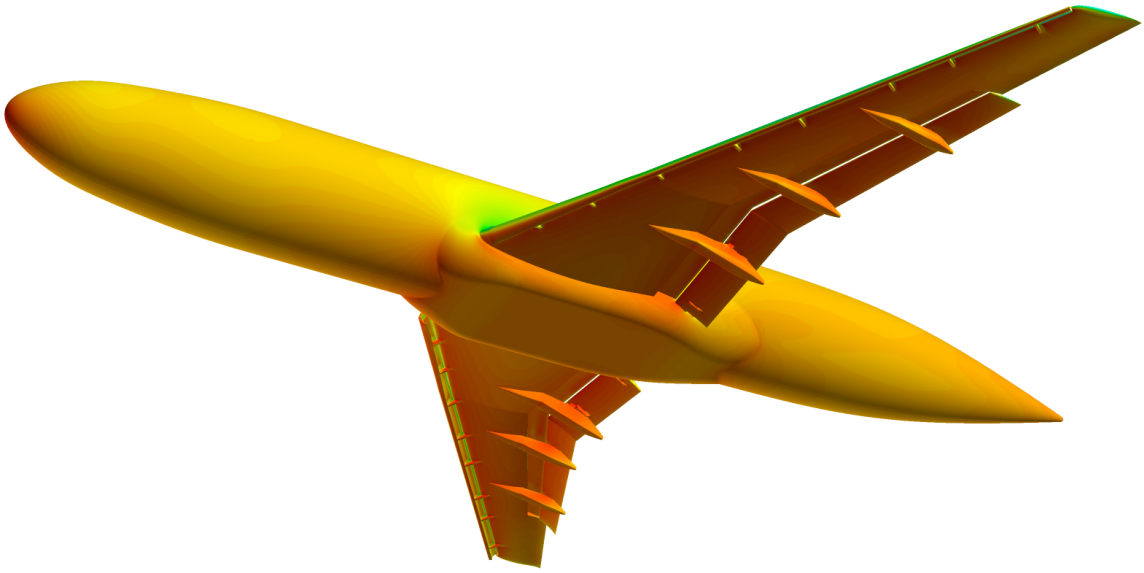
THE ARION UNSTRUCTURED GRIDS CFD SOLVER: THEORETICAL AND USER'S MANUAL

Version 2.80 (Build 71c4bb5)

ISCFDC Report 2026-04



Prepared by

Ya'eer Kidron, Sahar Shpitz, and Yuval Levy



ISCFDC LTD

COPYRIGHT © September 2026

by

The authors and the Israeli CFD Center

Copyright and Trademark Information

© 2025 ISCFDC LTD. All rights reserved. Unauthorized use, distribution or duplication is prohibited. See the legal information in the product license for the complete Legal Notice for ISCFDC proprietary software. If you are unable to access the Legal Notice, please contact ISCFDC.

Third-Party Software

The PETSc Open Source Library

This software makes use of the PETSc open-source library.
The following is legal information pertaining to the use and redistribution of PETSc:

Copyright (c) 1991-2021, UChicago Argonne, LLC and the PETSc Development Team
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ”AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,

EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The CGNS Library

This software makes use of the CGNS (CFD General Notation System) library.

The HDF Library

This software makes use of the HDF5 (Hierarchical Data Format 5) Software Library and Utilities Copyright 2006 by The HDF Group and the NCSA HDF5 (Hierarchical Data Format 5) Software Library and Utilities Copyright 1998-2006 by The Board of Trustees of the University of Illinois.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted for any purpose (including commercial purposes) provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the following disclaimer in the documentation and/or materials provided with the distribution.

3. Neither the name of The HDF Group, the name of the University, nor the name of any Contributor may be used to endorse or promote products derived from this software without specific prior written permission from The HDF Group, the University, or the Contributor, respectively.

DISCLAIMER: THIS SOFTWARE IS PROVIDED BY THE HDF GROUP AND THE CONTRIBUTORS "AS IS" WITH NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED. IN NO EVENT SHALL THE HDF GROUP OR THE CONTRIBUTORS BE LIABLE FOR ANY DAMAGES SUFFERED BY THE USERS ARISING OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

You are under no obligation whatsoever to provide any bug fixes, patches, or upgrades to the features, functionality or performance of the source code ("Enhancements") to anyone; however, if you choose to make your Enhancements available either publicly, or directly to The HDF Group, without imposing a separate written license agreement for such Enhancements, then you hereby grant the following license: a non-exclusive, royalty-free perpetual license to install, use, modify, prepare derivative works, incorporate into other computer software, distribute, and sublicense such enhancements or derivative works thereof, in binary and source code form.

Limited portions of HDF5 were developed by Lawrence Berkeley National Laboratory (LBNL). LBNL's Copyright Notice and Licensing Terms can be found here: COPYING_LBNL_HDF5 file in this directory or at COPYING_LBNL_HDF5.

Contributors: National Center for Supercomputing Applications (NCSA) at the University of Illinois, Fortner Software, Unidata Program Center (netCDF), The Independent JPEG Group (JPEG), Jean-loup Gailly and Mark Adler (gzip), and Digital Equipment Corporation (DEC).

Portions of HDF5 were developed with support from the Lawrence Berkeley National Laboratory (LBNL) and the United States Department of Energy under Prime Contract No. DE-AC02-05CH11231.

Portions of HDF5 were developed with support from Lawrence Livermore National Laboratory and the United States Department of Energy under Prime Contract No. DE-AC52-07NA27344.

Portions of HDF5 were developed with support from the University of California, Lawrence Livermore National Laboratory (UC LLNL). The following statement applies to those portions of the product and must be retained in any redistribution of source code, binaries, documentation, and/or accompanying materials:

This work was partially produced at the University of California, Lawrence Livermore National Laboratory (UC LLNL) under contract no. W-7405-ENG-48 (Contract 48) between the U.S. Department of Energy (DOE) and The Regents of the University of California (University) for the operation of UC LLNL.

DISCLAIMER: THIS WORK WAS PREPARED AS AN ACCOUNT OF WORK SPONSORED BY AN AGENCY OF THE UNITED STATES GOVERNMENT. NEITHER THE UNITED STATES GOVERNMENT NOR THE UNIVERSITY OF CALIFORNIA NOR ANY OF THEIR EMPLOYEES, MAKES ANY WARRANTY, EXPRESS OR IMPLIED, OR ASSUMES ANY LIABILITY OR RESPONSIBILITY FOR THE ACCURACY, COMPLETENESS, OR USEFULNESS OF ANY INFORMATION, APPARATUS, PRODUCT, OR PROCESS DISCLOSED, OR REPRESENTS THAT ITS USE WOULD NOT INFRINGE PRIVATELY- OWNED RIGHTS. REFERENCE HEREIN TO ANY SPECIFIC COMMERCIAL PRODUCTS, PROCESS, OR SERVICE BY TRADE NAME, TRADEMARK, MANUFACTURER, OR OTHERWISE, DOES NOT NECESSARILY CONSTITUTE OR IMPLY ITS ENDORSEMENT, RECOMMENDATION, OR FAVORING BY THE UNITED STATES GOVERNMENT OR THE UNIVERSITY OF CALIFORNIA. THE VIEWS AND OPINIONS OF AUTHORS EXPRESSED HEREIN DO NOT NECESSARILY STATE OR REFLECT THOSE OF THE UNITED STATES GOVERNMENT OR THE UNIVERSITY OF CALIFORNIA, AND SHALL NOT BE USED FOR ADVERTISING OR PRODUCT ENDORSEMENT PURPOSES.

The METIS Library

Copyright 1995-2013, Regents of the University of Minnesota

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at METIS License (Apache License Version 2.0).

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for

the specific language governing permissions and limitations under the License.

Contents

Abstract **xxvii**

1 Introduction **1**
1.1 Current Release . . . 1
1.2 Users's Manual Arrangement . . . 2

2 Physical Models **3**
2.1 Introduction . . . 3
2.2 Single-Component Perfect Gas (SPG) . . . 3
2.2.1 Governing Equations . . . 3
2.2.2 Integral Form For Moving Grids . . . 5
2.2.3 Equation of State . . . 6
2.2.4 Transport Properties . . . 7
2.3 Multi-component Gas (MCG) . . . 8
2.3.1 Basic Assumptions . . . 9
2.3.2 Governing Equations . . . 9
2.3.3 Integral Form For Moving Grids . . . 12
2.3.4 Thermodynamic Properties . . . 14
2.3.4.1 Thermal Equilibrium . . . 14
2.3.5 Transport Properties . . . 14
2.3.5.1 Wilke's Model . . . 15
2.3.6 Chemical Reaction Modeling . . . 16
2.3.6.1 Equilibrium Gas Properties Tabulation . . . 17
2.3.6.2 Finite-rate Chemical Kinetics . . . 17

2.3.6.3 Reaction Rate Coefficients . . . 17
2.4 Gas Selection . . . 18
2.4.1 Single Component Perfect Gas . . . 18
2.4.2 Multi Component Gas . . . 19

3 Turbulence Models 20
3.1 RANS Turbulence Model Equations . . . 21
3.2 The Spalart Allmaras Turbulence Model . . . 23
3.2.1 SA-2020 Model . . . 25
3.2.2 Boundary Conditions . . . 26
3.3 Formulation of k-ω Turbulence Models . . . 27
3.3.1 The k-ω-TNT Turbulence Model . . . 28
3.3.2 The k-ω-SST Model . . . 28
3.3.2.1 k-ω-SST-2003 Model . . . 29
3.3.2.2 k-ω-SST-SAS Model . . . 31
3.4 Boundary Conditions for the k-ω-TNT and k-ω-SST Turbulence Models 32
3.5 Evaluation of the Reynolds Stress Tensor . . . 33
3.6 Turbulence-Mean-flow Coupling . . . 33
3.7 Hybrid RANS/LES Turbulence Models . . . 34
3.7.1 X-LES Hybrid RANS/LES Turbulence Model . . . 34
3.7.2 IDDES Formulation . . . 36
3.7.2.1 The SA-IDDES Turbulence Model . . . 39
3.7.2.2 The k-ω-SST-IDDES Hybrid Turbulence Model . . . 39
3.7.3 Subgrid Length Scales . . . 40
3.7.3.1 IDDES Subgrid Length Scale . . . 40
3.7.3.2 Shear-Layer-Adapted Length Scale . . . 41

4 Computational Methods 43
4.1 Spatial Discretization . . . 43
4.2 Flux Approximation Schemes . . . 44
4.2.1 HLLC . . . 44
4.2.2 AUSM . . . 47

4.2.2.1 AUSM$^+$-up . . . 48
4.2.2.2 AUSM-DV . . . 50
4.2.3 Steger-Warming Flux Vector Splitting . . . 52
4.2.4 Passive Scalar Approach . . . 54
4.2.5 High Order Flux Approximations . . . 54
4.3 Diffusive Flux Vector Discretization . . . 55
4.4 Time Marching Schemes . . . 55
4.4.1 Explicit schemes . . . 56
4.4.1.1 Explicit Euler Scheme . . . 56
4.4.1.2 Runge-Kutta Schemes . . . 56
4.4.1.3 Time-Accurate Third Order Runge-Kutta TVD . . . 57
4.4.2 Implicit Time Marching Formulation . . . 57
4.4.2.1 Point Gauss-Seidel . . . 58
4.4.2.2 Runge-Kutta Schemes . . . 58
4.5 Convergence Acceleration Methods . . . 58
4.5.1 Krylov Methods . . . 58
4.5.1.1 Preconditioning . . . 59
4.5.1.2 Portable, Extensible Toolkit for Scientific Computation (PETSc) . . . 60
4.5.2 Line Search . . . 60
4.5.2.1 Back-Traced Line Search . . . 60

5 Boundary Conditions **63**
5.1 Introduction . . . 63
5.2 Wall Boundary Conditions . . . 63
5.2.1 Impermeable Wall Conditions . . . 63
5.2.2 No-Slip Condition . . . 64
5.2.3 Adiabatic Wall . . . 64
5.2.4 Wall Function . . . 64
5.3 Far Field Conditions . . . 66
5.3.1 Turkel Type Conditions . . . 66



5.3.1.1 Turkel Inlet . . . 66
5.3.1.2 Turkel Outlet . . . 66
5.3.2 Riemann Type Conditions . . . 67
5.3.2.1 Riemann Inlet . . . 67
5.3.2.2 Riemann Outlet . . . 67
5.3.3 Fixed (Supersonic Inlet) . . . 68
5.3.4 Extrapolation (Supersonic Outlet) . . . 68
5.3.5 Inlet . . . 68
5.3.6 Outlet . . . 68
5.3.6.1 Prescribed Mass Flow Rate . . . 69
5.3.7 Inout . . . 69
5.3.8 Mass Flow Inlet . . . 70
5.4 Stagnation Inlet . . . 73
5.5 Symmetry Boundary Conditions . . . 74

6 Computational Mesh **75**
6.1 Star-CD Export . . . 75
6.2 CGNS Export . . . 75
6.3 Automated Mesh Adaptation . . . 76

7 Aeroelasticity **77**
7.1 Static Aeroelasticity . . . 78
7.2 Aeroelastic Model . . . 79
7.2.1 Structural Modal Data . . . 79
7.2.2 Elastic and Rigid-Body Modes . . . 80
7.3 Structural and Aerodynamic Coordinate Systems . . . 81
7.4 Structural-to-Aerodynamic Mapping . . . 82
7.4.1 Work-Conservative Generalized-Force Projection . . . 83
7.4.2 Radial Basis Function Mapping . . . 83
7.4.3 Rotation-Aware RBF Mapping . . . 84
7.4.4 One-Dimensional Beam Mapping . . . 85
7.4.5 Infinite Beam Spline Mapping . . . 85



7.4.6 Infinite Plate Spline Mapping . . . 86
7.4.7 Choice of Mapping Method . . . 86
7.4.8 Aeroelastic Module Mapping Setup . . . 87
7.5 Static Aeroelastic Coupling . . . 87
7.6 Rigidly-Attached Aerodynamic Components . . . 89
7.7 Aeroelastic Surface and Volume-Mesh Deformation . . . 90
7.7.1 Mesh Stiffness Distribution . . . 91
7.7.2 Mesh-Deformation Solution Accuracy . . . 92
7.8 Restart and Continuation . . . 92
7.9 Assumptions and Range of Applicability . . . 93

8 Parallelization **95**

9 Input File, Run Preparation, and Execution **96**
9.1 Preface . . . 96
9.1.1 Scripting Language . . . 97
9.1.2 Basic Input File: Examples . . . 98
9.1.2.1 Single Component Gas Example . . . 98
9.1.2.2 Multi Component Gas Example . . . 100
9.2 Grid Files . . . 105
9.3 Macros . . . 106
9.3.1 Def . . . 106
9.4 Conversion . . . 107
9.4.1 Conversion Between Different Orderings . . . 110
9.5 Parallel Options . . . 111
9.6 Equation of State Options . . . 113
9.7 Molecular Options . . . 117
9.8 Tabulate Options . . . 124
9.9 Multi Component Gas Options . . . 128
9.10 Aeroelasticity . . . 140
9.10.1 Aeroelasticity Options . . . 140
9.10.1.1 Radial Basis Function Options . . . 148



9.10.2 Aeroelasticity Component . . . 152
9.10.2.1 Structural Input Options . . . 154
9.10.2.2 Structural Transformation Options . . . 156
9.10.2.3 Mapping Options . . . 157
9.10.2.4 Radial Basis Function Component Options . . . 158
9.10.2.5 Rigid Attachment Options . . . 160
9.11 BC Options . . . 162
9.11.1 BC Log Options . . . 169
9.11.2 Wall Distance Calculations . . . 171
9.11.3 Free-Stream Conditions Override . . . 172
9.12 System Options . . . 177
9.12.1 Flow Equations . . . 179
9.12.1.1 Turbulence Model Equations System Options . . . 188
9.12.1.2 Mean Flow Equations for Ideal Gas . . . 190
9.12.1.3 Mean Flow Equations for Multi Component Gas under Chemical Equilibrium . . . 191
9.12.1.4 Mean Flow Equations for Multi Component Gas . . . 192
9.12.1.5 Turbulence Model Equations . . . 194
9.12.2 Flow Equations System Outputs . . . 195
9.12.2.1 System Log Options . . . 196
9.12.2.2 System Plot Functions . . . 198
9.12.3 Elastic Mesh-Deformation System . . . 202
9.12.3.1 Solution using PETSc . . . 207
9.13 Solve Options . . . 208
9.13.1 Wall Distance Evaluation . . . 234
9.13.2 Line Search . . . 236
9.14 PETSc Toolkit . . . 239
9.15 Domain Options . . . 246
9.16 Init Options . . . 250
9.17 Library Options . . . 254
9.18 User Defined Source Options . . . 255



9.19 Var Options . . . 258
9.19.1 Var Examples . . . 259
9.19.1.1 Constant Function . . . 259
9.19.1.2 Linear Function . . . 259
9.19.1.3 Trigonometric Function . . . 259
9.19.1.4 Tabulated Function . . . 259
9.20 Vertex Option . . . 260
9.21 UNS Options . . . 261
9.21.1 Motion Options . . . 271
9.22 Point Cloud Data . . . 279
9.23 Log Control Options . . . 284
9.24 Plot Control Options . . . 287
9.25 Table Control Options . . . 291
9.26 Post Option . . . 296
9.27 Converge Options . . . 297
9.28 Run-time Options . . . 300
9.28.1 Run-time Options Examples . . . 300
9.29 User Source Library Interface . . . 303
9.29.1 Input Definition . . . 303
9.29.2 Exported Library Functions . . . 304
9.29.3 Runtime Flow . . . 306
9.29.4 User Context . . . 307
9.29.5 Variable Lookup . . . 308
9.29.6 Available Variables . . . 309
9.29.7 Equation Lookup . . . 309
9.29.8 Available Equations . . . 311
9.29.9 RHS Convention . . . 312
9.29.10 Jacobian Convention . . . 313
9.29.11 Thread Safety . . . 314
9.29.12 ABI Versioning . . . 314
9.29.13 Minimal Heat Source Example . . . 315



Bibliography **325**

List of Figures

4.1 Wave structure of the HLLC approximate Riemann solver 45

9.1 Observer and body reference frames 271

List of Tables

2.1 Default coefficients for Sutherland formulae (type 1) 8
2.2 Default coefficients for Sutherland formulae (type 2) 8

3.1 SST coefficients 30

9.1 Def macro 106
9.2 Conversion 107
9.3 Metis options 108
9.3 Metis options (continued) 109
9.4 Conversion between different ordering (cnv2cnv) option 110
9.5 Parallel cache option 111
9.6 Parallel rank option 112
9.7 equation.of.state name option 113
9.8 equation.of.state type option 114
9.9 EOS types 114
9.10 equation.of.state R option 115
9.11 equation.of.state gamma option 115
9.12 equation.of.state ref option 116
9.13 Molecular name option 118
9.14 Molecular type option 119
9.15 Molecular viscosity option 120
9.16 Molecular conductivity option 120
9.17 Molecular Mu1 option 120
9.18 Molecular Mu2 option 121
9.19 Molecular Ka1 option 121

9.20 Molecular Ka2 option . . . 121
9.21 Molecular Eta0 option . . . 122
9.22 Molecular C0 option . . . 122
9.23 Molecular T0 option . . . 122
9.24 Molecular Prandtl option . . . 123
9.25 Tabulate name option . . . 125
9.26 Tabulate type option . . . 125
9.27 Tabulate distribution option . . . 126
9.28 Tabulate key option . . . 126
9.29 Tabulate pT.chemistry option . . . 127
9.30 Multi component gas name option . . . 128
9.31 Multi component gas composition option . . . 129
9.32 Multi component gas composition molar option . . . 129
9.33 Multi component gas cy.eq option . . . 130
9.34 Multi component gas cy.cutoff option . . . 130
9.35 Multi component gas thrm.db option . . . 131
9.36 Multi component gas trns.db option . . . 131
9.37 Multi component gas react.db option . . . 132
9.38 Multi component gas cantera.db option . . . 132
9.39 Multi component gas thrm option . . . 133
9.40 Multi component gas trns option . . . 134
9.41 Multi component gas viscosity option . . . 135
9.42 Multi component gas conductivity option . . . 135
9.43 Multi component gas kinetics option . . . 136
9.44 Multi component gas eqconst option . . . 137
9.45 Multi component gas eq.engine option . . . 137
9.46 Multi component gas cantera.diagonal.jacobian option . . . 138
9.47 Multi component gas Sc option . . . 138
9.48 Multi component gas Sc.trb option . . . 138
9.49 Multi component gas phase transition option . . . 139
9.50 Aeroelastic relax option . . . 141



9.51 Aeroelastic start from option . . . 141
9.52 Aeroelastic update every option . . . 141
9.53 Aeroelastic convergence relative tolerance option . . . 142
9.54 Aeroelastic convergence absolute tolerance option . . . 142
9.55 Aeroelastic displacement relative tolerance option . . . 143
9.56 Aeroelastic displacement absolute tolerance option . . . 143
9.57 Aeroelastic minimum updates option . . . 144
9.58 Aeroelastic maximum updates option . . . 144
9.59 Aeroelastic elastic modes option . . . 144
9.60 Aeroelastic modes option . . . 145
9.61 Aeroelastic modes rigid tolerance option . . . 145
9.62 Aeroelastic modes normalization option . . . 146
9.63 Aeroelastic modes eigenvalue relative tolerance option . . . 147
9.64 Aeroelastic RBF singular tolerance option . . . 148
9.65 Aeroelastic RBF geometry dimension tolerance option . . . 148
9.66 Aeroelastic RBF projection duplicate tolerance option . . . 149
9.67 Aeroelastic RBF enable mapping warnings option . . . 149
9.68 Aeroelastic export option . . . 150
9.69 Aeroelastic import option . . . 150
9.70 Aeroelastic plot option . . . 151
9.71 Aeroelastic log option . . . 151
9.72 Aeroelastic component name option . . . 153
9.73 Aeroelastic component bc option . . . 153
9.74 Aeroelastic component nastran.grid.file option . . . 154
9.75 Aeroelastic component nastran.modes.file option . . . 154
9.76 Aeroelastic component zaero.free.format.file option . . . 155
9.77 Aeroelastic component points.for.spline.file option . . . 155
9.78 Aeroelastic component structural transform option . . . 156
9.79 Aeroelastic component mapping option . . . 157
9.80 Mapping types . . . 157
9.81 Aeroelastic component RBF kernel option . . . 158



9.82 Aeroelastic component RBF coordinates option 158
9.83 Aeroelastic component RBF polynomial option 159
9.84 Aeroelastic component attach to option 160
9.85 Aeroelastic component attach by option 161
9.86 BC name option 162
9.87 BC type option 163
9.88 Boundary condition types 164
9.89 BC mass.flow option 165
9.90 BC relax option 165
9.91 BC delay option 166
9.92 BC reference.pressure option 166
9.93 BC gas composition option 167
9.94 BC gas composition molar option 167
9.95 BC gas cy.eq option 168
9.96 BC aeroelastic deformation slip option 168
9.97 Boundary conditions log option 169
9.98 BC log options 170
9.99 BC wall.distance option 171
9.100 BC no.wall.distance option 171
9.101 BC Mach option 172
9.102 BC Reynolds option 172
9.103 BC velocity.magnitude option 173
9.104 BC alpha option 173
9.105 BC beta option 173
9.106 BC u option 174
9.107 BC v option 174
9.108 BC w option 174
9.109 BC T option 175
9.110 BC p option 175
9.111 BC trb.intensity option 176
9.112 BC trb.dnu option 176

9.113 System name option . . . 178
9.114 System type option . . . 178
9.115 System cell.gradient option . . . 179
9.116 Cell gradient types . . . 180
9.117 System face.gradient option . . . 180
9.118 Face gradient types . . . 181
9.119 Limiter option . . . 181
9.120 Limiter types . . . 182
9.121 System venka.k option . . . 182
9.122 System CFL option . . . 183
9.123 System implicit.jacobi option . . . 184
9.124 System convection.noslip.diagonal option . . . 184
9.125 System convection.impermeable.diagonal option . . . 185
9.126 System convection.symmetry.diagonal option . . . 185
9.127 System damp option . . . 186
9.128 System relaxation option . . . 186
9.129 System iteration.convergence option . . . 187
9.130 System user.source option . . . 187
9.131 System realizability.trb.w option . . . 188
9.132 System realizability.trb.w.Smag option . . . 188
9.133 System source.mpk option . . . 189
9.134 Mean flow system types for ideal gas . . . 190
9.135 Mean flow system types for multi component gas under chemical equilibrium . . . 191
9.136 Mean flow system types for multi component gas . . . 193
9.137 Turbulence models system types . . . 194
9.138 System log option . . . 195
9.139 System plot option . . . 195
9.140 General system log options . . . 196
9.141 Turbulence model system log options . . . 197
9.142 General system plot functions . . . 198



