Cranfield University

Alan Le Moigne

A discrete Navier-Stokes adjoint method for aerodynamic optimisation of Blended Wing-Body configurations

College of Aeronautics

PhD Thesis

Cranfield University

College of Aeronautics

PhD Thesis

Academic Year 2002-2003

Alan Le Moigne

A discrete Navier-Stokes adjoint method for aerodynamic optimisation of Blended Wing-Body configurations

Supervisor: Prof. N.Qin

December 2002

Abstract

An aerodynamic shape optimisation capability based on a discrete adjoint solver for Navier-Stokes flows is developed and applied to a Blended Wing-Body future transport aircraft. The optimisation is gradient-based and employs either directly a Sequential Quadratic Programming optimiser or a variable-fidelity optimisation method that combines low- and high-fidelity models. The shape deformations are parameterised using a Bézier-Bernstein formulation and the structured grid is automatically deformed to represent the design changes. The flow solver at the heart of this optimisation chain is a Reynolds averaged Navier-Stokes code for multiblock structured grids. It uses Osher's approximate Riemann solver for accurate shock and boundary layer capturing, an implicit temporal discretisation and the algebraic turbulence model of Baldwin-Lomax. The discrete Navier-Stokes adjoint solver based on this CFD code shares the same implicit formulation but has to calculate accurately the flow Jacobian. This implies a linearisation of the Baldwin-Lomax model. The accuracy of the resulting adjoint solver is verified through comparison with finite-difference.

The aerodynamic shape optimisation chain is applied to an aerofoil drag minimisation problem. This serves as a test case to try and reduce computing time by simplifying the fidelity of the model. The simplifications investigated include changing the convergence level of the adjoint solver, reducing the grid size and modifying the physical model of the adjoint solver independently or in the entire optimisation process. A feasible optimiser and the use of a penalty function are also tested. The variable-fidelity method proves to be the most efficient formulation so it is employed for the three-dimensional optimisations in addition to parallelisation of the flow and adjoint solvers with OpenMP. A three-dimensional Navier-Stokes optimisation of the ONERA M6 wing is presented. After describing the concept of Blended Wing-Body and the studies carried out on this aircraft, several aerodynamic optimisations are performed on this geometry with the capability developed in this thesis.

This page has been left intentionally blank.

To my mother and my late father (R.I.P.)
for supporting me for many years.

This page has been left intentionally blank.

Acknowledgements

I sincerely appreciated the guidance and helpful advice of my supervisor Prof. Ning Qin and would like to thank him for making this work possible and for introducing me to the very interesting world of aerodynamic optimisation. I am also indebted to Dr. Scott Shaw for his availability and all the help and explanations he gave me on the MERLIN code and CFD in general. I also thank the past and present members of the CFD group, in particular Dr. David Perigo and Armando Vavalle.

This work was funded by a Scholarship from Cranfield College of Aeronautics and the EU under the “Growth” Programme Contract No GARD-CT1999-0172 as part of the MOB project. All the partners in the MOB project are thanked, especially Prof. Alan Morris for his advice on optimisation and the whole Cranfield team. A special thank must go to Dr. Paul Smith from the High Performance Computing Facility of the University of Cambridge for helping me port my codes onto the IBM parallel machine. AEMDesign is also acknowledged for kindly providing the optimisation routine FFSQP employed in the present work.

Finally I would like to thank my mother for always supporting me through the difficult moments despite the distance. This goes also to my brother who had the additional quality of being able to understand CFD- and PhD-related problems, being in this position too.

This page has been left intentionally blank.

Table of contents

Abstract **v**

Acknowledgements **ix**

Table of contents **xi**

List of Figures **xv**

List of Tables **xxi**

Nomenclature **xxiii**

1 Introduction **1**

1.1 Background on aerodynamic optimisation 1
1.2 Objectives and novelty of the thesis 2
1.3 Outline of the thesis 3

2 Literature review **5**

2.1 Gradient evaluation for aerodynamic optimisation methods 6
2.1.1 Finite-difference methods 6
2.1.2 Complex variable method 8
2.1.3 Methods using Automatic Differentiation 9
2.1.4 Quasi-analytical methods 11
2.1.4.1 Direct differentiation method 11
2.1.4.2 Adjoint method 13
2.1.5 Use of the Hessian matrix 14
2.2 Other methods of optimisation 14
2.2.1 Response surfaces 15
2.2.2 Genetic algorithms 17

3 Constrained optimisation **19**

3.1 Basic concepts in optimisation 19
3.2 Sequential Quadratic Programming 23
3.3 The optimisation subroutines used in this work 24

3.3.1 The NAG subroutine E04UCF . . . 24
3.3.2 The subroutine FFSQP . . . 25
3.4 Variable-fidelity method . . . 26

4 Surface parameterisation and grid update 33
4.1 Shape representation . . . 33
4.1.1 Existing methods . . . 33
4.1.2 The Bézier-Bernstein parameterisation . . . 35
4.2 Wing representation . . . 38
4.3 Grid update . . . 40
4.3.1 Existing methods . . . 40
4.3.2 Surface grid update . . . 42
4.3.3 Volume grid update . . . 43
4.4 Grid sensitivities . . . 48

5 Fundamental equations and discretisation 53
5.1 Introduction . . . 53
5.2 The governing equations . . . 54
5.3 Primitive variables and non-dimensionalisation . . . 55
5.4 Finite volume formulation . . . 56
5.5 Time discretisation . . . 57
5.6 Explicit formulation . . . 61
5.6.1 Convective terms . . . 61
5.6.1.1 Osher's approximate Riemann solver . . . 61
5.6.1.2 Higher-order spatial accuracy . . . 63
5.6.2 Diffusive terms . . . 64
5.6.2.1 Laminar viscous fluxes . . . 64
5.6.2.2 Turbulence modelling . . . 66
5.6.3 Boundary conditions . . . 69
5.6.3.1 Inviscid wall boundary condition . . . 70
5.6.3.2 Viscous wall boundary condition . . . 70
5.6.3.3 Symmetry boundary condition . . . 71
5.6.3.4 Supersonic inflow boundary condition . . . 71
5.6.3.5 Supersonic outflow boundary condition . . . 72
5.6.3.6 Farfield boundary condition . . . 72
5.6.3.7 Interface boundary condition . . . 74
5.6.3.8 Corner points . . . 75
5.7 Implicit formulation . . . 76
5.7.1 Solution methodology . . . 77
5.7.2 Contribution from the convective terms to the Jacobian . . . 78
5.7.3 Contribution from the diffusive terms to the Jacobian . . . 81
5.7.3.1 Laminar contributions . . . 81
5.7.3.2 Turbulent contributions . . . 82

5.7.4 Implicit boundary conditions 82
5.8 Validation 83

6 Discrete adjoint solver 89
6.1 Discrete adjoint method 89
6.2 Continuous adjoint method 90
6.3 Choice between the continuous and discrete formulations 93
6.4 Solution methodology 95
6.5 Innovative content in this adjoint solver 97
6.6 Calculation of the exact RHS Jacobian 99
6.6.1 First-order inviscid components 99
6.6.1.1 Inside the domain 99
6.6.1.2 At the boundaries 101
6.6.1.3 Interface boundary 102
6.6.2 Higher-order inviscid components 102
6.6.2.1 Inside the domain 102
6.6.2.2 At the boundaries 104
6.6.2.3 Interface boundary 105
6.6.3 Viscous laminar components 105
6.6.3.1 Inside the domain 105
6.6.3.2 At the boundaries 107
6.6.3.3 Interface boundary 112
6.6.4 Turbulent components 112
6.6.4.1 Inside the domain 112
6.6.4.2 At the boundaries 115
6.6.4.3 Interface boundary 116
6.6.4.4 In the wake 117
6.7 Verification 118

7 Optimisation tests and results in two dimensions 125
7.1 Influence of approximation on the accuracy of the gradient 125
7.1.1 Influence of the levels of convergence 126
7.1.1.1 Flow solver on objective function 126
7.1.1.2 Flow solver on gradient 127
7.1.1.3 Adjoint solver on gradient 130
7.1.2 Influence of the grid size 131
7.1.3 Influence of the physical model 132
7.2 Two-dimensional optimisation using a direct SQP method 134
7.2.1 The optimisation problem 135
7.2.2 Scaling and reference optimisation 138
7.2.3 Influence of approximation on optimisation 144
7.2.3.1 Influence of the level of convergence of the adjoint solver 144
7.2.3.2 Influence of the grid size 148

7.2.3.3 Influence of the physical model of the adjoint solver . . 153
7.2.3.4 Comparison of Euler vs Navier-Stokes optimisation . . 157
7.2.4 Handling the constraints 161
7.2.4.1 FFSQP vs E04UCF 161
7.2.4.2 Penalty function vs hard constraints 163
7.3 Two-dimensional optimisation using the variable-fidelity method 172
7.3.1 Alternative methods 172
7.3.2 Variable-fidelity results 175

8 Three-dimensional optimisations 183
8.1 Parallel computing 183
8.2 Optimisation of the ONERA M6 wing 185
8.3 Optimisation of a BWB 192
8.3.1 Background about the BWB and about the present work 192
8.3.2 Preliminary work 195
8.3.3 Baseline geometry and optimisation problem 197
8.3.4 Euler optimisations of the BWB without constraint on C_m 201
8.3.5 Euler optimisation of the BWB with constraint on C_m 211
8.3.6 Navier-Stokes optimisations of the BWB on a coarse grid 222
8.4 Discussion 232

9 Conclusions 239
9.1 Summary of achievements and findings 239
9.2 Future work 240
9.3 Perspectives 241

References 243

A Linearisation of the boundary conditions 265
A.1 Calculation of $\frac{\partial \mathbf{P}_1}{\partial \mathbf{P}_2}$ 265
A.1.1 Inviscid wall boundary condition 265
A.1.2 Viscous wall boundary condition 266
A.1.3 Symmetry boundary condition 266
A.1.4 Supersonic inflow boundary condition 266
A.1.5 Supersonic outflow boundary condition 266
A.1.6 Farfield boundary condition 266
A.1.7 Interface boundary condition 267
A.2 Calculation of $\frac{\partial \mathbf{P}_0}{\partial \mathbf{P}_2}$ 267
A.3 Calculation of $\frac{\partial \mathbf{P}_0}{\partial \mathbf{P}_3}$ 268

List of Figures

1.1 Schematic diagram of the optimisation process developed in this work. . . 3

4.1 Example of grid update with 20 Bézier parameters as design variables. (Shape modified manually, not the result of an optimisation) 45
4.2 Example of grid update for parameters controlling the wing geometry. (Shape modified manually, not the result of an optimisation) 46
4.3 Same wings as in Figure 4.2 but viewed from a different angle. (Shape modified manually, not the result of an optimisation) 47
4.4 Grid sensitivity $\frac{\partial y}{\partial \beta}$ of the y coordinate with respect to the 6th Bézier parameter out of 10 that parameterise the upper surface of the RAE2822 aerofoil. 50
4.5 Grid sensitivity $\frac{\partial x}{\partial \beta}$ of the x coordinate with respect to the parameter $scale$ associated with the 15th spanwise grid section (shown in white) on the upper surface of the ONERA M6 wing. 51

5.1 A typical cell with its centre, its grid points and its metric vectors. 58
5.2 Flux at the interface between two cells. 62
5.3 The dual volume used to calculate the viscous flux at the face between cell i, j, k and cell $i + 1, j, k$. 65
5.4 Schematic diagram of halo cells (in shaded area). Left: cell numbering starting at the boundary. Right: cell numbering finishing at the boundary. 69
5.5 Schematic diagram for the interface boundary condition between two adjacent blocks for a particular choice of cell numbering. 75
5.6 Corner point at the intersection of two boundaries: cell 1,1. 76
5.7 Computational stencil for a first-order inviscid Jacobian. 80
5.8 Computational stencil for the laminar part of the Jacobian. 81
5.9 M6 wing computational grid. 84
5.10 Contours of pressure coefficient on the upper surface of the ONERA M6 wing. 85
5.11 Chordwise C_p distributions for the ONERA M6 wing. 87
5.11 Chordwise C_p distributions for the ONERA M6 wing. (Concluded) . . . 88

6.1 First-order inviscid fluxes. 100
6.2 First-order inviscid fluxes at the boundaries. 101
6.3 Higher-order inviscid fluxes. 102

6.4 Computational stencil for a higher-order inviscid Jacobian. 103
6.5 Higher-order inviscid fluxes at the boundaries. 104
6.6 Higher-order inviscid fluxes at an interface boundary with the additional halo cell. 105
6.7 Viscous laminar fluxes. 106
6.8 Viscous fluxes at the boundaries: first case. 108
6.9 Viscous fluxes at the boundaries: second case. 109
6.10 Viscous fluxes at a corner point: third case. 110
6.11 Domain of dependency of μ_t. 113
6.12 First component of the adjoint vector when the objective function is the drag coefficient for a laminar viscous flow around the NACA0012 aerofoil. 121
6.13 First component of the adjoint vector when the objective function is the drag coefficient for a fully turbulent flow around the RAE2822 aerofoil. . 122

7.1 Influence of the level of convergence of the flow solver on the value of lift and drag coefficients. Accuracy of $C_{\frac{D}{L}}$ at j orders $= \frac{\left|C_{\frac{D}{L}\ \text{12 orders}} - C_{\frac{D}{L}\ j\ \text{orders}}\right|}{\left|C_{\frac{D}{L}\ \text{12 orders}}\right|}$. . 126
7.2 Influence of the level of convergence of the flow solver on the value of the sensitivity derivatives of drag coefficient. Accuracy of k^{th} gradient at j^{th} order $=$ $\frac{\left|\frac{dC_D}{d\beta_k}_{\text{12 orders}} - \frac{dC_D}{d\beta_k}_{j\ \text{orders}}\right|}{\left|\frac{dC_D}{d\beta_k}_{\text{12 orders}}\right|}$. 128
7.3 Influence of the level of convergence of the flow solver on the value of the sensitivity derivatives of lift coefficient. Accuracy of k^{th} gradient at j^{th} order $=$ $\frac{\left|\frac{dC_L}{d\beta_k}_{\text{12 orders}} - \frac{dC_L}{d\beta_k}_{j\ \text{orders}}\right|}{\left|\frac{dC_L}{d\beta_k}_{\text{12 orders}}\right|}$. 129
7.4 Influence of the level of convergence of the adjoint solver on the value of the sensitivity derivatives of drag coefficient. Accuracy of k^{th} gradient at j^{th} order $=$ $\frac{\left|\frac{dC_D}{d\beta_k}_{\text{12 orders}} - \frac{dC_D}{d\beta_k}_{j\ \text{orders}}\right|}{\left|\frac{dC_D}{d\beta_k}_{\text{12 orders}}\right|}$. 129
7.5 Influence of the level of convergence of the adjoint solver on the value of the sensitivity derivatives of lift coefficient. Accuracy of k^{th} gradient at j^{th} order $=$ $\frac{\left|\frac{dC_L}{d\beta_k}_{\text{12 orders}} - \frac{dC_L}{d\beta_k}_{j\ \text{orders}}\right|}{\left|\frac{dC_L}{d\beta_k}_{\text{12 orders}}\right|}$. 130
7.6 Influence of the grid size on the value of the sensitivity derivatives of drag, lift and pitching moment coefficients. Ratio $= \frac{\text{sensitivity obtained on } 65\times17 \text{ grid}}{\text{sensitivity obtained on } 129\times33 \text{ grid}}$. . 131
7.7 Influence of the physical model (here turbulent adjoint but with $\mu_t = constant$) on the value of the sensitivity derivatives of drag, lift and pitching moment coefficients. Ratio $= \frac{\text{sensitivity obtained with a turbulent adjoint with } \mu_t = constant}{\text{sensitivity obtained with a turbulent adjoint}}$. 133
7.8 Influence of the physical model (here viscous laminar adjoint) on the value of the sensitivity derivatives of drag, lift and pitching moment coefficients. Ratio $= \frac{\text{sensitivity obtained with a viscous laminar adjoint}}{\text{sensitivity obtained with a turbulent adjoint}}$. 133

7.9 Baseline geometry and the CFD grid around it. 137
7.10 Evolution of different parameters during the aerofoil optimisations comparing the method with scaling that serves as reference to the method without scaling. 140
7.11 Result of the aerofoil optimisations comparing the method with scaling that serves as reference to the method without scaling. 142
7.12 Contours of pressure coefficient on the initial shape and on the optimised shape obtained by using the scaled optimisation method. 143
7.13 Evolution of different parameters during the aerofoil optimisations comparing the influence of the level of convergence of the adjoint solver. . . . 145
7.14 Result of the aerofoil optimisations comparing the influence of the level of convergence of the adjoint solver. 147
7.15 Evolution of different parameters during the aerofoil optimisations comparing the influence of the grid size. 149
7.15 Evolution of different parameters during the aerofoil optimisations comparing the influence of the grid size. (Concluded) 150
7.16 Result of the aerofoil optimisations comparing the influence of the grid size.152
7.17 Evolution of different parameters during the aerofoil optimisations comparing the influence of the physical model of the adjoint solver. 154
7.17 Evolution of different parameters during the aerofoil optimisations comparing the influence of the physical model of the adjoint solver. (Concluded)155
7.18 Result of the aerofoil optimisations comparing the influence of the physical model of the adjoint solver. 156
7.19 Evolution of different parameters during the aerofoil optimisations comparing an Euler to a Navier-Stokes optimisation. 158
7.20 Result of the aerofoil optimisations comparing an Euler to a Navier-Stokes optimisation. 159
7.21 Evolution of different parameters during the aerofoil optimisations comparing the optimisation routine FFSQP (feasible SQP) to the NAG routine E04UCF (standard SQP). 162
7.22 Result of the aerofoil optimisations comparing the optimisation routine FFSQP (feasible SQP) to the NAG routine E04UCF (standard SQP). . . . 164
7.23 Evolution of different parameters during the aerofoil optimisations using a penalty term approach with scaling. 166
7.23 Evolution of different parameters during the aerofoil optimisations using a penalty term approach with scaling. (Concluded) 167
7.24 Evolution of different parameters during the aerofoil optimisations using a penalty term approach without scaling. 169
7.24 Evolution of different parameters during the aerofoil optimisations using a penalty term approach without scaling. (Concluded) 170
7.25 Result of the aerofoil optimisations using a penalty term approach without scaling. 171

7.26 Coarse Euler grid used for the low-fidelity model in the variable-fidelity method. (to be compared to Figure 7.9) 177
7.27 Evolution of different parameters during the aerofoil optimisations comparing the variable-fidelity method to the standard SQP method. 178
7.27 Evolution of different parameters during the aerofoil optimisations comparing the variable-fidelity method to the standard SQP method. (Concluded) 179
7.28 Result of the aerofoil optimisations comparing the variable-fidelity method to the standard SQP method. 181

8.1 The two grid levels used for the optimisation of the ONERA M6 wing. The red sections of the high-fidelity grid are the master sections that are optimised and correspond to the sections of the low-fidelity grid. 186
8.2 Evolution of different parameters during the variable-fidelity optimisation of the ONERA M6 wing. 188
8.3 Contours of pressure coefficient on the upper surface of the optimised ONERA M6 wing. (to be compared to Figure 5.10). 190
8.4 Chordwise C_p distributions for the optimised ONERA M6 wing. 191
8.4 Chordwise C_p distributions for the optimised ONERA M6 wing. (Concluded) 192
8.5 Shape modification of the master sections of the ONERA M6 wing. . . . 193
8.6 Baseline BWB geometry. 197
8.7 Fine Navier-Stokes grid around the baseline geometry. 198
8.8 Contours of pressure coefficient on the upper surface of the baseline BWB geometry at the design C_L. 199
8.9 The two grid levels used for the Euler optimisations of the BWB. The red sections of the high-fidelity grid are the master sections that are optimised and correspond to the sections of the low-fidelity grid. 202
8.10 Evolution of different parameters during the Euler variable-fidelity optimisations of the BWB without any constraint on C_m. 204
8.11 Comparison of the contours of pressure coefficient on the upper surface of the initial BWB and of the Euler optimised BWB without any constraint on C_m. Euler calculations. 206
8.12 Chordwise C_p distributions for the Euler optimised BWBs without any constraint on C_m. Euler calculations. 207
8.13 Shape modification of some master sections for the Euler optimised BWBs without any constraint on C_m. 208
8.14 Spanwise twist distribution in degrees for the Euler optimised BWBs without any constraint on C_m. 209
8.15 Contours of pressure coefficient on the upper surface of the Euler optimised BWB without any constraint on C_m. Navier-Stokes calculation on the fine Navier-Stokes grid (to be compared to Figure 8.8). 210

8.16 Chordwise C_p distributions for the Euler optimised BWBs without any constraint on C_m. Navier-Stokes calculations on the fine Navier-Stokes grid. 212
8.17 Spanwise lift distribution and loading for the Euler optimised BWBs without any constraint on C_m. Navier-Stokes calculations on the fine Navier-Stokes grid. 213
8.18 Evolution of different parameters during the Euler variable-fidelity optimisation of the BWB with constraint on C_m. 214
8.19 Comparison of the contours of pressure coefficient on the upper surface of the initial BWB and of the Euler optimised BWB with constraint on C_m. Euler calculations. 215
8.20 Chordwise C_p distributions for the Euler optimised BWB with constraint on C_m. Euler calculations. 216
8.21 Shape modification of some master sections for the Euler optimised BWB with constraint on C_m. 217
8.22 Spanwise twist distribution in degrees for the Euler optimised BWB with constraint on C_m. 218
8.23 Contours of pressure coefficient on the upper surface of the Euler optimised BWB with constraint on C_m. Navier-Stokes calculation on the fine Navier-Stokes grid (to be compared to Figure 8.8). 220
8.24 Chordwise C_p distributions for the Euler optimised BWB with constraint on C_m. Navier-Stokes calculations on the fine Navier-Stokes grid. 221
8.25 Spanwise lift distribution and loading for the Euler optimised BWB with constraint on C_m. Navier-Stokes calculations on the fine Navier-Stokes grid.222
8.26 The coarse Navier-Stokes grid used by the high-fidelity model. The red sections are the master sections. 223
8.27 Evolution of different parameters during the Navier-Stokes variable-fidelity optimisations of the BWB on a coarse grid with and without constraint on C_m. 225
8.28 Comparison of the contours of pressure coefficient on the upper surface of the initial BWB and of the Navier-Stokes optimised BWB without any constraint on C_m. Coarse grid Navier-Stokes calculations. 226
8.29 Chordwise C_p distributions obtained on the coarse grid for the Navier-Stokes optimised BWBs. 227
8.30 Shape modification of some master sections for the Navier-Stokes optimised BWBs. 229
8.31 Contours of pressure coefficient on the upper surface of the Navier-Stokes optimised BWB without any constraint on C_m. Navier-Stokes calculation on the fine Navier-Stokes grid (to be compared to Figure 8.8). 230
8.32 Chordwise C_p distributions for the Navier-Stokes optimised BWBs. Navier-Stokes calculations on the fine Navier-Stokes grid. 231

This page has been left intentionally blank.

List of Tables

5.1 Non-dimensionalisation applied in MERLIN. 57
5.2 Osher's flux formulae for $\mathbf{F}_{i+1/2}$ (P-variant). ($\mathbf{Q}_L \equiv \mathbf{Q}_0$ and $\mathbf{Q}_R \equiv \mathbf{Q}_1$) . 62
5.3 Calculation of $\dfrac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{Q}_L}$. ($\mathbf{Q}_L \equiv \mathbf{Q}_0$ and $\mathbf{Q}_R \equiv \mathbf{Q}_1$) 79
5.4 Calculation of $\dfrac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{Q}_R}$. ($\mathbf{Q}_L \equiv \mathbf{Q}_0$ and $\mathbf{Q}_R \equiv \mathbf{Q}_1$) 79
5.5 Comparison of aerodynamic coefficients obtained with MERLIN and those available in the literature. 88

6.1 Comparison of sensitivity derivatives calculated by finite difference and by the adjoint method for the NACA0012 aerofoil for a laminar flow. . . 119
6.2 Comparison of sensitivity derivatives calculated by finite difference and by the adjoint method for the RAE2822 aerofoil for a fully turbulent flow. 120

7.1 Aerodynamic coefficients of the optimised aerofoils at the target C_L. . . . 139
7.2 Aerodynamic coefficients of the optimised aerofoils for the optimisations comparing the influence of the level of convergence of the adjoint solver, at the target C_L. 146
7.3 Aerodynamic coefficients of the optimised aerofoils on the 137×35 grid for the optimisations comparing the influence of the grid size, at the target C_L. 151
7.4 Aerodynamic coefficients of the optimised aerofoils for the optimisations comparing the influence of the physical model of the adjoint solver, at the target C_L. 153
7.5 Aerodynamic coefficients of the Euler and Navier-Stokes optimised aerofoils calculated from a Navier-Stokes solution on the reference grid at the target C_L. 160
7.6 Aerodynamic coefficients of the optimised aerofoils for the optimisations comparing the optimisation routine FFSQP (feasible SQP) to the NAG routine E04UCF (standard SQP), at the target C_L. 163
7.7 Aerodynamic coefficients of the optimised aerofoils for the optimisations using a penalty term approach with scaling, at the target C_L. 167
7.8 Aerodynamic coefficients of the optimised aerofoils for the optimisations using a penalty term approach without scaling, at the target C_L. 168

7.9 Aerodynamic coefficients of the optimised aerofoils for the optimisations comparing the variable-fidelity method to the standard SQP method, at the target C_L. 179

8.1 Aerodynamic coefficients of the optimised ONERA M6 wing. 189
8.2 Comparison of aerodynamic coefficients obtained for optimised ONERA M6 wings. 189
8.3 Grid sensitivity study for the baseline BWB. $M_\infty = 0.85$, $\alpha = 3.0\,°$. 195
8.4 Comparison of the performance of the three redesigned BWB geometries. 196
8.5 Aerodynamic coefficients for the baseline BWB geometry at the design C_L.199
8.6 Value of the coefficients used in the evaluation of Φ_{hi} in Equation (8.3). . 203
8.7 Inviscid aerodynamic coefficients of the Euler optimised BWBs without any constraint on C_m. 205
8.8 Navier-Stokes check on the fine Navier-Stokes grid of the Euler optimised BWBs without any constraint on C_m. 209
8.9 Inviscid aerodynamic coefficients of the Euler optimised BWB with constraint on C_m. 215
8.10 Navier-Stokes check on the fine Navier-Stokes grid of the Euler optimised BWB with constraint on C_m. 219
8.11 Value of the coefficients used in the evaluation of Φ_{hi} for the Navier-Stokes optimisations of the BWB. 223
8.12 Aerodynamic coefficients obtained on the coarse grid for the Navier-Stokes optimised BWBs. 225
8.13 Navier-Stokes check on the fine Navier-Stokes grid of the optimised BWBs obtained by a Navier-Stokes optimisation on a coarse grid. 228

Nomenclature

Roman symbols

$\mathbf{A}$	generic LHS matrix in a linear system
$\mathbf{A}$	Jacobian $= \partial\mathbf{F}^i/\partial\mathbf{Q}$
a	speed of sound
A^+	Van Driest constant in the Baldwin-Lomax model
arc	non-dimensional arc length in the volume grid update
$\mathbf{b}$	generic RHS vector in a linear system
$\mathbf{B}$	Jacobian $= \partial\mathbf{G}^i/\partial\mathbf{Q}$
$\mathbf{B}_{i,j,k}$	component of the Jacobian $= \partial\mathbf{R}_{i,j,k}/\partial\mathbf{P}_{i,j,k-1}$
$B_{k,N}$	Bernstein polynomial
$\mathbf{BB}_{i,j,k}$	component of the Jacobian $= \partial\mathbf{R}_{i,j,k}/\partial\mathbf{P}_{i,j,k-2}$
$\mathbf{C}_{i,j,k}$	component of the Jacobian $= \partial\mathbf{R}_{i,j,k}/\partial\mathbf{P}_{i,j,k}$
c	aerofoil or wing section chord
c_1	constant in the variable-fidelity method
c_2	constant in the variable-fidelity method
C_{CP}	constant in the Baldwin-Lomax model
C_D	drag coefficient
$C_{D\,fric}$	friction drag coefficient
$C_{D\,press}$	pressure drag coefficient
$C_{D\,total}$	total drag coefficient
$C_{D\,wave}$	wave drag coefficient
CFL	CFL number
C_{Kleb}	Klebanoff constant in the Baldwin-Lomax model
C_L	lift coefficient

C_m	pitching moment coefficient
C_p	pressure coefficient
C_{wake}	wake constant in the Baldwin-Lomax model
$displacement$	increment in displacement of a master section
$\mathbf{e}_k$	unit vector in the k^{th} direction of the design space
$\mathbf{E}_{i,j,k}$	component of the Jacobian $= \partial \mathbf{R}_{i,j,k} / \partial \mathbf{P}_{i,j+1,k}$
E	total energy
e	internal energy
$\mathbf{EB}_{i,j,k}$	component of the Jacobian $= \partial \mathbf{R}_{i,j,k} / \partial \mathbf{P}_{i,j+1,k-1}$
$\mathbf{EE}_{i,j,k}$	component of the Jacobian $= \partial \mathbf{R}_{i,j,k} / \partial \mathbf{P}_{i,j+2,k}$
$\mathbf{EF}_{i,j,k}$	component of the Jacobian $= \partial \mathbf{R}_{i,j,k} / \partial \mathbf{P}_{i,j+1,k+1}$
$\mathbf{F}$	flux vector
$\mathbf{F}$	flux vector in the x direction
$\mathbf{F}_{i,j,k}$	component of the Jacobian $= \partial \mathbf{R}_{i,j,k} / \partial \mathbf{P}_{i,j,k+1}$
F	function in the Baldwin-Lomax model
F	objective function
f	generic function
$\mathbf{FF}_{i,j,k}$	component of the Jacobian $= \partial \mathbf{R}_{i,j,k} / \partial \mathbf{P}_{i,j,k+2}$
f_i	NACA 4-series function
f_k	Hicks-Henne function or Wagner function
F_{Kleb}	Klebanoff function in the Baldwin-Lomax model
$flag$	parameter used in the linearisation of the farfield boundary condition
F_{max}	maximum value of the function F across the boundary layer in the Baldwin-Lomax model
F_{wake}	function in the Baldwin-Lomax model
$\mathbf{G}$	flux vector in the y direction
g	generic function
g_i	inequality constraint
g_j	hat function
$\mathbf{H}$	Hessian matrix or its inverse
$\mathbf{H}$	flux vector in the z direction
h	generic function

h_j	equality constraint
$\mathbf{I}$	indentity matrix
I	objective function in the continuous adjoint method
in	maximum number of grid points in the i direction
j_{maxF}	j position where the function F is equal to F_{max} in the Baldwin-Lomax model
j_{maxU}	j position where the magnitude of velocity is maximum in the Baldwin-Lomax model
j_{minU}	j position where the magnitude of velocity is minimum in the Baldwin-Lomax model
jn	maximum number of grid points in the j direction
k	kinetic energy
K_{ij}	spring stiffness in the tension-spring analogy method
kn	maximum number of grid points in the k direction
$\mathbf{L}$	lower triangular matrix in a BILU decomposition
L	Lagrangian function
l	mixing length in the Baldwin-Lomax model
L/D	lift to drag ratio
L_l	distance between two successive grid points in the volume grid update
M_∞	freestream Mach number
$\mathbf{n}$	normal unit vector
$\mathbf{N}_{i,j,k}$	component of the Jacobian $= \partial\mathbf{R}_{i,j,k}/\partial\mathbf{P}_{i+1,j,k}$
$N_{k,l}$	B-spline basis function of order l
$\mathbf{NB}_{i,j,k}$	component of the Jacobian $= \partial\mathbf{R}_{i,j,k}/\partial\mathbf{P}_{i+1,j,k-1}$
NCON	number of aerodynamic constraints
NDV	number of design variables
$\mathbf{NE}_{i,j,k}$	component of the Jacobian $= \partial\mathbf{R}_{i,j,k}/\partial\mathbf{P}_{i+1,j+1,k}$
$\mathbf{NF}_{i,j,k}$	component of the Jacobian $= \partial\mathbf{R}_{i,j,k}/\partial\mathbf{P}_{i+1,j,k+1}$
$\mathbf{NN}_{i,j,k}$	component of the Jacobian $= \partial\mathbf{R}_{i,j,k}/\partial\mathbf{P}_{i+2,j,k}$
$\mathbf{NW}_{i,j,k}$	component of the Jacobian $= \partial\mathbf{R}_{i,j,k}/\partial\mathbf{P}_{i+1,j-1,k}$
$\mathbf{P}$	vector of primitive variables $= \begin{pmatrix} \rho & u & v & w & p \end{pmatrix}^t$
$\mathbf{P}_k$	vector of Bézier control points

$\overline{\mathbf{P}}$	vector of primitive variables with the velocity calculated in the body fitted transformed coordinates $= (\ \rho\ \ \overline{U}\ \ \overline{V}\ \ \overline{W}\ \ p\ )^t$
p	static pressure
$p^\star$	target pressure in the continuous adjoint method
P_k	orthonormalised polynomial
Pr	Prandtl number, taken equal to 0.7
$\mathbf{Q}$	vector of conservative variables $= \begin{pmatrix} \rho & \rho u & \rho v & \rho w & \rho E \end{pmatrix}^t$
$\mathbf{q}$	heat flux vector
$\mathbf{R}$	residual vector
R	perfect gas constant, for air $= 287$ J/kg K
r	parameter measuring the performance of the low-fidelity model in the variable-fidelity method
R, R_+, R_-	Riemann invariants
r_1	constant in the variable-fidelity method
r_2	constant in the variable-fidelity method
Re	Reynolds number
$refpoint$	non-dimensionalised chordwise position of a reference point for a master section
$\mathbf{rhs}_{i,j,k}$	vector containing the contribution of the RHS product $\left[\frac{\partial \mathbf{R}(\mathbf{Q}^*)}{\partial \mathbf{P}}\right]^t \boldsymbol{\lambda}^n$ at the cell i, j, k in the discrete adjoint method
r_p	penalty coefficient
R_{total}	total residual
$\mathbf{S}$	vector of search direction
$\mathbf{s}$	vector of variables in the low-fidelity optimisation of the variable-fidelity method
$\mathbf{S}_2$	vector of coordinates for a 2-D curve
$\mathbf{S}_{i,j,k}$	component of the Jacobian $= \partial \mathbf{R}_{i,j,k} / \partial \mathbf{P}_{i-1,j,k}$
S	area
s	entropy
s_i	slope limiter in the MUSCL scheme
S_{ij}	strain-rate
$\mathbf{SB}_{i,j,k}$	component of the Jacobian $= \partial \mathbf{R}_{i,j,k} / \partial \mathbf{P}_{i-1,j,k-1}$

$scale$	increment in scaling of a master section
$\mathbf{SE}_{i,j,k}$	component of the Jacobian $= \partial\mathbf{R}_{i,j,k}/\partial\mathbf{P}_{i-1,j+1,k}$
$\mathbf{SF}_{i,j,k}$	component of the Jacobian $= \partial\mathbf{R}_{i,j,k}/\partial\mathbf{P}_{i-1,j,k+1}$
$\mathbf{SS}_{i,j,k}$	component of the Jacobian $= \partial\mathbf{R}_{i,j,k}/\partial\mathbf{P}_{i-2,j,k}$
$\mathbf{SW}_{i,j,k}$	component of the Jacobian $= \partial\mathbf{R}_{i,j,k}/\partial\mathbf{P}_{i-1,j-1,k}$
T	temperature
t	time
$twist$	increment in twist of a master section
$twistcoeff_i$	coefficients in the spanwise twist distribution function
$\mathbf{U}$	upper triangular matrix in a BILU decomposition
$\overline{U}$	component of the velocity normal to the boundary or interface
u	component of velocity in the x direction
u	normalised computational arclength along a curve
u_τ	friction velocity
U_{diff}	maximum difference in velocity amplitude across the boundary layer in the Baldwin-Lomax model
$\mathbf{V}$	velocity vector $= (u, v)$ in 2-D, $= (u, v, w)$ in 3-D
$\overline{V}$	component of the velocity parallel to the boundary
V	volume
v	component of velocity in the y direction
$\mathbf{W}_{i,j,k}$	component of the Jacobian $= \partial\mathbf{R}_{i,j,k}/\partial\mathbf{P}_{i,j-1,k}$
$\overline{W}$	component of the velocity parallel to the boundary
w	component of velocity in the z direction
$\mathbf{WB}_{i,j,k}$	component of the Jacobian $= \partial\mathbf{R}_{i,j,k}/\partial\mathbf{P}_{i,j-1,k-1}$
$\mathbf{WF}_{i,j,k}$	component of the Jacobian $= \partial\mathbf{R}_{i,j,k}/\partial\mathbf{P}_{i,j-1,k+1}$
W_k	weight coefficient in the definition of a NURBS
$\mathbf{WW}_{i,j,k}$	component of the Jacobian $= \partial\mathbf{R}_{i,j,k}/\partial\mathbf{P}_{i,j-2,k}$
$\mathbf{X}$	vector of grid points
$\mathbf{x}$	generic LHS vector of unknown in a linear system
x, y, z	Cartesian coordinates
X_k	x coordinate of a control point
$xref, yref$	coordinates of the reference point of a master section

y^+	non-dimensional normal distance
Y_k	y coordinate of a control point
y_{max}	value of y_n for which $F(y_n) = F_{max}$ in the Baldwin-Lomax model
y_n	normal distance from the body surface in the Baldwin-Lomax model

Greek symbols

α	length of the step taken in the search direction
α	Clauser constant in the Baldwin-Lomax model
α	angle of incidence
$\boldsymbol{\beta}$	vector of design variables
$\mathcal{B}$	coefficient in the beta-correction technique of the variable-fidelity method
$\mathcal{B}_q$	linear approximation of $\mathcal{B}$
Γ	surface
γ	ratio of specific heats, for air $= 1.4$
$\bar{\delta}$	coefficient in the Sequential Quadratic Programming method
Δ	increment
δ_i	coefficients in the Sequential Quadratic Programming method
δ_{ij}	Kronecker symbol
Δ_{max}	upper bound on the radius of the trust-region in the variable-fidelity method
Δ_q	bound on the radius of the trust-region at iteration q in the variable-fidelity method
Δx_i	node displacement in the tension-spring analogy method
δy	shape increment in the y direction
ε	small incremental step or small number
$\boldsymbol{\zeta}$	metric vector / cell face normal vector in the k direction
$\boldsymbol{\eta}$	metric vector / cell face normal vector in the j direction
η	non-dimensional spanwise position
κ	parameter in the MUSCL scheme
κ	thermal conductivity coefficient
κ	von Kármán constant in the Baldwin-Lomax model
$\boldsymbol{\lambda}$	adjoint vector $= \begin{pmatrix} \lambda_1 & \lambda_2 & \lambda_3 & \lambda_4 & \lambda_5 \end{pmatrix}^t$ in 3-D
λ	second coefficient of viscosity

μ	Lagrange multiplier on the surface boundary in the continuous adjoint method
μ	molecular viscosity
μ_i	coefficients in the Sequential Quadratic Programming method
μ_j	penalty parameter used in FFSQP
μ_t	eddy or turbulent viscosity
$\boldsymbol{\xi}$	metric vector / cell face normal vector in the i direction
ρ	fluid density
$\boldsymbol{\tau}$	stress tensor
τ	shear stress
Φ	merit function composed of the objective function augmented with a penalty term
Ω	bounded domain
ω	vorticity in the Baldwin-Lomax model
$\partial\Omega$	surface boundary of domain Ω

Subscripts

b	on the boundary
e	extrapolated from inside the domain
hi	high-fidelity model
∞	freestream
i	index
j	index
k	index
L	relative to the left hand side of the interface in the Riemann solver
LE	leading edge
lo	low-fidelity model
R	relative to the right hand side of the interface in the Riemann solver
$surface$	on the surface of the aerofoil or wing
w	at the wall
x	$= \partial/\partial x$
y	$= \partial/\partial y$

Superscripts

$*$	converged flow solution
$*$	non-dimensional flow variable
$\sim$	approximate Jacobian
$\sim$	indicates a corrected model in the variable-fidelity method
$\sim$	small increment in the continuous adjoint method
i	convective or inviscid
l	lower bound
n	iteration number (relative to time) for the flow and adjoint solvers
q	optimisation iteration number
t	transpose operator
u	upper bound
v	diffusive or viscous

Acronyms

BFGS	Broyden-Fletcher-Goldfarb-Shanno
BILU	Block Incomplete Lower-Upper
BWB	Blended Wing-Body
CAD	Computer Aided Design
CDE	Computational Design Engine
CFD	Computational Fluid Dynamics
CFL	Courant-Friedrichs-Lewy
CPU	Central Processing Unit
DNS	Direct Numerical Simulation
LES	Large Eddy Simulation
LHS	Left Hand Side
MDO	Multidisciplinary Design and Optimisation
MOB	Multidisciplinary design and Optimisation for Blended wing-body configuration
MPI	Message Passing Interface
NURBS	Non-Uniform Rational B-Spline
PVM	Parallel Virtual Machine

RHS	Right Hand Side
SQP	Sequential Quadratic Programming

Chapter 1

Introduction

1.1 Background on aerodynamic optimisation

The aerospace industry has integrated Computational Fluid Dynamics (CFD) within the design process of aircraft and wings for about three decades. Within the past decade or so CFD has become an indispensable tool that is taking increasing importance alongside costly wind tunnel experiments. The advantage of CFD is that it offers several levels of approximation of the equations of fluid motion that are suited for different stages of the design process:[1,2] fast and simple tools such as the panel method are used for the conceptual design; medium fidelity codes, solving the full potential or the Euler equations for example, can be employed in the preliminary design phase; while high-fidelity and computationally expensive codes such as Reynolds Averaged Navier-Stokes solvers are kept for the detailed design and analysis.

Aerodynamic optimisation concerns automating parts of this well-established aerodynamic design process. Its aim is to help the aerodynamicist who, up to now, was doing repetitive tasks such as calculating the flow around a geometry to assess its performance, modifying this geometry to get improved performance, assessing the new geometry by another flow solution to see if the design has been improved and so on until the best possible shape within the available resources (man power, computing time, flow solver accuracy) is found. This can be called an iterative analysis design process. Aerodynamic optimisation is the automation of such a task by coupling the CFD solver to an optimiser that seeks the best possible shape using the information it receives from the flow solver.

Aerodynamic optimisation can be performed at several stages of the design process. Cheap aerodynamic optimisation methods involving panel codes for example are used in the conceptual phase, generally associated to other disciplines to form a Multidisciplinary Design and Optimisation (MDO) capability.[3] More expensive higher-fidelity aerodynamic optimisation methods are employed for the detailed design, generally on their own as part of a pure aerodynamic exercise. One of the aims of this thesis is to develop such a high-fidelity method.

Aerodynamic optimisation started more or less at the same time as CFD but, at that time in the late 1970s - early 1980s, the methodology employed and the computing power available made it impossible to use in practice. At the end of the 1980s Jameson[4] developed a new methodology i.e. the adjoint method, that greatly reduced the computing time required. This started a growing interest that is culminating today with the introduction of aerodynamic optimisation based on the adjoint method into the industrial environment. In References [5,6], aerodynamic optimisation is employed in the design of a racing aircraft. It is used for the design of a regional jet aircraft in Reference [7] and of several transonic wings in Reference [6]. Finally supersonic transport aircraft are also designed with such methods.[8,9]

At the beginning of this work, every aerospace company in Europe had its own adjoint method for aerodynamic optimisation or was in the process of developing it: in no particular order, Rolls-Royce, QinetiQ and BAE Systems were supporting some work done at Oxford University on a three-dimensional optimisation method for Navier-Stokes flows based on a discrete adjoint method;[10,11] BAE Airbus UK[12,13] and Dassault Aviation[14,15] both had a three-dimensional continuous adjoint solver for a viscous/inviscid interaction method; Airbus France was developing a three dimensional Euler discrete adjoint capability;[16] NLR had a continuous adjoint method for two-dimensional Navier-Stokes optimisation;[17] Saab Aerospace had a three-dimensional inviscid continuous adjoint solver.[18] This proves the industrial relevance of such shape optimisation methods as well as the relative novelty of the present work aimed at developing a three-dimensional adjoint capability for Navier-Stokes flows that few possess. The next section gives more details about the goals of the present work.

1.2 Objectives and novelty of the thesis

This thesis has two objectives: to develop an aerodynamic optimisation method based on a discrete adjoint solver for three-dimensional Navier-Stokes flows and to apply this capability to the optimisation of a Blended Wing-Body (BWB) aircraft.

Within the first objective, the main task is the development of the discrete adjoint solver based on an existing three-dimensional Reynolds averaged Navier-Stokes flow solver. The novelty in this work is coming from the use of Osher's numerical scheme in the flow solver that will be employed in the adjoint code. In the flow solver this is used for accurate shock and boundary layer capturing and it is believed that employing the same technique in the adjoint solver will produce gradients that are very accurate for industrial problems involving shock waves and boundary layers. The other aspect of novelty is coming from the turbulence model employed in both the flow and adjoint solvers i.e. the algebraic model of Baldwin-Lomax. The author only knows one other reference[19] using this model for an adjoint solver. The main problem to overcome is that differentiation of such a model has to

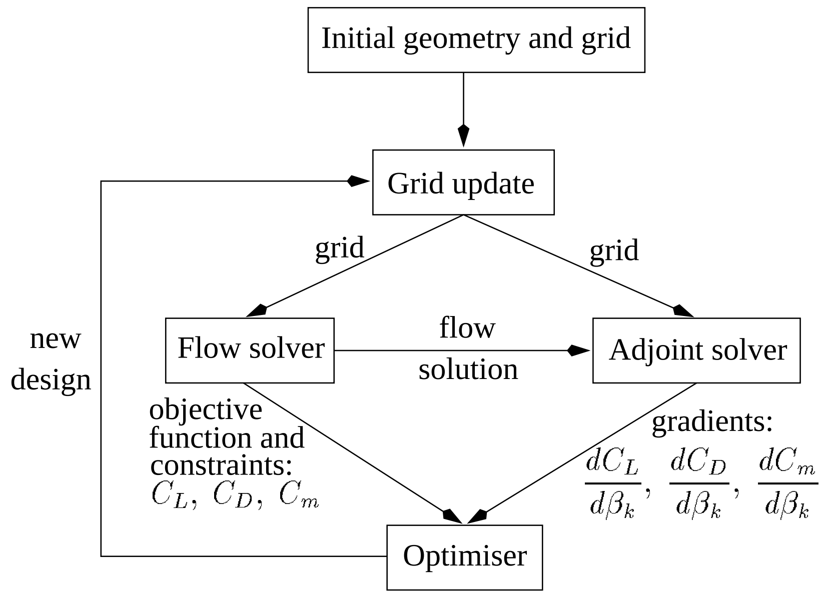

Initial geometry and grid
Grid update
grid
grid
Flow solver
flow solution
Adjoint solver
new design
objective function and constraints: C_L, C_D, C_m
gradients: $\frac{dC_L}{d\beta_k}$, $\frac{dC_D}{d\beta_k}$, $\frac{dC_m}{d\beta_k}$
Optimiser

Figure 1.1: Schematic diagram of the optimisation process developed in this work.

be accurate. Its main advantage is that this algebraic turbulence model is cheap to calculate while relatively accurate for attached wing flows and enables viscous turbulent flows to be addressed at a reduced cost compared to other turbulence models. This is very important since three-dimensional geometries in turbulent flows will be optimised and computing time is a big issue in such cases.

