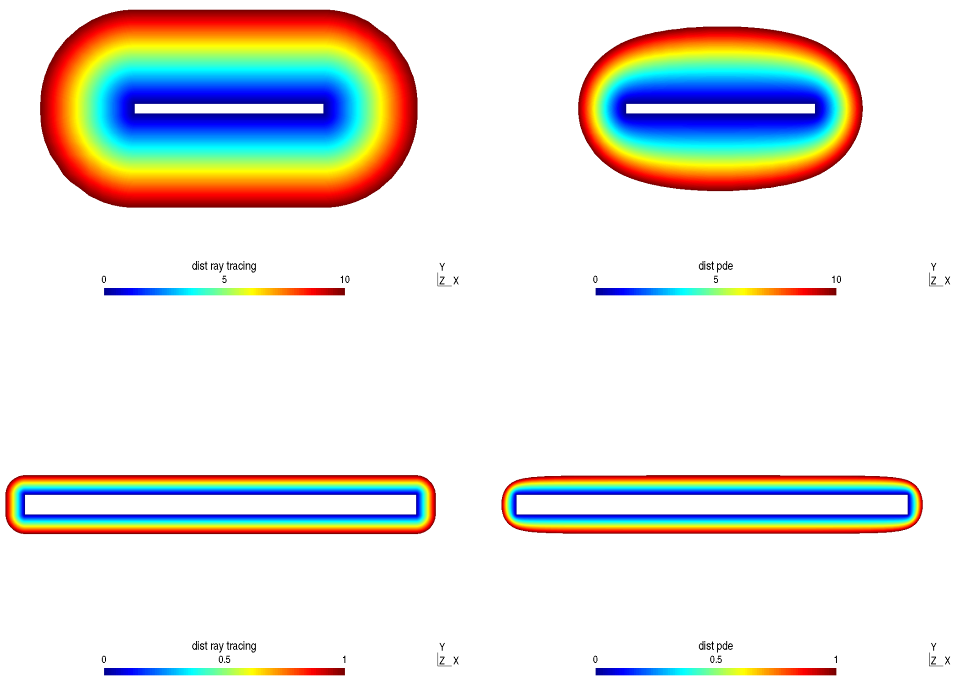

Figure 5.6: Wall distance calculation comparison: ray tracing (left) and Spalding's method (right). Top: wall distance smaller than 10 shown; bottom: wall distance smaller than 1 shown.

distance and the worst accuracy is near the sharp edge of the rectangle. Additionally, the accuracy improves when approaching the wall. The qualitative agreement between the two methods is good enough for the wall distance to be used for scaling Young's modulus for linear elasticity mesh deformation.

5.3.4 Mesh deformation allowing mesh sliding

For certain applications, it is beneficial to allow the mesh to slide along the underlying geometry to better preserve the mesh quality, especially when a large deformation is applied. One such example is a turbine blade with a small tip gap. When the blade tip deforms and results in a relative movement of the blade tip in the circumferential direction relative to the casing surface, it would be helpful to let the mesh on the casing surface slide along the underlying circular arc that defines the casing.

To implement this functionality, only a small change to the boundary condition in solving the linear elasticity is needed. Recall that for design surfaces, a finite known displacement is prescribed as

$$R(u) = 0 \quad \text{and} \quad u = g \quad \text{on} \quad \partial\Omega_{\text{design}} \tag{5.3}$$

and on non-design surfaces,

$$R(u) = 0 \quad \text{and} \quad u = 0 \quad \text{on} \quad \partial\Omega_{\text{non-design}} \tag{5.4}$$

For surfaces allowing sliding, the following BC is applied

$$R(u) = (I - \mathbf{B})R(u) \quad \text{and} \quad u = (I - \mathbf{B})u \quad \text{on} \quad \partial\Omega_{\text{sliding}} \tag{5.5}$$

where matrix $\mathbf{B}$ is the projection matrix used for imposing slip wall boundary condition in the nonlinear flow solver. If the underlying geometry is not planar, then sliding along the tangent direction would cause the nodes to leave the geometry. Therefore, each deformation has to be a small increment to avoid the nodes on the sliding surface deviating too far away from the geometry. And after each incremental mesh deformation, a correction step using Newton iteration needs to be applied to bring the nodes back onto the geometry. This also implies that the analytic definition of the underlying geometry needs to be known.

As a simple example, the sliding capability is tested on the NACA0012 airfoil with a circular outer boundary. The airfoil is rotated by 180° about the leading edge. Without the sliding capability, the deformed mesh would be invalid with cells of negative volume appearing. With the sliding capability applied on the outer circular boundary, the deformed mesh is of good quality, undergoing almost a rigid body motion. The original and the deformed meshes are shown in fig. 5.7 for comparison. For the sliding mesh case, the total rotation is applied as 6 rotations, 30° each. After each small rotation, the surface nodes are brought back to the known outer circle with the volume mesh deformed accordingly using the mesh deformation algorithm without the sliding capability.

The second example to demonstrate the linear elasticity mesh deformation with sliding mesh capability is shown in fig. 5.8 for a three dimensional rotor blade with tip gap. The mesh is taken at a constant axial position near the rear of the rotor blade. The casing surface is a circular arc with known radius. The rotor tip surface undergoes an arbitrary displacement and the volume mesh is perturbed using the linear elasticity mesh deformation algorithm. The result without the sliding capability is shown in the top

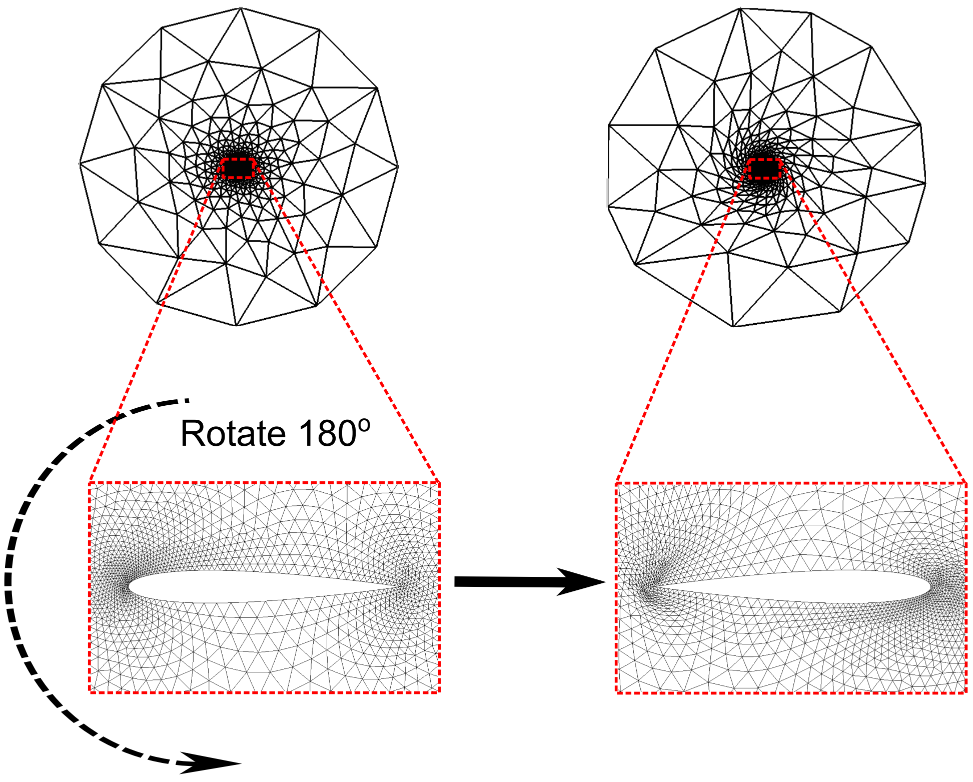

Figure 5.7: NACA0012 airfoil rotates 180° with sliding free stream boundary. Left: original, right: after rotating 180°.

panel of fig. 5.8. Since the mesh is fixed along the casing, the mesh quality severely deteriorated after deformation. The deformed mesh with the sliding capability is shown at the bottom of fig. 5.8. It is obvious that the mesh quality is much better than the one without sliding capability.

5.4 Summary

Two mesh deformation techniques, spring-based and linear-elasticity based, are explained in detail, in terms of both the mathematical formulation and the implementation. The spring-based deformation is easier to solve but only useful for isotropic meshes undergoing small deformation. It's therefore only useful for projecting the volume sensitivity to the design surface using its 'backward' mode. Linear-elasticity based algorithm on the other hand, is much more reliable when applied to deform an anisotropic mesh used for viscous flow calculations undergoing large displacement. To facilitate the efficient wall distance calculation, important for specifying the material property used by the linear elasticity solver, Spalding's Poisson's equation based approximate wall distance calculation algorithm is implemented and compared with the exact but much less scalable ray-tracing algorithm. Further, the sliding mesh capability is introduced and applied to test cases to demonstrate the enhanced robustness of the resulting linear-elasticity mesh deformation algorithm.

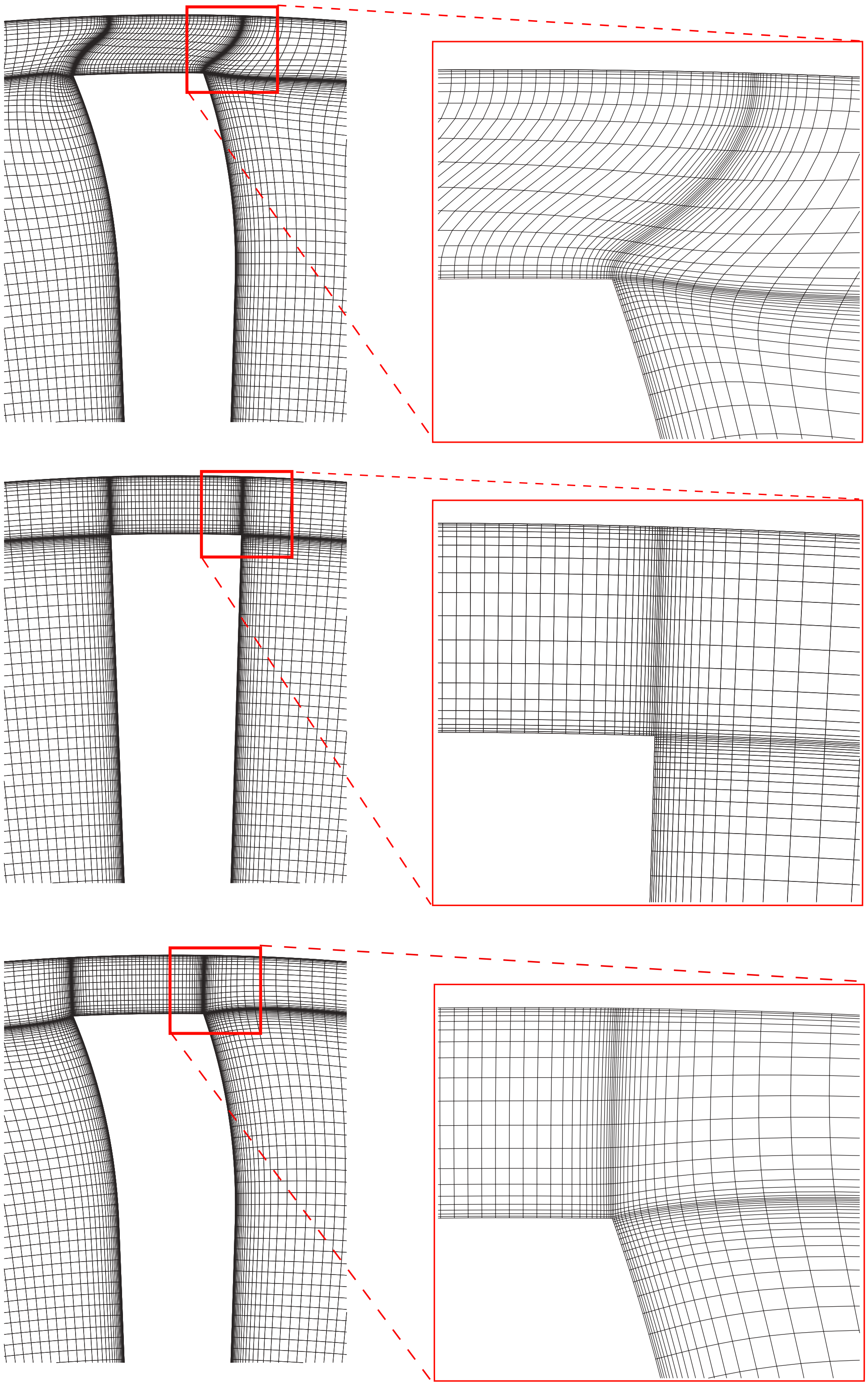
Figure 5.8: The original mesh (middle), the deformed mesh due to tip displacement with fixed casing mesh fixed (top) and the deformed mesh due to tip displacement with casing mesh allowed to slide along the casing (bottom).

Chapter 6

CAD/NURBS-based parametrisation with geometric constraints

6.1 Introduction

In this chapter, the author aims to tackle another major challenge in industrial CFD shape optimisation: parametrisation [83, 27]. For routine industrial applications, an approach is desirable that (1) supports a relative rich design space, (2) permits the imposition of geometric constraints and (3) compatible with the 'master' CAD file that is created for designer and where various geometries are derived for aerodynamic, structural and thermal analysis. The third requirement is important for a smooth integration of various analysis tools into a unified environment. It requires that the design optimisation operate on the same parametrised CAD model that is simplified from the 'master' CAD design.

A number of approaches have been proposed in the past: node-based [39, 43], Radial Basis Function (RBF) [38], shape functions on lattices [75], free-form deformation (FFD) [84] and CAD-based parametrisation [78]. Note-based parametrisation operates on the CFD mesh rather than the underlying geometry. The coordinates of the surface mesh are used as design variables. The major issue is the surface design sensitivity for flows with high Reynolds number is usually very noisy and substantial amount of smoothing is needed to regularise the resulting perturbation mesh [39, 37]. Node-based parametrisation can be compatible with some kind of CAD format, such as the standard STereoLithography (STL) format. Lattice-based shape functions [75, 76] and free-form deformations [84], both using the morphing technique, require auxiliary shape grids, whose perturbation is then interpolated to the CFD mesh nodes smoothly. The advantage is that it can be used to morph either the underlying geometry or the mesh, or both. One way of morphing is by using Radial Basis Function (RBF) [8]. In fact, commercial tools are already available to perform RBF based morphing with relative ease and two application examples can be found in [9] and [10]. Such RBF based morphing can used to morph either the surface mesh and the underlying geometry. For shortcoming of lattice-based method, including RBF morphing is that it takes a top-down approach, i.e., using the coarse bounding box to smoothly perturb the surface and volume meshes, and the consequence is a reduced design space, which sometimes could reversely restrict the design improvement.