9.143 Mean flow system plot functions . . . 200
9.144 Turbulence model system plot functions . . . 201
9.145 Multi component gas plot functions . . . 201
9.146 Deformation system types . . . 202
9.147 Elastic mesh deformation from the reference mesh . . . 203
9.148 Elastic mesh-deformation stiffness distribution . . . 204
9.149 Elastic mesh-deformation maximum stiffness . . . 205
9.150 Elastic mesh-deformation stiffness power . . . 205
9.151 Elastic mesh-deformation wall-distance stiffness offset . . . 206
9.152 Solve convection.flux option . . . 209
9.153 Convection flux types . . . 209
9.154 Solve convection.jacobian option . . . 210
9.155 Convection Jacobian types . . . 210
9.156 Solve spatial.order option . . . 211
9.157 Solve sweeps option . . . 211
9.158 Solve axisymmetric option . . . 212
9.159 Solve time.march option . . . 212
9.160 Time marching methods . . . 213
9.161 Solve time.accurate.explicit option . . . 214
9.162 Solve iterations option . . . 214
9.163 Solve time.step.reduction option . . . 215
9.164 Solve time.step.reduction.value option . . . 215
9.165 Solve dual.time option . . . 216
9.166 Solve global.minimal.timestep option . . . 216
9.167 Solve save option . . . 217
9.168 Solve save.path option . . . 217
9.169 Solve save.sequential option . . . 217
9.170 Solve load.path option . . . 218
9.171 Solve load.sequence option . . . 218
9.172 Solve eos option . . . 219
9.173 Solve molecular option . . . 219



9.174 Solve p option . . . 219
9.175 Solve T option . . . 220
9.176 Solve Mach option . . . 220
9.177 Solve Reynolds option . . . 220
9.178 Solve velocity magnitude option . . . 221
9.179 Solve u option . . . 221
9.180 Solve v option . . . 221
9.181 Solve w option . . . 222
9.182 Solve alpha option . . . 222
9.183 Solve beta option . . . 222
9.184 Solve trb.intensity option . . . 223
9.185 Solve trb.mt option . . . 223
9.186 Solve Prandtl.trb option . . . 224
9.187 Solve reference.velocity option . . . 224
9.188 Solve reference.mach option . . . 224
9.189 Solve reference.pressure option . . . 225
9.190 Solve reference.velocity.trb option . . . 225
9.191 Solve reference.mach.trb option . . . 225
9.192 Solve reference.length option . . . 226
9.193 Solve ref option . . . 226
9.194 Solve plot option . . . 227
9.195 Solve plot options . . . 227
9.196 Solve table option . . . 227
9.197 Solve table options . . . 228
9.198 Solve log option . . . 228
9.199 Solve log options . . . 229
9.200 Solve source.mom option . . . 229
9.201 Solve hyb.diss.min option . . . 230
9.202 Solve hyb.diss.type option . . . 230
9.203 Hybrid Dissipation Factor Types . . . 231
9.204 Solve grid.length.scale option . . . 231

9.205 Grid Length Scale Types . . . 231
9.206 Solve gravitation.acceleration option . . . 232
9.207 Solve 2d.tolerance option . . . 232
9.208 Solve geometric.tolerance option . . . 233
9.209 Solve wall distance option . . . 235
9.210 Solve line search iterations option . . . 236
9.211 Solve line search minimum dq option . . . 236
9.212 Solve line search polynomial order option . . . 237
9.213 Solve line search beta option . . . 237
9.214 Solve line search flavor option . . . 238
9.215 Petsc ksp_view option . . . 239
9.216 Petsc ksp_monitor_true_residual option . . . 240
9.217 Petsc ksp_norm_type option . . . 240
9.218 Petsc ksp_max_it option . . . 241
9.219 Petsc ksp_rtol option . . . 241
9.220 Petsc ksp_atol option . . . 241
9.221 Petsc ksp_type option . . . 242
9.222 Petsc KSP Types . . . 242
9.223 Petsc ksp_gmres_restart option . . . 242
9.224 Petsc pc_type option . . . 243
9.225 Petsc sub_pc_type option . . . 243
9.226 Petsc PC Types . . . 244
9.227 Petsc sub_pc_factor_levels option . . . 244
9.228 Petsc sub_pc_factor_fill option . . . 245
9.229 Domain name option . . . 246
9.230 Domain init option . . . 247
9.231 Domain type option . . . 247
9.232 Domain box type option . . . 248
9.233 Domain cylinder type option . . . 249
9.234 Init name option . . . 250
9.235 Init T option . . . 251



9.236 Init p option . . . 251
9.237 Init u option . . . 251
9.238 Init v option . . . 252
9.239 Init w option . . . 252
9.240 Init composition option . . . 252
9.241 Init composition molar option . . . 253
9.242 Libraries name option . . . 254
9.243 Libraries prefix option . . . 254
9.244 User source name option . . . 255
9.245 User source lib option . . . 255
9.246 User source ctor option . . . 256
9.247 User source dtor option . . . 256
9.248 User source rhs option . . . 256
9.249 User source lhs option . . . 257
9.250 Var directive and its options . . . 258
9.251 Vertex option . . . 260
9.252 UNS name option . . . 261
9.253 UNS prefix option . . . 262
9.254 UNS block option . . . 262
9.255 UNS base option . . . 263
9.256 UNS zone option . . . 263
9.257 UNS crc option . . . 263
9.258 UNS key option . . . 264
9.259 UNS geometric.tolerance option . . . 264
9.260 UNS offset option . . . 265
9.261 UNS scale option . . . 265
9.262 UNS rotate option [rad] . . . 266
9.263 UNS rotate option [deg] . . . 266
9.264 UNS vertex option . . . 267
9.265 UNS log option . . . 267
9.266 UNS log options . . . 270



9.267 UNS DOF initial position option 272
9.268 UNS DOF initial orientation option 272
9.269 UNS DOF from/until option 273
9.270 DOF instrcutions 276
9.271 DOF log options 278
9.272 PCD name option 279
9.273 PCD prefix option 280
9.274 PCD mach option 280
9.275 PCD pressure option 280
9.276 PCD density option 281
9.277 PCD velocity.magnitude option 281
9.278 PCD power option 282
9.279 PCD K option 282
9.280 PCD percentage option 282
9.281 PCD spacing option 283
9.282 PCD decay option 283
9.283 Log name option 284
9.284 Log prefix option 285
9.285 Log per option 285
9.286 Log stdout option 286
9.287 Log stderr option 286
9.288 Plot name option 287
9.289 Plot prefix option 288
9.290 Plot interval option 288
9.291 Plot sequential/overwrite option 289
9.292 Plot sids.names option 289
9.293 Plot fvuns/cgns option 290
9.294 Plot bc option 290
9.295 Table name option 291
9.296 Table prefix option 292
9.297 Table interval option 292



9.298 Table sequential/overwrite option 292
9.299 Table horizontal/vertical option 293
9.300 Table loc face option 293
9.301 Table point option 293
9.302 Table ray option 294
9.303 Table plane option 294
9.304 Table bc option 295
9.305 Table plane.bc option 295
9.306 Post directive 296
9.307 Converge name option 297
9.308 Converge converge option 298
9.309 Convergence criteria types 299
9.310 Run-time options 301
9.311 Run-time options examples 302
9.312 User source input options 305
9.313 Library input options 305
9.314 System connection option 306
9.315 Available user-source variables 310
9.316 Available user-source equations 311



Listings

9.1 Example of a simple **Arion** input file (single-component gas) 98
9.2 Example of a simple **Arion** input file (multi-component gas) 100
9.3 Example of the def macro usage 106
9.4 Example of parallel section 111
9.5 Example of equation.of.state section 113
9.6 Example of the molecular section 117
9.7 Example of the tabulate section 124
9.8 Example of the multi-component gas section 128
9.9 Example of aeroelasticity options section 140
9.10 Example of the aeroelastic component section 152
9.11 Example of the bc section 162
9.12 Example of the system section 177
9.13 Example of aeroelastic calculations system section 202
9.14 Example of aeroelastic solver PETSc options 207
9.15 Example of the solve section 208
9.16 Example of the petsc section 239
9.17 Example of the domain section 246
9.18 Example of the init section 250
9.19 Example of the uns section 261
9.20 Example of a pcd section 279
9.21 Example of the log section 284
9.22 Example of the plot section 287
9.23 Example of the table section 291
9.24 Example of a simple **Arion** user source section 303



Abstract

This manual describes the algorithms, methods, and input and output files of the **Arion** code. In addition, the manual serves as the user's manual of the code. The current version of the code has gone through a major revision. As of the current version, the code provides the capability to simulate inviscid, laminar, or turbulent steady flows of single-component or multi-component gas. Currently, the flow solver contains the HLLC approximate Riemann solver, the AUSM$^+$-up scheme, or the AUSM-DV schemes for the approximation of the convective fluxes. Turbulence may be modeled using either the k$-\omega$-TNT, the k$-\omega$-SST, or the Spalart-Allmaras turbulence models. The code provides the capability to simulate the flow of any perfect gas under a chemical equilibrium assumption. Near future versions of the code will support simulations under a chemical non-equilibrium assumption. The code is fully parallel using a distributed architecture. Domain decomposition is carried out using the Metis open source library. Convergence acceleration is obtained using a Krylov solver and a line search algorithm.

Chapter 1

Introduction

The **Arion** code is the Israeli CFD Center flow solver for unstructured grids. Along recent years, computational capabilities have been ported from the EZAir Suite of codes into the **Arion** code. This process shall continue in coming months and years at an accelerated pace. In addition, new capabilities are being introduced into the **Arion** code. The goal is to obtain a versatile flow solver that would possess the entire capabilities that the EZAir Suite of codes possesses. This manual describes the algorithms, methods, and input/output files of the **Arion** code. In addition, the manual serves as the user's manual of the code.

1.1 Current Release

The current version of the code has gone through a major revision. As of the current version, the code provides the capability to simulate inviscid, laminar, or turbulent steady flows of single-component or multi-component gas. Currently, the flow solver contains the HLLC approximate Riemann solver, the AUSM$^+$-up scheme, or the AUSM-DV schemes for the approximation of the convective fluxes. Turbulence may be modeled using either the k$-\omega$-TNT, the k$-\omega$-SST, or the Spalart-Allmaras turbulence models. The code provides the capability to simulate the flow of any perfect gas under a chemical equilibrium assumption. Near future versions of the code will support simulations under a chemical non-equilibrium assumption. The code is fully

parallel using a distributed architecture. Domain decomposition is carried out using the Metis open source library. Convergence acceleration is obtained using a Krylov solver and a line search algorithm.

1.2 Users's Manual Arrangement

The report is arranged in the following manner: Chapter 2 contains a description of the available physical models while Chapter 3 describes the turbulence models that are used in the code. Chapter 4 is dedicated to a detailed description of the computational methods. Chapter 5 briefly describes the boundary conditions while Chapter 6 describes the type and format of unstructured meshes that the code supports. Chapter 8 describes the parallelization of the code. Finally, Chapter 9 contains a detailed description of the input file syntax, and actually serves as the code's reference manual.

Chapter 2

Physical Models

2.1 Introduction

Computer simulations are generally based upon the numerical solution of the model equations in a discretized mode. The accuracy of the computations depends mainly on the physical modeling, the numerical algorithms, and the quality of the computational mesh. At its current developmental stage, the **Arion** code provides the capability to simulate single-component perfect gas flows, or multi-component perfect gas flows under chemical equilibrium. This chapter contains a description of the physical models that are available in the solver.

2.2 Single-Component Perfect Gas (SPG)

2.2.1 Governing Equations

The equations governing single-component, perfect gas fluid flow are derived from the laws of conservation of mass, momentum, and total energy. The set of five partial differential equations is known as the Navier-Stokes equations and can be represented in a conservation-law form that is convenient for numerical simulations, namely

$$\frac{\partial Q}{\partial t} + \frac{\partial \left(E_c - E_d\right)}{\partial x} + \frac{\partial \left(F_c - F_d\right)}{\partial y} + \frac{\partial \left(G_c - G_d\right)}{\partial z} = 0 \tag{2.1}$$

where Q is the vector of conserved mass, momentum, and energy

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E \end{bmatrix} \tag{2.2}$$

where the density is denoted by ρ, the Cartesian velocity vector components are denoted by u,v and w, and E denotes the total (internal and kinetic) energy of the gas. The inviscid flux vectors, E_c, F_c, and G_c, are

$$E_c = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ u\left(E + p\right) \end{bmatrix}, \quad F_c = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ v\left(E + p\right) \end{bmatrix}, \quad G_c = \begin{bmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ w\left(E + p\right) \end{bmatrix} \tag{2.3}$$

and the viscous flux vectors, E_d, F_d, and G_d, are

$$E_d = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{yx} \\ \tau_{zx} \\ \beta_x \end{bmatrix}, \quad F_d = \begin{bmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ \tau_{zy} \\ \beta_y \end{bmatrix}, \quad G_d = \begin{bmatrix} 0 \\ \tau_{xz} \\ \tau_{yz} \\ \tau_{zz} \\ \beta_z \end{bmatrix} \tag{2.4}$$

where

$$\begin{aligned}
\tau_{xx} &= \lambda\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}\right)+2\mu\frac{\partial u}{\partial x}\\
\tau_{yy} &= \lambda\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}\right)+2\mu\frac{\partial v}{\partial y}\\
\tau_{zz} &= \lambda\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}\right)+2\mu\frac{\partial w}{\partial z}\\
\tau_{xy} &= \tau_{yx} = \mu\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\\
\tau_{xz} &= \tau_{zx} = \mu\left(\frac{\partial u}{\partial z}+\frac{\partial w}{\partial x}\right)\\
\tau_{yz} &= \tau_{zy} = \mu\left(\frac{\partial v}{\partial z}+\frac{\partial w}{\partial y}\right)\\
\beta_x &= u\tau_{xx}+v\tau_{xy}+w\tau_{xz}+\kappa\frac{\partial T}{\partial x}\\
\beta_y &= u\tau_{yx}+v\tau_{yy}+w\tau_{yz}+\kappa\frac{\partial T}{\partial y}\\
\beta_z &= u\tau_{zx}+v\tau_{zy}+w\tau_{zz}+\kappa\frac{\partial T}{\partial z}
\end{aligned} \tag{2.5}$$

where T is the temperature. Stokes hypothesis, $\lambda = -\frac{2}{3}\mu$, is typically used to further simplify Equation (2.5).

Finally, in the perfect-gas model, the source-term vector, S, may only contain contributions from the turbulence model (see Chapter 3).

2.2.2 Integral Form For Moving Grids

For a three-dimensional flow through a finite volume Ω, enclosed by the boundary surface $\partial\Omega \equiv S$ that is moving with a grid velocity $\overline{V}_g$, the integral form of the conservation equations in an inertial frame of reference is given as:

$$\frac{\partial}{\partial t}\int_{\Omega} Q d\Omega + \oint_{S} d\overline{S}\cdot\overline{H} = 0 \tag{2.6}$$

where

$$\overline{H} = (E_c - E_d)\,\hat{i} + (F_c - F_d)\,\hat{j} + (G_c - G_d)\,\hat{k} \tag{2.7}$$

In Equation (2.6) the vector of dependent variables, Q, remains as in Equation (2.2).Similarly, the viscous flux vectors, E_d, F_d, and G_d, appearing in Equation (2.7), remain as in Equations (2.4) and (2.5) (see Section 2.2.1). In contrast, the inviscid flux vectors

take a form that reflects the motion of the grid as follows:

$$
\begin{aligned}
E_c &= \begin{bmatrix} \rho\left(u-u_g\right) \\ \rho u\left(u-u_g\right)+p \\ \rho v\left(u-u_g\right) \\ \rho w\left(u-u_g\right) \\ \left(E+p\right)\left(u-u_g\right)+u_g p \end{bmatrix} \\
F_c &= \begin{bmatrix} \rho\left(v-v_g\right) \\ \rho u\left(v-v_g\right) \\ \rho v\left(v-v_g\right)+p \\ \rho w\left(v-v_g\right) \\ \left(E+p\right)\left(v-v_g\right)+v_g p \end{bmatrix} \\
G_c &= \begin{bmatrix} \rho\left(w-w_g\right) \\ \rho u\left(w-w_g\right) \\ \rho v\left(w-w_g\right) \\ \rho w\left(w-w_g\right)+p \\ \left(E+p\right)\left(w-w_g\right)+w_g p \end{bmatrix}
\end{aligned}
\tag{2.8}
$$

where u_g, v_g, and w_g are the Cartesian components of the grid velocity vector $\overline{V}_g$. Note that currently the grid velocity vector is set to $\overline{V}_g \equiv 0$.

2.2.3 Equation of State

To close the system of fluid dynamics equations, it is necessary to establish relations between the thermodynamics variables, p, ρ, T, and the internal energy, e_I. Assuming a perfect gas, the pressure and temperature may be obtained from the following equation of state:

$$p = \rho R T \tag{2.9}$$

where R is the gas constant ($R = 287.0$ for air). By assuming further that the gas is a calorically perfect gas (and hence the specific heats C_p and C_v are constant), the

equation of state takes the form:

$$p = \rho\,(\gamma - 1)\,e_I \tag{2.10}$$

where e_I is the internal energy of the gas, and γ is the (constant) ratio of specific heats (c_p/c_v). In terms of the flow variables, the pressure and temperature are calculated using:

$$\begin{aligned} p &= (\gamma - 1)\left[E - \frac{1}{2}\rho\left(u^2 + v^2 + w^2\right)\right] \\ T &= \frac{\gamma - 1}{R}\left[e - \frac{1}{2}\left(u^2 + v^2 + w^2\right)\right] \end{aligned} \tag{2.11}$$

where $e = \frac{E}{\rho}$ is the specific total energy.

2.2.4 Transport Properties

In addition to the equation of state, it is also necessary to establish constitutive relations, namely, relations for the coefficient of viscosity, μ, and the coefficient of thermal conductivity, κ. In the single-component perfect gas (SPG) model, the Sutherland formulae are exclusively used to evaluate these coefficients. The **Arion** code provides two different ways to evaluate μ and κ.

The first (called "molecular.sutherland.air.1", see Table 9.14) is based on the following relations:

$$\begin{aligned} \mu &= C_{\mu_1}\frac{T^{\frac{3}{2}}}{T + C_{\mu_2}} \\ \kappa &= C_{\kappa_1}\frac{T^{\frac{3}{2}}}{T + C_{\kappa_2}} \end{aligned} \tag{2.12}$$

The default of the coefficients C_{μ_1}, C_{μ_2}, C_{κ_1}, and C_{κ_2} correspond to air and are given in Table 2.1. For the purpose of simulating a gas whose transport properties are different than air, the coefficients can be set by the user using the directives as described in Tables 9.17, 9.18, 9.19, 9.20.

Fluid type	C_{μ_1}	C_{μ_2}	C_{κ_1}	C_{κ_2}
Air	1.458×10^{-6}	110.4	2.495×10^{-3}	194

Table 2.1: Default coefficients for Sutherland formulae (type 1)

The second way to evaluate μ and κ (called "molecular.sutherland.air.2", see Table 9.14) is based on the following relations:

$$\begin{aligned} \mu &= \eta_0 \frac{T_0 + C_0}{T + C_0} \left(\frac{T}{T_0} \right)^{\frac{3}{2}} \\ \kappa &= \frac{c_p \mu}{Pr} \end{aligned} \tag{2.13}$$

where P_r is the Prandtl number. The default of the coefficients η_0, T_0, and C_0 correspond to air and are given in Table 2.2. The coefficients can be set by the user using the directives as described in Tables 9.21, 9.22, 9.23, and 9.24.

Fluid type	η_0	C_0	T_0	P_r
Air	1.827×10^{-5}	120	291.15	0.72

Table 2.2: Default coefficients for Sutherland formulae (type 2)

2.3 Multi-component Gas (MCG)

High-speed flows contain physical phenomena that generally may not be modeled by the perfect gas form of the Navier-Stokes equations presented in Section 2.2. Such phenomena include chemical and thermal non-equilibrium, vibrational and electronic excitation, and ionization. These phenomena require the use of a "real gas" model. This section presents the extended form of the Navier-Stokes equations governing the flow of a general gas in chemical and thermal non-equilibrium, and the thermodynamic and transport models that are used to describe the gas. Focus is given to the numerical algorithms that are used in the solution of the extended Navier-Stokes equations and

the finite-rate chemical kinetics and energy relaxation models that are employed to model non-equilibrium flow.

2.3.1 Basic Assumptions

The flow is modeled assuming that the continuum approximation is valid. Thermal equilibrium (*i.e.*, all energy modes are described by a single temperature) is assumed for general gases. For simulations involving air, the *two-temperature model* of Park [1] is employed to account for vibrational and electronic non-equilibrium. According to the two-temperature model, the translational and rotational energy modes are described by the translational temperature, T, while the vibrational and electronic energy modes of all species and the electron translational energy mode are described by a second temperature, denoted by T_v. The latter assumption considerably simplifies the system of equations by eliminating additional translational and vibrational energy equations for each polyatomic species and an energy equation for the free electrons. This model was shown to provide accurate results for aerodynamic coefficients and convective heat transfer rates in external flows of air at Mach numbers exceeding 20 [2]. It is therefore assumed adequate for the problems considered for simulation with the present code.

2.3.2 Governing Equations

Under the above assumptions, the extended three-dimensional Navier-Stokes equation set may be expressed in Cartesian coordinates as follows [3]:

$$\frac{\partial Q}{\partial t} + \frac{\partial \left(E_c - E_d\right)}{\partial x} + \frac{\partial \left(F_c - F_d\right)}{\partial y} + \frac{\partial \left(G_c - G_d\right)}{\partial z} = S \tag{2.14}$$

where Q is the vector of conserved variables,

$$Q = \begin{bmatrix} \rho u \\ \rho v \\ \rho w \\ E \\ \rho_1 \\ \vdots \\ \rho_N \\ E_v \end{bmatrix} \tag{2.15}$$

where u,v and w, denote the the Cartesian velocity vector components, E denotes the total (internal and kinetic) energy of the mixture, and E_v denotes the vibrational-electronic energy of the mixture. The mixture density, ρ, is given by a simple sum of the individual chemical species densities, ρ_s, namely,

$$\rho = \sum_{s=1}^{N} \rho_s = \sum_{s=1}^{N} \rho Y_s \tag{2.16}$$

where $Y_{s=1,\ldots,N}$ denote the individual chemical species mass fractions. Note that in the case of thermal equilibrium, the mixture vibrational-electronic energy is identically zero ($E_v = 0$) and the last equation in the set is omitted.

The convective (inviscid) flux vectors, E_c, F_c, and G_c, are

$$E_c = \begin{bmatrix} \rho u^2 + p \\ \rho uv \\ \rho uw \\ u\left(E+p\right) \\ \rho_1 u \\ \vdots \\ \rho_N u \\ uE_v \end{bmatrix}, \quad F_c = \begin{bmatrix} \rho uv \\ \rho v^2 + p \\ \rho vw \\ v\left(E+p\right) \\ \rho_1 v \\ \vdots \\ \rho_N v \\ vE_v \end{bmatrix}, \quad G_c = \begin{bmatrix} \rho uw \\ \rho vw \\ \rho w^2 + p \\ w\left(E+p\right) \\ \rho_1 w \\ \vdots \\ \rho_N w \\ wE_v \end{bmatrix} \tag{2.17}$$

and the diffusive (viscous) flux vectors, E_d, F_d, and G_d, are

$$E_d = \begin{bmatrix} \tau_{xx} \\ \tau_{yx} \\ \tau_{zx} \\ \beta_x \\ -J_{x,1} \\ \vdots \\ -J_{x,N} \\ \beta_{v,x} \end{bmatrix}, \quad F_d = \begin{bmatrix} \tau_{xy} \\ \tau_{yy} \\ \tau_{zy} \\ \beta_y \\ -J_{y,1} \\ \vdots \\ -J_{y,N} \\ \beta_{v,y} \end{bmatrix}, \quad G_d = \begin{bmatrix} \tau_{xz} \\ \tau_{yz} \\ \tau_{zz} \\ \beta_z \\ -J_{z,1} \\ \vdots \\ -J_{z,N} \\ \beta_{v,z} \end{bmatrix} \tag{2.18}$$

where

$$\begin{aligned}
\tau_{xx} &= \lambda\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) + 2\mu\frac{\partial u}{\partial x} \\
\tau_{yy} &= \lambda\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) + 2\mu\frac{\partial v}{\partial y} \\
\tau_{zz} &= \lambda\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) + 2\mu\frac{\partial w}{\partial z} \\
\tau_{xy} &= \tau_{yx} = \mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \\
\tau_{xz} &= \tau_{zx} = \mu\left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}\right) \\
\tau_{yz} &= \tau_{zy} = \mu\left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right) \\
\beta_x &= u\tau_{xx} + v\tau_{xy} + w\tau_{xz} - (q_x + q_{v,x}) - \sum_s (J_{x,s}h_s) \\
\beta_y &= u\tau_{yx} + v\tau_{yy} + w\tau_{yz} - (q_y + q_{v,y}) - \sum_s (J_{y,s}h_s) \\
\beta_z &= u\tau_{zx} + v\tau_{zy} + w\tau_{zz} - (q_z + q_{v,z}) - \sum_s (J_{z,s}h_s) \\
\beta_{v,x} &= -q_{v,x} - \sum_s (J_{x,s}e_{v,s}) \\
\beta_{v,y} &= -q_{v,y} - \sum_s (J_{y,s}e_{v,s}) \\
\beta_{v,z} &= -q_{v,z} - \sum_s (J_{z,s}e_{v,s})
\end{aligned} \tag{2.19}$$

the pressure is denoted by p, and h_s and $e_{v,s}$ are the specific enthalpy and vibrational energy of species s, respectively. The translational-rotational heat flux is given by $q = -\kappa\nabla T$ while $q_v = -\kappa_v\nabla T_v$ is the vibrational-electronic heat flux, where T, T_v are the translational and vibrational temperatures, respectively. The terms κ, κ_v are the

thermal conductivities associated with the various energy modes (*i.e.*, translational and vibrational-electronic).