The other objective of this thesis is to optimise the three-dimensional shape of a BWB with this adjoint-based optimisation method. This in itself is novel: little work has been reported so far on the aerodynamic design of a BWB[20,21] and the author has no knowledge of any reference about a high-fidelity shape optimisation of such a novel aircraft.

1.3 Outline of the thesis

The optimisation process developed in this work is presented schematically in Figure 1.1. The process is iterative and is governed by the gradient-based optimiser that sits at the bottom of this diagram. The optimiser determines the design changes to make. These are fed into a grid updating program that modifies the grid around the baseline geometry according to these design changes. The new CFD grid is used by the flow solver to calculate the flow solution sent to the adjoint code and also to output the aerodynamic coefficients used by the optimiser. On the other branch, the adjoint solver calculates the gradient of these aerodynamic coefficients from the CFD grid and the flow solution and sends them to the optimiser. From this information this latter determines the new design changes to make and the loop is started again until the optimum is found. Each of the boxes in this diagram will be described in detail in one of the chapters of this thesis. It thus serves as a backbone for what is presented in Chapter 3 to Chapter 6.

The outline of the thesis is as follows: Chapter 2 presents the existing aerodynamic optimisation methods through a literature survey. This essentially describes gradient-based methods and ways to calculate the gradient. This will explain why the adjoint approach is more efficient than other non-adjoint methods and thus why it is chosen for this work. Chapter 3 gives some information about optimisation in general and then details the two SQP algorithms used as optimisers in this work. It also describes the variable-fidelity optimisation method employed for all the three-dimensional optimisations of this thesis. Chapter 4 presents the existing parameterisation techniques before concentrating on the one used here i.e. the Bézier-Bernstein parameterisation. It then explains how this is used in three dimensions. The grid deforming algorithm is then described after reviewing existing techniques. A differentiation of this algorithm leads to the calculation of the grid sensitivities needed by the adjoint solver. Chapter 5 describes the flow solver MERLIN employed in this thesis. This is a thorough description since the differentiation of this code leads to the adjoint solver presented in the next chapter, Chapter 6. This latter comes back to the description of the adjoint method by detailing the two possible approaches: the discrete or the continuous methods. It is useful to explain why the discrete approach is chosen in this work. A description of the derivation of the adjoint solver follows. Now that all the pieces of the optimisation chain have been presented, they are put together and optimisation can begin. Results for a two-dimensional aerofoil optimisation are shown in Chapter 7. This serves as a test case for the method: several parameters such as convergence levels, grid size, physical models or optimiser are changed to try and make the optimisation method as efficient as possible. Chapter 8 then presents optimisation results in three dimensions. It starts by explaining how the flow and adjoint solvers are parallelised to reduce perceived computing time. The Navier-Stokes optimisation of the ONERA M6 wing is then detailed. After giving some background information on the BWB, several optimisations on this geometry are described. Finally Chapter 9 concludes this thesis by summarising the main achievements and gives some indications for future work.

Chapter 2

Literature review

The aim of this chapter is to present the diverse methods currently used to perform aerodynamic optimisation, and to give the preliminary reasons for choosing the particular method employed in this work. The methods of aerodynamic optimisation can be globally classified depending on the type of optimisation method they are using. Be it for aerodynamic optimisation, structural optimisation or even financial optimisation, the optimiser requires some information on the objective function and its relationship with the design space in order to find the optimum design point. These requirements dictate what the rest of the optimisation process will be.

The most widely used, certainly for historical reasons because when computing performances were very limited they were the only possible choice, are gradient-based optimisation methods. These methods require, as their name indicates, the evaluation of the gradient of the objective function. This class of aerodynamic optimisation method will be described in the first part of this chapter and can also be divided into finite-difference methods, complex variable methods, Automatic Differentiation methods and quasi-analytical methods.

The aerodynamic optimisation methods that are not gradient-based include the response surface technique and genetic algorithms. These methods which will be described in the second part of this chapter, gained some popularity over the last decade because they can take full advantage of developments in computing sciences especially parallel computing.

Before starting the description of each method, it is necessary to point out that each of them will not be presented in detail, the aim of this chapter being to show that techniques other than the adjoint method exist. Likewise, the references cited in this chapter do not constitute an extensive literature review of each method, they are only references encountered by the author during this study. This is particularly true for the response surface and the genetic algorithms methods that constitute on their own a very extensive field.

2.1 Gradient evaluation for aerodynamic optimisation methods

Chapter 3 will describe in detail the mathematical formulation of a gradient-based optimiser for which the gradient or vector of sensitivity derivatives of the objective function is required i.e. the vector composed of $\frac{dF}{d\beta_k}$ where F is the aerodynamic objective function and β_k one of the design variables. Depending on the way this term $\frac{dF}{d\beta_k}$ is computed, four classes of methods can be distinguished: the finite-difference method, the complex variable method, the Automatic Differentiation method and the quasi-analytical methods. Among this latter class, two categories exist: the direct differentiation method and the adjoint method. All of these methods are described independently below.

2.1.1 Finite-difference methods

The calculation of sensitivity derivatives by a finite-difference method, also called divided differences, is the simplest and most obvious method. It comes from the definition of a derivative. Two main methods are normally used: one-sided (forward or backward) differencing where the sensitivity derivative is approximated by

$$\frac{dF}{d\beta_k} \approx \frac{F(\boldsymbol{\beta} \pm \varepsilon \mathbf{e}_k) - F(\boldsymbol{\beta})}{\pm\varepsilon}$$

and central differencing where

$$\frac{dF}{d\beta_k} \approx \frac{F(\boldsymbol{\beta} + \varepsilon \mathbf{e}_k) - F(\boldsymbol{\beta} - \varepsilon \mathbf{e}_k)}{2\varepsilon}$$

Here ε is a small incremental step and $\mathbf{e}_k$ is the kth unit vector of the design space base. It has to be noticed that the central-difference method (2^{nd} order) should be more accurate than the one-sided difference method (1^{st} order) but it is also more expensive, at least when the gradient in several directions at the same point is desired, because in the one-sided method the value $F(\boldsymbol{\beta})$ is common to all derivatives and has to be calculated only once.

Finite-difference methods are simple because they only require the calculation of the objective function at different design points and any analysis code can do that. Hence one can perform some design optimisation with existing CFD codes without modifying them. This is the main advantage of the method. Because of its simplicity, this technique was the first to be used historically in aerodynamic optimisation[22–24] and was inherited from structural optimisation. However two major drawbacks quickly appear when using finite-difference methods for aerodynamic optimisation, that might be alleviated when considering structural optimisation due to the different nature of the equations to be solved.

The first drawback is that finite-difference methods are very time-consuming. Indeed, calculating the complete gradient of the objective function F requires NDV+1 flow calculations for the forward-difference method, where NDV is the number of design variables, and $2\times$NDV when using central-differencing. It might not be a problem when using low-fidelity analysis codes such as panel method codes because flow solutions can be run in a few seconds but when using high-fidelity codes such as Reynolds averaged Navier-Stokes codes or even Euler codes, it becomes prohibitive. Indeed a complete optimisation process will require at least 20 cycles with, each time, one gradient calculation and about 5 additional flow solutions for the line search, and NDV is usually taken between 20 and 100. Hence a minimum of 520 flow solutions are needed for a forward-difference method and, when each flow solution requires an hour or more CPU time to be run, the procedure is clearly not viable.

This problem is further amplified by the fact that the flow solutions need to be well converged. Indeed as we will see shortly, the step size ε is often very small and hence, in order that the difference $F(\boldsymbol{\beta} \pm \varepsilon \mathbf{e}_k) - F(\boldsymbol{\beta})$ is of significance, both solutions at $\boldsymbol{\beta}$ and at $\boldsymbol{\beta} \pm \varepsilon \mathbf{e}_k$ must be well converged. However the computing time problem is slightly reduced by the fact that already computed solutions can be used to restart each calculation taking advantage this time of the fact that the solutions at $\boldsymbol{\beta}$ and at $\boldsymbol{\beta} \pm \varepsilon \mathbf{e}_k$ will be very close.

The second main drawback of finite-difference methods is the choice of the incremental step ε which influences the result of the sensitivity derivative. A very small incremental step faces the problem of computer round-off errors and a large step will give an erroneous value of the derivative. This problem is illustrated in References [25, 26]. It also appears that the problem of the choice of the perturbation size is linked with the choice of the convergence criterion for the residual of the flow solutions.[27–29] Of course grid refinement also has some influence on the accuracy of the finite-difference method.[27]

Despite all of these problems, references using the finite-difference method to calculate sensitivity derivatives for optimisation can be found. The following is not an exhaustive list but References [30, 31] employ the finite-difference method for low-fidelity aerodynamics with full potential and panel codes. Since these codes have a fast turnaround, the finite-difference method is appropriate. It is less so for Navier-Stokes CFD codes but References [32–37] prove that it is possible, though very expensive. An Euler optimisation using finite-differenced gradients can be found in Reference [38].

However the main use of the finite-difference method in aerodynamic optimisation is to check the accuracy of sensitivity derivatives calculated by a different method. Indeed since exact analytical values of sensitivity derivatives can only be computed for very simple cases[39] and since there is no experimental data available for sensitivity, it is the only way of checking the value of a sensitivity derivative calculated by another method. This is what is done in Chapter 6 to assess the accuracy of the adjoint solver developed in this work. There is a large number of references which compare their sensitivity derivatives with finite-difference. Some examples are given next. For quasi-analytical methods, the finite-difference method is used to assess the accuracy of the direct differentiation[40–43] solver or of the adjoint solver, be it continuous[27,44–46] or discrete.[44,47–49] The finite-difference method is also employed to assess the accuracy of codes based on Automatic Differentiation[26,29,50,51] or the complex variable method.[52]

Finite-differencing is also used to assess the computing time[50,53–55] of other methods because every other method invented to calculate sensitivity derivatives however simple or complex it is, must be able to compute the gradient of the objective function much faster than the finite-difference method. Otherwise it is of no interest because as mentioned earlier, without changing anything to already existing CFD codes, the finite-difference method could do better.

2.1.2 Complex variable method

A method that is gaining popularity to calculate sensitivity derivatives for aerodynamic optimisation is the complex variable method. It is very similar to the finite-difference method but without some of its disadvantages. The complex variable method approximates the sensitivity derivative according to

$$\frac{dF}{d\beta_k} \approx \frac{Im[F(\boldsymbol{\beta} + i\,\varepsilon \mathbf{e}_k)]}{\varepsilon}$$

where F is now a complex analytic function and $Im[\,]$ denotes the imaginary part of this function. Like the finite-difference method this is derived from a Taylor series expansion of F. The advantage of this formulation is that there is no longer any subtraction $F(\boldsymbol{\beta} \pm \varepsilon \mathbf{e}_k) - F(\boldsymbol{\beta})$ in the numerator, which is one of the problems of the finite-difference method. Hence with the complex variable method, the evaluation of the derivative does not require extremely well converged flow solutions and a very small step size ε can be chosen without losing accuracy. It was actually found that the accuracy is increasing with decreasing step size.[56]

All that is needed to use the complex variable method is to change the variables inside an existing flow solver from `REAL` to `COMPLEX` and make sure that all the functions employed work for complex variables. This should be fine for most of the code except for the minimum or maximum functions and the absolute value function that will have to be recoded. Overall this does not require major development work unlike the quasi-analytical methods.

However the complex variable method has also some disadvantages. Like the finite-difference method, it has to be repeated for every design variable to get the complete gradient. Also since the variables are now complex variables, the complex variable flow solver will require at least twice the memory and computing time as the original flow solver that worked with real variables. This makes the calculation of the gradient as (in)efficient as a central finite-difference method and two to three times slower than a one-sided finite-difference computation. Hence it is not very efficient but it should be more accurate than the finite-difference method and one does not have to worry too much about the choice of the step size. A very interesting study comparing the efficiency of the complex variable, the finite-difference and the adjoint methods against the number of design variables, can be found in Reference [57].

References [11, 58] use the complex variable method only to calculate some terms in an adjoint solver or to check the consistency of some terms during code development. References [56, 57] employ it to check the accuracy of sensitivity derivatives calculated by other methods while References [52, 59, 60] calculate sensitivity derivatives with it that are then used for optimisation.

2.1.3 Methods using Automatic Differentiation

Automatic Differentiation is another way to calculate sensitivity derivatives. We saw that the finite-difference and complex variable methods are very time-consuming and the choice of the perturbation step for the finite-difference method is not easy and can lead to poor accuracy in the calculation of sensitivity derivatives. Automatic Differentiation offers a way to obtain accurate sensitivity derivatives relatively easily although it might be at the cost of computational efficiency.

Automatic Differentiation[61, 62] calculates derivatives of the outputs of a computer program with respect to its inputs. In practice an Automatic Differentiation program is applied to another program, the result being a third program that calculates the outputs like the initial program but also the derivatives of these outputs with respect to defined inputs. This requires some modifications to the initial program prior to automatic differentiation by inserting specialised instructions to identify independent and dependent variables. Automatic Differentiation is the automatic implementation of the chain rule of differentiation which for example calculates the derivative of $f\{g[h(t)], h(t)\}$ as

$$\frac{d\ f\{g[h(t)], h(t)\}}{dt} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial h}\frac{dh}{dt} + \frac{\partial f}{\partial h}\frac{dh}{dt}$$

There exist two modes of Automatic Differentiation. The forward mode computes the differentiation starting from the input (t) to the output (f) while the reverse mode computes from the output to the input. The reverse mode is faster for a small number of outputs compared to inputs, but requires larger computing storage.

In aerodynamic optimisation, an Automatic Differentiation program widely used is ADIFOR.[63, 64] Its method of differentiation is hybrid between the forward and reverse modes and it may be applied to programs written in FORTRAN. ADIFOR calculates the product of the Jacobian that contains the derivatives, by a seed matrix instead of the

Jacobian alone. Some improvement in efficiency can be obtained since the entire Jacobian is often not required because it is multiplied in the calculations though it is possible to get it by taking the seed matrix as the identity matrix. ADJIFOR is an extension of ADIFOR that works exclusively in reverse mode and is much more efficient for a small number of outputs.[55]

The easiest way to use ADIFOR is to employ it as a black-box to differentiate existing CFD analysis codes. With little user intervention, it will provide a code that calculates accurate sensitivity derivatives. Unfortunately the differentiated code will require a lot of memory and will be very slow. In Reference [26], ADIFOR is applied to a 3D unsteady potential code and its associated grid generator while in Reference [65] it is used in the multidisciplinary optimisation of a High Speed Civil Transport aircraft. In Reference [29, 66], ADIFOR is applied as a black-box to a 3-D thin layer Navier-Stokes code plus its grid generator and although the sensitivity derivatives are as accurate as those obtained by finite-difference, their computation requires 3 times the memory needed by the finite-difference method and 2.5 times more CPU time. Reasons for the inefficiency of such a code appear, for example, when you consider a quantity Q calculated with a preconditioner P as[50, 51, 63, 66]

$$Q^{n+1} = Q^n - P^n R^n$$

The application of ADIFOR to such a formula will give

$$Q'^{\,n+1} = Q'^{\,n} - P'^{\,n} R^n - P^n R'^{\,n}$$

If it were differentiated by hand, the term $P'^{\,n} R^n$ would be discarded and although in this formulation it is eliminated at convergence when $R^n \to 0$, it has nevertheless to be calculated, which may require a lot of additional memory and CPU time. This shows that a simple application of ADIFOR as a black-box is not viable and some user intervention is needed, in this case to set $P'^{\,n} = 0$. Other intervention might be needed to restore code vectorisation as in the initial program[67] because the "`DO` loops" introduced by ADIFOR do not generally vectorise well.

Another way of using Automatic Differentiation is to employ it as a tool to calculate derivatives needed in the quasi-analytical methods presented in the next subsection, but the main sensitivity derivatives are calculated by a quasi-analytical method. This keeps the efficiency of quasi-analytical methods while relieving the burden of calculating by hand complicated derivatives. This is for example used in a direct differentiation formulation to calculate the terms $\dfrac{\partial \mathbf{R}}{\partial \mathbf{Q}}$, $\dfrac{\partial \mathbf{R}}{\partial \mathbf{X}}$ and $\dfrac{\partial \mathbf{R}}{\partial \beta_k}$ that appear in equation (2.5).[28, 51, 68, 69] Automatic Differentiation is also employed in the direct differentiation method of the SAADO (Simultaneous Aerodynamic Analysis and Design Optimization)[70–72] or the similar SASDO (Simultaneous Aerodynamic and Structural Design Optimization)[73] suites of codes. This technique is much more efficient than the application of Automatic Differentiation as a black-box[74] and generally the computing time and memory required are situated between hand-differentiated quasi-analytical methods and finite-difference methods.[51] However it requires that a subroutine calculating the residual is clearly defined in the computer program, which might not always be the case.

Odyssée[75] is another Automatic Differentiation software tool similar to ADIFOR but enabling reverse mode. It is employed in Reference [76] to calculate some derivatives used in an adjoint code. This reference shows how much human intervention, almost at the level of each line of code of the original program, is needed if the application of an Automatic Differentiation tool is to be efficient. This also shows that Automatic Differentiation can be used to build an adjoint solver, Reference [77] being another example.

2.1.4 Quasi-analytical methods

Sobieszczanski-Sobieski[78] was the first, in 1986, to launch the interest in aerodynamic sensitivity derivatives for multidisciplinary design optimisation at a time when little was done on this subject. But even at that time he had noticed that finite-difference methods already used in structural optimisation were not viable to calculate aerodynamic sensitivity derivatives and proposed to use quasi-analytical methods i.e. methods based on the differentiation of the governing equations of the flow field.

There exist two methods within the quasi-analytical methods to compute sensitivity derivatives. These are the direct differentiation formulation and the adjoint variable formulation. They will be described in the next two subsections.

2.1.4.1 Direct differentiation method

The direct differentiation formulation is one of the two quasi-analytical methods that exist to compute sensitivity derivatives. To this end the objective function is written

$$F = F(\mathbf{Q}^*(\boldsymbol{\beta}), \mathbf{X}(\boldsymbol{\beta}), \boldsymbol{\beta}) \tag{2.1}$$

where $\mathbf{Q}$ is the vector of fluid variables and the subscript * indicates that this vector is the converged value of the flow solution. $\mathbf{X}$ is the vector of grid variables and $\boldsymbol{\beta}$ the vector of design variables. F, $\mathbf{Q}$ and $\mathbf{X}$ may depend explicitly on $\boldsymbol{\beta}$ as it is written.

The differentiation of equation (2.1) provides an expression for the sensitivity derivatives:

$$\frac{dF}{d\beta_k} = \left(\frac{\partial F}{\partial \mathbf{Q}}\right)^t \frac{d\mathbf{Q}^*}{d\beta_k} + \left(\frac{\partial F}{\partial \mathbf{X}}\right)^t \frac{d\mathbf{X}}{d\beta_k} + \frac{\partial F}{\partial \beta_k} \tag{2.2}$$

In this equation, $\left(\frac{\partial F}{\partial \mathbf{Q}}\right)^t$, $\left(\frac{\partial F}{\partial \mathbf{X}}\right)^t$ and $\frac{\partial F}{\partial \beta_k}$ should be relatively easy to obtain depending on the nature of the objective function F. The term $\frac{d\mathbf{X}}{d\beta_k}$ is the vector of grid sensitivities and the way to calculate it will be discussed in Chapter 4. The only term difficult to compute is $\dfrac{d\mathbf{Q}^*}{d\beta_k}$. It is calculated as follows.

The governing equations of the flow field are written in a residual form

$$\mathbf{R}(\mathbf{Q}^*(\boldsymbol{\beta}),\mathbf{X}(\boldsymbol{\beta}),\boldsymbol{\beta}) = \mathbf{0} \tag{2.3}$$

Note that this can represent any kind of approximation of the equations of fluid motion from full potential equations to Reynolds averaged Navier-Stokes equations. This residual is then differentiated with respect to the design variables:

$$\frac{d\mathbf{R}}{d\beta_k} = \frac{\partial\mathbf{R}}{\partial\mathbf{Q}}\frac{d\mathbf{Q}^*}{d\beta_k} + \frac{\partial\mathbf{R}}{\partial\mathbf{X}}\frac{d\mathbf{X}}{d\beta_k} + \frac{\partial\mathbf{R}}{\partial\beta_k} = \mathbf{0} \tag{2.4}$$

which is rearranged as

$$\frac{\partial\mathbf{R}}{\partial\mathbf{Q}}\frac{d\mathbf{Q}^*}{d\beta_k} = -\frac{\partial\mathbf{R}}{\partial\mathbf{X}}\frac{d\mathbf{X}}{d\beta_k} - \frac{\partial\mathbf{R}}{\partial\beta_k} \tag{2.5}$$

to provide an equation that is solved for $\dfrac{d\mathbf{Q}^*}{d\beta_k}$. This is the main equation of the direct differentiation method. Depending on the nature of the design variable β_k, either shape variable or flow field parameter such as Mach number and angle of attack, the term $\dfrac{\partial\mathbf{R}}{\partial\beta_k}$ or $\dfrac{\partial\mathbf{X}}{\partial\beta_k}$ respectively is zero and the RHS of equation (2.5) can be simplified.

The matrix $\dfrac{\partial\mathbf{R}}{\partial\mathbf{Q}}$ is the Jacobian matrix of the flow field and is theoretically the same as the one used in an implicit flow solver. This is true in theory but in practice, as we will see later, the Jacobian in a flow solver is often approximated while here for the direct differentiation method, an exact Jacobian is needed. However most of the Jacobian employed for the flow analysis can be used for the sensitivity analysis. A point to notice is that this matrix has to be calculated only once. Equation (2.5) is solved for each design variable but only its RHS has to be recalculated each time before being solved. This implies NDV solutions of equation (2.5).

Another point to notice is that equation (2.5) is a linear system of equations despite the usual non-linearity of the flow equation. If a direct inversion of the Jacobian is possible, this means that equation (2.5) can be solved very quickly. However this will only happen for simple two-dimensional Euler problems. For problems involving the Navier-Stokes equations, either for laminar or turbulent flows, or for three-dimensional problems, the LHS Jacobian is likely to be too complicated and require too much memory to be inverted directly and an iterative solution process has to be adopted. In this case the solution of equation (2.5) requires the same effort as the solution of the flow equations. Finally equation (2.5) must treat consistently the boundary conditions if accurate sensitivity derivatives are to be calculated.

If the flow field residual in equation (2.3) is differentiated before it is discretised, the direct differentiation method is called a continuous approach. If on the contrary it is differentiated after discretisation, the method is called a discrete formulation. In practice the discrete direct differentiation method is preferred.

Historically the direct differentiation method started at the beginning of the 90's following Sobieszczanski-Sobieski's appeal[78] and provided a counterpart to Jameson's continuous adjoint method. It then evolved towards the discrete adjoint method that is based on the same principles but that is much more efficient for aerodynamic shape optimisation. It is nowadays less employed for optimisation except maybe for optimisations combining the aerodynamic and structural disciplines where the number of constraints is important.

On unstructured grids, its use is limited to the Euler equations[79–83] while on structured grids, it is employed more widely. It starts with its application to simple transonic small perturbation[84] or full potential[85] codes. Numerous references use the direct differentiation method for the Euler equations on structured grids either by hand-differentiating the method[40, 42, 54, 74, 86–90] or by using Automatic Differentiation for some of the terms[68–73] as mentioned in the previous section. References [53, 91–95] provides some examples of its use for viscous laminar flows while References [41,49,96] employ it for turbulent flows but neglect the linearisation of the turbulent viscosity. An accurate treatment of the turbulent viscosity is only provided in References [19, 43, 47] and with the help of Automatic Differentiation in Reference [28].

2.1.4.2 Adjoint method

The adjoint method, which is the technique employed in this study, will be described in full detail in Chapter 6. This method is very similar to the direct differentiation method presented in the previous subsection: a linear system of equations equivalent in complexity to equation (2.5) has to be solved as well. The main difference is that this system has to be solved only NCON+1 times where NCON is the number of aerodynamics constraints. Hence if NDV>NCON+1, the adjoint variable method is more efficient than the direct differentiation method. In aerodynamic optimisation there is at most a handful of aerodynamic constraints i.e. constraints involving aerodynamic quantities that are output from a CFD calculation such as lift and drag coefficients, while a few dozens of design variables is usually the norm. This is the reason why the adjoint formulation is employed in this work rather than any other gradient-based method.

Again if the residual in equation (2.3) is differentiated before discretisation, the adjoint method is called continuous while it is called a discrete method if the residual is discretised first and then differentiated. However unlike for the direct differentiation method, both types of adjoint methods can be found in the literature. Theoretically they should give the same results in the limit of increasing grid resolution and reasons why the discrete method has been chosen in this study will be described in Chapter 6.

2.1.5 Use of the Hessian matrix

This subsection very briefly details optimisation methods that employ the Hessian matrix of the objective function i.e. the matrix composed of the second derivatives $\frac{d^2F}{d\beta_k d\beta_l}$ of the objective function with respect to the design variables (these are not properly gradient-based optimisation methods but rather second-order methods). Compared to gradient-based methods, the use of the Hessian matrix offers more information to the optimiser hence the optimisation process will converge using fewer iterations.[97] However this section has shown, up to this point, that the evaluation of the gradient of the objective function is already quite difficult, hence the calculation of the Hessian matrix is even more difficult. References [51, 77] propose some techniques to calculate accurately this matrix by using combinations of finite-difference, Automatic Differentiation and quasi-analytical methods. In Reference [98] second order derivatives are also calculated using a combination of adjoint and direct differentiation methods for a heat transfer problem.

In most of the applications however, the Hessian matrix is approximated. In Reference [99] this is done for a full potential program coupled to a boundary layer code. In traditional quasi-Newton methods, the approximate Hessian matrix is built up during optimisation from an initial guess, usually the identity matrix. Anderson *et al*[52] use an initial guess closer to the real Hessian by computing accurately its diagonal with the complex variable method but this does not speed up the optimisation process as it would be expected. In fact Arian *et al*[97, 100] show that a traditional quasi-Newton method is not very efficient for aerodynamic optimisation. They derive a preconditioner that approximates the inverse of the Hessian and apply it to the gradient at the continuous level. Their method seems efficient for two-dimensional inviscid inverse design problems.

This concludes this part detailing the calculation of sensitivity derivatives for gradient-based optimisation methods in aerodynamic optimisation. It quickly presented the simple but quite costly finite-difference method as well as the similar complex variable method. One promising field is Automatic Differentiation but at the present moment quasi-analytical methods based on hand-differentiated codes appear to be the most efficient methods. This includes the direct differentiation method and the discrete and continuous adjoint formulations that will be presented later in this work. The final part of this chapter is dedicated to other optimisation methods that do not need gradient information.

2.2 Other methods of optimisation

The principal source of problems encountered with classical optimisation techniques used in aerodynamic optimisation is the calculation of sensitivity derivatives. We saw that it is not an easy task since it requires either a lot of computing time and memory or a long effort in development before an efficient method is implemented. The second problem encountered with gradient-based optimisation methods is that the optimiser is not guaranteed to find the global minimum inside the design space but only a local minimum. Either the designer acknowledges this and is satisfied with the fact that the local minimum will be a better design than the initial design point or he really wants to find the global optimum. With a classical gradient-based method he has no choice but to restart the optimisation process from different design points inside the design space and see if the process converges towards the same optimum. This will be time consuming and all the advantages of quick convergence of a gradient-based method are lost. However other optimisation methods exist to overcome these problems.

The two optimisation methods presented in this section try to avoid using aerodynamic sensitivity derivatives and are hence relatively easy to implement. Moreover they are more likely to find the global optimum and not a local optimum. The drawback is that they might require more computing time than efficient gradient-based optimisation using quasi-analytical methods. These two techniques are response surfaces and genetic algorithms.

2.2.1 Response surfaces

The idea behind response surface techniques is to model the objective function by a smooth analytical surface over the entire design space. Once the analytical expression for the surface has been found, it is easy to use classical optimisation methods to find its minimum. If the representation of the objective function is good, the minimum of the response surface should lie close to the true minimum. To construct a response surface that captures the global features of the objective function on the design space, many evaluations of the objective function are needed at different points in the domain.

The first step in creating a response surface is to sample the design space, this is called the Design of Experiment. Techniques such as the full factorial or the Latin Hypercube determine the number of sampling points and their location in the design space. The evaluation of the objective function at these points is then performed and we will return later to this point. The next step is to select the surface type. References [101–103] use a quadratic surface while References [104, 105] employ a sum of 4th order polynomials in each design direction with no cross-terms. The last step is to build the response surface which is reduced to the determination of the coefficients of the analytic expression. This can be performed by solving a least-squares problem.[102, 106]

Once the response surface has been created, standard optimisation techniques are used to find its global minimum. Since an analytic expression is known it is easy to use gradient-based optimisation even of second order. Since the optimisation should not be time consuming, several optimisations starting from different points in the design space can be performed to have confidence that the minimum found is the global minimum and not a local one. A true evaluation of the objective function is then calculated at this minimum and is compared with the value given by the response surface. Depending on the deviation between these two values, the process is either stopped or continues with the creation of another response surface around that minimum with a reduction of the design space. Hopefully, the new response surface will be more accurate and the process should converge after several cycles. Because the convergence can be slow, once the region of the global minimum has been found, standard optimisation techniques using sensitivity derivatives can be employed on a small part of the design space to find the minimum. In this case, the interest in using a response surface will have been to seek the global minimum and not a local one.

To evaluate the objective function at selected points to construct the response surface, the most obvious technique is to perform complete CFD calculations at these points.[102] This can be time-consuming and therefore the choice of the calculation points should be optimised. The natural development in this case is to use parallel computing[101, 102] to perform these calculations since they are totally independent.

When using a response surface technique, the aim is always to reduce the number of CFD calculations without degrading the accuracy of the surface, even if parallel computing is employed. In Reference [101], the number of CFD calculations is further reduced by combining low-fidelity (linear theory) and high-fidelity aerodynamics (Euler equations). In Reference [106] this is done by combining neural network and response surface methods. Chung and Alonso[103] use both the value of the objective function and its gradient at several design points to build the surface. This greatly reduces the number of sampling points but it is at the expense of calculating a gradient with an adjoint method at each of these points. Hence the only advantage of the response surface here is in seeking the global minimum but this method requires the availability of an adjoint solver.

Another technique to evaluate the objective function is to approximate the flow solution. In References [104, 105] this is done with a method called Projected Implicit Reconstruction (PIR). The starting point is to calculate a flow solution using a CFD code at the centre of the design space. Then the direct differentiation sensitivity equation (2.5) is solved at this point to find the increment in $\mathbf{Q}$ given an increment in $\boldsymbol{\beta}$. This enables the flow solution to be approximated at a new point. Another sensitivity equation is solved at this point to enable a new displacement in the design space and so on. In this way the objective function is evaluated at selected points in the design space using only one initial CFD calculation. Of course this reduction in computing cost is accompanied by a reduction in the accuracy of the response surface and also by the cost involved in solving the sensitivity equations.

This terminates this brief presentation of response surface techniques to perform aerodynamic optimisation. They seem relatively easy to implement although they might require a large number of flow analyses before converging. The next subsection details the use of genetic algorithms.

2.2.2 Genetic algorithms

Genetic algorithm techniques[8, 107–111] offer an approach to optimisation totally different from what has been presented so far. These techniques apply the principles of natural selection and genetics to optimisation. Each design is represented as an individual with its own chromosome that identifies it. Hence the first task in design optimisation using genetic algorithms is to convert each shape or design into a chromosome. The design variables previously used can be assembled as a chain to construct a chromosome. A binary coding is usually performed to obtain better results.

An initial population of say 100 individuals is created at random and each individual is assigned with a value of fitness. This requires obtaining a CFD solution for each individual to calculate the objective function and from it, the fitness value of each individual, following the principle that the designs with a low objective function (if the optimisation requires minimising the objective function) get a high fitness value while those with a large objective function get a low fitness value. From this initial population the process of reproduction begins with a selection of the parents according to their level of fitness to create another population of 100 individuals. A method called roulette wheel can be used and it ensures that individuals with a high fitness are selected with a higher probability than those with a low fitness.

In this new population, the process of crossover is carried out: first, pairs of parents are formed at random, then the parents exchange parts of their chromosome to create two new individuals. In the exchange, the chain of the chromosome is cut at a random location, identical for the same pair of parents, and both parts of the chromosome are reconnected to the other part coming from the other parent. A new population of 100 children is thus created. To ensure that characters not present in the parents appear in the new population, a process of mutation is carried out. It consists in modifying at random the chromosome of the children but with a low probability of occurrence. This population replaces the initial one and its fitness has to be evaluated before the whole process is started again. After a large number of cycles or generations, the process should converge towards an individual with very high fitness that is the optimum design. This description is however only the basis of the method. In practice, advanced selection processes, crossover and mutation operators are used to try and reduce the population size and the number of generations needed, to reduce the overall computing time of the optimisation.

The use of genetic algorithms makes it possible to find the global optimum of the objective function over the entire design space. This is attractive when there are several local optima and classical gradient-based optimisation techniques would fail.[8, 111] However, genetic algorithm methods usually necessitate a large number of CFD calculations to evaluate the fitness of the population at each generation, which might restrict their use to 2-D optimisation. However parallel computing[8, 110] may offer a solution to this problem.

This concludes this brief presentation of the use of genetic algorithms in optimisation. It is also the end of this section that described response surface and genetic algorithm principle as alternatives to gradient-based optimisation methods. This also terminates this chapter on the literature review about optimisations methods. The next chapter starts the detailed description of the optimisation chain set up in this work by looking at the optimiser.

Chapter 3

Constrained optimisation

The aim of this chapter is to present numerical optimisation and how to solve an optimisation problem. Optimisation is a very wide domain and this chapter will only concentrate on non-linear gradient-based and constrained optimisation since it is the area of interest for aerodynamic optimisation. The first part of this chapter will present optimisation in general and define some basic concepts. The second part will detail the Sequential Quadratic Programming algorithm which is the basis of the two optimisation algorithms used in this work. Finally an optimisation technique called variable-fidelity method will be presented.

3.1 Basic concepts in optimisation

The aim of this section is to present some basics concerning optimisation as it is used in multidisciplinary optimisation and aerodynamic optimisation. These are needed because they show where sensitivity derivatives, which are at the centre of this thesis, are used in optimisation and thus highlight the interest of this thesis.

A general problem of optimisation can be presented mathematically as:[112,113]

$$\begin{array}{llll} \underset{\boldsymbol{\beta}}{\text{Minimise}} & F(\boldsymbol{\beta}) & & \text{objective function} \\ \text{Subject to:} & g_i(\boldsymbol{\beta}) \leq 0 & i = 1, l & \text{inequality constraints} \\ & h_j(\boldsymbol{\beta}) = 0 & j = 1, m & \text{equality constraints} \\ & \beta_k^l \leq \beta_k \leq \beta_k^u & k = 1, NDV & \text{side constraints} \end{array} \tag{3.1}$$

where $\boldsymbol{\beta} = \left\{ \begin{array}{c} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{NDV} \end{array} \right\}$ is the vector of design variables.

The objective function $F(\boldsymbol{\beta})$ in the case of aerodynamic shape optimisation can be the drag coefficient of a wing, its lift/drag ratio (note that minimising $-F(\boldsymbol{\beta})$ is equivalent to maximising $F(\boldsymbol{\beta})$) or its gross weight in the case of multidisciplinary optimisation. The inequality constraints can represent for example the wing section area which must remain greater than a certain value to accommodate wing structure and fuel or the maximum value of the drag coefficient while the lift coefficient is optimised. The lift coefficient can be an equality constraint when you want to minimise drag at constant lift. Finally the side constraints impose direct constraints on the design variables and provide some limits to the design space.

The design variables themselves can be of different nature. In conceptual design i.e. the first stage of the design of an aircraft, they might be parameters defining the geometry of the wing planform such as root chord, tip chord, sweep angle, spanwise crank location, etc.. In aerodynamic shape optimisation, the planform geometry is generally fixed and the design variables usually are parameters used to define the shape of the wing section at different spanwise stations.

The optimisation problem defined in this way is a constrained problem since in addition to minimising $F(\boldsymbol{\beta})$, the vector of design variables must satisfy some constraints. To ease the optimisation process, it can be advantageous to transform the problem into an unconstrained one by defining a pseudo-objective function. By adding a penalty function to the objective function, the pseudo-objective function that has then to be minimised, can be defined for example by

$$\Phi(\boldsymbol{\beta}) = F(\boldsymbol{\beta}) + r_p \left[\sum_{i=1}^{l} (\max[0, g_i(\boldsymbol{\beta})])^2 + \sum_{j=1}^{m} [h_j(\boldsymbol{\beta})]^2 \right] \tag{3.2}$$

where $r_p > 0$ is a penalty parameter of usually “large” magnitude. If $\boldsymbol{\beta}$ is in the feasible region of the design space (i.e. the region where the constraints are not violated) then $\Phi = F$ and minimising Φ is minimising F. Otherwise F is greatly penalised and the minimisation of Φ should drive $\boldsymbol{\beta}$ towards the feasible domain.

There exist different methods to solve an optimisation problem but most use an iterative procedure. Starting from an initial value of the vector of design variables $\boldsymbol{\beta}$ which in the case of aerodynamic optimisation is usually called the baseline configuration, the design is updated iteratively until a minimum of F is encountered. $\boldsymbol{\beta}$ is usually updated as follows:

$$\boldsymbol{\beta}^q = \boldsymbol{\beta}^{q-1} + \alpha^q \mathbf{S}^q \tag{3.3}$$

where q is the iteration number, $\mathbf{S}^q$ is the vector of search direction in the design space and α^q is a scalar which defines the length of the step taken in this direction. This poses two problems: determining α^q and determining $\mathbf{S}^q$.

Assuming that the search direction $\mathbf{S}^q$ is known, determining α^q becomes a simple one-dimensional search, also called a line search. Indeed α^q must be found so as to sufficiently reduce F following a line defined by the search direction. For a one-dimensional search, two well-known methods are polynomial approximation and the golden section method.[112,113]

For the determination of the search direction $\mathbf{S}^q$, a number of methods are possible. Zero-order methods, based only on the evaluation of F, are simple but are not really efficient and a large number of evaluations of F may be necessary before reaching a minimum. In multidisciplinary optimisation and aerodynamic optimisation, first-order methods, based on the value of F and of its gradient ∇F, are usually preferred because they converge faster. The simplest is the method of steepest descent where $\mathbf{S}^q$ is taken as the opposite of the gradient of the objective function so that

$$\mathbf{S}^q = -\nabla F(\boldsymbol{\beta^q})$$

A more efficient method is the conjugate direction method of Fletcher and Reeves[114] which takes into account the history of the optimisation process. $\mathbf{S}^q$ is defined as

$$\mathbf{S}^q = -\nabla F(\boldsymbol{\beta^q}) + \frac{|\nabla F(\boldsymbol{\beta^q})|^2}{|\nabla F(\boldsymbol{\beta^{q-1}})|^2}\mathbf{S}^{q-1}$$

Still more efficient methods are the variable metric methods[115] also called quasi-Newton methods where

$$\mathbf{S}^q = -\mathbf{H}.\nabla F(\boldsymbol{\beta^q})$$

where $\mathbf{H}$ is a matrix. Its initial value is the identity matrix and as the optimisation process goes on, it approaches the inverse of the Hessian matrix, giving to the method super-linear convergence characteristics. Popular methods are the Davidson-Fletcher-Powell (DFP)[115,116] and the Broyden-Fletcher-Goldfarb-Shanno (BFGS)[117–120] methods.

Indeed there exists higher order methods such as second-order Newton's methods but these require the knowledge of the Hessian matrix of F. As it has already been pointed out, the gradient of F is already difficult to obtain computationally so its Hessian matrix is even more involved and time-consuming to calculate. In practice for aerodynamic optimisation, very few people try to use a second-order method as indicated in Chapter 2 and first-order optimisation methods are usually preferred.

Let us now consider the problem of optimisation specifically in the context of aerodynamic optimisation where the design variables are shape parameters and the objective function is the drag coefficient of a wing for example. Each new value of the objective function F can only be determined after a complete calculation of the flow field around the wing has been performed by a CFD code. Hence the optimisation process can be summarised as:

1. Initialisation
2. Calculation of the flow field at a given design point $\boldsymbol{\beta}$ to calculate F ($F = C_D$ in this case)
3. Calculation of the gradient of the objective function $dC_D/d\beta_k$ in order to find the search direction for the one-dimensional search

4. One-dimensional search with as little flow field calculations as possible to find an approximate minimum of C_D in that direction

5. If the process is not converged, return to 2. with calculation of the flow field at the line search minimum and repeat the loop until convergence

As we can see, the optimisation process can be computationally intensive and really efficient methods to perform both the flow field analysis (i.e. calculation of the flow field by a CFD code) and the determination of the sensitivity derivatives (i.e. calculation of the gradient of the objective function) are needed. This brief presentation explains why there is an interest in calculating efficiently the sensitivity derivatives $dF/d\beta_k$, which is part of the subject of this thesis.

Note that the optimisation procedure described in this section does not guarantee that we will find the global minimum of the objective function F on the entire feasible space. The process may converge only towards a local minimum. Two possibilities exist then: restarting the whole optimisation from different initial design points and see if the method converges towards the same optimum; or as is the case in aerodynamic optimisation, accepting the optimum found knowing that the baseline configuration is often not too far from the optimum design and that changes in the design are expected to be small and the optimum found will be better than the baseline configuration, which is what is expected after all.

First order methods to find the search direction as presented above are quite simple and easy to implement. The use of a penalty function is also an easy way to get around the problem of constraints in optimisation. However the combination of these simple methods, even if it gives the correct answer, is unlikely to be very efficient and might require a lot of evaluations of the objective function and its gradient. We have just pointed out that having a very efficient flow solver and a very efficient way of calculating the sensitivity derivatives are necessary in aerodynamic optimisation but having a very efficient optimiser that will call these programs as little as possible is also of major importance to save some computing time. Hence the author of this thesis chose to use more advanced optimisation algorithms that are supposed to be more efficient. Since these algorithms can be quite complex and difficult both to understand and implement, already available optimisation routines were used in this work.

The constraints to choose one of these algorithms are as follows:

- Non-linear objective function
- Equality and inequality constraints
- Non-linear constraints

Two algorithms satisfying these requirements were used in this work i.e. the NAG Fortran subroutine E04UCF[121,122] and the subroutine called FFSQP kindly provided by AEMDesign[123] and originally developed at the University of Maryland.[124] These two algorithms will be presented in the following sections. They are both based on a Sequential Quadratic Programming (SQP) method which is described in the next section.

3.2 Sequential Quadratic Programming

Sequential Quadratic Programming[112,113,125] is an optimisation technique where the search direction $\mathbf{S}^q$ is found by solving an optimisation subproblem with a quadratic approximation of the original objective function and a linear approximation of the constraints. The subproblem that needs to be solved is

$$\begin{array}{lll} \underset{\mathbf{S}}{\text{Minimise}} & \widetilde{F}(\mathbf{S}) = F(\boldsymbol{\beta}^q) + \nabla F(\boldsymbol{\beta}^q)^t.\mathbf{S} + \frac{1}{2}\mathbf{S}^t\mathbf{H}^q\mathbf{S} & \\ \text{Subject to:} & \nabla g_i(\boldsymbol{\beta}^q)^t.\mathbf{S} + \delta_i g_i(\boldsymbol{\beta}^q) \leq 0 & i = 1, l \\ & \nabla h_j(\boldsymbol{\beta}^q)^t.\mathbf{S} + \bar{\delta} h_j(\boldsymbol{\beta}^q) = 0 & j = 1, m \end{array} \tag{3.4}$$

with reference to the original problem (3.1). Here the design variables are the components of $\mathbf{S}$ and the optimum is the search direction $\mathbf{S}^q$. The matrix $\mathbf{H}^q$ is a positive definite matrix which is initially the identity matrix and is updated during the optimisation to approximate the Hessian matrix of the Lagrangian function of problem (3.1). The parameters δ_i and $\bar{\delta}$, both in the interval [0,1], are used to prevent the linearisation of the constraints from creating an inconsistent problem.

Contrary to problem (3.1), problem (3.4) is a well-posed problem with a quadratic objective function and linear constraints, known as a quadratic programming problem and very efficient methods exist to solve it.[112,125]

Once the search direction has been found, a line search is still needed since problem (3.4) is only an approximation of the real problem (3.1). However $\alpha^q = 1$ is a very good first estimate for this line search and a simple quadratic polynomial interpolation is usually used to find a better α^q. For this one-dimensional search, a merit function incorporating a penalty term, very similar to equation (3.2), is employed to ensure the satisfaction of the constraints

$$\Phi(\boldsymbol{\beta}^q) = F(\boldsymbol{\beta}^q) + \sum_{i=1}^{l} \mu_i \max[0, g_i(\boldsymbol{\beta}^q)] + \sum_{j=1}^{m} \mu_{l+j} |h_j(\boldsymbol{\beta}^q)|$$

where the μ_i are based on the value of the Lagrange multipliers obtained during the resolution of the approximate quadratic problem giving the search direction.

Once α^q has been found, the design is updated with equation (3.3). Before starting a new iteration, the approximation of the Hessian matrix $\mathbf{H}^q$ also needs to be updated. One of the formulae widely employed in this case is the BFGS update formula with Powell's modification,[126] that maintains the positive definiteness of $\mathbf{H}^q$.

The optimisation algorithm provided here is a very powerful tool and that is why it is at the basis of the two optimisation subroutines that are used in this work.

3.3 The optimisation subroutines used in this work

3.3.1 The NAG subroutine E04UCF

The subroutine E04UCF[121, 122] is designed to solve the non-linear programming problem of minimising a smooth non-linear objective function of n variables subject to some constraints. These constraints are lower and upper bounds on the variables, linear and non-linear inequality constraints. Equality constraints can be dealt with by setting the same lower and upper bound to inequality constraints.