Modern CAD systems are usually very sophisticated and many features and implementations are vendor-dependant and proprietary. Recognising that the CAD-based parametrisation and the volume mesh deformation due to surface mesh perturbation could be dealt with separately, the main task is then first the binding of the surface mesh with the underlying parametrised CAD model, and then a mesh deformation algorithm that propagates the surface displacement into the volume. This procedure is less straightforward for non body-fitted mesh than for body-fitted mesh. For example, a Cartesian mesh with cut-cell is used in combination with a CAD-based parametrisation is used in [68]. The advantage is that Cartesian meshing with embedded boundary representation can be much more easily automated than body fitted meshes. Using body-fitted meshing tools, either structured or unstructured, would allows the design sensitivity chain to be assembled more easily.

Another CAD-based approach is to directly employ the parametrisation defined in a CAD system [68, 79]. These parametrisations are typically built to generate families of parts and in general will need to be substantially modified to produce a suitable design space to capture the optimum, hence still requiring significant user input. The manual set-up restricts the design space and an important mode may be not represented. On the other hand the provided parametrisations often have the relevant geometric constraints built in. The limitation with this approach. is that firstly there is no universal standard for these CAD parametrisations, they are defined using standards specific to a CAD system that are in general not transferable. Secondly, current CAD systems do not offer derivatives of surface displacements with respect to the design parameters which are needed in the chain rule to compute the sensitivities of the cost function with respect to design variables. The only available option is to apply finite differences to the CAD system, resulting in significant robustness issues as finite-size displacements could lead to topological changes in the surface description and changes in patch numbering. Robinson et al. [79] also report significant runtimes for a three dimensional air duct case with 174 design variables. If one uses an efficient simultaneous time-stepping method, the so-called one shot method [36, 42], this penalty would increase dramatically as these methods use a large number of small design steps.

CAD-free and CAD-based approaches have been thoroughly discussed and compared in [27]. In that work, the CAD-free method refers to one that uses the control points of the reconstructed B-spline surfaces as design variables while the CAD-based method refers to one that directly uses the parameters from the CAD software as design variables. The CAD-free approach is similar to the NURBS-based parametrisation proposed in this work, except that the water-tightness of the neighbouring patches is maintained by letting the control points on one edge of a B-spline surface follow the control points on the edge of the neighbouring B-spline surface. This guarantees G_0 continuity, provides the B-spline surfaces of the patches are conformable along the shared edge. The CAD-based approach is achieved through the use of the Computational Analysis Programming Interface (CAPRI) [33]. CAPRI is a tool that bridges the CAD systems and CAE such as computational fluid dynamics analysis tool by hiding the vendor dependant CAD parameters from the users using the API that reads the topology and geometry (solid BRep) from vendor dependant geometry file and internally reconstruct the solid models and at the same time, provides Solid Boolean operation for the reconstructed solid models. The parametrisation available from CAPRI is then limited by its capability to read, load and operate the various features from each individual CAD software. In [27], the CAD-based parametrisation is limited to lofted bodies. Another features, such as wireframes can be read but cannot be modified or used as design variables. This limitation is not a problem for most applications. Both methods are applied to an aircraft with fuselage and wing for geometry control successfully. The fuselage and wing intersection is not dealt implicitly via the CAD-based parametrisation, but rather, is enforced at the surface mesh point level using a 'negative sensor'.

We propose an alternative approach which avoids the robustness and runtime issues of directly using the CAD parametrisation. The approach, first presented in two dimensional cases [99], utilises the fact that the relevant output of the CAD system to the analysis and manufacturing tools is the boundary representation (BRep) given in the STEP standard as a set of NURBS surface patches [72]. The approach uses the displacements of the control points of the NURBS patches as design variables, the updated set of control point positions fully define the CAD shape of the optimal design at convergence. This approach produces a modified CAD-description of the surface at output. It is fully automatic as it does not require any additional user input and it works with a generic, vendor-independent parametrisation. Compared with RBF morphing, which could also be CAD-based, NURBS-based approach takes a bottom-up approach and the smoothing property is guaranteed by definition when modelling the original NURBS surfaces while a large design space could still be retained. For example, typical RBF-morphing uses tens of design variables while NURBS-based approach allows thousands of design variables.

Another advantage of using NURBS parametrisation is that after each design step, the control points of the NURBS patches can be readily exported in the vendor-independent STEP file format, which then can be read in by most widely used CAD software.

A number of authors have used the NURBS control points as design variables [100, 58], but considered only geometries with a single patch or only deformation inside a patch. The main challenge to generalise this approach is to maintain the required level of continuity of tangency and curvature between adjacent NURBS patches when a control point on or near a patch interface is displaced. The proposed method introduces constraints for geometric continuity across NURBS patch interfaces and is hence termed "NURBS-based parametrisation with complex constraints", or NsPCC. This extension of functionality greatly enhances the applicability of the CAD/NURBS-based parametrisation method for shape optimisation of complex geometries.

The remainder of the chapter is structured as follows: sec. 6.2 discusses the formulation of the CAD/NURBS-based parametrisation. Its implementation and how to satisfy and maintain constraints of geometric continuity is discussed in secs. 6.3 and 6.5.

6.2 CAD/NURBS-based parametrisation

The CAD/NURBS-based parametrisations evaluate the deformation of the surface resulting from a perturbation in design variables. The surface deformation then needs to be interpolated onto the surface nodes of the CFD mesh and propagated smoothly into the volume to maintain volume mesh validity and quality using a mesh deformation algorithm. The mesh deformation method used is based on linear elasticity, which has been discussed in chapter. 5.

The design sensitivity can be written as

$$\frac{dJ}{d\alpha} = \frac{dJ}{dX_s}\frac{dX_s}{d\alpha}$$

where X_s is the surface node coordinate and α is a CAD parameter.

Robinson et al. [79] use finite differences to compute the shape derivative $\frac{dX_s}{d\alpha}$, where α are user- and system-defined CAD parameters. The main issues with this approach have been discussed in the introduction. In contrast to this, we propose to base the CAD parametrisation on the boundary representation (BRep) which uses a collection of NURBS patches as given in the standardised STEP format. To modify the shape, each control point of a NURBS patch is allowed to move in all directions, hence each representing 3 degrees of freedom. A simple example of a NURBS patch with its associated control points is shown in fig. 6.1 which also illustrates how the perturbation of a control point changes the shape of the NURBS patch. Allowing every control point to move in all

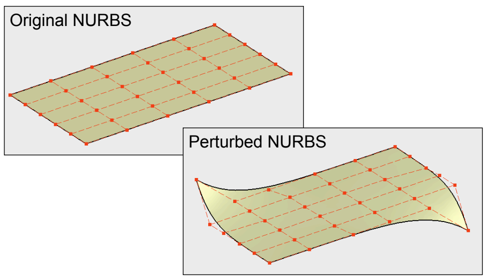

Figure 6.1: A NURBS patch with the net of original (upper left) and perturbed (lower right) control points

directions represents the richest design space the CAD model is able to express. Adaptive control point enrichment or order elevation of NURBS patches can be implemented to refine the design in areas of high sensitivity.

Imposing geometric build-space constraints can either be done at the level of the control points exploiting the convex-hull property of splines [72], or alternatively at the level of the surface mesh points which achieves tighter bounds at higher computational cost. In addition to requiring smooth shapes, industrial applications often seek to limit the local curvature, e.g. at trailing edges of aero-engine turbine blades. Unfortunately there is no known method to derive analytic bounds for the maximal curvature in a NURBS patch, but control of maximal curvature can be performed at the surface mesh level, which is the appropriate level of detail resolved by the CFD simulation. These build constraints can be implemented in the same framework as the patch continuity constraints discussed in Sec. 6.3.

For an independently moveable control point with coordinates P, the gradient of the cost function can then be written as

$$\frac{dJ}{dP} = \frac{dJ}{dX_s}\frac{\partial X_s}{\partial P} \tag{6.1}$$

The shape derivatives $\dfrac{\partial X_s}{\partial P}$ can be calculated analytically, their definition is independent of the CAD system. Specifically, a NURBS is defined [72] as

$$X_s(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,j}(u,v)P_{i,j}, \tag{6.2}$$

where $P_{i,j}$ are the control points, u and v are the parametric variables of the surface mesh point. The rational basis functions $B_{i,j}(u,v)$ are defined as

$$B_{i,j}(u,v) = \frac{N_{i,p}(u)N_{j,q}(v)w_{i,j}}{\sum_{k=0}^{n}\sum_{l=0}^{m} N_{k,p}(u)N_{l,q}(v)w_{k,l}} \tag{6.3}$$

where $N_{i,p}(u)$ and $N_{j,q}(v)$ are the p-th and q-th degree basis functions defined on the parametric space u, v. Note that the shape derivative matrix $\partial X_s/\partial P$ is equal to the basis functions $B_{i,j}$, and thus the gradient of the cost function w.r.t. the control points is simply

$$\frac{dJ}{dP} = \frac{dJ}{dX_s} B \tag{6.4}$$

where B is a matrix with elements $B_{i,j}$.

In addition to the shape derivative we also need to compute actual displacements following a shape update by the optimiser. We have hence chosen to implement a basic CAD-kernel in FORTRAN that evaluates NURBS surfaces, this code has then been differentiated using the AD-Tool Tapenade [34] in forward mode to provide the necessary derivatives.

6.3 Imposing the continuity constraint

The finite displacement of a control point P on or near a patch interface typically results in violation of the continuity constraints, e.g. control points on an interface to a fixed/non-moveable patch must not move at all to maintain G_0 continuity (no gaps). Requiring in this case G_1 (tangent) and G_2 (curvature) continuity will additionally lock the second and third rows of control points. Similarly, control point displacements on moveable patch interfaces imply constraints on the neighbouring rows of control points along the interfaces.

In general, G_0 continuity requires that the adjacent patches share a common edge. G_1 continuity requires firstly that G_0 is satisfied, and secondly, that the adjacent NURBS patches share the same tangent plane for any point along the common edge. However, this does not necessarily require that the adjacent patches have the same number and distribution of control points or degrees of basis function. In our approach the constraints are evaluated at a number of test points that are distributed along a coincident parametric edge or the intersection line of both patches. Fig. 6.2 shows two NURBS patches sharing one common edge. Note that the number of control points along the common edge could be different for the left and right patches, but the test points are always deployed in pairs, with one on each NURBS patch. The required number of test points is discussed in Sec. 6.4. Continuity constraints for each pair of test points then express that location, tangent plane (for G_1) and curvature (for G_2) are identical on all patches containing this pair of test points. E.g., for G_1 the position vectors and the normal vectors of the tangent planes need to be identical at the test point when evaluated in either patch. A kink between patches, e.g. as a manufacturing constraint for mould deforming, can be specified by requiring normals to differ by the imposed deforming angle. For G_0 and

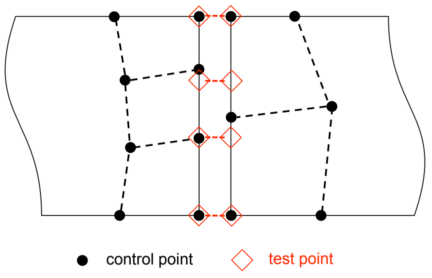

Figure 6.2: Test points along the NURBS boundaries and the relevant control points on each patch

G_1 continuity along an edge shared by two NURBS patches, the following constraint functions need to remain zero for the test points evaluated in each of the patches:

$$G_0 = X_{s,L} - X_{s,R} \tag{6.5}$$

$$G_1 = \vec{n}_L \times \vec{n}_R \tag{6.6}$$

where the positional and tangent continuity constraint functions are denoted as G_0 and G_1 respectively and $\vec{n}$ is the unit normal of the tangent plane at the test point defined as

$$\vec{n} = \frac{\left(\dfrac{\partial X_s}{\partial u} \times \dfrac{\partial X_s}{\partial v}\right)}{\left\|\dfrac{\partial X_s}{\partial u} \times \dfrac{\partial X_s}{\partial v}\right\|}, \tag{6.7}$$

where u and v are the surface parametric variables as introduced in Eq. (6.2) and the subscripts 'L' or 'R' mean that the term is evaluated for the test point either on the left or the right patch, with the notation suitably extended for corner vertices. Eqs. (6.5) and (6.6) are for positional and tangential continuity, respectively. To maintain the geometric continuity of the initial geometry, each constraint function is required to remain zero after each design update. To evaluate the change in constraints, each constraint function is linearised as

$$G^{n+1} \approx G^n + \sum_{i=1}^{N} \frac{\partial G}{\partial P_i} \delta P_i \tag{6.8}$$

where P_i denotes the displacement of the x, y and z components of the i-th control point, with a total of N control point. Eq. (6.8) uses the superscripts $n, n+1$ to express the numerical evaluation of the constraint at the constraint iterations $n, n+1$.

Assembling (6.8) for each test point displacement, and requiring $G^n - G^{n+1} = 0$, the following linear equation system is obtained:

$$\mathbf{C}\delta P = 0 \tag{6.9}$$

where

$$\mathbf{C} = \left[\frac{\partial G_j^i}{\partial P_k}\right] = \begin{bmatrix} \frac{\partial G_0^1}{\partial P_1} & \frac{\partial G_0^1}{\partial P_2} & \cdots & \frac{\partial G_0^1}{\partial P_N} \\ \frac{\partial G_1^1}{\partial P_1} & \frac{\partial G_1^1}{\partial P_2} & \cdots & \frac{\partial G_1^1}{\partial P_N} \\ \frac{\partial G_0^2}{\partial P_1} & \frac{\partial G_0^2}{\partial P_2} & \cdots & \frac{\partial G_0^2}{\partial P_N} \\ \frac{\partial G_1^2}{\partial P_1} & \frac{\partial G_1^2}{\partial P_2} & \cdots & \frac{\partial G_1^2}{\partial P_N} \\ \cdots & \cdots & \ddots & \cdots \\ \frac{\partial G_1^M}{\partial P_1} & \frac{\partial G_1^M}{\partial P_2} & \cdots & \frac{\partial G_1^M}{\partial P_N} \end{bmatrix} \quad \text{and} \quad \delta P = \begin{bmatrix} \delta P_1 \\ \delta P_2 \\ \vdots \\ \delta P_N \end{bmatrix}.$$

if G_1 is imposed. In this notation for $\mathbf{G}$ the super-script 'i' refers to the 'i'-th of M pairs of test points and the sub-script 'j' denotes the $\mathbf{G}_\mathrm{j}$ constraint function ($j = 0$ for positional constraint, and $j = 1$ for tangent constraint). The subscript k to P_k refers to the displacement of the k-th of N NURBS control points.