The species mass diffusion fluxes, $J_{x,s}$, $J_{y,s}$, $J_{z,s}$ are modeled using Fick's law,

$$J_s = -\rho D_s \nabla Y_s \tag{2.20}$$

where D_s is the diffusion coefficient of species s with respect to the mixture (see Section 2.3.5). Stokes hypothesis, $\lambda = -\frac{2}{3}\mu$, is employed to further simplify Equation (2.19).

2.3.3 Integral Form For Moving Grids

For a three-dimensional flow through a finite volume Ω, enclosed by the boundary surface $\partial\Omega \equiv S$ that is moving with a grid velocity $\overline{V}_g$, the integral form of the conservation equations in an inertial frame of reference is given as:

$$\frac{\partial}{\partial t}\int_{\Omega} Q d\Omega + \oint_{S} d\overline{S} \cdot \overline{H} = 0 \tag{2.21}$$

where

$$\overline{H} = (E_c - E_d)\,\hat{i} + (F_c - F_d)\,\hat{j} + (G_c - G_d)\,\hat{k} \tag{2.22}$$

In Equation (2.21) the vector of dependent variables, Q, remains as in Equation (2.15). Similarly, the viscous flux vectors, E_d, F_d, and G_d, appearing in Equation (2.22), remain as in Equations (2.18) and (2.19) (see Section 2.3.2). In contrast, the inviscid

flux vectors take a form that reflects the motion of the grid as follows:

$$
\begin{aligned}
E_c &= \begin{bmatrix} \rho u\left(u-u_g\right)+p \\ \rho v\left(u-u_g\right) \\ \rho w\left(u-u_g\right) \\ \left(E+p\right)\left(u-u_g\right)+u_g p \\ \rho_1\left(u-u_g\right) \\ \vdots \\ \rho_N\left(u-u_g\right) \\ \left(u-u_g\right)E_v \end{bmatrix} \\
F_c &= \begin{bmatrix} \rho u\left(v-v_g\right) \\ \rho v\left(v-v_g\right)+p \\ \rho w\left(v-v_g\right) \\ \left(E+p\right)\left(v-v_g\right)+v_g p \\ \rho_1\left(v-v_g\right) \\ \vdots \\ \rho_N\left(v-v_g\right) \\ \left(v-v_g\right)E_v \end{bmatrix} \\
G_c &= \begin{bmatrix} \rho u\left(w-w_g\right) \\ \rho v\left(w-w_g\right) \\ \rho w\left(w-w_g\right)+p \\ \left(E+p\right)\left(w-w_g\right)+w_g p \\ \rho_1\left(w-w_g\right) \\ \vdots \\ \rho_N\left(w-w_g\right) \\ \left(w-w_g\right)E_v \end{bmatrix}
\end{aligned}
\tag{2.23}
$$

where u_g, v_g, and w_g are the Cartesian components of the grid velocity vector $\overline{V}_g$.

2.3.4 Thermodynamic Properties

The pressure, p, is obtained from Dalton's law of partial pressures for a mixture of thermally perfect gases,

$$p = \rho \sum_s R_s Y_s T + \rho R_e Y_e T_v \tag{2.24}$$

where $R_s = \frac{R_0}{W_s}$ is the specific gas constant of species s, with R_0 denoting the universal gas constant and W_s denoting the molecular weight of that species. The subscript e denotes the free electron species.

2.3.4.1 Thermal Equilibrium

If the gas is in thermal equilibrium (*i.e.*, $T = T_v$), then nine-term polynomials of temperature are used for calculating the specific enthalpy, h_s, and the heat capacity at constant pressure, $C_{p,s}$, of individual gaseous species [4]. The total energy is then given by:

$$E = \sum_s \rho Y_s h_s - p + \frac{1}{2}\rho\left(u^2 + v^2 + w^2\right) \tag{2.25}$$

2.3.5 Transport Properties

The transport properties of the mixture may be calculated with three different models. The first, most simple model, employs the Sutherland formulae (see Section 2.2.4) to evaluate the transport properties of standard air at low temperatures (up to 2,000 K). Therefore, it is recommended to use this model only for equilibrium air flows at relatively low speeds. A more elaborate model that accounts for the various components of the gas is based on Wilke's mixing rule [5] together with Blottner's [6] or Mcbride's [7] curve fits for approximating the viscosity, and Eucken's modified formula [8] for calculating the thermal conductivities of individual species. This model is suitable for air at temperatures of up to 10,000 K, and may not be used in simulation of ionized flows. For air at higher temperatures, arising typically in re-entry simulations, a second, more complex model is available owing to Gupta [9].

2.3.5.1 Wilke's Model

In this model, mixture transport properties are obtained using Wilke's mixing rule [5]:

$$\mu = \sum_s \frac{\mu_s X_s}{\phi_s} \quad , \quad \kappa = \sum_s \frac{\kappa_s X_s}{\phi_s} \tag{2.26}$$

where

$$\phi_s = \sum_r X_r \left[1 + \sqrt{\frac{\mu_s}{\mu_r}} \left(\frac{M_r}{M_s}\right)^{\frac{1}{4}}\right]^2 \left[\sqrt{8\left(1 + \frac{M_s}{M_r}\right)}\right]^{-1} , \tag{2.27}$$

the term X_s denotes the molar fraction of species s in the mixture, μ_s is the species coefficient of viscosity and κ_s is the species thermal conductivity for each energy mode (*i.e.*, translational and vibrational).

The species viscosities are calculated using Blottner's [6] curve-fits of temperature for components of air, or McBride [7] curve-fits for general gaseous species. The species thermal conductivities, κ_s^{tr} and κ_s^{vib}, are determined from a generalized Eucken's relation, according to Hirschfelder [8]:

$$\kappa_s^{tr} = \mu_s \left(\frac{5}{2} C_{v,s}^t + \frac{C_{v,s}^r}{Sc}\right) \tag{2.28}$$

$$\kappa_s^{vib} = \mu_s \left(C_{v,s}^{vib}\right) \tag{2.29}$$

The terms $C_{v,s}^t, C_{v,s}^r, C_{v,s}^{vib}$ denote the translational, rotational and vibrational specific heats at constant volume of species s, respectively, and Sc is a constant Schmidt number, assumed equal for all species.

Finally, the mass diffusion constant of species s with respect to the mixture, D_s, is replaced by a single binary coefficient, D, that is evaluated from a constant, user-supplied Schmidt number, as follows:

$$D = \frac{\mu}{\rho Sc} \tag{2.30}$$

2.3.6 Chemical Reaction Modeling

The local chemical composition of the mixture may be determined in three different methods, ordered here by increasing computational complexity:

1. *Frozen gas*: Under this assumption, the chemical composition remains fixed throughout the flow-field, and is set to the composition of the free-stream. This mode is suitable only for air at low temperatures (up to 2000 K), and for general gases that are known to be chemically non-reacting at the simulated conditions.

2. *Equilibrium gas*: Under this assumption, the local chemical composition is determined uniquely as a function of two thermodynamic variables (e.g., temperature and pressure) via minimization of Gibbs or Helmholtz free energy [10]. For enhanced efficiency, the equilibrium properties of the mixture are tabulated upon start-up, so that the cumbersome process of non-linear minimization is avoided throughout the normal operation of the program. The chemical equilibrium mode is applicable to flows at moderate speeds, where chemical reactions occur at a much faster rate than the traversal rate of the molecules through the domain of interest. The **Arion** code utilizes tabulation for the thermodynamic and molecular properties of the gas (see Section 9.8 for tabulation directives and options). Note, it is required to utilize the eos.table type for the equilibrium gas assumption (see Tables 9.8 and 9.9). In addition, it is required to addd the −−multi.component.gas directive and options (see Section 9.9).

3. *Non-Equilibrium flow*: Under this assumption, finite-rate chemical kinetics models (detailed ahead) are employed to calculate the local composition of the mixture from the solution of additional mass conservation equations for the individual components. This is the most accurate (and expensive) method for calculating the local composition of the gas. It should be used in high-speed flows of air, and in other cases where finite-rate chemistry effects are of interest (*e.g.*, combustion).

2.3.6.1 Equilibrium Gas Properties Tabulation

The tabulation of the equilibrium gas properties is conducted as described in Section 9.8, Tables 9.25-9.48. Note that viscosity, conductivity and heat capacity are also tabulated, based on the selected thermo/transport models.

2.3.6.2 Finite-rate Chemical Kinetics

A reaction mechanism composed of N_r reversible reactions may be generally described by the following chemical formulae:

$$\sum_{s=1}^{N} \nu'_{s,q} M_s \underset{c_{r,q}}{\overset{c_{f,q}}{\rightleftharpoons}} \sum_{s=1}^{N} \nu''_{s,q} M_s \quad \forall \quad q = 1 \ldots N_r \tag{2.31}$$

where $\nu'_{s,q}$ and $\nu''_{s,q}$ are the stoichiometric coefficients of species s (represented by the chemical symbol M_s), acting as a reactant or product in reaction q, respectively. The code currently assumes laminar chemistry, according to which the Arrhenius approach is used to describe the mean reaction rates (*i.e.*, based only on mean values of temperature and species mass fractions). The formula for the mean reaction rate for species s, S_s, appearing as a source-term in the respective species mass conservation equation, is then given by:

$$S_s = W_s \sum_{q=1}^{N_r} \left(\nu''_{s,q} - \nu'_{s,q}\right) \left[c_{f,q} \prod_{p=1}^{N} \left(\frac{\rho Y_p}{W_p}\right)^{\nu'_{p,q}} - c_{r,q} \prod_{p=1}^{N} \left(\frac{\rho Y_p}{W_p}\right)^{\nu''_{p,q}} \right] \tag{2.32}$$

where $c_{f,q}$ and $c_{r,q}$ are the forward and reverse rate coefficients of reaction q, respectively.

2.3.6.3 Reaction Rate Coefficients

The forward rate coefficients are expressed in Arrhenius form as:

$$c_q(T_c) = A_q T_c^{\gamma_q} e^{-(E_q/R_0 T_c)} \tag{2.33}$$

where γ_q and A_q are constants, E_q is the activation energy, and T_c is a controlling temperature to be defined later. Note that the default units of the Arrhenius coefficient and the activation energy are $[A] = (m^3/kmol)^{n-1}/sec$ and $[E] = (J/kmol/K)$, respectively. Other units may be specified via a "UNITS" directive at the beginning of the reaction rates file. The reverse rate coefficients may either be given explicitly in Arrhenius form, for instance, as in Dunn and Kang's 5-species, 11-reactions model [11], or they may be calculated via the equilibrium constant of the reaction, as follows:

$$c_{r,q}(T_{bc}) = \frac{c_{f,q}(T_{bc})}{K_{eq,q}(T_{bc})} \tag{2.34}$$

Where $K_{eq,q}$ denotes the equilibrium constant of reaction q, and T_{bc} is the reverse rate controlling temperature that is not necessarily equal to the forward controlling temperature, T_c. The equilibrium constant, K_{eq}, itself may be calculated in two methods: First, for air only, the constants may be approximated with Park's curve-fits [12] of the local number density. For a general gas, the equilibrium constant of reaction q may only be calculated from Gibbs' free energy,

$$K_{eq}(T) = \left(\frac{10^5}{R_0 T}\right)^{\nu_q} \exp\left[-\sum_s \left(\nu''_{s,q} - \nu'_{s,q}\right)\left(\frac{h_s}{R_s T} - \frac{s_s}{R_s}\right)\right] \tag{2.35}$$

where $\nu_q = \sum_s \left(\nu''_{s,q} - \nu'_{s,q}\right)$, and the dimensionless enthalpy and entropy are obtained from curve-fits, as described in section 2.3.4.

2.4 Gas Selection

2.4.1 Single Component Perfect Gas

As mentioned above, the **Arion** code provides the means to simulate the flow of any perfect gas. This is facilitated via a series of directives that allow to set the specific gas constant, (R, see Table 9.10), the the heat capacity ratio, (γ, see Table 9.11, and the Sutherland formulae coefficients as described in Section 2.2.4.

2.4.2 Multi Component Gas

In the case of a multi-component gas, the actual composition of the gas has to be defined using the appropriate directives. The reader is referred to Section 9.9 and the tables therein.



Chapter 3

Turbulence Models

The unsteady Navier-Stokes equations are generally considered to govern turbulent flows in the continuum flow regime. However, turbulent flow cannot be numerically simulated as easily as laminar flow. To resolve a turbulent flow by direct numerical simulation (DNS) requires that all relevant length scales be properly resolved. Such requirements place great demands on the computer resources, a fact that renders the possibility of conducting DNS analysis about complete aircraft configurations infeasible.

A practical approach to simulating turbulent flows is to solve the time-averaged Navier-Stokes equations. These equations are know as the "Reynolds averaged Navier-Stokes" (RANS) equations. The averaging of the equations of motion gives rise to new terms that are called the Reynolds stresses. To solve the averaged equations, the Reynolds stress tensor must be related to the flow variables through turbulence models. The models are used to "close" the system through an additional set of assumptions. The models are classified based on the number of additional partial differential equations that must be solved. The **Arion** code currently contains only one turbulence model, a two-equation model.

3.1 RANS Turbulence Model Equations

The **Arion** code treats the mean flow and turbulence models equations in a unified manner. To this end, the Navier-Stokes equation set is extended to include the turbulence model equations. Consequently, the discretization of the various fluxes can be conducted in the same manner.

The equations governing turbulent flows are obtained by Favre-averaging the Navier-Stokes equations and by modeling the Reynolds stress tensor. The unknown averaged Reynolds stress tensor is modeled either using the Boussinesq assumption via a linear eddy-viscosity model or by directly solving a transport equation for each of the Reynolds stress components via a second moment closure. The general form of the resulting Navier-Stokes equations and the turbulence model equations has the form (for simplicity of the representation, with no loss of generality, the formulation under the perfect gas physical model assumption is brought herein; it can be easily extended to any physical model):

$$\frac{\partial Q}{\partial t} + \frac{\partial \left(E_c - E_d\right)}{\partial x} + \frac{\partial \left(F_c - F_d\right)}{\partial y} + \frac{\partial \left(G_c - G_d\right)}{\partial z} = S \tag{3.1}$$

where S is the source term associated with the turbulence model only (once again, under the perfect gas physical model assumption). Hence, for turbulent flow simulations, Equation (3.1) replaces Equation (2.1).

In what follows, the symbol $(\bar{\ })$ indicates non-weighted averaging, the symbol $(\tilde{\ })$ signifies mass weighted Favre averaging, and the symbol $(\ '')$ denotes Favre fluctuations. Depending on whether the turbulence model has one, two, or m equations, the vector of dependent variables, Q, and the vector of source terms S now take the

form:

$$Q = \begin{bmatrix} \bar{\rho} \\ \bar{\rho}\tilde{u} \\ \bar{\rho}\tilde{v} \\ \bar{\rho}\tilde{w} \\ \tilde{E} \\ \bar{\rho}q_1 \\ \dots \\ \bar{\rho}q_m \end{bmatrix}, \qquad S = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ s1 \\ \dots \\ s_m \end{bmatrix} \tag{3.2}$$

where q is the vector of turbulence model dependent variables and s, is the source terms vector. Note that the source terms differ from model to model. The vectors E_c, F_c, and G_c usually take the form:

$$E_c = \begin{bmatrix} \bar{\rho}\tilde{u} \\ \bar{\rho}\tilde{u}^2 + \bar{p} \\ \bar{\rho}\tilde{u}\tilde{v} \\ \bar{\rho}\tilde{u}\tilde{w} \\ \tilde{u}\left(\tilde{E} + \bar{p}\right) \\ \bar{\rho}\tilde{u}q_1 \\ \dots \\ \bar{\rho}\tilde{u}q_m \end{bmatrix}, \quad F_c = \begin{bmatrix} \bar{\rho}\tilde{v} \\ \bar{\rho}\tilde{u}\tilde{v} \\ \bar{\rho}\tilde{v}^2 + \bar{p} \\ \bar{\rho}\tilde{v}\tilde{w} \\ \tilde{v}\left(\tilde{E} + \bar{p}\right) \\ \bar{\rho}\tilde{v}q_1 \\ \dots \\ \bar{\rho}\tilde{v}q_m \end{bmatrix}, \quad G_c = \begin{bmatrix} \bar{\rho}\tilde{w} \\ \bar{\rho}\tilde{u}\tilde{w} \\ \bar{\rho}\tilde{v}\tilde{w} \\ \bar{\rho}\tilde{w}^2 + \bar{p} \\ \tilde{w}\left(\tilde{E} + \bar{p}\right) \\ \bar{\rho}\tilde{w}q_1 \\ \dots \\ \bar{\rho}\tilde{w}q_m \end{bmatrix} \tag{3.3}$$

Similarly, the vectors, E_d, F_d, and G_d, usually take the form:

$$E_d = \begin{bmatrix} 0 \\ \bar{\tau}_{xx} - \overline{\rho u'' u''} \\ \bar{\tau}_{yx} - \overline{\rho v'' u''} \\ \bar{\tau}_{zx} - \overline{\rho w'' u''} \\ \bar{\beta}_x \\ e_{d_1} \\ \dots \\ e_{d_m} \end{bmatrix}, \quad F_d = \begin{bmatrix} 0 \\ \bar{\tau}_{xy} - \overline{\rho u'' v''} \\ \bar{\tau}_{yy} - \overline{\rho v'' v''} \\ \bar{\tau}_{zy} - \overline{\rho w'' v''} \\ \bar{\beta}_y \\ f_{d_1} \\ \dots \\ f_{d_m} \end{bmatrix}, \quad G_d = \begin{bmatrix} 0 \\ \bar{\tau}_{xz} - \overline{\rho u'' w''} \\ \bar{\tau}_{yz} - \overline{\rho v'' w''} \\ \bar{\tau}_{zz} - \overline{\rho w'' w''} \\ \bar{\beta}_z \\ g_{d_1} \\ \dots \\ g_{d_m} \end{bmatrix} \tag{3.4}$$

where the vectors e_d, f_d, and g_d differ from model to model and

$$\begin{aligned}
\bar{\beta}_x &= \tilde{u}\left(\bar{\tau}_{xx} - \overline{\rho u'' u''}\right) + \tilde{v}\left(\bar{\tau}_{xy} - \overline{\rho u'' v''}\right) + \tilde{w}\left(\bar{\tau}_{xz} - \overline{\rho u'' w''}\right) + \left(\bar{\kappa} + \bar{\kappa}_t\right)\frac{\partial \tilde{T}}{\partial x} \\
\bar{\beta}_y &= \tilde{u}\left(\bar{\tau}_{yx} - \overline{\rho v'' u''}\right) + \tilde{v}\left(\bar{\tau}_{yy} - \overline{\rho v'' v''}\right) + \tilde{w}\left(\bar{\tau}_{yz} - \overline{\rho v'' w''}\right) + \left(\bar{\kappa} + \bar{\kappa}_t\right)\frac{\partial \tilde{T}}{\partial y} \\
\bar{\beta}_z &= \tilde{u}\left(\bar{\tau}_{zx} - \overline{\rho w'' u''}\right) + \tilde{v}\left(\bar{\tau}_{zy} - \overline{\rho w'' v''}\right) + \tilde{w}\left(\bar{\tau}_{zz} - \overline{\rho w'' w''}\right) + \left(\bar{\kappa} + \bar{\kappa}_t\right)\frac{\partial \tilde{T}}{\partial z}
\end{aligned} \tag{3.5}$$

The terms $\bar{\kappa}$ and $\bar{\kappa}_t$ are the averaged molecular and turbulent heat conductivities, respectively. The molecular heat conductivity is calculated using Sutherland's law (see Equation (2.12)) while the turbulent heat conductivity is calculated using

$$\bar{\kappa}_t = \frac{c_p \bar{\mu}_t}{Pr_t} \tag{3.6}$$

where $\bar{\mu}_t$ denotes the turbulent viscosity. The term c_p is the specific heat capacity at constant pressure, Pr is the Prandtl number set to $Pr = 0.72$, and Pr_t is the turbulent Prandtl number set to $Pr_t = 0.9$. The average turbulent viscosity, $\bar{\mu}_t$ differs from model to model.

3.2 The Spalart Allmaras Turbulence Model

The one-equation SA model was originally published in 1992 [13]. The original model, also known as the *standard* SA model (according to the TMR web site) heavily relies on calibration by references to a wide range of experimental data, including flows subjected to adverse pressure gradient. The rich body of empirical information that the model contains gives it advantages over other models when applied to 'complex' boundary layers approaching separation, and this has made the model popular for aeronautical practice. However, the *standard* SA model suffers from certain anomalies. To circumvent these anomalies, Edwards and Chandra [14] proposed a modification to the model, resulting in a far more robust model. Although the work by Edwards and Chandra concerns the incompressible form of the model, a compressible formulation of their model is adopted herein (denoted the *SA-Edwards* model). In addition, the EZAir code offers a second variant of the SA model, based on the work by Allmaras *et al* [15] with a slight modification as proposed in the work by Rumsey *et al* [16].

The second variant is denoted as the *SA-2020* model. Both models were desgined to overcome the above mentioned anomalies. The Spalart-Allmaras model that is implemented in **Arion** is the *SA-2020* variant of the model.

A unified formulation of the two aforementioned variants may be written as follows:

$$\frac{\partial \rho\tilde{\nu}}{\partial t} + \frac{\partial \rho\tilde{\nu}u_k}{\partial x_k} = \frac{\partial}{\partial x_k}\left[\frac{1}{\sigma}\left(\mu + \rho\tilde{\nu}\right)\frac{\partial \tilde{\nu}}{\partial x_k}\right] + P_{SA} - D_{SA} + SD_{SA} + C_{SA} \tag{3.7}$$

The model is comprised of a transport equation for the undamped eddy viscosity, $\tilde{\nu}$, where t denotes the time and x_j=[x,y,z] denote the Cartesian coordinates. The fluid density is denoted by ρ while the Cartesian velocity vector components are denoted by u_j=[u,v,w].

The production term, P_{SA}, and the destruction term, D_{SA}, are given as by:

$$P_{SA} = c_{b_1}\tilde{S}\rho\tilde{\nu} \tag{3.8}$$

and

$$D_{SA} = c_{w_1} f_w \rho \left(\frac{\tilde{\nu}}{d_w}\right)^2 \tag{3.9}$$

where d_w is the distance from the nearest wall. The near-wall damping function f_w is defined as:

$$f_w = g\left(\frac{1 + c_{w3}^6}{g^6 + c_{w3}^6}\right)^{1/6} \tag{3.10}$$

where

$$g = r + c_{w2}\left(r^6 - r\right) \tag{3.11}$$

The laminar suppression term is given by

$$f_{t2} = c_{t3}\exp\left(-c_{t4}\chi^2\right) \tag{3.12}$$

The source diffusion term, SD_{SA} is given by:

$$SD_{SA} = \rho\frac{c_{b_2}}{\sigma}\frac{\partial \tilde{\nu}}{\partial x_k}\frac{\partial \tilde{\nu}}{\partial x_k} \tag{3.13}$$

The definition of the term C_{SA} is addressed later on.

The turbulent viscosity, μ_t, is defined as:

$$\mu_t = \rho\tilde{\nu} f_{v1} \tag{3.14}$$

where the the damping function, f_{v1}, is given by:

$$f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3} \tag{3.15}$$

where χ is defined by (ν being the kinematic viscosity):

$$\chi \equiv \frac{\tilde{\nu}}{\nu} \tag{3.16}$$

The model constants are given as follows:

$$\begin{gathered} c_{b1} = 0.1355, \quad c_{b2} = 0.622, \quad \kappa = 0.41, \quad \sigma = 2/3 \\ c_{w1} = \frac{c_{b1}}{\kappa^2} + \frac{1 + c_{b2}}{\sigma}, \quad c_{w2} = 0.3, \quad c_{w3} = 2, \quad c_{v1} = 7.1 \end{gathered} \tag{3.17}$$

3.2.1 SA-2020 Model

The *SA-2020* model is given in a fully consistent compressible formulation. The term $\tilde{S}$ is defined as

$$\tilde{S} = \begin{cases} \Omega_s + \bar{S} & \bar{S} \geq -c_{v_2}\Omega_s \\ \Omega_s + \dfrac{\Omega_s \left(c_{v_2}^2 \Omega_s + c_{v_3}\bar{S}\right)}{\left(c_{v_3} - 2c_{v_2}\right)\Omega_s - \bar{S}} & \text{Otherwise} \end{cases}$$

where

$$\Omega_s = \sqrt{0.5\left(2s_{ij}s_{ij} + 2\omega_{ij}\omega_{ij}\right)} \tag{3.18}$$

with c_{v_2}=0.7 and c_{v_3}=0.9. The term $\bar{S}$ is defined as:

$$\bar{S} = \frac{\tilde{\nu}}{\kappa^2 d_w^2} f_{v2} \tag{3.19}$$

where the function f_{v2} is defined as:

$$f_{v2} = 1 - \frac{\chi}{1 + \chi f_{v1}} \tag{3.20}$$

The parameter r is defined as:

$$r = \min\left(\frac{\tilde{\nu}}{\tilde{S}\kappa^2 d_w^2}, 10\right) \tag{3.21}$$

Finally, the term C_{SA} complements the fully consistent compressible formulation of the model, namely:

$$C_{SA} = -\frac{1}{\sigma}\left(\nu + \tilde{\nu}\right)\frac{\partial \rho}{\partial x_k}\frac{\partial \tilde{\nu}}{\partial x_k} \tag{3.22}$$

3.2.2 Boundary Conditions

To close the solution of the model, the boundary conditions are specified. The no-slip, wall boundary condition of $\tilde{\nu}$, denoted by $\tilde{\nu}_{wall}$, is specified by:

$$\tilde{\nu}_{wall} = 0 \tag{3.23}$$

The recommended far field inlet boundary condition of $\tilde{\nu}$, denoted by $\tilde{\nu}_\infty$, for external flows is given by: [17]:

$$\tilde{\nu}_\infty = 3\nu_\infty \; to \; 5\nu_\infty \tag{3.24}$$

This range of inflow values is appropriate for fully turbulent behavior, with $\tilde{\nu}_\infty = 3\nu_\infty$ being somewhat preferable at the lower Reynolds numbers regime. This is because it results in a free stream turbulent viscosity of $\mu_t/\mu \approx 0.2$, well below 1.0. Higher values may slightly help convergence at high Reynolds numbers by smoothing out the edges of turbulent regions, but not enough to motivate giving up the simple recommendation.