This subroutine is a first-order optimisation method based on an SQP technique and hence requires the evaluation of the gradient of the objective function and of the constraints. If a subroutine calculating these gradients is not provided by the user, E04UCF will calculate them by finite-difference. Hence to save computing time, it is highly recommended to provide these gradients even if the user has no control on when they are used. For example during a line search, only evaluations of the objective function are traditionally used while in E04UCF the gradient is also employed and this cannot be changed.

The methodology behind this subroutine is very similar to what has been presented above for an SQP method. This is how E04UCF works:

1. The subroutine first determines a design point that satisfies the bounds and the linear constraints if any. During the remaining of the optimisation the design point will satisfy these bounds and linear constraints

2. It then solves a quadratic programming subproblem to find the search direction by using another NAG subroutine called E04NCF. This subroutine is based on a two-phase quadratic programming method where the first phase finds an initial feasible point by minimising the sum of infeasibilities while the second phase minimises the quadratic objective function within the feasible region.

3. A line search is carried out with an augmented Lagrangian objective function to provide a step length. The penalty part of this augmented function only involves the non-linear constraints since the linear constraints are already satisfied. The minimum found by the line search is the new design point.

4. A quasi-Newton update of the approximate Hessian matrix of the Lagrangian function is performed.

5. If the process has not converged, a new iteration is started with a return to 2.

An important point that results from this description is that the non-linear constraints are not generally satisfied during the optimisation process until an optimal design point is finally reached, unlike for the bounds and linear constraints. This is the major drawback of this subroutine because in aerodynamic optimisation, the satisfaction of the constraints is a major issue. For example geometric constraints can enforce that the upper and lower surface of an aerofoil do not cross and if this is not satisfied, this can pose a serious grid generation problem and a potential failure of the flow solver. Less critically, if the optimisation is very time consuming, which is generally the case in aerodynamic optimisation, the designer might want to stop it before an optimal design point is reached, provided that the objective function has been sufficiently reduced. In this case it is also interesting to have a feasible design at each iteration because whenever you stop the process, you can use the partially optimised design. With this subroutine this is not guaranteed but what is seen here as a major drawback could also be the strength of this subroutine. Indeed by not satisfying all the constraints, the process could converge more quickly to the optimum. Hence a compromise might have to be found and provided that the constraints are not too critical, this algorithm can be employed.

To overcome this problem of not satisfying the non-linear constraints at each iteration, the subroutine FFSQP is also employed in this work and is described in the next subsection.

3.3.2 The subroutine FFSQP

The optimisation subroutine FFSQP[124] was kindly provided by AEMDesign[123] which distributes this subroutine developed at the University of Maryland. FFSQP stands for FORTRAN Feasible Sequential Quadratic Programming hence this subroutine is based on SQP like the NAG subroutine, but also generates feasible design points at each iteration.

FFSQP is designed to minimise the maximum of a set of smooth objective functions but in this work only one objective function will be used at a time. This objective function can be subject to linear and non-linear equality constraints, to linear and non-linear inequality constraints and to upper and lower bounds on the variables. The first thing FFSQP does is to find a feasible point. The non-linear equality constraints $h_{j\,\text{nonlinear}}(\boldsymbol{\beta}) = 0$ are then turned into inequality constraints $h_{j\,\text{nonlinear}}(\boldsymbol{\beta}) \leq 0$ and the objective function is modified to reflect this transformation by

$$F_{\text{modified}}(\boldsymbol{\beta}, \boldsymbol{\mu}) = F(\boldsymbol{\beta}) - \sum_{j\ \text{nonlinear}} \mu_j h_j(\boldsymbol{\beta})$$

where the μ_j are positive penalty parameters that are iteratively adjusted. After this transformation that still forces $h_{j\,\text{nonlinear}}$ to be zero, the optimisation problem only involves non-linear inequality and linear equality and inequality constraints. At each iteration, FFSQP will generate design points that satisfy these constraints.

FFSQP requires the gradient of the objective function and of the constraints and, like the subroutine E04UCF, if the user does not provide explicit subroutines to compute these gradients, they will be calculated by finite-difference. Here again no control is allowed on when to use the gradients or just function evaluations.

A brief description of how FFSQP works follows:

1. The search direction is resulting from the successive computations of 3 search directions, each being the solution of a quadratic programming subproblem. The resulting direction is a feasible direction. FFSQP uses the subroutine QLD developed at the University of Bayreuth, Germany and provided with FFSQP, to solve this quadratic programming problem.

2. Two strategies are possible for the one-dimensional search: a line search that obliges the objective function to be reduced after each iteration called FFSQP-AL or a line search that requires a decrease within at most 4 iterations called FFSQP-NL. The minimum found by the line search is the new design point.

3. The Hessian of the Lagrangian is updated using the BFGS formula with Powell's modifications[126]

4. The μ_k are updated

5. If the process has not converged, a new iteration is started with a return to 1.

This terminates the presentation of the two SQP subroutines employed in this work. To make sure they were working correctly, they were tested on an analytical optimisation problem known as Rosen-Suzuki's problem found in Reference [125]. Of course both subroutines found the correct optimum. These two optimisers are the main optimisation routines used in this study. When applied directly to an aerodynamic optimisation problem, it was found however, as we will see in Chapter 7, that the optimisation still requires a lot of computing time despite the performance of these optimisers. Hence another way of performing optimisation was investigated. This led to what is called in this work, the variable-fidelity method that is described in the next section.

3.4 Variable-fidelity method

The variable-fidelity method has been developed by Alexandrov *et al*[127–132] from NASA Langley Research Center with an engineering approach rather than from a mathematical point of view. They start from the observation that the direct application of optimisation algorithms to high-fidelity expensive models is almost impossible due to the high cost involved in repeatedly calculating the value of the objective function and its gradient. For high-fidelity models, they have in mind a CFD calculation on a fine mesh involving the resolution of the Navier-Stokes equations, which perfectly suits the context of this thesis. Their idea is to do most of the optimisation on a low-fidelity cheap model that globally represents the behaviour of the high-fidelity model and to correct from time to time the low-fidelity model so that it better represents the high-fidelity one. In References [130, 132], they tested their method using the Euler equations on a coarse grid for the low-fidelity model and still the Euler equations but on a fine grid for the high-fidelity model. In Reference [131], they changed both the physics and the grid refinement with the Euler equations on a reasonably coarse Euler mesh for the low-fidelity model and the turbulent Navier-Stokes equations on a Navier-Stokes mesh for the high-fidelity model. They obtain up to a fivefold improvement in efficiency compared to traditional direct high-fidelity optimisation methods although this is only for a very low number of design variables.

Their approach is based on two concepts: the trust-region and the corrected low-fidelity model. The trust-region concept is similar to move limits in conventional optimisation. This means that the real high-fidelity model is successively approximated by a surrogate model and that this approximation is valid inside successive delimited regions of the design space, generally spheres. If during the previous iteration the surrogate model approximated very well the high-fidelity model, this trust-region is extended for the new iteration whereas if it performed badly, it is restricted. If the agreement between the surrogate and the real model was good but not exceptional, the radius of the trust-region is left unchanged.

The other component of the method is the corrected low-fidelity model. Let us assume that we have a high-fidelity model F_{hi} that is to be minimised and a low-fidelity model F_{lo}. At each iteration q, the corrected low-fidelity model $\widetilde{F_{lo}}$ that is used in the optimisation is required to satisfy first-order consistency with the high-fidelity model i.e.

$$\begin{aligned} \widetilde{F_{lo}}(\boldsymbol{\beta}^q) &= F_{hi}(\boldsymbol{\beta}^q) \\ \nabla \widetilde{F_{lo}}(\boldsymbol{\beta}^q) &= \nabla F_{hi}(\boldsymbol{\beta}^q) \end{aligned} \tag{3.5}$$

This ensures that the corrected low-fidelity model $\widetilde{F_{lo}}$ behaves like F_{hi} in the neighbourhood of $\boldsymbol{\beta}^q$. Alexandrov used a beta-correction technique to construct $\widetilde{F_{lo}}$ with

$$\widetilde{F_{lo}}(\boldsymbol{\beta}) = \mathcal{B}_q(\boldsymbol{\beta}) F_{lo}(\boldsymbol{\beta})$$

where $\mathcal{B}_q$ is the linear approximation

$$\mathcal{B}_q(\boldsymbol{\beta}) = \mathcal{B}(\boldsymbol{\beta}^q) + \nabla\mathcal{B}(\boldsymbol{\beta}^q)^t(\boldsymbol{\beta} - \boldsymbol{\beta}^q)$$

of $\mathcal{B}$ defined as

$$\mathcal{B}(\boldsymbol{\beta}) = \frac{F_{hi}(\boldsymbol{\beta})}{F_{lo}(\boldsymbol{\beta})}$$

around the point $\boldsymbol{\beta}^q$. It is easy to check that such a corrected model satisfies the requirements (3.5).

Let us now describe the algorithm of the method. The algorithm chosen is the one corresponding to an SQP method in the work of Alexandrov *et al.*[132] It supposes that a corrected low-fidelity model exists for the high-fidelity objective function F_{hi} but also for the constraints $g_{i\ hi}$ and $h_{j\ hi}$ of the optimisation problem (3.1). These are denoted $\widetilde{F}_{lo}$, $\widetilde{g}_{i\ lo}$ and $\widetilde{h}_{j\ lo}$ respectively. It also assumes that a global merit function for the high-fidelity constrained optimisation problem exists. Typically this is a function composed of the high-fidelity objective function associated to a penalty term for the constraints in a similar way to equation (3.2). Let us denote this merit function by

$$\Phi_{hi} = f(F_{hi}, g_{i\ hi}, h_{j\ hi})$$

A merit function for the corrected low-fidelity model is constructed in the same way

$$\widetilde{\Phi}_{lo} = f(\widetilde{F}_{lo}, \widetilde{g}_{i\ lo}, \widetilde{h}_{j\ lo})$$

Note that $\widetilde{\Phi}_{lo}$ is not a low-fidelity corrected version of Φ_{hi} despite the notation.

Starting from an initial design point $\boldsymbol{\beta}^0$, the iteration q of the variable-fidelity method is as follows:

1. Calculate $F_{hi}(\boldsymbol{\beta}^q)$, $g_{i\ hi}(\boldsymbol{\beta}^q)\ i = 1, l$, $h_{j\ hi}(\boldsymbol{\beta}^q)\ j = 1, m$ and $F_{lo}(\boldsymbol{\beta}^q)$, $g_{i\ lo}(\boldsymbol{\beta}^q)\ i = 1, l$, $h_{j\ lo}(\boldsymbol{\beta}^q)\ j = 1, m$.

2. Build the corrected low-fidelity model $\widetilde{F}_{lo}$ for the objective function around point $\boldsymbol{\beta}^q$. If they have not been already calculated, this requires the computation of $\nabla F_{hi}(\boldsymbol{\beta}^q)$ and of $\nabla F_{lo}(\boldsymbol{\beta}^q)$.

3. Calculate $\nabla g_{i\ hi}(\boldsymbol{\beta}^q)\ i = 1, l$, $\nabla h_{j\ hi}(\boldsymbol{\beta}^q)\ j = 1, m$ and $\nabla g_{i\ lo}(\boldsymbol{\beta}^q)\ i = 1, l$, $\nabla h_{j\ lo}(\boldsymbol{\beta}^q)\ j = 1, m$, if they do not already exist.

4. Solve the following simplified low-fidelity optimisation problem for s using an SQP algorithm

$$\begin{array}{lll} \underset{\mathrm{s}}{\text{Minimise}} & \widetilde{F}_{lo}(\boldsymbol{\beta}^q + \mathrm{s}) & \\ \text{Subject to:} & \widetilde{g}_{i\ lo}(\boldsymbol{\beta}^q) + \nabla\widetilde{g}_{i\ lo}(\boldsymbol{\beta}^q)^t.\mathrm{s} \leq 0 & i = 1, l \\ & \widetilde{h}_{j\ lo}(\boldsymbol{\beta}^q) + \nabla\widetilde{h}_{j\ lo}(\boldsymbol{\beta}^q)^t.\mathrm{s} = 0 & j = 1, m \\ & \beta_k^l \leq \beta_k^q + s_k \leq \beta_k^u & k = 1, NDV \\ & \|\mathrm{s}\| \leq \Delta_q & \end{array} \tag{3.6}$$

   It is a simplified problem since the constraints have been linearised. Note that due to the first-order consistency requirements, the linearised constraints are equivalent to

$$g_{i\ hi}(\boldsymbol{\beta}^q) + \nabla g_{i\ hi}(\boldsymbol{\beta}^q)^t.\mathrm{s} \leq 0$$

$$h_{j\ hi}(\boldsymbol{\beta}^q) + \nabla h_{j\ hi}(\boldsymbol{\beta}^q)^t.\mathrm{s} = 0$$

A constraint is added on the norm of s. This is the result of the trust-region method that limits the application of the corrected low-fidelity models to the neighbourhood of $\boldsymbol{\beta}^q$. Alexandrov *et al* used the L_∞ norm for the norm of s. We prefer to use the Euclidian L_2 norm that is differentiable and is hence easy to incorporate as a constraint in an SQP method.

5. When the optimum for s is found, assess the new design point at $\boldsymbol{\beta}^q + \mathrm{s}$ and the performance of the corrected low-fidelity model by computing

$$r = \frac{\Phi_{hi}(\boldsymbol{\beta}^q) - \Phi_{hi}(\boldsymbol{\beta}^q + \mathrm{s})}{\Phi_{hi}(\boldsymbol{\beta}^q) - \widetilde{\Phi}_{lo}(\boldsymbol{\beta}^q + \mathrm{s})}$$

This requires the calculation of $F_{hi}(\boldsymbol{\beta}^q + \mathrm{s})$, $g_{i\ hi}(\boldsymbol{\beta}^q + \mathrm{s})$ $i = 1, l$, $h_{j\ hi}(\boldsymbol{\beta}^q + \mathrm{s})$ $j = 1, m$ and $F_{lo}(\boldsymbol{\beta}^q + \mathrm{s})$, $g_{i\ lo}(\boldsymbol{\beta}^q + \mathrm{s})$ $i = 1, l$, $h_{j\ lo}(\boldsymbol{\beta}^q + \mathrm{s})$ $j = 1, m$.

6. Update $\boldsymbol{\beta}^q$ and Δ_q. This is the critical part of the method and our choice slightly differs from the work of Alexandrov *et al.* The update of the new design point is as follows:

```
if $\Phi_{hi}(\boldsymbol{\beta}^q + \mathrm{s}) > \Phi_{hi}(\boldsymbol{\beta}^q)$ then $\boldsymbol{\beta}^{q+1} = \boldsymbol{\beta}^q$
else $\boldsymbol{\beta}^{q+1} = \boldsymbol{\beta}^q + \mathrm{s}$
```

i.e. that if there is no improvement in the global merit function, the new design is discarded and the process has to start again from the same initial point, otherwise the new design point is kept. The update for the trust-region radius is as follows:

```
if $\Phi_{hi}(\boldsymbol{\beta}^q + \mathrm{s}) > \Phi_{hi}(\boldsymbol{\beta}^q)$ then $\Delta_{q+1} = c_1\Delta_q$
else
    if $r < r_1$ then $\Delta_{q+1} = c_1\Delta_q$
    else if $r > r_2$ then $\Delta_{q+1} = \min(c_2\|\mathrm{s}\|, \Delta_{max})$
    else $\Delta_{q+1} = \Delta_q$
```

with the following values for the constants: $r_1 = 0.1$, $r_2 = 0.75$, $c_1 = 0.5$ and $c_2 = 2.0$. This necessitates some explanations. It means that if the low-fidelity optimisation has not improved the global high-fidelity problem, then the radius of the trust-region is reduced by half and since the new design point has not been accepted, the low-fidelity optimisation has to start again from the same point with a smaller trust-region. If the initial radius of the trust-region was much too large, this process might be repeated several times but the first-order consistency requirements ensure that the corrected low-fidelity model behaves like the high-fidelity model hence very close to the initial design point, the low-fidelity optimisation is bound to find a better design point otherwise this means that the optimum has been reached.

If the low-fidelity optimisation has found a better point than the initial design then the radius of the trust-region is updated depending on the performance of the corrected low-fidelity model measured by the ratio r. If it did not approximate very well the high-fidelity model, then r is likely to be small or even negative ($< r_1$) and the radius of the trust-region is decreased for the next iteration. If it performed very well in representing the high-fidelity model, then r is likely to be high ($> r_2$) and the trust-region is expanded for the next iteration. Note that this expansion might in fact be a reduction of the trust-region. This is coming from the experience of the author and differs from the work of Alexandrov *et al* as is explained next. Typically at the beginning of the optimisation with the variable-fidelity method, the low-fidelity optimum will be found at the boundary of the trust region for $||\mathrm{s}|| \approx \Delta_q$ hence in this case the update $\Delta_{q+1} = c_2||\mathrm{s}||$ is equivalent to $\Delta_{q+1} = c_2\Delta_q$ (what Alexandrov *et al* used) and the trust-region is expanded. When the optimisation carries on and especially when it is close to the optimum, it often happens that the low-fidelity optimum is suddenly found well inside the trust region. If this happens several times and you keep expanding the trust-region, you end up after a few iterations with a very large trust-region compared to where the optimum is found and the low-fidelity optimisation might suddenly be tempted to find an optimum close to the boundary of the trust-region where the corrected low-fidelity model no longer represents at all the behaviour of the high-fidelity model. Of course the variable-fidelity method will discard this new point but the trust-region will only be reduced by half at each iteration and it will take some time to come back close to the optimum where you already were. To avoid this behaviour, the trust region here is expanded only with respect to the norm of s and thus is always kept within reasonable limits without restricting too much the search domain of the low-fidelity optimisation. Note that the size of the trust-region is also limited by Δ_{max} but this is a single constant for the whole optimisation and thus cannot evolve with the position of the optimum inside the trust-region. Finally if the corrected low-fidelity model found an improvement but did not represent particularly well the high-fidelity model ($r_1 < r < r_2$), the radius of the trust-region is left unchanged for the next iteration.

7. If the process has not converged, then start a new iteration ($q+1$) by coming back to step 2. The information calculated in step 1. should already exist for the new point from the calculation of r.

After this presentation of the variable-fidelity method, a few comments are needed. First, this method is in fact very similar to an SQP method but instead of approximating the high-fidelity model by a quadratic function, the variable-fidelity method approximates it by a corrected low-fidelity model. The quadratic approximation is only a mathematical model while the corrected low-fidelity model is supposed to contain some physical knowledge that makes it behave like the high-fidelity model and hence the variable-fidelity method should be better than an SQP method.

The other comment concerns the low-fidelity optimisation (3.6). To have a fast overall optimisation method, this low-fidelity optimisation does not need to be very accurate since it is done on an approximated model. Its aim is only to give an improvement in the high-fidelity objective function, the repetition of the iterations making this improvement grow. For this low-fidelity optimisation, both E04UCF and FFSQP were tested. It was found that the low-fidelity optimisation could fail when using either of these subroutines but this was more a problem with E04UCF. Indeed E04UCF only satisfies its non-linear constraints (here the constraint on the norm of $\mathbf{s}$) close to the optimum and if it fails before being close to it, the constraint on the norm of $\mathbf{s}$ is not satisfied. This constraint is really the core of the variable-fidelity method since when things are not going well, the only thing that the method does between iterations is to decrease the size of the trust-region until an improvement is found. If this constraint is not satisfied, the optimisation keeps looping and looping all over again without being able to go out of this bad situation and thus fails. Since FFSQP is a feasible algorithm, even when it fails, it always satisfies this contraint on the norm of $\mathbf{s}$ and thus the global optimisation can recover. Hence FFSQP was always chosen to perform the low-fidelity optimisation of the variable-fidelity results that will be presented later in this thesis.

Another point is about the update in step 6 of the algorithm. This is really the crucial part of the method. A very fine tuning of this update can really speed up the optimisation. If the update is not appropriate, the variable-fidelity method should still work but will be very slow because it will have to do a lot of iterations either because the improvement at each iteration is too small or because it will keep going back and forth if the trust-region is expanded or contracted too much at each iteration. As already explained, this update has been modified compared to the work of Alexandrov *et al*. The update relies heavily on the ratio r and the present author is not convinced that it is a very good choice. The method works as it is but some improvement in finding other performance functions might be possible. It would be better also if the update was not relying on a single function to accomodate different possible cases. The present author did not spend much time on this point but felt that since the method relies on very good ideas, it should converge much faster than it does now and this is the likely place for improvement in the method.

The final comment concerns the robustness of the method. The fact that the trust-region is updated and that the optimisation is done on the corrected low-fidelity model make the method very robust. As already explained, if the low-fidelity optimisation fails, it is highly likely that the method will recover on its own even if this takes some time. If the calculation of the high-fidelity merit function fails, the situation is awkward but since the merit function is mainly used to check the performance of a new design point, the overall optimisation might be able to accomodate and recover. The main problem is only when the calculation of the high-fidelity gradients goes wrong since it guides the low-fidelity optimisation. This can lead to the failure of the method. However this has to be contrasted with a high-fidelity SQP optimisation where every single function or gradient evaluation has to be accurate otherwise the optimiser cannot accurately compute a feasible descent direction and fails.

Recently Marduel *et al*[133] published some results of aerofoil optimisations using the variable-fidelity method of Alexandrov *et al*. They even went further by trying to improve the method and for example built higher-order correction models and a delta-correction that is no longer a multiplicative correction like the beta-correction but an additive correction. They confirm the significant computational savings brought by the variable-fidelity method compared to a standard direct optimisation method. However only the initial variable-fidelity method of Alexandrov *et al* will be used in this work.

This concludes this section presenting the variable-fidelity optimisation used in this study. It also ends this chapter where numerical optimisation in general and how it is employed in this thesis are presented. The chapter focussed on the method of Sequential Quadratic Programming and on the two algorithms based on this method that are used in this work. The next chapter details another component of the optimisation chain developed here i.e. the geometry modeler.

Chapter 4

Surface parameterisation and grid update

This chapter presents one of the key aspects of aerodynamic shape optimisation i.e. how to represent the shape to be optimised and how to link this shape to the design variables. Since the shape is evolving during the optimisation process, the CFD grid which is used to calculate the objective function and its gradient and which is based on this evolving shape, must undergo some modifications in order to follow this evolution. This process called grid update in this thesis, also depends on the way the surface has been parameterised. It is important to note that only structured grids are used in this work.

The first part of this chapter will deal with the chosen parameterisation for two-dimensional shapes after having surveyed existing possibilities. The second part will describe how the general geometry of the wing has been parameterised in the present study. The grid update will then be examined. Finally the calculation of the grid sensitivities which are needed for the adjoint method used in this thesis, will be detailed.

4.1 Shape representation

4.1.1 Existing methods

This sub-section surveys briefly the different methods that have been used in the literature for shape optimisation. We restrict ourselves here to parameters that control the shape of a 2D aerofoil section or of a wing surface in 3D. Samareh provides a very good survey of shape parameterisation techniques used in Multidisciplinary Design and Optimisation in Reference [134], part of it being also included in References [135, 136]. He identifies 8 categories: basis vector, domain element, partial differential equation, discrete (grid points), polynomial and spline, CAD-based, analytical (shape functions) and finally free-form deformation. In this section we will complement Samareh's survey by methods and references encountered by the author of this thesis. Hence not all of his 8 categories will be described and the reader is referred to his work for more details. The different methods are more or less presented here in increasing order of complexity.

The most obvious and simple choice is to consider the grid points that define the surface as design variables. More precisely the component in the y-direction of each grid point can be free to move and constitute a design variable. This immediately implies a very high number of design variables for general wing shapes but Jameson and his colleagues[137–141] as well as others employing Jameson's method[12,142] use this technique. In addition, some smoothing[137,140] is needed at each optimisation iteration to avoid getting irregular and oscillating curves or surfaces. Even if it is a possibility, techniques that can represent the geometry using much fewer parameters might nevertheless be preferred.

A simple technique is to use analytical shape functions like the NACA 4-digit series.[49,96] Extended to a three-dimensional wing, the surface can be expressed as[79]

$$y(x,z) = y_{init}(x,z) + \sum_i \sum_j \beta_{k_{i,j}} h_{i,j}(x,z)$$

with

$$h_{i,j}(x,z) = f_i(x/c) g_j(z)$$

where f_i is a NACA 4-series function and g_j a hat function for example. The design variables in this case are the $\beta_{k_{i,j}}$. The problem with these functions is that only a certain family of aerofoils can be employed for the design.

A very similar method consists in using Hicks-Henne[143] functions[19,34,36,43,45,57,79,83,144–150] or Wagner functions[34,36,144] to modify the initial or baseline shape. In two dimensions it is defined as

$$y(x,\boldsymbol{\beta}) = y_{init}(x) + \sum_{k=1}^{NDV} f_k(x)\beta_k$$

where f_k is a Hicks-Henne function or a Wagner function and β_k a design variable. This concept of adding shape functions to an initial geometry can be extended to any type of shape functions. Destarac *et al*[30] and Reneaux[31] propose to employ either analytical shape functions or a library of basic aerofoils or "aerofunctions" that are defined from an inverse design calculation and have a physical meaning for the aerodynamicist: for example, functions can be devised to move forward or backward the position of a shock wave or to move up or down the level of the pressure rooftop. Cubic patched polynomials[36] are also a possibility. If the shape is directly parameterised without adding a perturbation to the initial geometry, orthonormalised polynomials[28,151] can be used with

$$y/c = \sum_{k=1}^{NDV} P_k(x/c)\beta_k$$

The problem is that if the polynomials do not define a basis, some designs will not be possible which might pose problems for an inverse design optimisation.

The method used in this work is the Bézier-Bernstein parameterisation which will be described in detail in the next sub-section and is also employed in References [16,80,82,86,109,152–155].

Another similar method to the Bézier-Bernstein parameterisation and also inherited from Computer Aided Design (CAD) is a B-spline representation used in References [111,125, 156–158]. In two dimensions the surface is represented by

$$x(u) = \sum_{k=0}^{N} X_k N_{k,l}(u) \quad \text{and} \quad y(u) = \sum_{k=0}^{N} Y_k N_{k,l}(u)$$

where $N_{k,l}$ are the B-spline basis functions of order l and $\left\{ \begin{array}{c} X_k \\ Y_k \end{array} \right\}$ are the control points. An extensive detail of the properties of the B-spline functions and the interest of this method of representation is given by Lambert.[125] A special form of B-spline is the non-uniform rational B-spline or NURBS. Its formulation is as follows[134]

$$x(u) = \frac{\sum_{k=0}^{N} X_k W_k N_{k,l}(u)}{\sum_{k=0}^{N} W_k N_{k,l}(u)} \quad \text{and} \quad y(u) = \frac{\sum_{k=0}^{N} Y_k W_k N_{k,l}(u)}{\sum_{k=0}^{N} W_k N_{k,l}(u)}$$

where $N_{k,l}$ are still the B-spline basis functions of order l, $\left\{ \begin{array}{c} X_k \\ Y_k \end{array} \right\}$ the control points and W_k are some weight coefficients.

Samareh's survey[134] contains a category that he calls free-form deformation and that is a morphing technique inherited from computer imaging methods. This is a complicated but also very powerful technique that he applies to shape optimisation in Reference [136].

4.1.2 The Bézier-Bernstein parameterisation

The Bézier-Bernstein parameterisation is the technique which is chosen in this work. The reasons for this choice are the following:

- The Bézier-Bernstein parameterisation enables the use of a limited number of design variables to represent a geometry in a satisfactory manner
- The shape created is always regular and does not need any smoothing
- There is a wide number of obtainable shapes, so this choice does not limit the optimisation
- The mathematical representation is a little simpler than for the B-spline

The traditional Bézier-Bernstein parameterisation for a 2D curve is as follows:[16, 80, 82, 86, 109, 152–155]

$$\mathbf{S}_2(u) = \sum_{k=0}^{N} B_{k,N}(u)\mathbf{P}_k$$

where $\mathbf{S}_2(u) = \left\{ \begin{array}{c} x(u) \\ y(u) \end{array} \right\}$, the Bézier control points are $\mathbf{P}_k = \left\{ \begin{array}{c} P_x \\ P_y \end{array} \right\}_k$ and the Bernstein polynomials

$$B_{k,N}(u) = \frac{N!}{k!(N-k)!} u^k (1-u)^{N-k}$$

u is a normalised computational arclength along the curve. In the case that only the y coordinates of the surface are free to move, the design variables are the P_y.

This formulation poses two problems that are common to either Bézier-Bernstein or B-spline parameterisations. The first one is that some initial control points are needed to start the optimisation and have a baseline geometry. There are three solutions to this problem but none of them is ideal:

- The baseline geometry has been generated by a CAD program that handles Bézier or B-spline curves. In this case it is possible to get the control points directly from the CAD software. This is the ideal case, emphasised in Reference [125], because at the end of the aerodynamic optimisation, the shape can be used again by the CAD program without making any approximation. Hence there can be a lot of interactions between CAD (and other disciplines using CAD) and aerodynamic optimisation. The only problem is that a CAD definition of the geometry is not always available. An interesting discussion about the relationship between CAD and surface parameterisation is given in Reference [135].

- An inverse Bézier problem[156] has to be solved to find the initial Bézier control points knowing the baseline geometry. This usually involves the inversion of an overdetermined linear system and in any case the obtained Bézier parameterised geometry will only be an approximation of the initial geometry.

- The last solution which can handle any initial shape without any additional calculation, is to consider the Bézier-Bernstein parameterisation as a variation around this initial shape. This is not often employed with Bézier curves while it is much more common with shape functions, as described in the previous subsection. The only reference among these provided for the Bézier-Bernstein parameterisation in the previous subsection that adds a perturbation modelled by a Bézier curve, is Reference [16]. This is also the method employed in this work. The y coordinate is written:

$$y_{current} = y_{initial} + \delta y$$

where

$$\delta y = \sum_{k=0}^{N} B_{k,N}(u) P_{y\,k}$$

In this case when the P_y are zero the shape obtained is exactly the initial shape so at the start of the optimisation, all the shape design variables should be zero. This also gives some meaning to these control parameters since a negative parameter will result in a lower y than the initial geometry and vice versa for a positive parameter. However grasping the meaning of the magnitude of the parameter is more difficult since it depends on the magnitude of the original y coordinate. The optimal shape which is a composite between the initial shape and a Bézier-Bernstein curve might also be difficult to handle for further work.

The second problem raised by a Bézier-Bernstein or a B-spline parameterisation is the choice of the arclength u especially when the method of variation around an initial shape is chosen. The author of this thesis chose the same function as in Reference [16] i.e. $u = \sqrt{x}$ where x is the non-dimensionalised chordwise position of the point of ordinate $y_{initial}$ for an aerofoil section. This choice for u concentrates design changes to the leading edge region of the aerofoil or wing where x is small.

The Bézier-Bernstein parameterisation was used in this work as a 2D parameterisation for wing sections situated in an xy plane. This means that for a 3D geometry like a wing, a series of wing sections are considered as master sections and their shape is free to evolve during the optimisation. These wing sections are then linearly connected to form a 3D surface. Another approach could have been to use directly a 3D Bézier-Bernstein parameterisation of the surface.[86,109,153,154] However the approach taken here is considered more appropriate to the geometric representation of the wing as is explained in the next section.

Each aerofoil section is composed of two curves, one for the lower and one for the upper surface, each with its own set of Bézier parameters. The control parameters at both ends of each curve are kept constant at zero so that the trailing and leading edge points are kept fixed on each section. Hence the Bézier-Bernstein parameterisation is only managing the real shape of the section since twist and dihedral that could have been controlled if these points were free to move, are dealt with by the wing representation.

The aerofoil camber alone was also parameterised as an alternative to free shape deformation for the upper and lower surfaces. This is useful if you want to make minor changes to aerofoil shapes and if you want to keep the same chordwise thickness distribution. The camber line was modelled like the exterior surfaces using a Bézier-Bernstein parameterised perturbation added to the initial camber line. A closer look at this camber line deformation shows that it is exactly like that of the shape deformation of the upper or lower surface except that now only one set of Bézier parameters is needed and the deformation is applied to both the upper and lower surfaces. Hence once the free shape deformation has been implemented, it is very easy to implement a camber-only deformation.

Now that the shape of an aerofoil can be deformed in two dimensions, the next section looks at how this is incorporated in the wing parameterisation and what other degrees of freedom are added to get a deformable three-dimensional wing.

4.2 Wing representation

A complete wing parameterisation has been developed for this work. Everything that defines a wing can be modified even if all the possibilities will not be employed in this study. The method is inspired from References [153, 154]. Unlike the parameters in these references that represent true dimensions of twist angle, chord, etc., the wing parameterisation used in this work is again inspired by Reference [16] and is a variation around the original wing geometry.

As explained in the previous section, the wing is divided into master sections situated in an xy plane and these are the sections that command the geometry of the wing. For each master section, 4 parameters are defined that are:

- An increment in displacement ($displacement$)
- The non-dimensionalised chordwise position of a reference point ($refpoint$ with $0 \leq refpoint \leq 1$)
- An increment in scaling ($scale$)
- An increment in twist ($twist$ in $^\circ$)

A description of how the wing geometry modeler uses these parameters follows:

1. The initial twist around the leading edge (LE) point of each master section is removed so that all the shape modifications occurr on the true aerofoil shape.

$$\begin{aligned} x_1 &= (x_{init} - x_{LE})\ \cos(twist_{LE}) + (y_{init} - y_{LE})\ \sin(twist_{LE}) + x_{LE} \\ y_1 &= -(x_{init} - x_{LE})\ \sin(twist_{LE}) + (y_{init} - y_{LE})\ \cos(twist_{LE}) + y_{LE} \\ z_1 &= z_{init} \end{aligned}$$

2. The shape of each master section x_1, y_1, z_1 is updated as explained in section 4.1.2 to obtain new coordinates x_2, y_2, z_2.

3. Retwist all the master sections with their initial twist value.

$$\begin{aligned} x_3 &= (x_2 - x_{LE})\ \cos(twist_{LE}) - (y_2 - y_{LE})\ \sin(twist_{LE}) + x_{LE} \\ y_3 &= (x_2 - x_{LE})\ \sin(twist_{LE}) + (y_2 - y_{LE})\ \cos(twist_{LE}) + y_{LE} \\ z_3 &= z_2 \end{aligned}$$

4. The increment in displacement is applied to each master section k:

$$x_4 = x_3 + displacement_x(k)$$

and similarly for y_4, z_4. This displacement in the three coordinates controls the leading edge sweep angle, the span and the dihedral of the wing.

5. Each master section is then scaled by the increment $scale$. This is in fact a similarity transformation in the plane xy: the centre of this similarity transformation is the reference point of the section $(xref, yref)$ and its ratio is $1+scale$. The coordinates of the reference point are calculated for each section from $refpoint$.

$$x_5 = (x_4 - xref(k))\,(1 + scale(k)) + xref(k)$$

and similarly for y_5 while z_5 is unchanged and $z_5 = z_4$. This transformation controls the taper ratio of the wing and with the previous displacement, the trailing edge sweep angle.

6. A rotation of increment $twist$ and of centre the reference point of each section is then applied in the plane xy

$$\begin{aligned} x_6 &= (x_5 - xref(k))\,\cos(twist(k)) + (y_5 - yref(k))\,\sin(twist(k)) + xref(k) \\ y_6 &= -(x_5 - xref(k))\,\sin(twist(k)) + (y_5 - yref(k))\,\cos(twist(k)) + yref(k) \end{aligned} \tag{4.1}$$

and z_6 is unchanged and $z_6 = z_5$. This obviously controls the twist of each master section.

Since the master sections are linearly connected, all these transformations from 1. to 6. are enough to define a new wing geometry. Like for the shape representation, if all the parameters are set to zero, the initial wing is recovered provided it is also linearly connected. All the parameters, except $refpoint(k)$, can be used as design variables for an optimisation. Alternatively only a few of them or none of them may be used as design variables depending on what type of optimisation is carried out. In this work $refpoint(k)$ is always set to zero so all the twist changes are performed around the leading edge of the section.

If all the parameters are considered as design variables and if the number of master sections is important, this will give a lot of freedom in the optimisation but at the cost of a high number of design variables. Problems of spanwise smoothness of the geometry can also occur. A way to avoid this is to use spanwise distribution functions as proposed in References [153,154]. The design variables in this case are parameters defining these spanwise functions and to each master section depending on its spanwise position corresponds a value of these functions that can be used to perform the transformations described above.

Problems of smoothness were encountered in this work when using twist design variables for sections very close to each other in the spanwise direction. Hence a spanwise twist distribution function was implemented. A 6th order polynomial was chosen. Hence the parameters $twist(k)$ are calculated depending on the spanwise position $\eta(k)$ of the master section k according to

$$twist(k) = \sum_{i=0}^{6} twistcoeff_i \;\; \eta(k)^i$$

To have some smoothness in the twist distribution at the root of a full span wing, the first derivative of the polynomial at the root is forced to be zero. This implies that $twistcoeff_1 = 0$. Hence only 6 coefficients $twistcoeff_i$ are needed to define the spanwise twist distribution and can be used as design variables. The spanwise twist distribution was also extended to either represent a twist increment as presented so far or the real value of twist at each master section. All that is needed in this latter case is to change the value of $twist(k)$ in equation (4.1) by $twist(k) - twist_{init}(k)$ where $twist_{init}(k)$ is the value of twist for the initial geometry.

An important point to realise is that a geometry modeler that updates the wing geometry and shape as described in this and the previous sections is not needed on its own. Since to perform an aerodynamic optimisation, the geometry is taken into account only through a CFD grid, the geometry modeler can directly be applied to the grid to obtain an updated grid. This is described in the next section.

4.3 Grid update

Since the aim of aerodynamic shape optimisation is to modify the geometry of a wing or an aerofoil, this geometry will evolve during the optimisation process and new CFD calculations will be required for this new shape. Hence it is necessary to change the grid each time the geometry is modified. Different ways of performing this grid perturbation are presented next before giving more details about the actual technique used in this thesis.

4.3.1 Existing methods

Samareh also surveys grid regeneration or deformation techniques in Reference [134]. As in Section 4.1, the author of this thesis presents here references that he encountered during this study to complement Samareh's survey.

The simplest method is to regenerate the whole grid[28,42] with a grid generator. In this case care should be taken that the new grid has the same resolution as the initial one in order not to change the accuracy of the flow and adjoint solvers between optimisation iterations. This method also suffers from the fact that totally automatic grid generation for structured grids is only possible on simple geometries because for complex ones, human intervention is often necessary.

For unstructured grids, a popular mesh deformation technique is the tension-spring analogy method.[48, 79, 80, 82, 152, 157, 159, 160] It consists in considering the grid as a system of interconnecting springs in equilibrium with a spring stiffness K_{ij} equal to the inverse of the length of the side that links node i to node j. The equations of equilibrium for each node form a system

$$\sum_j K_{ij}(\Delta x_i - \Delta x_j) = 0$$

that is solved with a Jacobi iteration strategy. Δx_i and Δx_j are the displacements from the initial position at node i and j. In Reference [157], this technique is applied only for grid points far from the boundary surface because high-aspect ratio cells are used near the surface to solve the Navier-Stokes equations and this technique does not guarantee the conservation of aspect-ratio. Hence another technique, more geometrical, is employed near the surface and it allows large-scale changes in the geometry without any difficulty. Another amendment to the tension-spring analogy technique is made in Reference [161] where torsional springs are added to the tension springs to improve the capabilities of the method and allow large grid changes.

Nielsen and Anderson[162] show some limitation of the tension-spring analogy for unstructured grids and propose a better method based on linear elasticity. Other techniques for unstructured grid deformation are based on the solution of an elliptic equation for the mesh displacements[83] or consider mesh velocities.[163]

For structured grids, the technique widely employed is the flexible grid approach where the grid points are updated along grid lines starting from the deformed geometry and going to the outer farfield boundary. This supposes that an initial grid is created using any grid generator for the baseline configuration. The grid displacement is transmitted from the surface to the outer boundary with an attenuation depending on the arc-length position of the grid points along each grid line. The attenuation is introduced to keep the outer boundary fixed. The methods differ in the way the arc-length is calculated. It is based on coordinate positions in References [154, 155] but this is very limited. A much better way of calculating the arc-length is to use geometrical distances measured either from the deformed surface[16, 47, 88, 125, 145, 147, 153, 154] (this is the technique employed in this study that will be described in the next two sub-sections) or from the outer boundary.[138, 148] The flexible grid technique used in Reference [142] is slightly different and only allows grid movement in a radial direction. An innovative feature is introduced in Reference [164] i.e. the use of the cosine of the arc-length in the grid deformation algorithm.

The flexible grid approach is further developed in References [139, 150, 165] where multiblock meshes can be used and the grid perturbation is transmitted between different blocks. Furthermore the method enables orthogonal (at least in the computational domain, i.e. for example a boundary surface at j=1 and an adjacent one at i=1 that are orthogonal in the computational domain but not necessarily in the physical domain) boundary surfaces / block faces to be modified like at the junction between the wing surface and the fuselage, while the method of flexible grid used in this work would only allow one surface to be modified. An advanced technique for multiblock structured grid deformation is also provided in Reference [166].

The grid update in this work is done in two stages: the first stage is to create a surface grid, the second stage produces the volume grid around this surface grid. Each stage is now described in a separate sub-section.

4.3.2 Surface grid update

The surface grid on a wing is viewed as a spanwise succession of wing sections. The initial wing surface grid is then divided into master sections and normal sections, knowing that after the grid update the master sections will be connected linearly. Hence this implies that the root and tip sections be master sections as well as any section situated at a crank location.

The master sections of the initial surface grid then undergo the transformations described in Sections 4.1.2 and 4.2 of this chapter. Once their shape has been modified and they have been repositioned, the normal sections that existed originally between two master sections are recreated. This is done by a linear interpolation between two consecutive master sections depending on the original position of the normal section between these two master sections. In addition a special treatment is applied to the wing tip to obtain the new surface grid of the wing.

At this point it must be noted that to simplify the grid update process, the last master section at the wing tip is supposed to undergo only translation, rotation or scaling transformations. This last master section should be situated inboard of the actual wing tip to leave some grid sections to define a rounded tip for example. By limiting the possible transformations of the last master section to those cited, it is possible to apply these same transformations to the grid sections further outboard to create a tip geometry similar in shape and grid quality to the initial grid. If the last master section was also allowed to be deformed, this deformation would have to be applied somehow to the tip sections to get a continuous shape. This is not a trivial task and it was avoided for simplicity. This treatment at the wing tip is also applied in the same way at the wing-winglet junction when a winglet is connected to the wing. The winglet is considered here as an almost non-deformable piece of wing stuck at the tip of a normal wing. Hence the same simple transformations applied to a tip can be applied to the winglet but shape deformations that would imply a modification of the wing-winglet junction are avoided for simplicity. This would require some major work to implement and to make sure it works in any situation.

Another potential problem is that almost all the transformations applied to the grid wing sections are carried out in the xy plane. This supposes that the grid wing sections are perfectly planar and situated in this xy plane. Yet this is hardly ever the case for an habitual CFD grid since if the grid sections are usually aligned with a streamwise direction, they are not necessarily planar, let alone in the xy plane. However this does not constitute too much of a problem, the transformations can still be applied, they will generate a geometry that would have been different if the wing section had been in the xy plane but the geometry changes will be reflected in the grid sensitivities and that is all that matters to the optimisation. The only inconvenience is that the parameters that control the wing geometry lose their physical meaning: if for example, a wing section is planar but tilted with respect to the xy plane, the scaling parameter $scale$ will not represent the ratio of a similarity transformation of the wing profile since the z coordinate is unchanged.

To end this description of the surface grid update for a wing, it is necessary to mention the multiblock capability included in this study. This is quite limited compared to the references cited in the previous subsection but works well for a grid around a conventional wing or a BWB with winglet. Multiblock decomposition of a wing is allowed here only in the spanwise k direction. Hence there must be only one block in the chordwise i direction from leading edge to trailing edge and only one block in the normal direction j from wing surface to outer farfield boundary. In the spanwise direction, grid sections situated at the interface between two adjacent blocks, have to be master sections in order to make possible the linear connection between master sections.

When the grid on the surface of the wing has been regenerated, the next part of the surface grid that needs to be created, is the wake surface grid for a C-type of grid around a wing or an aerofoil. The technique used is the same as the one used for the volume grid which will be described in the next sub-section. In this case it is used in a streamwise direction instead of a normal direction and the points on the surface that are moving, are the trailing edge points.

Once the surface grid on the wing and in the wake has been created, the volume grid can be regenerated as explained next.

4.3.3 Volume grid update

The technique employed to modify the volume grid is taken from References [16, 88, 125, 153, 154]. The deformation of the volume grid is considering individually every grid line originating from the internal surface grid already updated and linking the outer farfield boundary. If along one of these lines the grid points are $(x_j, y_j, z_j), j = 1, \ldots, jn$ then the update is done according to

$$x_j^{new} = x_j^{old} + [1 - arc(j)]\,(x_{surface}^{new} - x_{surface}^{old})$$

where

$$arc(j) = \frac{\sum_{l=2}^{j} L_l}{\sum_{l=2}^{jn} L_l}$$

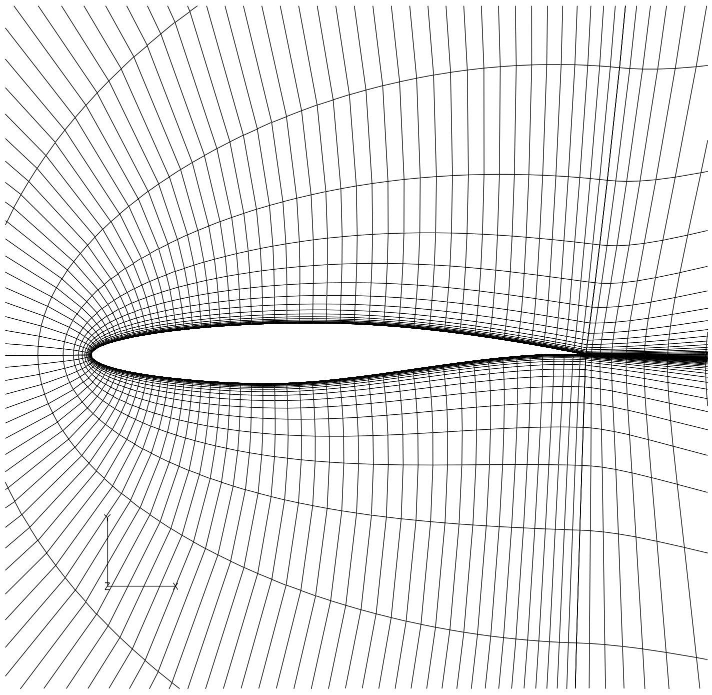
$$L_l = \sqrt{(x_l - x_{l-1})^2 + (y_l - y_{l-1})^2 + (z_l - z_{l-1})^2}$$

and $x_{surface} = x_1$ and the point at the farfield boundary is x_{jn}. A similar transformation is done for the y and z coordinates. It can be seen clearly that the outer boundary is kept fixed by this transformation. This also shows why, in the multiblock case, it is not easy to have more than one block in the normal direction from the wing surface to the outer boundary.