The linearised constraint matrix $\mathbf{C}$ has $2 \times 3 \times M$ rows corresponding to a total of M pairs of test points with G_1 constraint, and $3 \times N$ columns corresponding to a total of N control points. To satisfy the continuity constraints in a linearised sense, the perturbations of the control points δP have to lie within the null space of the constraint matrix $\mathbf{C}$. We compute the null-space of $\mathbf{C}$ with a singular value decomposition (SVD):

$$\mathbf{C} = \mathbf{U\Sigma V}^T, \tag{6.10}$$

where $\mathbf{U}$ is an $m \times m$ unitary matrix, $\mathbf{\Sigma}$ is an $m \times n$ diagonal matrix with non-negative real numbers on the diagonal, and the $n \times n$ unitary matrix $\mathbf{V}^T$ denotes the transpose of $\mathbf{V}$. The rank of the matrix, r, is the number of the non-zero diagonal entries in $\mathbf{\Sigma}$. The last $(m - r)$ columns of the matrix $\mathbf{V}$ span the null space of $\mathbf{C}$, denoted by $\mathrm{Ker}(\mathbf{C})$ As rounding error may lead to small but non-zero singular values, it is necessary to find a demarcation point below which the singular values are regarded as zero. The SVD orders the diagonal elements of $\mathbf{\Sigma}$ by the magnitude of the singular value σ_i, ranking highest those modes that are most important to approximate $\mathbf{C}$. This property enables us to truncate the null space at an appropriate level as discussed in Sec. 6.4. The allowable control point perturbations hence become

$$\delta P = \sum_{k=1}^{m-r} v_{k+r} \delta\alpha_k = \mathrm{Ker}(\mathbf{C})\delta\alpha \tag{6.11}$$

where $\delta\alpha_k, k = 1, ..., m-r$ are the perturbations in design parameters. The gradient formulated in equation (6.1) can then be further modified to be

$$\frac{dJ}{d\alpha} = \frac{dJ}{dX_s}\frac{\partial X_s}{\partial P}\frac{\partial P}{\partial \alpha} = \frac{dJ}{dX_s}\frac{\partial X_s}{\partial P}\mathrm{Ker}(\mathbf{C}) \tag{6.12}$$

6.4 Required number of test points

As NURBS are based on polynomials over specific knot-vector intervals, testing the values of two NURBS at a sufficient number of distinct points will allow to test exactly whether they match. It is however not straightforward to pre-determine the number of test points that are needed [72]. We propose to use the number of non-zero singular values of the linearised constraint matrix $\mathbf{C}$ to determine the number of test points needed along each edge. The SVD will filter out redundant constraints arising from excessive test points: once there are sufficient test points to determine the null space, the number of singular values will no longer increase when a further control point is added.

Singular values are considered zero if they fall below a certain threshold. G_0 constraints lead to a very cleanly conditioned SVD with singular values either clearly non-zero or of the order of machine precision. In the case of G_1 or higher, the distribution of singular values is more gradual with a long tail of small singular values gradually tailing off in size. We select a cut-off for the singular value at 10^{-7}, which may reduce the accuracy of SVD, but increases the design space. As G_1 and higher constraints are non-linear, the design step in the null space will not satisfy the constraints exactly and a recovery step is needed, as shown in Sec. 6.5. A very exact representation of the null space is hence not required.

This methodology is illustrated using the S-Bend optimisation case (for details of this case, refer to chapter 7). When only G_0 is imposed, the number of non-zero eigenvalues remains unchanged if there are more than 12 test points (figure not shown here). Fig 6.3 shows the behaviour for G_1, the number non-zero singular values ($\lambda > 10^{-7}$) ceases to increase for more than 30 pairs of test points along each edge.

6.5 G_1 continuity recovery

The geometric continuity constraint developed in Sec 6.3 restricts the perturbation to be tangent to the linearised constraint functions. The G_1 constraint function is non-linear, hence each linearised tangent step will slightly violate the constraint.

To recover the constraint after each tangent step, additional normal steps in the range space of $\mathbf{C}$ are required to recover the non-linear continuity constraints G_1 and G_2. The

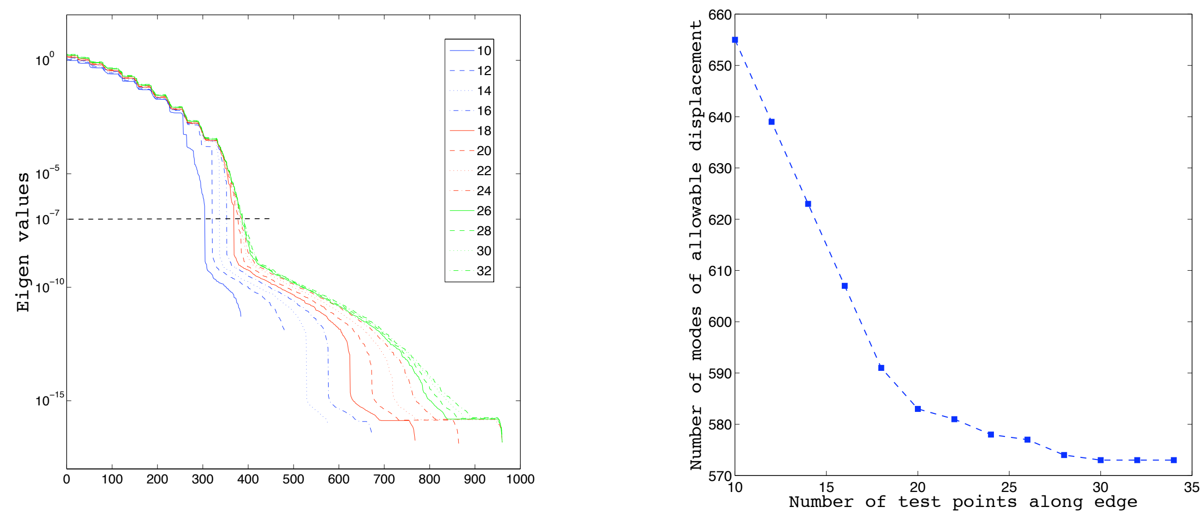

Figure 6.3: Left: singular values for the linearised G$_1$ constraint matrix using different number of test points along the edges; right: the number of allowable displacement modes according to eigenvalues-value analysis with cut-off value of $\lambda = 10^{-7}$

following development shows recovery of G$_1$, and the extension to G$_2$ is straightforward. Defining the deviation of the G$_1$ constraint function from the target by δG_1,

$$\delta G_1 = \sum_{i=1}^{M} ||G_1^i||, \tag{6.13}$$

where M is the number of pairs of test points, $\mathbf{G}_1^i$ is the constraint function evaluated at the i-th pair of test points, and denoting the linearised G_1 constraint function by $\mathbf{C}_1$ as in Eq. (6.9), we can compute the recovery step of the control points $\delta P_\perp$ as

$$\mathbf{C}_1 \delta P_\perp + \delta G_1 = 0 \tag{6.14}$$

In addition to minimising G_1, we also require G$_0$ to be still strictly satisfied, and thus δP should satisfy during each recovery step

$$\delta P_\perp = \mathrm{Ker}(\mathbf{C}_0)\delta\alpha_\perp \tag{6.15}$$

where $\mathbf{C}_0 = \dfrac{\partial G_0}{\partial P}$ is calculated and stored at the beginning of the whole optimisation, as $\mathbf{C}_0$ is independent of control point displacements. Algorithmically, we first solve for $\delta\alpha_\perp$ which satisfies

$$\mathbf{C}_1 \mathrm{Ker}(\mathbf{C}_0)\delta\alpha_\perp + \delta G_1 = 0 \tag{6.16}$$

using SVD for pseudo inversion of the system matrix $\mathbf{C}_1\mathrm{Ker}(\mathbf{C}_0)$. As a second step we compute the control point corrections $\delta P_\perp$ using Eq. (6.15), which guarantees that they

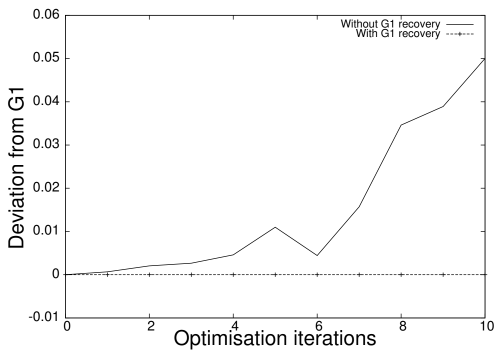

Figure 6.4: Deviation from exact G_1 continuity with v.s. without G_1 recovery steps

lie strictly in the null space of $\mathbf{C}_0$ and thus G_0 continuity is exactly satisfied in the G_1 recovery step.

In the test case presented in this work, only a few Newton steps are needed to bring the G_1 continuity below the chosen threshold, 10^{-5}.

The effect of applying G_1 continuity recovery steps is shown in fig. 6.4. Without G_1 recovery steps, the optimised shape gradually deviates from the initially exact G_1 continuity, and introduces an average error of around 1.1° between the tangent plane normals after only 10 design steps. With the G_1 recovery steps applied after each cost-function-based perturbation, the deviation remains below 10^{-5}.

Chapter 7

CAD-based optimisation of a duct

7.1 Introduction

The NsPCC method is applied to the 3D segment of an S-Bend air duct shown in fig. 7.1. This testcase has been provided by Volkswagen AG to the FlowHead research project. The goal is to deform the central S-section such as to minimise the mass-averaged total pressure loss defined as

$$J = \frac{\int_{inlet} P_{tot}\vec{u}\cdot\vec{n}dS + \int_{outlet} P_{tot}\vec{u}\cdot\vec{n}dS}{\int_{inlet} \vec{u}\cdot\vec{n}dS}.$$

An inlet velocity of 0.1 m/s, a zero back pressure and no-slip walls are used as boundary conditions. The Reynolds number using the inlet height as the reference length is Re=300.

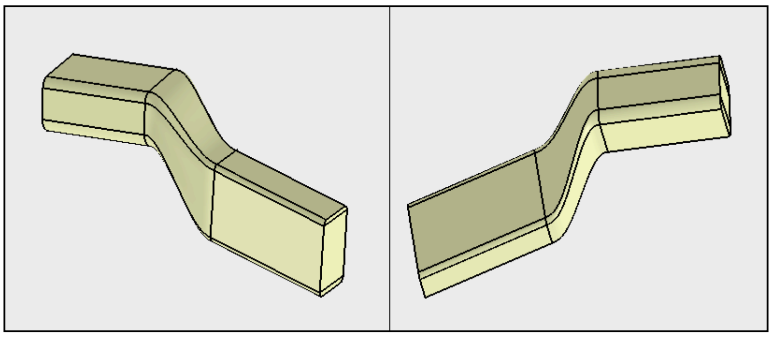

Figure 7.1: Initial geometry consisting of 30 NURBS patches viewed in CATIA V5

The emphasis of this chapter is on the parametrisation method, and the flow and adjoint solver can be seen as a black box. In this work, the in-house incompressible flow and adjoint solver GPDE is used for computing the flow and the adjoint field. For details of the development of the incompressible adjoint code, refer to [45].

7.2 Parametrisation and geometric constraints

The initial geometry is shown in fig. 7.1, the duct bends upwards and sideways, so the optimisation result is also expected to be asymmetric w.r.t. the vertical plane. Only the patches in the cranked S-section of the duct are allowed to move in order to represent build constraints for the inlet and outlet sections. The fixed patches at inlet and outlet are G_0 continuous, G_1 continuity is initially satisfied and imposed across interfaces of the moveable patches.

The deformable S-section consists of 8 NURBS patches in total. Four of those patches each have 96 (16×6) control points, the other four, each have 64 (16×4) control points, resulting in a total of 640 control points for the S-section, equivalent to 1920 degrees of freedom (DoF) as each control point is allowed to move in the x, y and z directions.10^{-5} At common edges between the 8 moveable and the fixed NURBS patches upstream and downstream of the S-section, the first 3 layers of control points of the moveable NURBS patches are fixed so that the entry and exit throats both have G_2 continuity and meet with zero curvature. In summary, a total of 400 control points are allowed to move in three directions, equivalent to 1200 DoF.

To impose continuity constraints a total of 240 pairs of test points are distributed along the 8 deformable joint edges, 30 on each edge, resulting in 1440 constraint equations. The number 30 is determined as described in Sec 6.3, and this choice leads to a null space with around 570 allowable modes, which may vary slightly over optimisation iterations due to the non-linearity of G_1 constraint function. The corresponding computational surface mesh of the S-section has a total of 3840 boundary nodes. The steepest descent method is used as an optimiser, and the step size is determined using a simple line search bounded by a maximum allowable step size, which requires that the maximum surface node perturbation does not exceed the minimum spacing of the surface mesh. This ensures that the surface mesh perturbation after each design iteration is bounded and the volume mesh deformation algorithm based on linear elasticity remains stable.

7.3 Optimisation results

The convergence history of the cost function is shown in fig. 7.2. As is shown in fig. 6.4, and also confirmed by manually checking the optimised geometry in the CAD program CATIA V5, the final shape remains G_1 continuous across patch interfaces where specified. The cost function drops about 25.5% after 60 design optimisation iterations; during each iteration 5 normal steps are used to recover G_1 continuity.

The initial shape and the optimised shape after 60 iterations are shown in fig. 7.3. The optimised shape shows a distinct bulge at the top and bottom of the S-section, as

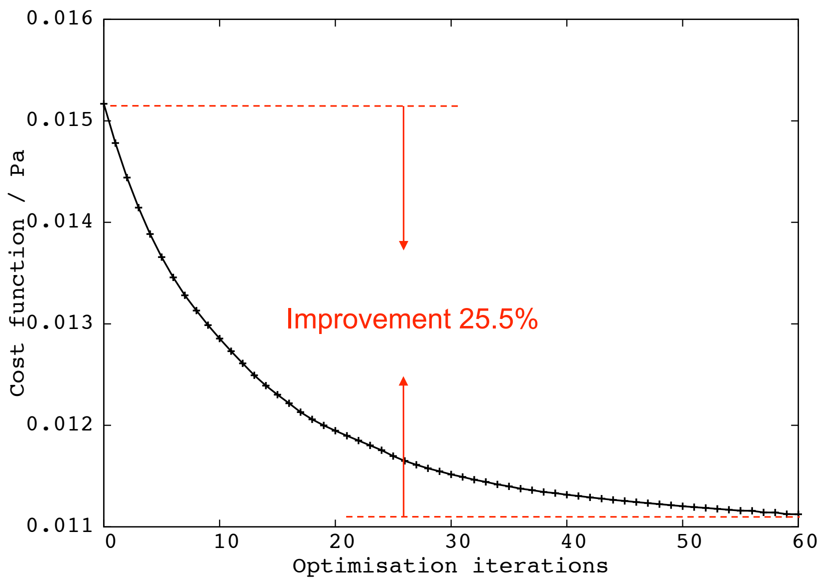

Figure 7.2: Cost function convergence history

well as along the four edges, while hollowing inwards from both sides. Also shown in fig. 7.3 is the surface sensitivity of the initial and the optimised shapes. The sensitivity of the cost function w.r.t. the normal displacement of the surface is reduced to almost zero everywhere expect at the inlet and outlet throats where the surface is not allowed to move.