3.3 Formulation of k-ω Turbulence Models

There are numerous suggestions in the literature for two-equation turbulence models, many being variations of a few baseline models. As most two equation models contain one equation for the turbulence kinetic energy, k, one important issue is the quantity chosen to represent the length scale. While ϵ is the most obvious and popular option, it is one that causes significant problems in practice, especially in near-wall flows approaching separation. In computational aerodynamics, the most popular alternative to ϵ itself is the turbulent specific dissipation rate, ω ($\epsilon \propto k\omega$). The attraction of ω-based models is rooted in the observation that it gives a superior representation of the near-wall behavior, especially in adverse pressure gradient regions.

The transport form of the compressible k-ω turbulence model contains two transport equations. A transport equation for the turbulence kinetic energy, k,

$$\frac{\partial \rho k}{\partial t} + \frac{\partial \rho u_j k}{\partial x_j} = \frac{\partial}{\partial x_j}\left[\left(\mu + \sigma_k \mu_t\right)\frac{\partial k}{\partial x_j}\right] + P_k - \beta_k \rho k \omega \tag{3.25}$$

and a second transport equation for the turbulent specific dissipation rate, ω,

$$\frac{\partial \rho \omega}{\partial t} + \frac{\partial \rho u_j \omega}{\partial x_j} = \frac{\partial}{\partial x_j}\left[\left(\mu + \sigma_\omega \mu_t\right)\frac{\partial \omega}{\partial x_j}\right] + P_\omega - \beta_\omega \rho \omega^2 + \mathcal{CD} \tag{3.26}$$

where t denotes the time and x_j=[x,y,z] denote the Cartesian coordinates. The fluid density is denoted by ρ while the Cartesian velocity vector components are denoted by u_j=[u,v,w]. Based on the Boussinesq approximation, the production term, denoted by P_k, is defined as:

$$P_k = \mathfrak{R}_{ij}\frac{\partial u_i}{\partial x_j} \tag{3.27}$$

where $\mathfrak{R}_{ij} = \overline{\rho u_i'' u_j''}$ are the Reynolds stress tensor components. The production term is therefore modeled as:

$$P_k = \left\{\mu_t\left[\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{2}{3}\delta_{ij}\frac{\partial u_k}{\partial x_k}\right] - \frac{2}{3}\rho k \delta_{ij}\right\}\frac{\partial u_i}{\partial x_j} \tag{3.28}$$

The model coefficients as well as the cross diffusion term, $\mathcal{CD}$, depend on the actual

k-ω model. Apart from that, in general, most of the k-ω turbulence models have the same formulation as in Equations (3.25) and (3.26). The various models differ by the coefficients and the form of the source terms. The above formulation may also serve as the based model for advanced hybrid RANS/LES models, as will be outlined ahead.

3.3.1 The k-ω-TNT Turbulence Model

The TNT turbulence model has two clear advantages over other two-equation turbulence models: it uses a topology-free approach, and it is insensitive to the specific turbulence dissipation rate free-stream boundary condition. The source term of the model is given by:

$$S = \begin{Bmatrix} P_k - \beta_k \bar{\rho} k \omega \\ \alpha_\omega \frac{\omega}{k} P_k - \beta_\omega \bar{\rho} \omega^2 + \max(\mathcal{E}, 0) \end{Bmatrix} \tag{3.29}$$

The production term is denoted by P_k and it is based on the Boussinesq approximation while $\mathcal{E}$ is the cross diffusion term. The turbulent viscosity is defined as

$$\mu_t = \frac{\overline{\rho} k}{\omega} \tag{3.30}$$

The remaining model constants are $\sigma_k = 1.5$, $\sigma_\omega = 2.0$, $\sigma_d = 0.5$, $\beta_\omega = 0.075$, $\beta_k = 0.09$, $\alpha_\omega = \frac{\beta}{\beta^*} - \frac{\sigma_\omega \kappa^2}{\sqrt{\beta^*}}$, with $\kappa = 0.41$.
A description of the boundary conditions appears in Section 3.4.

3.3.2 The k-ω-SST Model

There are numerous suggestions in the literature for two-equation turbulence models, many being variations of a few baseline models. As most two equation models contain one equation for the turbulence kinetic energy, k, one important issue is the quantity chosen to represent the length scale. While ϵ is the most obvious and popular option, it is one that causes significant problems in practice, especially in near-wall flows approaching separation. In computational aerodynamics, the most popular alternative to ϵ itself is the turbulent specific dissipation rate, ω. The attraction of

ω-based models is rooted in the observation that it gives a superior representation of the near-wall behavior, especially in adverse pressure gradient regions. On the other hand, one serious flaw exhibited by ω based models is the extreme sensitivity to the value of ω at irrotational boundaries of shear flows and, by implication, also to the value in weak-shear regions within a complex shear flow. This, as well as other defects, have led Menter [18] to formulate a hybrid model which blends the k-ω model near-wall regions with the k-ϵ model in regions that are far from walls. In recent years, this model has become the most popular two-equation model in aeronautical CFD practice, especially in weakly separated flows. Again, over the years several variations and modifications to the original k-ω-SST model have appeared. The k-ω-SST-2003 (following the naming convention from the TMR website) has been chosen and implemented in the EZAir suite.

3.3.2.1 k-ω-SST-2003 Model

The transport form of the compressible k-ω-SST-2003 turbulence model contains two transport equations. A transport equation for the turbulence kinetic energy, k, and a second transport equation for the turbulent specific dissipation rate, ω. These transport equations take the following form:

$$\frac{\partial \rho k}{\partial t} + \frac{\partial \rho u_j k}{\partial x_j} = \frac{\partial}{\partial x_j}\left[\left(\mu + \sigma_k \mu_t\right)\frac{\partial k}{\partial x_j}\right] + P_k - \beta^* \rho \omega k \tag{3.31}$$

$$\begin{aligned}\frac{\partial \rho \omega}{\partial t} + \frac{\partial \rho u_j \omega}{\partial x_j} &= \frac{\partial}{\partial x_j}\left[\left(\mu + \sigma_\omega \mu_t\right)\frac{\partial \omega}{\partial x_j}\right] + \frac{\alpha_\omega}{\mu_t}\rho P_k - \beta \rho \omega^2 \\ &+ 2\left(1 - F_1\right)\frac{\rho \sigma_{\omega 2}}{\omega}\max\left(\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j}, 0\right)\end{aligned} \tag{3.32}$$

where t denotes the time and x_j=[x,y,z] denote the Cartesian coordinates. The fluid density is denoted by ρ while the Cartesian velocity vector components are denoted

by u_j=[u,v,w]. The production term, denoted by P_k, is defined as

$$P_k = \mathfrak{R}_{ij} \frac{\partial u_i}{\partial x_j} \tag{3.33}$$

where $\mathfrak{R}_{ij} = \overline{\rho u_i'' u_j''}$ are the Reynolds stress tensor components.

The coefficient β^* is a constant equal to $\beta^* = 0.09$. The rest of the model coefficients, ϕ_c=($\sigma_k,\ \sigma_\omega,\ \alpha_\omega,\ \beta$) are blended according to

$$\phi_c = F_1 \phi_1 + (1 - F_1)\, \phi_2 \tag{3.34}$$

where the coefficients ϕ_1 and ϕ_2 are given in Table 3.1.

	σ_k	σ_ω	α_ω	β
ϕ_1	0.856	0.5	5/9	0.075
ϕ_2	1.0	0.856	0.44	0.0828

Table 3.1: SST coefficients

The F_1 function, proposed by Menter [18] himself, is given as:

$$F_1 = \tanh\left(\mathfrak{Z}_1{}^4\right) \tag{3.35}$$

with the argument $\mathfrak{Z}_1$ given by:

$$\mathfrak{Z}_1 = \min\left[\max\left(\frac{\sqrt{k}}{\beta^* \omega d_w}, \frac{500\mu}{\rho \omega d_w^2}\right), \frac{4\sigma_{\omega 2} \rho k}{\max\left(2\sigma_{\omega 2} \frac{\rho}{\omega} \frac{\partial k}{\partial x_j} \frac{\partial \omega}{\partial x_j}, 10^{-10}\right) d_w^2}\right] \tag{3.36}$$

The turbulent viscosity is defined as

$$\mu_t = \frac{a_1 \rho k}{\max\left(a_1 \omega, S F_2\right)} \tag{3.37}$$

where S is:

$$S = \sqrt{2 S_{ij} S_{ij}} \tag{3.38}$$

with

$$S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{3.39}$$

and the function F_2 is defined as follow:

$$F_2 = \tanh\left(\mathfrak{Z}_2{}^2\right) \tag{3.40}$$

with

$$\mathfrak{Z}_2 = \max\left(2\frac{\sqrt{k}}{\beta^*\omega d}, \frac{500\nu}{\omega d^2}\right) \tag{3.41}$$

In the k-ω-SST-2003 version, the production term, P_k is limited as follows

$$P_k = \min\left(P_k, 10\beta^*\rho\omega k\right) \tag{3.42}$$

A description of the boundary conditions appears in Section 3.4.

3.3.2.2 k-ω-SST-SAS Model

The k-ω-SST scale-adaptive-simulation (SAS) model is based on the formulation appearing in the work by Egorov and Menter [19]. The difference between the k-ω-SST model and the k-ω-SST-SAS model is the addition of the Q_{SAS} source term to the equation for ω, namely it is added to Equation 3.32, resulting in:

$$\begin{aligned}\frac{\partial\rho\omega}{\partial t} + \frac{\partial\rho u_j\omega}{\partial x_j} &= \frac{\partial}{\partial x_j}\left[\left(\mu + \sigma_\omega\mu_t\right)\frac{\partial\omega}{\partial x_j}\right] + \frac{\alpha_\omega}{\mu_t}\rho P_k - \beta\rho\omega^2 \\ &+ 2\left(1 - F_1\right)\frac{\rho\sigma_{\omega 2}}{\omega}\max\left(\frac{\partial k}{\partial x_j}\frac{\partial\omega}{\partial x_j}, 0\right) + Q_{SAS}\end{aligned} \tag{3.43}$$

where Q_{SAS} is defined as:

$$Q_{SAS} = \max\left[\rho\xi\kappa S^2\left(\frac{L_l}{L_{vk}}\right)^2 - C\frac{2\rho k}{\sigma_\phi}\max\left(\frac{|\nabla\omega|^2}{\omega^2}, \frac{|\nabla k|^2}{k^2}\right), 0\right] \tag{3.44}$$

The modeled turbulence length scale, L_l, is given by

$$L_l = \sqrt{k} / \left(C_\mu^{1/4} \omega \right) \tag{3.45}$$

and the von Karman length scale, L_{vk}, is given by:

$$L_{vk} = \frac{\kappa S}{\left| \frac{\partial}{\partial x_j} \left(\frac{\partial u_i}{\partial x_j} \right) \right|} \tag{3.46}$$

The model constants are $C = 2.0$, $\xi = 3.51$, $\kappa = 0.41$, $\sigma_\phi = 0.67$, and $C_\mu = 0.09$.

3.4 Boundary Conditions for the k-ω-TNT and k-ω-SST Turbulence Models

To close the solution of any of the k-ω models, the boundary conditions should be specified. The no-slip wall boundary conditions of k and ω, denoted by k_{wall} and ω_{wall}, respectively, are specified as follows:

$$k_{wall} = 0 \tag{3.47}$$

$$\omega_{wall} = 10 \frac{6\nu}{\beta_1 \left(\Delta d_1 \right)^2} \tag{3.48}$$

where Δd_1 denotes the distance between the center of the first cell neighboring the wall and the wall. The inflow boundary condition of k, denoted by k_∞ for external flows is

$$k_\infty = \frac{3}{2} \left(T_u \cdot U_\infty \right)^2 \tag{3.49}$$

where Tu represents the turbulence intensity and U_∞ is the magnitude of the inflow velocity. The inflow boundary condition of ω, denoted as ω_∞ is specified as follows:

$$\omega_\infty = \frac{\bar{\rho}_\infty k_\infty}{\left(\mu_t \right)_\infty} \tag{3.50}$$

with the recommended values of $(\mu_t)_\infty$ for external flows are as follows:

$$0.01 < \frac{(\mu_t)_\infty}{(\mu)_\infty} < 1.0 \tag{3.51}$$

3.5 Evaluation of the Reynolds Stress Tensor

The TNT model is a linear eddy viscosity model (LEVM), and therefore the Reynolds stress tensor that is added to the mean flow equations is defined based on the Boussinesq assumption, namely:

$$\Re_{ij} = \Re_{ij}^{LEVM} \tag{3.52}$$

where

$$\Re_{ij}^{LEVM} = 2\mu_t \left(S_{ij} - \frac{1}{3}\frac{\partial u_k}{\partial x_k}\delta_{ij} \right) - \frac{2}{3}\bar{\rho}k\delta_{ij} \tag{3.53}$$

and

$$S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \tag{3.54}$$

Note that the turbulence kinetic energy k is not available with the SA-Edwards model, therefore the term $\frac{2}{3}\bar{\rho}k\delta_{ij}$ is neglected when using the SA-Edwards model.

3.6 Turbulence-Mean-flow Coupling

In eddy-viscosity-based models, the coefficients of viscosity μ and thermal conductivity κ are replaced by the relations

$$\begin{aligned} \mu &= \bar{\mu}_l + \bar{\mu}_t \\ \kappa &= \bar{\kappa}_l + \frac{C_p\bar{\mu}_t}{Pr_t} \end{aligned} \tag{3.55}$$

to account for the effects of turbulence on the mean-flow. The turbulent Prandtl number is assumed constant ($Pr_t = 0.9$).

3.7 Hybrid RANS/LES Turbulence Models

Hybrid RANS/LES turbulence models provide the means to conduct complex, realistic flow simulations in an affordable manner. Among others, these types of flows include unsteady flows that exhibit massive flow separation. Such flows are known to be challenging for RANS turbulence models. The need for an accurate description of complex, unsteady, separated flows while limiting computational cost has been the driving force behind the development of hybrid RANS/LES models. Hybrid models typically act in RANS mode in near-wall regions and switch to an LES mode farther from walls and/or in separated regions, where accurate prediction of turbulent mixing is critical for the prediction of complex flow, *e.g.,* flows involving chemical reactions or combustion. Hybrid models relieve the need for the high grid resolution near the wall that is required by wall-resolved LES.

Currently, the hybrid RANS/LES X-LES turbulence model and the Improved Delayed Detached Eddy Simulation (IDDES) family of hybrid RANS/LES models are available for use in the **Arion** flow solver. Two variants of the IDDES family are supported: the first is based on the Spalart-Allmaras turbulence model, namely, SA-IDDES, and the second is based on the k-ω-SST turbulence model, namely, k-ω-SST-IDDES.

3.7.1 X-LES Hybrid RANS/LES Turbulence Model

The X-LES model is a hybrid model that is based on the k-ω-TNT model (see Section 3.3.1). Developed by Kok *et al.* [20], it consists of a composite formulation incorporating both RANS and LES equations; it uses a clearly defined subgrid-scale (SGS) model in the LES mode. Namely, in the LES mode, the k equation takes the form of a one-equation SGS model for the subgrid-scale turbulent kinetic energy. The form of the convective and diffusive terms of the X-LES model equations is identical to that described in Section 3.3 (Equations (3.25) and (3.26)). A general, unified formulation of the k-ω-TNT/X-LES hybrid turbulence model source term is given

by:

$$S = \left\{ \begin{array}{c} P_k - \rho \dfrac{k^{1.5}}{l_k} \\ P_\omega - \beta_\omega \rho \omega^2 + \mathcal{CD} \end{array} \right\} \tag{3.56}$$

The dissipation length scale, l_k, is given by the RANS-LES switch as follows:

$$l_k = \min\left(l_{RANS}, l_{LES}\right) \tag{3.57}$$

where the background RANS model length scale, l_{RANS}, is given as

$$l_{RANS} = \frac{k^{1/2}}{\beta_k \omega} \tag{3.58}$$

and l_{LES} is the LES length scale defined via the subgrid length scale, Δ:

$$l_{LES} = C_{DES} \Delta \tag{3.59}$$

where $C_{DES} = 0.65$.

The turbulent viscosity is defined as

$$\mu_t = l_\nu \rho k^{1/2} \tag{3.60}$$

The eddy-viscosity length scale, l_ν, is given by:

$$l_\nu = \min\left(l_{\nu_{RANS}}, l_{\nu_{LES}}\right) \tag{3.61}$$

where

$$l_{\nu_{RANS}} = \frac{k^{1/2}}{\omega} \tag{3.62}$$

and

$$l_{\nu_{LES}} = C_1 \Delta. \tag{3.63}$$

The coefficient C_1 is set to $C_1 = 0.06$.

In the **Arion** flow solver, several definitions of the subgrid length scale Δ are available for the X-LES model. The subgrid length scale can be based on the cell

volume,

$$\Delta = V^{1/3}, \tag{3.64}$$

where V is the cell volume, on the maximum distance to a neighbor cell,

$$\Delta = h_{\max}, \tag{3.65}$$

or on the IDDES subgrid length scale defined in Section 3.7.3.1. When the IDDES subgrid length scale is selected for the X-LES model, its limiting length scale is fixed to

$$\Delta_{ls} = h_{\max}. \tag{3.66}$$

The available options are summarized in Table 9.204.

3.7.2 IDDES Formulation

The IDDES formulation, developed by Shur *et al.* [21], provides a common hybrid RANS/LES length scale that is used by both the SA-IDDES and k-ω-SST-IDDES turbulence models. The IDDES length scale, l_{IDDES}, is given by:

$$l_{IDDES} = \tilde{f}_d \left(1 + f_e\right) l_{RANS} + \left(1 - \tilde{f}_d\right) l_{LES}. \tag{3.67}$$

The RANS model length scale is given by:

$$\begin{cases} l_{RANS} = d_w & \text{SA model} \\ l_{RANS} = \dfrac{k^{1/2}}{\beta^* \omega} & \text{k-}\omega\text{-SST model.} \end{cases} \tag{3.68}$$

When $\tilde{f}_d = 1$ and $f_e = 0$, the IDDES length scale reduces to the background RANS length scale.

The LES length scale is given by:

$$l_{LES} = C_{DES} \Psi \Delta \tag{3.69}$$

where C_{DES} depends on the background RANS model and Ψ is a low-Reynolds-

number correction. When either of the IDDES turbulence models is selected, the IDDES subgrid length scale is automatically used for Δ. Its formulation and the available choices for its limiting length scale Δ_{ls} are described in Section 3.7.3.1.

For the SA-IDDES model,

$$C_{DES} = 0.65. \tag{3.70}$$

For the k-ω-SST-IDDES model, C_{DES} is given by:

$$C_{DES} = C_{DES1} F_1 + C_{DES2} (1 - F_1) \tag{3.71}$$

where $C_{DES1} = 0.78$, $C_{DES2} = 0.61$, and the function F_1 is given in Equation (3.35).

For the k-ω-SST-IDDES model,

$$\Psi = 1. \tag{3.72}$$

For the SA-IDDES model, the low-Reynolds-number correction is evaluated according to:

$$\Psi^2 = \min \left[10^2, \frac{1 - \frac{c_{b1}}{c_{w1}\kappa^2 f_w^*} \left[f_{t2} + (1 - f_{t2}) f_{v2} \right]}{f_{v1} \max \left(10^{-10}, 1 - f_{t2} \right)} \right] \tag{3.73}$$

where $f_w^* = 0.424$, and the remaining functions and coefficients are those of the SA turbulence model.

The function $\tilde{f}_d$ is a blending function given by:

$$\tilde{f}_d = \max \left[(1 - f_{dt}), f_b \right] \tag{3.74}$$

where

$$f_{dt} = 1 - \tanh \left[\left(C_{dt1} \mathbf{r}_{dt} \right)^{C_{dt2}} \right] \tag{3.75}$$

and

$$\mathbf{r}_{dt} = \frac{\nu_t}{\kappa^2 d_w^2 \sqrt{0.5 \left(S^2 + \Omega^2 \right)}}. \tag{3.76}$$

Here d_w is the distance to the nearest wall, S and Ω are the magnitudes of the strain-rate and rotation-rate tensors, respectively, and $\kappa = 0.41$.

The constants C_{dt1} and C_{dt2} depend on the background RANS model. For the original SA-IDDES formulation of Shur *et al.* [21],

$$C_{dt1} = 8, \qquad C_{dt2} = 3. \tag{3.77}$$

For the k-ω-SST-IDDES formulation, the delay function was recalibrated by Gritskevich *et al.* [22] in order to provide shielding of the attached boundary layer comparable to that of the SA-based formulation. The resulting coefficients are:

$$C_{dt1} = 20, \qquad C_{dt2} = 3. \tag{3.78}$$

The function f_b is given by:

$$f_b = \min\left[2\exp\left(-9\alpha^2\right), 1.0\right] \tag{3.79}$$

where

$$\alpha = 0.25 - \frac{d_w}{h_{\max}}. \tag{3.80}$$

The function f_e is given by:

$$f_e = \Psi f_{e2} \max\left[\left(f_{e1} - 1\right), 0\right] \tag{3.81}$$

where

$$f_{e1} = \begin{cases} 2\exp\left(-11.09\alpha^2\right) & \alpha \geq 0 \\ 2\exp\left(-9.0\alpha^2\right) & \alpha < 0 \end{cases} \tag{3.82}$$

and

$$f_{e2} = 1 - \max\left(f_t, f_l\right). \tag{3.83}$$

The functions f_t and f_l are given by:

$$f_t = \tanh\left[\left(C_t^2 \mathbf{r}_{dt}\right)^3\right] \tag{3.84}$$

and

$$f_l = \tanh\left[\left(C_l^2 \mathbf{r}_{dl}\right)^{10}\right] \tag{3.85}$$

where

$$\mathbf{r}_{dl} = \frac{\nu}{\kappa^2 d_w^2 \sqrt{0.5\left(S^2 + \Omega^2\right)}}. \tag{3.86}$$

The coefficients C_l and C_t depend on the background RANS model. For the SA-IDDES formulation,

$$C_l = 3.55, \qquad C_t = 1.63, \tag{3.87}$$

while for the k-ω-SST-IDDES formulation,

$$C_l = 5.0, \qquad C_t = 1.87. \tag{3.88}$$

3.7.2.1 The SA-IDDES Turbulence Model

The SA-IDDES formulation differs from the original SA turbulence model formulation through the destruction term (Equation 3.9), which now reads:

$$D_{SA} = c_{w_1} f_w \rho \left(\frac{\tilde{\nu}}{l_{IDDES}}\right)^2 \tag{3.89}$$

where l_{IDDES} is the IDDES length scale defined in Section 3.7.2.

3.7.2.2 The k-ω-SST-IDDES Hybrid Turbulence Model

The k-ω-SST-IDDES formulation follows the SST-specific adaptation of the IDDES model described by Gritskevich *et al.* [22]. A unified formulation for the k-ω-SST-IDDES hybrid turbulence model source term is given by:

$$S = \left\{ \begin{array}{c} P_k - \rho \dfrac{k^{1.5}}{l_{IDDES}} \\ \dfrac{\alpha_\omega}{\mu_t} \rho P_k - \beta \rho \omega^2 + 2\left(1 - F_1\right) \dfrac{\rho \sigma_{\omega 2}}{\omega} \dfrac{\partial k}{\partial x_j} \dfrac{\partial \omega}{\partial x_j} \end{array} \right\} \tag{3.90}$$

where l_{IDDES} is the IDDES length scale defined in Section 3.7.2.

3.7.3 Subgrid Length Scales

Hybrid RANS/LES turbulence models require the definition of a characteristic subgrid length scale. In the **Arion** flow solver, several grid- and flow-dependent definitions are available. The IDDES subgrid length scale and the Shear-Layer-Adapted (SLA) length scale are described in the following sections. The available options are summarized in Table 9.204.

3.7.3.1 IDDES Subgrid Length Scale

When either the SA-IDDES or k-ω-SST-IDDES turbulence model is selected, the subgrid length scale Δ is evaluated automatically using the IDDES formulation [21]:

$$\Delta = \min\left[\max\left(C_\omega d_w, C_\omega h_{\max}, h_{\min}\right), \Delta_{ls}\right]. \tag{3.91}$$

The coefficient C_ω is set to

$$C_\omega = 0.15. \tag{3.92}$$

Here d_w is the distance between the cell center and the nearest wall, $h_{\max}$ is the largest distance to the nearest neighboring cell, and $h_{\min}$ is the smallest distance to the nearest neighboring cell. In the original IDDES formulation [21], the corresponding third length scale is the local grid spacing in the wall-normal direction, h_{wn}. In **Arion**, $h_{\min}$ is used as an approximation to h_{wn}.