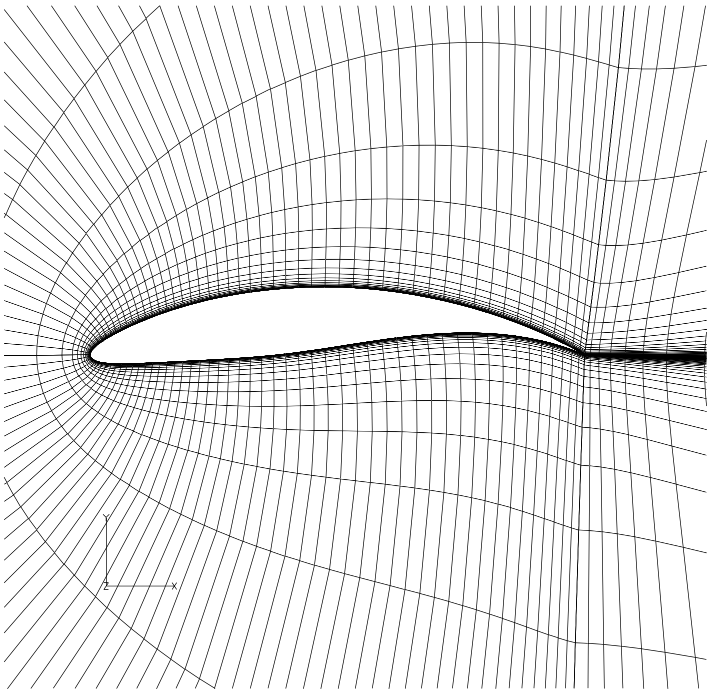
Figure 4.1 to Figure 4.3 give some examples of grid update and summarise what has been presented in this chapter so far. In Figure 4.1, only design variables that are Bézier parameters are used to modify the upper and lower surface of the RAE2822 aerofoil. 10 parameters are used on each surface. Figure 4.1 shows the kind of shape that could be obtained with the technique of Bézier-Bernstein parameterisation chosen in this work. It also demonstrates that the grid deformation technique is working well since the modified grid is still of good quality.

Figure 4.2 and Figure 4.3 give an example of the level of freedom in geometry that can be achieved with the wing geometry representation. Both figures represent the same wing that has been obtained by modifying the ONERA M6 wing.[167] No Bézier parameters were used so the aerofoil shape is the same as for the M6 wing. Only the parameters $displace\text{-}ment$, $scale$ and $twist$ were employed on 10 wing sections to obtain the deformed wing. In Figure 4.2, the background grid is shown in the plane of symmetry and in a cut through the volume grid in the wake of the wing while in Figure 4.3, only the symmetry plane is shown. Despite important deformations, the updated grid is still of acceptable quality. Finally it should be noted that the two examples presented from Figure 4.1 to Figure 4.3 are 4-block grids demonstrating the use of the grid update technique on multiblock grids.

This technique of grid update might be a bit limited in the case of large geometrical changes since the outer boundary is kept fixed and the general behaviour of the grid is conserved (for example if the concavity of a surface changes during the optimisation, some problems can occur). However it should be very efficient for “small” geometric changes, which is usually the case in aerodynamic optimisation. Its main advantage is certainly that it is a simple analytical transformation, hence it is fast and can be easily differentiated to obtain the grid sensitivities as is described in the next section.

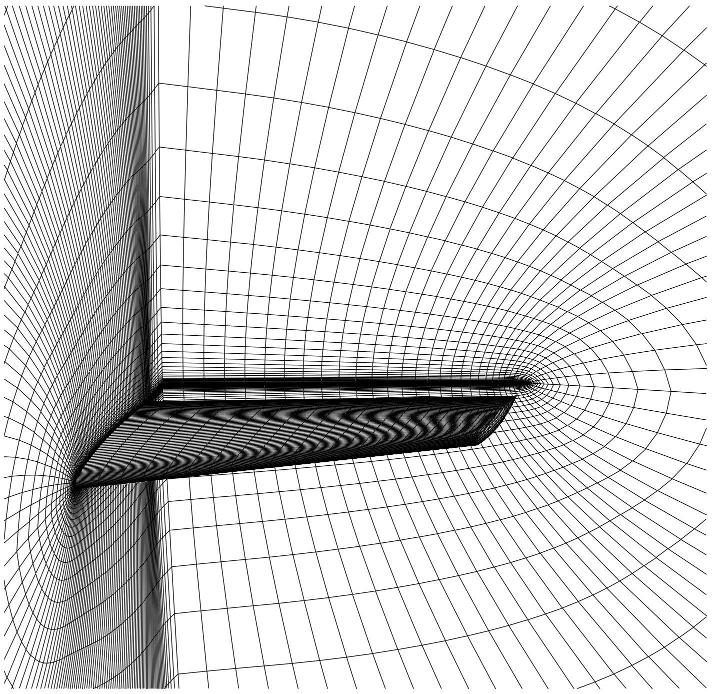


(a) Original RAE2822 aerofoil grid.



(b) Modified shape obtained from the same grid.

Figure 4.1: Example of grid update with 20 Bézier parameters as design variables. (Shape modified manually, not the result of an optimisation)


(a) Original ONERA M6 wing grid.

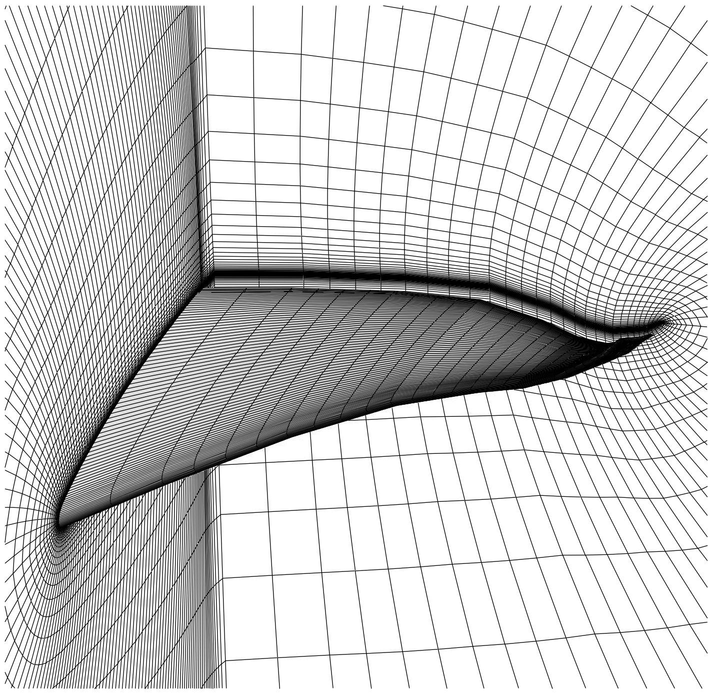
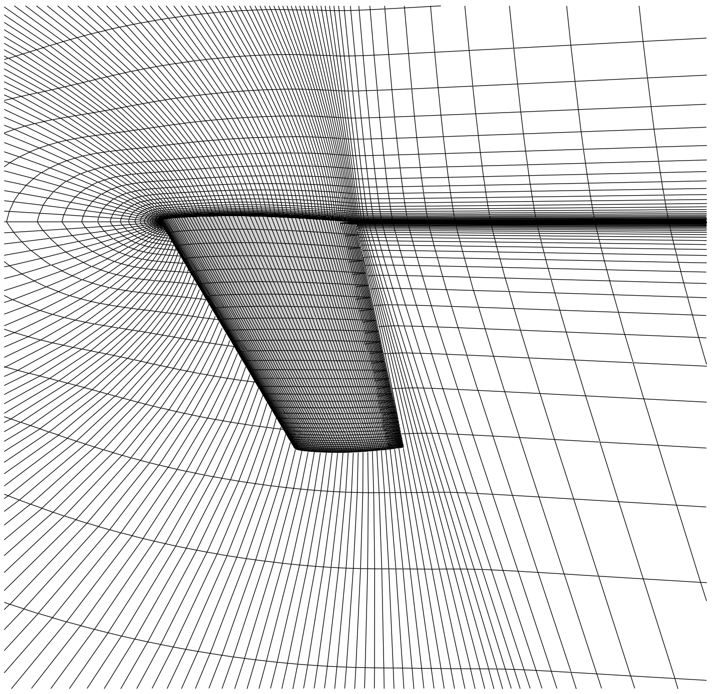

(b) Modified wing obtained from the same grid viewed from the same angle.

Figure 4.2: Example of grid update for parameters controlling the wing geometry. (Shape modified manually, not the result of an optimisation)


(a) Original ONERA M6 wing grid.

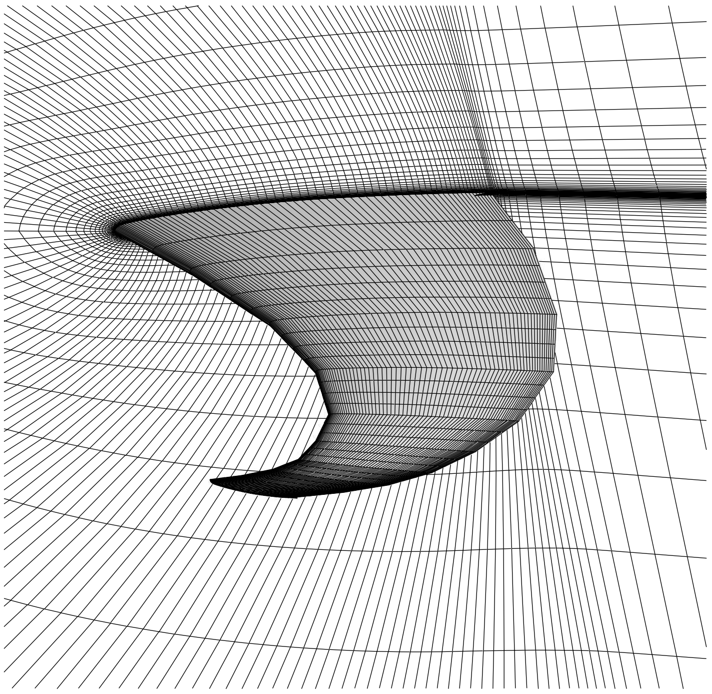

(b) Modified wing obtained from the same grid viewed from the same angle.

Figure 4.3: Same wings as in Figure 4.2 but viewed from a different angle. (Shape modified manually, not the result of an optimisation)

4.4 Grid sensitivities

The grid sensitivities are the vector of derivatives $\frac{\partial \mathbf{X}}{\partial \beta_k}$ where $\mathbf{X} = \{x_{i,j,k}, y_{i,j,k}, z_{i,j,k}\}$ is the vector of grid variables and $\boldsymbol{\beta}$ the vector of design variables for the optimisation. In short, the grid sensitivities represent the movement of each grid point due to a change in design variable. We have seen in Chapter 2 that it was needed for the direct differentiation method. Likewise we will see in Chapter 6 that these quantities are needed by the adjoint solver and hence have to be calculated.

It is often possible, knowing the process of grid perturbation, to calculate the grid sensitivities. When a new grid is regenerated each time with a grid generation package or when a fast grid perturbation method is used, a method of finite-difference can be used to compute the grid sensitivities.[9,43,46,53,89,139,145–147,150,165] This does not avoid however the usual problems associated with finite-differencing i.e. choice of the step length and an inevitable large number of grid generations.

It is also possible to apply Automatic Differentiation to a grid generation code or a grid perturbation code to obtain the grid sensititivities.[28,42,46,70–73,80–82,166] An Automatic Differentiation program applied to a grid generator will provide a code that generates both the grid and the grid sensitivities.

When the algorithm behind a grid generator is known, analytical methods[89] are also possible where the grid sensitivities are written

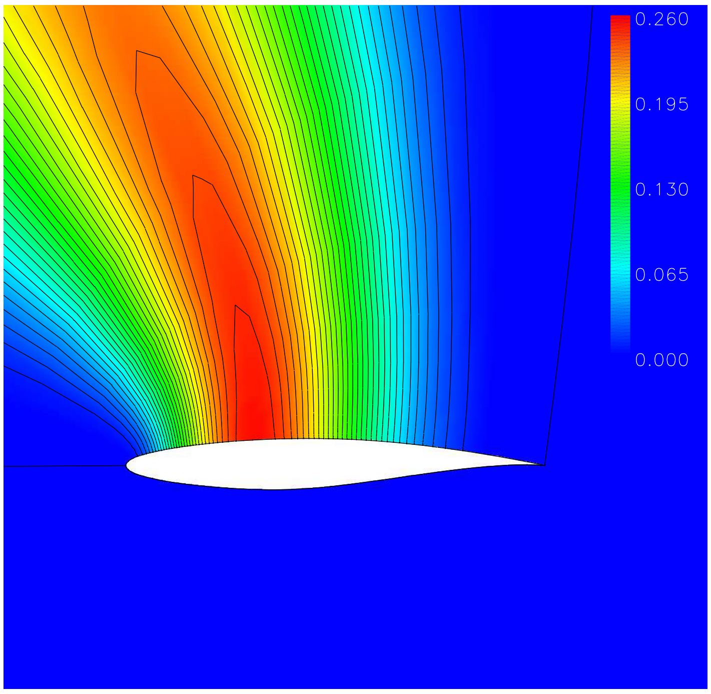
$$\frac{\partial \mathbf{X}}{\partial \beta_k} = \frac{\partial \mathbf{X}}{\partial \mathbf{X}_b} \frac{\partial \mathbf{X}_b}{\partial \beta_k}$$

where $\mathbf{X}_b$ is the vector of grid points situated on the wing or aerofoil surface. This separates the term $\frac{\partial \mathbf{X}}{\partial \mathbf{X}_b}$ that is specific to the grid generator, the calculation of which can be incorporated into the grid generation code, from the term $\frac{\partial \mathbf{X}_b}{\partial \beta_k}$ that is different for each configuration and also depends on the choice of shape parameterisation and design variables. A way of calculating the term $\frac{\partial \mathbf{X}}{\partial \mathbf{X}_b}$ is provided in Reference [168] where the differentiation of the equations of a hyperbolic grid generator and an elliptic grid generator is performed. This differentiation enables the grid sensitivities to be calculated much faster than using a finite-difference method.

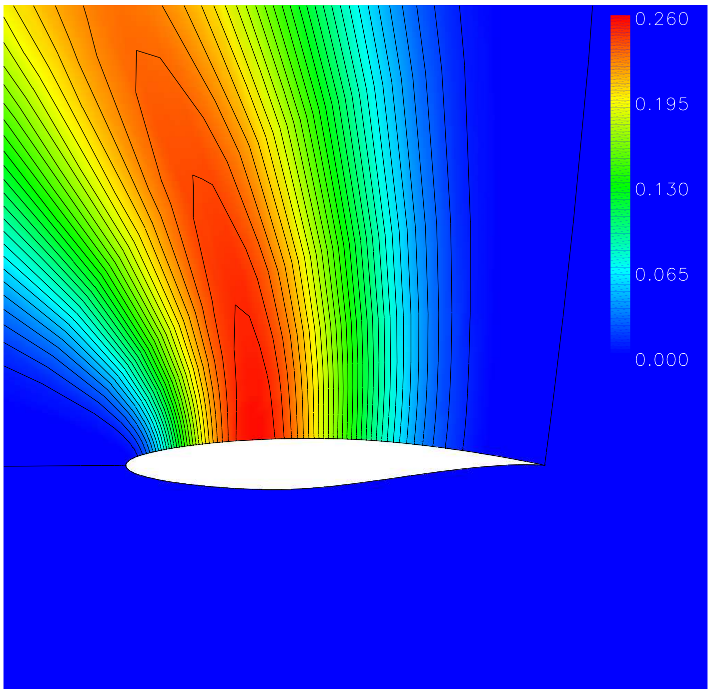
When an analytical method is used to update the grid, it is often quite easy to differentiate directly this algorithm to obtain the grid sensitivities either for structured[138,153,155] or unstructured grids.[48,83] This is what is done in this work. It is an application of the chain rule of differentiation throughout the grid perturbation code. To accelerate the computation of these analytical grid sensitivities, Kim *et al*[83] neglect sensitivities for the interior grid points and only take into account grid sensitivities on the deformed surfaces. They show that it is a valid approximation but this is not considered in this work.

Examples of grid sensitivity calculations made with the hand-differentiated grid perturbation code of this study, are given in Figure 4.4 for a 2D case and in Figure 4.5 for a 3D case. Figure 4.4 shows the grid sensitivity $\dfrac{\partial y}{\partial \beta}$ of the y coordinate of all the grid points around the RAE2822 aerofoil. Here β is the 6th Bézier parameter out of 10 that are used to parameterise the upper surface of the aerofoil. Comparison is made between the analytical computation of the grid sensitivities and a finite-difference computation where the step size was taken as 10^{-6}. Both results are identical, this very good agreement certainly resulting from the fact that only expressions using linear combinations are employed in this case. Figure 4.4 shows that the 6th Bézier parameter controls the shape of the upper surface of the aerofoil at one third of the chord since the grid sensitivities are high in this region. However the Bézier parameterisation is a global parameterisation in the sense that the variation of one parameter implies changes along the whole parameterised curve and not just local changes That is why grid sensitivities are non-zero on the rest of the upper surface. The y direction is the vertical direction in these pictures oriented positively from bottom to top. Since all the sensitivities are positive, this means that a positive increment in β will displace the grid points towards the top of the pictures as is expected and vice-versa for a negative increment. The effect of the volume grid update can also be seen in Figure 4.4: since there is an attenuation of the grid displacement in the volume grid as you move away from the aerofoil surface, the influence of β vanishes and the grid sensitivities become smaller. Finally 4 blocks are employed for this grid and it is clear that the 6th Bézier parameter only influences the block situated above the upper surface of the aerofoil and not the three blocks situated in the wake of the aerofoil and below it, where the grid sensitivities are zero.

Figure 4.5 represents the grid sensitivity $\dfrac{\partial x}{\partial \beta}$ of the x coordinate of all the grid points around the ONERA M6 wing. Here β is the parameter $scale$ associated to the 15th out of 33 spanwise grid sections of the wing. This 15th spanwise grid section is represented in white in the pictures. The grid sensitivities are shown on the upper surface of the wing and on the horizontal planes starting from the leading edge and from the trailing edge. Here again the agreement between the analytical computation and the finite-difference calculation is excellent but here again only linear combinations are differentiated. The x direction is the horizontal direction in these pictures oriented positively from right to left. Hence positive grid sensitivities mean that a positive increment in $scale$ (i.e. an increase in the chord of the 15th wing section) displaces the grid points towards the left hand side of the pictures whereas grid points with negative grid sensitivities are displaced towards the right hand side. It can hence be seen that in this example the scaling of the 15th wing section is done around the quarter chord point since grid points situated at the righ hand side of 25$\%$ chord have negative grid sensitivities and grid points situated at the left hand

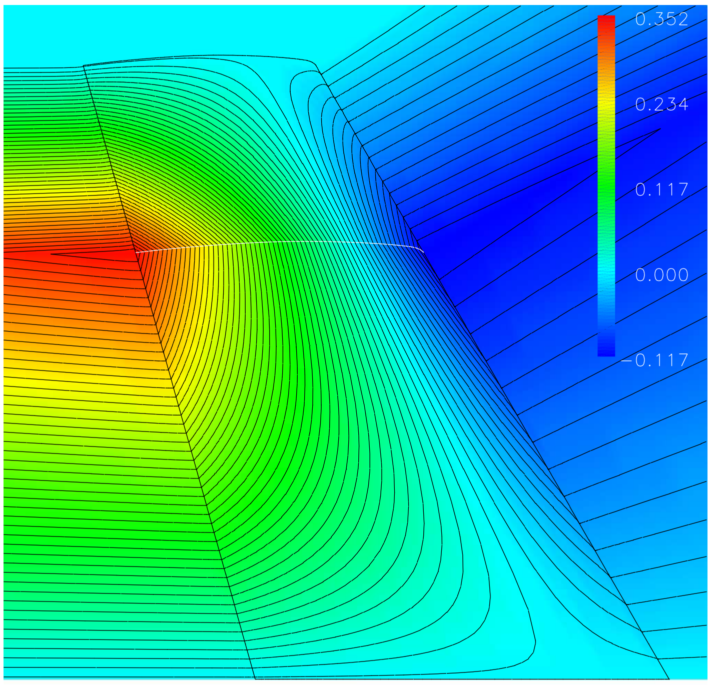


(a) Analytical computation.

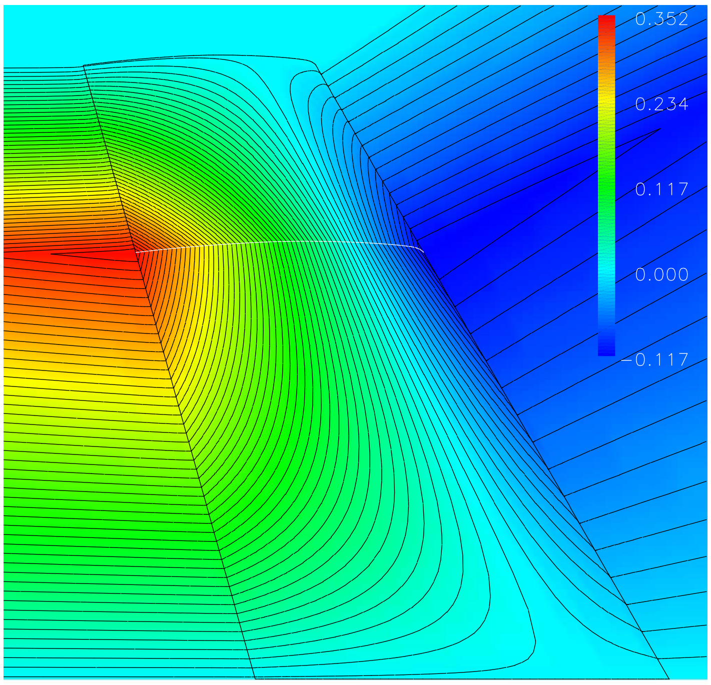


(b) Finite-difference computation.

Figure 4.4: Grid sensitivity $\frac{\partial y}{\partial \beta}$ of the y coordinate with respect to the 6th Bézier parameter out of 10 that parameterise the upper surface of the RAE2822 aerofoil.



(a) Analytical computation.



(b) Finite-difference computation.

Figure 4.5: Grid sensitivity $\frac{\partial x}{\partial \beta}$ of the x coordinate with respect to the parameter $scale$ associated with the 15th spanwise grid section (shown in white) on the upper surface of the ONERA M6 wing.

side of this location have positive grid sensitivities. In the spanwise direction, it can be seen that only 3 master sections were used to parameterise the wing in this case, i.e. the root section, the 15th grid section and the tip section since the whole wing surface is affected by changes in the 15th grid section. Indeed since the geometry modeler assumes a linear connection between the master sections, changes in one master section affect only the spanwise region situated between the previous and the next master section. Since β here only affects the 15th grid section and not the root or the tip, its influence vanishes as you move away from this 15th section in the spanwise direction and the grid sensitivities tend to zero. Finally the volume grid update propagates the changes on the wing to the rest of the grid around the wing. This is shown in the horizontal planes ahead of the wing and in its wake.

This concludes this chapter on the surface parameterisation and grid update. It started by looking at existing shape parameterisation and then by detailing the one chosen in this work i.e. the Bézier-Bernstein parameterisation. The way the wing geometry can be modified, was then examined. The process of grid deformation employed in this thesis was also described. It is an analytical method that can be easily differentiated to provide the grid sensitivities needed by the adjoint solver and that were considered in this last section. The next chapter details an essential component of the optimisation chain developed in this work, i.e. the CFD flow solver MERLIN.

Chapter 5

Fundamental equations and discretisation

The aim of this chapter is to describe the equations that are used in the flow solver and to present the methodology employed to solve them. The first part of this chapter describes the governing equations, the way they are nondimensionalised, their discretisation in space and time. Two formulations are shown: an explicit and an implicit formulation. The explicit approach is described first with the evaluation of the convective terms and then the diffusive terms, followed by the application of the boundary conditions. The implicit formulation is then explained: the general solution methodology is first presented and then the calculation of Jacobians for the convective and diffusive terms, and finally the treatment of the boundary conditions in the Jacobian. The rest of the chapter presents a case of validation, the ONERA M6 wing.

5.1 Introduction

The CFD code employed in this work to perform flow field analyses is called MERLIN and is an in-house code developed at the Centre for Computational Aerodynamics of Cranfield College of Aeronautics. It has been used in References [169–172]. MERLIN is a 3D Reynolds averaged Navier-Stokes flow solver that works on structured multiblock grids. The equations are cast in a cell-centred finite-volume form and the convective flux calculation follows Osher's approximate Riemann solver[173,174] with a MUSCL scheme[175,176] for higher order accuracy. For the time discretisation, either an explicit or a more efficient implicit method can be used. The turbulence model employed in this work for viscous turbulent flows is the algebraic turbulence model of Baldwin-Lomax.[177]

After the summary of the key elements of the flow solver, each of them will be described separately afterwards, starting with a presentation of the governing equations.

5.2 The governing equations

The governing equations are the three-dimensional Navier-Stokes equations which, written in integral form for a bounded domain Ω with surface boundary $\partial\Omega$, are

$$\frac{\partial}{\partial t}\iiint_{\Omega} \mathbf{Q}\, dV + \iint_{\partial\Omega} \mathbf{F}.\mathbf{n}\, dS = 0 \tag{5.1}$$

where the vector of conserved variables $\mathbf{Q}$ is given by

$$\mathbf{Q} = \left(\begin{array}{ccccc} \rho & \rho u & \rho v & \rho w & \rho E \end{array}\right)^t$$

in which ρ is the fluid density, u, v and w are the Cartesian velocity components in the respective x, y and z directions and E is the total energy

$$E = e + \tfrac{1}{2}(u^2 + v^2 + w^2)$$

with the internal energy e.

The flux vector $\mathbf{F}$ is composed of an inviscid and a viscous contribution in the three directions:

$$\mathbf{F} = \begin{bmatrix} \mathbf{F}^i - \mathbf{F}^v \\ \mathbf{G}^i - \mathbf{G}^v \\ \mathbf{H}^i - \mathbf{H}^v \end{bmatrix}.$$

These convective and diffusive contributions are:

$$\begin{aligned}
\mathbf{F}^i &= \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ u(\rho E + p) \end{bmatrix}, \mathbf{F}^v = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ u\tau_{xx} + v\tau_{xy} + w\tau_{xz} - q_x \end{bmatrix} \\
\mathbf{G}^i &= \begin{bmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ v(\rho E + p) \end{bmatrix}, \mathbf{G}^v = \begin{bmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yz} \\ u\tau_{yx} + v\tau_{yy} + w\tau_{yz} - q_y \end{bmatrix} \\
\mathbf{H}^i &= \begin{bmatrix} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + p \\ w(\rho E + p) \end{bmatrix}, \mathbf{H}^v = \begin{bmatrix} 0 \\ \tau_{zx} \\ \tau_{zy} \\ \tau_{zz} \\ u\tau_{zx} + v\tau_{zy} + w\tau_{zz} - q_z \end{bmatrix}
\end{aligned} \tag{5.2}$$

Here p is the static pressure, $\boldsymbol{\tau}$ the stress tensor and $\mathbf{q}$ the heat flux vector. The stress tensor is given by

$$\tau_{ij} = 2\mu S_{ij} + \lambda \frac{\partial u_k}{\partial x_k}\delta_{ij} \tag{5.3}$$

with the strain-rate

$$S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right),$$

the molecular viscosity μ, the second coefficient of viscosity λ and the Kronecker symbol δ_{ij}.

The molecular viscosity is calculated using Sutherland's law, which written in a non-dimensional form is

$$\frac{\mu}{\mu_\infty} = \left(\frac{T}{T_\infty}\right)^{\frac{3}{2}} \frac{\dfrac{110.4}{T_\infty} + 1}{\dfrac{110.4}{T_\infty} + \dfrac{T}{T_\infty}}$$

where T is the temperature, while the second coefficient of viscosity is given by

$$\lambda = -\frac{2}{3}\mu$$

The heat flux vector is given by

$$q_i = -\kappa \frac{\partial T}{\partial x_i}$$

where κ is the thermal conductivity coefficient.

In order to close this mathematical system and be able to solve equation (5.1) for the components of vector $\mathbf{Q}$, two further relationships are needed. The first one relates pressure, density and temperature through the equation of state for a perfect gas

$$p = \rho R T$$

The second one relates internal energy, pressure and density with

$$e = \frac{1}{\gamma - 1}\frac{p}{\rho}$$

where γ is the ratio of specific heats and is taken as $\gamma = 1.4$ throughout this work.

5.3 Primitive variables and non-dimensionalisation

Before going any further into the discretisation of the Navier-Stokes equations (5.1), it is necessary to point out that MERLIN solves for the vector of primitive variables $\mathbf{P}$ instead of the vector of conservative variables $\mathbf{Q}$ presented in the last section. $\mathbf{P}$ is defined as

$$\mathbf{P} = \begin{pmatrix} \rho & u & v & w & p \end{pmatrix}^t$$

Although $\mathbf{P}$ and $\mathbf{Q}$ can often be easily interchanged to mean the flow variables, the author of this thesis will try to be consistent in the utilisation of the notations to match what is actually done in the CFD code. The Jacobians of the transformation from one set of variables to the other are[178]

$$\frac{\partial \mathbf{Q}}{\partial \mathbf{P}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ u & \rho & 0 & 0 & 0 \\ v & 0 & \rho & 0 & 0 \\ w & 0 & 0 & \rho & 0 \\ \frac{u^2+v^2+w^2}{2} & \rho u & \rho v & \rho w & \frac{1}{\gamma-1} \end{bmatrix}$$

and

$$\frac{\partial \mathbf{P}}{\partial \mathbf{Q}} = \frac{\partial \mathbf{Q}}{\partial \mathbf{P}}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -\frac{u}{\rho} & -\frac{1}{\rho} & 0 & 0 & 0 \\ -\frac{v}{\rho} & 0 & -\frac{1}{\rho} & 0 & 0 \\ -\frac{w}{\rho} & 0 & 0 & -\frac{1}{\rho} & 0 \\ \frac{\gamma-1}{2}(u^2+v^2+w^2) & -(\gamma-1)u & -(\gamma-1)v & -(\gamma-1)w & \gamma-1 \end{bmatrix}$$

The other point that is worth mentioning is that MERLIN is not actually solving for the physical value of the variables but for non-dimensional variables. In the version of MERLIN employed in this work, all the variables are non-dimensionalised by the freestream conditions. This is summarised in Table 5.1. In the rest of the thesis, all the variables are non-dimensionalised and for ease of reading, the superscript * is dropped out.

5.4 Finite volume formulation

In order to solve the Navier-Stokes equations (5.1), they are spatially discretised using a finite volume formulation. This means that the whole domain is divided into a large number of small volumes and the integral form of the Navier-Stokes equations is applied to each of these volumes. These small volumes are in fact the cells defined by the computational grid. Since the conservation of mass, momentum and energy is satisfied through the Navier-Stokes equations at the level of each cell, it is also satisfied for the entire domain as would happen if the equations were directly applied to this entire domain. The advantage of working at the cell level is that the Navier-Stokes equations can now be simplified and for a cell i of volume V_i, equation (5.1) becomes

$$V_i \frac{\partial \mathbf{Q}_i}{\partial t} = -\mathbf{R}_i \tag{5.4}$$

where $\mathbf{Q}_i$ is the cell-averaged state variables for cell i and the residual vector $\mathbf{R}_i$ is the sum of all the fluxes passing through each of the cell faces

$$\mathbf{R}_i = \sum_{faces} \mathbf{F}(\mathbf{Q}_i).\mathbf{n}S \tag{5.5}$$

Physical variable	Non-dimensionalisation
ρ	$\rho^* = \dfrac{\rho}{\rho_\infty}$
u	$u^* = \dfrac{u}{\sqrt{{u_\infty}^2 + {v_\infty}^2 + {w_\infty}^2}}$
v	$v^* = \dfrac{v}{\sqrt{{u_\infty}^2 + {v_\infty}^2 + {w_\infty}^2}}$
w	$w^* = \dfrac{w}{\sqrt{{u_\infty}^2 + {v_\infty}^2 + {w_\infty}^2}}$
p	$p^* = \dfrac{p}{\rho_\infty({u_\infty}^2 + {v_\infty}^2 + {w_\infty}^2)} = \dfrac{p}{p_\infty \gamma {M_\infty}^2}$
μ	$\mu^* = \dfrac{\mu}{\mu_\infty}$
T	$T^* = \dfrac{T}{T_\infty}$

Table 5.1: Non-dimensionalisation applied in MERLIN.

$\mathbf{n}$ is the vector normal to the face pointing outwards and S is the area of this face.

A cell-centred approach is also taken which means that the cell-averaged vector $\mathbf{P}$ is stored at the centre of the cell. A typical three-dimensional cell is shown in Figure 5.1: the primitive variables $\mathbf{P}$ are stored at the cell centre; the relationship between this centre and the actual grid point coordinates is indicated; also shown are the metric vectors / cell face normal vectors used in the code: $\boldsymbol{\xi}$ in the i direction, $\boldsymbol{\eta}$ in the j direction and $\boldsymbol{\zeta}$ in the k direction.

In addition to being discretised in space, the Navier-Stokes equations also need to be discretised in time to be solved.

5.5 Time discretisation

All the problems treated in this work are not time dependent hence only the steady state Navier-Stokes equations could be considered. However it is a common practice to use the time dependency as an efficient way to drive the resolution process towards a steady-state converged solution. This is what is done in MERLIN. The semi-discrete form of the Navier-Stokes equations (5.1) has already been presented in equation (5.4) after spatial

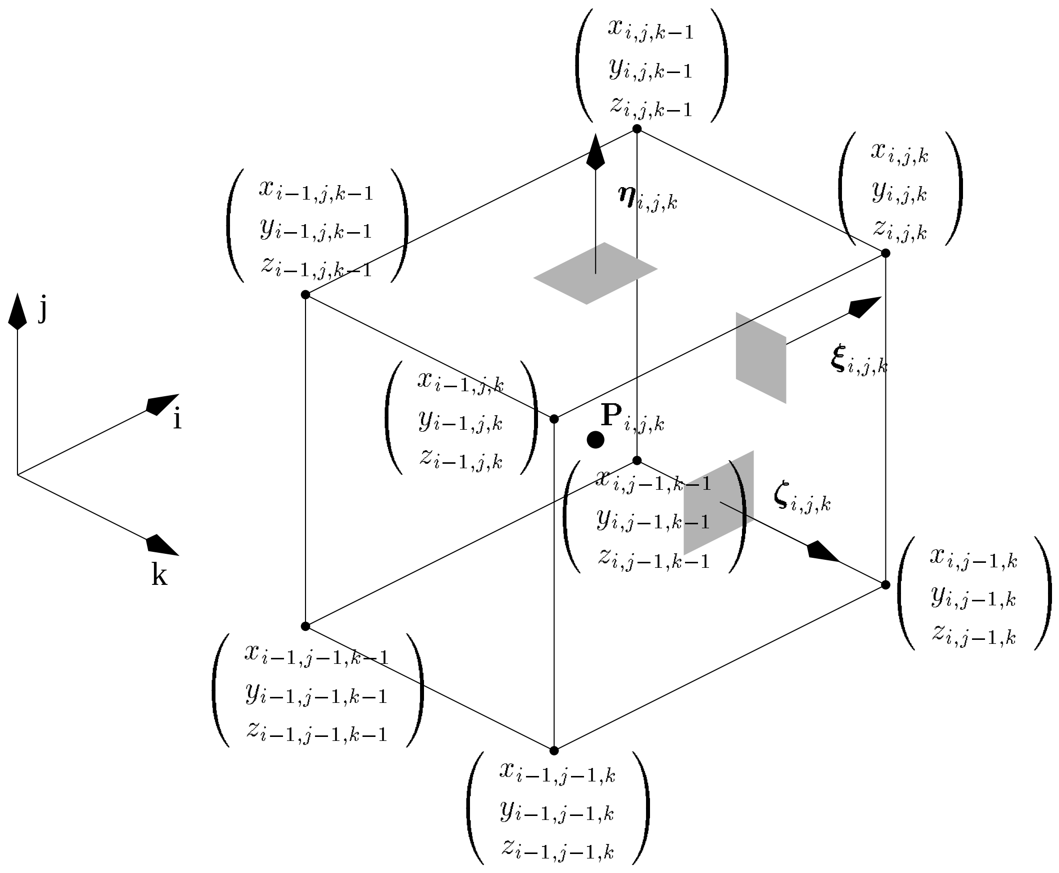

$\begin{pmatrix} x_{i,j,k-1} \\ y_{i,j,k-1} \\ z_{i,j,k-1} \end{pmatrix}$
$\begin{pmatrix} x_{i,j,k} \\ y_{i,j,k} \\ z_{i,j,k} \end{pmatrix}$
$\begin{pmatrix} x_{i-1,j,k-1} \\ y_{i-1,j,k-1} \\ z_{i-1,j,k-1} \end{pmatrix}$
$\boldsymbol{\eta}_{i,j,k}$
j
$\boldsymbol{\xi}_{i,j,k}$
i
$\begin{pmatrix} x_{i-1,j,k} \\ y_{i-1,j,k} \\ z_{i-1,j,k} \end{pmatrix}$
$\mathbf{P}_{i,j,k}$
$\begin{pmatrix} x_{i,j-1,k-1} \\ y_{i,j-1,k-1} \\ z_{i,j-1,k-1} \end{pmatrix}$
$\boldsymbol{\zeta}_{i,j,k}$
k
$\begin{pmatrix} x_{i,j-1,k} \\ y_{i,j-1,k} \\ z_{i,j-1,k} \end{pmatrix}$
$\begin{pmatrix} x_{i-1,j-1,k-1} \\ y_{i-1,j-1,k-1} \\ z_{i-1,j-1,k-1} \end{pmatrix}$
$\begin{pmatrix} x_{i-1,j-1,k} \\ y_{i-1,j-1,k} \\ z_{i-1,j-1,k} \end{pmatrix}$

Figure 5.1: A typical cell with its centre, its grid points and its metric vectors.

discretisation. This can be rewritten

$$V\frac{\partial \mathbf{Q}}{\partial t} = -\mathbf{R}(\mathbf{Q}) \tag{5.6}$$

Since the problem is time independent, a converged steady-state solution is characterised by

$$\mathbf{R}(\mathbf{Q}) = \mathbf{0}$$

This serves, within numerical errors, as convergence criterion to check if the steady-state is reached or not.

A possible temporal discretisation of equation (5.6) is

$$\frac{\mathbf{Q}^{n+1} - \mathbf{Q}^{n}}{\Delta t} = -\mathbf{R}(\mathbf{Q}^{n}) \tag{5.7}$$

where the cell volume is incorporated in the time step Δt. This constitutes the explicit method available in MERLIN and is rather simple since the residual vector only depends on the current value of the vector $\mathbf{Q}$. Hence the update for the next time step is straightforward:

$$\mathbf{Q}^{n+1} = \mathbf{Q}^{n} - \Delta t\, \mathbf{R}(\mathbf{Q}^{n}) \tag{5.8}$$

A method of local time-stepping is employed in MERLIN which means that the time step is not uniform across the domain at the same time iteration but depends on the size of the cell and the flow characteristics at the cell considered. Indeed since the time dependency is only used to reach a steady state, the time step should not have an effect on the final steady state solution and can be chosen more or less arbitrarily from one cell to the other. Since different regions of the flow field are marched in time at different speeds, the whole process will reach a steady state much quicker, in terms of iteration number and hence computing time, than if the same time step was used for the entire domain at each iteration.

In MERLIN, the time step is calculated according to

$$\Delta t = \frac{\text{CFL}}{|\mathbf{V}.\boldsymbol{\xi}| + |\mathbf{V}.\boldsymbol{\eta}| + |\mathbf{V}.\boldsymbol{\zeta}| + a(\|\boldsymbol{\xi}\| + \|\boldsymbol{\eta}\| + \|\boldsymbol{\zeta}\|) + \dfrac{2\gamma\mu}{\rho RePrV}(\|\boldsymbol{\xi}\|^2 + \|\boldsymbol{\eta}\|^2 + \|\boldsymbol{\zeta}\|^2)} \tag{5.9}$$

where $\mathbf{V} = (u, v, w)$ is the velocity vector in three dimensions. The last term in the denominator involving the Reynolds Re and Prandtl Pr numbers relates to the viscous terms and thus is not employed for inviscid computations. All the quantities in equation (5.9) are local quantities relating to the cell where the time step is computed, except the Courant-Friedrichs-Lewy (CFL) number (and of course γ, Re and Pr) that has the same value for the whole domain. In the explicit method, the CFL number is kept constant during all the solution process.

However for numerical stability reasons, the time step Δt cannot be chosen arbitrarily. The stability analysis[179] imposes that the CFL number be smaller than unity for the one-dimensional convection equation. For viscous problems in three dimensions, the CFL number has to be even smaller, leading to small values for the time step. Even by using local time-stepping, this limitation in the time step value makes the whole process very slow to converge towards a steady-state solution. Hence the explicit method is simple to implement but not very efficient and an implicit method will be preferred.

In an implicit method, the residual vector is evaluated at the future time level and hence the discretisation of equation (5.6) becomes

$$\frac{\mathbf{Q}^{n+1} - \mathbf{Q}^n}{\Delta t} = -\mathbf{R}(\mathbf{Q}^{n+1}) \tag{5.10}$$

This discretisation is much more complicated than the explicit method since a simple update like equation (5.8) is no longer possible. The method chosen consists in linearising the residual vector around its value at the time level n

$$\mathbf{R}(\mathbf{Q}^{n+1}) = \mathbf{R}(\mathbf{Q}^n) + \frac{\partial \mathbf{R}(\mathbf{Q}^n)}{\partial \mathbf{Q}}\, {}^n\Delta\mathbf{Q} + \text{higher order terms} \tag{5.11}$$

where

$${}^n\Delta\mathbf{Q} = \mathbf{Q}^{n+1} - \mathbf{Q}^n$$

The introduction of this linearisation (5.11) into equation (5.10) gives

$$\left[\frac{\mathbf{I}}{\Delta t} + \frac{\partial \mathbf{R}(\mathbf{Q}^n)}{\partial \mathbf{Q}}\right] {}^n\Delta\mathbf{Q} = -\mathbf{R}(\mathbf{Q}^n) \tag{5.12}$$

Equation (5.12) is now a linear system that has to be solved for ${}^n\Delta\mathbf{Q}$. The way of solving this linear system will be presented in a latter subsection. To be precise, equation (5.12) is further modified in MERLIN to be solved for primitive variables:

$$\left[\frac{1}{\Delta t}\frac{\partial \mathbf{Q}}{\partial \mathbf{P}} + \frac{\partial \mathbf{R}(\mathbf{Q}^n)}{\partial \mathbf{P}}\right] {}^n\Delta\mathbf{P} = -\mathbf{R}(\mathbf{Q}^n) \tag{5.13}$$

Once this is solved for ${}^n\Delta\mathbf{P}$, the update formula is simply

$$\mathbf{P}^{n+1} = \mathbf{P}^n + {}^n\Delta\mathbf{P} \tag{5.14}$$

The implicit method enables the use of much larger time steps than the explicit method or, equivalently, much larger CFL numbers. In the implicit version of MERLIN, the CFL number is increased logarithmically with respect to the total residual at each iteration by the formula

$$\mathrm{CFL}^n = \mathrm{CFL}^0 - \alpha \log_{10}\left(\frac{R_{total}(\mathbf{Q}^n)}{R_{total}(\mathbf{Q}^0)}\right)$$

The total residual R_{total} is the square root of the sum of the square of all the components of the residual vector for each computational cell. Hence as the residual tends to zero i.e. the more the solution is converged, the higher the CFL number is and the larger the time step is. Even if a maximum value for the CFL number is set inside the code, this method is very efficient and converges quickly. Its only drawback is that it is much more complicated to implement than the explicit method.

Now that the explicit and the implicit methods available in MERLIN have been introduced, each of them is described in more detail in the next two sections.

5.6 Explicit formulation

This section presents first the calculation of the convective fluxes in the explicit methodology. This is followed by the description of the diffusive fluxes and the application of the boundary conditions in this methodology.

5.6.1 Convective terms

The calculation of the convective part of the flux $\mathbf{F}(\mathbf{Q}_i)$ in equation (5.5) could be a simple application of the formulae (5.2) at the face between two adjacent cells. However such a simple numerical scheme would be unstable in an explicit method and would fail to capture flow discontinuities like shock waves and shear layers. Some flow physics has to be incorporated into the calculation of the flux. This is done by using Osher's approximate Riemann solver presented in the next subsection.

5.6.1.1 Osher's approximate Riemann solver

An approximate Riemann solver calculates a flux at an interface by incorporating local information on the flow characteristics at this interface. This is done by solving approximately a one-dimensional Riemann problem also called shock-tube problem at this interface. The interface represents the membrane of the problem with two different flow conditions $\mathbf{Q}_L$ and $\mathbf{Q}_R$ on each side of this membrane. This is represented in Figure 5.2. Depending on the state of the flow on each side of the interface, different expressions are provided to calculate the flux and hence be able to capture any flow discontinuity.

Osher's scheme[173, 174] is a flux splitting method. It has the nice property of being continuously differentiable and can thus be used in an implicit solution methodology. The complete derivation of Osher's approximate Riemann solver is given in References [180,181], summaries of this derivation being found in References [182, 183] for example. During this derivation an integral involving the Jacobian has to be evaluated across the interface from $\mathbf{Q}_L \equiv \mathbf{Q}_0$ to $\mathbf{Q}_R \equiv \mathbf{Q}_1$. The value of this integral depends on the integration path followed. We follow the P-variant of Osher's scheme that integrates in the physical space. This integration introduces the intermediate intersection points $1/3$ and $2/3$ and

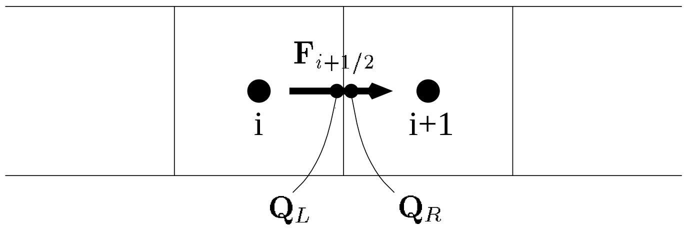


Figure 5.2: Flux at the interface between two cells.