The resulting shape changes are not intuitive, hence to better understand the optimisation result, we compare the flow fields. Fig. 7.4 and fig. 7.5 show the contour plots of velocity magnitude and the streamlines for the initial and optimised shapes. It can be seen that the flow field of the initial shape has very strong secondary flows, the cross-sectional cuts exhibit strong non-uniformity in the flow speed. This phenomenon of secondary flow in bent ducts is well known as Dean vortices [6]. There is a large area of separated flow at the bottom of the outlet section in the nearside corner, which will add to the total pressure loss.

The flow field of the optimised shape on the right of fig. 7.4 and fig. 7.5 exhibits much weaker secondary flows. The streamlines and velocity magnitude contours show that the separation is significantly reduced. The main difference compared to the original flow lies in the suppressed Dean vortices.

It is apparent that this suppression is achieved by the hollowing in of both sides of the S-section, which resembles strakes to suppress the formation of vortices, such as widely used on aeroplanes, ship hulls and pipelines. The strake-like shape is more apparent in fig. 7.6, which shows a cross cut in the S-section viewed from the outlet into the S-Bend.

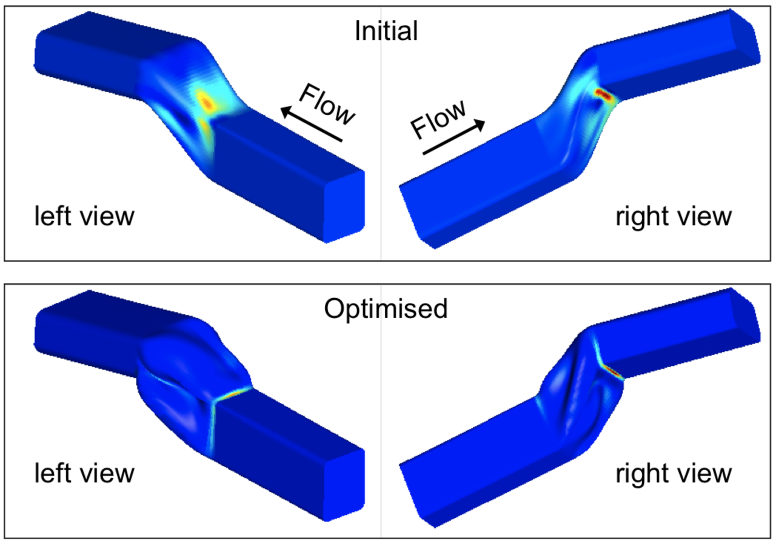

Figure 7.3: Comparison between the initial (upper) and optimised (lower) shapes and the surface sensitivity map for both shapes.

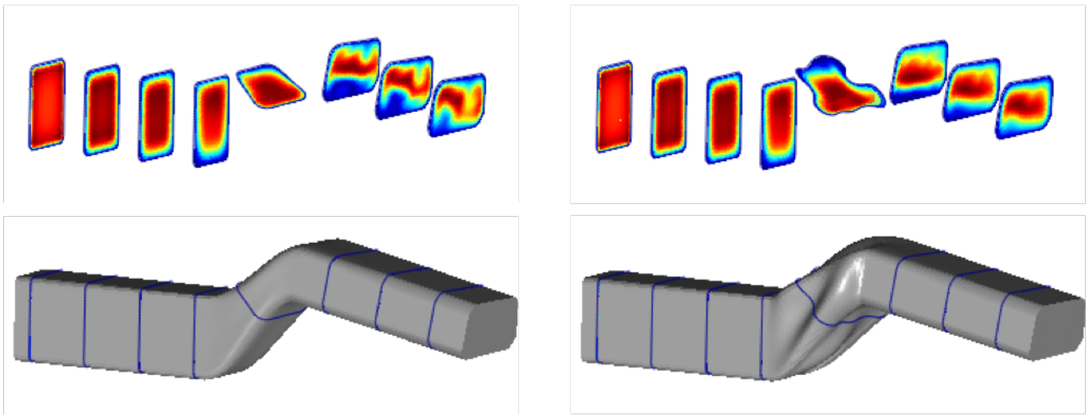

Figure 7.4: Contour plots of velocity magnitude for the initial (left) and optimised (right) ducts

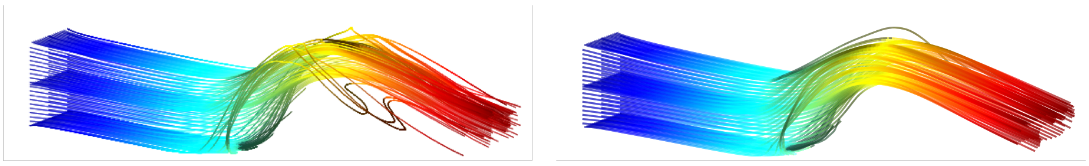

Figure 7.5: Streamlines plots for the initial (left) and optimised (right) ducts. Colour along the streamlines is for illustrative purpose, not related to any flow variable.

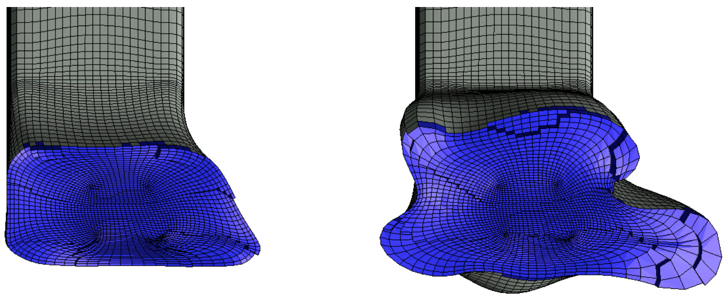
Figure 7.6: Cross section view of the S-section for the initial and optimised S-Bend

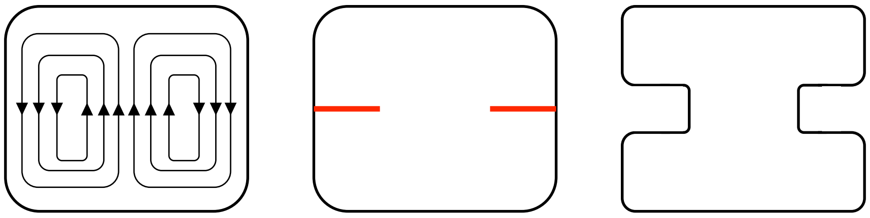
Figure 7.7: Schematics illustrating the formation of strake-like shape. Left: original cross section with a pair of vortices whirling in counter directions; middle: two strakes can be added to suppress the Dean vortices to minimise energy loss; right: alternatively, the sides can hollow inward to mimic the effect of stakes

Ideally, a pair of stakes along the side of the S-section (fig. 7.7, middle) are to be formed to suppress the Dean vortices, as is shown on the left of fig. 7.7. However, a shape optimisation methodology as NsPCC does not allow a change of topology, the optimised shape (fig. 7.7, right) clearly shows the tendency to form two strakes on both sides to suppress the secondary flow due to Dean vortices.

The whole optimisation process takes a total of 60 optimisation steps on a mesh with 250,000 hexahedra with the dominant cost arising from the primal and the adjoint solves. The main steps of the optimisation loop are (a) the primal and adjoint computation, (b) the SVD at each optimisation step to compute the null space of the continuity constraint and thus to find the shape displacement modes, as well as (c) the G_1 recovery steps and (d) the shape perturbation step. The costs of each of these steps are shown in table 7.1. It can be seen that the steps related to the CAD-based parametrisation (b)-(d), i.e., SVD null space, G_1 recovery steps and the shape perturbation, have a negligible computational cost compared to the primal and adjoint computations.

Table 7.1: Computational cost breakdown for different chains of the optimisation loop

	percentage over total time
primal (Res=1E-5)	17.28 %
adjoint (Res=1E-5)	78.67 %
SVD null space	2.95 %
G_1 recovery steps	1.01 %
shape perturbation	0.08 %

7.4 Summary

The novel NsPCC shape optimisation methodology has been presented that allows to include the modification of a CAD-description inside the design loop. The method uses the NURBS control points of the generic boundary representation (BRep) as design variables and writes the updated set of NURBS control points to a STEP file. The parametrisation is hence fully automatic, not requiring the user to set up a parametrisation in a proprietary CAD system. Seamless connection from and to CAD software packages is done through the standardised STEP file format.

Geometric continuity at interfaces between NURBS patches is imposed as constraints throughout the optimisation process. The constraints are numerically evaluated at a set of test points, and the number of required test points is determined by evaluating the change in rank of the constraint matrix using SVD. A well-conditioned orthogonal basis for the design space arises from this SVD, allowing to neglect minor modes with little effect on the constraints by appropriately selecting the threshold for singular values considered to be zero. Constraints are maintained linearly by requiring the control point displacements to remain in the null space of the constraint matrix and by perpendicular recovery steps in the range of the constraint matrix to correct for the non-linear behaviour of G_1 and higher constraints.

The effectiveness of the method is demonstrated by applying it to the 3D segment of a Volkswagen air duct, achieving a performance improvement of 25.5%. The strake-like shape on the sides of the optimised duct very effectively suppresses the formation of Dean vortices and flow separation, thus effectively reduces the energy loss. The resulting shape shows very strong and detailed deformations with guaranteed smoothness.

The proposed method hence satisfies the key requirements on design parametrisations for industrial application: a) the parametrisation is fully automatic in an open, vendor-neutral form, able to be coupled interchangeably to a range of CAD systems such as NX and CATIA; b) the design space is as rich as can possibly be expressed in the given BRep and is by construction smooth: the regularity inside patches is given by the order of the patch basis functions, regularity across patch interfaces is controlled by user-imposed geometric continuity constraints; c) permits imposition of geometric constraints such as geometric continuity, but also build-space, manufacturing or maximum curvature; d) returns the optimised geometry in a CAD format for further analysis and processing; and e) the run-time of the derivative evaluation of the surface node displacements with respect to CAD shape parameters is very low, allowing tight coupling into a simultaneous (“one-shot”) design loop.

Chapter 8

CAD-based optimisation of a turbine

8.1 Introduction

As demonstrated in the previous chapter, the unique feature of the current work is that it allows to impose continuity constraints across patch interfaces, such as tangency (G^1) or curvature (G^2). This makes the approach applicable not just to isolated NURBS surfaces with fixed perimeter, but to smooth geometries described by an arbitrary number of NURBS patches. In airfoil or blade design, critical design parameters are thickness and trailing edge radius. Unconstrained optimisation considering only the fluid dynamics would result in a blade with sharp trailing edge as this improves efficiency. Multidisciplinary optimisation could take into account the thermal loads which would lead to a rounded trailing edge, but this is too expensive to conduct. As an alternative, a geometric constraint for a minimal trailing edge radius is typically imposed. Similarly, purely aerodynamic shape optimisation does not provide any mechanism to prevent the blade becoming thinner. However, a thinner blade will not be able to accommodate the internal cooling channels and a minimal thickness as a function of chord length is typically imposed.

In this chapter, the CAD-based parametrisation method in [98] is extended to include both thickness and radius constraints, hence termed ‘NURBS-based parametrisation with complex constraints, or NsPCC. The method uses a test-point approach to impose both the continuity constraint and the thickness/TE radius constraints. NsPCC is applied to the optimisation of a one-stage high pressure turbine with fillets and rotor tip gap, subject to both flow and geometric constraints. The remainder of the chapter is structured as follows: sec. 8.2 describes the CAD-based parametrisation using NURBS and the methodology for imposing both thickness and radius constraints is explained. The results of the constrained optimisation are presented in sec. 8.3, followed by conclusions in sec.8.4.

8.2 Thickness and TE radius (T/R) constraints

The thickness constraint is to ensure that the turbine blade is thick enough to to accommodate the trailing edge cooling channels and to prevent overheating, which is especially important toward the trailing edge. The radius constraint on the TE is to prevent an undesirable sharp TE from appearing. This can be done by implicitly incorporating the constraint in the parametrisation, e.g., parametrise a two dimensional airfoil with thickness and camber, but only use the camber line control points as design variables. Alternatively, one can also drastically reduce the number of design variables and allow as design variables only the translation and rotation of the airfoil, leading to a rather limited design space. The two methods can be combined if a three dimensional blade is optimised [87].

When the CAD-based parametrisation is used, the test point approach used for imposing the continuity constraint is adopted for imposing the thickness and radius constraints. Fig. 8.1 shows the test points for imposing the thickness (box with dashed line in fig. 8.1) and trailing edge radius (box with solid line in fig. 8.1) constraints for one section of the rotor blade. The thickness test points are deployed on the last 1/3 chord length of the rotor blade and the trailing edge radius test points are deployed on the trailing edge radius. A total of 21 sets of test points are deployed on the whole rotor blade evenly from root to the tip. Each set of test points at a certain blade height consists of 6 pairs of test points ($P_i, S_i \;\; i = 1$ to 6) for the thickness constraint and 5 test points ($A_i \;\; i = 0$ to 4) for the TE radius constraint, as illustrated in fig. 8.2. The number of the test points and their distribution are determined to approximately match the distribution of the the control points of the NURBS patches, in both the chord-wise and spanwise directions.

The thickness and radius constraint functions at n-th design step are formulated as

$$\begin{aligned} C_T^n(i) &= 1 - \min\left(\frac{d_i^n}{d_i^0}, 1\right) \quad \text{for } i = 1 \text{ to } 6 \\ C_R^n(i) &= 1 - \min\left(\frac{d_{i+6}^n}{d_{i+6}^0}, 1\right) \quad \text{for } i = 1 \text{ to } 7 \end{aligned}$$

where the distance functions d_i^n at n-th step are defined as

$$\begin{aligned} d_i^n &= \quad \|X^n(S_i) - X^n(P_i)\| \qquad \text{for } i = 1 \text{ to } 6 \\ d_{6+i}^n &= \quad \|X^n(A_0) - X^n(A_i)\| \qquad \text{for } i = 1 \text{ to } 4 \\ d_{10+i}^n &= \quad \|X^n(A_i) - X^n(A_{i+1})\| \qquad \text{for } i = 1 \text{ to } 3 \end{aligned}$$

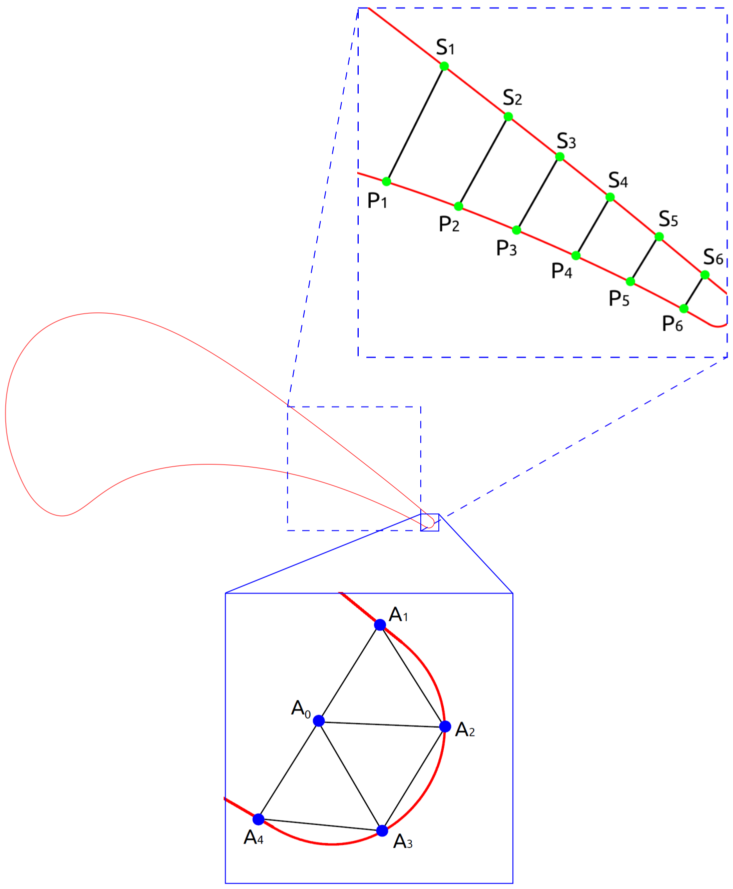

Figure 8.1: Test points for imposing thickness (box with dashed line) and TE radius (box with solid line) constraints at one blade height.