The limiting length scale Δ_{ls} can be evaluated using the maximum local grid length scale,

$$\Delta_{ls} = h_{\max}, \tag{3.93}$$

the cell-volume length scale,

$$\Delta_{ls} = V^{1/3}, \tag{3.94}$$

or the shear-layer-adapted length scale,

$$\Delta_{ls} = \Delta_{\mathrm{SLA}}, \tag{3.95}$$

described in Section 3.7.3.2.

When the IDDES subgrid length scale is used by the SA-IDDES or k-ω-SST-IDDES turbulence models, the limiting length scale Δ_{ls} can be selected from the options given above. When the same IDDES subgrid length scale is selected for the X-LES model, Δ_{ls} is fixed to $h_{\max}$.

3.7.3.2 Shear-Layer-Adapted Length Scale

The Shear-Layer-Adapted (SLA) length scale proposed by Shur *et al.* [23] is defined as:

$$\Delta_{\mathrm{SLA}} = \tilde{\Delta}_{\omega} F_{\mathrm{KH}} \left(\langle \mathrm{VTM} \rangle \right) \tag{3.96}$$

where $\tilde{\Delta}_{\omega}$ is a vorticity-oriented grid length scale defined as:

$$\tilde{\Delta}_{\omega} = \frac{1}{\sqrt{3}} \max_{n,m} |\mathbf{l}_n - \mathbf{l}_m| \tag{3.97}$$

with

$$\mathbf{l}_n = \mathbf{n}_{\omega} \times \mathbf{r}_n. \tag{3.98}$$

Here $\mathbf{n}_{\omega}$ is the unit vector aligned with the vorticity vector and $\mathbf{r}_n$ denotes the position of vertex n of the cell. The length scale $\tilde{\Delta}_{\omega}$ can be viewed as a measure of the cell size projected onto a plane normal to the local vorticity direction.

The vortex-tilting measure (VTM) is defined as:

$$\mathrm{VTM} = \frac{\sqrt{6}\,|(S \cdot \boldsymbol{\omega}) \times \boldsymbol{\omega}|}{\boldsymbol{\omega}^2 \sqrt{3\,\mathrm{tr}\,(S^2) - [\mathrm{tr}\,(S)]^2}} \max\left(1, \frac{\nu^*}{\nu_t}\right) \tag{3.99}$$

where

$$\nu^* = 0.2\nu. \tag{3.100}$$

Here S is the rate-of-strain tensor, $\boldsymbol{\omega}$ is the vorticity vector, ν is the molecular kinematic viscosity, and ν_t is the turbulent kinematic viscosity. The VTM measures the degree to which the rate-of-strain tensor tilts the vorticity vector toward another direction, rather than merely stretching or compressing it. The VTM is close to zero in quasi-two-dimensional shear layers and generally assumes larger values in developed

three-dimensional turbulence.

The factor $\max(1, \nu^*/\nu_t)$ in Equation (3.99) prevents activation of the SLA correction in regions with very small turbulent viscosity. In such regions, the factor increases the VTM and therefore drives F_{KH} toward its neutral value of unity.

In Equation (3.96), $\langle \mathrm{VTM} \rangle$ denotes the VTM averaged over the current cell and its face-neighboring cells. The averaging is used to suppress the strong local downward excursions of VTM that can occur in developed three-dimensional turbulence.

The function F_{KH} is aimed at unlocking the Kelvin–Helmholtz instability in the initial part of shear layers by reducing the subgrid length scale in quasi-two-dimensional flow regions. It is defined as:

$$F_{\mathrm{KH}}\left(\langle \mathrm{VTM} \rangle\right) = \max\left[F_{\mathrm{KH}}^{\min}, \min\left(F_{\mathrm{KH}}^{\max}, F_{\mathrm{KH}}^{\min} + \frac{F_{\mathrm{KH}}^{\max} - F_{\mathrm{KH}}^{\min}}{a_2 - a_1}\left(\langle \mathrm{VTM} \rangle - a_1\right)\right)\right]. \tag{3.101}$$

The coefficients are set to:

$$F_{\mathrm{KH}}^{\min} = 0.1, \qquad F_{\mathrm{KH}}^{\max} = 1.0, \qquad a_1 = 0.15, \qquad a_2 = 0.3. \tag{3.102}$$

Thus,

$$0.1 \leq F_{\mathrm{KH}}\left(\langle \mathrm{VTM} \rangle\right) \leq 1.0. \tag{3.103}$$

For small values of the vortex-tilting measure, F_{KH} reduces Δ_{SLA} and consequently the modeled subgrid viscosity. This promotes the development of the Kelvin–Helmholtz instability and accelerates the transition from modeled to resolved turbulence. In developed three-dimensional turbulence, F_{KH} approaches unity and the effect of the correction is reduced.

Chapter 4

Computational Methods

4.1 Spatial Discretization

A conservative cell-centered finite volume methodology is employed to discretize the governing equations. The computational domain is a unstructured hybrid grid that is discretized into N_{cv} non-overlapping control volumes. A control volume, C_v is defined by a grid volume element and $\partial\Gamma$ is the volume control surface, with $\mathbf{n} = [n_x, n_y, n_z]^T$ being the outward-pointing, unit normal vector to $\partial\Gamma$. Therefore, Equation (2.6) for a control volume C_v can be expressed as:

$$\frac{\partial}{\partial t}\int_{C_v} Q dV + \int_{\partial\Gamma} H dS = 0 \tag{4.1}$$

where H is the rotated flux, namely,

$$\begin{aligned} H &= H_c + H_d \\ H_c &= E_c n_x + F_c n_y + G_c n_z \\ H_d &= E_d n_x + F_d n_y + G_d n_z \end{aligned} \tag{4.2}$$

The term H_c is the convective part of the flux while H_d is the diffusive part of the flux. The semi-discrete form of Equation(4.1) for a non-deforming cell i is given by:

$$V_i \frac{dQ_i}{dt} = - \sum_{j \in N(i)} H_{ij} S_{ij} = R_i \tag{4.3}$$

where V_i denotes the cell volume, Q_i is the vector of cell-averaged conservative dependent variables, $N(i)$ denotes the set of cell i neighbors, H_{ij} is the rotated flux vector normal to the interface ij shared by cell i and cell j, and S_{ij} is the interface area. The number of cell neighbors, $N(i)$, depends on the type of the cell element. For example, a tetrahedron has 4 neighbors and therefore $N(i) = 4$ whereas for a hexahedron it is $N(i) = 6$. The term R_i signifies the residual of the equations. In what follows, the subscript "i" is dropped for compactness of the representation.

4.2 Flux Approximation Schemes

In flux difference splitting, the problem of computing the cell-face fluxes for a control volume is viewed as a series of one-dimensional Riemann problems along the direction normal to the control-volume faces. Because some of the details of the exact solution, obtained at considerable cost, are lost in the cell-averaged representation of the data, the solution of the full Riemann problem is usually replaced by methods referred to as approximate Riemann solvers. In what follows, the currently employed HLLC scheme is described in detail.

4.2.1 HLLC

The concept of average-state approximations was introduced by Harten, Lax and van-Leer [24] in 1983. The Harten, Lax and van-Leer (HLL) scheme is attractive because of its robustness, conceptual simplicity, and ease of coding, but it has the serious flaw of a diffusive contact surface. This is mainly because the HLL solver reduces the exact Riemann problem to two pressure waves and therefore neglects the contact surface. Toro *et al* [25] discussed this limitation, and proposed a modified

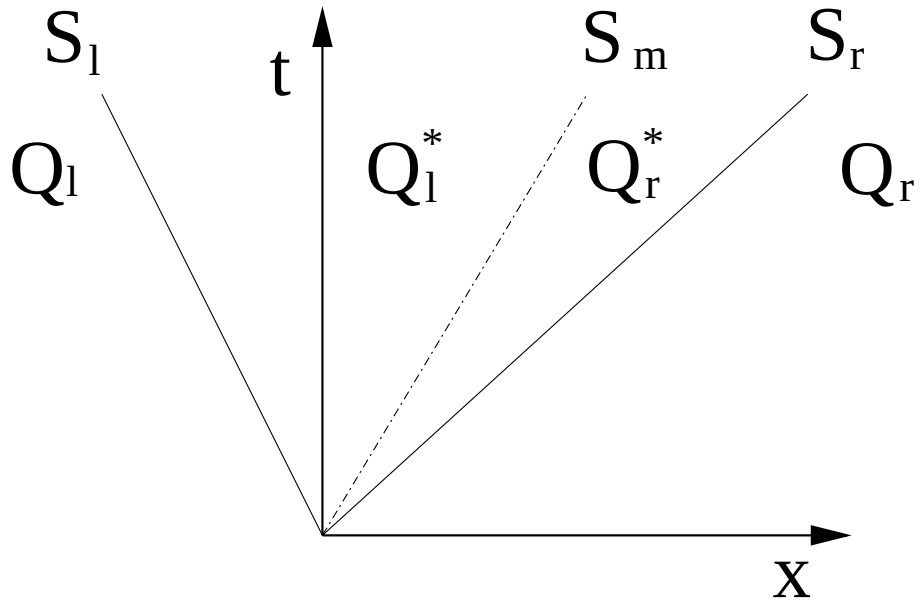


Figure 4.1: Wave structure of the HLLC approximate Riemann solver

three wave solver, named HLLC, where the contact discontinuity is explicitly present. The HLLC scheme is found to have the following properties:

1. Exact preservation of isolated contact discontinuities and shear waves.
2. Positivity preserving of a scalar quantity.
3. Enforcement of the entropy condition.

The resulting scheme greatly improves contact discontinuity resolution and has been successfully used to compute compressible viscous and turbulent flows [26].

The HLLC approximate Riemann solver that is implemented in the **Arion** code is as proposed by Batten *et al* [26]. The HLLC scheme assumes two intermediate states, $\mathbf{Q}_l^*$ and $\mathbf{Q}_r^*$ within the region bounded by the left moving wave, S_l, and the right moving wave, S_r (the subscripts "l" and "r" denote the left and right states of the approximate Riemann problem, respectively). The states $\mathbf{Q}_l^*$ and $\mathbf{Q}_r^*$ are split by the contact discontinuity, which moves with the velocity S_m (see Figure 4.1).

Two wave speed estimates can be used. In the first, the wave speeds S_l and S_r are computed according to Einfeldt *et al* [27] as follows:

$$S_l = \min[\lambda_{min}, \lambda_{min}^{Roe}] \tag{4.4}$$

$$S_r = \max[\lambda_{max}, \lambda_{max}^{Roe}] \tag{4.5}$$

where λ_{min} is the smallest eigenvalue and λ_{max} is the largest eigenvalue, evaluated at the interface. Similarly, λ_{min}^{Roe} and λ_{max}^{Roe} are the smallest and largest eigenvalues of Roe's average matrix [28], respectively. In the second, the wave speeds S_l and S_r are computed according to Davis [29] as follows:

$$S_l = \min[\lambda_{min}(L), \lambda_{min}(R)] \tag{4.6}$$

$$S_r = \max[\lambda_{max}(L), \lambda_{max}(R)] \tag{4.7}$$

The normal velocity to the interface, denoted by q, is defined as:

$$q = (u - u_g) n_x + (v - v_g) n_y + (w - w_g) n_z \tag{4.8}$$

The contact discontinuity speed S_m is evaluated according to Batten *et al* [26] by

$$S_m = \frac{\rho_r q_r (S_r - q_r) - \rho_l q_l (S_l - q_l) + p_l - p_r}{\rho_r (S_r - q_r) - \rho_l (S_l - q_l)} \tag{4.9}$$

This choice of S_m enforces the equality of the two star pressures, *i.e.*, $p^* = p_l^* = p_r^*$ which is obtained from

$$p^* = \rho_l (q_l - S_l)(q_l - S_m) + p_l = \rho_r (q_r - S_r)(q_r - S_m) + p_r \tag{4.10}$$

Introducing the intermediate left state vector

$$\mathbf{Q}_l^* = \begin{Bmatrix} \rho_l^* \\ (\rho u)_l^* \\ (\rho v)_l^* \\ (\rho w)_l^* \\ E_l^* \end{Bmatrix} = \Omega_l \begin{Bmatrix} \rho_l \left(S_l - q_l\right) \\ \left(S_l - q_l\right)(\rho u)_l + \left(p^* - p_l\right) n_x \\ \left(S_l - q_l\right)(\rho v)_l + \left(p^* - p_l\right) n_y \\ \left(S_l - q_l\right)(\rho w)_l + \left(p^* - p_l\right) n_z \\ \left(S_l - q_l\right) E_l - p_l q_l + p^* S_m \end{Bmatrix} \tag{4.11}$$

where $\Omega_l \equiv \left(S_l - S_m\right)^{-1}$. The left state flux vector becomes

$$\mathbf{H}_{c_l}^* \equiv \mathbf{H}_c\left(\mathbf{Q}_l^*\right) = \mathbf{Q}_l^* S_m + \begin{Bmatrix} 0 \\ p^* n_x \\ p^* n_y \\ p^* n_z \\ p^* S_m \end{Bmatrix} \tag{4.12}$$

and the corresponding intermediate right state vector and right flux vector are obtained from Equations (4.11), (4.12) by interchanging the subscripts $l \rightarrow r$. Finally, the numerical HLLC flux is defined as follow

$$\mathbf{H}_c\left(\mathbf{Q}_l, \mathbf{Q}_r\right) = \begin{Bmatrix} \mathbf{H}_c\left(\mathbf{Q}_l\right) & if\ S_l > 0 \\ \mathbf{H}_c\left(\mathbf{Q}_l^*\right) & if\ S_l \leq 0 < S_m \\ \mathbf{H}_c\left(\mathbf{Q}_r^*\right) & if\ S_m \leq 0 \leq S_r \\ \mathbf{H}_c\left(\mathbf{Q}_r\right) & if\ S_r < 0 \end{Bmatrix} \tag{4.13}$$

where $\mathbf{H}_c\left(\mathbf{Q}_l\right)$ and $\mathbf{H}_c\left(\mathbf{Q}_r\right)$ are the left and right analytic flux vectors, respectively.

4.2.2 AUSM

The advection upstream splitting method (AUSM) was first introduced in the year 1993 by Liou and Steffen [30]. The development of the AUSM was motivated by the desire to combine the efficiency of flux vector splitting methods (FVS) and the accuracy of flux differencing splitting methods (FDS). The key idea behind AUSM

schemes is the the fact that the inviscid flux vector consists of two physically distinct parts, namely the *convective* terms and the *pressure* terms. The convective terms can therefore be considered as passive scalar quantities convected by a suitably defined velocity. On the other hand, the pressure flux terms are governed by the acoustics wave speeds.

4.2.2.1 AUSM$^+$-up

Although AUSM schemes enjoy a demonstrated improvement in accuracy, efficiency, and robustness over existing schemes, they have been found to have deficiencies in some cases. In the year 1996, Liou improved the original AUSM, termed now the AUSM$^+$ [31]. Among the improvement features of the original AUSM scheme are the following properties: (1) exact resolution of a one-dimensional contact discontinuity and shock discontinuities, (2) positivity preserving of scalar quantities, (3) free of "carbuncle phenomenon".

In the year 2006, Liou introduced a sequel scheme to the AUSM$^+$ called the AUSM$^+$-up [32] extended for all speed flows. The AUSM$^+$-up is implemented in the **Arion** code and it is given as follows:

$$\mathbf{F}_c\left(\mathbf{Q}_l, \mathbf{Q}_r\right) = \mathbf{p}_{1/2} + \dot{m}_{1/2} \begin{Bmatrix} \boldsymbol{\psi}_l & \text{if } M_{1/2} > 0 \\ \boldsymbol{\psi}_r & \text{otherwise} \end{Bmatrix} \tag{4.14}$$

where

$$\boldsymbol{\psi}_{l/r} = \begin{Bmatrix} 1 \\ u_{l/r} \\ v_{l/r} \\ w_{l/r} \\ H_{l/r} \end{Bmatrix}, \tag{4.15}$$

the mass flux, $\dot{m}_{1/2}$ is defined as

$$\dot{m}_{1/2} = a_{1/2} M_{1/2} \begin{Bmatrix} \rho_l & \text{if } M_{1/2} > 0 \\ \rho_r & \text{otherwise} \end{Bmatrix}, \tag{4.16}$$

and the pressure flux, $\mathbf{p}_{1/2}$ is given as

$$\mathbf{p}_{1/2} = \begin{Bmatrix} 0 \\ p_{1/2}n_x \\ p_{1/2}n_y \\ p_{1/2}n_z \\ 0 \end{Bmatrix} \tag{4.17}$$

where

$$p_{1/2} = \mathcal{P}^{+}_{(5)}(M_l)p_l + \mathcal{P}^{-}_{(5)}(M_r)p_r - K_u\mathcal{P}^{+}_{(5)}(M_l)\mathcal{P}^{-}_{(5)}(M_r)(\rho_l + \rho_r)(f_a a_{1/2})(q_r - q_l) \tag{4.18}$$

is the interface pressure. The interface-normal velocity is denoted by q, H denotes the specific total enthalpy, and K_u is a constant that equals 0.75. The remaining functions are given below. The left/right Mach number at the interface, $M_{l/r}$, is defined as follows:

$$M_{l/r} = \frac{q_{l/r}}{a_{1/2}} \tag{4.19}$$

where $a_{1/2}$ is the speed of sound at the interface and it may be calculated by a simple average of a_l and a_r:

$$a_{1/2} = \frac{a_l + a_r}{2} \tag{4.20}$$

Next, the Mach number at the interface, $M_{1/2}$ is calculated as follows:

$$\begin{aligned}
\overline{M}^2 &= \frac{q_l^2 + q_r^2}{2a_{1/2}^2} \\
M_o^2 &= min\left(1, max\left(\overline{M}^2, M_\infty^2\right)\right) \\
f_a\left(M_o\right) &= M_o\left(2 - M_o\right) \\
\rho_{1/2} &= \frac{\rho_l + \rho_r}{2} \\
M_{1/2} &= \mathcal{M}^{+}_{(4)}\left(M_l\right) + \mathcal{M}^{-}_{(4)}\left(M_r\right) - \frac{K_p}{f_a} max\left(1 - \sigma\overline{M}^2, 0\right)\frac{p_r - p_l}{\rho_{1/2}a_{1/2}^2}
\end{aligned} \tag{4.21}$$

with the constants $K_p = 0.25$ and $\sigma = 1.0$. The split Mach numbers $\mathcal{M}_m^{+/-}$ are

polynomial functions of degree m (=1,2,4) of the Mach number, M, given as follows:

$$\begin{aligned}
\mathcal{M}_1^{\pm} &= \frac{1}{2}\left(M \pm |M|\right) \\
\mathcal{M}_2^{\pm} &= \pm\frac{1}{4}\left(M \pm 1\right)^2 \\
\mathcal{M}_{(4)}^{\pm}(M) &= \begin{Bmatrix} \mathcal{M}_{(1)}^{\pm} & if\ |M| > 0 \\ \mathcal{M}_{(2)}^{\pm}(1 \mp 16\beta\mathcal{M}_{(2)}^{\mp}) & otherwise \end{Bmatrix}
\end{aligned} \tag{4.22}$$

with the constant $\beta = 1/8$. Finally, the pressure polynomials are given as:

$$\mathcal{P}_{(5)}^{\pm}(M) = \begin{Bmatrix} \frac{1}{M}\mathcal{M}_{(1)}^{\pm} & if\ |M| \geq 1 \\ \mathcal{M}_{(2)}^{\pm}[(\pm 2 - M) \mp 16\alpha M\mathcal{M}_{(2)}^{\mp}) & otherwise \end{Bmatrix} \tag{4.23}$$

with the function $\alpha = \frac{3}{16}(-4 + 5f_a^2)$.

4.2.2.2 AUSM-DV

The AUSMDV [33] flux uses a blending function to combine the flux-difference (AUSMD) and flux-vector (AUSMV) versions of the basic AUSM flux to obtain a robust scheme that accurately resolves shock waves. The AUSMDV scheme has several properties that are favorable for hypersonic flows (*e.g.*, high-resolution of contact discontinuities, conservation of enthalpy for steady flows and applicability to chemically reacting flows). Therefore, this variant of the AUSM flux is recommended for simulations with the multi-component gas (MCG) physical model.

The AUSMDV flux is given as follows [33]. First, define the splittings of the interface-normal velocity, q, as follows:

$$q_l^+ = \begin{cases} \alpha_l \frac{(q_l + a_m)^2}{4a_m} + (1 - \alpha_l)q_l^{pos} & \text{if } |q_l| \leq a_m, \\ q_l^{pos} & \text{otherwise} \end{cases} \tag{4.24}$$

$$q_r^- = \begin{cases} -\alpha_r \frac{(q_r - a_m)^2}{4a_m} + (1 - \alpha_r)q_r^{neg} & \text{if } |q_r| \leq a_m, \\ q_r^{pos} & \text{otherwise} \end{cases} \tag{4.25}$$

Where,

$$q_l^{pos} = \frac{1}{2}\left(q_l + |q_l|\right) \tag{4.26}$$

$$q_r^{neg} = \frac{1}{2}\left(q_r - |q_r|\right) \tag{4.27}$$

with q_l, q_r denoting the interface-normal velocities evaluated from the left and right states (cells), respectively. Furthermore, the interface speed of sound is given by

$$a_m = max(a_l, a_r); \tag{4.28}$$

and

$$\alpha_l = \frac{2(p_l/\rho_l)}{p_l/\rho_l + p_r/\rho_r} \tag{4.29}$$

$$\alpha_r = \frac{2(p_r/\rho_r)}{p_l/\rho_l + p_r/\rho_r} \tag{4.30}$$

are functions designed to avoid dissipation at contact discontinuities.

Then, the mass flux is simply defined as

$$\dot{m}_{1/2} = \rho_l q_l^+ + \rho_r q_r^- \tag{4.31}$$

Next, define the interface pressure as

$$p_{1/2} = p_l^+ + p_r^- \tag{4.32}$$

where the splittings of the pressure, p_l^+, p_r^- are given as follows:

$$p_l^+ = \begin{cases} p_l \frac{(q_l+a_m)^2}{4a_m^2}(2 - \frac{q_l}{a_m}) & \text{if } |q_l| \le a_m, \\ p_l \frac{q_l^{pos}}{q_l} & \text{otherwise} \end{cases} \tag{4.33}$$

$$p_r^- = \begin{cases} p_r \frac{(q_r-a_m)^2}{4a_m^2}(2 + \frac{q_r}{a_m}) & \text{if } |q_r| \le a_m, \\ p_r \frac{q_l^{neg}}{q_r} & \text{otherwise} \end{cases} \tag{4.34}$$

$$\tag{4.35}$$

Then, the momentum flux vector of AUSMDV is defined by blending of the AUSMD and AUSMV fluxes, as follows:

$$(\rho \mathbf{u} q)_{1/2} = \frac{1}{2}\left[(1+s)(\rho \mathbf{u} q)_{AUSMV} + (1-s)(\rho \mathbf{u} q)_{AUSMD}\right] + p_{1/2}\mathbf{n} \tag{4.36}$$

Where the momentum flux vectors of the AUSMV and AUSMD variants are given by:

$$(\rho \mathbf{u} q)_{AUSMV} = \rho_l q_l^+ \mathbf{u_l} + \rho_r q_r^- \mathbf{u_r} \tag{4.37}$$

$$(\rho \mathbf{u} q)_{AUSMD} = \frac{1}{2}\left[\dot{m}_{1/2}(\mathbf{u_l} + \mathbf{u_r}) - |\dot{m}_{1/2}|(\mathbf{u_r} - \mathbf{u_l})\right] \tag{4.38}$$

where $\mathbf{u}_{l/r} = (u, v, w)_{l/r}$ denote the velocity vectors evaluated from the left and right states (cells) of the interface, and $s = min\left[1, K\frac{|p_r - p_l|}{min(p_l, p_r)}\right]$ is the blending function. K is typically set equal to 10.