	$\overline{U}_0 - a_0 \geq 0$ $\overline{U}_1 + a_1 \geq 0$	$\overline{U}_0 - a_0 \geq 0$ $\overline{U}_1 + a_1 \leq 0$	$\overline{U}_0 - a_0 \leq 0$ $\overline{U}_1 + a_1 \geq 0$	$\overline{U}_0 - a_0 \leq 0$ $\overline{U}_1 + a_1 \leq 0$
$\overline{U}^* \geq 0$ $\overline{U}^* - a_{1/3} \geq 0$	$\mathbf{F}(\mathbf{Q}_0)$	$\mathbf{F}(\mathbf{Q}_0) + \mathbf{F}(\mathbf{Q}_1)$ $-\mathbf{F}(\mathbf{Q}_{S1})$	$\mathbf{F}(\mathbf{Q}_{S0})$	$\mathbf{F}(\mathbf{Q}_{S0}) - \mathbf{F}(\mathbf{Q}_{S1})$ $+\mathbf{F}(\mathbf{Q}_1)$
$\overline{U}^* \geq 0$ $\overline{U}^* - a_{1/3} \leq 0$	$\mathbf{F}(\mathbf{Q}_0) - \mathbf{F}(\mathbf{Q}_{S0})$ $+\mathbf{F}(\mathbf{Q}_{1/3})$	$\mathbf{F}(\mathbf{Q}_0) - \mathbf{F}(\mathbf{Q}_{S0})$ $+\mathbf{F}(\mathbf{Q}_{1/3}) - \mathbf{F}(\mathbf{Q}_{S1})$ $+\mathbf{F}(\mathbf{Q}_1)$	$\mathbf{F}(\mathbf{Q}_{1/3})$	$\mathbf{F}(\mathbf{Q}_1) + \mathbf{F}(\mathbf{Q}_{1/3})$ $-\mathbf{F}(\mathbf{Q}_{S1})$
$\overline{U}^* \leq 0$ $\overline{U}^* + a_{2/3} \geq 0$	$\mathbf{F}(\mathbf{Q}_0) - \mathbf{F}(\mathbf{Q}_{S0})$ $+\mathbf{F}(\mathbf{Q}_{2/3})$	$\mathbf{F}(\mathbf{Q}_0) - \mathbf{F}(\mathbf{Q}_{S0})$ $+\mathbf{F}(\mathbf{Q}_{2/3}) - \mathbf{F}(\mathbf{Q}_{S1})$ $+\mathbf{F}(\mathbf{Q}_1)$	$\mathbf{F}(\mathbf{Q}_{2/3})$	$\mathbf{F}(\mathbf{Q}_{2/3}) - \mathbf{F}(\mathbf{Q}_{S1})$ $+\mathbf{F}(\mathbf{Q}_1)$
$\overline{U}^* \leq 0$ $\overline{U}^* + a_{2/3} \leq 0$	$\mathbf{F}(\mathbf{Q}_0) - \mathbf{F}(\mathbf{Q}_{S0})$ $+\mathbf{F}(\mathbf{Q}_{S1})$	$\mathbf{F}(\mathbf{Q}_0) - \mathbf{F}(\mathbf{Q}_{S0})$ $+\mathbf{F}(\mathbf{Q}_1)$	$\mathbf{F}(\mathbf{Q}_{S1})$	$\mathbf{F}(\mathbf{Q}_1)$

Table 5.2: Osher's flux formulae for $\mathbf{F}_{i+1/2}$ (P-variant). ($\mathbf{Q}_L \equiv \mathbf{Q}_0$ and $\mathbf{Q}_R \equiv \mathbf{Q}_1$)

sonic points $S0$ and $S1$. The flow variables at these intermediate points are calculated from $\mathbf{Q}_0$ and $\mathbf{Q}_1$ using Riemann invariants. Here it is sufficient to know that $\mathbf{Q}_{1/3}$ and $\mathbf{Q}_{2/3}$ depend on both $\mathbf{Q}_0$ and $\mathbf{Q}_1$ whereas $\mathbf{Q}_{S0}$ only depends on $\mathbf{Q}_0$ and $\mathbf{Q}_{S1}$ only depends on $\mathbf{Q}_1$. The integration leads to 16 possible combinations depending on the value of $\mathbf{Q}_0$ and $\mathbf{Q}_1$ and the value of $\mathbf{Q}$ at the intersection points. These are given in Table 5.2 where $\overline{U}$ represents the velocity normal to the interface, a is the speed of sound and $\overline{U}^* = \overline{U}_{1/3} = \overline{U}_{2/3}$. Table 5.2 is to be read in the following way:

> if, for example, $\overline{U}_0 - a_0 \geq 0, \overline{U}_1 + a_1 \geq 0$ and $\overline{U}^* \geq 0, \overline{U}^* - a_{1/3} \leq 0$
>
> then $\mathbf{F}_{i+1/2} = \mathbf{F}(\mathbf{Q}_0) - \mathbf{F}(\mathbf{Q}_{S0}) + \mathbf{F}(\mathbf{Q}_{1/3})$.

where the flux $\mathbf{F}(\mathbf{Q})$ is given by

$$\mathbf{F}(\mathbf{Q}) = \begin{bmatrix} \rho\overline{U} \\ \rho\overline{U}u + n_x p \\ \rho\overline{U}v + n_y p \\ \rho\overline{U}w + n_z p \\ \overline{U}(\rho E + p) \end{bmatrix}$$

and the unit vector normal to the interface is $\mathbf{n} = (n_x, n_y, n_z)$.

5.6.1.2 Higher-order spatial accuracy

If to calculate the flux at the interface between cell i and cell $i+1$, the value of $\mathbf{Q}_L$ and $\mathbf{Q}_R$ are taken as $\mathbf{Q}_i$ and $\mathbf{Q}_{i+1}$ respectively in Osher's scheme, then the discretisation is only first order accurate in space. This will lead to poor accuracy in smooth regions of the flow field. To increase this accuracy a MUSCL scheme[175,176] is used and is basically a linear extrapolation to the flux interface of the flow properties in the two cells adjacent to the interface on each side of this interface. Considering the interface between cell i and cell $i+1$, it is written as

$$\begin{aligned} \mathbf{P}_L &= \mathbf{P}_i \quad + \tfrac{1}{4}[(1-\kappa)(\mathbf{P}_i - \mathbf{P}_{i-1}) + (1+\kappa)(\mathbf{P}_{i+1} - \mathbf{P}_i)] \\ \mathbf{P}_R &= \mathbf{P}_{i+1} - \tfrac{1}{4}[(1+\kappa)(\mathbf{P}_{i+1} - \mathbf{P}_i) + (1-\kappa)(\mathbf{P}_{i+2} - \mathbf{P}_{i+1})] \end{aligned} \tag{5.15}$$

where κ is chosen as $\kappa = 1/3$ in this work. With this value of κ the discretisation is third order accurate in space. Note that the MUSCL scheme is applied to the primitive variables in MERLIN, while Osher's Riemann solver is applied to the conservative variables.

Though this version of the MUSCL scheme gives very good accuracy in smooth regions of the flow field, it will generate spurious oscillations and even prevent convergence of the solution in the neighbourhood of flow discontinuities like shock waves or shear layers. In these regions of large flow gradients, it is necessary to revert to a first order approximate Riemann solver designed to tackle flow discontinuities. Hence the idea is to keep a third order scheme in the smooth regions of the flow field and to change to a first order scheme in the neighbourhood of discontinuities. This is achieved by introducing a slope limiter. The slope limiter chosen in this study is

$$s_i = \frac{2(P_{i+1} - P_i)(P_i - P_{i-1}) + \varepsilon}{(P_{i+1} - P_i)^2 + (P_i - P_{i-1})^2 + \varepsilon}$$

and is applied component by component. ε is a small number preventing the denominator from becoming zero in smooth flow regions. The advantage of this limiter is that it is differentiable.

The MUSCL scheme of equation (5.15) is modified to introduce this limiter

$$\begin{aligned} \mathbf{P}_L &= \mathbf{P}_i \quad + \tfrac{1}{4}s_i \quad [(1 - s_i\kappa)(\mathbf{P}_i - \mathbf{P}_{i-1}) + (1 + s_i\kappa)(\mathbf{P}_{i+1} - \mathbf{P}_i)] \\ \mathbf{P}_R &= \mathbf{P}_{i+1} - \tfrac{1}{4}s_{i+1}[(1 + s_{i+1}\kappa)(\mathbf{P}_{i+1} - \mathbf{P}_i) + (1 - s_{i+1}\kappa)(\mathbf{P}_{i+2} - \mathbf{P}_{i+1})] \end{aligned} \tag{5.16}$$

In regions of zero flow curvature ($P_{i+1} - P_i \approx P_i - P_{i-1} = \Delta P$), s_i is close to 1 and the MUSCL scheme is the one in equation (5.15) whereas in regions of large flow curvature (for example when $P_{i+1} - P_i \gg P_i - P_{i-1}$), s_i is close to 0 and the MUSCL scheme reverts to a simple first order scheme. In addition, whenever the application of the MUSCL scheme and its limiter produces non physical values (negative density and/or pressure), a first order method is employed.

5.6.2 Diffusive terms

This subsection details the calculation of the diffusive terms $\mathbf{F}^v$, $\mathbf{G}^v$ and $\mathbf{H}^v$ in equation (5.2). It starts with the laminar viscous part and then examines the turbulence modelling.

5.6.2.1 Laminar viscous fluxes

The viscous fluxes are calculated using a central discretisation which means that the formulae (5.2) for $\mathbf{F}^v$, $\mathbf{G}^v$ and $\mathbf{H}^v$ are applied at the centre of the face where we want to calculate the flux. However velocity and temperature gradients need to be evaluated at this centre point in order to calculate the stress tensor τ. They are approximated using Gauss' theorem.[184] For the component u of the velocity in the x direction this is written for a bounded domain Ω with surface boundary $\partial\Omega$:

$$\int_\Omega \nabla u \, dV = \oint_{\partial\Omega} u\mathbf{n} \, dS$$

which, once discretised on an hexahedral volume, becomes

$$\nabla u = \frac{1}{V}\sum_{l=1}^{6} u_l \mathbf{n}_l S_l \tag{5.17}$$

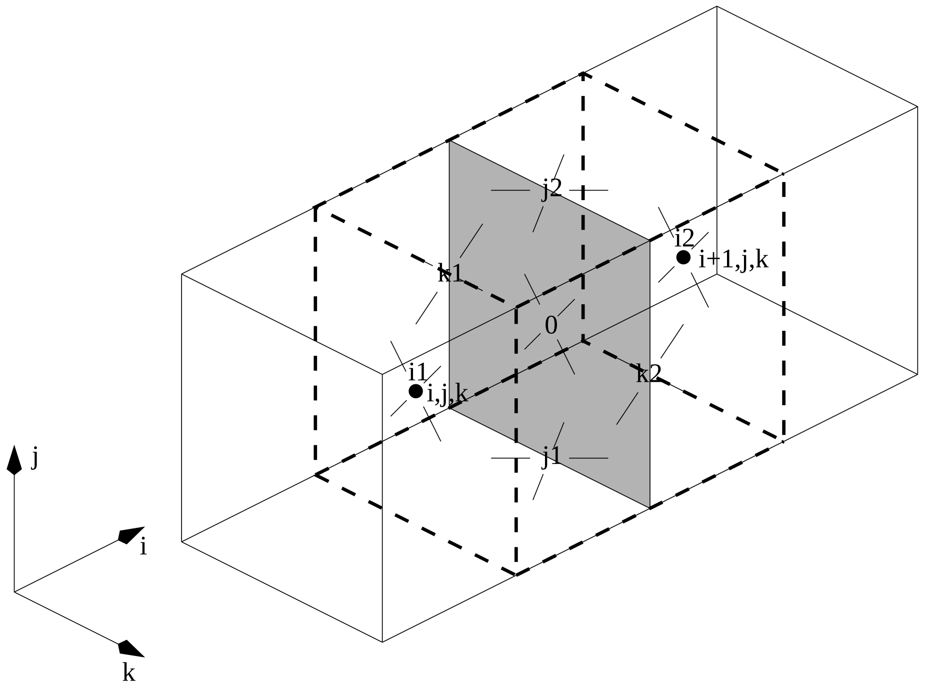


Figure 5.3: The dual volume used to calculate the viscous flux at the face between cell i, j, k and cell $i + 1, j, k$.

where V is the volume of this domain, u_l the averaged value of u on the six faces of this domain, $\mathbf{n}_l$ the unit normal pointing outwards on these faces and S_l the area of these faces.

Let's assume we want to calculate the viscous flux through the face between cell i, j, k and cell $i + 1, j, k$ represented as the shaded area in Figure 5.3. Gauss' theorem (5.17) is applied to a dual volume that encloses this face and the edges of which are represented by a dashed line in Figure 5.3. The diagonal crosses show the location of the face centres. With the notation of that figure, this gives

$$\begin{aligned}\nabla u|_0 &= \frac{1}{V}[u_{i2}\mathbf{n}_{i2}S_{i2} + u_{i1}\mathbf{n}_{i1}S_{i1} + u_{j2}\mathbf{n}_{j2}S_{j2} + u_{j1}\mathbf{n}_{j1}S_{j1} + u_{k2}\mathbf{n}_{k2}S_{k2} + u_{k1}\mathbf{n}_{k1}S_{k1}] \\ &= \frac{1}{V}[u_{i2}\boldsymbol{\xi}_{i2} - u_{i1}\boldsymbol{\xi}_{i1} + u_{j2}\boldsymbol{\eta}_{j2} - u_{j1}\boldsymbol{\eta}_{j1} + u_{k2}\boldsymbol{\zeta}_{k2} - u_{k1}\boldsymbol{\zeta}_{k1}]\end{aligned} \tag{5.18}$$

with the introduction of the metric vectors $\boldsymbol{\xi}$,$\boldsymbol{\eta}$ and $\boldsymbol{\zeta}$ already presented in Figure 5.1. The values needed at the centre of the faces of the dual volume are evaluated by making an average of known quantities. For example,

$$u_{k2} = \frac{1}{4}(u_{i,j,k} + u_{i+1,j,k} + u_{i,j,k+1} + u_{i+1,j,k+1})$$

$$\boldsymbol{\zeta}_{k2} = \frac{1}{2}(\boldsymbol{\zeta}_{i,j,k} + \boldsymbol{\zeta}_{i+1,j,k})$$

A projection of equation (5.18) on the three Cartesian directions gives the value of the three components of the gradient. For example,

$$\left.\frac{\partial u}{\partial y}\right|_0 = \frac{1}{V}[u_{i2}\xi_{y\,i2} - u_{i1}\xi_{y\,i1} + u_{j2}\eta_{y\,j2} - u_{j1}\eta_{y\,j1} + u_{k2}\zeta_{y\,k2} - u_{k1}\zeta_{y\,k1}]$$

The same methodology is employed to evaluate the gradients of the velocity components v and w as well as temperature gradients needed to calculate $\mathbf{F}^v$, $\mathbf{G}^v$ and $\mathbf{H}^v$. The laminar viscous contributions are then easy to calculate. This however only concerns the laminar part of the viscous fluxes. How MERLIN deals with turbulent flows is described in the next subsection.

5.6.2.2 Turbulence modelling

The use of the Navier-Stokes equations (5.1) for turbulent flows requires some further development. Turbulence is a physical phenomenon which is essentially three-dimensional, unsteady and involves a wide range of time and space scales. The complete resolution of the Navier-Stokes equations for such a phenomenon would necessitate extraordinarily fine grid resolution and the use of very small time steps. Direct Numerical Simulation (DNS) can address this nowadays but only on very small domains and at the cost of very high computing efforts. For engineering problems, even Large Eddy Simulation (LES) is very demanding and the use of Reynolds averaged Navier-Stokes equations is the only possibility. This involves a time averaging of the Navier-Stokes equations: each flow variable is decomposed into the sum of an averaged component and a fluctuating component and the governing equations are then time averaged. The averaging does not change the structure of the Navier-Stokes equations (5.1) for which the variables are now the averaged components, except that additional terms, called the Reynolds stresses, which involve an averaging of the product of two fluctuating velocity components, are generated. These Reynolds stresses which are unknown, need to be approximated through the use of a turbulence model.

There exists a wide variety of turbulence models[185] ranging, in order of complexity, from simple algebraic models to models solving the Reynolds stress equations. The turbulence models situated in the lower part of this scale rely on the Boussinesq assumption that the Reynolds stresses act as the laminar viscous stresses and are the product of an eddy viscosity and the mean strain-rate tensor. In this way equation (5.3) is changed to

$$\tau_{ij} = 2\mu S_{ij} + \lambda\frac{\partial u_k}{\partial x_k}\delta_{ij} - \frac{2}{3}\rho k\delta_{ij}$$

where k is the kinetic energy and μ is now the sum of the molecular viscosity and the eddy viscosity μ_t. The problem of modelling the Reynolds stresses is then reduced to finding an expression for μ_t.

In this work the algebraic Baldwin-Lomax model[177] is used. It divides the turbulent boundary layer into an inner and outer region with a different expression for the eddy viscosity in each region

$$\mu_t = \begin{cases} (\mu_t)_{inner} \ , & y_n \leq y_{n\,c} \\ (\mu_t)_{outer} \ , & y_n > y_{n\,c} \end{cases}$$

where y_n is the normal distance from the body surface and $y_{n\,c}$ is the distance for which the inner and outer eddy viscosities are equal.

In the inner region, the mixing-length hypothesis is applied and

$$(\mu_t)_{inner} = \rho l^2 |\omega|$$

where $|\omega|$ is the magnitude of the vorticity

$$|\omega| = \sqrt{\left(\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial z} - \frac{\partial w}{\partial y}\right)^2 + \left(\frac{\partial w}{\partial x} - \frac{\partial u}{\partial z}\right)^2}$$

and the mixing length l is calculated with Van Driest function by

$$l = \kappa y_n \left(1 - e^{-\dfrac{y^+}{A^+}}\right)$$

where κ is the von Kármán constant $\kappa = 0.41$, A^+ is the Van Driest constant $A^+ = 26$ and y^+ is the non-dimensional normal distance

$$y^+ = \frac{\rho_w u_\tau y_n}{\mu_w} = \frac{y_n \sqrt{\rho_w \tau_w}}{\mu_w}$$

Here u_τ is the friction velocity, τ is the shear stress and the subscript w refers to quantities evaluated at the wall.

In the outer region, the eddy viscosity is defined by

$$(\mu_t)_{outer} = \rho \alpha C_{CP} F_{wake} F_{Kleb}(y_n)$$

where α is the Clauser constant $\alpha = 0.0168$, C_{CP} is another constant $C_{CP} = 1.6$. The function F_{wake} is defined as

$$F_{wake} = \min\left(y_{max} F_{max}, \frac{C_{wake} y_{max} {U_{diff}}^2}{F_{max}}\right) \tag{5.19}$$

where the wake constant $C_{wake} = 0.25$, U_{diff} is the maximum difference in velocity amplitude across the boundary layer

$$U_{diff} = \max_{y_n}\left(\sqrt{u^2 + v^2 + w^2}\right) - \min_{y_n}\left(\sqrt{u^2 + v^2 + w^2}\right) ,$$

F_{max} is the maximum value across the boundary layer of the function

$$F(y_n) = y_n|\omega| \left(1 - e^{-\dfrac{y^+}{A^+}}\right)$$

and y_{max} is the value of y_n for which $F(y_n) = F_{max}$. This definition of F_{wake} in equation (5.19) is the standard definition but, as we will see in Section 6.6.4, we had to simplify it to $F_{wake} = y_{max}F_{max}$ in both the flow and adjoint solvers. Finally the Klebanoff function is defined as

$$F_{Kleb}(y_n) = \left[1 + 5.5\left(\frac{C_{Kleb}y_n}{y_{max}}\right)^6\right]^{-1}$$

where the Klebanoff constant $C_{Kleb} = 0.3$.

Practically, the eddy viscosity is calculated along rays that start from the wall and expand up to the farfield boundary. These rays are based on the computational grid which in practice limits the possibilities of grid topology. For each cell along these rays, the inner eddy viscosity is calculated as well as the value of the function F. For each ray F_{max} is then determined which makes it possible to calculate the outer eddy viscosity for each cell. A comparison of the values of the inner and outer eddy viscosities gives the actual value of the eddy viscosity. This is the way the Baldwin-Lomax is implemented above and in front of the surfaces for a C type of grid. In the wake though, the eddy viscosity is not actually calculated but the value along the last upstream ray situated above the surface is copied. This is done to simplify the calculation of the eddy viscosity in the wake.

The Baldwin-Lomax model is reasonably simple and gives very good results for attached boundary layers. For boundary layers under strong adverse pressure gradient and for separated boundary layers, it performs relatively poorly. Its main advantage compared to other turbulence models is that it is robust and is an algebraic model and therefore is less computationally demanding than one- or two-equation models. This is important for three-dimensional optimisations where a large number of calls to the flow solver are made on geometries that are changing: the flow solver needs to be as fast as possible and if you use a two-equation model, you must add 2 equations to the laminar viscous code hence possibly increase the computational cost by 40%. The turbulence model also has to be robust since the optimisation process is automated: different shapes have to be handled without difficulties by the flow solver without any user intervention. The popularity of the Baldwin-Lomax model for aeronautical flows, at least among algebraic models, has to be added to this list of reasons for chosing this turbulence model in this study.

This ends this subsection on the evaluation of the diffusive terms with the description of the viscous laminar contribution followed by the turbulence modelling. The next subsection details the application of the boundary conditions.

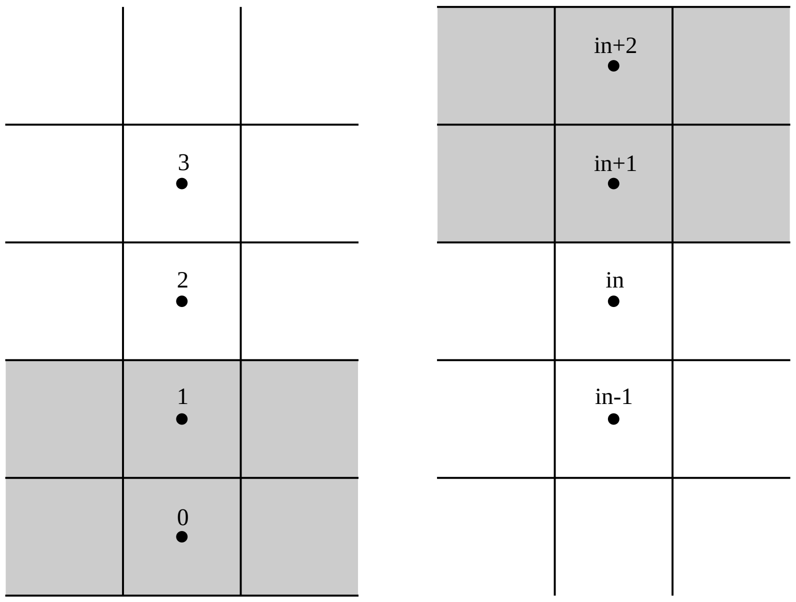


Figure 5.4: Schematic diagram of halo cells (in shaded area). Left: cell numbering starting at the boundary. Right: cell numbering finishing at the boundary.

5.6.3 Boundary conditions

A special treatment has to be applied to the boundaries of the domain in order to introduce the physics of the problem we are trying to model. This is known as the boundary conditions. They are applied in MERLIN through the use of halo cells. These are two layers of fictitious computational cells that are added to the exterior of the domain. This enables the fluxes in cells close to a boundary to be calculated as if they were inside the domain, without having to modify the flux calculation. Figure 5.4 represents schematically these two computational fictitious cells, whether the cell numbering starts at the considered boundary or ends at this boundary. The notation of the former case (cell 1 for the first halo cell directly adjacent to the boundary and cell 0 for the second halo cell away from the boundary) will be kept afterwards for the description of the boundary condition formulae but it is trivial to apply them to the latter case.

For turbulent computations, it is also needed to have a value for the turbulent viscosity inside the halo cells, hence in addition to boundary conditions for the 5 primitive variables ρ, u, v, w, p, another one is needed for μ_t. However the calculation of the diffusive fluxes only requires one layer of halo cells hence the boundary condition for μ_t is only applied to cell 1 compared to cells 1 and 0 for the primitive variables.

Seven types of boundary conditions are used in this work namely the inviscid wall, viscous wall, symmetry, supersonic inflow, supersonic outflow, farfield and interface boundary conditions. Each of them is briefly introduced next. This is followed by a description of our treatment of the corner points.

5.6.3.1 Inviscid wall boundary condition

The inviscid wall boundary condition is applied at a surface only when the inviscid Euler equations are solved. This is a slip boundary condition that requires that the component of the velocity normal to the wall at the surface be zero. Let us denote $\overline{U}$ the component of the velocity normal to the wall and $\overline{V}$ and $\overline{W}$ the two velocity components parallel to the wall. These are hence the components of the velocity in a reference frame attached to the wall and should not be confused with u, v and w which are the components of the velocity in the general Cartesian reference frame (x, y, z) and are part of the vector of variables $\mathbf{P}$. A suitable transformation makes it possible to pass from one reference frame to the other and vice versa. The inviscid wall boundary condition, applied using halo cells, is hence

$$\begin{aligned} \rho_1 &= \rho_2 \\ \overline{U}_1 &= -\overline{U}_2 \\ \overline{V}_1 &= \overline{V}_2 \\ \overline{W}_1 &= \overline{W}_2 \\ p_1 &= p_2 \end{aligned} \quad , \quad \begin{aligned} \rho_0 &= \rho_3 \\ \overline{U}_0 &= -\overline{U}_3 \\ \overline{V}_0 &= \overline{V}_3 \\ \overline{W}_0 &= \overline{W}_3 \\ p_0 &= p_3 \end{aligned} \tag{5.20}$$

This is because the flow properties are stored at the cell centres but globally this ensures that the normal component of the velocity at the wall is zero. The turbulent viscosity at the wall has also to be zero hence to cancel out the turbulent viscosity in the cell number 2, the following is applied

$$(\mu_t)_1 = -(\mu_t)_2$$

However this introduces a non-physical value in the halo cells since the turbulent viscosity will be negative there. This should not matter since only the eddy viscosity at the face (which will be zero) is used.

5.6.3.2 Viscous wall boundary condition

The viscous wall boundary condition is applied to the halo cells next to a surface when the Navier-Stokes equations are solved. This is a no-slip boundary condition and hence the velocity at the wall is required to be zero. It is also an adiabatic boundary condition so pressure and density are transferred unchanged to the halo cells. This translates as

$$\begin{aligned} \rho_1 &= \rho_2 \\ u_1 &= -u_2 \\ v_1 &= -v_2 \\ w_1 &= -w_2 \\ p_1 &= p_2 \end{aligned} \quad , \quad \begin{aligned} \rho_0 &= \rho_3 \\ u_0 &= -u_3 \\ v_0 &= -v_3 \\ w_0 &= -w_3 \\ p_0 &= p_3 \end{aligned} \tag{5.21}$$

Here the boundary conditions have been applied directly to u, v and w in order to avoid any change of reference frame but can be equally applied to $\overline{U}$, $\overline{V}$ and $\overline{W}$. The boundary condition for the turbulent viscosity is the same as for an inviscid wall because μ_t also has to be zero at the wall so

$$(\mu_t)_1 = -(\mu_t)_2$$

To be precise it is necessary to add that although the boundary conditions for an inviscid and a viscous wall are applied in MERLIN as described above, the convective fluxes through the wall are specified explicitly and do not rely on the values inside the halo cells. This is done in order to get an accurate flux value that does not depend on the accuracy of the application of the boundary conditions in the halo cells and also because it is possible to get an explicit formulation for the wall flux.

5.6.3.3 Symmetry boundary condition

The symmetry boundary condition is applied when the symmetry of a problem is exploited so that only half of the domain needs to be solved. In this case the boundary condition is applied in the plane of symmetry of the problem which should be one side of the computational domain. Physically, there is a condition of symmetry for the velocity components parallel to this plane of symmetry and the velocity component normal to this plane should be set to zero. The application of this to the halo cells leads to the same formulae (5.20) as for the inviscid wall boundary condition although the physics is slightly different.

For the turbulent viscosity however, the boundary condition is different. The physics here is that there should be a continuity of μ_t across the boundary in the same way as there is a continuity of density and pressure. Hence

$$(\mu_t)_1 = (\mu_t)_2$$

5.6.3.4 Supersonic inflow boundary condition

The inflow and outflow boundary conditions are all based on one-dimensional wave propagation theory. In the case of a supersonic inflow boundary, all the information are coming from the exterior of the domain and entering it. Outside the domain, the flow properties are supposed to be the freestream conditions and hence the flow in the halo cells is set to a freestream value:

$$\begin{array}{rcl} \rho_1 = \rho_\infty & & \rho_0 = \rho_\infty \\ u_1 = u_\infty & & u_0 = u_\infty \\ v_1 = v_\infty & , & v_0 = v_\infty \\ w_1 = w_\infty & & w_0 = w_\infty \\ p_1 = p_\infty & & p_0 = p_\infty \end{array} \tag{5.22}$$

The freestream flow is assumed to be turbulence free hence

$$(\mu_t)_1 = 0$$

5.6.3.5 Supersonic outflow boundary condition

In the case of a supersonic outflow boundary all the information are coming from the interior of the domain and leaving it. Hence extrapolated boundary conditions are needed

$$\begin{array}{r} \rho_1 = \rho_2 \\ u_1 = u_2 \\ v_1 = v_2 \\ w_1 = w_2 \\ p_1 = p_2 \end{array} \quad , \quad \begin{array}{r} \rho_0 = \rho_2 \\ u_0 = u_2 \\ v_0 = v_2 \\ w_0 = w_2 \\ p_0 = p_2 \end{array} \tag{5.23}$$

Note that only cell 2 is involved in this case. A more sophisticated extrapolation using cell 3 could equally be applied. The extrapolation is also applied for the same reasons to the turbulent viscosity hence

$$(\mu_t)_1 = (\mu_t)_2$$

5.6.3.6 Farfield boundary condition

The boundary condition called here farfield boundary condition is applied to an exterior face when it is not known in advance if this boundary will be an inflow or an outflow and if the flow at this boundary will be subsonic or supersonic. This is typically the case when the freestream Mach number is subsonic.

The boundary condition implemented here makes use of Riemann invariants.[184, 186] These are defined as

$$R_+ = \overline{U} + \frac{2a}{\gamma - 1}$$

$$R_- = \overline{U} - \frac{2a}{\gamma - 1}$$

where a is the speed of sound. R_+ is associated with one-dimensional waves of speed $\overline{U} + a$ going out of the domain and is constant along C_+ characteristics. On the contrary R_- is associated with waves of speed $\overline{U} - a$ entering the domain and is constant along C_- characteristics. It is important to notice that here the vector normal to the boundary used to calculate the normal velocity $\overline{U}$, is pointing towards the exterior of the domain. In the opposite case, the minus and plus sign have to be inverted in R_+ and R_-.

Let us denote by the subscript e quantities that are extrapolated from the inside of the domain to the boundary b on one side of this boundary, while on the other side, at the exterior of the domain, freestream conditions ∞ exist. Along a C_+ characteristic, the following can be written

$$R_e = \overline{U}_e + \frac{2a_e}{\gamma - 1} = \overline{U}_b + \frac{2a_b}{\gamma - 1} \tag{5.24}$$

while along a C_- characteristic,

$$R_\infty = \overline{U}_\infty - \frac{2a_\infty}{\gamma - 1} = \overline{U}_b - \frac{2a_b}{\gamma - 1} \tag{5.25}$$

Combining equations (5.24) and (5.25) leads to

$$\overline{U}_b = \frac{1}{2}(R_e + R_\infty)$$

and

$$a_b = \frac{\gamma - 1}{4}(R_e - R_\infty)$$

Depending on the value of $\overline{U}_b$ with respect to a_b, the boundary is either an inflow or an outflow and is either supersonic or subsonic.

If $|\overline{U}_b| \leq a_b$, the boundary is subsonic and if:

- $\overline{U}_b \geq 0$, it is an outflow boundary and the other boundary properties are determined by

$$\overline{V}_b = \overline{V}_e$$
$$\overline{W}_b = \overline{W}_e$$
$$s_b = s_e$$

  where s is the entropy defined by

$$s = \frac{p}{\rho^\gamma}$$

- $\overline{U}_b < 0$, it is an inflow boundary and the other boundary properties are determined by

$$\overline{V}_b = \overline{V}_\infty$$
$$\overline{W}_b = \overline{W}_\infty$$
$$s_b = s_\infty$$

Knowing entropy and speed of sound at the boundary it is easy to determine density and pressure by

$$\rho_b = \left(\frac{{a_b}^2}{\gamma s_b}\right)^{\frac{1}{\gamma-1}}$$

and

$$p_b = \frac{\rho_b {a_b}^2}{\gamma}$$

This subsonic boundary condition is applied in MERLIN to the halo cells according to

$$\begin{array}{rl} \rho_1 = \rho_b & \rho_0 = 2\rho_1 - \rho_2 \\ \overline{U}_1 = \overline{U}_b & u_0 = 2u_1 - u_2 \\ \overline{V}_1 = \overline{V}_b \quad , & v_0 = 2v_1 - v_2 \\ \overline{W}_1 = \overline{W}_b & w_0 = 2w_1 - w_2 \\ p_1 = p_b & p_0 = 2p_1 - p_2 \end{array} \tag{5.26}$$

i.e. the first halo cell takes the boundary values while a linear extrapolation is carried out for the second halo cell.

If $|\overline{U}_b| > a_b$, the boundary is supersonic and if:

- $\overline{U}_b \geq 0$, it is an outflow boundary and conditions (5.23) are directly applied
- $\overline{U}_b < 0$, it is an inflow boundary and conditions (5.22) are directly applied

For the turbulent viscosity an approximation is made that will simplify the linearisation of the turbulence model in the adjoint solver. The approximation is that whatever the characteristic of the boundary i.e. subsonic or supersonic, inflow or outflow, an extrapolation is carried out for the value of μ_t in the halo cell that only relies on one cell inside the domain, hence

$$(\mu_t)_1 = (\mu_t)_2$$

Since at the farfield boundary the viscosity calculated with the Baldwin-Lomax model should be very small, the boundary condition applied is not too important and not very different from the inflow condition that would be $(\mu_t)_1 = 0$ or a proper linear extrapolation based on two cells inside the domain.

5.6.3.7 Interface boundary condition

Since MERLIN works with multiblock grids, an additional boundary condition is the one that treats the interface between two adjacent blocks. Since at each iteration, each block is dealt with separately as if it was a single block domain, MERLIN considers what is happening in an adjacent block as being exterior to its domain and hence a boundary condition is needed to ensure continuity between blocks from the point of view of the entire domain. The operation is mathematically simple since the values in the halo cells of one block are just set equal to the corresponding values in the cells adjacent to the interface of the next block. The schematic diagram in Figure 5.5 illustrates this principle for a particular choice of cell numbering. For this same numbering, the interface boundary condition applied to the halo cells of block 2 is

$$\begin{array}{rl} \rho_1^2 = \rho_{in}^1 & \rho_0^2 = \rho_{in-1}^1 \\ u_1^2 = u_{in}^1 & u_0^2 = u_{in-1}^1 \\ v_1^2 = v_{in}^1 \quad , & v_0^2 = v_{in-1}^1 \\ w_1^2 = w_{in}^1 & w_0^2 = w_{in-1}^1 \\ p_1^2 = p_{in}^1 & p_0^2 = p_{in-1}^1 \end{array}$$

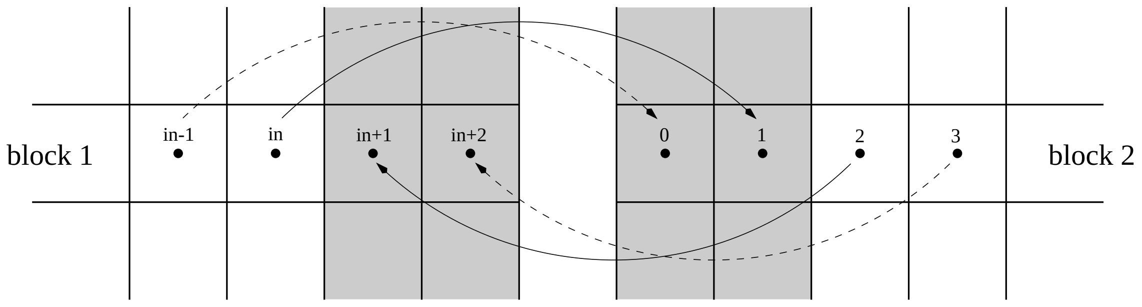


Figure 5.5: Schematic diagram for the interface boundary condition between two adjacent blocks for a particular choice of cell numbering.

where the superscript refers to the block number while the subscript refers to the cell number. For the turbulent viscosity the same treatment is applied and

$$(\mu_t)_1^2 = (\mu_t)_{in}^1$$

5.6.3.8 Corner points

What is termed corner point here is a halo cell that is situated at the intersection of two boundaries as depicted in Figure 5.6. Physically in a three-dimensional block, these kinds of cells are situated at the edges of the block. These cells are only used for the calculation of the diffusive terms. Clearly some kind of boundary conditions has to be applied to attribute them a value but since they are situated at the intersection of two boundaries, it is not obvious which of the two boundary conditions has to be applied or if an average of some sort has to be made. The type of treatment chosen should have little influence on the accuracy of the flow solver since it affects only a small number of cells and then only the diffusive terms of those cells. Nevertheless a choice has to be made and this will have some impact on the adjoint solver as we will see in the next chapter. However the choices detailed are very case-dependent, are limited to what is encountered in this thesis and cannot be generalised.

When one of the boundaries is an interface boundary, there is no choice, it has to be the boundary condition of the other boundary that needs to be applied. For this work, it has been decided that whenever one of the two boundaries is a symmetry boundary, it wins over the other and a symmetry boundary condition is applied. The grid topology and the cases used in this study ensure that a situation where two symmetry boundaries intersect at a corner point never happens. The other case encountered in this thesis is the intersection of two farfield boundary conditions. This happens at the intersection of the downstream plane boundary with the outer boundary in the case of a wing or an aircraft in a C-O type of grid. In this case the choice made is to do an interpolation in the streamwise direction (i direction in this case) from the two upstream halo cells to the corner point.

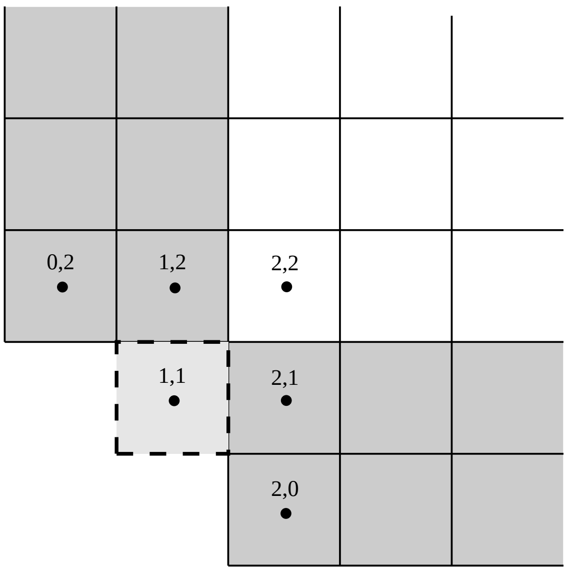


Figure 5.6: Corner point at the intersection of two boundaries: cell 1,1.

These are the only cases that will be found in this work.

The turbulent viscosity in a corner point is never actually used hence no boundary condition is applied, μ_t is left to the default value zero.

This concludes this subsection dedicated to the implementation of the boundary conditions. It also ends this long section describing the explicit methodology with first the calculation of the convective fluxes through the use of Osher's approximate Riemann solver. This was followed by the evaluation of the viscous fluxes with the laminar contributions and the description of the Baldwin-Lomax algebraic turbulence model. The next section details the other methodology issued from the temporal discretisation of the Navier-Stokes equations, i.e. the implicit formulation.

5.7 Implicit formulation

This section describes the implicit approach[187] already introduced in Section 5.5. It first presents the methodology used to solve the implicit equations, then examines the construction of the Jacobian with the contribution of the convective terms and of the diffusive terms. The contribution of the boundary conditions to the Jacobian is described at the end of this section.

5.7.1 Solution methodology

The use of an implicit method for the time discretisation requires solving the large linear system of equation (5.13) which can be put in the form

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{5.27}$$

In equation (5.13), the RHS $\mathbf{R}(\mathbf{Q}^n)$ is actually calculated in the same way as in the explicit method. It is only the LHS that is now needed and its calculation is detailed in this section.

Since the Navier-Stokes equations (5.1) are solved iteratively by the means of equations (5.13) and (5.14), equation (5.13) does not need to be solved exactly at each iteration. Hence approximations are allowed in the LHS of equation (5.13).

The first approximation concerns the Jacobian $\dfrac{\partial \mathbf{R}}{\partial \mathbf{P}}$ and this is described in the next two subsections. This is allowed since in such a linear system, it is only the RHS that carries the physics of the problem and will ensure that once the solution has converged, it has converged to the correct value. The LHS just serves to drive the solution towards zero in this case since for a converged steady-state

$$^n\Delta\mathbf{P} = \mathbf{0}$$

and $\mathbf{P}$ no longer evolves. To denote that the Jacobian is approximate, the symbol $\tilde{}$ is added and equation (5.13) is rewritten as

$$\left[\frac{1}{\Delta t}\frac{\partial \mathbf{Q}}{\partial \mathbf{P}} + \frac{\partial \widetilde{\mathbf{R}}(\mathbf{Q}^n)}{\partial \mathbf{P}}\right] {}^n\Delta\mathbf{P} = -\mathbf{R}(\mathbf{Q}^n) \tag{5.28}$$

The second approximation is that the linear system (5.28) does not need to be solved exactly for $^n\Delta\mathbf{P}$. In this work it is solved using an approximate direct inversion method, the Block Incomplete Lower-Upper decomposition with no fill-in or BILU(0). The block diagonal LHS matrix is approximated by

$$\mathbf{A} \approx \mathbf{L}\mathbf{U}$$

where $\mathbf{L}$ is a lower triangular matrix and $\mathbf{U}$ is an upper triangular matrix. It is an approximation since the matrices $\mathbf{L}$ and $\mathbf{U}$ keep the same diagonal structure as the original matrix $\mathbf{A}$ (no fill-in) i.e. 7 diagonals. The original system (5.27) is then inverted and

$$\mathbf{x} = \mathbf{U}^{-1}\mathbf{L}^{-1}\mathbf{b}$$

Since $\mathbf{L}$ and $\mathbf{U}$ are triangular, their inversion is much simpler than the inversion of the original matrix $\mathbf{A}$.

5.7.2 Contribution from the convective terms to the Jacobian

As was seen in equation (5.28), when using the implicit method for the time discretisation, the Jacobian matrix $\frac{\partial \widetilde{\mathbf{R}}}{\partial \mathbf{P}}$ is needed. This subsection explains how the convective terms are linearised analytically to form this Jacobian matrix.

The residual vector at a cell i is the sum of the 6 fluxes through each of the 6 faces of a 3D computational cell as was explained in equation (5.5). Hence

$$\frac{\partial \mathbf{R}_i}{\partial \mathbf{P}_j} = \sum_{faces} \frac{\partial [\mathbf{F}(\mathbf{Q}_i).\mathbf{n}S]}{\partial \mathbf{P}_j}$$

and the calculation of the derivative of the residual comes down to the calculation of the derivative of the fluxes. Since the convective fluxes themselves are the result of the application of Osher's approximate Riemann solver presented in subsection 5.6.1.1, their linearisation comes down to the linearisation of Osher's solver.

This is done by applying the chain rule of differentiation:[181, 183, 188]

$$\frac{\partial \mathbf{F}}{\partial \mathbf{P}_j} = \frac{\partial \mathbf{F}}{\partial \mathbf{Q}_L}\frac{\partial \mathbf{Q}_L}{\partial \mathbf{P}_L}\frac{\partial \mathbf{P}_L}{\partial \mathbf{P}_j} + \frac{\partial \mathbf{F}}{\partial \mathbf{Q}_R}\frac{\partial \mathbf{Q}_R}{\partial \mathbf{P}_R}\frac{\partial \mathbf{P}_R}{\partial \mathbf{P}_j} \tag{5.29}$$

where $\mathbf{Q}_L$ and $\mathbf{Q}_R$ are the left and right states used in the approximate Riemann solver. Note that the Riemann solver is applied to conservative variables, hence the need for the matrices $\frac{\partial \mathbf{Q}_L}{\partial \mathbf{P}_L}$ and $\frac{\partial \mathbf{Q}_R}{\partial \mathbf{P}_R}$.

The terms $\frac{\partial \mathbf{F}}{\partial \mathbf{Q}_L}$ and $\frac{\partial \mathbf{F}}{\partial \mathbf{Q}_R}$ involve the differentiation of Osher's scheme i.e the differentiation of the terms in Table 5.2. The resulting terms for all the cases are presented in Table 5.3 for $\frac{\partial \mathbf{F}}{\partial \mathbf{Q}_L}$ and in Table 5.4 for $\frac{\partial \mathbf{F}}{\partial \mathbf{Q}_R}$. This is a straight differentiation of Table 5.2. The fact that the sonic point $S0$ only depends on $\mathbf{Q}_0$ and that $S1$ only depends on $\mathbf{Q}_1$, reduces the number of possible combinations to 8 for both Jacobians. Spekreijse in Reference [181] provides all the matrix terms $\frac{\partial \mathbf{F}(\mathbf{Q})}{\partial \mathbf{Q}}$ as well as $\frac{\partial \mathbf{Q}_{S0}}{\partial \mathbf{Q}_0}$, $\frac{\partial \mathbf{Q}_{1/3}}{\partial \mathbf{Q}_0}$, etc. that appear in these tables. They are given for a two-dimensional problem in Cartesian coordinates but it is not difficult to derive the additional terms corresponding to a three-dimensional problem in a body-fitted reference frame.