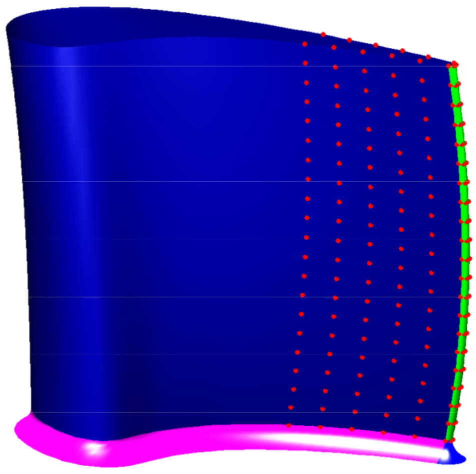
Figure 8.2: Deployment of the test points on the rotor blade.

A total of $6 \times 21 = 126$ thickness cost functions C_T^n and $7 \times 21 = 147$ TE radius cost functions C_R^n are imposed as equality constraints:

$$C_T^n(i) = 0 \quad \text{for } i = 1 \text{ to } 126$$

$$C_R^n(i) = 0 \quad \text{for } i = 1 \text{ to } 147$$

which implies that the thickness and radius can not be reduced below the original values throughout the optimisation. For ease of notation, the two constraint functions are combined to form the thickness the thickness and TE radius (T/R) constraint function

$$C_{T/R} = \begin{bmatrix} C_T \\ C_R \end{bmatrix}$$

In order to impose the T/R constraint without violating the continuity constraint, the derivative of the T/R constraint w.r.t. the control points, $dC_{T/R}/dP$, is projected into the null-space of the continuity constraint, i.e., KerC, to arrive at the derivative w.r.t. the design variables α:

$$\frac{dC_{T/R}}{d\alpha} = \frac{dC_{T/R}}{dX_s} \cdot \frac{dX_s}{dP} \cdot \text{KerC}$$

Once the derivative w.r.t. the design variable is computed, a Newton step is taken to perturb the design variable

$$\Delta\alpha^n = -\left(\frac{dC_{T/R}^n}{d\alpha}\right)^+ \cdot C_{T/R}^n$$

where the superscript $^+$ denotes pseudo inverse of the matrix which is rectangular but not rank deficient. The pseudo inversion is done with a standard QR decomposition algorithm. To demonstrate the effectiveness of the thickness and radius constraints, the optimisation is performed with and without the constraint, results are described in sec. 8.3.

8.3 Result

This section will first describe the optimisation case in terms of geometry, meshing, parametrisation and the flow characteristics.Then the optimisation strategy is explained, followed by the optimisation results illustrated by both the change in the flow field and the shape. For computing both the steady and unsteady nonlinear flow solution and the steady discrete adjoint solutions, the HYDRA code is used. The linear elasticity based volume mesh deformation algorithm is implemented as a standalone code, and at each design step, the mesh information and the perturbation surface mesh are exported from HYDRA to be deformed and then read back into HYDRA for computation.

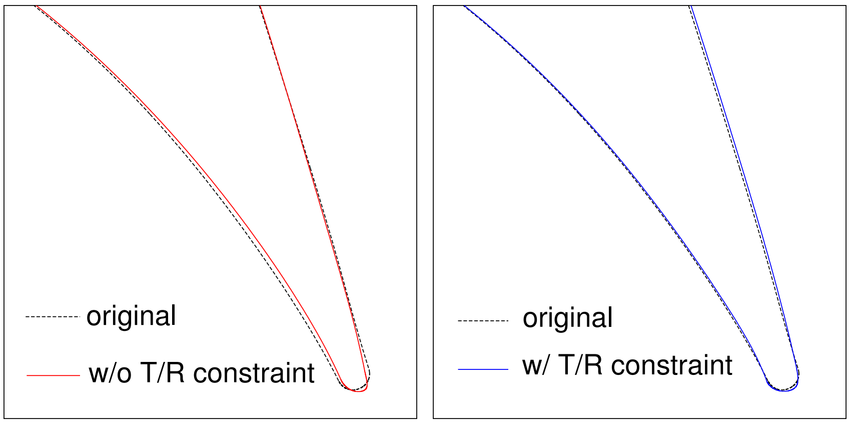

Figure 8.3: Comparison of the cross section profile without (left) and with (right) the thickness and TE radius constraint at 40% blade height.

8.3.1 Case description

The case used in this work is a one-stage high pressure (HP) turbine consisting of a nozzle guide vane (NGV) and a rotor. This case is produced to have a representative geometry of an actual engine component. Optimisation for a similar case can be found in [87, 86] where the geometry is more realistic. Shown in fig. 8.4 is the stage with multiple NGVs and rotors, and the casing surfaces, inlet, outlet and periodic boundary are not shown for illustrative purposes. For the steady state simulation, only one NGV and one rotor are used. The NGV has fillets at both ends and the rotor has a fillet at the hub, to avoid sharp corners. The rotor tip is a surface of revolution intersected by the pressure and suction sides, and the tip clearance is constant along the axial position.

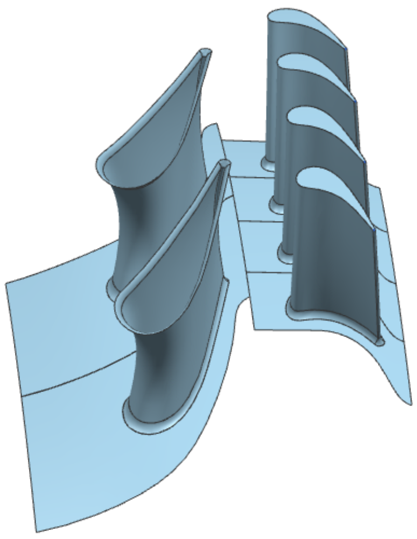

Figure 8.4: Geometry of the HP turbine stage.

The 3D geometry of both blades are modelled with NURBS patches, four patches form the rotor surface except the tip, their distribution with the control points for the rotor is shown in fig. 8.5. The four patches consist of 33×20, 6×20, 33×5 and 6×5 control points respectively, resulting in a total of 2925 DoF. After constraints are taken into account, the design degree of freedom reduces to around 2000. The density of the control points is chosen such that the mesh points are one order of magnitude denser than the control points. Otherwise, undesirable wavy shape will be produced during optimisation unless the waviness could be penalised using an additional, but most likely, ad-hoc criteria. Note that the tip surface is not a design surface, as the tip clearance is not allowed to change. The tip surface is updated by following the rim of the blade surfaces.

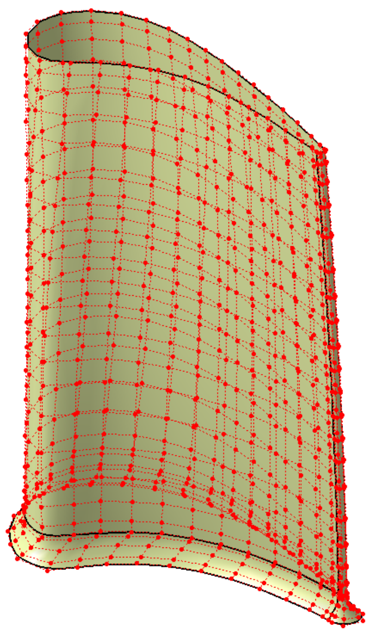
Figure 8.5: Rotor blade along with the control points.

8.3.2 Meshing

The stage is meshed using ANSYS ICEM CFD with a total of 3.5 million hexahedral elements. The rotor tip gap is meshed with 20 layers of cells with growth rate around 1.3. To improve the accuracy in computing the flow in the boundary layer, the first layer of cells on viscous walls maintains a Y+ value in the order of 1, and a growth ratio of no greater than 1.3 in the normal direction for at least 30 layers. A post-processing step is taken after the flow is computed to verify that indeed the Y+ value over all the viscous walls never exceeds 5. A preliminary grid convergence study has been performed regarding both (a) the global volume mesh density and (b) the tip gap mesh density, and it is found that that the current mesh is fine enough to capture the main feature of the flow.

8.3.3 Flow solution

As boundary conditions, the total pressure ($p_{0,1}$), total temperature ($T_{0,1}$) and the flow angles are prescribed at the NGV inlet boundary as a radial profile while the static pressure (p_3) profile is prescribed at the rotor exit. The NGV and the rotor are coupled using a mixing plane. The turbulence is modelled with a one equation Spalart–Allmaras turbulence model with wall function based on Spalding's law of the wall. The flow is assumed to be fully turbulent, and no transition considered in this work.

The nonlinear flow solver fully converges the steady state flow calculation and the discrete adjoint solver using the same time-marching scheme fully converges the adjoint solution. The typical convergence history of both flow and adjoint is shown in fig. 8.6. In practice, for computational efficiency, both the flow and adjoint solvers are run for only 300 iterations with residual reduction of approximately 5 orders of magnitude. Running more iterations has negligible influence on the flow solution and design sensitivity. The optimisation terminates due to mesh failure before reaching the local minimum, it is also assumed that the chosen convergence level for flow and adjoint are adequate and do not have a significant impact on the optimisation result. Let us also recall that the aim of this chapter is not to find the optimal rotor shape, but to demonstrate the NURBS-based parametrisation with geometric constraints.

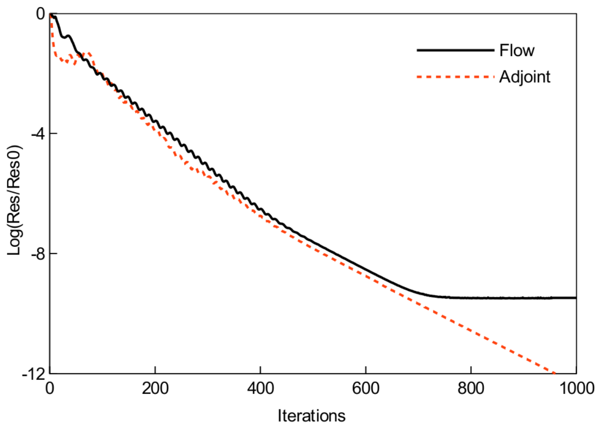

Figure 8.6: Convergence history of flow and adjoint solvers.

The converged flow for the initial geometry is shown in fig. 8.7 and fig. 8.8, illustrating the tip vortex and two passage vortices. The stage is operating at design point and thus the overall flow does not exhibit any major separation except the rotor tip vortex and passage vortices, which are believed to be the main sources of loss. Only the visualisation

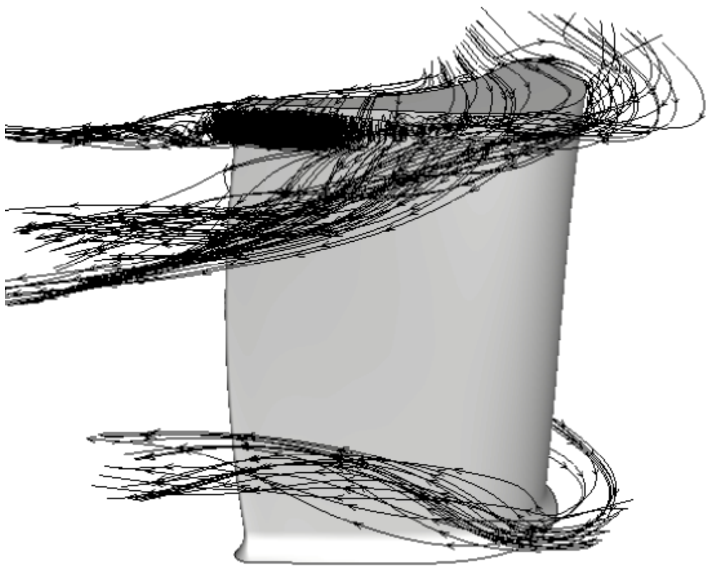
Figure 8.7: Streamline plot at different axial locations for rotor.

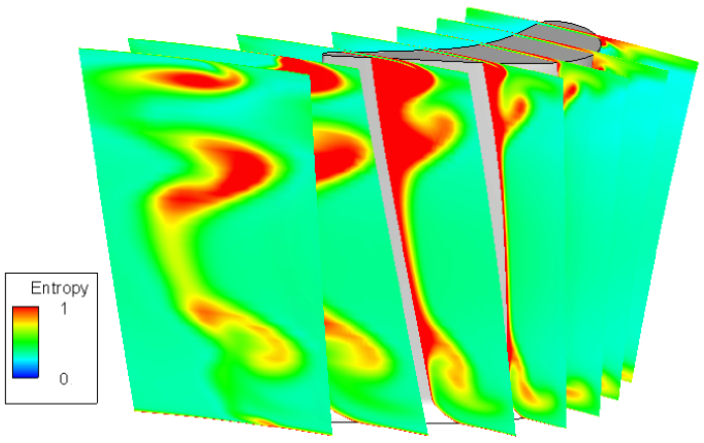

Figure 8.8: Normalised entropy contour plot at different axial locations of the rotor.

of the flow for the rotor is shown in the figures since only the rotor shape will be optimised in this work. Nevertheless, the same approach can be used to optimise the NGV + rotor simultaneously to explore a larger design space, however this is beyond the scope of this thesis.