The total energy flux is defined by

$$(\rho H q)_{1/2} = \frac{1}{2}\left[\dot{m}_{1/2}(H_l + H_r) - |\dot{m}_{1/2}|(H_r - H_l)\right] \tag{4.39}$$

And the final expression for the interface-normal AUSMDV flux vector is:

$$\mathbf{F}_c(\mathbf{Q}_l, \mathbf{Q}_r) = \begin{Bmatrix} \dot{m}_{1/2} \\ (\rho u q)_{1/2} \\ (\rho v q)_{1/2} \\ (\rho w q)_{1/2} \\ (\rho H q)_{1/2} \end{Bmatrix} \tag{4.40}$$

4.2.3 Steger-Warming Flux Vector Splitting

The Steger-Warming scheme uses the homogeneous property of the inviscid flux vector, namely $E_c(\alpha Q) = \alpha E_c(Q)$, resulting in:

$$E_c(Q) = \frac{\partial E_c}{\partial Q} Q \equiv AQ \tag{4.41}$$

where A denotes the *Jacobian matrix* of the inviscid flux vector. The matrix A may be diagonalized, so that $A = L\Lambda R$, where L, R are the corresponding left- and right-eigenvectors of A, and Λ is a diagonal matrix composed of the eigenvalues of A. To produce an upwind-biased scheme, the inviscid flux is split into upstream (positive) and downstream (negative) contributions as follows:

$$E_{c,SW} = E_c^+ + E_c^- = A_l^+ Q_l + A_r^- Q_r \tag{4.42}$$

where l and r denote values from cells left and right of an interface, and $A_l^+ = (L\Lambda^+ R)_l$ and $A_r^- = (L\Lambda^- R)_r$. The positive and negative parts of the *n-th* eigenvalue of A are given by:

$$\lambda_n^{\pm} = \frac{1}{2}\left(\lambda_n \pm |\lambda_n|\right) \tag{4.43}$$

The discretization of the convective flux using the Steger-Warming scheme may yield non-physical solutions, specifically such that violate the second law of thermodynamics, *i.e.*, result in an impossible increase in entropy [34]. One way that such non-physical phenomena can be avoided is by redefining the eigenvalues in the following manner [35]:

$$\lambda_n^{\pm} = \frac{1}{2}\left(\lambda_n \pm \sqrt{(\lambda_n)^2 + (\epsilon a)^2}\right) \tag{4.44}$$

where ϵ is a constant typically set to 0.1-0.2, and a is the local speed of sound. The entropy correction is applied to all the eigenvalues, even though only the acoustic waves need to be corrected to prevent non-physical solutions. Applying the entropy correction to all the eigenvalues does not affect the accuracy of the solution and allows to use larger time steps in the pseudo-time integration of the equations [35].

The Steger-Warming Jacobian are highly recommended for use in supersonic flows, and in other, "numerically stiff" problems where the increased dissipation introduced by the Steger-Warming Jacobians may enhance overall robustness, without compromising accuracy.

4.2.4 Passive Scalar Approach

The **Arion** code implements the passive scalar approach [26] in spatial discretization of the various model equations (*e.g.*, turbulence, finite-rate chemistry, etc.). The passive scalar approach enables to treat the extended governing equation set (including the model equations) in a similar manner to that presented for the Navier-Stokes equations. For example, when using any of the $k-\omega$ models with the HLLC scheme, the left and right state vectors are extended as follows:

$$\mathbf{Q}_l^* = \begin{Bmatrix} \rho_l^* \\ (\rho u)_l^* \\ (\rho v)_l^* \\ (\rho w)_l^* \\ E_l^* \\ (\rho k)^* \\ (\rho \phi)^* \end{Bmatrix} = \Omega_l \begin{Bmatrix} \rho_l \left(S_l - q_l\right) \\ \left(S_l - q_l\right)(\rho u)_l + \left(p^* - p_l\right) n_x \\ \left(S_l - q_l\right)(\rho v)_l + \left(p^* - p_l\right) n_y \\ \left(S_l - q_l\right)(\rho w)_l + \left(p^* - p_l\right) n_z \\ \left(S_l - q_l\right) E_l - p_l q_l + p^* S_m \\ \rho k \\ \rho \phi \end{Bmatrix} \tag{4.45}$$

The HLLC inviscid flux is then easily evaluated using:

$$\mathbf{H}_{c_l}^* \equiv \mathbf{H}_c\left(\mathbf{Q}_l^*\right) = \mathbf{Q}_l^* S_m + \begin{Bmatrix} 0 \\ p^* n_x \\ p^* n_y \\ p^* n_z \\ p^* S_m \\ 0 \\ 0 \end{Bmatrix} \tag{4.46}$$

4.2.5 High Order Flux Approximations

For a first-order-accurate approximation, the left and right state vectors are simply calculated from cell-center values left and right of the interface, respectively. To obtain a higher order flux approximation, the left and right state vectors of the convective flux are evaluated using a linear reconstruction using Green's theorem or a Taylor

series expansion and least squares method. A cell-wise gradient of the primitive variables is constructed, followed by a second order Taylor series expansion, the left and right states are reconstructed. Let the vector $\boldsymbol{W} = (W_m; m = 1, ...7)$ denote the primitive variables vector (for a general $k - \omega$ turbulence model),

$$\boldsymbol{W} = [\bar{\rho}, \widetilde{u}, \widetilde{v}, \widetilde{w}, \bar{p}, k, \omega] \tag{4.47}$$

then the left and right primitive variables are reconstructed as follows:

$$(W_m)_L = (W_m)_i + (\psi_m)_i (\nabla W_m)_i \cdot \boldsymbol{d}_i^{ij} \tag{4.48a}$$

$$(W_m)_R = (W_m)_j + (\psi_m)_j (\nabla W_m)_j \cdot \boldsymbol{d}_j^{ij} \tag{4.48b}$$

where $\boldsymbol{d}_i^{ij}$ $\left(\boldsymbol{d}_j^{ij}\right)$ is the distance vector from the mid-point of face ij to the center of cell i (j), and ψ_m is the cell limiter that is used to suppress oscillations in the solution.

4.3 Diffusive Flux Vector Discretization

The diffusive flux vector normal to the interface, $\mathcal{H}_d$, is a function of the primitive variables vector, $\mathcal{W}$, evaluated at the mid-point of face ij, $\mathcal{W}_{ij}$, and of its derivatives. The vector $\mathcal{W}_{ij}$ is calculated by averaging of adjacent cell-center values (*i.e.*, $\mathcal{W}_i$ and $\mathcal{W}_j$).

4.4 Time Marching Schemes

The **Arion** code provides various possibilities for advancing Equation (4.3) in time. This section contains a brief description of the available schemes. The schemes may be classified as follows:

1. Explicit (single and multi stage) schemes
   (a) Explicit Euler
   (b) Third and fourth order Runge-Kutta schemes

2. Implicit schemes
   (a) Point Gauss-Seidel
3. Multi-stage implicit schemes
   (a) Third fourth and fifth order Runge-Kutta implicit schemes

4.4.1 Explicit schemes

4.4.1.1 Explicit Euler Scheme

Consider the semi-discrete equation[1]:

$$V\frac{dQ}{dt} = R \tag{4.49}$$

A simple, first-order Euler explicit time marching scheme is given by:

$$\Delta Q^n = \frac{\Delta t}{V} R^n \tag{4.50}$$

where ΔQ^n is the increment of the solution between time levels , namely,

$$\Delta Q^n = Q^{n+1} - Q^n \tag{4.51}$$

4.4.1.2 Runge-Kutta Schemes

Arion provides the choice of third or fourth order Runge-Kutta schemes. Consider the semi-discrete formulation as presented in Equation (4.49), the Runge-Kutta scheme formulation is as follows:

$$\begin{aligned} Q^{(0)} &= Q^n \\ Q^{(k)} &= Q^n + \alpha_k \frac{\Delta t}{V} R^{(k-1)}, \quad k = 1, \ldots, K \\ Q^{n+1} &= Q^{(K)} \end{aligned} \tag{4.52}$$

[1] Equation (4.3) without the index

where k is the Runge-Kutta sub-step number, $K = 3$ for third order and $K = 4$ for fourth order, and α_k are the appropriate weights.

4.4.1.3 Time-Accurate Third Order Runge-Kutta TVD

Arion provides the capability to use the third order, time-accurate, Runge-Kutta TVD scheme, defined as:

$$\begin{aligned} Q^{(1)} &= Q^n + \frac{\Delta t}{V} R^n, \\ Q^{(2)} &= \frac{3}{4} Q^n + \frac{1}{4} Q^{(1)} + \frac{1}{4} \frac{\Delta t}{V} R^{(1)}, \\ Q^{n+1} &= \frac{1}{3} Q^n + \frac{2}{3} Q^{(2)} + \frac{2}{3} \frac{\Delta t}{V} R^{(2)} \end{aligned} \tag{4.53}$$

4.4.2 Implicit Time Marching Formulation

The fine grid spacing required to resolve the normal viscous terms close to the body surface requires in turn very small time steps and therefore it rules out the use of explicit methods. In explicit time-marching schemes the maximum time step is proportional to the minimum grid spacing. As a result the time-step limit imposed by stability is very small. In contrast, even though the operation count per time step is high, it is more efficient to use implicit methods. The development of a non-iterative implicit algorithm for the solution of the Navier-Stokes equations requires a time linearization of the nonlinear vectors (R). The linearization procedure is simple since the equations are written in conservation-law form. Applying the first order Euler implicit method and utilizing the Delta form of the equations results in the following implicit scheme:

$$\left(\frac{V}{\Delta t} I - \frac{\partial R}{\partial Q} \right)^n \Delta Q^n = R^n \tag{4.54}$$

where R^n is the residual at time level n as define by Equation (4.3), I is the identity matrix, Δt is the time increment between levels n and $n+1$, the term $\frac{\partial R}{\partial Q}$ is the Jacobian matrix. Note that the Jacobian matrix is first order and that it may be altered to improve stability.

Applying Equation (4.54) at every grid point results in a block-hepta-diagonal

matrix in three dimensions. The inversion of the matrix, or its approximation, may be conducted in various manners, resulting in a wide variety of implicit time marching schemes.

4.4.2.1 Point Gauss-Seidel

The exact, first order Jacobian matrix is retained only for the diagonal elements and the off diagonal Jacobian matrices are linearized based on the previous time step as follows:

$$\frac{\partial R}{\partial Q}\Delta Q \approx \left(\frac{\partial R}{\partial Q}\right)^{n} \Delta Q^{n} \tag{4.55}$$

Consequently, they can be moved to the right hand side.

4.4.2.2 Runge-Kutta Schemes

Runge-Kutta implicit schemes combine the explicit Runge-Kutta schemes as described in Section 4.4.1.2 with the PGS scheme that is described in Section 4.4.2.1 to form a robust implicit, multi-stage time marching scheme. note that this scheme requires the matrix inversions warranted by the PGS scheme at each stage and therefore requires more computer time per iteration.

4.5 Convergence Acceleration Methods

4.5.1 Krylov Methods

Krylov subspace methods were originally introduced as direct methods [36] for solving linear systems, but only gained popularity after being reintroduced as iterative methods [37]. Krylov methods are actually projection (Galerkin) methods for solving a linear set of equations $Ax = b$ using the Krylov subspace, K_j, spanned by:

$$K_j = span\left(r_0, Ar_0, A^2 r_0, ..., A^{j-1} r_0\right) \tag{4.56}$$

where $r_0 = b - Ax_0$. The main advantage of Krylov methods for large systems of equations, lies in the fact that they require only matrix-vector products to carry out

the iteration, and not the individual elements of A. This also makes them especially suitable for use with Newton's method.

The widely used Generalized Minimal RESidual (GMRES) Krylov method [38] employs Arnoldi basis vectors to form the trial subspace out of which the solution is constructed. One matrix-vector product is required per iteration to create each new trial vector, and the iterations are terminated based on an estimate of the residual that does not require explicit construction of intermediate residual vectors or solutions, which constitutes a major benefit of the algorithm. Several studies [39–41] have established the superiority of GMRES over other Krylov methods, especially in a Jacobian-Free Newton-Krylov framework. Consequently, GMRES is used to solve the large equation set arising in each Newton iteration, given by Equation 4.54.

4.5.1.1 Preconditioning

The intensive memory-pressure typically encountered in modern CFD codes has put an increased emphasis on quality preconditioning to complement Krylov methods. In fact, Krylov methods were shown to be efficient for large-scale problem only when coupled with effective preconditioning of the matrix A [39]. The **Arion** code employs the additive Schwartz method in which the action of the pre-conditioner is broken-down to smaller sub-pre-conditioners defined on individual geometric subdomains. The primary motivation for such an approach is its parallel scalability [42, 43]. Furthermore, domain-based parallelism is recognized as the form of data parallelism that most effectively exploits modern processors that possess a multi-level memory hierarchy [44].

Specifically, the **Arion** code makes use of the restricted additive Schwarz method (RASM), which eliminates interprocess communication during either the restriction or prolongation phase of the additive Schwarz technique [45], thus further increasing its suitability for massive-scale, distributed computations. The relatively small subsets of linear equations arising in the process of applying the preconditioner are solved with a local, block incomplete lower-upper (ILU) decomposition method.

4.5.1.2 Portable, Extensible Toolkit for Scientific Computation (PETSc)

PETSc, the Portable, Extensible Toolkit for Scientific Computation is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. PETSc is developed as open-source. Among the various solution methods that are available by the PETSc Toolkit, the **Arion** code utilizes certain Krylov solvers. The reader is referred to Section 9.14 for the PETSc directives that are supported by the **Arion** code.

4.5.2 Line Search

The **Arion** code uses the back-traced lines search algorithm as briefly described in this section. Let a general implicit iterative scheme be defined as:

$$\left(\frac{V}{\Delta t} - J^n\right)\Delta Q^n = R^n \tag{4.57}$$

where V is the cell volume, ΔQ^n is the vector of dependent variables correction term at iteration n, R is the residual vector, and J is the Jacobian matrix defined as:

$$J \triangleq \frac{\partial R}{\partial Q} \tag{4.58}$$

A relaxed updated solution may be given by

$$Q^{n+1} = Q^n + \alpha \Delta Q^n \tag{4.59}$$

where α is relaxation parameter. The relaxation parameter can manually prescribed or optimally calculated using a back-traced line search algorithm [46].

4.5.2.1 Back-Traced Line Search

Let the function $f(\alpha)$ be defined as

$$f(\alpha) = \frac{1}{2} R^T R \tag{4.60}$$

Consequently,

$$\begin{aligned} f(0) &= \frac{1}{2} R^{T^n} R^n \\ f(1) &= \frac{1}{2} R^{T^{n+1}} R^{n+1} \\ f'(0) &= R^{T^n} J^n \Delta Q^n \end{aligned} \tag{4.61}$$

Let the polynomial function satisfying the conditions in Equation 4.61 be defined as:

$$f(\alpha) = a\alpha^2 + b\alpha + c \tag{4.62}$$

The optimal relaxation parameter is such that minimizes the function f.

The algorithm that is implemented in the **Arion** code is as follows:

1. Solve the linear system:

$$J(Q^n)\,\Delta Q^n = -R(R^n)$$

2. Calculate the new residual:

$$R(Q^n + \Delta Q^n)$$

3. Check if the convergence is satisfactory:

$$f(Q^n + \Delta Q^n) < f(Q^n) + \beta\alpha \nabla f(Q^n)^T \Delta Q^n$$

4. If convergence is satisfactory, move on to the next Newton iteration. If not perform a line search:

$$\alpha | \min_{\alpha} f(Q^n + \alpha \Delta Q^n)$$

5. Repeat (go to 3) until the new residual is good enough or the maximum number of line search iterations is iteration reached or the minimum value of α is obtained.

The application of the lines search algorithm in the **Arion** code is in addition to

the Krylov methods (see Section 4.5.1). It is invoked through a directive (or a series thereof) to set the number of line search iterations, the convergence criterion constant, β, etc. The reader is referred to Section 9.13.2 for all of the relevant directives.

Chapter 5

Boundary Conditions

5.1 Introduction

The **Arion** code contains a wide variety of boundary conditions. Being an unstructured finite volume code, the notion of a ghost cell is utilized throughout. However, in certain cases the Jacobian and flux are explicitly dictated rather than calculated based on the ghost. In particular, the convection Jacobian and flux. In what follows, the subscript "g" signifies a ghost cell, the subscript "r" signifies a real cell where the flow is solved (a "real" cell), and the subscript "f" signifies the face (prescribed) value.

5.2 Wall Boundary Conditions

5.2.1 Impermeable Wall Conditions

Let $\hat{n}$ be the unit vector normal to the face of a boundary cell, with components (n_x, n_y, n_z). Then, the velocity in the ghost cell is taken as

$$\overline{V}_g = \overline{V}_r - 2\left(\overline{V}_r \cdot \hat{n} - \overline{V}_f \cdot \hat{n}\right)\hat{n} \tag{5.1}$$

The prescribed velocity vector $\overline{V}_f$ includes any motion of the boundary surface.

5.2.2 No-Slip Condition

The no-slip condition is much easier to implement:

$$\overline{V}_g = 2\overline{V}_f - \overline{V}_r \tag{5.2}$$

5.2.3 Adiabatic Wall

The temperature is set using:

$$T_g = T_r \tag{5.3}$$

while the pressure is set using

$$p_g = p_r \tag{5.4}$$

The density is evaluated using the equation of state.

5.2.4 Wall Function

The wall function follows the formulation of Nichols and Nelson [47] (see Table 5.2.1 for wall function activation).. The near-wall velocity profile is given by

$$\begin{aligned} y^+ = u^+ + y_0^+ \Bigg[\exp\left(\frac{\kappa}{\sqrt{\Gamma}}\left(\sin^{-1}\left(\frac{2\Gamma u^+ - \beta}{Q}\right) - \varphi\right)\right) \\ - 1 - \kappa u^+ - \frac{1}{2}(\kappa u^+)^2 - \frac{1}{6}(\kappa u^+)^3 \Bigg]. \end{aligned} \tag{5.5}$$

with

$$u^+ = \frac{u}{u_\tau}, \qquad y^+ = \frac{\rho_w u_\tau y}{\mu_w}, \qquad u_\tau = \sqrt{\frac{|\tau_w|}{\rho_w}}, \tag{5.6}$$

where u is the local tangential velocity magnitude, y is the wall-normal distance from the wall, and $\nu_w = \mu_w/\rho_w$ is the kinematic viscosity at the wall.

In addition,

$$\Gamma = \frac{r\,u_\tau^2}{2c_p T_w}, \qquad \beta = \frac{q_w \mu_w}{\rho_w T_w k_w u_\tau}, \qquad \varphi = \sin^{-1}\left(-\frac{\beta}{Q}\right), \tag{5.7}$$

$$Q = \sqrt{\beta^2 + 4\Gamma}, \qquad y_0^+ = e^{-\kappa B}. \tag{5.8}$$

The dimensionless parameter Γ models compressibility effects, whereas β models wall heat-transfer effects. The wall shear stress τ_w appears in the definitions of u^+, y^+, Γ, and β. The wall heat flux q_w appears only in the definition of β.

The temperature profile is given by

$$T = T_w \left(1 + \beta u^+ - \Gamma (u^+)^2\right). \tag{5.9}$$

For an adiabatic wall, the velocity and temperature equations are decoupled. The velocity equation is solved first for τ_w, and the temperature equation is then solved for T_w. For an isothermal wall, the equations are solved simultaneously for τ_w and q_w. Once the solution is obtained, the turbulent viscosity in the first off-wall cell is computed as

$$\mu_{t,w+1} = \mu_w \left.\frac{dy^+}{du^+}\right|_{w+1} - \mu_{w+1}. \tag{5.10}$$

Here, μ_w is the molecular viscosity at the wall, and the subscript $w+1$ denotes the value of a variable in the first cell adjacent to the wall. The turbulence variables at the first off-wall cell are then evaluated according to the selected model.

For k–ω models, the specific dissipation rate is computed as

$$\omega_{w+1} = \sqrt{\left(\frac{6\nu_w}{b_1\ d^2}\right)^2 + \left(\frac{u_\tau}{\sqrt{\beta^*}\,\kappa d}\right)^2}. \tag{5.11}$$

Where d is the distance from the wall. The turbulent kinetic energy depends on the k–ω variant. For the TNT model,

$$k_{w+1} = \frac{\mu_{t,w+1}\,\omega_{w+1}}{\rho_{w+1}}. \tag{5.12}$$

For the SST model the implicit relation is used,

$$k_{w+1} = \frac{\mu_{t,w+1}}{\rho_{w+1}} \max\left(\omega_{w+1}, \frac{\Omega_{w+1} F_2\left(k_{w+1}\right)}{a_1}\right). \tag{5.13}$$

For the Spalart–Allmaras model, the model variable is obtained from the implicit relation

$$\mu_{t,w+1} = \rho_{w+1}\,\hat{\nu}_{w+1}\,f_{v1}(\hat{\nu}_{w+1}). \tag{5.14}$$

Note that the wall-function procedure is applied only when $y^+ > 5$; otherwise, the wall shear stress and heat flux are computed in the same way as for a fully resolved wall.

5.3 Far Field Conditions

5.3.1 Turkel Type Conditions

The characteristic relations that are due-to Turkel are utilized. For supersonic inflow, the flow quantities at the inflow boundary are set based on the current values of the corresponding boundary:

5.3.1.1 Turkel Inlet

$$\rho_g = \rho_\infty \tag{5.15}$$

$$\overline{V}_g = \overline{V}_\infty \tag{5.16}$$

$$p_g = \frac{\rho_\infty}{\rho_r} p_r \tag{5.17}$$

5.3.1.2 Turkel Outlet

$$\rho_g = \rho_r + \frac{p_\infty - p}{a_\infty^2} \tag{5.18}$$

$$\overline{V}_g = \overline{V}_r \tag{5.19}$$

$$p_g = p_\infty \tag{5.20}$$

5.3.2 Riemann Type Conditions

Let q be the normal to the boundary face velocity. The Riemann invariants are calculated based on the following:

$$\begin{aligned} R^+ &= q_r - \frac{2}{\gamma - 1} a_\infty \\ R^- &= q_\infty - \frac{2}{\gamma - 1} a_r \end{aligned} \tag{5.21}$$

The ghost Riemann invariants are obtained using:

$$R_g = \frac{1}{2}\left(R^+ + R^-\right) \tag{5.22}$$

5.3.2.1 Riemann Inlet

$$\begin{aligned} \rho_g &= \frac{\rho_\infty}{a_\infty^{\frac{1}{\gamma-1}}} \left\{ \underbrace{\left[\frac{\gamma - 1}{4}\left(R^- - R^+\right) \right]^{\frac{1}{\gamma-1}}}_{a_g} \right\}^2 \\ p_g &= \sqrt{a_g}\rho_g\gamma \end{aligned} \tag{5.23}$$

5.3.2.2 Riemann Outlet

Let s be the entropy, the relations for ρ_g and p_g are given by:

$$\begin{aligned} s &= \frac{\rho^\gamma}{\gamma p} \\ \rho_g &= \left(a_g^2 s\right)^{\frac{1}{\gamma-1}} \\ p_g &= \sqrt{a_g}\rho_g\gamma \end{aligned} \tag{5.24}$$

5.3.3 Fixed (Supersonic Inlet)

$$\begin{aligned} \overline{V}_g &= \overline{V}_\infty \\ p_g &= p_\infty \\ T_g &= T_\infty \end{aligned} \tag{5.25}$$

5.3.4 Extrapolation (Supersonic Outlet)

$$\begin{aligned} \overline{V}_g &= \overline{V}_r \\ p_g &= p_r \\ T_g &= T_r \end{aligned} \tag{5.26}$$

5.3.5 Inlet

For supersonic inlet, a fixed boundary condition is applied (see Section 5.3.3). For subsonic inlet, the pressure is extrapolated from within the physical domain and the rest of the flow variables are fixed. This is automatically repeated every time step.

$$\begin{aligned} (\rho, u, v, w)_g &= (\rho, u, v, w)_\infty \\ p_g &= p_r \end{aligned} \tag{5.27}$$

5.3.6 Outlet

For supersonic outlet, an extrapolated boundary condition is applied (see Section 5.3.4). For subsonic outlet, a "Fixed," prescribed value is used for the pressure and the rest of the flow variables are extrapolated. This is automatically repeated every time step. Formally, this is a pressure outlet type boundary condition.

$$\begin{aligned} (\rho, u, v, w)_g &= (\rho, u, v, w)_r \\ p_g &= p_{bc} \end{aligned} \tag{5.28}$$

where p_{bc} is the prescribed pressure outlet. If no pressure outlet is prescribed then $p_{bc} \equiv p_\infty$.

5.3.6.1 Prescribed Mass Flow Rate

To obtain the prescribed mass flow rate at an outlet, the pressure is iteratively corrected in the following manner:

$$p_g = p_{bc} + \delta p \tag{5.29}$$

where p_{bc} is the prescribed pressure outlet (see Table 9.110). The pressure correction, δp, is calculated as follows:

$$\delta p_{new} = \omega \frac{|\dot{m}|^n \left(\dot{m}^n - \dot{m}_{bc}\right)}{\rho_{ave}^n A^2} + \delta p_{old} \tag{5.30}$$

where $\dot{m}^n$ is the computed mass flow rate at the current iteration, ρ_{ave}^n is the computed average density at the current iteration, A is the total area of the boundary, and δp_{new} and δp_{old} are the new and previous calculated pressure corrections, respectively. The parameter ω is a relaxation parameter (see Table 9.90). The mass flow rate is used in conjunction with the "outlet" boundary condition (see Table 9.89).

5.3.7 Inout

These boundary conditions are specific to low subsonic flows. With the exception of the pressure, these boundary conditions set "Fixed" conditions for inlet and "Extrapolation" conditions for outlet. Namely,

- **Inlet :**

$$\begin{aligned} (\rho, u, v, w)_g &= (\rho, u, v, w)_\infty \\ p_g &= p_r \end{aligned} \tag{5.31}$$

- **Outlet :**

$$\begin{aligned}(\rho, u, v, w)_g &= (\rho, u, v, w)_r \\ p_g &= p_\infty \end{aligned} \tag{5.32}$$

5.3.8 Mass Flow Inlet

The mass flow inlet boundary condition prescribes the total mass flow rate $\dot{m}$ and total temperature T_0. The flow direction is normal to the inlet boundary by default, but an arbitrary direction may also be specified. The mass flow is prescribed using the `-mass.flow` boundary condition option (see Table 9.89).