The terms $\frac{\partial \mathbf{P}_L}{\partial \mathbf{P}_j}$ and $\frac{\partial \mathbf{P}_R}{\partial \mathbf{P}_j}$ involve the differentiation of the MUSCL scheme presented in subsection 5.6.1.2. It has been seen in the previous subsection 5.7.1 that approximations to the LHS Jacobian were allowed. The first one is that only a first order Jacobian is employed in MERLIN. This saves a lot of computational effort and memory, especially for 3D problems like in this work. Indeed the MUSCL scheme makes the value of $\mathbf{P}_L$

	$\overline{U}_0 - a_0 \geq 0$	$\overline{U}_0 - a_0 \leq 0$
$\overline{U}^* \geq 0$ $\overline{U}^* - a_{1/3} \geq 0$	$\dfrac{\partial \mathbf{F}(\mathbf{Q}_0)}{\partial \mathbf{Q}_0}$	$\dfrac{\partial \mathbf{F}(\mathbf{Q}_{S0})}{\partial \mathbf{Q}_{S0}} \dfrac{\partial \mathbf{Q}_{S0}}{\partial \mathbf{Q}_0}$
$\overline{U}^* \geq 0$ $\overline{U}^* - a_{1/3} \leq 0$	$\dfrac{\partial \mathbf{F}(\mathbf{Q}_0)}{\partial \mathbf{Q}_0} - \dfrac{\partial \mathbf{F}(\mathbf{Q}_{S0})}{\partial \mathbf{Q}_{S0}} \dfrac{\partial \mathbf{Q}_{S0}}{\partial \mathbf{Q}_0} + \dfrac{\partial \mathbf{F}(\mathbf{Q}_{1/3})}{\partial \mathbf{Q}_{1/3}} \dfrac{\partial \mathbf{Q}_{1/3}}{\partial \mathbf{Q}_0}$	$\dfrac{\partial \mathbf{F}(\mathbf{Q}_{1/3})}{\partial \mathbf{Q}_{1/3}} \dfrac{\partial \mathbf{Q}_{1/3}}{\partial \mathbf{Q}_0}$
$\overline{U}^* \leq 0$ $\overline{U}^* + a_{2/3} \geq 0$	$\dfrac{\partial \mathbf{F}(\mathbf{Q}_0)}{\partial \mathbf{Q}_0} - \dfrac{\partial \mathbf{F}(\mathbf{Q}_{S0})}{\partial \mathbf{Q}_{S0}} \dfrac{\partial \mathbf{Q}_{S0}}{\partial \mathbf{Q}_0} + \dfrac{\partial \mathbf{F}(\mathbf{Q}_{2/3})}{\partial \mathbf{Q}_{2/3}} \dfrac{\partial \mathbf{Q}_{2/3}}{\partial \mathbf{Q}_0}$	$\dfrac{\partial \mathbf{F}(\mathbf{Q}_{2/3})}{\partial \mathbf{Q}_{2/3}} \dfrac{\partial \mathbf{Q}_{2/3}}{\partial \mathbf{Q}_0}$
$\overline{U}^* \leq 0$ $\overline{U}^* + a_{2/3} \leq 0$	$\dfrac{\partial \mathbf{F}(\mathbf{Q}_0)}{\partial \mathbf{Q}_0} - \dfrac{\partial \mathbf{F}(\mathbf{Q}_{S0})}{\partial \mathbf{Q}_{S0}} \dfrac{\partial \mathbf{Q}_{S0}}{\partial \mathbf{Q}_0}$	$\mathbf{0}$

Table 5.3: Calculation of $\dfrac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{Q}_L}$. ($\mathbf{Q}_L \equiv \mathbf{Q}_0$ and $\mathbf{Q}_R \equiv \mathbf{Q}_1$)

	$\overline{U}_1 + a_1 \geq 0$	$\overline{U}_1 + a_1 \leq 0$
$\overline{U}^* \geq 0$ $\overline{U}^* - a_{1/3} \geq 0$	$\mathbf{0}$	$-\dfrac{\partial \mathbf{F}(\mathbf{Q}_{S1})}{\partial \mathbf{Q}_{S1}} \dfrac{\partial \mathbf{Q}_{S1}}{\partial \mathbf{Q}_1} + \dfrac{\partial \mathbf{F}(\mathbf{Q}_1)}{\partial \mathbf{Q}_1}$
$\overline{U}^* \geq 0$ $\overline{U}^* - a_{1/3} \leq 0$	$\dfrac{\partial \mathbf{F}(\mathbf{Q}_{1/3})}{\partial \mathbf{Q}_{1/3}} \dfrac{\partial \mathbf{Q}_{1/3}}{\partial \mathbf{Q}_1}$	$\dfrac{\partial \mathbf{F}(\mathbf{Q}_1)}{\partial \mathbf{Q}_1} + \dfrac{\partial \mathbf{F}(\mathbf{Q}_{1/3})}{\partial \mathbf{Q}_{1/3}} \dfrac{\partial \mathbf{Q}_{1/3}}{\partial \mathbf{Q}_1} - \dfrac{\partial \mathbf{F}(\mathbf{Q}_{S1})}{\partial \mathbf{Q}_{S1}} \dfrac{\partial \mathbf{Q}_{S1}}{\partial \mathbf{Q}_1}$
$\overline{U}^* \leq 0$ $\overline{U}^* + a_{2/3} \geq 0$	$\dfrac{\partial \mathbf{F}(\mathbf{Q}_{2/3})}{\partial \mathbf{Q}_{2/3}} \dfrac{\partial \mathbf{Q}_{2/3}}{\partial \mathbf{Q}_1}$	$\dfrac{\partial \mathbf{F}(\mathbf{Q}_{2/3})}{\partial \mathbf{Q}_{2/3}} \dfrac{\partial \mathbf{Q}_{2/3}}{\partial \mathbf{Q}_1} - \dfrac{\partial \mathbf{F}(\mathbf{Q}_{S1})}{\partial \mathbf{Q}_{S1}} \dfrac{\partial \mathbf{Q}_{S1}}{\partial \mathbf{Q}_1} + \dfrac{\partial \mathbf{F}(\mathbf{Q}_1)}{\partial \mathbf{Q}_1}$
$\overline{U}^* \leq 0$ $\overline{U}^* + a_{2/3} \leq 0$	$\dfrac{\partial \mathbf{F}(\mathbf{Q}_{S1})}{\partial \mathbf{Q}_{S1}} \dfrac{\partial \mathbf{Q}_{S1}}{\partial \mathbf{Q}_1}$	$\dfrac{\partial \mathbf{F}(\mathbf{Q}_1)}{\partial \mathbf{Q}_1}$

Table 5.4: Calculation of $\dfrac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{Q}_R}$. ($\mathbf{Q}_L \equiv \mathbf{Q}_0$ and $\mathbf{Q}_R \equiv \mathbf{Q}_1$)

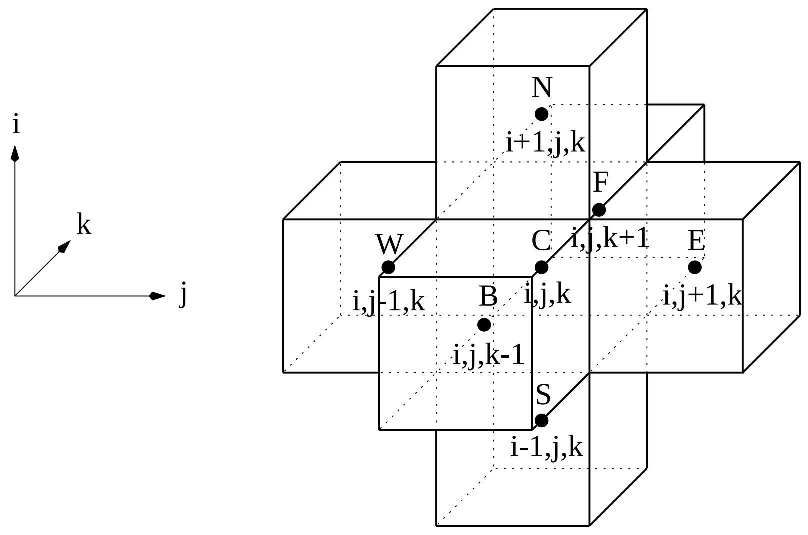


Figure 5.7: Computational stencil for a first-order inviscid Jacobian.

and $\mathbf{P}_R$ depend on the value of $\mathbf{P}$ inside 4 cells: 2 on one side of the face where the flux is calculated, 2 on the other side. Hence for a given flux $\mathbf{F}$, 4 contributions $\dfrac{\partial \mathbf{F}}{\partial \mathbf{P}_j}$ are non zero when using the MUSCL scheme and need to be stored in the Jacobian. When a first-order method is used, the flux only depends on the value of $\mathbf{P}$ in the two cells situated on each side of the face where the flux is calculated and hence the Jacobian is more sparse and requires less storage. Since only a first order Jacobian is employed in MERLIN, the terms $\dfrac{\partial \mathbf{P}_L}{\partial \mathbf{P}_j}$ and $\dfrac{\partial \mathbf{P}_R}{\partial \mathbf{P}_j}$ are straightforward to calculate.

The above presented the calculation of the flux Jacobian $\dfrac{\partial \mathbf{F}}{\partial \mathbf{P}_j}$ as a one-dimensional problem. As explained previously, the residual vector at a given computational cell is the sum of 6 fluxes, each of which depends on the flow value in two distinct cells for a first-order spatial accuracy. Hence the residual vector $\mathbf{R}_{i,j,k}$ at the cell i, j, k of a 3D problem depends on the flow value in the 7 cells shown in the stencil of Figure 5.7. Hence the Jacobian matrix $\dfrac{\partial \mathbf{R}}{\partial \mathbf{P}}$ is a block-banded matrix of 7 bands of blocks called in MERLIN $\mathbf{C}$, $\mathbf{W}$, $\mathbf{E}$, $\mathbf{S}$, $\mathbf{N}$, $\mathbf{B}$ and $\mathbf{F}$ with, according to the notation of Figure 5.7,

$$\mathbf{C}_{i,j,k} = \frac{\partial \mathbf{R}_{i,j,k}}{\partial \mathbf{P}_{i,j,k}}, \quad \mathbf{W}_{i,j,k} = \frac{\partial \mathbf{R}_{i,j,k}}{\partial \mathbf{P}_{i,j-1,k}}, \quad \mathbf{E}_{i,j,k} = \frac{\partial \mathbf{R}_{i,j,k}}{\partial \mathbf{P}_{i,j+1,k}}, \quad \ldots$$

Each matrix $\mathbf{C}_{i,j,k}$, $\mathbf{W}_{i,j,k}$, etc. is a 5×5 matrix.

This is only for the linearisation of the convective fluxes, the next subsection presents the treatment of the diffusive fluxes inside the Jacobian.

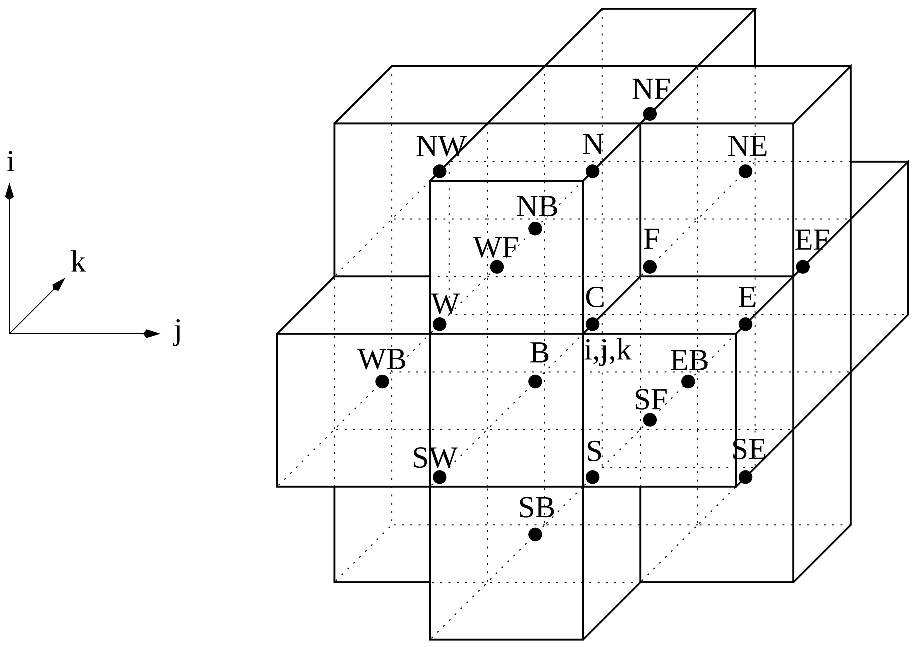


Figure 5.8: Computational stencil for the laminar part of the Jacobian.

5.7.3 Contribution from the diffusive terms to the Jacobian

The contribution from the diffusive terms is further divided into the contribution associated with the laminar viscous terms and the contribution of the turbulent part.

5.7.3.1 Laminar contributions

The calculation of the laminar viscous flux presented in subsection 5.6.2 has to be linearised and included in the Jacobian. This requires the linearisation of the application of Gauss' theorem but does not present any difficulty.

From subsection 5.6.2 and Figure 5.3, it is clear however that the laminar flux through one face depends on the value of $\mathbf{P}$ in 10 cells. Hence when adding them up in three dimensions to constitute the residual $\mathbf{R}$ for one cell i, j, k, you end up with the viscous laminar stencil shown in Figure 5.8 which contains 19 cells. This means that 19 terms $\dfrac{\partial \mathbf{R}_{i,j,k}}{\partial \mathbf{P}_l}$ are non-zero for each i, j, k. If this was to be stored in a Jacobian matrix, the resulting matrix would be huge. In MERLIN, we take into account the viscous laminar contributions that correspond to the inviscid first-order stencil of Figure 5.7 and to discard the other terms. This is the second approximation made to the Jacobian and is quite crude, but it maintains the sparsity of the Jacobian matrix with its 7 bands of blocks that is very convenient when doing the Lower-Upper decomposition and inversion presented in subsection 5.7.1. With a 19-band matrix, this would be much more difficult.

5.7.3.2 Turbulent contributions

The third approximation made to the LHS Jacobian of equation (5.28) is that the turbulent viscosity μ_t is frozen i.e. the contribution from the turbulence model is not linearised but whenever the fluid viscosity is needed, it is still composed of the molecular and turbulent viscosities. Indeed the turbulence model of Baldwin-Lomax presented in subsection 5.6.2.2 is quite difficult to linearise due to the fact that the eddy viscosity μ_t is calculated along rays that are normal to the wall. The value of a particular μ_t in one of these rays depends potentially on the value of the flow variables in all the cells along this ray. Indeed for the outer layer a search is made for F_{max} along the whole ray and the location of the cell where F_{max} is found cannot be known in advance. For the inner layer, the location of the change from inner to outer layer is not known a priori either, causing the same problem. This makes the eventual storage of a turbulent Jacobian into a banded matrix impossible. For these reasons, the linearisation of the eddy viscosity is not taken into account in the Jacobian in MERLIN, i.e. μ_t in the Baldwin-Lomax model has not been treated implicitly.

Now that the contributions to the Jacobian from the convective and the viscous terms have been presented, the next section details the implicit treatment of the boundary conditions presented in subsection 5.6.3.

5.7.4 Implicit boundary conditions

Close to a domain boundary the application of equation (5.29) involves the linearisation of the boundary conditions.[188] Indeed, with the cell numbering of Figure 5.4 Left, the flux across the boundary between cells 1 and 2 is

$$\mathbf{F} = \mathbf{F}[\mathbf{Q}_1(\mathbf{Q}_2), \mathbf{Q}_2]$$

and the application of equation (5.29) gives

$$\frac{\partial \mathbf{F}}{\partial \mathbf{P}_2} = \frac{\partial \mathbf{F}}{\partial \mathbf{Q}_1}\frac{\partial \mathbf{Q}_1}{\partial \mathbf{P}_1}\frac{\partial \mathbf{P}_1}{\partial \mathbf{P}_2} + \frac{\partial \mathbf{F}}{\partial \mathbf{Q}_2}\frac{\partial \mathbf{Q}_2}{\partial \mathbf{P}_2} \tag{5.30}$$

for a first order flux Jacobian. This implicit treatment of the boundary conditions is often neglected in CFD codes since, like a higher order Jacobian, it is not necessary to obtain a converged solution and this facilitates the implementation of an implicit method. However a consistent treatment of the boundary conditions should accelerate the rate of convergence of a solution and it will be seen later on, that it is needed in the adjoint solver. In addition it is possible to incorporate this linearisation of the boundary conditions without having to change the structure of the Jacobian. For these reasons, the implicit treatment of the boundary conditions is implemented consistently in MERLIN. The details of the matrices $\dfrac{\partial \mathbf{P}_1}{\partial \mathbf{P}_2}$ are given in Appendix A.

The only problem when looking at the different linearisations in Appendix A concerns the interface boundary condition. At a first glance it is right to say that the values in the halo cells for this boundary condition do not depend on what is happening inside the current block, since they have been copied from the neighbouring block. Hence a null matrix is acceptable. However when looking at equation (5.30), this implies that the flux at this boundary is only one-sided and this is clearly wrong since this type of boundary occurs usually in the middle of the domain where nothing special is happening to the flow. The flow solver should not be able to “see” this boundary since there is a continuity between the blocks but this is not happening with the chosen linearisation. We will see later for the adjoint solver that this boundary is in fact very difficult to implement accurately even if the theory behind it is easy to understand. Hence the present simplification for the LHS Jacobian is perfectly acceptable considering all the other approximations already made.

This treatment of the boundary conditions has to be applied not only to the convective fluxes but also to the laminar fluxes and here another approximation is made. This could seem a detail after having discarded more than half of the viscous contributions but has to be mentioned because it will also require a lot of work to be done correctly inside the adjoint solver. The implicit treatment of the boundary conditions for the viscous fluxes is only implemented in MERLIN in the direction of the flux and not in the other directions. For example if the flux is calculated in the i direction, the term

$$\mathbf{W}_{i,2,k} = \frac{\partial \mathbf{R}_{i,2,k}}{\partial \mathbf{P}_{i,1,k}}$$

is calculated as if inside the domain and then is discarded whereas a consistent treatment would account for the influence of $\mathbf{P}_{i,1,k}$ on to $\mathbf{P}_{i,2,k}$ and include that into the Jacobian. This however necessitates taking into account a lot of different cases so it is perfectly legitimate for the LHS Jacobian in MERLIN to neglect these implicit boundary conditions.

This concludes this subsection on the contributions from the diffusive fluxes to the Jacobian. It also terminates this long section describing the implicit method available in MERLIN that showed that the implicit approach is much more complicated mathematically than the explicit approach presented before that. This has to be traded with the computational time saving that the implicit method enables. Before ending this chapter on the flow solver MERLIN, the next section presents a test case of validation for this code i.e. the ONERA M6 wing.

5.8 Validation

The ONERA M6 wing is a classical test case for CFD flow solvers because of the extensive experimental data[167] available. A shape optimisation of the ONERA M6 wing will be presented in a later chapter but here only the flow solution around this geometry serves as a validation test case of the CFD code MERLIN.

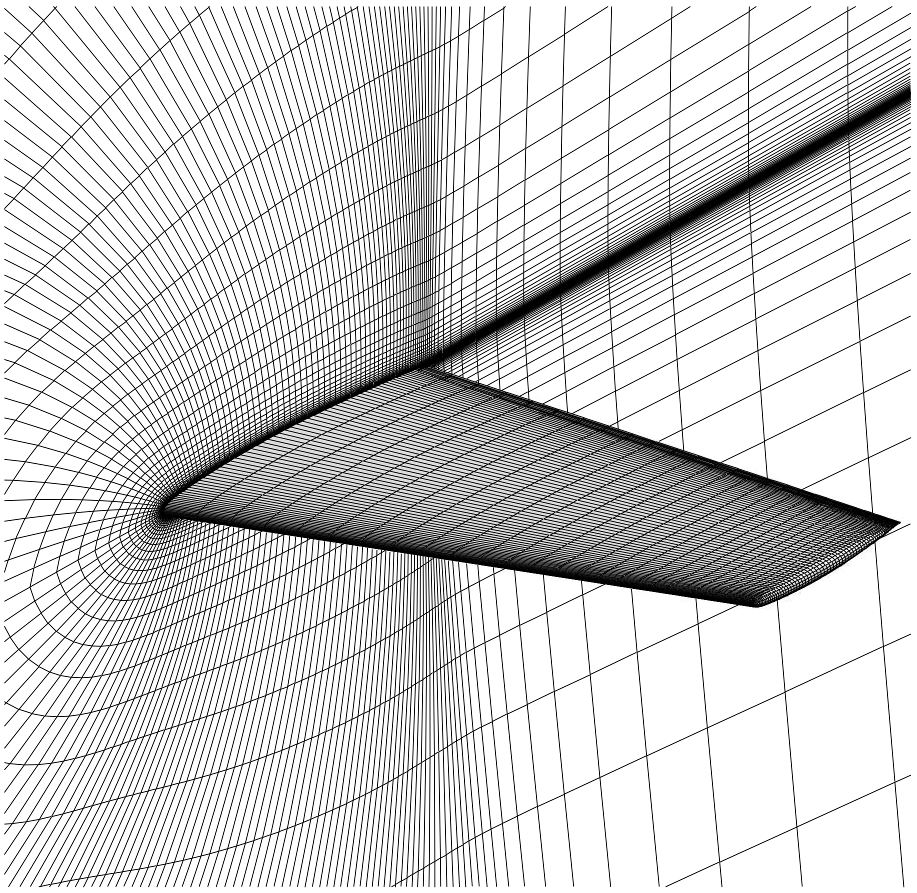


Figure 5.9: M6 wing computational grid.

The test case selected is the classical $3.06\,^\circ$ of incidence, Mach number $M_\infty = 0.84$ and Reynolds number $Re = 11.7 \times 10^6$ based on the mean aerodynamic chord. At these transonic conditions a lambda shock wave develops on the upper surface of the wing but the flow remains attached thus providing a good test case for the Baldwin-Lomax turbulence model. The grid employed for this validation test case is shown in Figure 5.9. It is a $193 \times 49 \times 33$ C-O type grid that was downloaded from Reference [189].

The contours of pressure coefficient C_p on the upper surface of the wing, obtained for this geometry, are shown in Figure 5.10. The characteristic lambda shock wave is clearly visible and the overall contours compare well with contour plots available in other studies[145, 162, 184, 190–192] that also solve the Navier-Stokes equations on this geometry and at these conditions.

Chordwise C_p distributions at stations where experimental data are available, are presented in Figure 5.11. Results obtained with MERLIN are compared to the experimental points that are plotted with errorbars. In the inner part of the wing (section $\eta = 0.20$ and

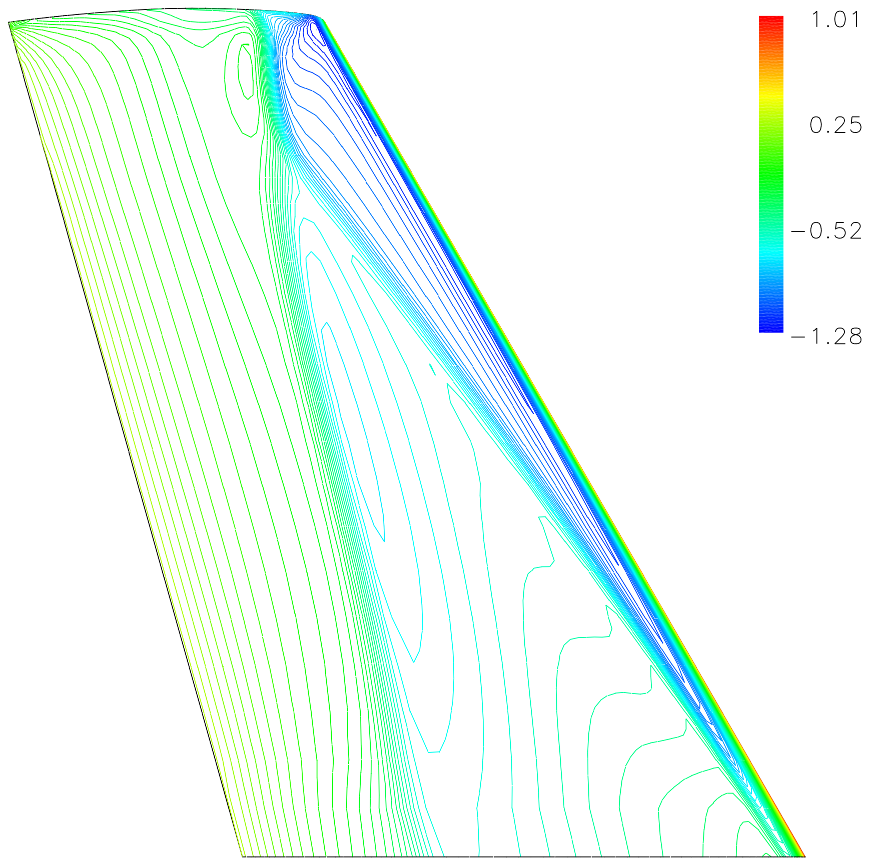
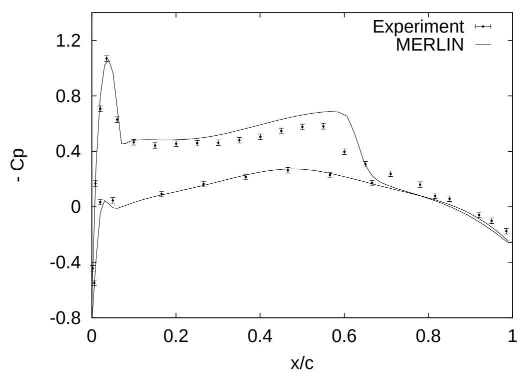


Figure 5.10: Contours of pressure coefficient on the upper surface of the ONERA M6 wing.

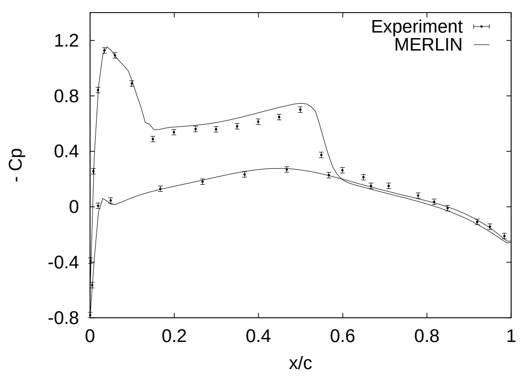
0.44), the first shock wave is correctly predicted in magnitude and position. The following plateau of C_p is overestimated and the second shock wave is not too well predicted. A fairly good agreement is obtained at the section $\eta = 0.65$ with a second shock wave very well predicted. The merging of the shock waves appears slightly too soon when looking at the station $\eta = 0.80$. In the remaining sections, the position and strength of the shock wave is very well predicted even though the trailing edge tip distribution is not captured (station $\eta = 0.99$). Overall the prediction is quite good when comparing to others' computations.[48, 184, 189, 192–199] Like the present results, none of the other studies manage to match the experimental data for all the spanwise stations.

A more challenging issue than the surface pressure distribution, is the calculation of overall aerodynamic coefficients. This is what the optimisation relies on so it has to be as accurate as possible in order to have some confidence in the optimum found by the optimisation process. Aerodynamic coefficients for the ONERA M6 wing are rare: they are not provided with the experimental data[167] and only a small number of studies mention them. The aerodynamic coefficients of the present study and these found in the literature by the author are presented in Table 5.5. The oldest values available are found in Reference [200] but the grid is very coarse and the contour plot and chordwise C_p distributions show that the computation did not capture the features of the flow solution, hence the overall aerodynamic coefficients might be questionable. The geometry of Reference [184] is slightly different from the one used for the MERLIN computation since the wing tip is straight and not rounded. This might have some influence on the resulting aerodynamic coefficients. In Reference [162] the Reynolds number reported is different from the one used in the experiment so the results might not be comparable. Since there is quite a disparity in the aerodynamic coefficients available in the literature, it is difficult to make a conclusion. The only possible comment is that MERLIN is within the range one would expect when comparing with the available data but this is still inconclusive regarding the accuracy of the force calculations.

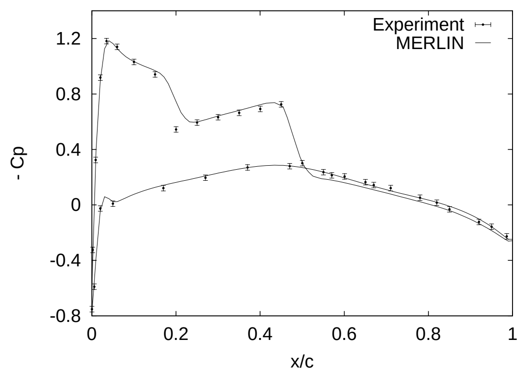
This concludes this part presenting some computational results on the ONERA M6 wing for validation purpose of the CFD code MERLIN employed in this study. This showed that MERLIN is accurate when looking at surface pressure distributions. However when considering overall aerodynamic coefficients, no definitive conclusion can be drawn due to the scarcity of comparison data although MERLIN gives reasonable answers. This also concludes the chapter that presented the fundamental equations used in the CFD code and how MERLIN solves them. It detailed the explicit method available as well as the more efficient yet more complicated implicit method. This implicit method is quite important since it will be the basis of the solution of the adjoint equations that are described in the next chapter.



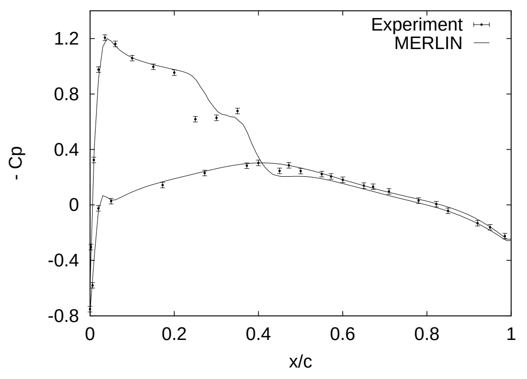
(a) $\eta = 0.20$



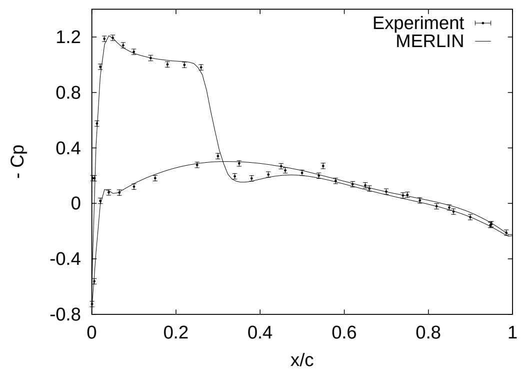
(b) $\eta = 0.44$



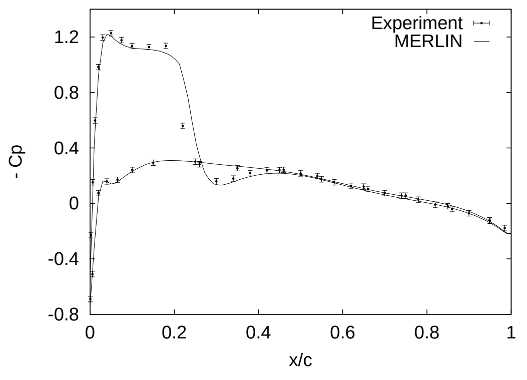
(c) $\eta = 0.65$



(d) $\eta = 0.80$

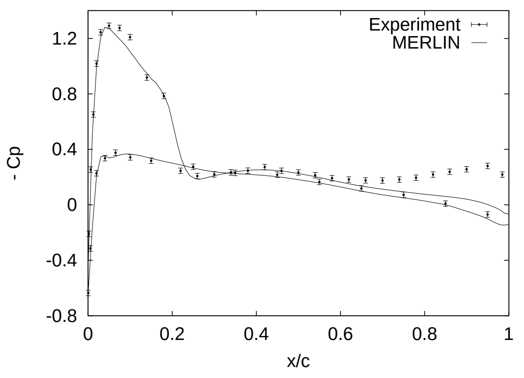


(e) $\eta = 0.90$



(f) $\eta = 0.95$

Figure 5.11: Chordwise C_p distributions for the ONERA M6 wing.



(g) $\eta = 0.99$

Figure 5.11: Chordwise C_p distributions for the ONERA M6 wing. (Concluded)

Origin of data	C_L	$C_{D\,total}$	$C_{D\,press}$	$C_{D\,fric}$
MERLIN	0.2697	0.01736	0.01241	0.00495
Müller and Rizzi[200]	0.2728	0.0157	N/A	N/A
Radespiel *et al*[197]	0.2677	0.01782	0.01261	0.00521
Vatsa *et al*[191]	$\approx$ 0.252	N/A	N/A	N/A
McNeil:[184] Central scheme	0.2764	0.01671	0.01400	0.00272
McNeil:[184] Upwind scheme	0.3113	0.02141	0.01596	0.00545
Nielsen and Anderson[162]	0.253	0.0168	N/A	N/A
Lee *et al*[190]	0.2622	0.01751	N/A	N/A

Table 5.5: Comparison of aerodynamic coefficients obtained with MERLIN and those available in the literature.

Chapter 6

Discrete adjoint solver

The aim of this chapter is to describe the discrete adjoint solver employed in the optimisation chain. The equations it solves are derived and the solution methodology is detailed. The first section of this chapter presents the discrete approach and its equations. Then the continuous approach is introduced before explaining the reasons for choosing the discrete rather than the continuous formulation in this work. Once the formulation of the equations has been chosen, the methodology used to solve them is presented and the innovative content of the present adjoint solver is described. This will introduce the next part dedicated to the calculation of the RHS Jacobian, starting from a simple first order inviscid flow to a fully turbulent flow. The last part of this chapter presents verification test cases to assess the accuracy of this adjoint solver.

6.1 Discrete adjoint method

The discrete adjoint method is very similar to the direct differentiation method presented in subsection 2.1.4.1 and started at approximately the same time. When people realised that it was much more efficient than the direct differentiation method for problems with a lot of design variables and few constraints, it became more popular. It is now widely employed. For unstructured grids, References [15, 79, 81–83] give some examples of discrete adjoint solvers for the Euler equations while solvers for the Navier-Stokes equations can be found in References [10, 11, 43, 48, 156, 162]. On structured grids, Euler discrete adjoint solvers are used in References [16, 74, 88–90, 125, 153, 154, 201, 202]. Turbulent Navier-Stokes discrete adjoint solvers are presented in References [19, 47, 164, 190] for structured grids, while in between, adjoint solvers for just the laminar viscous equations can be found in References [93, 95] and a turbulent adjoint solver with frozen turbulent viscosity in Reference [49].

The derivation of the discrete adjoint method starts like the direct differentiation method presented in subsection 2.1.4.1, except that now the derivative of the residual vector (equation (2.4)) is added to the derivative of the objective function (equation (2.2)) with the help of an adjoint vector $\boldsymbol{\lambda}$ to give

$$\frac{dF}{d\beta_k} = \left(\frac{\partial F}{\partial \mathbf{Q}}\right)^t \frac{d\mathbf{Q}^*}{d\beta_k} + \left(\frac{\partial F}{\partial \mathbf{X}}\right)^t \frac{d\mathbf{X}}{d\beta_k} + \frac{\partial F}{\partial \beta_k} + \boldsymbol{\lambda}^t \left(\frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\frac{d\mathbf{Q}^*}{d\beta_k} + \frac{\partial \mathbf{R}}{\partial \mathbf{X}}\frac{d\mathbf{X}}{d\beta_k} + \frac{\partial \mathbf{R}}{\partial \beta_k}\right)$$

which is rearranged as

$$\frac{dF}{d\beta_k} = \left[\left(\frac{\partial F}{\partial \mathbf{Q}}\right)^t + \boldsymbol{\lambda}^t\frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\right] \frac{d\mathbf{Q}^*}{d\beta_k} + \left[\left(\frac{\partial F}{\partial \mathbf{X}}\right)^t + \boldsymbol{\lambda}^t\frac{\partial \mathbf{R}}{\partial \mathbf{X}}\right] \frac{d\mathbf{X}}{d\beta_k} + \frac{\partial F}{\partial \beta_k} + \boldsymbol{\lambda}^t\frac{\partial \mathbf{R}}{\partial \beta_k}$$

The adjoint vector at this point is an undefined vector. To avoid having to calculate $\frac{d\mathbf{Q}^*}{d\beta_k}$ for each design variable as in the direct differentiation method, the term multiplying this quantity is set to zero to give the adjoint equation

$$\left(\frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\right)^t \boldsymbol{\lambda} = -\frac{\partial F}{\partial \mathbf{Q}} \tag{6.1}$$

Once this adjoint equation has been solved for $\boldsymbol{\lambda}$, the sensitivity derivative is calculated by

$$\frac{dF}{d\beta_k} = \left[\left(\frac{\partial F}{\partial \mathbf{X}}\right)^t + \boldsymbol{\lambda}^t\frac{\partial \mathbf{R}}{\partial \mathbf{X}}\right] \frac{d\mathbf{X}}{d\beta_k} + \frac{\partial F}{\partial \beta_k} + \boldsymbol{\lambda}^t\frac{\partial \mathbf{R}}{\partial \beta_k} \tag{6.2}$$

As was already explained, the adjoint equation (6.1) needs to be solved only once for each of the aerodynamic functions for which the sensitivity derivatives are needed. This usually refers to the aerodynamic objective function and any aerodynamic constraint function but not to geometric constraint functions that have nothing to do with the aerodynamic flow vector $\mathbf{Q}$ and for which analytical expressions (for the functions and their gradients) have to be found. Hence for a problem where the number of design variables is greater than the number of objective functions and aerodynamic constraints, which is the case in most aerodynamic shape optimisation problems, the adjoint method is more efficient than the direct differentiation method as was stated in Chapter 2.

Presented in the form of equation (6.1), the adjoint equation is shown in the discrete formulation since the residual vector $\mathbf{R}$ has been discretised as in equation (5.5) and the Jacobian $\frac{\partial \mathbf{R}}{\partial \mathbf{Q}}$ is of the same form as in equation (5.12). The continuous formulation presented in the next section differentiates the objective function and the Navier-Stokes equations before discretising them as is explained next.

6.2 Continuous adjoint method

The continuous adjoint method for aerodynamic optimisation was pioneered by Pironneau[203,204] in the 1970's but due to limitation in numerical methods, his work was only theoretical. The continuous adjoint method is based on the calculus of variation which was later used in simple problems such as heat transfer optimisation problems[205,206] or problems of pollutant diffusion in the atmosphere.[207] Koda also applied a variational method to an aerodynamic problem[208,209] but still without numerical implementation.

Jameson was the first to apply this technique successfully to an aerodynamic optimisation problem. This technique has gained a lot of popularity since then thanks to his contribution. His early work[4] was limited to the optimisation of 2D aerofoils using potential flow and conformal mapping. It then evolved to conventional computational grids,[149] Euler equations[140,148] and 3D optimisation.[138] Further improvements included the use of multiblock grids,[137,150,165] parallel computing[27,139,165,210] and multipoint optimisation.[139] More recently, Jameson and his colleagues applied the continuous adjoint equation to the Navier-Stokes equations.[45,137,211,212] They freeze the turbulent viscosity in the adjoint solver and use this latter with a turbulent Navier-Stokes flow solution.

The continuous adjoint method is now widely employed: Euler continuous adjoint solvers for unstructured grids can be found in References [25,152] and viscous laminar solvers in References [157,159,160]. On structured grids, References [18,46,145,146,158,213–218] give some examples of the continuous adjoint method for the Euler equations. Going a little further into the complexity of the physics, Cross[12] and Szmelter[13] use an Euler adjoint with the Euler equations with viscous/inviscid interaction for the flow solution. A continuous laminar adjoint solver is described in Reference [219] for a laminar flow solver. An continuous adjoint solver with frozen turbulence is used, as Jameson does, with a turbulent CFD code in Reference [147]. A truly turbulent continuous adjoint solver with linearisation of the turbulent viscosity can be found in References [17, 220]. Continuous adjoint solvers are not limited to aerodynamics and can be employed in hydrodynamics[142] and ship hull shape optimisation.[221]

To describe the derivation of the continuous method, we follow the work of Iollo *et al.*[222–225] This derivation is based on the two-dimensional steady-state Euler equations applied to an aerofoil to keep things as simple as possible. Soemarwoto gives a very detailed derivation of this method for the 2D Navier-Stokes equations in Reference [220].

The 2D Euler equations are written in the form

$$\frac{\partial(\mathbf{AQ})}{\partial x} + \frac{\partial(\mathbf{BQ})}{\partial y} \equiv (\mathbf{AQ})_x + (\mathbf{BQ})_y = \mathbf{0}$$

in the flow domain Ω where $\mathbf{A} = \dfrac{\partial \mathbf{F}^i}{\partial \mathbf{Q}}$ and $\mathbf{B} = \dfrac{\partial \mathbf{G}^i}{\partial \mathbf{Q}}$ are the Jacobian matrices, with the corresponding boundary conditions

$$\rho \mathbf{V}.\mathbf{n} = 0$$

on the surface Γ of the aerofoil. $(\mathbf{V} = (u, v))$

A particular objective function I is chosen for this optimisation problem

$$I = \frac{1}{2} \oint_{\Gamma} [p(\Gamma) - p^{\star}]^2 \, ds$$

and corresponds to an inverse design problem where the pressure p on the aerofoil surface has to match a target pressure $p^{\star}$.

A Lagrangian functional is created to augment this objective function:

$$\begin{aligned} L(\mathbf{Q}, \Gamma, \boldsymbol{\lambda}, \mu) &= I + \int_{\Omega} \boldsymbol{\lambda}^t [(\mathbf{AQ})_x + (\mathbf{BQ})_y] d\Omega + \int_{\Gamma} \mu\rho \mathbf{V}.\mathbf{n} ds \\ &= I + \int_{\Gamma} \boldsymbol{\lambda}^t (\mathbf{AQ} n_x + \mathbf{BQ} n_y) ds - \int_{\Omega} (\boldsymbol{\lambda}_x{}^t \mathbf{AQ} + \boldsymbol{\lambda}_y{}^t \mathbf{BQ}) d\Omega + \int_{\Gamma} \mu\rho \mathbf{V}.\mathbf{n} ds \end{aligned}$$

after integration by parts of the middle term. The vector $\boldsymbol{\lambda}$ ($\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \lambda_3, \lambda_4)^t$ in 2D) in Ω and the scalar μ on Γ are the equivalent of Lagrange multipliers. The minimisation of L is equivalent to the minimisation of I.

Applying the method of calculus of variation to L gives:

$$\delta L = \delta L_{\mathbf{Q}} + \delta L_{\Gamma} + \delta L_{\boldsymbol{\lambda}} + \delta L_{\mu}$$

with

$$\begin{aligned} \delta L_{\mathbf{Q}} &= \oint_{\Gamma} \frac{\partial p}{\partial \mathbf{Q}} [p(\Gamma) - p^{\star}] \widetilde{\mathbf{Q}} ds + \int_{\Gamma} \boldsymbol{\lambda}^t (\mathbf{A} n_x + \mathbf{B} n_y) \widetilde{\mathbf{Q}} ds - \int_{\Omega} (\boldsymbol{\lambda}_x{}^t \mathbf{A} + \boldsymbol{\lambda}_y{}^t \mathbf{B}) \widetilde{\mathbf{Q}} d\Omega \\ &+ \int_{\Gamma} \mu \mathbf{n} \frac{\partial \rho \mathbf{V}}{\partial \mathbf{Q}} \widetilde{\mathbf{Q}} ds \\ \delta L_{\Gamma} &= \oint_{\Gamma} \frac{\partial p}{\partial \mathbf{n}} [p(\Gamma) - p^{\star}] \widetilde{\mathbf{n}} ds + \int_{\Gamma} \boldsymbol{\lambda}^t [(\mathbf{AQ})_x + (\mathbf{BQ})_y] \widetilde{\mathbf{n}} ds - \int_{\Gamma} \mu\rho \mathbf{V}.\widetilde{\mathbf{n}} ds \\ &+ \int_{\Gamma} \mu \frac{\partial \rho \mathbf{V}}{\partial \mathbf{n}}.\mathbf{n}\widetilde{\mathbf{n}} ds \\ \delta L_{\boldsymbol{\lambda}} &= \int_{\Omega} \widetilde{\boldsymbol{\lambda}}^t [(\mathbf{AQ})_x + (\mathbf{BQ})_y] d\Omega \\ \delta L_{\mu} &= \oint_{\Gamma} \widetilde{\mu} \rho \mathbf{V}.\mathbf{n} ds \end{aligned}$$

This corresponds to a displacement $\varepsilon\widetilde{\mathbf{n}}$ of each point on the aerofoil surface in the normal direction $\mathbf{n}$ which generates an increase $\varepsilon\widetilde{\mathbf{Q}}$ in $\mathbf{Q}$, $\varepsilon\widetilde{\boldsymbol{\lambda}}$ in $\boldsymbol{\lambda}$ and $\varepsilon\widetilde{\mu}$ in μ. $\delta L_{\mathbf{Q}}$ is the variation of L due to the variation of $\mathbf{Q}$ when all the other variables are kept fixed and similarly for the other components of δL.

Two equivalent approaches are then possible. The first one consists in saying that at the optimal design point, δL must be equal to zero whatever $\widetilde{\mathbf{n}}$, $\widetilde{\mathbf{Q}}$, $\widetilde{\boldsymbol{\lambda}}$ and $\widetilde{\mu}$, hence $\delta L_{\mathbf{Q}} =$

$\delta L_\Gamma = \delta L_{\boldsymbol{\lambda}} = \delta L_\mu = 0$. The condition $\delta L_{\boldsymbol{\lambda}} = 0$ leads to the satisfaction of the Euler equations $(\mathbf{AQ})_x + (\mathbf{BQ})_y = \mathbf{0}$ in Ω. The condition $\delta L_\mu = 0$ gives the boundary conditions $\rho\mathbf{V}.\mathbf{n} = 0$ on Γ. The condition $\delta L_{\mathbf{Q}} = 0$ gives the continuous adjoint equation

$$\mathbf{A}^t\boldsymbol{\lambda}_x + \mathbf{B}^t\boldsymbol{\lambda}_y = \mathbf{0} \tag{6.3}$$

in Ω followed by the boundary conditions for the adjoint vector on Γ:

$$\frac{\partial p}{\partial \mathbf{Q}}\left[p(\Gamma) - p^\star\right] + \boldsymbol{\lambda}^t(\mathbf{A}n_x + \mathbf{B}n_y) + \mu\mathbf{n}\frac{\partial \rho\mathbf{V}}{\partial \mathbf{Q}} = 0 \tag{6.4}$$

The remaining condition $\delta L_\Gamma = 0$ is the condition of optimality. In gradient-based optimisation, it is not used directly but rather serves to compute the sensitivity derivatives $\dfrac{\partial L}{\partial \Gamma_k}$ once $\mathbf{Q}$, $\boldsymbol{\lambda}$ and μ have been calculated.

The other equivalent approach consists in saying that since the Euler equations and boundary conditions are satisfied then $\delta L_{\boldsymbol{\lambda}} = 0$ and $\delta L_\mu = 0$. Hence $\delta L = \delta L_{\mathbf{Q}} + \delta L_\Gamma$. To further simplify this, the adjoint vector $\boldsymbol{\lambda}$ and the scalar μ are required to satisfy the adjoint equation and boundary conditions leaving $\delta L = \delta L_\Gamma$ which contains the desired sensitivity derivatives.