8.3.4 Optimisation

The optimisation goal is to improve the stage isentropic efficiency. Meanwhile, it is important that the stage inlet capacity and the rotor reaction ratio do not deviate by more than 0.1% and 0.05% respectively. The three output functions are formulated as follows:

$$\begin{aligned}
&\text{Efficiency:} && \eta = \frac{(\dot{m}h)_{0,1} - (\dot{m}h)_{0,3}}{(\dot{m}h)_{0,1} - (\dot{m}h)_{0s,3}} \\
&\text{Inlet capacity:} && \phi = \frac{\dot{m}\sqrt{T_{0,1}}}{p_{0,1}} \\
&\text{Reaction:} && \chi = \frac{h_2 - h_3}{h_{0,1} - h_{0,3}}
\end{aligned}$$

The flow constraints are enforced using the inlet capacity and reaction as penalty functions and the extended objective function is formulated as

$$J = \frac{\eta - \eta^0}{\eta^0} - a_1 \left(\frac{\phi - \phi^0}{\phi^0}\right)^2 - a_2 \left(\frac{\chi - \chi^0}{\chi^0}\right)^2 \tag{8.1}$$

and the design derivative is

$$\frac{dJ}{d\alpha} = \frac{d\eta}{d\alpha} - 2a_1(\phi - \phi^0)\frac{d\phi}{d\alpha} - 2a_2(\chi - \chi^0)\frac{d\chi}{d\alpha} \tag{8.2}$$

where the weighting coefficients a_1 and a_2 are chosen manually at each design step based on experience.

A steepest descent optimiser is used to drive the design perturbation

$$\alpha^{n+1} \leftarrow \alpha^n - c\frac{dJ/d\alpha}{\|dJ/d\alpha\|_2} \tag{8.3}$$

and the step size c is chosen such that the resulting mesh perturbation at each design iteration is bounded, to help stabilise the mesh deformation. The use of a more sophisticated optimiser such as those based on Sequential Quadratic Programming (SQP), would greatly improve the efficiency of the optimisation. However, it is found in this work that the robustness of the mesh deformation algorithm has the most critical influence on the overall optimisation, and consequently, we chose the steepest descent optimiser in order to have a direct control on the perturbation step size, because other advanced optimisers using quasi-Newton algorithms tend to predict a large perturbation step that the current mesh deformation would not be able to cope with. Alternatively, one could limit the predicted step size. However, this is not explored in this thesis and a simple steepest descent method is used with limited step size.

The flow chart of the optimisation is shown in fig. 8.3.4. First, the baseline mesh is prepared using the geometry from the STEP file. Once the flow and adjoint computations have finished, the design sensitivity is assembled, and a design perturbation $\delta\alpha$ is returned by the steepest descent optimiser to perturb the geometry and the surface mesh simultaneously. The linear elasticity mesh deformation module is used to perturb the volume mesh in order to run the next optimiser iteration. The optimisation procedure is automated to iterate until the design criterion is met and both the mesh and the STEP file for the optimised shape are available for output.

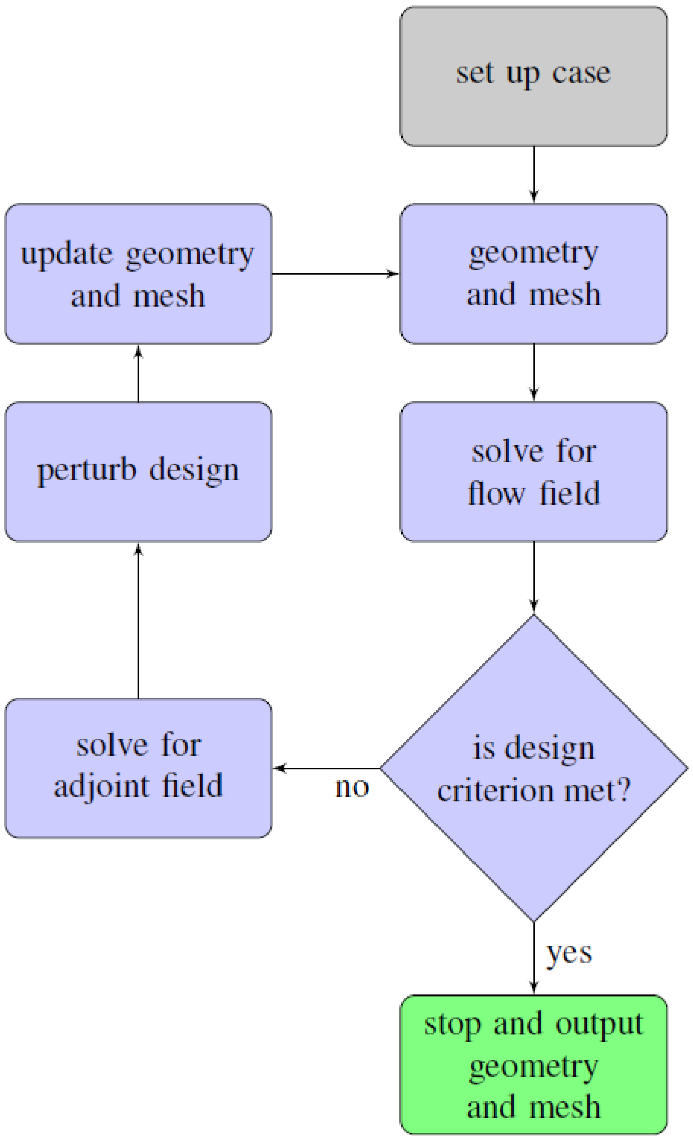

Figure 8.9: Optimisation flow chart.

8.3.5 Optimisation result

A total of 20 iterations of optimisation with the continuity and geometric constraints are performed before the mesh becomes invalid with elements of negative volume and the optimisation terminates. Although not shown here, with the CAD-based parametrisation, one is able to re-mesh using the updated STEP file and continue with the optimisation using the new mesh. To show the impact of incorporating constraints on both the capacity and reaction ratio, the optimisation is performed first without flow constraints, just to evaluate the range of deviation for both capacity and reaction if not constrained. The evolution of all three flow outputs are shown in fig. 8.10. After 20 design iterations, efficiency is improved by 0.6%, while the capacity and reaction have deviated by 0.2% and 1.2% respectively, well over the thresholds, especially the reaction ratio. Next, the same optimisation is performed with constraints, where the weighting coefficients are tuned each design step manually based on the trend of the cost function evolution. At the final stage, i.e., from 15-th design step, the step size and weighting functions are carefully chosen to ensure the reaction ratio evolve to a value within the tolerance by the 20-th design step. The results is shown in fig. 8.10 compared with the unconstrained one. The efficiency is improved by 0.4%, while both the capacity and reaction are within the threshold of 0.1% and 0.05%.

To examine further the loss reduction mechanism, the normalised entropy contours at the exit plane are shown in fig. 8.11. It is clear that the passage vortex is weakened while the tip vortex on the contrary is strengthened.

To examine the shape change after 20 design iterations with constraints, the original and optimised rotor blade surfaces compared in fig. 8.12 by importing both STEP file into NX 8.0. On the pressure side, the major shape change is the outward displacement of the leading edge and inward displacement of the blade elsewhere. For the suction side, the shape mode is slightly more complex. The suction side for the upper half of the blade is moving outward except at 30% chord, which is where the main tip vortex starts to shed.

Finally, it is worth noting that the efficiency improvement without incorporating the thickness and TE radius constraints is 0.58%, a further 0.18% improvement compared to that with the constraints, indicating that a significant but infeasible improvement is suggested by merely making the blade thinner and TE sharper, further underlining the importance of imposing the thickness and TE radius constraints in order to achieve a useful design improvement.

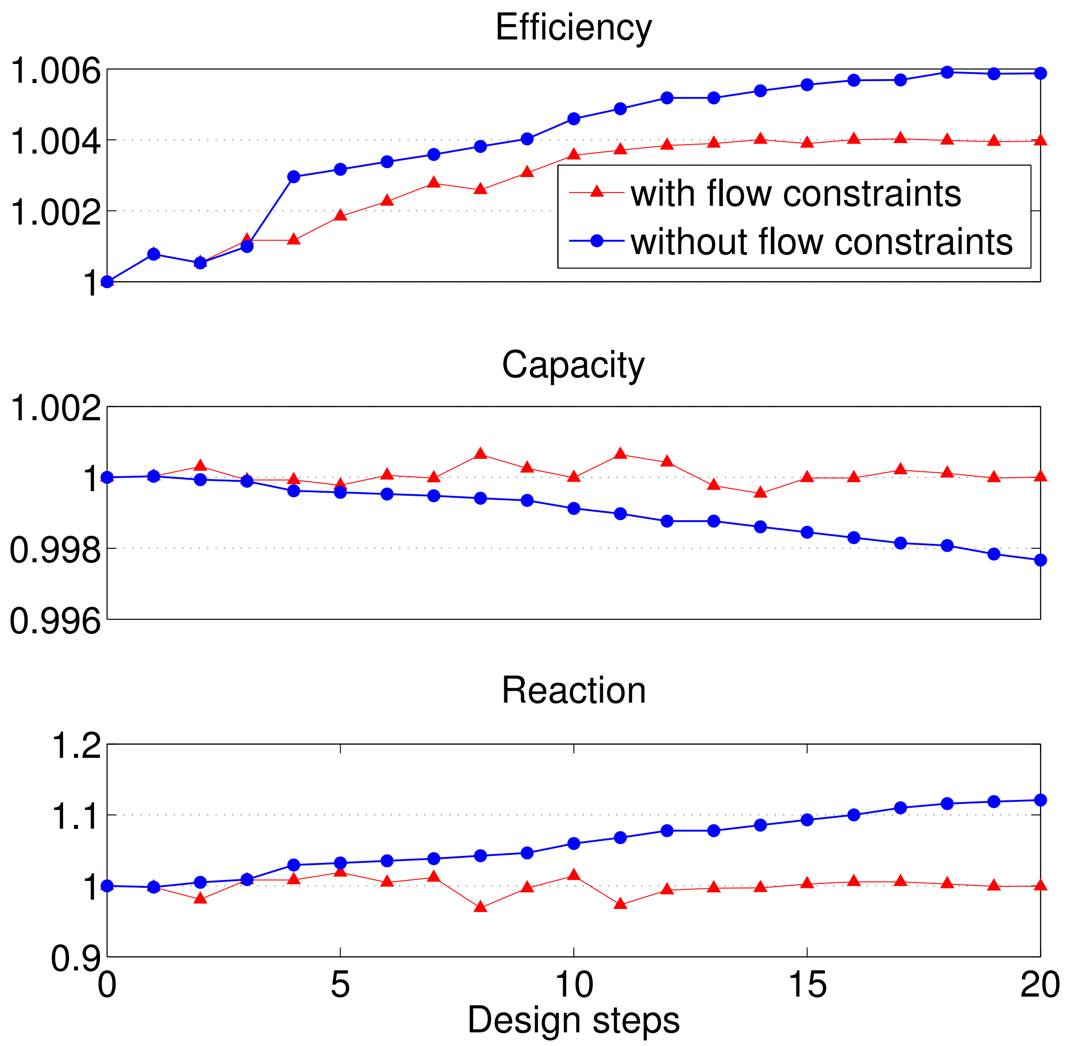

Figure 8.10: Convergence history of various flow output (efficiency improvement and the deviation of both inlet capacity and reaction) at different optimisation steps.

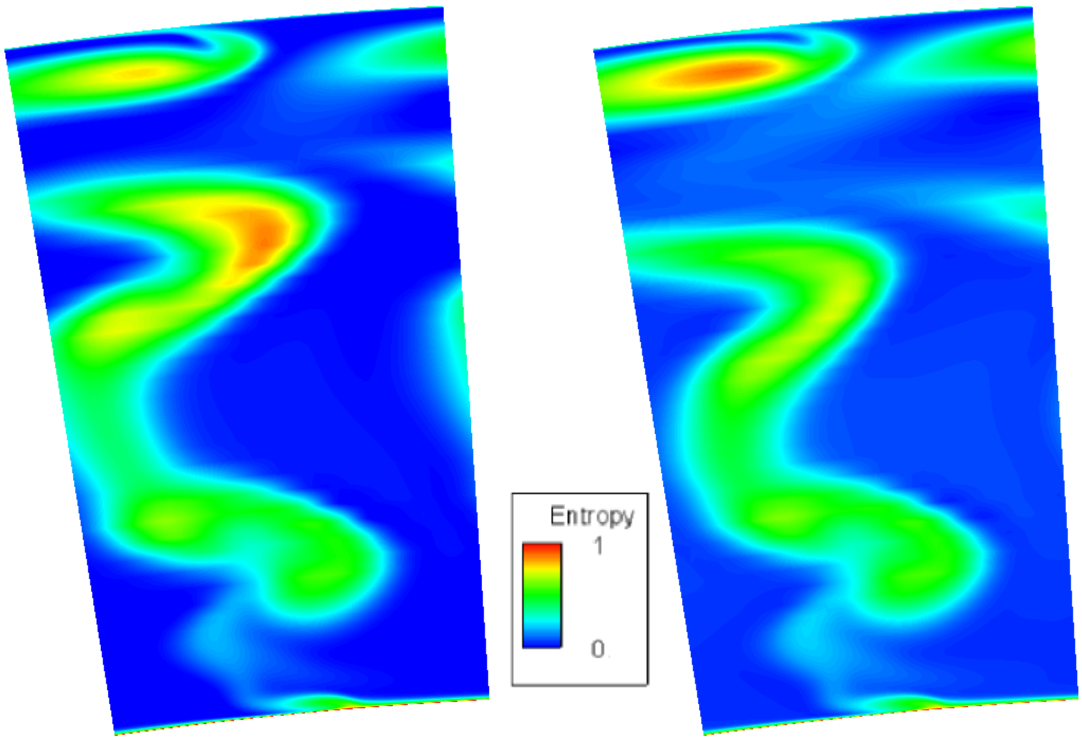

Figure 8.11: Entropy generation contour at the rotor exit. Left: original, right: optimised.

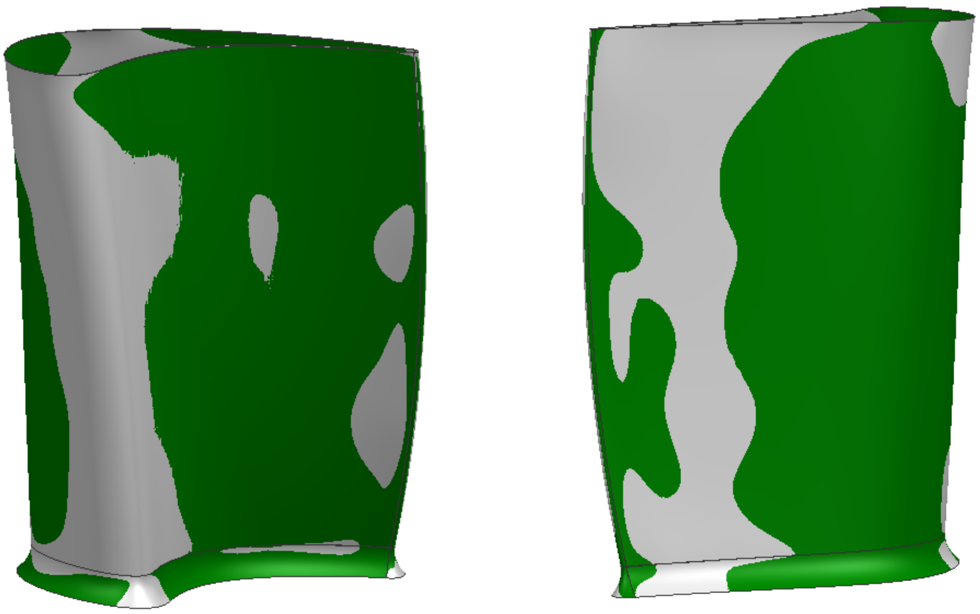

Figure 8.12: Comparison of the original (dark green) and optimised (silver) shapes, visualised in NX 8.0.