The prescribed mass flow rate is distributed over the inlet boundary according to the face area:

$$\dot{m}_f = \dot{m}\frac{A_f}{A} \tag{5.33}$$

where A_f is the area of the boundary face and A is the total inlet area. The corresponding mass flux is

$$\dot{m}'' = \frac{\dot{m}}{A} \tag{5.34}$$

The velocity at each boundary face is defined by

$$\mathbf{V}_f = U_f \hat{\mathbf{s}} \tag{5.35}$$

where U_f is the velocity magnitude and $\hat{\mathbf{s}}$ is the prescribed unit direction vector. By default, $\hat{\mathbf{s}} = -\hat{\mathbf{n}}$, where $\hat{\mathbf{n}}$ is the outward unit normal to the boundary. Conservation of mass gives

$$\dot{m}'' = -\rho_f \mathbf{V}_f \cdot \hat{\mathbf{n}} \tag{5.36}$$

and therefore

$$U_f = -\frac{\dot{m}''}{\rho_f(\hat{\mathbf{s}} \cdot \hat{\mathbf{n}})} \tag{5.37}$$

For subsonic inflow, the static pressure is extrapolated from the adjacent interior cell:

$$p_f = p_r \tag{5.38}$$

The static temperature is determined from the prescribed total temperature and the conservation of total enthalpy:

$$H(T_0) = H(T_f) + \frac{U_f^2}{2} \tag{5.39}$$

where the appropriate thermodynamic formulation is used according to the gas model.

Ideal Gas For a calorically perfect ideal gas,

$$\rho_f = \frac{p_r}{RT_f}, \qquad H(T) = C_p T \tag{5.40}$$

Combining the mass flux and total enthalpy equations gives

$$C_p T_0 = C_p T_f + \frac{1}{2}\left(\frac{\dot{m}'' R T_f}{p_r(\hat{\mathbf{s}}\cdot\hat{\mathbf{n}})}\right)^2 \tag{5.41}$$

Rearranging gives a quadratic equation for the static temperature:

$$\frac{1}{2}\left(\frac{\dot{m}'' R}{p_r(\hat{\mathbf{s}}\cdot\hat{\mathbf{n}})}\right)^2 T_f^2 + C_p T_f - C_p T_0 = 0 \tag{5.42}$$

The positive root is selected, and the density and velocity are calculated from the equations above.

Equilibrium Gas For an equilibrium gas, density, enthalpy, and entropy depend on pressure and temperature:

$$\rho = \rho(p,T), \qquad H = H(p,T), \qquad s = s(p,T) \tag{5.43}$$

The total enthalpy is determined from the specified stagnation temperature T_0 and the corresponding stagnation pressure p_0. Assuming an isentropic stagnation process,

$$s(p_r, T_f) = s(p_0, T_0) \tag{5.44}$$

and conservation of total enthalpy gives

$$H(p_0, T_0) = H(p_r, T_f) + \frac{1}{2}\left(\frac{\dot{m}''}{\rho(p_r, T_f)(\hat{\mathbf{s}} \cdot \hat{\mathbf{n}})}\right)^2 \tag{5.45}$$

These two equations are solved for the static temperature T_f and stagnation pressure p_0. The static density and velocity are then reconstructed using the equilibrium equation of state.

Non-Equilibrium Gas For a chemically non-equilibrium gas, the inlet species mass fractions Y_i are prescribed. The composition is assumed frozen when converting total temperature to static temperature.

The mixture density is given by

$$\rho_f = \frac{p_r}{R(\mathbf{Y})T_f} \tag{5.46}$$

where $R(\mathbf{Y})$ is the mixture gas constant. The mixture enthalpy is calculated from the species enthalpies:

$$H(T, \mathbf{Y}) = \sum_i Y_i H_i(T) \tag{5.47}$$

Conservation of total enthalpy gives

$$H(T_0, \mathbf{Y}) = H(T_f, \mathbf{Y}) + \frac{1}{2}\left(\frac{\dot{m}'' R(\mathbf{Y}) T_f}{p_r(\hat{\mathbf{s}} \cdot \hat{\mathbf{n}})}\right)^2 \tag{5.48}$$

or equivalently,

$$\frac{1}{2}\left(\frac{\dot{m}'' R(\mathbf{Y})}{p_r(\hat{\mathbf{s}} \cdot \hat{\mathbf{n}})}\right)^2 T_f^2 + H(T_f, \mathbf{Y}) - H(T_0, \mathbf{Y}) = 0 \tag{5.49}$$

This nonlinear equation is solved iteratively using the temperature-dependent species thermodynamic properties. The density and velocity are subsequently reconstructed.

Supersonic Inflow The subsonic formulation applies when the normal Mach number satisfies

$$M_n = \frac{|\mathbf{V}_f \cdot \hat{\mathbf{n}}|}{a_f} < 1 \tag{5.50}$$

where a_f is the local speed of sound.

For supersonic inflow, the static pressure cannot be extrapolated from the interior. An additional prescribed static pressure is required to fully determine the inlet state. The thermodynamic state is then reconstructed using the prescribed pressure instead of p_r.

5.4 Stagnation Inlet

For subsonic flow, the stagnation inlet boundary condition is implemented as follows. The user prescribes the stagnation temperature and stagnation pressure (see Tables 9.109 and 9.110). Let the negative Riemann invariant be defined as:

$$R^- = q - \frac{2}{\gamma - 1} a \tag{5.51}$$

where $q = \overline{V} \cdot \hat{n}$ is the velocity normal to the face, a is the speed of sound, and γ is the ratio of specific heats.

Applying the Riemann invariance results in

$$R_g^- = q_g - \frac{2}{\gamma - 1} a_g = R_r^- \tag{5.52}$$

The subscript r pertains to the real flow cell, *i.e.*, extrapolated from the interior of the domain, while the subscript g pertains to the value at the ghost cell (boundary). By assuming a calorically perfect gas one can write on the boundary

$$c_p T_0 = c_p T_g + \frac{q_g^2}{2} \tag{5.53}$$

Consequently, the speed of sound at the ghost cell can be evaluated using

$$a_g = \sqrt{(\gamma - 1)\, c_p \left(T_0 - \frac{q_g^2}{2c_p} \right)} \tag{5.54}$$

Substituting Equation 5.54 into Equation 5.52 results in a quadratic equation in q_g. The pressure, density, and temperature at the ghost cell are calculated using isentropic relations.

5.5 Symmetry Boundary Conditions

Symmetric boundary conditions are treated exactly as an adiabatic impermeable wall.

Chapter 6

Computational Mesh

The **Arion** code is considered a hybrid code since it supports various cell element types. The code supports tetrahedra, hexahedra, prism, and pyramids. Mesh generation may be conducted by any unstructured grid generator, however, users must export the mesh using a Star-CD export or a CGNS export.

6.1 Star-CD Export

The Star-CD export results in three files. The first, a file containing the vertex information, having the extension ".vrt." The second, a file containing the cell elements, having the extension ".cel." And finally, a file containing the boundary elements (triangles or quads), having the extension ".bnd." The ".bnd" file contains the boundary elements as well as a name for each element. The naming is conducted by the user using the grid generation package (*e.g.,* Pointwise). The code supports two variations of the Star-CD format, the export by Pointwise and the export by CENTAUR (only 3-D export is supported).

6.2 CGNS Export

The CGNS export results in a single file. The file extension of the CGNS file is ".cgns."

6.3 Automated Mesh Adaptation

An automated mesh adaptation procedure is available in the **Arion** code. It requires the use of the Pointwise mesh generation package and can be facilitated through the EZOpt optimization software package. A feature based adaptation sensor that is proportional to an interpolation error is utilized. Using a series of options, a point cloud data file is generated and the mesh is re-generated using the previously generated computational mesh and the point cloud data. The method preserves the boundary layer mesh and it maintains smooth mesh size. The sensor function is given by:

$$S = K \left|\vec{h}\right|^p \left|\left[\hat{\vec{h}} \cdot \frac{\partial \hat{\Phi}}{\partial x_2} - \hat{\vec{h}} \cdot \frac{\partial \hat{\Phi}}{\partial x_1}\right]\right| \tag{6.1}$$

where $\hat{\vec{h}}$ is the distance vector between adjacent points and Φ is problem dependent. The parameters K and p may be prescribed by the user. Currently the **Arion** code supports sensor functions that depend on the Mach number, pressure, density, and velocity magnitude (Φ). The available adaptation options are described in Section 9.22.

Chapter 7

Aeroelasticity

A typical structural finite-element model that is used for an aeroelastic analysis may contain a very large number of degrees of freedom. In many applications, however, the deformation of interest can be accurately represented by a smaller set of structural vibration modes. The structural displacement is then expressed in terms of generalized, or modal, coordinates [48]. Namely,

$$\{\boldsymbol{u}\} = [\Phi]\{\boldsymbol{\xi}\}, \tag{7.1}$$

where $[\Phi]$ contains the selected structural mode shapes and $\{\boldsymbol{\xi}\}$ is the corresponding vector of generalized displacements.

The structural modes are obtained from the free-vibration eigenvalue problem

$$\left([K] - \omega_i^2[M]\right)\{\Phi_i\} = \{0\}, \tag{7.2}$$

where $[K]$ and $[M]$ are the structural stiffness and mass matrices, respectively, and ω_i is the natural angular frequency of mode i.

For a modal basis formed from structural eigenmodes, the generalized mass and stiffness matrices are diagonal. The structural problem can therefore be expressed using the selected generalized coordinates rather than the complete finite-element displacement vector. The number of retained modes should be chosen so that the structural response of interest is adequately represented; this should be established

by modal-convergence assessment rather than by a fixed number of modes [48].

Arion uses such a modal structural representation to couple the flow solution to the structural model, calculate the aeroelastic surface deformation, and transfer the aerodynamic loads back to the structural generalized coordinates. The resulting aeroelastic surface deformation is subsequently propagated into the CFD volume mesh using the **Arion** elastic mesh-deformation system. A comprehensive description of the specific mesh-deformation method that is used by **Arion** is described in ISCFDC Report 2025-15 [49].

7.1 Static Aeroelasticity

The static structural deformation is represented by the retained elastic modes,

$$\{\boldsymbol{u}_E\} = [\Phi_E]\{\boldsymbol{\xi}_E\}, \tag{7.3}$$

where $[\Phi_E]$ contains the retained elastic modes and $\{\boldsymbol{\xi}_E\}$ is the generalized elastic displacement vector. The corresponding static equilibrium equation in generalized coordinates is

$$[K_E]\{\boldsymbol{\xi}_E\} = \{\boldsymbol{F}_E\}, \tag{7.4}$$

where $[K_E]$ is the generalized stiffness matrix and $\{\boldsymbol{F}_E\}$ is the generalized aerodynamic-force vector. For structural eigenmodes, the generalized stiffness matrix is diagonal. The static equilibrium can therefore be evaluated independently for each retained elastic mode. The generalized aerodynamic forces are obtained by projecting the aerodynamic loads onto the corresponding aerodynamic-interface modal shapes,

$$\{\boldsymbol{F}_E\} = [\Phi_A]^T\{\boldsymbol{F}_A\}, \tag{7.5}$$

where $[\Phi_A]$ contains the retained structural modes after they have been mapped to the aerodynamic interface. This projection follows directly from virtual work. Using consistent deformation and load-transfer operators is particularly important when the structural and aerodynamic discretization are non-coincident [50].

7.2 Aeroelastic Model

The aeroelastic model is assembled from one or more aerodynamic components. An elastic component represents a portion of the aerodynamic surface for which structural modal data are available. Several elastic components may therefore be used to represent different aerodynamic portions of the same structural system. Section 9.10.2 describes how to add and manage *Aeroelastic Components*.

All elastic components participating in the same aeroelastic solution use the same retained generalized coordinates and must represent a consistent modal basis. In particular, corresponding retained modes must have consistent physical mode identification, eigenvalues, generalized modal masses, and generalized modal stiffnesses.

A component may also be defined as rigidly attached to an elastic component. Such a component has no independent elastic degrees of freedom. Its deformation is determined from the local deformation of an attachment frame on its parent component. This representation is useful for stores, pylons, sensors, fins, or other nominally rigid aerodynamic bodies that are mounted on an elastic structure (see Section 9.10.2.5).

7.2.1 Structural Modal Data

For each elastic component, the structural description consists of structural reference coordinates and the modal displacement field at these coordinates. These are supplied either through a NSTRAN grid and modes file or the ZAERO FREE FORMAT (see Section 9.10.2.1, Tables 9.74, 9.75, and 9.76). For a general three-dimensional structural model, a mode may contain a translational part, $\boldsymbol{\phi}_i^T$, and a rotational part, $\boldsymbol{\phi}_i^R$. The modal translation defines the displacement of the structural reference points. Modal rotations may additionally be used by mapping methods that account for the deformation of aerodynamic points offset from a structural reference line or surface. Each retained elastic mode has an eigenvalue

$$\lambda_i = \omega_i^2 \tag{7.6}$$

and generalized modal mass M_i. The corresponding generalized modal stiffness is

$$K_i = M_i \omega_i^2 = M_i \lambda_i. \tag{7.7}$$

For mass-normalized modes,

$$M_i = 1, \tag{7.8}$$

and therefore

$$K_i = \lambda_i. \tag{7.9}$$

If the modes are not mass normalized, the supplied generalized modal masses must be consistent with the normalization that was used when the structural modes were generated. When structural data are supplied in ZAERO FREE FORMAT, the generalized modal mass is supplied by `GENM`, while `FREQ` specifies the natural angular frequency in rad/s [51].

The complete structural GRID set need not necessarily be used as the mapping source. A selected subset may be more appropriate when, for example, the aerodynamic surface is to be related to a structural elastic axis or to a selected structural skin surface. The selected structural source points define the geometry and modal data from which the displacement field is transferred to the aerodynamic interface (see Table 9.77).

7.2.2 Elastic and Rigid-Body Modes

A free structural model may contain rigid-body modes in addition to elastic modes. In three dimensions there may be up to six rigid-body modes, corresponding to three translations and three rotations. The exact eigenvalue of a rigid-body mode is zero, although small nonzero values may occur numerically. **Arion** classifies modes below the user-supplied rigid-mode eigenvalue tolerance as rigid and excludes them from the static elastic equilibrium (see Table 9.61).

This distinction is required because the static modal equation contains the modal stiffness in the denominator, whereas a rigid-body mode has zero modal stiffness. The retained static basis therefore contains elastic modes with positive modal stiff-

ness. General rigid-body equilibrium and six-degree-of-freedom vehicle motion are not solved by the present static formulation. The user may retain a prescribed set of physical elastic modes, a specified number of the lowest non-rigid modes, or all available non-rigid modes. The retained modes set should be sufficiently complete to represent the deformation generated by the aerodynamic loading over the range of interest. The selected modes can be set by using the options that are described in Tables 9.59 and 9.60.

7.3 Structural and Aerodynamic Coordinate Systems

The structural and aerodynamic models need not be defined in the same coordinate system. Before constructing the structural-to-aerodynamic mapping, structural coordinates and modal vectors are expressed in a common CFD/global Cartesian frame. Structural coordinate-system definitions are first resolved, after which the user may supply an ordered sequence of rigid coordinate transformations. For a rotation represented by matrix $[R]$ about pivot $\boldsymbol{c}$, a structural reference coordinate is transformed according to

$$\boldsymbol{x}' = \boldsymbol{c} + [R]\,(\boldsymbol{x} - \boldsymbol{c})\,. \tag{7.10}$$

A subsequent translation $\boldsymbol{t}$ gives

$$\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{t}. \tag{7.11}$$

When several operations are supplied, they are applied in the specified order. The accumulated transformation may be written as

$$\boldsymbol{x}' = [R]_{\text{tot}}\boldsymbol{x} + \boldsymbol{t}_{\text{tot}}. \tag{7.12}$$

Modal translation and rotation vectors are transformed only by the rotational part,

$$\boldsymbol{\phi}_i^{T\prime} = [R]_{\text{tot}}\boldsymbol{\phi}_i^{T}, \qquad \boldsymbol{\phi}_i^{R\prime} = [R]_{\text{tot}}\boldsymbol{\phi}_i^{R}. \tag{7.13}$$

Translations and rotation pivots therefore modify the structural reference position but do not introduce artificial components into the modal vectors. All mapping operations described below are performed after the structural coordinate transformation. The structural transformations are described in Section 9.10.2.2, Table 9.78.

7.4 Structural-to-Aerodynamic Mapping

The structural finite-element discretization and the CFD aerodynamic-surface discretization are generally non-coincident. An interpolation or mapping procedure is therefore required to evaluate the structural mode shapes at the aerodynamic interface. Spline interpolation of structural mode shapes has long been used specifically for aeroelastic calculations [52].

Let $\boldsymbol{x}_{s,j}$ denote the selected structural source points. For mode i, the structural data at these points consist of modal translation $\boldsymbol{\phi}^T_{i,j}$ and, when available and required by the selected mapping method, modal rotation $\boldsymbol{\phi}^R_{i,j}$. The mapping defines an aerodynamic-interface modal displacement

$$\boldsymbol{\Phi}_{A,i}(\boldsymbol{x}) \tag{7.14}$$

at an arbitrary aerodynamic point $\boldsymbol{x}$. The aerodynamic-surface displacement is then

$$\boldsymbol{u}_A(\boldsymbol{x}) = \sum_{i=1}^{N_m} \boldsymbol{\Phi}_{A,i}(\boldsymbol{x})\, \xi_i. \tag{7.15}$$

The mapping is evaluated at aerodynamic surface nodes, where it defines the surface deformation, and at aerodynamic face centers, where the same modal information is used to project aerodynamic forces onto the generalized coordinates. The use of consistent deformation and load-transfer operators is particularly important for coupled fluid/structure calculations with non-matching interfaces [50].

7.4.1 Work-Conservative Generalized-Force Projection

The same mapped modal shapes that are used to deform the aerodynamic surface are used to project aerodynamic forces into the generalized coordinates. For aerodynamic face-force vectors $\boldsymbol{F}_f$, the generalized force associated with mode i is

$$F_{E,i} = \sum_f \boldsymbol{\Phi}_{A,i}(\boldsymbol{x}_f) \cdot \boldsymbol{F}_f. \tag{7.16}$$

For multiple aerodynamic components, the contribution from every component is summed for each retained mode. Equation (7.16) follows directly from virtual work. A virtual modal increment $\delta\xi_i$ gives

$$\delta\boldsymbol{u}_f = \boldsymbol{\Phi}_{A,i}(\boldsymbol{x}_f)\delta\xi_i. \tag{7.17}$$

Consequently,

$$\sum_f \boldsymbol{F}_f \cdot \delta\boldsymbol{u}_f = \sum_i F_{E,i}\,\delta\xi_i. \tag{7.18}$$

The displacement transfer and generalized-force projection are therefore energetically consistent when the same mapped modal shapes are used for both operations. Conservation of work in load and deformation transfer between non-matching fluid and structural interfaces is discussed in detail by Farhat *et al* [50].

7.4.2 Radial Basis Function Mapping

Radial basis functions (RBF) provide a mesh-independent interpolation procedure for transferring data between non-coincident structural and aerodynamic discretization. Their use for fluid–structure information transfer and aeroelastic coupling is described, for example, by Beckert and Wendland [53] and by Rendall and Allen [54].

For one component of a modal field, the RBF interpolant may be written as

$$s(\boldsymbol{x}) = \sum_{j=1}^{N_s} a_j \varphi\left(\|\boldsymbol{\chi}(\boldsymbol{x}) - \boldsymbol{\chi}(\boldsymbol{x}_{s,j})\|\right) + p(\boldsymbol{\chi}(\boldsymbol{x})), \tag{7.19}$$

where φ is the selected radial kernel, p is the polynomial augmentation, and $\boldsymbol{\chi}$ is the coordinate representation that is used for interpolation. Depending on the geometry of the structural source points, the interpolation coordinates may be three-dimensional, projected onto a plane, represented in a local surface coordinate system, or reduced to a principal structural axis. In the translation-only RBF formulation, Equation (7.19) is applied independently to the three components of $\boldsymbol{\phi}_i^T$. Structural modal rotations do not explicitly contribute to the aerodynamic displacement. This form is appropriate when the aerodynamic interface lies sufficiently close to the structural mapping surface or when offset-induced rotational deformation is not important.

7.4.3 Rotation-Aware RBF Mapping

When an aerodynamic surface is offset from the structural reference line or surface, translation alone may not reproduce the deformation produced by structural bending and torsion. Structural rotational information can therefore be used together with translational deformation when reconstructing the aerodynamic deformation [55].

The rotation-aware RBF formulation that is used by **Arion** interpolates modal translation and modal rotation. At an aerodynamic point $\boldsymbol{x}$, the modal displacement is evaluated using the small-rotation rigid-arm relation

$$\boldsymbol{\Phi}_{A,i}(\boldsymbol{x}) = \boldsymbol{T}_i(\boldsymbol{x}_r) + \boldsymbol{\theta}_i(\boldsymbol{x}_r) \times (\boldsymbol{x} - \boldsymbol{x}_r)\,, \tag{7.20}$$

where $\boldsymbol{x}_r$ is the associated structural reference position, $\boldsymbol{T}_i$ is the interpolated modal translation, and $\boldsymbol{\theta}_i$ is the interpolated small-rotation vector. The definition of $\boldsymbol{x}_r$ depends on the structural source geometry. For a beam-like source, the reference position is obtained from the structural reference line. For a surface-like source, a local structural surface is used. Using the local structural geometry avoids introducing an artificial rigid-arm offset at the structural source points themselves.

7.4.4 One-Dimensional Beam Mapping

The one-dimensional beam mapping is intended for structures that are represented by an elastic reference line. Line- and beam-based spline coupling is an established approach for aeroelastic transfer between structural and aerodynamic models [56]. The structural source points form an ordered piecewise-linear beam. Each aerodynamic target is associated with the closest beam segment, and modal translation and rotation are linearly interpolated between neighboring structural stations. The aerodynamic modal displacement is therefore given by:

$$\boldsymbol{\Phi}_{A,i} = \boldsymbol{T}_i + \boldsymbol{\theta}_i \times \boldsymbol{r}, \tag{7.21}$$

where $\boldsymbol{r}$ is the offset from the interpolated beam reference position to the aerodynamic target. Because this mapping is local and piecewise linear, it can be used with beam-like structural reference lines that are curved or piecewise straight.

7.4.5 Infinite Beam Spline Mapping

The infinite beam spline provides a smooth interpolation for a straight, line-like structural elastic axis. Beam-spline transfer methods of this type are widely used in aeroelastic structural-to-aerodynamic coupling [56]. The structural source points are ordered along their principal axis. Axial translation and torsional rotation are interpolated as one-dimensional value fields, whereas the two transverse translations are interpolated using smooth beam-type interpolation that also accounts for bending slope. The resulting interpolation reproduces the supplied structural translation and rotation at the structural source stations. The displacement of an off-axis aerodynamic point is then evaluated from

$$\boldsymbol{\Phi}_{A,i} = \boldsymbol{T}_i + \boldsymbol{\theta}_i \times \boldsymbol{r}. \tag{7.22}$$

The infinite beam spline is therefore suitable for a straight structural elastic axis when rotational modal data are available. For a curved or strongly non-collinear structural reference line, the piecewise-linear beam mapping is more appropriate.

7.4.6 Infinite Plate Spline Mapping

The infinite plate spline is a classical surface interpolation that is used for transfer between discrete structural and aerodynamic points [57]. The structural source points are represented in a local two-dimensional surface coordinate system. Each translational modal component is interpolated using the thin-plate radial kernel

$$\varphi(r) = r^2 \ln r \tag{7.23}$$

together with an affine polynomial augmentation. The surface-spline formulation of Harder and Desmarais is based on the small-deflection equation of an infinite plate and provides a smooth interpolant for scattered surface data [57].

In **Arion**, the IPS mapping is translation-only; structural modal rotations are not used explicitly. It is therefore most suitable when a two-dimensional structural source surface adequately represents the aerodynamic-interface deformation.

7.4.7 Choice of Mapping Method

The mapping method should be selected according to the geometry of the structural source data and the structural deformation that must be transferred to the aerodynamic surface. A translation-only RBF or IPS mapping is generally suitable when the structural source surface is close to the aerodynamic interface. Rotation-aware RBF mapping is useful when the aerodynamic surface is offset from the structural reference surface or line and rotational deformation makes a significant contribution. Beam mapping is appropriate for a general one-dimensional structural reference line, whereas the infinite beam spline provides a smooth representation for a straight elastic axis.

Independent of the selected method, the structural source geometry must contain sufficient information to represent the deformation being transferred. Excessive extrapolation, nearly coincident source points, or a mapping dimensionality inconsistent with the structural geometry can produce ill-conditioned interpolation or inaccurate aerodynamic displacement. RBF interpolation provides one established approach for

avoiding the requirement that the structural and aerodynamic meshes share coincident nodes [53, 54], while spline methods have historically served the same role in aeroelastic mode-shape transfer [52, 57].

7.4.8 Aeroelastic Module Mapping Setup

The **Arion** aeroelastic module mapping procedure starts by declaring the individual *Aeroelastic Components* (see Section 9.10.2). Each aeroelastic component is assigned a structural grid and structural modes. Two formats are supported, the NASTRAN format (see Tables 9.74 and 9.75) or the ZAERO FREE FORMAT (see Table 9.76). Selection of the mapping is described in Section 9.10.2.3, Tables 9.79 and 9.80.