The resulting adjoint equation (6.3) is the continuous version corresponding to the discrete one in equation (6.1). This continuous equation is very similar to the flow equation and hence can be discretised in the same way making use of the same numerical routines developed for the flow solver. This is not however an obligation and any method of discretisation is valid. It should be noted that boundary conditions for the adjoint vector are explicitly appearing in the continuous method while they are hidden in the discrete method. Hence it is in fact an entirely new code similar to the flow solver that has to be written to solve the adjoint equation and its boundary conditions.

Furthermore not any objective function can be considered and it was thought for a long time that only "admissible" cost functions i.e. functions that behave "well" in the method of calculus of variation, could be used. For the Euler equations, this meant only functions depending directly on pressure. However Arian and Salas[226, 227] found that by adding extra terms to the Lagrangian function, this problem can be avoided and that any objective function might be used.

6.3 Choice between the continuous and discrete formulations

The continuous and discrete adjoint methods are in fact very close to each other and for example, Reference [157] mixes both methods and it is not easy to recognise which is which. It is hence very difficult to choose which one to implement.

The literature does not help a lot in this matter either. Indeed numerical comparisons made by Nadarajah and Jameson[44,56] found that the continuous and discrete methods give very similar results in terms of performance and accuracy of the sensitivity derivatives. The discrete method is supposed to give the exact numerical gradient of the objective function i.e. the gradient that would be obtained by finite difference whatever the grid refinement and hence is more consistent with the flow solution than the continuous method.[44,56,58,125,157] Although the continuous adjoint method is less consistent, as the mesh size is increased, its agreement with the finite-difference method increases.[44]As pointed out by Anderson and Venkatakrishnan,[157] the fact that the objective function explicitly appears in the boundary conditions of the continuous method can be a problem: the derivation of these boundary conditions is mathematically demanding and has to be done over again for any new objective function while it is much simpler to change a subroutine that adds $\dfrac{\partial F}{\partial \mathbf{Q}}$ to the RHS of the adjoint equations in the discrete method. There is furthermore an issue with shock waves for the continuous method: Giles and Pierce[228] showed that in theory an adjoint boundary condition has to be applied along the shock wave. This would be very complicated to implement since the location of the shock wave would have to be determined inside the flow field. In practice, none of this is done and results by Jameson and others using the continuous method do not seem to suffer from this inconsistency. Giles and Pierce[228] have indeed checked that the continuous method behaved numerically well at the shock for the quasi-1D Euler equations and that there is no need in practice to enforce an internal boundary condition. There is still however some uncertainty about this mathematical problem and it favours the discrete method that does not appear to have such problem.

The discrete adjoint method is however supposed to require more computing memory than the continuous one.[44,56,58,157] Giles and Pierce[58] nevertheless point out that the discrete adjoint method is almost straightforward to implement since a lot of its routines are taken from the flow solver. They conclude their comparison by saying that in fact none of the two methods has a true advantage compared to the other and that the choice between them is a “matter of personal taste” [1].

Hence the choice of the discrete adjoint method made in this study relies on personal reasons. The first one was that the CFD code MERLIN was available and with it all the implicit method detailed in section 5.7. In particular, the calculation of the LHS Jacobian was already existing with a consistent treatment of the boundary conditions. However it is noted that Hiernaux and Essers[159,160,229] have an implicit flow solver with a LHS Jacobian and nevertheless chose to develop a continuous adjoint method. The second reason was that the discrete method as it is pointed out by Giles and Pierce,[58] seemed rather straightforward to understand and implement to the author, while the continuous method

[1]Reference [58], p. 409

is mathematically a lot more complicated especially when dealing with viscous flows.

6.4 Solution methodology

This section explains how equation (6.1) is solved. The first element is that equation (6.1) has been derived in section 6.1 considering the vector of conservative variables $\mathbf{Q}$ as it is usually presented. It can be transformed to use primitive variables $\mathbf{P}$ without changing the nature of the adjoint vector. This is what is chosen for the remainder of the thesis and the discrete adjoint equation (6.1) becomes

$$\left(\frac{\partial \mathbf{R}}{\partial \mathbf{P}}\right)^t \boldsymbol{\lambda} = -\frac{\partial F}{\partial \mathbf{P}} \tag{6.5}$$

As it has just been explained, the advantage of the discrete adjoint method is that the same solution methodology as for the flow analysis can be applied to the adjoint equation. However written in the form of equation (6.5), the adjoint equation is a linear equation that needs to be solved exactly. The Jacobian $\dfrac{\partial \mathbf{R}}{\partial \mathbf{P}}$ for example needs to be the exact higher-order Jacobian that takes into account the boundary conditions and all the laminar and turbulent contributions in order to find the correct value of the adjoint vector. It was explained in subsection 5.7 that this would require a lot of computing memory and was one of the reasons why a simplified Jacobian was employed for the flow solver. To be able to use the same Jacobian as in the flow solver, the adjoint equation is written in the incremental form

$$\left[\frac{\partial \widetilde{\mathbf{R}}(\mathbf{Q}^*)}{\partial \mathbf{P}}\right]^t {}^n\Delta\boldsymbol{\lambda} = -\left\{\frac{\partial F}{\partial \mathbf{P}} + \left[\frac{\partial \mathbf{R}(\mathbf{Q}^*)}{\partial \mathbf{P}}\right]^t \boldsymbol{\lambda}^n\right\} \tag{6.6}$$

with

$$\boldsymbol{\lambda}^{n+1} = \boldsymbol{\lambda}^n + {}^n\Delta\boldsymbol{\lambda}$$

and is solved iteratively. The incremental iterative form is found in the literature, primarily applied to the direct differentiation method detailed in Chapter 2[28,41,49,51,69,70,74,81,82,88] but also used in the discrete adjoint method.[16,48,49,51,70,74,77,88,162]

In equation (6.6), it is the RHS that carries the physics of the equation and ensures that the solution is correct while the LHS is only here to drive ${}^n\Delta\boldsymbol{\lambda}$ to zero. Hence as for the flow equation (5.12), some approximations are allowed for the Jacobian of this LHS. That is why the symbol $\widetilde{\ }$ is used. In the present work even a fictitious time term, the same as in the flow solver, is added to the LHS to improve diagonal dominance, if necessary, at the beginning of the iterations. Equation (6.6) becomes then

$$\left[\frac{1}{\Delta t}\frac{\partial \mathbf{Q}}{\partial \mathbf{P}} + \frac{\partial \widetilde{\mathbf{R}}(\mathbf{Q}^*)}{\partial \mathbf{P}}\right]^t {}^n\Delta\boldsymbol{\lambda} = -\left\{\frac{\partial F}{\partial \mathbf{P}} + \left[\frac{\partial \mathbf{R}(\mathbf{Q}^*)}{\partial \mathbf{P}}\right]^t \boldsymbol{\lambda}^n\right\} \tag{6.7}$$

With this modification, the LHS matrix has the same form as in equation (5.28) for the flow solver and in this study, the LHS matrix of the adjoint equation is exactly the same as in the flow solver before applying the transpose operation. The original first-order Jacobian in MERLIN is a matrix of 7 diagonals of blocks called $\mathbf{C}$, $\mathbf{W}$, $\mathbf{E}$, $\mathbf{S}$, $\mathbf{N}$, $\mathbf{B}$ and $\mathbf{F}$ as was explained in section 5.7. Each matrix $\mathbf{C}_{i,j,k}$, $\mathbf{W}_{i,j,k}$, etc. is a 5×5 matrix. The operation of transposing this matrix does not change its structure, only the ordering of the matrices is changed as follows:

$$\begin{array}{lcl}
\mathbf{C}_{i,j,k} & \rightarrow & \mathbf{C}^{\,t}_{i,j,k} \\
\mathbf{W}_{i,j,k} & \rightarrow & \mathbf{E}^{\,t}_{i,j-1,k} \\
\mathbf{E}_{i,j,k} & \rightarrow & \mathbf{W}^{\,t}_{i,j+1,k} \\
\mathbf{N}_{i,j,k} & \rightarrow & \mathbf{S}^{\,t}_{i+1,j,k} \\
\mathbf{S}_{i,j,k} & \rightarrow & \mathbf{N}^{\,t}_{i-1,j,k} \\
\mathbf{F}_{i,j,k} & \rightarrow & \mathbf{B}^{\,t}_{i,j,k+1} \\
\mathbf{B}_{i,j,k} & \rightarrow & \mathbf{F}^{\,t}_{i,j,k-1}
\end{array}$$

where the “$\rightarrow$” has to be read by “is replaced by”. Hence when the original Jacobian has been calculated, it is relatively easy to transpose it and keep the same structure and name for the variables in the computer program.

For the RHS Jacobian of equation (6.7) however no approximation is allowed since the RHS of this equation has to be exact. Hence the full higher order Jacobian has still to be calculated for this RHS. However it does not need to be stored since it is multiplied by the adjoint vector. In this work this multiplication is done term by term to constitute the RHS vector, saving some computing memory and some unnecessary computing time spent on very large matrix multiplications. All of this is detailed later in this chapter.

Equation (6.7) has exactly the same structure as equation (5.28) i.e. the form $\mathbf{Ax} = \mathbf{b}$ of equation (5.27) with the same banded matrix $\mathbf{A}$. Hence the same solution methodology is employed to solve the adjoint equations i.e. an approximate direct inversion method with a BILU(0) technique. To save some computing time, the terms of equation (6.7) that do not change at each iteration, are calculated once at the beginning of the computation and are then stored for the rest of the iterations. This concerns the LHS $\dfrac{\partial \widetilde{\mathbf{R}}(\mathbf{Q}^*)}{\partial \mathbf{P}}$ and $\dfrac{\partial F}{\partial \mathbf{P}}$. In the LHS Jacobian, the time-term depends on Δt that changes at each iteration depending on the value of the total residual as in the flow solver. For the RHS term, as said above a term by term multiplication is performed and since the adjoint vector $\boldsymbol{\lambda}^n$ changes at each iteration, nothing can be stored. Besides this was the advantage of the incremental iterative method of not having to store an exact Jacobian. If this solves an eventual problem of storage and memory, this is at the cost of computing time since the components of the RHS Jacobian have to be recalculated at each iteration. Nevertheless the resulting adjoint solver is quite efficient with a turbulent two-dimensional test showing a cost per iteration equal to 2.7 times that of the flow solver. The same test performed with a LHS Jacobian and $\frac{\partial F}{\partial \mathbf{P}}$ not stored but recalculated at each iteration, gave a ratio of 4.6. This clearly shows the interest of storing these terms.

The sensitivity equation (6.2) is rewritten in the form

$$\frac{dF}{d\beta_k} = \left(\frac{\partial F}{\partial \mathbf{X}}\right)^t \frac{d\mathbf{X}}{d\beta_k} + \boldsymbol{\lambda}^t \frac{\partial \mathbf{R}}{\partial \mathbf{X}} \frac{d\mathbf{X}}{d\beta_k} \tag{6.8}$$

because for a pure aerodynamic shape optimisation, the design variables β_k only influence the flow field solution and objective function through the computational grid variations. Hence there is no $\frac{\partial F}{\partial \beta_k}$ or $\frac{\partial \mathbf{R}}{\partial \beta_k}$ term.

Furthermore equation (6.8) better represents what the computer code does. Instead of having a first differentiation with respect to the coordinates of the grid points and then a multiplication between the resulting matrix and the grid sensitivity matrix $\frac{d\mathbf{X}}{d\beta_k}$ as would be suggested by equation (6.2), the differentiation is done directly in the code by the use of the chain rule. For example the term $\frac{\partial \mathbf{R}}{\partial \mathbf{X}} \frac{d\mathbf{X}}{d\beta_k}$ is calculated directly in the form of $\frac{\partial \mathbf{R}}{\partial(\boldsymbol{\xi}, \boldsymbol{\eta}, \boldsymbol{\zeta})} \frac{\partial(\boldsymbol{\xi}, \boldsymbol{\eta}, \boldsymbol{\zeta})}{\partial \mathbf{X}} \frac{d\mathbf{X}}{d\beta_k}$ where $(\boldsymbol{\xi}, \boldsymbol{\eta}, \boldsymbol{\zeta})$ are the metric terms involved in the calculation of the residual vector. This analytical differentiation does not involve any multiplication and is quite straightforward since all the terms that have to be differentiated are already present in the flow solver and can be differentiated one by one as they appear inside the code. It is similar to applying Automatic Differentiation in forward mode, but by hand, and does not present any difficulty. This linearisation is done consistently each time the metric terms or a unit normal vector are used. To be more specific, this includes the normal vector in Osher's scheme, the use of Gauss' theorem for the diffusive terms including in the Baldwin-Lomax model and any other time when a transformation from a Cartesian to a body-fitted reference frame is needed.

6.5 Innovative content in this adjoint solver

Now that the adjoint solver used in this work has been presented in its globality (more details about the RHS Jacobian are to be explained in the next section), it is time to explain what makes this adjoint solver special and different from any other existing adjoint solvers.

The first point to mention is that this adjoint solver has been entirely differentiated by hand: no use of Automatic Differentiation, finite difference or complex variable method has been made to calculate the components of the Jacobians or the grid sensitivities.

Secondly, when people think about the adjoint method for aerodynamic optimisation, they always think about Jameson's work so the first obvious difference between this adjoint solver and Jameson's work that can be included into a wider continous adjoint community, is that it employs the discrete method as was already explained.

Among the existing discrete adjoint solvers, this one is, to the author's knowledge, the only one that uses Osher's approximate Riemann solver to calculate the inviscid fluxes. This scheme has been designed to calculate accurately flow discontinuities such as shock waves and shear layers.[230] Since shock waves are a characteristic and important feature of the transonic flow around the wings or aircraft we are interested in optimising, it is considered to be an advantage to have Osher's scheme in the adjoint solver rather than a more diffusive scheme that might miscalculate the shock wave, the strength of which you are trying to minimise through the optimisation process to reduce the wave drag.

The other important aspect is that this discrete adjoint solver has been derived for the fully turbulent Navier-Stokes equations. Thus compared to the ten simpler inviscid adjoint solvers mentioned in section 6.1, the physical phenomena of viscosity and turbulence are added. Since this flow physics is again important for the kind of problems we are interested in, this is an advantage. Incorporating accurately the turbulent part in the adjoint solver is also considered to be better than just using a frozen turbulence adjoint with a turbulent flow solver as it is sometimes done, probably more in the continuous adjoint community[45, 137, 147, 211, 212] than in the discrete adjoint community.[49] The difference is the linearisation of the eddy viscosity and hence of the turbulence model and this is quite complex as will be shown in the next section.

However other fully turbulent discrete adjoint solvers exist and have already been mentioned: the one-equation turbulence model of Spalart-Allmaras[231] is frequently used in this case[10, 11, 47, 48, 156, 162, 164] although two-equation models can also be employed.[43, 190] To the author's knowledge, the only other existing adjoint solver based on the algebraic model of Baldwin-Lomax like in this work can be found in Reference [19]. No details are given however on the way in which the turbulence model is actually differentiated. The author of this thesis believes that one- or two-equations models are simpler to incorporate into an adjoint solver than the Baldwin-Lomax model, which might explain the rarity of adjoint solvers based on this algebraic model. Indeed with a field turbulence model, the eddy viscosity is calculated using quantities that are situated in a defined neighbourhood around the point of calculation, hence as for the rest of the flow equations, a well-structured Jacobian can be devised. This Jacobian might however be quite difficult to calculate but should already exist in an implicit flow solver employing this type of turbulence model, although possibly with some approximations that an accurate adjoint solver will have to eliminate. However this is already a good start for the adjoint solver and might explain why such models are popular for turbulent solvers although the overhead of having one or two additional equations in the flow and adjoint solvers is certainly high especially in optimisation. On the other hand, with the Baldwin-Lomax model, the eddy viscosity is calculated along rays from quantitites that could be situated anywhere along these rays. It is thus a full stencil model. Hence, as was already mentioned in section 5.7, it is quite difficult to linearise this algebraic model and this linearisation is not usually done in implicit flow solvers. The technique developed in the solution of the adjoint equation (6.7) with a term by term multiplication between the RHS Jacobian and the adjoint vector makes it possible to treat the Baldwin-Lomax model accurately without the worry of storing the full Jacobian matrices. The other problem posed by the Baldwin-Lomax model is that it uses the maximum function that is not differentiable. Details of how this is overcome will be presented in the next section.

A summary of this section would be that the present adjoint solver is different from any other because it has been entirely hand-differentiated, it is a discrete adjoint solver, it uses Osher's approximate Riemann solver and has been derived for turbulent flows with an accurate linearisation of the Baldwin-Lomax model. All of these points put together make this adjoint unique.

6.6 Calculation of the exact RHS Jacobian

This section details the calculation of the components of the RHS Jacobian of equation (6.7). It is divided into four parts that correspond to the way in which the adjoint solver has been built: an inviscid first-order adjoint solver was first coded and tested. Then upon this basis, a higher-order inviscid adjoint solver was implemented before being upgraded to a viscous laminar solver. The last stage was to incorporate the linearisation of the turbulence model to obtain a fully turbulent adjoint solver. This way of presenting the calculation of the RHS Jacobian is not only a chronological account of how the code was built but it is also very relevant to the final product since, for example, what was done for the convective terms of the inviscid adjoint solver is also employed unchanged in the turbulent adjoint solver. Hence each component is as useful as any other in the final code. Each part will be further divided into what is happening inside the domain, far from the boundaries and then how the boundaries are treated. The behaviour of the interface boundaries between adjacent blocks will be dealt with separately from the rest of the boundaries because of the very different nature and treatment of this type of boundary condition.

6.6.1 First-order inviscid components

6.6.1.1 Inside the domain

The LHS Jacobian of the flow solver MERLIN presented in section 5.7 is already first-order accurate for the convective terms so what is done in MERLIN is used unchanged to create a first-order inviscid adjoint solver. The aim of building this adjoint solver was to create the structure of the code and to be sure that the discrete adjoint method had been well understood but once this is done, the development itself should be very quick.

The basics are however presented since they will be reused for the other parts of the solver. This is explained by an example. Let us consider the one-dimensional problem of Fig-

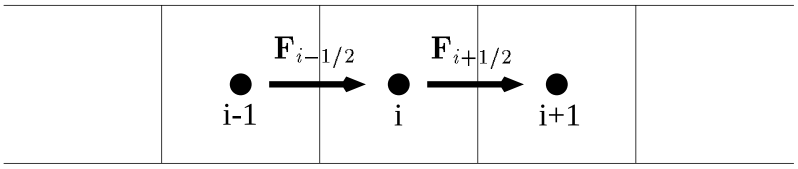


Figure 6.1: First-order inviscid fluxes.

ure 6.1. The residual at cell i is

$$\mathbf{R}_i = \mathbf{F}_{i+1/2} - \mathbf{F}_{i-1/2}$$

where $\mathbf{F}_{i+1/2}$ and $\mathbf{F}_{i-1/2}$ are the convective fluxes $\mathbf{F}^i$ from equation (5.2). Hence the Jacobians based on the residual are

$$\mathbf{C}_i \equiv \frac{\partial \mathbf{R}_i}{\partial \mathbf{P}_i} = \frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_i} - \frac{\partial \mathbf{F}_{i-1/2}}{\partial \mathbf{P}_i} \tag{6.9}$$

$$\mathbf{S}_i \equiv \frac{\partial \mathbf{R}_i}{\partial \mathbf{P}_{i-1}} = -\frac{\partial \mathbf{F}_{i-1/2}}{\partial \mathbf{P}_{i-1}} \tag{6.10}$$

$$\mathbf{N}_i \equiv \frac{\partial \mathbf{R}_i}{\partial \mathbf{P}_{i+1}} = \frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_{i+1}} \tag{6.11}$$

since the fluxes are only first-order accurate. The terms of the form $\frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_i}$ are coming from the linearisation of Osher's scheme presented in section 5.7. The terms $\mathbf{C}_i$, $\mathbf{S}_i$ and $\mathbf{N}_i$ form the RHS Jacobian of equation 6.7. As already explained a term by term matrix-vector product is performed for this RHS instead of assembling the whole Jacobian and then doing a global matrix-vector product with the whole adjoint vector. Let us detail this considering the three-dimensional problem: if for each computational cell i, j, k, the term $\left[\frac{\partial \mathbf{R}(\mathbf{Q}^*)}{\partial \mathbf{P}}\right]^t \boldsymbol{\lambda}^n$ is called $\mathbf{rhs}_{i,j,k}$ then it is calculated according to:

$$\begin{aligned}
\mathbf{rhs}_{i,j,k} = {} & \frac{\partial \mathbf{R}_{i,j,k}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i,j,k} \\
& + \frac{\partial \mathbf{R}_{i-1,j,k}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i-1,j,k} + \frac{\partial \mathbf{R}_{i+1,j,k}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i+1,j,k} \\
& + \frac{\partial \mathbf{R}_{i,j-1,k}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i,j-1,k} + \frac{\partial \mathbf{R}_{i,j+1,k}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i,j+1,k} \\
& + \frac{\partial \mathbf{R}_{i,j,k-1}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i,j,k-1} + \frac{\partial \mathbf{R}_{i,j,k+1}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i,j,k+1}
\end{aligned} \tag{6.12}$$

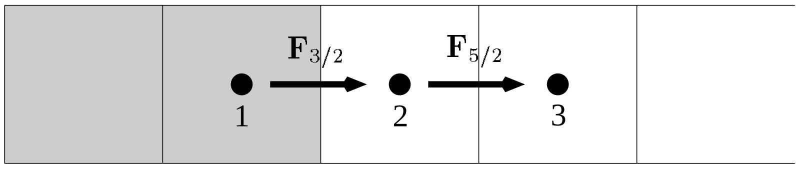


Figure 6.2: First-order inviscid fluxes at the boundaries.

$$\begin{aligned}\mathbf{rhs}_{i,j,k} &= \mathbf{C}_{i,j,k}^{\,t}.\boldsymbol{\lambda}_{i,j,k} \\ &+ \mathbf{N}_{i-1,j,k}^{\,t}.\boldsymbol{\lambda}_{i-1,j,k} + \mathbf{S}_{i+1,j,k}^{\,t}.\boldsymbol{\lambda}_{i+1,j,k} \\ &+ \mathbf{E}_{i,j-1,k}^{\,t}.\boldsymbol{\lambda}_{i,j-1,k} + \mathbf{W}_{i,j+1,k}^{\,t}.\boldsymbol{\lambda}_{i,j+1,k} \\ &+ \mathbf{F}_{i,j,k-1}^{\,t}.\boldsymbol{\lambda}_{i,j,k-1} + \mathbf{B}_{i,j,k+1}^{\,t}.\boldsymbol{\lambda}_{i,j,k+1}\end{aligned}$$

Each term of this sum involves a 5×5 matrix-vector multiplication. The terms $\mathbf{C}_i$, $\mathbf{S}_i$ and $\mathbf{N}_i$, calculated as shown in equations (6.9) to (6.11), are then transposed and multiplied by the appropriate adjoint vectors to form the RHS vector. This has to be done in the three directions to obtain equation (6.12).

As in MERLIN, the adjoint solver works face by face rather than cell by cell as presented above. Hence a term like $\dfrac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_i}$ is calculated only once and its contribution is added to the relevant RHS vectors $\mathbf{rhs}_{i,j,k}$.

6.6.1.2 At the boundaries

The problem is depicted in Figure 6.2. $\mathbf{F}_{5/2}$ is inside the domain so its treatment has been reviewed in the previous subsection, it is only $\mathbf{F}_{3/2}$ that matters here and it is present in the calculation of $\mathbf{C}_2$. Indeed the treatment of the boundary conditions is governed by the application of equation (6.12) (considered only in one dimension here) to the boundary. When $i = 2$ it is written

$$\mathbf{rhs}_2 = \mathbf{C}_2^t.\boldsymbol{\lambda}_2 + \mathbf{N}_1^t.\boldsymbol{\lambda}_1 + \mathbf{S}_3^t.\boldsymbol{\lambda}_3 \tag{6.13}$$

but since $\boldsymbol{\lambda}_1$ is outside the domain, the term $\mathbf{N}_1^t.\boldsymbol{\lambda}_1$ has to be discarded and $\mathbf{rhs}_2$ becomes

$$\mathbf{rhs}_2 = \mathbf{C}_2^t.\boldsymbol{\lambda}_2 + \mathbf{S}_3^t.\boldsymbol{\lambda}_3$$

Hence $\mathbf{N}_1$ and $\mathbf{S}_2$ do not need to be evaluated.

The fact that the value of $\mathbf{P}_1$ depends on $\mathbf{P}_2$ from the application of the boundary conditions detailed in section 5.6.3 implies that

$$\mathbf{C}_2 \equiv \frac{\partial \mathbf{R}_2}{\partial \mathbf{P}_2} = \frac{\partial \mathbf{F}_{5/2}}{\partial \mathbf{P}_2} - \frac{\partial \mathbf{F}_{3/2}}{\partial \mathbf{P}_2} - \frac{\partial \mathbf{F}_{3/2}}{\partial \mathbf{P}_1}\frac{\partial \mathbf{P}_1}{\partial \mathbf{P}_2}$$

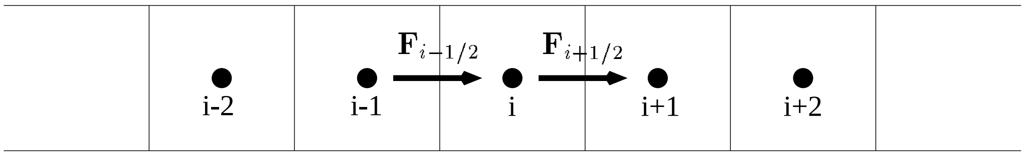


Figure 6.3: Higher-order inviscid fluxes.

This is exactly what also happens in MERLIN and this was presented in section 5.7.4. Besides the matrices $\frac{\partial \mathbf{P}_1}{\partial \mathbf{P}_2}$ are taken from Appendix A except for the interface boundaries, the treatment of which is explained next. This also has to be applied at the other end of the i direction when $i = in$ and of course in the other two directions.

6.6.1.3 Interface boundary

The philosophy that is constantly behind the treatment of the interface boundaries is that the adjoint solver should not be able to see that there is a boundary there and should behave the same as inside the domain. Hence equation (6.13) has to be applied as written, this time $\boldsymbol{\lambda}_1$ is taken from the adjacent block. The use of halo cells that are filled with values of the adjoint vector coming from the adjacent block is made to ease this process.

Since everything is as in the interior of the domain, $\mathbf{C}_2$ and $\mathbf{N}_1$ are calculated with equations (6.9) and (6.11) respectively.

Now that the calculation of the RHS Jacobian for a first-order inviscid adjoint solver has been presented, the complexity is slightly increased with the introduction of the MUSCL scheme to obtain a higher-order inviscid adjoint solver.

6.6.2 Higher-order inviscid components

6.6.2.1 Inside the domain

The use of the MUSCL scheme extends the dependency of the convective fluxes to 4 cells and the dependency of the cell residual to 5 cells. Considering the example of Figure 6.3, $\mathbf{R}_i$ now depends on $\mathbf{P}_{i-2}$, $\mathbf{P}_{i-1}$, $\mathbf{P}_i$, $\mathbf{P}_{i+1}$ and $\mathbf{P}_{i+2}$ hence for this one-dimensional example two additional non zero terms have to be introduced in the Jacobian. The terms become:

$$\mathbf{C}_i \equiv \frac{\partial \mathbf{R}_i}{\partial \mathbf{P}_i} = \frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_i} - \frac{\partial \mathbf{F}_{i-1/2}}{\partial \mathbf{P}_i} \tag{6.14}$$

$$\mathbf{S}_i \equiv \frac{\partial \mathbf{R}_i}{\partial \mathbf{P}_{i-1}} = \frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_{i-1}} - \frac{\partial \mathbf{F}_{i-1/2}}{\partial \mathbf{P}_{i-1}} \tag{6.15}$$

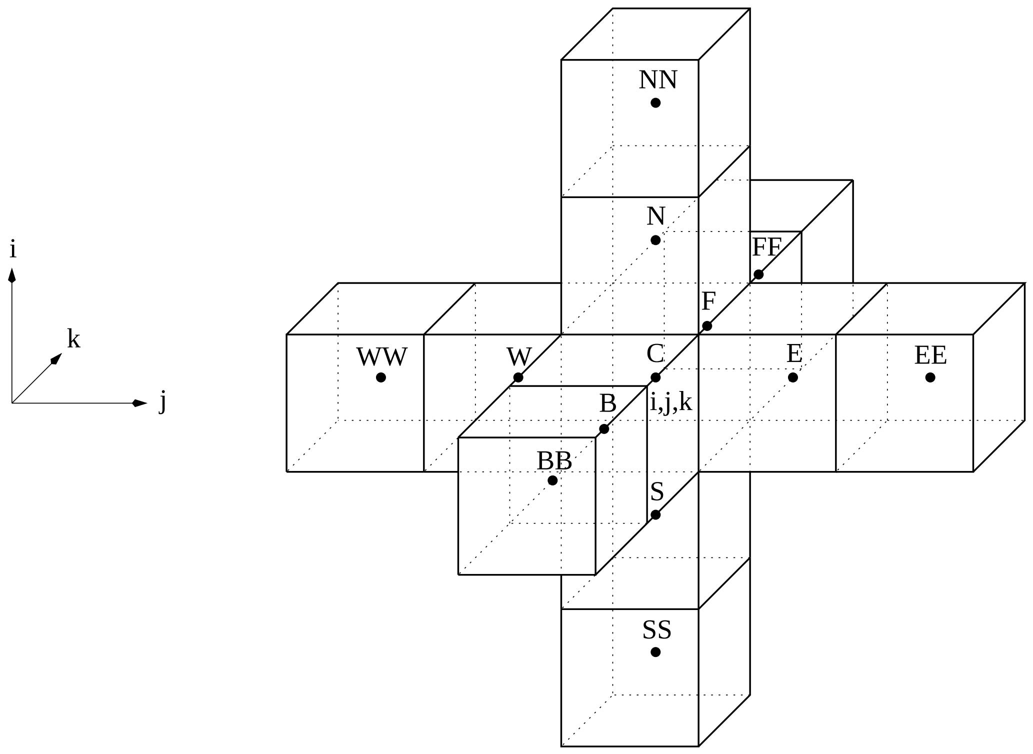


Figure 6.4: Computational stencil for a higher-order inviscid Jacobian.

$$\mathbf{SS}_i \equiv \frac{\partial \mathbf{R}_i}{\partial \mathbf{P}_{i-2}} = -\frac{\partial \mathbf{F}_{i-1/2}}{\partial \mathbf{P}_{i-2}} \tag{6.16}$$

$$\mathbf{N}_i \equiv \frac{\partial \mathbf{R}_i}{\partial \mathbf{P}_{i+1}} = \frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_{i+1}} - \frac{\partial \mathbf{F}_{i-1/2}}{\partial \mathbf{P}_{i+1}} \tag{6.17}$$

$$\mathbf{NN}_i \equiv \frac{\partial \mathbf{R}_i}{\partial \mathbf{P}_{i+2}} = \frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_{i+2}} \tag{6.18}$$

A term like $\frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_i}$ involves the linearisation of Osher's Riemann solver and of the MUSCL scheme in addition to the transformation between conservative and primitive variables. Hence

$$\frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_i} = \frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{Q}_R}\frac{\partial \mathbf{Q}_R}{\partial \mathbf{P}_R}\frac{\partial \mathbf{P}_R}{\partial \mathbf{P}_i} + \frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{Q}_L}\frac{\partial \mathbf{Q}_L}{\partial \mathbf{P}_L}\frac{\partial \mathbf{P}_L}{\partial \mathbf{P}_i}$$

The linearisation of Osher's Riemann solver has already been presented in the implicit part of MERLIN (see section 5.7.2). The linearisation of the MUSCL scheme has not been detailed but is quite straightforward when considering equation (5.16). Since the slope limiter is differentiable, its linearisation inside the MUSCL scheme, does not pose any problem either. To be noticed in equation (5.16) is that $\mathbf{P}_L$ depends on $\mathbf{P}_{i-1}$, $\mathbf{P}_i$ and $\mathbf{P}_{i+1}$ and $\mathbf{P}_R$ on $\mathbf{P}_i$, $\mathbf{P}_{i+1}$ and $\mathbf{P}_{i+2}$.

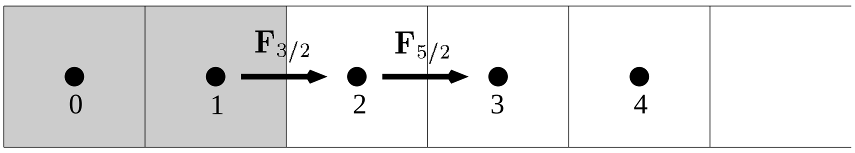


Figure 6.5: Higher-order inviscid fluxes at the boundaries.

This has to be performed in the three directions hence the stencil for this inviscid higher-order Jacobian involves the 13 cells shown in Figure 6.4. Considering this, the component of the RHS vector of the adjoint equation that involves the matrix-vector product between the Jacobian and the adjoint vector, becomes

$$\begin{aligned}
\mathbf{rhs}_{i,j,k} &= \frac{\partial \mathbf{R}_{i,j,k}}{\partial \mathbf{P}_{i,j,k}}^t.\boldsymbol{\lambda}_{i,j,k} \\
&+ \frac{\partial \mathbf{R}_{i-1,j,k}}{\partial \mathbf{P}_{i,j,k}}^t.\boldsymbol{\lambda}_{i-1,j,k} + \frac{\partial \mathbf{R}_{i-2,j,k}}{\partial \mathbf{P}_{i,j,k}}^t.\boldsymbol{\lambda}_{i-2,j,k} + \frac{\partial \mathbf{R}_{i+1,j,k}}{\partial \mathbf{P}_{i,j,k}}^t.\boldsymbol{\lambda}_{i+1,j,k} + \frac{\partial \mathbf{R}_{i+2,j,k}}{\partial \mathbf{P}_{i,j,k}}^t.\boldsymbol{\lambda}_{i+2,j,k} \\
&+ \frac{\partial \mathbf{R}_{i,j-1,k}}{\partial \mathbf{P}_{i,j,k}}^t.\boldsymbol{\lambda}_{i,j-1,k} + \frac{\partial \mathbf{R}_{i,j-2,k}}{\partial \mathbf{P}_{i,j,k}}^t.\boldsymbol{\lambda}_{i,j-2,k} + \frac{\partial \mathbf{R}_{i,j+1,k}}{\partial \mathbf{P}_{i,j,k}}^t.\boldsymbol{\lambda}_{i,j+1,k} + \frac{\partial \mathbf{R}_{i,j+2,k}}{\partial \mathbf{P}_{i,j,k}}^t.\boldsymbol{\lambda}_{i,j+2,k} \\
&+ \frac{\partial \mathbf{R}_{i,j,k-1}}{\partial \mathbf{P}_{i,j,k}}^t.\boldsymbol{\lambda}_{i,j,k-1} + \frac{\partial \mathbf{R}_{i,j,k-2}}{\partial \mathbf{P}_{i,j,k}}^t.\boldsymbol{\lambda}_{i,j,k-2} + \frac{\partial \mathbf{R}_{i,j,k+1}}{\partial \mathbf{P}_{i,j,k}}^t.\boldsymbol{\lambda}_{i,j,k+1} + \frac{\partial \mathbf{R}_{i,j,k+2}}{\partial \mathbf{P}_{i,j,k}}^t.\boldsymbol{\lambda}_{i,j,k+2} \\
\mathbf{rhs}_{i,j,k} &= \mathbf{C}_{i,j,k}^t.\boldsymbol{\lambda}_{i,j,k} \\
&+ \mathbf{N}_{i-1,j,k}^t.\boldsymbol{\lambda}_{i-1,j,k} + \mathbf{NN}_{i-2,j,k}^t.\boldsymbol{\lambda}_{i-2,j,k} + \mathbf{S}_{i+1,j,k}^t.\boldsymbol{\lambda}_{i+1,j,k} + \mathbf{SS}_{i+2,j,k}^t.\boldsymbol{\lambda}_{i+2,j,k} \\
&+ \mathbf{E}_{i,j-1,k}^t.\boldsymbol{\lambda}_{i,j-1,k} + \mathbf{EE}_{i,j-2,k}^t.\boldsymbol{\lambda}_{i,j-2,k} + \mathbf{W}_{i,j+1,k}^t.\boldsymbol{\lambda}_{i,j+1,k} + \mathbf{WW}_{i,j+2,k}^t.\boldsymbol{\lambda}_{i,j+2,k} \\
&+ \mathbf{F}_{i,j,k-1}^t.\boldsymbol{\lambda}_{i,j,k-1} + \mathbf{FF}_{i,j,k-2}^t.\boldsymbol{\lambda}_{i,j,k-2} + \mathbf{B}_{i,j,k+1}^t.\boldsymbol{\lambda}_{i,j,k+1} + \mathbf{BB}_{i,j,k+2}^t.\boldsymbol{\lambda}_{i,j,k+2}
\end{aligned} \tag{6.19}$$

6.6.2.2 At the boundaries

Since the fluxes are higher-order accurate, two fluxes are now involved in the boundary conditions i.e. $\mathbf{F}_{5/2}$ and $\mathbf{F}_{3/2}$ as shown in Figure 6.5. This potentially implies some modifications in a large number of Jacobian components but writing equation (6.19) at the boundary eliminates a number of terms due to their multiplication to an adjoint vector that is outside the domain. A careful check shows that in the end, the boundary conditions only affects $\mathbf{C}_2$, $\mathbf{S}_3$ and $\mathbf{N}_2$ in

$$\mathbf{rhs}_2 = \mathbf{C}_2^t.\boldsymbol{\lambda}_2 + \mathbf{S}_3^t.\boldsymbol{\lambda}_3 + \mathbf{SS}_4^t.\boldsymbol{\lambda}_4$$

and

$$\mathbf{rhs}_3 = \mathbf{C}_3^t.\boldsymbol{\lambda}_3 + \mathbf{N}_2^t.\boldsymbol{\lambda}_2 + \mathbf{S}_4^t.\boldsymbol{\lambda}_4 + \mathbf{SS}_5^t.\boldsymbol{\lambda}_5$$

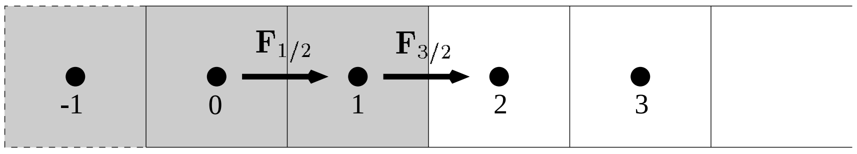


Figure 6.6: Higher-order inviscid fluxes at an interface boundary with the additional halo cell.

These matrices become

$$
\begin{aligned}
\mathbf{C}_2 &\equiv \frac{\partial \mathbf{R}_2}{\partial \mathbf{P}_2} = \frac{\partial \mathbf{F}_{5/2}}{\partial \mathbf{P}_2} + \frac{\partial \mathbf{F}_{5/2}}{\partial \mathbf{P}_1}\frac{\partial \mathbf{P}_1}{\partial \mathbf{P}_2} - \frac{\partial \mathbf{F}_{3/2}}{\partial \mathbf{P}_2} - \frac{\partial \mathbf{F}_{3/2}}{\partial \mathbf{P}_1}\frac{\partial \mathbf{P}_1}{\partial \mathbf{P}_2} - \frac{\partial \mathbf{F}_{3/2}}{\partial \mathbf{P}_0}\frac{\partial \mathbf{P}_0}{\partial \mathbf{P}_2}\\
\mathbf{S}_3 &\equiv \frac{\partial \mathbf{R}_3}{\partial \mathbf{P}_2} = \frac{\partial \mathbf{F}_{7/2}}{\partial \mathbf{P}_2} - \frac{\partial \mathbf{F}_{5/2}}{\partial \mathbf{P}_2} - \frac{\partial \mathbf{F}_{5/2}}{\partial \mathbf{P}_1}\frac{\partial \mathbf{P}_1}{\partial \mathbf{P}_2}\\
\mathbf{N}_2 &\equiv \frac{\partial \mathbf{R}_2}{\partial \mathbf{P}_3} = \frac{\partial \mathbf{F}_{5/2}}{\partial \mathbf{P}_3} - \frac{\partial \mathbf{F}_{3/2}}{\partial \mathbf{P}_3} - \frac{\partial \mathbf{F}_{3/2}}{\partial \mathbf{P}_0}\frac{\partial \mathbf{P}_0}{\partial \mathbf{P}_3}
\end{aligned}
$$

The matrices $\frac{\partial \mathbf{P}_1}{\partial \mathbf{P}_2}$ have already been presented in Appendix A since they are used in the implicit part of MERLIN, the matrices $\frac{\partial \mathbf{P}_0}{\partial \mathbf{P}_2}$ and $\frac{\partial \mathbf{P}_0}{\partial \mathbf{P}_3}$ did not appear so far. They result from the application of the boundary conditions as detailed in section 5.6.3. They are easily calculated and their value is also provided in Appendix A. A similar treatment has to be applied at the other end of the i direction and of course in the other two directions.

6.6.2.3 Interface boundary

The philosophy that everything should be as if inside the domain for this kind of boundary condition, is still the same so equation (6.19) is applied as

$$\mathbf{rhs}_2 = \mathbf{C}_2^t.\boldsymbol{\lambda}_2 + \mathbf{N}_1^t.\boldsymbol{\lambda}_1 + \mathbf{NN}_0^t.\boldsymbol{\lambda}_0 + \mathbf{S}_3^t.\boldsymbol{\lambda}_3 + \mathbf{SS}_4^t.\boldsymbol{\lambda}_4$$

and similarly for $\mathbf{rhs}_3$. The terms in this equation are calculated like in the interior of the domain with equations (6.14) to (6.18), which implies the calculation of the flux $\mathbf{F}_{1/2}$ inside the halo cells. As shown in Figure 6.6, this necessitates the value of $\mathbf{P}_{-1}$ that is outside the normal halo cells. Hence a third layer of halo cells has to be implemented in the adjoint solver for this type of boundary conditions. Once again this has to be done for the three directions.

6.6.3 Viscous laminar components

6.6.3.1 Inside the domain

This part relies on what was presented for the flow solver MERLIN in sections 5.6.2.1 and 5.7.3.1. As explained there, the residual at one cell depends on the value of the variables

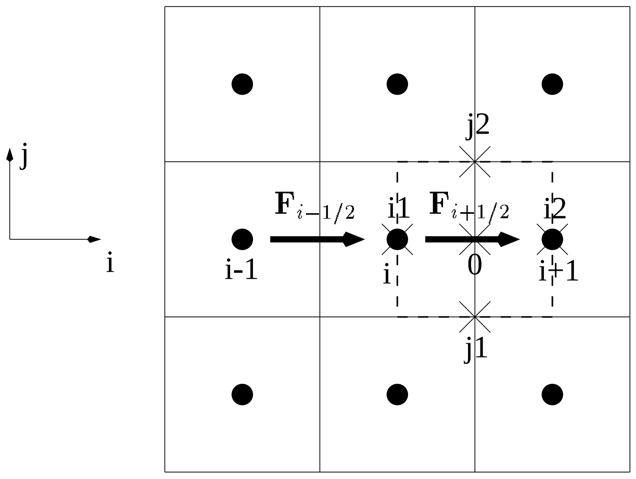


Figure 6.7: Viscous laminar fluxes.

in 19 surrounding cells that are depicted in the stencil of Figure 5.8.

This paragraph will only look at the fluxes in the i direction. The situation is depicted in two dimensions in Figure 6.7. This is a simplification of Figure 5.3 for ease of understanding but this should however be viewed in three dimensions even if only fluxes in the i direction are considered here. The residual at cell i is still

$$\mathbf{R}_i = \mathbf{F}_{i+1/2} - \mathbf{F}_{i-1/2}$$

but this time, $\mathbf{F}_{i+1/2}$ and $\mathbf{F}_{i-1/2}$ are the diffusive fluxes $\mathbf{F}^v$ from equation (5.2). The residual depends on 15 cells, 9 which are shown in Figure 6.7, 6 further being in a perpendicular plane. This makes 15 non-null contributions $\dfrac{\partial \mathbf{R}_i}{\partial \mathbf{P}_{l,m,n}}$ to the Jacobian. Instead of detailing all of these terms, only a sample of what is involved is given here. This is the calculation of $\dfrac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_{i,j,k}}$ with, according to the notation of Figure 6.7 and remembering from section 5.6.2.1 that $\mathbf{P}_0 = \frac{1}{2}(\mathbf{P}_{i,j,k} + \mathbf{P}_{i+1,j,k})$, $\mathbf{P}_{i1} = \mathbf{P}_{i,j,k}$, $\mathbf{P}_{j2} = \frac{1}{4}(\mathbf{P}_{i,j,k} + \mathbf{P}_{i+1,j,k} + \mathbf{P}_{i,j+1,k} + \mathbf{P}_{i+1,j+1,k})$, etc.,

$$\begin{aligned}
\frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_{i,j,k}} &= \frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_0}\frac{\partial \mathbf{P}_0}{\partial \mathbf{P}_{i,j,k}} + \frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_{j1}}\frac{\partial \mathbf{P}_{j1}}{\partial \mathbf{P}_{i,j,k}} + \frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_{j2}}\frac{\partial \mathbf{P}_{j2}}{\partial \mathbf{P}_{i,j,k}} + \frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_{k1}}\frac{\partial \mathbf{P}_{k1}}{\partial \mathbf{P}_{i,j,k}} \\
&\quad + \frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_{k2}}\frac{\partial \mathbf{P}_{k2}}{\partial \mathbf{P}_{i,j,k}} + \frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_{i1}}\frac{\partial \mathbf{P}_{i1}}{\partial \mathbf{P}_{i,j,k}} \\
&= \frac{1}{2}\frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_0} + \frac{1}{4}\left(\frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_{j1}} + \frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_{j2}} + \frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_{k1}} + \frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_{k2}}\right) + \frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_{i1}}
\end{aligned}$$

This is also what is done in MERLIN or in the LHS Jacobian of the adjoint solver except that only the 7 Jacobians corresponding to the first-order inviscid stencil are kept (hence computed). This derivation is coming from the evaluation of the variables at the face centres of the dual volume used in the calculation of the viscous fluxes. This is just a straightforward linearisation of what was presented in section 5.6.2.1. The calculation of terms like $\frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_{j1}}$ is not presented either because again it is only a differentiation of the way the viscous fluxes are calculated from these face centre values and this does not present any difficulty.