8.4 Summary

An extension of the CAD-based parametrisation termed ‘NURBS-based parametrisation with complex constraints’, or NsPCC, is developed and applied to the optimisation of a one-stage high pressure turbine with tip gap and fillets. The parametrisation method automatically takes into account the G^1 continuity constraint among the NURBS patches forming the rotor blade and returns an optimised shape in a CAD format. In addition, the blade thickness and trailing edge radius are also constrained to avoid the thinning and sharpening using a test-point approach.

Using an advanced mesh deformation algorithm based on linear elasticity, a relatively large deformation can be achieved without having to re-mesh. After 20 design cycles, an increase of 0.4% in stage efficiency is achieved while deviation of both the inlet capacity and reaction ratio are kept under prescribed thresholds.

Future work will be focused on incorporating more advanced geometric features such as a rotor tip squealer, cooling slot, moving intersection lines, etc. The effect of a more robust mesh deformation algorithm or automatic re-meshing will also be examined in future work to achieve the full convergence of the optimisation.

Chapter 9

Conclusions and future work

9.1 Conclusions

The results presented in this thesis are focused on aerodynamic shape optimisation based on Reynolds-Averaged Navier-Stokes (RANS) equations using the discrete adjoint method with emphasis on the robustness of the adjoint solver when applied to industrial applications. When applying the adjoint method for industrial shape optimisation, the constantly encountered frustration for application engineers is the LCO-convergence of the nonlinear flow and the subsequent non-convergence of the adjoint solver which is a show stopper for gradient-based shape optimisation. The two existing algorithms for stabilising the adjoint in these cases are GMRES and RPM, both of which are able to solve linear equation despite a non-contractive system Jacobian. Both stabilisation methods tend to have high memory requirement for cases with complex geometry and unstable flow features. The author believes that a more important drawback is the non-convergence of the nonlinear flow solution. There are two consequences of not being able to fully converge the flow: (i) the stabilised and fully converged adjoint solution depends on the particular flow solution snapshot the adjoint solution is based on. Neither using an averaged flow solutions nor averaging the adjoint solutions based on different flow solutions sampled from the LCO convergence seems to give a suitably averaged adjoint solution in the sense that none of these options can be validated to machine accuracy, due to the uncertainty of the finite residual/error of the nonlinear flow solution that is not fully converged. In addition to the lack of robustness of the adjoint solver, another difficulty of using the adjoint method for industrial aerodynamic shape optimisation concerns the parametrisation methods. Parametrisation is the first decision to be made when aerodynamic shape optimisation is performed and it directly determines the optimal design that could be reached. Although there is no universally preferred parametrisation as it strongly depends on the applications, the author advocates the use of the NURBS-based parametrisation method due to its better compatibility with the CAD software which allows the design-analyse-optimise process to be integrated more organically.

This thesis contributes to the two aspects of the aerodynamic shape optimisation mentioned above: i) a more robust adjoint solver that can be applied to a wider flow regime and ii) a NURBS-based parametrisation method with various geometric constraints. The contribution to the two aspects are summarised in the two subsections below.

9.1.1 Robust adjoint solver using JT-KIRK

To facilitate the wide use of discrete adjoint solvers for industrial shape optimisation applications, a novel stabilisation method has been developed for stabilising some typically unstable cases of industrial relevance.the The algorithm development of the proposed method is done in the HYDRA code. The time-stepping method of HYDRA nonlinear RANS flow solver and its corresponding adjoint solver is based on the block-Jacobi preconditioner, which has been well accepted to be the most efficient and reliable algorithm when combined with multigrid acceleration for solving viscous flows. The block-Jacobi solvers usually perform well on cases at a design point, i.e., well attached flows, not circulation zone, etc. However, for cases with more complex configuration, i.e., tip gap, off-design point, etc., the block-Jacobi flow solver tends to converge to LCO, and consequently the adjoint diverges.

The proposed adjoint stabilisation method is named Jacobian-Trained Krylov-Implicit-Runge-Kutta, or 'JTKIRK'. JTKIRK algorithm formulates the implicit time-stepping scheme by replacing the block-Jacobi preconditioner by the 1st-order approximate Jacobian. When inverting the preconditioner matrix, instead of inverting each diagonal block directly using Gauss elimination, ILU(0) preconditioned GMRES solver is used. The Runge-Kutta integration that wraps around the block-Jacobi preconditioning is kept intact, to provides sufficient high-frequency damping in order for the implicit scheme to be a good smoother for multigrid. For the simplicity of the parallelism, the approximate Jacobian and its ILU preconditioner are formed for the nodes local to each partition and the parallel communication is only needed when computing the residual. The performance thus degenerates with increased number of processors. The JT-KIRK algorithm is compared with a competing implicit algorithm using Symmetric Gauss Seidel (SGS), and results show that JT-KIRK nonlinear flow solver outperforms the block-Jacobi solver by 70% and SGS by 20% for cases that can be converged by block-Jacobi, thus called 'stable' cases. In addition, two 'unstable' cases, i.e., cases that the block-Jacobi flow solver converged to LCO and the adjoint solver diverged, are used to demonstrate the stabilisation effect of the JT-KIRK algorithm. The JT-KIRK flow solver can fully converge both unstable cases and subsequently the corresponding adjoint solver converges as well for both cases. Eigen analysis shows that the JT-KIRK algorithm significantly clusters the eigenvalues of the system Jacobian towards the origin and thus eliminates the outliers that are previously responsible for the divergence of the block-Jacobi adjoint solver.

9.1.2 NURBS-based parametrisation with geometric constraints

A NURBS-based parametrisation method using a CAD kernel written in FORTRAN has been previously developed in the group, and applied to the inverse design of a two dimensional airfoil using prescribed pressure distribution [99]. The method uses the coordinates of the control points of the Non-uniform Rational B-spline (NURBS) patches as the design variables for performing shape optimisation. The author's main contribution is to develop the algorithm for imposing various geometric constraints: i) continuity constraints, ii) thickness constraint and iii) radius constraint, to enable the application to more complex geometries with multiple NURBS patches.

The geometric continuity across the connecting patches are guaranteed via a test point approach. A set of test points are deployed along the edges shared by two patches, and the penalty functions measuring the deviation of the geometric continuity are linearised with respect to the control points. After calculating the null space of the linearised penalty functions, the allowable movement of the control points can then be formulated as the linear combination of the basis vectors of the kernel space. Since the penalty function for G^1 continuity is nonlinear, a few additional steps are taken to perturb the design in the range space of the linearised penalty function to fully recover the geometric constraints. To formulate the thickness and radius constraints, the same testpoint approach is followed. A group of test points are deployed on the part of the patches where thickness and radius are formulated and constraints are required. The penalty function measuring the deviation of the meta-thickness and radius from the threshold is then formulated and linearised to compute their derivatives. The derivatives of the thickness and radius penalty functions are then used to take a Newton step to perturb the design variable to drive the deviation to zero. Note that the derivatives of the thickness and radius penalty functions are projected into the null space of the continuity constraint first before the Newton step is taken, so that the design perturbation does not in turn affect the continuity constraint.

The resulting novel parametrisation method, named NURBS-based Parametrisation with Complex Constraints, or 'NsPCC', is then successfully applied to two industrial test cases: i) an S-Bend from the automotive industry and ii) a one-stage HP turbine from the turbomachinery industry.

9.2 Future work

The findings and development documented in this thesis has contributed to the maturity of the adjoint-based shape optimisation for industrial applications. However, in order to further increase its technology readiness level and to make a disruptive impact on the current aerodynamic shape design and optimisation routines practised in industry, which is mainly non-gradient based optimisation, the following aspects, as an extension of this work, need to be strengthened.

9.2.1 Towards a direct flow solver

From the first-hand experience of many application engineers using adjoint solvers for shape optimisation or simply using the surface sensitivity map to guide the manual optimisation, the lack of robustness of the nonlinear flow and adjoint solvers is still the major concern. Nowadays RANS simulation aims at capturing the separation as accurately as possible and thus the focal point is on turbulence models. Unless for a simple subsonic Euler flow simulation or a turbulence flow calculation with the one-equation SA model, full convergence of the RANS solver with other more sophisticated turbulence models ranging from 2 to 4 equations is neither practically sought nor necessary, let alone when there is shock/boundary layer interaction or unstable separation/circulation.

The JT-KIRK algorithm provides a good starting point for moving towards a direct solver. The infrastructure already laid for computing the 1st-order Jacobian can be easily be rearranged for computing the 2nd-order exact Jacobian and the 1st-order approximate Jacobian preconditioner matrix could be replaced by first the blending of 1st and 2nd-order Jacobian matrices and eventually the 2nd-order exact Jacobian at the vicinity of the stationary point. This would then need a substantial amount of expertise in solution steering technique in order to finally reach the stationary point.

At the final stage of the solver steering towards the direct solver, the main obstacle will again be the large memory required to store the Krylov vectors. Similar to the JT-KIRK algorithm proposed in this work, a multigrid technique should be explored in combination with an ILU preconditioner to reduce the Krylov vectors needed.

9.2.2 Robust meshing capability

Once an adjoint solution and the design sensitivity can be computed with a robust direct solver, the main factor that could hinder the shape optimisation is a reliable meshing tool. Here the meshing tools not only refer to a particular mesher, but also includes automatic re-meshing, mesh repair and adaptation capability. When surveying most industrial shape optimisation results using adjoints, including the ones in this thesis, none of them has converged in terms of the optimisation steps, i.e., the local minimum is not reached. This is quite often due to mesh failure after a few design steps. If one does not have a parametric meshing tool that allows automatic mesh update or re-meshing after a design update, mesh deformation could be used. However, most mesh deformation algorithm nowadays cannot cope well with very large deformation for a high quality body-fitted mesh refined in the boundary layer and wake. In addition, the resulting mesh quality usually decreases monotonically over optimisation iterations. Therefore, the ideal way of updating the mesh at optimisation steps is to use mesh deformation until the accumulated deformation is too large and mesh quality has decreased below a threshold, and then a re-meshing step or mesh repair step is performed to restore a high quality mesh for the following optimisation steps. In addition, a natural question to ask is how much of the design improvement is due to the difference in mesh quality at each optimisation iteration. This then leads to another related topic on meshing, error estimation. Only when the error could be quantified and bounded during each optimisation step, the adjoint-based shape optimisation could gain much more attention from industrial users.

Bibliography

[1] http://cfl3d.larc.nasa.gov/.

[2] http://turbmodels.larc.nasa.gov/flatplate.html.

[3] PG A. Cizmas and Joaquin I Gargoloff. Mesh generation and deformation algorithm for aeroelasticity simulations. *Journal of Aircraft*, 45(3):1062–1066, 2008.

[4] FDP WG04 AGARD. Experimental data base for computer program assessment. Technical report, AGARD-AR-138, 1979.

[5] W Kyle Anderson and V Venkatakrishnan. Aerodynamic design optimization on unstructured grids with a continuous adjoint formulation. *Computers & Fluids*, 28(4):443–480, 1999.

[6] SA Berger, L. Talbot, and LS Yao. Flow in curved pipes. *Annual Review of Fluid Mechanics*, 15(1):461–512, 1983.

[7] Michel Bergmann, Laurent Cordier, and Jean-Pierre Brancher. Optimal rotary control of the cylinder wake using proper orthogonal decomposition reduced-order model. *Physics of Fluids*, 17(9):097101, 2005.

[8] Marco Evangelos Biancolini. Mesh morphing and smoothing by means of radial basis functions (RBF). *Handbook of Research on Computational Science and Engineering: Theory and Practice: Theory and Practice*, page 347, 2011.

[9] ME Biancolini, C Biancolini, E Costa, D Gattamelata, and PP Valentini. Industrial application of the meshless morpher RBF morph to a motorbike windshield optimisation. In *the proceedings of The European Automotive Simulation Conference (EASC)*, pages 6–7, 2009.

[10] ME Biancolini, IM Viola, and M Riotte. Sails trim optimisation using CFD and RBF mesh morphing. *Computers & Fluids*, 93:46–60, 2014.

[11] Robert T Biedron and James L Thomas. Recent enhancements to the FUN3D flow solver for moving-mesh applications. AIAA Paper 2009-1360, 2009.

[12] A. Brandt. Multigrid techniques: 1984 guide. In *14th VKI Lecture Series on Computational Fluid Dynamics 1984-04*, Rhode-St.-Genèse, Belgium, March 12-16 1984. Von Karman Institute for Fluid Dynamics.

[13] M Sergio Campobasso and Michael B Giles. Effects of flow instabilities on the linear analysis of turbomachinery aeroelasticity. *Journal of Propulsion and Power*, 19(2):250–259, 2003.

[14] Michele Sergio Campobasso. *Effects of flow instabilities on the linear harmonic analysis of unsteady flow in turbomachinery.* PhD thesis, University of Oxford, 2004.

[15] Michele Sergio Campobasso and Michael B Giles. Stabilization of a linear flow solver for turbomachinery aeroelasticity using recursive projection method. *AIAA Journal*, 42(9):1765–1774, 2004.

[16] David A Caughey and Mohamed M Hafez. *Frontiers of Computational Fluid Dynamics.* World Scientific, 1998.

[17] Faidon Christakopoulos, Dominic Jones, and Jens-Dominik Müller. Pseudo-timestepping and verification for automatic differentiation derived CFD codes. *Computers & Fluids*, 46(1):174–179, 2011.

[18] B. Christianson. Reverse accumulation and implict functions. *Optimization Methods and Software*, 9(4):307–322, 1998.

[19] John G Cleary and Geoff Wyvill. Analysis of an algorithm for fast ray tracing using uniform space subdivision. *The Visual Computer*, 4(2):65–83, 1988.

[20] F. Courty, A. Dervieux, B. Koobus, and L. Hascoët. Reverse automatic differentiation for optimum design: from adjoint state assembly to gradient computation. *Optimization Methods and Software*, 18(5):615–627, 2003.

[21] PI Crumpton, P Moinier, and MB Giles. An unstructured algorithm for high Reynolds number flows on highly stretched grids. *Numerical methods in laminar and turbulent flow*, pages 561–572, 1997.

[22] A De Boer, MS Van der Schoot, and H Bijl. Mesh deformation based on radial basis function interpolation. *Computers & structures*, 85(11):784–795, 2007.

[23] Christoph Degand and Charbel Farhat. A three-dimensional torsional spring analogy method for unstructured dynamic meshes. *Computers & structures*, 80(3):305–316, 2002.

[24] Richard P Dwight. *Efficiency improvements of RANS-based analysis and optimization using implicit and adjoint methods on unstructured grids.* PhD thesis, the University of Manchester, 2006.

[25] Richard P Dwight and Joël Brezillon. Efficient and robust algorithms for solution of the adjoint compressible Navier–Stokes equations with applications. *International Journal for Numerical Methods in Fluids*, 60(4):365–389, 2009.

[26] Kivanc Ekici, Kenneth C Hall, Huang Huang, and Jeffrey P Thomas. Stabilization of explicit flow solvers using a proper-orthogonal-decomposition technique. *AIAA Journal*, 51(5):1095–1104, 2013.

[27] D.M. Fudge, D.W. Zingg, and R. Haimes. A CAD-free and a CAD-based geometry control system for aerodynamic shape optimization. AIAA Paper 2005-0451, 2005.

[28] M. B. Giles, M. C. Duta, J.-D Müller, and N. A. Pierce. Algorithm developments for discrete adjoint methods. *AIAA Journal*, 41(2):198–205, 2003.

[29] M.B. Giles, D. Ghate, and M.C. Duta. Using automatic differentiation for adjoint CFD code development. In *Recent Trends in Aerospace Design and Optimization.* Tata-McGraw Hill, New Delhi, 2005. Post-SAROD-2005, Bangalore, India.

[30] Michael B. Giles. On the iterative solution of adjoint equations. In George Corliss, Christèle Faure, Andreas Griewank, Laurent Hascoët, and Uwe Naumann, editors, *Automatic Differentiation of Algorithms*, pages 145–151. Springer New York, 2002.

[31] Michael B Giles, Mihai C Duta, Jens-Dominik Müller, and Niles A Pierce. Algorithm developments for discrete adjoint methods. *AIAA Journal*, 41(2):198–205, 2003.

[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] Robert Haimes. CAPRI: Computational Analysis Programming Interface): A Solid Modeling Based Infra-structure for Engineering Analysis and Design. CAPRI user's guide, MIT, Revision 1.00, Dec. 18, 2000.

[34] L. Hascoët and V. Pascual. Tapenade 2.1 user's guide. Technical Report 0300, INRIA, 2004.

[35] L. Hascoët and V. Pascual. The Tapenade Automatic Differentiation tool: Principles, Model, and Specification. *ACM Transactions On Mathematical Software*, 39(3), 2013.

[36] S.B. Hazra, V. Schulz, J. Brezillon, and N.R. Gauger. Aerodynamic shape optimization using simultaneous pseudo-timestepping. *Journal of Computational Physics*, 204:46–64, 2005.

[37] M. Hojjat, E. Stavropoulou, and K.-U. Bletzinger. The vertex morphing method for node-based shape optimization. *Computer Methods in Applied Mechanics and Engineering*, 268:494–513, 2014.

[38] Stefan Jakobsson and Olivier Amoignon. Mesh deformation using radial basis functions for gradient-based aerodynamic shape optimization. *Computers & Fluids*, 36(6):1119–1136, 2007.

[39] A Jameson and JC Vassberg. Studies of alternative numerical optimization methods applied to the brachistochrone problem. *Computational Fluid Dynamics Journal*, 9(3):281–296, 2000.

[40] Antony Jameson, Wolfgang Schmidt, Eli Turkel, et al. Numerical solutions of the Euler equations by finite volume methods using Runge-Kutta time-stepping schemes. AIAA-CP 81-1259, 1981.

[41] H Jasak and HG Weller. Application of the finite volume method and unstructured meshes to linear elasticity. *International Journal for Numerical Methods in Engineering*, 48(2):267–287, 2000.

[42] A Jaworski, P Cusdin, and JD Müller. Uniformly converging simultaneous timestepping methods for optimal design. *Eurogen, Munich*, 2005.

[43] A. Jaworski and J.-D. Müller. Toward modular multigrid design optimisation. *Lecture Notes in Comp. Sci. Eng.*, 64:281–291, 2008.

[44] D. Jones, J.D. Müller, and F. Christakopoulos. Preparation and assembly of discrete adjoint CFD codes. *Computers & Fluids*, 2011.

[45] Dominic Jones and Jens-Dominik Müller. CFD development with automatic differentiation. *50th AIAA Aerospace Sciences Meeting*, AIAA 2012-573, 2012.

[46] S Kammerer, JF Mayer, M Paffrath, U Wever, and AR Jung. Three-dimensional optimization of turbomachinery bladings using sensitivity analysis. In *ASME Turbo Expo 2003, collocated with the 2003 International Joint Power Generation Conference*, pages 1093–1101. American Society of Mechanical Engineers, 2003.

[47] Dana A Knoll and David E Keyes. Jacobian-free Newton–Krylov methods: a survey of approaches and applications. *Journal of Computational Physics*, 193(2):357–397, 2004.

[48] Joshua A Krakos and David L Darmofal. Effect of small-scale unsteadiness on adjoint-based output sensitivity. AIAA Paper 2009-4274, 2009.

[49] M. Lallemand, H. Steve, and A. Dervieux. Unstructured multigridding by volume agglomeration: current status. *Computers and Fluids*, 21(3):397–433, 1992.

[50] Stefan Langer. Agglomeration multigrid methods with implicit Runge–Kutta smoothers applied to aerodynamic simulations on unstructured grids. *Journal of Computational Physics*, 277:72–100, 2014.

[51] L Lapworth. Hydra-CFD: a framework for collaborative CFD development. In *International Conference on Scientific and Engineering Computation (IC-SEC), Singapore, June*, volume 30, 2004.

[52] Randall J LeVeque. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge University Press, 2002.

[53] H-D Li, L He, YS Li, and R Wells. Blading aerodynamics design optimization with mechanical and aeromechanical constraints. In *ASME Turbo Expo 2006: Power for Land, Sea, and Air*, pages 1319–1328. American Society of Mechanical Engineers, 2006.

[54] Weiyu Liu. *Development of gradient-enhanced kriging approximations for multidisciplinary design optimization.* PhD thesis, University of Notre Dame, 2003.

[55] Karthik Mani and Dimitri J Mavriplis. Geometry optimization in three-dimensional unsteady flow problems using the discrete adjoint. AIAA Paper 2013-0662, 2013.

[56] MJ Martin-Burgos, E Andrés-Pérez, and M Gómez. Aerodynamic shape optimization of a 3D wing via volumetric B-splines. *Eccomas 2014, Barcelona, Spain*, 2014.

[57] L. Martinelli. *Calculations of viscous flows with a multigrid method.* PhD thesis, Princeton University, 1987.

[58] M. J. Martín, E. Andrés, M. Widhalm, P. Bitrián, and C. Lozano. Non-uniform rational B-splines-based aerodynamic shape design optimization with the DLR TAU code. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 226(10):1225–1242, 2012.

[59] Dimitri J Mavriplis. Multigrid strategies for viscous flow solvers on anisotropic unstructured meshes. *Journal of Computational Physics*, 145(1):141–165, 1998.

[60] Dimitri J Mavriplis. An assessment of linear versus nonlinear multigrid methods for unstructured mesh solvers. *Journal of Computational Physics*, 175(1):302–325, 2002.

[61] Dimitri J Mavriplis. Solution of the unsteady discrete adjoint for three-dimensional problems on dynamically deforming unstructured meshes. AIAA paper 2008-727, 2008.

[62] Andrew J McCracken, Sebastian Timme, and Kenneth J Badcock. Accelerating convergence of the CFD linear frequency domain method by a preconditioned linear solver. In *6th European Congress on Computational Methods in Applied Sciences and Engineering*, 2012.

[63] Pierre Moinier. *Algorithm developments for an unstructured viscous flow solver.* PhD thesis, Oxford University, 1999.

[64] Pierre Moinier, Jens-Dominik Müller, and Michael B Giles. Edge-based multigrid and preconditioning for hybrid grids. *AIAA Journal*, 40(10):1954–1960, 2002.

[65] J-D Müller and P Cusdin. On the performance of discrete adjoint CFD codes using automatic differentiation. *International Journal for Numerical Methods in Fluids*, 47(8-9):939–945, 2005.

[66] Jens-Dominik Müller. Coarsening 3-D hybrid meshes for multigrid methods. In *Proceedings of the 9th Copper Mountain Multigrid Conference. Copper Mountain (Colorado, USA)*, 1999.

[67] Anna Naumovich, Malte Förster, and Richard Dwight. Algebraic multigrid within defect correction for the linearized Euler equations. *Numerical Linear Algebra with Applications*, 17(2-3):307–324, 2010.

[68] M. Nemec and M.J. Aftosmis. Adjoint algorithm for CAD-based shape optimization using a Cartesian method. AIAA paper 2005-4987, 2005.

[69] C Othmer. A continuous adjoint formulation for the computation of topological and surface sensitivities of ducted flows. *International Journal for Numerical Methods in Fluids*, 58(8):861–877, 2008.

[70] Carsten Othmer. Adjoint methods for car aerodynamics. *Journal of Mathematics in Industry*, 4(1):6, 2014.

[71] 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.

[72] L. Piegl and W. Tiller. *The NURBS Book*. 2nd edition. Springer, 1997.

[73] Bartosz Protas, Thomas R Bewley, and Greg Hagen. A computational framework for the regularization of adjoint analysis in multiscale PDE systems. *Journal of Computational Physics*, 195(1):49–89, 2004.

[74] J Reuther and Antony Jameson. *Control theory based airfoil design for potential flow and a finite volume discretization*. Technical Report, National Aeronautics and Space Administration, 1994.

[75] James J Reuther, Antony Jameson, Juan J Alonso, Mark J Rimlinger, and David Saunders. Constrained multipoint aerodynamic shape optimization using an adjoint formulation and parallel computers, part 1. *Journal of Aircraft*, 36(1):51–60, 1999.

[76] James J Reuther, Antony Jameson, Juan J Alonso, Mark J Rimlinger, and David Saunders. Constrained multipoint aerodynamic shape optimization using an adjoint formulation and parallel computers, part 2. *Journal of Aircraft*, 36(1):61–74, 1999.

[77] Thomas W Roberts and RC Swanson. A study of multigrid preconditioners using eigensystem analysis. AIAA paper 2005-5229, 2005.

[78] T.T. Robinson, C.G. Armstrong, H.S. Chua, C. Othmer, and T. Grahs. Sensitivity-based optimization of parameterised CAD geometries. In *8th World Congress on Structural and Multidisciplinary Optimisation*, Lisbon, 2009.

[79] T.T. Robinson, C.G. Armstrong, H.S. Chua, C. Othmer, and T. Grahs. Optimizing parameterized CAD geometries using sensitivities based on adjoint functions. *Computer-Aided Design & Applications*, 9(3):253–268, 2012.

[80] P.L. Roe. Approximate Riemann solvers, parameter vectors and difference schemes. *Journal of Computational Physics*, 43(2), 1981.

[81] Cord-Christian Rossow. Efficient computation of compressible and incompressible flows. *Journal of Computational Physics*, 220(2):879–899, 2007.

[82] Yousef Saad. *Iterative methods for sparse linear systems.* Society for Industrial and Applied Mathematics, 2003.

[83] J.A. Samareh. Survey of shape parameterization techniques for high-fidelity multidisciplinary shape optimization. *AIAA Journal*, 39(5):877–884, 2001.

[84] J.A. Samareh. Aerodynamic shape optimization based on free-form deformation. AIAA-Paper 2004-4630, 2004.

[85] Jamshid A Samareh. A survey of shape parameterization techniques. In *NASA Conference Publication*, pages 333–344. Citeseer, 1999.

[86] Shahrokh Shahpar and Stefano Caloni. Adjoint optimisation of a high pressure turbine stage for a lean-burn combustion system. *10th ETC, Lappeenranta, Finland*, 2013.

[87] Shahrokh Shahpar and Stefano Caloni. Aerodynamic optimization of high-pressure turbines for lean-burn combustion system. *Journal of Engineering for Gas Turbines and Power*, 135(5):055001, 2013.

[88] Thomas M Smith, Russell W Hooper, Curtis C Ober, and Alfred A Lorber. Intelligent nonlinear solvers for computational fluid dynamics. AIAA-CP 2006-1483, 2006.

[89] Phillipe R Spalart and Steven R Allmaras. A one equation turbulence model for aerodynamic flows. *AIAA Journal*, 94, 1992.

[90] DB Spalding. Calculation of turbulent heat transfer in cluttered spaces. In *Proceedings of the 10th international heat transfer conference, Brighton.* Society for Industrial and Applied Mathematics, 1994.

[91] RC Swanson and C-C Rossow. An efficient solver for the RANS equations and a one-equation turbulence model. *Computers & Fluids*, 42(1):13–25, 2011.

[92] RC Swanson, E Turkel, and S Yaniv. Analysis of a RK/implicit smoother for multigrid. In *Computational Fluid Dynamics 2010*, pages 409–417. Springer, 2011.

[93] RC Swanson, Eli Turkel, and C-C Rossow. Convergence acceleration of Runge–Kutta schemes for solving the Navier–Stokes equations. *Journal of Computational Physics*, 224(1):365–388, 2007.

[94] B. Van Leer. Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method. *Journal of Computational Physics*, 32:101–136, 1979.

[95] Bram Van Leer, Wen-Tzong Lee, Philip L Roe, Kenneth G Powell, and Chang-Hsien Tai. Design of optimally smoothing multistage schemes for the Euler equations. *Communications in Applied Numerical Methods*, 8(10):761–769, 1992.

[96] Gerhard Venter. Review of optimization techniques. *Encyclopedia of aerospace engineering*, 2010, Wiley Online Library.

[97] Frank M White and Isla Corfield. *Viscous fluid flow*, volume 3. McGraw-Hill New York, 1991.

[98] Shenren Xu, Wolfram Jahn, and Jens-Dominik Müller. CAD-based shape optimisation with CFD using a discrete adjoint. *International Journal for Numerical Methods in Fluids*, 74(3):153–168, 2014.

[99] Guangxu Yu, Jens-Dominik Müller, Dominic Jones, and Faidon Christakopoulos. CAD-based shape optimisation using adjoint sensitivities. *Computers & Fluids*, 46(1):512–516, 2011.

[100] Q. Zhang, J.-D. Lee, and J.-H. Wendisch. An adjoint-based optimization method for helicopter fuselage backdoor geometry. In *Thirty-Sixth European Rotorcraft Forum, Paris, France*, 2010.