Let us go back to three dimensions with the calculation of the viscous fluxes in the three directions and write the equation forming the RHS vector of the adjoint equation. This is written once for completeness and reference but it is not very helpful for understanding and a one-dimensional version, looking at each direction separately, would be preferred.

$$\begin{aligned}
\mathbf{rhs}_{i,j,k} &= \frac{\partial \mathbf{R}_{i,j,k}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i,j,k} + \frac{\partial \mathbf{R}_{i-1,j,k}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i-1,j,k} + \frac{\partial \mathbf{R}_{i+1,j,k}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i+1,j,k} \\
&+ \frac{\partial \mathbf{R}_{i,j+1,k}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i,j+1,k} + \frac{\partial \mathbf{R}_{i-1,j+1,k}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i-1,j+1,k} + \frac{\partial \mathbf{R}_{i+1,j+1,k}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i+1,j+1,k} \\
&+ \frac{\partial \mathbf{R}_{i,j-1,k}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i,j-1,k} + \frac{\partial \mathbf{R}_{i-1,j-1,k}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i-1,j-1,k} + \frac{\partial \mathbf{R}_{i+1,j-1,k}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i+1,j-1,k} \\
&+ \frac{\partial \mathbf{R}_{i,j,k+1}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i,j,k+1} + \frac{\partial \mathbf{R}_{i-1,j,k+1}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i-1,j,k+1} + \frac{\partial \mathbf{R}_{i+1,j,k+1}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i+1,j,k+1} \\
&+ \frac{\partial \mathbf{R}_{i,j,k-1}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i,j,k-1} + \frac{\partial \mathbf{R}_{i-1,j,k-1}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i-1,j,k-1} + \frac{\partial \mathbf{R}_{i+1,j,k-1}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i+1,j,k-1} \\
&+ \frac{\partial \mathbf{R}_{i,j-1,k+1}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i,j-1,k+1} + \frac{\partial \mathbf{R}_{i,j+1,k+1}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i,j+1,k+1} \\
&+ \frac{\partial \mathbf{R}_{i,j-1,k-1}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i,j-1,k-1} + \frac{\partial \mathbf{R}_{i,j+1,k-1}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{i,j+1,k-1} \\
\mathbf{rhs}_{i,j,k} &= \mathbf{C}_{i,j,k}^t . \boldsymbol{\lambda}_{i,j,k} + \mathbf{N}_{i-1,j,k}^t . \boldsymbol{\lambda}_{i-1,j,k} + \mathbf{S}_{i+1,j,k}^t . \boldsymbol{\lambda}_{i+1,j,k} \\
&+ \mathbf{W}_{i,j+1,k}^t . \boldsymbol{\lambda}_{i,j+1,k} + \mathbf{NW}_{i-1,j+1,k}^t . \boldsymbol{\lambda}_{i-1,j+1,k} + \mathbf{SW}_{i+1,j+1,k}^t . \boldsymbol{\lambda}_{i+1,j+1,k} \\
&+ \mathbf{E}_{i,j-1,k}^t . \boldsymbol{\lambda}_{i,j-1,k} + \mathbf{NE}_{i-1,j-1,k}^t . \boldsymbol{\lambda}_{i-1,j-1,k} + \mathbf{SE}_{i+1,j-1,k}^t . \boldsymbol{\lambda}_{i+1,j-1,k} \\
&+ \mathbf{B}_{i,j,k+1}^t . \boldsymbol{\lambda}_{i,j,k+1} + \mathbf{NB}_{i-1,j,k+1}^t . \boldsymbol{\lambda}_{i-1,j,k+1} + \mathbf{SB}_{i+1,j,k+1}^t . \boldsymbol{\lambda}_{i+1,j,k+1} \\
&+ \mathbf{F}_{i,j,k-1}^t . \boldsymbol{\lambda}_{i,j,k-1} + \mathbf{NF}_{i-1,j,k-1}^t . \boldsymbol{\lambda}_{i-1,j,k-1} + \mathbf{SF}_{i+1,j,k-1}^t . \boldsymbol{\lambda}_{i+1,j,k-1} \\
&+ \mathbf{EB}_{i,j-1,k+1}^t . \boldsymbol{\lambda}_{i,j-1,k+1} + \mathbf{WB}_{i,j+1,k+1}^t . \boldsymbol{\lambda}_{i,j+1,k+1} \\
&+ \mathbf{EF}_{i,j-1,k-1}^t . \boldsymbol{\lambda}_{i,j-1,k-1} + \mathbf{WF}_{i,j+1,k-1}^t . \boldsymbol{\lambda}_{i,j+1,k-1}
\end{aligned} \tag{6.20}$$

6.6.3.2 At the boundaries

What happens at the boundaries for the linearisation of the viscous fluxes is quite complicated and a lot of cases have to be considered. Only examples will be given in this

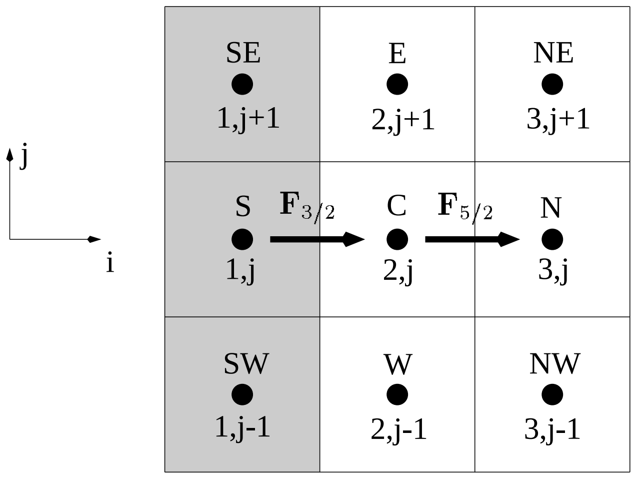


Figure 6.8: Viscous fluxes at the boundaries: first case.

subsection that treat only the viscous fluxes in one direction, the i direction.

The most simple case is when the boundary is in the same direction as the flux studied as in Figure 6.8 where the boundary is at $i = 3/2$. It is obvious that because the value of $\mathbf{P}_{1,j}$ depends on $\mathbf{P}_{2,j}$, there is a contribution of cell $1,j$ that needs to be added to cell $2,j$ hence

$$\mathbf{C}_{2,j} \equiv \frac{\partial \mathbf{R}_{2,j}}{\partial \mathbf{P}_{2,j}} = \frac{\partial \mathbf{F}_{5/2}}{\partial \mathbf{P}_{2,j}} - \frac{\partial \mathbf{F}_{3/2}}{\partial \mathbf{P}_{2,j}} - \frac{\partial \mathbf{F}_{3/2}}{\partial \mathbf{P}_{1,j}} \frac{\partial \mathbf{P}_{1,j}}{\partial \mathbf{P}_{2,j}}$$

where again $\frac{\partial \mathbf{P}_{1,j}}{\partial \mathbf{P}_{2,j}}$ results from the linearisation of the boundary condition and can be found in Appendix A. A bit more difficult is the contribution from cell $1, j-1$ via cell $2, j-1$ to cell $2,j$

$$\mathbf{W}_{2,j} \equiv \frac{\partial \mathbf{R}_{2,j}}{\partial \mathbf{P}_{2,j-1}} = \frac{\partial \mathbf{F}_{5/2}}{\partial \mathbf{P}_{2,j-1}} - \frac{\partial \mathbf{F}_{3/2}}{\partial \mathbf{P}_{2,j-1}} - \frac{\partial \mathbf{F}_{3/2}}{\partial \mathbf{P}_{1,j-1}} \frac{\partial \mathbf{P}_{1,j-1}}{\partial \mathbf{P}_{2,j-1}}$$

and similarly between cell $1, j+1$ via cell $2, j+1$ to cell $2, j$.

Now that this has been presented, a few words are needed to come back to the flow solver MERLIN and explain what is done about the boundary conditions of the viscous fluxes in the LHS Jacobian. As already mentioned in section 5.7.4 a lot of simplifying approximations are made. The contribution from cell $1,j$ to cell $2,j$ in $\mathbf{C}_{2,j}$ is nevertheless taken into account properly. This is however relatively easy to take into account because it only depends on the value of i and occurs when $i = 2$ and $i = in$. The other contribution from cell $1, j-1$ via cell $2, j-1$ is not accounted for in MERLIN because it depends not only on the value of i but also on j as we will see next. As mentioned in section 5.7.4, whatever the situation $\mathbf{W}_{2,j}$ is calculated as if inside the domain without worrying about

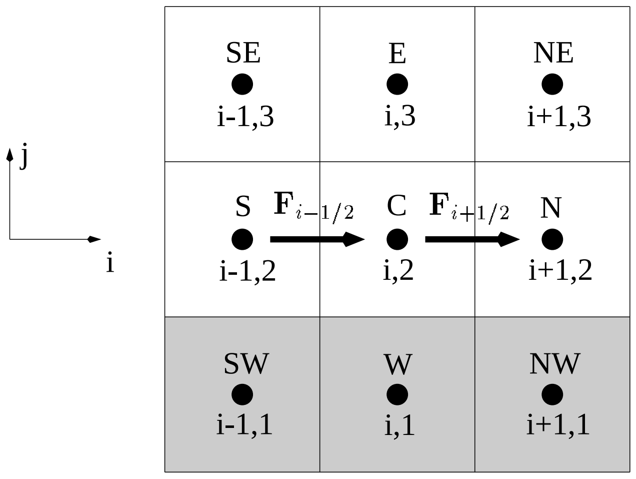


Figure 6.9: Viscous fluxes at the boundaries: second case.

the boundaries. All the cases presented next are not accounted for in the LHS Jacobian of MERLIN or in the LHS of the adjoint solver. Only the RHS of the adjoint solver accounts for them.

Another possible case is when the boundary is not in the direction of the flux as depicted in Figure 6.9 where the boundary is at $j = 3/2$. Like in the previous case, there is a contribution from cell $i, 1$ to cell $i, 2$ as well as from cell $i \pm 1, 1$ to cell $i \pm 1, 2$ to give

$$\mathbf{C}_{i,2} \equiv \frac{\partial \mathbf{R}_{i,2}}{\partial \mathbf{P}_{i,2}} = \frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_{i,2}} + \frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_{i,1}} \frac{\partial \mathbf{P}_{i,1}}{\partial \mathbf{P}_{i,2}} - \frac{\partial \mathbf{F}_{i-1/2}}{\partial \mathbf{P}_{i,2}} - \frac{\partial \mathbf{F}_{i-1/2}}{\partial \mathbf{P}_{i,1}} \frac{\partial \mathbf{P}_{i,1}}{\partial \mathbf{P}_{i,2}}$$

$$\mathbf{S}_{i,2} \equiv \frac{\partial \mathbf{R}_{i,2}}{\partial \mathbf{P}_{i-1,2}} = -\frac{\partial \mathbf{F}_{i-1/2}}{\partial \mathbf{P}_{i-1,2}} - \frac{\partial \mathbf{F}_{i-1/2}}{\partial \mathbf{P}_{i-1,1}} \frac{\partial \mathbf{P}_{i-1,1}}{\partial \mathbf{P}_{i-1,2}}$$

The final case happens at a corner point as shown in Figure 6.10. This is the superposition of the two previous cases and starts to become complicated. As previously, there is a contribution from cell $1, 3$ to cell $2, 3$ and of cell $3, 1$ to cell $3, 2$. Cell $2, 2$ now receives a contribution from cell $1, 2$, from cell $2, 1$ and from the corner point $1, 1$. This gives

$$\mathbf{C}_{2,2} \equiv \frac{\partial \mathbf{R}_{2,2}}{\partial \mathbf{P}_{2,2}} = \frac{\partial \mathbf{F}_{5/2}}{\partial \mathbf{P}_{2,2}} + \frac{\partial \mathbf{F}_{5/2}}{\partial \mathbf{P}_{2,1}} \frac{\partial \mathbf{P}_{2,1}}{\partial \mathbf{P}_{2,2}} - \frac{\partial \mathbf{F}_{3/2}}{\partial \mathbf{P}_{2,2}} - \frac{\partial \mathbf{F}_{3/2}}{\partial \mathbf{P}_{2,1}} \frac{\partial \mathbf{P}_{2,1}}{\partial \mathbf{P}_{2,2}} - \frac{\partial \mathbf{F}_{3/2}}{\partial \mathbf{P}_{1,2}} \frac{\partial \mathbf{P}_{1,2}}{\partial \mathbf{P}_{2,2}} - \frac{\partial \mathbf{F}_{3/2}}{\partial \mathbf{P}_{1,1}} \frac{\partial \mathbf{P}_{1,1}}{\partial \mathbf{P}_{2,2}}$$

The term $\dfrac{\partial \mathbf{P}_{1,1}}{\partial \mathbf{P}_{2,2}}$ depends on the way the boundary condition is applied in the corner point. Either

$$\frac{\partial \mathbf{P}_{1,1}}{\partial \mathbf{P}_{2,2}} = \frac{\partial \mathbf{P}_{1,1}}{\partial \mathbf{P}_{1,2}} \frac{\partial \mathbf{P}_{1,2}}{\partial \mathbf{P}_{2,2}}$$

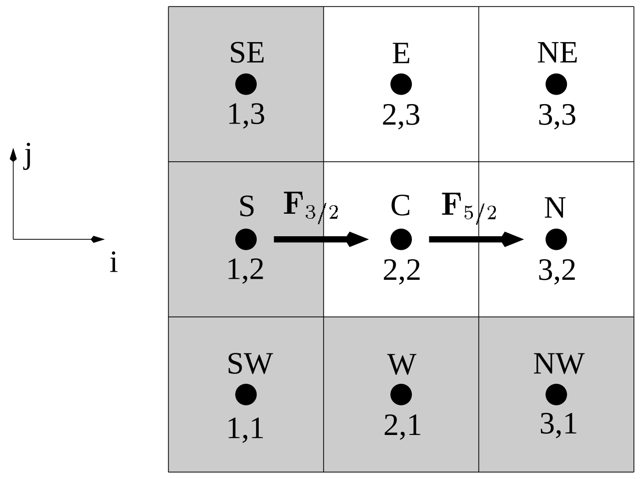


Figure 6.10: Viscous fluxes at a corner point: third case.

or

$$\frac{\partial \mathbf{P}_{1,1}}{\partial \mathbf{P}_{2,2}} = \frac{\partial \mathbf{P}_{1,1}}{\partial \mathbf{P}_{2,1}} \frac{\partial \mathbf{P}_{2,1}}{\partial \mathbf{P}_{2,2}}$$

The 3 cases presented above were for boundaries situated at $i = 2$ but of course similar cases occur when $i = in$. In the same way, the boundary at $j = jn$ has to be considered without forgetting that the third dimension k plays a similar role to j when looking at the fluxes in the i direction and has also to be accounted for. To summarise, all the following cases have to be considered:

- $i = 2$ that is further divided into
  - $j = 2$
  - $2 < j < jn$
  - $j = jn$

  and

  - $k = 2 < kn$
  - $2 < k < kn$
  - $k = kn > 2$
  - $k = 2 = kn$
- $2 < i < in$ that is further divided into
  - $j = 2$

  - $2 < j < jn$
  - $j = jn$

  and

  - $k = 2 < kn$
  - $2 < k < kn$
  - $k = kn > 2$
  - $k = 2 = kn$

- $i = in$ that is further divided into
  - $j = 2$
  - $2 < j < jn$
  - $j = jn$

  and

  - $k = 2 < kn$
  - $2 < k < kn$
  - $k = kn > 2$
  - $k = 2 = kn$

Notice the case where $k = 2 = kn$ that is needed when doing (quasi-)two-dimensional calculations around aerofoils that will be presented at the end of this chapter. This case is well distinct from the cases $k = 2$ and $k = kn$ and has to be treated separately.

All the material presented in this subsection refers only to the viscous fluxes in the i direction but of course a similar methodology must be applied for the viscous fluxes in the j and k directions. The application of the boundary conditions for the viscous fluxes becomes very complicated so a careful and systematic process is needed to code this into the adjoint solver. The last thing to mention is of course that once the Jacobian contributions have been calculated, they are employed in equation (6.20) written for the relevant values of i, j and k. Close to a boundary this implies that the terms multiplied by an adjoint vector that is outside the domain, are discarded as it was in subsections 6.6.1.2 and 6.6.2.2 for the convective fluxes. Whenever this happens, it means that a contribution from a halo cell has to be incorporated in a cell inside the domain.

6.6.3.3 Interface boundary

There is not much to say about the interface boundary for the viscous fluxes. Once again, the philosophy that the adjoint solver should not be able to see these boundaries, is applied and so equation (6.20) is employed as it is written with all its terms. This is relatively easy to do and note that the terms that have to be coded in addition to the normal terms used inside the domain, are precisely the same terms that are discarded at a boundary other than an interface. Things are slightly more complicated when dealing with corner points, especially when one of the two boundaries is not an interface. In this case, what has been said for normal boundaries in the previous subsection is applied but without forgetting that there is an interface boundary and that terms associated with it have to be accounted for as well.

This concludes this section on the calculation of the contributions to the exact RHS Jacobian coming from the laminar viscous fluxes. The linearisation of the turbulence model to obtain a fully turbulent adjoint solver is described next.

6.6.4 Turbulent components

6.6.4.1 Inside the domain

This subsection presents first the linearisation of the turbulent viscosity μ_t and then how this is used in the adjoint solver.

The description of the linearisation of μ_t relies heavily on the description of the algebraic turbulence model of Baldwin-Lomax made in section 5.6.2.2. This showed that the turbulent viscosity is calculated along rays and that its value depends on all the cells along these rays since the boundary between the inner and outer eddy viscosity regions is not known in advance and neither is the location of F_{max} for the outer turbulent viscosity. As an approximation when these positions are known for a particular ray, the next paragraph shows that μ_t depends numerically on a fixed number of cells, the position of which varies from ray to ray. This makes it possible to linearise μ_t.

The turbulent viscosity is either equal to the inner eddy viscosity or to the outer eddy viscosity of the model. These two cases can be treated separately. It is assumed that the rays along which μ_t is calculated are j rays of cells of constant i, k. When $\mu_t = (\mu_t)_{inner}$, its value depends on the 8 cells shown in Figure 6.11(a). These are the actual cell i, j, k where μ_t is calculated, 6 cells around it that are used to calculate the magnitude of vorticity and all its velocity gradients and the cell at $j = 2$ that represents values at the wall needed to calculate y^+. When $\mu_t = (\mu_t)_{outer}$, its value depends on the 10 cells shown in Figure 6.11(b). These 10 cells are the cell i, j, k for the density; 7 cells around $j = j_{maxF}$ that is the j position where the function F is equal to F_{max}, to calculate the magnitude of vorticity $|\omega|_{j_{maxF}}$; the cell at $j = 2$ to calculate $y^+_{j_{maxF}}$; 2 cells at $j = j_{maxU}$ and $j = j_{minU}$ that represent respectively the position of maximum velocity amplitude and the position

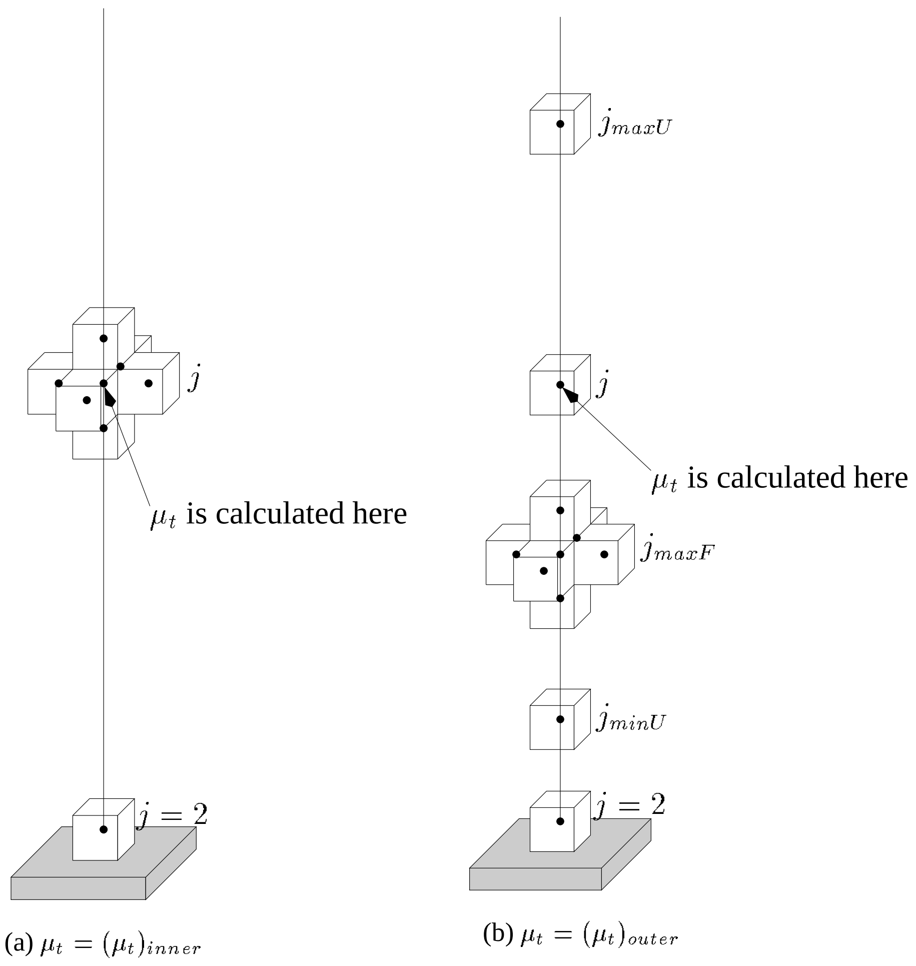

j
μ_t is calculated here
$j = 2$
(a) $\mu_t = (\mu_t)_{inner}$
j_{maxU}
j
μ_t is calculated here
j_{maxF}
j_{minU}
$j = 2$
(b) $\mu_t = (\mu_t)_{outer}$

Figure 6.11: Domain of dependency of μ_t.

of minimum velocity amplitude that are used to calculate U_{diff}. When these two cases are combined, the turbulent viscosity potentially depends on 17 cells. No assumption is made about the position of these cells along the ray although it is possible that two cells from the dependency study are the same.

Hence 17 terms $\dfrac{\partial(\mu_t)_{i,j,k}}{\partial \mathbf{P}_l}$ have to be calculated for each i, j, k. This is a straight differentiation of the way the eddy viscosity is calculated and is not presented in detail. The only problem encountered concerns the maximum and minimum functions used for the outer eddy viscosity that are non-differentiable. As it has been hinted earlier with the dependency drawings, the approximation made is that the derivative of the maximum is equal to the derivative of the function at the point where it is maximum hence for example

$$\frac{\partial F_{max}}{\partial \mathbf{P}_l} \approx \left.\frac{\partial F}{\partial \mathbf{P}_l}\right|_{j=j_{maxF}}$$

This is true only if the maximum function is smooth. This process was also applied to the minimum function in the calculation of F_{wake}

$$F_{wake} = \min\left(y_{max}F_{max}, \frac{C_{wake}y_{max}{U_{diff}}^2}{F_{max}}\right)$$

However this did not work well at all. A thorough investigation would be needed to understand what is happening exactly in this case but it was found, rather crudely, that if F_{wake} was always calculated as

$$F_{wake} = y_{max}F_{max}$$

this avoided any problem and the overall sensitivity derivatives calculated by the adjoint solver agreed quite well with derivatives calculated by finite difference. With the minimum function, the agreement was not as good. Physically the term $y_{max}F_{max}$ comes[185] from Cebeci-Smith's model for wall-bounded boundary layers whereas the term $\dfrac{C_{wake}y_{max}{U_{diff}}^2}{F_{max}}$ comes from Prandtl's model for free shear flows and should be used in the aerofoil wake. Since in our case, μ_t is always calculated above the surface of an aerofoil or wing, the modification of the definition of F_{wake} is valid and should not change the accuracy of the model. It was adopted for all the turbulent computations. To be consistent, this modification was also included in the flow solver MERLIN.

Once the terms $\dfrac{\partial(\mu_t)_{i,j,k}}{\partial \mathbf{P}_l}$ have been calculated, they are used in the RHS Jacobian as follows. Since for a turbulent calculation, the viscosity μ is the sum of the laminar viscosity and the turbulent viscosity, the derivative of the viscous flux $\mathbf{F}^v$ can be written

$$\frac{\partial \mathbf{F}^v}{\partial \mathbf{P}_l} = \left.\frac{\partial \mathbf{F}^v}{\partial \mathbf{P}_l}\right|_{\mu_t = constant} + \left.\frac{\partial \mathbf{F}^v}{\partial \mathbf{P}_l}\right|_{all\,other\,terms\,except\,\mu_t = constant}$$

In the previous section on the calculation of the viscous laminar contributions to the Jacobian, the assumption that the turbulent viscosity was constant [2] was made hence it is the term $\left.\frac{\partial \mathbf{F}^v}{\partial \mathbf{P}_l}\right|_{\mu_t\,=\,constant}$ that was actually calculated there. This section is concerned with $\left.\frac{\partial \mathbf{F}^v}{\partial \mathbf{P}_l}\right|_{all\ other\ terms\ except\ \mu_t\,=\,constant}$. Since only the turbulent viscosity has to be differentiated, this derivative is not difficult to calculate from the definition of $\mathbf{F}^v$ and is not detailed here. Note however that the viscous flux $\mathbf{F}^v$ is calculated at the face between two cells at the position 0 of Figure 5.3 hence the turbulent viscosity that has to be considered is

$$(\mu_t)_0 = \frac{1}{2}[(\mu_t)_{i,j,k} + (\mu_t)_{i+1,j,k}]$$

with the notations of that Figure.

In one dimension, the residual at one cell i is still

$$\mathbf{R}_i = \mathbf{F}_{i+1/2} - \mathbf{F}_{i-1/2}$$

and it has to be differentiated with respect to all the cells that contribute to the turbulent viscosities used in $\mathbf{F}_{i+1/2}$ and $\mathbf{F}_{i-1/2}$. Clearly this makes a lot of contributions to calculate, that are then used into an equation equivalent to equation (6.20) to compute the RHS vector $\mathbf{rhs}_{i,j,k}$. This equation is too complicated to be written and in fact is not needed to implement the linearisation of the turbulent viscosity into the adjoint solver. Indeed as already mentioned, the adjoint solver works face by face and hence flux by flux rather than cell by cell. For each flux $\mathbf{F}$, the relevant derivatives $\frac{\partial \mathbf{F}}{\partial \mathbf{P}_l}$ are calculated. It is then possible to identify the terms $\frac{\partial \mathbf{R}}{\partial \mathbf{P}_l}$ and then the RHS vectors $\mathbf{rhs}_{i,j,k}$ to which they contribute and to add these contributions to these RHS vectors. To do this it is necessary to notice that a RHS vector $\mathbf{rhs}_{i,j,k}$ is composed of terms of the form $\frac{\partial \mathbf{R}_{l,m,n}}{\partial \mathbf{P}_{i,j,k}}^t . \boldsymbol{\lambda}_{l,m,n}$ where l, m, n vary. Once again this has to be repeated for the fluxes in the three directions.

6.6.4.2 At the boundaries

Two different problems have to be distinguished and are detailed next: the first one is when the turbulent viscosity is calculated inside the domain and the second one, when it is taken from halo cells.

When the turbulent viscosity is calculated inside the domain, this means that the whole ray along which it is computed is inside the domain as well. However cells at $i+1, j, k$

[2]Constant here means independent of $\mathbf{P}$ in the sens that $\frac{\partial \mu_t}{\partial \mathbf{P}} = \mathbf{0}$ but of course μ_t is not a constant in the flow solver since it is calculated using the Baldwin-Lomax model and hence depends on the local flow properties. A synonym is μ_t is frozen.

or $i, j_{maxF} + 1, k$ for example might be outside the domain. As usual for these types of cells, their direct contributions to the RHS vectors $\mathbf{rhs}_{i,j,k}$ are discarded but they are added indirectly to cells that are inside the domain. This is not detailed but is very similar to what is happening to the first-order inviscid fluxes in section 6.6.1.2.

When the value of the turbulent viscosity is taken from halo cells, it results from the application of the boundary conditions described in section 5.6.3 and is not calculated along rays inside the halo cells. This in fact simplifies the problem. Let us consider the flux $\mathbf{F}_{3/2}$ of Figure 6.8. Its turbulent viscosity is

$$(\mu_t)_{3/2} = \frac{1}{2}[(\mu_t)_1 + (\mu_t)_2] \tag{6.21}$$

Depending on the boundary type, this is either

$$(\mu_t)_{3/2} = 0$$

or

$$(\mu_t)_{3/2} = (\mu_t)_2$$

or

$$(\mu_t)_{3/2} = \frac{1}{2}(\mu_t)_2$$

$(\mu_t)_2$ is now inside the domain so the computation of $\dfrac{\partial \mathbf{F}_{3/2}}{\partial \mathbf{P}_l}$ comes back to the previous case where the turbulent viscosity is calculated inside the domain but has dependency cells outside the domain. In this particular case, the dependency cells $1, j, k$ or $1, j_{maxF}, k$ are in the halo cells and their contribution would have to be reflected in cells $2, j, k$ and $2, j_{maxF}, k$ respectively. Again this is only a one-dimensional example while in three dimensions, the fluxes in all three directions have to be accounted for.

6.6.4.3 Interface boundary

The interface boundary condition ensures that the value of μ_t in the halo cells is taken from the neighbouring block where it was calculated. Hence everything is as if the turbulent viscosity was calculated along j rays inside the halo cells. Note that this assumes that the interface boundary is an i or k boundary aligned with the j direction. Since it is assumed that the turbulent viscosity is calculated along j rays starting from the surface at $j = 2$ and going to the farfield boundary at $j = jn$, this is not restrictive since a j boundary is never an interface boundary for these blocks. What happens in the wake is described in the next subsection.

Let us consider again the case of Figure 6.8 where this time the interface at $j = 3/2$ is an interface boundary. Equation (6.21) is still valid but this time there is not any connection between $(\mu_t)_1$ and $(\mu_t)_2$ hence the calculation of $\dfrac{\partial \mathbf{F}_{3/2}}{\partial \mathbf{P}_l}$ involves the calculation of

$\frac{\partial(\mu_t)_1}{\partial \mathbf{P}_l}$ and $\frac{\partial(\mu_t)_2}{\partial \mathbf{P}_l}$. However this linearisation is meant to be used in the computation of the RHS vector $\mathbf{rhs}_{i,j,k}$ and this RHS is only needed inside the domain. Hence only $\mathbf{rhs}_{2,j,k}$ is affected by $(\mu_t)_1$. Since $\mathbf{rhs}_{2,j,k}$ is only composed of terms of the form $\frac{\partial \mathbf{R}_{l,m,n}}{\partial \mathbf{P}_{2,j,k}}^t.\boldsymbol{\lambda}_{l,m,n}$, only terms involving $\frac{\partial(\mu_t)_1}{\partial \mathbf{P}_{2,j,k}}$ need to be calculated. This is convenient since it means that only two terms have to be calculated involving $(\mu_t)_1$, i.e. $\frac{\partial(\mu_t)_1}{\partial \mathbf{P}_{2,j,k}}$ and/or $\frac{\partial(\mu_t)_1}{\partial \mathbf{P}_{2,j_{maxF},k}}$. No such simplification occurs for $\frac{\partial(\mu_t)_2}{\partial \mathbf{P}_l}$ that is computed as a turbulent viscosity inside the domain.

6.6.4.4 In the wake

What has been presented so far in this section on the linearisation of the turbulence model and its incorporation into the calculation of the RHS vector of the adjoint equation, is only valid for blocks where the turbulent viscosity is calculated along rays with the Baldwin-Lomax model. As mentioned in section 5.6.2.2, in the wake behind an aerofoil or a wing, the turbulent viscosity is not actually computed but is copied from the last ray at the trailing edge of the aerofoil or wing. This necessitates a different treatment of the linearisation of μ_t from what has been described so far.

The problem is similar to what happens at the interface boundary because there needs to be an interface boundary between the block where μ_t is calculated and the block in the wake where it is copied. Let us take the example of Figure 6.8 again where the block considered is in the wake and the boundary at $i = 3/2$ is the interface with the block where μ_t is calculated. Hence cells with $i \leq 1$ are situated above the aerofoil or wing and cells with $i \geq 2$ are in the wake. Since the turbulent viscosity is copied,

$$(\mu_t)_{3/2} = (\mu_t)_{5/2} = (\mu_t)_{i+1/2} = (\mu_t)_1$$

Hence the derivative $\frac{\partial \mathbf{F}_{i+1/2}}{\partial \mathbf{P}_l}$ involves only the calculation of $\frac{\partial(\mu_t)_1}{\partial \mathbf{P}_l}$. Like in the case of the interface, only values of $\mathbf{rhs}_{i,j,k}$ with i, j, k inside the domain need to be calculated. Since everywhere inside the domain, μ_t only depends on what happens at the interface boundary, the only terms that remain are $\frac{\partial(\mu_t)_1}{\partial \mathbf{P}_{2,j,k}}$. More specifically, only two terms remain as in the case of the interface boundary i.e. $\frac{\partial(\mu_t)_1}{\partial \mathbf{P}_{2,j,k}}$ and/or $\frac{\partial(\mu_t)_1}{\partial \mathbf{P}_{2,j_{maxF},k}}$. This means that for the whole wake block, only RHS vectors of the form $\mathbf{rhs}_{2,j,k}$ where j and k vary and $\mathbf{rhs}_{2,j_{maxF},k}$ where k varies, need to be calculated. The formula to calculate them is simple with for example

$$\mathbf{rhs}_{2,j,k} = \sum_{i=1}^{in} \frac{\partial \mathbf{R}_{i,j,k}}{\partial \mathbf{P}_{2,j,k}}^t .\boldsymbol{\lambda}_{i,j,k}$$

Note that in the sum, the term at $i = in$ will have to take into account a boundary condition, likewise for the whole equation when j or k are close to a boundary. Another aspect not to forget is that this treatment has only been presented when the interface between the block where μ_t is calculated and the wake block is at $i = 3/2$ but of course this could happen at $i = in + 1/2$. Finally this was only for the fluxes in the i direction but a similar methodology has to be applied for the fluxes in the other two directions.

This concludes this long section that detailed the calculation of the RHS vector of the adjoint equation with the term by term matrix-vector product involving contributions to an exact Jacobian and the adjoint vector. This showed first how the contribution from the convective fluxes is incorporated in this equation before presenting what happens to the diffusive fluxes with first the viscous laminar part and then the linearisation of the Baldwin-Lomax turbulence model. In each case, details were given for the general treatment inside the domain and then it was shown what happens close to a boundary, the interface boundary being very different from the other boundaries. To end this chapter on the discrete adjoint solver, now that it has been presented in detail, results that assess the accuracy of the adjoint code are shown.

6.7 Verification

One of the problems encountered when trying to implement an adjoint solver is its validation. The only possibility is to compute a sensitivity derivative both with equation (6.8) and the adjoint vector, and by finite-differences. It is the only way to check the correctness of the method.

This subsection presents examples of calculation of the adjoint vector and of sensitivity derivatives that are compared with finite-difference calculations. All these calculations are quasi-two-dimensional i.e. the flow and adjoint solvers are three-dimensional but the computational grid has only one cell in the k direction with a symmetry boundary condition in the two k planes. To be able to compute the flow on such a grid, the convective fluxes in the k direction are limited to first-order. The choice of doing only two-dimensional accurracy studies is dictated by computing reasons. Indeed to perform finite-difference sensitivity calculations, the flow solution has to be very well-converged in order to obtain a difference in the computed objective functions that is due to design variable changes and not to poor convergence. This implies two things: first that it is CPU time consuming to do such calculations hence the smaller the grid, the better and secondly that you need a problem that is able to converge to the level you require. From the experience of the author, three-dimensional problems rarely converge to machine zero while two-dimensional problems are more likely to do it. For these two reasons, only two-dimensional problems were used to assess the accuracy of the adjoint solver.

		Central finite difference			Adjoint	Percentage difference
		$\varepsilon = 10^{-4}$	$\varepsilon = 10^{-6}$	$\varepsilon = 10^{-8}$		
$\frac{dC_L}{d\beta_k}$	3rd Bézier parameter	0.5273168	0.5273169	0.5273169	0.5273075	0.002 %
	6th Bézier parameter	-0.1796504	-0.1796505	-0.1796505	-0.1795517	0.055 %
$\frac{dC_D}{d\beta_k}$	3rd Bézier parameter	0.09127480	0.09127479	0.09127475	0.09128026	0.006 %
	6th Bézier parameter	0.13399757	0.13399757	0.1339976	0.13401192	0.011 %

Table 6.1: Comparison of sensitivity derivatives calculated by finite difference and by the adjoint method for the NACA0012 aerofoil for a laminar flow.

Two tests are presented here. The first one involves only laminar viscous flow computations. They are performed on a NACA0012 aerofoil on a 4-block coarse $129 \times 33 \times 2$ grid. The flow conditions are: Mach number $M_\infty = 0.80$, incidence $\alpha = 1.0\,°$, Reynolds number $Re = 500$ based on chord and freestream temperature $T_\infty = 16.0\,K$. The second test is made on the RAE2822 aerofoil for a fully turbulent flow. The grid has the same characteristics: 4 blocks for a total of $129 \times 33 \times 2$ points. It is shown in Figure 4.1(a). The flow conditions for this test case are: Mach number $M_\infty = 0.725$, incidence $\alpha = 2.54\,°$, Reynolds number $Re = 6.5 \times 10^6$ based on chord and freestream temperature $T_\infty = 283.0\,K$. At these conditions a shock wave forms on the upper surface of the RAE2822 aerofoil at 60% of the chord. The upper surface of both aerofoils was parameterised using 10 Bézier design variables and sensitivities to the 3rd and 6th are investigated. For the RAE2822 aerofoil sensitivities to the angle of incidence α are also considered. The objective functions are lift and drag coefficients in addition to pitching moment coefficient for the RAE2822 aerofoil.

The results for the NACA0012 aerofoil for a viscous laminar flow are presented in Table 6.1. The flow solver and the adjoint solver are always converged to a total residual of 10^{-12} for this case. This accuracy study also investigated the accuracy of the finite difference method. Three step sizes ε were chosen i.e. 10^{-4}, 10^{-6} and 10^{-8}. Table 6.1 shows that the finite difference method is not very sensitive to the step size with a very good agreement between the three results. For a step size of 10^{-8}, there were not enough available significative digits to be able to distinguish the three results for $\frac{dC_D}{d\beta_k}$. The agreement between finite difference and the adjoint solver is excellent with 3 to 4 significative digits in common and obviously the correct sign each time (so the right direction in optimisation).

		Central finite difference ($\varepsilon = 10^{-6}$)	Adjoint	Percentage difference
$\dfrac{dC_L}{d\beta_k}$	3rd Bézier parameter	1.29579	1.29312	0.21 %
	6th Bézier parameter	-0.440811	-0.439373	0.33 %
	Incidence α	0.141057	0.140811	0.17 %
$\dfrac{dC_D}{d\beta_k}$	3rd Bézier parameter	-0.00825440	-0.00824008	0.17 %
	6th Bézier parameter	0.206073	0.205986	0.04 %
	Incidence α	0.00817769	0.00817888	0.01 %
$\dfrac{dC_m}{d\beta_k}$	3rd Bézier parameter	-0.126766	-0.128028	1.00 %
	6th Bézier parameter	0.130785	0.131409	0.48 %
	Incidence α	0.0342890	0.0343446	0.16 %

Table 6.2: Comparison of sensitivity derivatives calculated by finite difference and by the adjoint method for the RAE2822 aerofoil for a fully turbulent flow.

The results for the RAE2822 aerofoil for a fully turbulent flow are presented in Table 6.2. For this test case the flow solver is converged to 2.7×10^{-12} because the convergence stalled around this value and did not go beyond. This is however already a very good convergence level. There was no problem for the adjoint solver so each time it is converged to 10^{-12}. Since the previous study did not show a significant sensitivity of the finite difference method to the step size, a value of 10^{-6} that seemed reasonable is chosen here. The agreement between finite difference and adjoint method is not as good as for the previous case, but it is nevertheless still very good. The difference is supposed to come from the slight approximations made when differentiating the maximum function in the linearisation of the Baldwin-Lomax turbulence model. Notice however that for this case the order of magnitude of the sensitivity derivatives is quite different from one objective function to another or from one design variable to another, yet the adjoint method is accurate each time and of course finds the correct sign even for very low values.

As explained previously, the adjoint vector does not have any obvious physical meaning so very few people[19,43,157,232] show a representation of it. Since it has the same number of components as the flow variable vector, it can be depicted as a flow field. This is what is done in Figure 6.12 and Figure 6.13. Figure 6.12 represents the first component of the adjoint vector when the objective function is the drag coefficient, for the previous laminar viscous flow around the NACA0012 aerofoil. Figure 6.13 shows the same component of the adjoint vector for the same objective function but this time for the RAE2822 aerofoil in the fully turbulent flow mentioned above. In these two Figures, the outline of the 4 blocks is also represented showing the reasonable continuity of the adjoint vector at the interface boundaries. Similar features to these Figures 6.12 and 6.13 can be found in References [19, 232] i.e. that the contours look like the flow contours that could be obtained if the flow was coming from the opposite direction, with a sort of stagnation

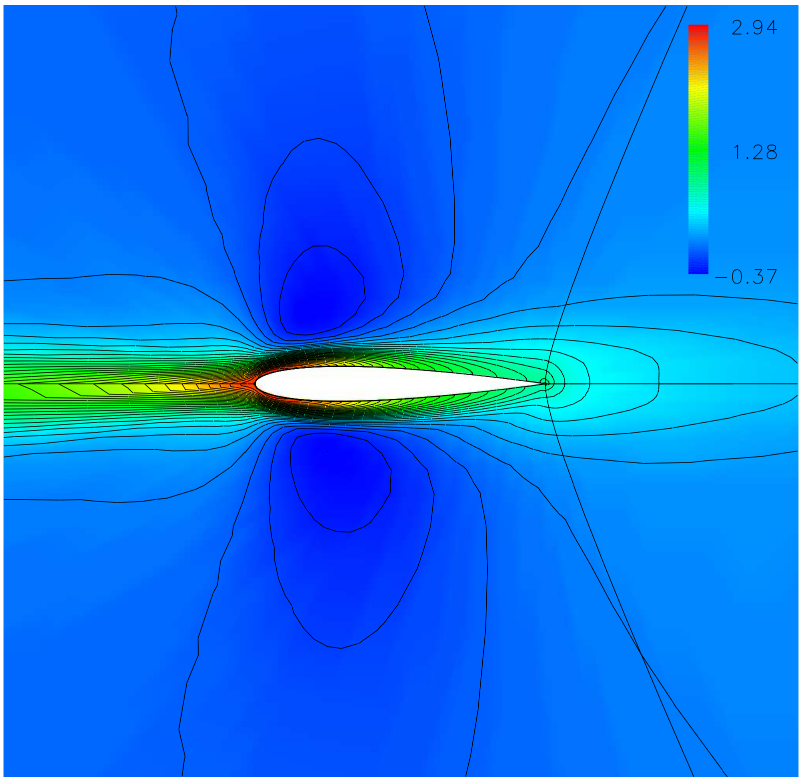


Figure 6.12: First component of the adjoint vector when the objective function is the drag coefficient for a laminar viscous flow around the NACA0012 aerofoil.

point at the trailing edge and a wake ahead of the leading edge. This is particularly true for the laminar flow of Figure 6.12.

The conclusion of this section on the validation of the discrete adjoint solver, is that it is very accurate when comparing to finite difference sensitivity derivatives. Hence it can be incorporated without any problem into an optimisation chain to provide the gradients needed by the optimiser.

This is the end of the chapter describing the discrete adjoint solver that was coded for this study. This chapter first presented the discrete and continous adjoint methods before giving some reasons for chosing the discrete method in this work. The adjoint equations were cast in a form very similar to that of the flow solver MERLIN, enabling the use of the same numerical methodology to solve them. Then the innovative content of this adjoint solver was highlighted, most of it coming from the accurate treatment of the RHS Jacobian that was then described. That lengthy part detailed the derivation of all the contributions to this RHS Jacobian starting with the first-order accurate convective fluxes, then the higher-order accurate convective fluxes and finally the viscous laminar fluxes and the linearisation

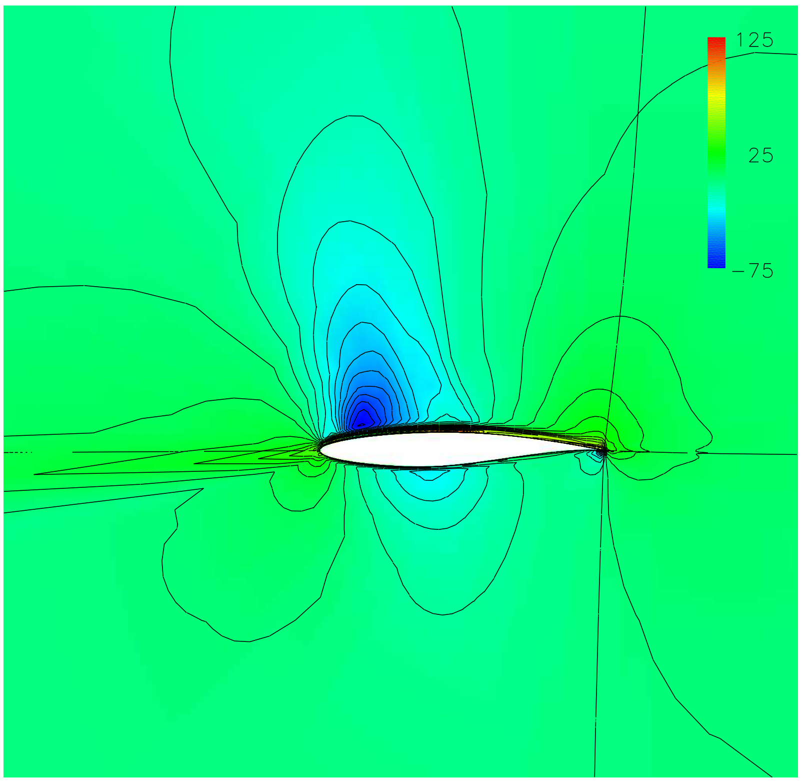


Figure 6.13: First component of the adjoint vector when the objective function is the drag coefficient for a fully turbulent flow around the RAE2822 aerofoil.

of the Baldwin-Lomax turbulence model. For all these contributions, their matrix-vector product with the adjoint vector and their incorporation into the RHS vector of the adjoint equation was presented. The last part demonstrated the accuracy of this adjoint solver that is now ready to be introduced into an optimisation chain to perform some optimisation as the next chapter shows.

This page has been left intentionally blank.