7.5 Static Aeroelastic Coupling

Even with a linear structural model, the coupled static aeroelastic problem is iterative because the aerodynamic loads depend on the deformed aerodynamic shape, while the deformation depends on the aerodynamic loads. **Arion** therefore obtains the coupled static solution by iteration between the aerodynamic solution and the modal structural equilibrium. Partitioned solution procedures of this general type are widely used for coupled fluid–structure systems [58]. Each aeroelastic update is composed from the following operations:

1. Aerodynamic forces are evaluated on the current aerodynamic surface;
2. The aerodynamic forces are projected onto the retained structural modes using Equation (7.16);
3. Contributions from all aeroelastic components are summed to form the global generalized aerodynamic-force vector;
4. The static modal equilibrium is evaluated to obtain a target generalized displacement;
5. The generalized coordinates are relaxed toward that target state;

6. The updated modal coordinates are mapped onto the aerodynamic surface;

7. The aerodynamic boundary and interior CFD mesh are deformed;

8. The flow solution continues on the updated geometry.

Because the retained modal stiffness matrix is diagonal, the unrelaxed target for mode i is

$$\xi_i^* = \frac{F_{E,i}}{K_i}. \tag{7.24}$$

To improve robustness of the coupled iteration, the accepted modal state is under-relaxed,

$$\xi_i^{k+1} = \xi_i^k + \alpha \left(\xi_i^* - \xi_i^k \right), \qquad 0 < \alpha \leq 1, \tag{7.25}$$

where α is the user-supplied aeroelastic relaxation factor. Smaller values of α generally improve robustness for strongly coupled cases at the cost of slower convergence. A useful convergence measure is the unrelaxed modal fixed-point defect,

$$\boldsymbol{d}^k = \boldsymbol{\xi}^* - \boldsymbol{\xi}^k. \tag{7.26}$$

Arion evaluates the relative modal defect as

$$\epsilon_\xi = \frac{\left\| \boldsymbol{d}^k \right\|_2}{\max \left(\left\| \boldsymbol{\xi}^* \right\|_2, \left\| \boldsymbol{\xi}^k \right\|_2 \right)}. \tag{7.27}$$

When both denominator norms are zero, the relative defect is defined as zero. Because Equation (7.27) is evaluated before relaxation, the convergence measure does not become artificially small simply because a small relaxation factor is used. The convergence assessment may additionally include absolute modal changes and changes in aerodynamic-surface displacement. Update frequency, minimum and maximum numbers of aeroelastic updates, relaxation, and convergence tolerances are supplied by the user. Details of the various options are found in Section 9.10.1.

7.6 Rigidly-Attached Aerodynamic Components

A rigidly-attached aerodynamic component is useful when an aerodynamic body is assumed rigid but it is mounted on an elastic parent structure. The attachment frame is defined at a point on the elastic parent. Its modal deformation is obtained from the local deformation of the parent structure.

For mode i, let the modal translation and rotation of the attachment frame be $\boldsymbol{T}_{a,i}$ and $\boldsymbol{\theta}_{a,i}$ at reference position $\boldsymbol{x}_a$. The corresponding modal displacement of a point $\boldsymbol{x}$ on the rigid child is therefore:

$$\boldsymbol{\Phi}_{A,i}(\boldsymbol{x}) = \boldsymbol{T}_{a,i} + \boldsymbol{\theta}_{a,i} \times (\boldsymbol{x} - \boldsymbol{x}_a)\,. \tag{7.28}$$

Thus the child follows the translation and orientation of its attachment point and moves as a rigid body within the small-rotation modal approximation. Rigidly-attached components do not introduce additional generalized coordinates. Their aerodynamic loads nevertheless are added to the generalized forces of the parent structure through the same virtual-work projection that is used for elastic components.

Substitution of Equation (7.28) into Equation (7.16) gives

$$F_{E,i}^{\text{child}} = \boldsymbol{T}_{a,i} \cdot \boldsymbol{F} + \boldsymbol{\theta}_{a,i} \cdot \boldsymbol{M}_a, \tag{7.29}$$

where

$$\boldsymbol{F} = \sum_f \boldsymbol{F}_f \tag{7.30}$$

is the resultant aerodynamic force on the rigid child and

$$\boldsymbol{M}_a = \sum_f (\boldsymbol{x}_f - \boldsymbol{x}_a) \times \boldsymbol{F}_f \tag{7.31}$$

is its resultant aerodynamic moment about the attachment frame. Thus both force and moment on the rigid aerodynamic component contribute to the generalized structural loading, while remaining consistent with the same virtual-work projection that is used for the rest of the aerodynamic interface [50].

Rigid attachments may be hierarchical. A rigid child that is attached to another rigid component inherits the deformation of the parent attachment frame, allowing several aerodynamic components to move together while remaining separately identified for aerodynamic-force integration and mesh-boundary treatment. Attachment options are described in Section 9.10.2.5 Tables 9.84 and 9.85.

7.7 Aeroelastic Surface and Volume-Mesh Deformation

The modal solution determines the displacement of the aeroelastic CFD boundary through Equation (7.15). The displacement of the interior CFD mesh is then obtained with the **Arion** elastic mesh-deformation system (see Section 9.12.3 and Listing 9.13). The volume mesh is treated as an artificial isotropic elastic solid whose boundary displacement is prescribed by the aeroelastic surface deformation. The mesh elasticity uses a Poisson ratio $\nu = 0.30$ and a spatially varying relative Young's modulus E.

This propagates the structural boundary deformation smoothly through the computational domain while allowing selected regions of the mesh to be made more resistant to deformation. Elasticity-based mesh deformation with spatially varying stiffness is commonly used to protect sensitive parts of deforming fluid meshes [59].

At selected boundaries, mesh nodes may also be allowed to tangentially slide along the boundary surface during deformation. The displacement normal to the local boundary is constrained, while tangential displacement is permitted. This behavior is enabled using the `-deform.slip` boundary condition option (see Table 9.96).

The mesh deformation may be evaluated relative to the undeformed reference mesh or incrementally from the current mesh configuration. The reference-mesh formulation is the default and avoids accumulation of aeroelastic surface-displacement history in the definition of the structural shape.

The aeroelastic solution therefore requires the elastic mesh-deformation system whenever the CFD mesh itself must follow the structural deformation. Note that in the absence of an elastic mesh-deformation system **Arion** automatically adds such a

system.

7.7.1 Mesh Stiffness Distribution

The artificial elastic modulus may be uniform or varied spatially according to the reference-cell volume or reference wall distance. Only the relative variation of the modulus is relevant to the deformation distribution. For uniform stiffness,

$$E_c = 1. \tag{7.32}$$

For volume-based stiffness, the preliminary modulus of cell c is

$$\widetilde{E}_c = \frac{V_{\max}}{V_c}, \tag{7.33}$$

where V_c is the reference volume of the cell and $V_{\max}$ is the maximum reference-cell volume in the mesh. Smaller cells therefore receive a larger stiffness than larger cells. This reduces deformation of fine cells, which is particularly useful in refined or near-wall regions.

Alternatively, the stiffness may be based on the reference wall distance. Let

$$\hat{d}_c = \frac{d_c}{d_{\max}}, \qquad \hat{d}_0 = \frac{d_0}{d_{\max}}, \tag{7.34}$$

where d_c is the reference wall distance of cell c, $d_{\max}$ is the maximum reference wall distance, and d_0 is the user-specified stiffness offset. The preliminary modulus is then

$$\widetilde{E}_c = \left[\max\left(\frac{1}{E_{\max}}, \frac{\hat{d}_c - \hat{d}_0}{1 - \hat{d}_0} \right) \right]^{-1}, \tag{7.35}$$

where $E_{\max}$ limits the allowable stiffness contrast.

With this formulation, cells at the maximum wall distance have $\widetilde{E}_c = 1$, while cells closer to the wall become progressively stiffer. The offset d_0 shifts the stiffened region away from the wall and can therefore be used to maintain a relatively stiff layer around moving boundaries.

For either spatial stiffness model, an additional exponent p may be applied, followed by the maximum-stiffness limit,

$$E_c = \min\left(\widetilde{E}_c^{\,p}, E_{\max}\right). \tag{7.36}$$

For $p > 1$, differences in stiffness are amplified, concentrating deformation more strongly in the relatively compliant regions of the mesh. For $0 < p < 1$, the stiffness variation is reduced. With $p = 1$, the basic stiffness distribution is used directly.

All cell volumes and wall distances used in these stiffness definitions are evaluated from the reference mesh. The stiffness field therefore does not change as a consequence of the mesh deformation itself. The specific formulation and implementation that are used by **Arion** are described in ISCFDC Report 2025-15 [49].

7.7.2 Mesh-Deformation Solution Accuracy

At each aeroelastic update, the prescribed aerodynamic-surface displacement must be propagated accurately into the CFD volume mesh. The elastic mesh-deformation equations are therefore solved as a separate linear system. Accurate convergence of this system is important because the resulting mesh defines the geometry on which the subsequent CFD solution is performed. An insufficiently converged mesh-deformation solution may produce an inaccurate or distorted mesh even when the aeroelastic modal solution itself is correct. The resulting mesh error can then cause degradation or divergence of the CFD solution. For aeroelastic calculations, a relative linear-solver tolerance of approximately 10^{-9}–10^{-10} is recommended.

If a previously stable CFD solution becomes unstable or diverges immediately following an aeroelastic update, the convergence of the elastic mesh-deformation system and the resulting mesh quality should therefore be examined as part of the diagnosis.

7.8 Restart and Continuation

A static aeroelastic restart preserves the accepted generalized modal coordinates together with structural information required to verify that the saved state is compatible

with the current aeroelastic model. The saved aeroelastic state therefore contains the retained mode identification and ordering, the generalized-displacement vector, and structural consistency information for the aeroelastic components.

The aerodynamic mesh itself is not a part of the aeroelastic modal state. Consequently, the saved state is independent of the CFD mesh partitioning and surface ordering, provided that the retained structural modal basis remains compatible with the saved state.

On restart, the saved generalized modal coordinates are first restored. The corresponding aeroelastic displacement is then reconstructed on the current aerodynamic surface using the current structural-to-aerodynamic mapping. The CFD mesh is subsequently deformed to the restored aeroelastic configuration before continuation of the coupled solution. This procedure reconstructs the physical aeroelastic shape from the saved modal state rather than storing the deformed CFD mesh as part of the aeroelastic state.

The aeroelastic state can therefore be reused after reconstruction of the aerodynamic mapping, for example following a change in CFD partitioning or surface ordering, provided that the structural modal model remains compatible. After the restored configuration has been established, the static aeroelastic iteration continues normally from the loaded generalized coordinates. Subsequent aerodynamic loads are projected onto the retained modal basis, the static modal equilibrium is updated, and any resulting change in generalized displacement is transferred to the aerodynamic surface and CFD mesh in the same manner as during the original calculation.

7.9 Assumptions and Range of Applicability

The present formulation is a static modal aeroelastic method. Its principal assumptions are:

- structural deformation is represented by linear superposition of a finite set of elastic modes;
- the retained modal basis adequately represents the structural deformation pro-

duced by the aerodynamic loading;

- retained elastic modes have positive modal stiffness and rigid-body modes are excluded from the static elastic equilibrium;
- modal rotations used by rotation-aware mappings and rigid attachments are small-rotation vectors;
- the structural modal basis and its normalization remain fixed during the aeroelastic solution;
- the aerodynamic and structural discretization may differ, with their interaction represented by the selected mapping method; and
- the static coupled state is obtained iteratively, so adequate aerodynamic and aeroelastic convergence are required for a converged equilibrium shape.

The formulation is suitable for static deformation of wings, fins, fuselages, stores, pylons, and complete configurations when the retained modal basis captures the structural response of interest. Time-accurate structural dynamics, modal velocity and acceleration, structural damping, and general finite-amplitude six-degree-of-freedom rigid-body motion are outside the present static formulation.

Chapter 8

Parallelization

The **Arion** flow solver is designed to work in a distributed memory architecture using the MPI interface. The design of the code distinguishes between managing a simulation and solving the flow field. Within the distributed MPI universe, the first rank is responsible for managing the simulation and henceforth named 'manager rank'. The rest of the ranks are responsible for the flow solution and are named 'worker ranks'. The manager rank responsibility starts with the input analysis, and continuing with reading the initial geometric problem (the grid files), splitting the computational domain into parts and sending those parts to the worker ranks. Therefore, the worker ranks know only part of the computational domain, and pass boundary data among themselves. The manager rank is responsible for the the assembly of the 'restart' files, for timing all the worker threads, and for log output. Since there is a distinction between the manager rank and the worker ranks, one must have at least two ranks running, even if they reside on a single shared memory machine.



Chapter 9

Input File, Run Preparation, and Execution

9.1 Preface

The **Arion** code is driven through the command line with optionally additional input files having a certain syntax. An input file may be constructed using a simple text editor or using EPPIC (highly recommended). The solver is invoked by typing the MPI command (depending on the MPI version of the actual machine): `mpirun [mpi options]... arion [--f input_file] [command-line arguments]`

The input file is made of groups of directives, each group is marked by two consecutive `-` signs (*i.e.*, `--`). Each group has a series of options, with each option marked with one `-` sign. Within the input file the sign `!` means a remark until the end of the line. A brief help of all the options may be printed to the screen by using the `--h` option. The latest additions to the code may be printed to the screen using the `--new` option.

9.1.1 Scripting Language

The input file is a simple ASCII file with certain important rules. This chapter contains a detailed description of all the available command-line options. The syntax of each of the input options, or input file lines, follows the proceeding rules:

- The "# " symbol means that there is a comment until the next option. One may use as many comment lines as necessary.
- Each group of directives starts with two consecutive - signs, *i.e.*, -- followed by the group name and a series of options.
- Each option is marked with one - sign.
- An option may have no parameters, one parameter, or a few parameters.
- Parameters may be assigned values.

The proceeding description of the input options, or input file lines, makes use of the following symbols:

- The "$" symbol signifies a string input.
- The "#" symbol signifies a numerical input.
- The "|" symbol signifies a choice selection.
- The "..." symbol signifies an option that can be repeated.
- Input entries that are enclosed by curly brackets, { }, are required.
- Input entries that are enclosed by square brackets, [ ], are optional.

9.1.2 Basic Input File: Examples

9.1.2.1 Single Component Gas Example

Listing 9.1 contains a simple, single component gas, basic input file for a simple simulation of the ideal-gas flow about a two-dimensional airfoil.

```
--equation.of.state
   -name air
   -type eos.ideal.gas
   -ref.p 101325
   -ref.T 288

--bc
   -name RIE_FREE
   -type riemann

--bc
   -name IWALL
   -type impermeable.wall

--bc
   -name XZSYM
   -type 2d

--system
   -name ideal.gas.mf.inviscid
   -type ideal.gas.mf.inviscid
   -time.step 5 100 cfl.exponential 250
   -cell.gradient green.gauss.node
   -limiter mlp.2d
```

```
   -log convergence cfl residual log.residual

--solve
   -time.march            implicit.rk3.R.pgs
   -convection.flux       hllc.roe
   -convection.jacobian      hllc.roe
   -sweeps 4
   -spatial.order 2

   -iterations  2000
   -save      100
   -save.path ./save/

   -eos air
   -p       101325
   -T       288
   -Mach    0.8
   -alpha   1.25
   -beta    0
   -log convergence iter

--log
   -name convergence
   -prefix ./logs/convergence
   -per iter
   -screen

--plot
   -name naca0012
   -prefix ./plots/plot
   -interval 100
```

```
--uns
   -name naca0012
   -prefix naca0012/naca0012
   -log convergence cl cd

--table
   -name P_table
   -prefix ./tables/table
   -interval 10
   -plane.bc 0 0 0 0 0.5 0 IWALL
```

Listing 9.1: Example of a simple **Arion** input file (single-component gas)

9.1.2.2 Multi Component Gas Example

Listing 9.2 contains a multi component gas input file.

```
--parallel
    -all.ranks
    -threads 1
    -parts 1

--equation.of.state
    -name neq
    -type eos.mcg
    -ref.p 11.51
    -ref.T 234.2

--plot
    -name plot
    -prefix ./restart/plot
```

```
    -interval 100
    -overwrite
    -cgns

--log
    -name convergence
    -prefix ./logs/convergence
    -per iter
    -screen

--table
    -name table
    -prefix ./tables/table
    -interval 1000
    -plane.bc 0 -1e-4 0 0 -1 0 Wall

--system
    -name meanflow
    -type mcg.sst
    -time.step 0.1 5000 cfl.exponential 2
    -cell.gradient green.gauss.cell
    -face.gradient thin.layer
    -limiter mlp.3d
    -mlp.k 1
    -relax 0.4
    -plot plot velocity p T cy residual
    -log convergence cfl residual log.residual
    -table table Cp p qflux cy

--system
    -name trb
```

```
   -type turbulent.kw.sst
   -time.step 0.1 5000 cfl.exponential 2
   -cell.gradient green.gauss.cell
   -face.gradient thin.layer
   -limiter 1st
   -relax 0.4
   -plot plot mt residual
   -log convergence cfl residual log.residual max(mt)

--solve
   -eos neq
   -p 278
   -T 251
   -Mach 5
   -alpha 0
   -beta 0
   -trb.intensity 0.0035
   -trb.mt 0.1
   -time.march implicit.petsc
   -convection.flux ausm.dv
   -convection.jacobian  steger
   -spatial.order 2
   -iterations 30000
   -save 1000
   -time.step.reduction 4
   -save.path ./save/
   -log convergence iter exec.time
   -reference.longitudinal 4.4

--multi.component.gas
   -name air
```

```
   -composition O2=0.23, NO=0, N=0, O=0, N2=0.77 composition.end
   -react.db Dunn_Kang_5sp.dat
   -eqconst arrhenius
   -trns cantera
   -cantera.db gri30.yaml
   -cy.cutoff

--petsc
   -ksp_norm_type unpreconditioned
   -ksp_max_it 100
   -ksp_rtol 2e-2
   -ksp_atol 1e-30
   -ksp_type fgmres
   -ksp_gmres_restart 30
   -pc_type asm
   -sub_pc_type ilu
   -sub_pc_factor_levels 1
   -sub_pc_factor_fill 1
   -sub_ksp_type preonly
   -sys_trb_ksp_norm_type unpreconditioned
   -sys_trb_ksp_max_it 100
   -sys_trb_ksp_rtol 1e-3
   -sys_trb_ksp_atol 1e-30
   -sys_trb_ksp_type fgmres
   -sys_trb_ksp_gmres_restart 30
   -sys_trb_pc_type asm
   -sys_trb_sub_pc_type ilu
   -sys_trb_sub_pc_factor_levels 1
   -sys_trb_sub_pc_factor_fill 1
   -sys_trb_sub_ksp_type preonly
```

```
--uns
    -name IXV
    -prefix.starcd ./grid/IXV_blocks1_mirrored

--bc
    -name Free
    -type inout

--bc
    -name Wall
    -type noslip.wall
    -T 300

--bc
    -name Sym
    -type symmetry

--bc
    -name Ext
    -type inout
```

Listing 9.2: Example of a simple **Arion** input file (multi-component gas)

9.2 Grid Files

Currently, **Arion** supports the Star-CD and CGNS Version 3.4 exports only. A detailed description of the files is given in Chapter 6. Starting from Version 1.31, **Arion** uses the Metis open source library to convert the Star-CD or CGNS export to binary format. In addition, Metis assists in decomposing the computational domain into several partitions for the purpose of efficient parallel computations. The following section describes the use of the Metis library for conversion of the files to binary format. The conversion results in a new set of files with "+32" added to all file extensions, signifying that the integers are 32 bit wide.

The new set of files now contains 5 files. The vertex information has the extension".vrt+32." The cell elements file has the extension ".cel+32." The boundary elements file has the extension ".bnd+32." In addition, two new files are generated, one with the extension "nam+32" and the other with the extension "fac+32." As a result, the grid is read by the code in a fast manner.

9.3 Macros

9.3.1 Def

Def is a macros that starts with the directive `--def` followed by a key and a value. The syntax is as described in Table 9.1. Listing 9.3 shows a detailed example of the `--def` directive usage.

Def		
Syntax	`--def $key $value`	
Description	Sets a macro. Any '${key}' in the options will be replaces by the $value .	
Parameters	$key	Key.
	$value	Value.
Examples	`--def mypath /Users/data/restart/`	

Table 9.1: Def macro

```
--def PATH .
--def PATHPLOT ${PATH}/plot
--def PATHLOG ${PATH}/log
--def PATHTABLE ${PATH}/table
--def PATHSAVE ${PATH}/save
```

Listing 9.3: Example of the def macro usage

9.4 Conversion

The conversion is conducted using the `--star2metis` or `--cgns2metis` directive as described in Table 9.2. Note that the Metis options that are described in Tables 9.2 and 9.3 are taken directly from the Metis runtime help. For further information the user is referred to the complete Metis User's Manual.

The conversion generates a new set of files, replacing the original files that were generated by the grid generation software package (*e.g.*, Pointwise). The actual files are not split but are reordered in a manner that prepares the files for a parallel read that is efficient in terms of domain decomposition.

Conversion		
Syntax	`arion {--star2metis | --cgns2metis}$prefix #ranks $output [metis-options]...`	
Description	Convert the Star-CD or CGNS files to the Arion binary format and use Metis to decompose the computational domain. See Table 9.3 for Metis options.	
Parameters	star2metis	Convert Star-CD files.
	star2cgns	Convert CGNS files.
	$prefix	Prefix of grid files.
	#ranks	Number of partitions.
	$output	Prefix of output files.
Examples	`arion --star2metis Star-CD-file-name 64 New-file-name`	
	`arion --cgns2metis CGNS-file-name 64 New-file-name`	

Table 9.2: Conversion

Metis Options

Metis Option	Description
`-ptype {rb | kway}`	Specifiy the scheme to be used for computing the k-way partitioning: rb - Recursive bisectioning, kway - Direct k-way partitioning [default].
`-ctype {rm | shem}`	Specify the scheme to be used to match the vertices of the graph; rm - Random matching, shem - Sorted heavy-edge matching [default].
`-iptype {grow | random}`	Specify the scheme to be used to compute the initial partitioning, (applies only when -ptype = rb); grow - Grow a bisection using a greedy scheme [default], random - Compute a bisection at random.
`-objtype {cut | vol}`	Specify the objective that the partitioning routines will optimize, (applies only when -ptype = kway); cut - Minimize the edgecut [default]; vol - Minimize the total communication volume.
`-no2hop`	Specify that the coarsening will not perform any 2-hop matchings when the standard matching fails to sufficiently contract the graph.
`-contig`	Specify that the partitioning routines should try to produce (applies only when -ptype = kway) partitions that are contiguous. Note that if the input graph is not connected this option is ignored.
`-minconn`	Specify that the partitioning routines should try to minimize the (applies only when -ptype = kway) maximum degree of the subdomain graph, i.e., the graph in which each partition is a node, and edges connect subdomains with a shared interface.

Table 9.3: Metis options



Metis Options (continued)

Metis Option	Description
`-ufactor #x` (#x is an integer)	Specify the maximum allowed load imbalance among the partitions. A value of x indicates that the allowed load imbalance is 1+x/1000. For -ptype=rb, the load imbalance is measured as the ratio of the 2*max(left,right)/(left+right), where left and right are the sizes of the respective partitions at each bisection. For -ptype=kway, the load imbalance is measured as the ratio of max_i(pwgts[i])/avgpwgt, where pwgts[i] is the weight of the ith partition and avgpwgt is the sum of the total vertex weights divided by the number of partitions requested. For -ptype=rb, the default value is 1 (i.e., load imbalance of 1.001) For -ptype=kway, the default value is 30 (i.e., load imbalance of 1.03).
`-niter #i`	Specify the number of iterations for the refinement algorithms at each stage of the uncoarsening process. Default is 10.
`-ncuts #c`	Specify the number of different partitionings that it will compute The final partitioning is the one that achieves the best edgecut or communication volume. Default is 1.
`-seed #s`	Select the seed of the random number generator.

Table 9.3: Metis options (continued)

9.4.1 Conversion Between Different Orderings

Arion provides the capabilities to start a simulation from a certain ordering and continue the simulation with a different ordering. This is conducted using the `--cnv2cnv` directive as described in Table 9.4.

cnv2cnv		
Syntax	`arion --cnv2cnv $cnv1 $cnv2 $load-prefix $save-prefix {$system1-name}...`	
Description	Converts **Arion** solution files between different orderings; creates new restart files.	
Parameters	$cnv1	Ordering type to convert from.
	$cnv2	Ordering type to convert to.
	$load-prefix	Prefix of input files.
	$save-prefix	Prefix of output files.
	$system1-name	Name of converted system.
Examples	`arion --cnv2cnv bin arion ./newsolution/oldfiles ./newsolution/newfiles ./restart/oldfiles`	

Table 9.4: Conversion between different ordering (cnv2cnv) option

9.5 Parallel Options

The parallel input options section starts with the `--parallel` directive, followed by a series of parallel options. Listing 9.4 shows an example of a parallel section.

```
--parallel
   -all.ranks
   -threads 1
   -parts 1
```

Listing 9.4: Example of parallel section

Cache		
Syntax	`-cache #`	
Description	Size of the cache memory (in bytes).	
Parameters	#	Cache size.
Examples	`-cache 1000`	

Table 9.5: Parallel cache option

Rank		
Syntax	`[{{-rank #rank} | -all.ranks} | {-rank.span #rank_min #rank_max} | {-rank.head #head} | {-rank.tail #tail}} {-threads | -parts | -load | -read.threads} # ]...`	
Description	Define how many OpenMP threads will be open on each rank.	
Parameters	#rank	Rank number.
	-threads	Number of threads for the current rank.
	-all.ranks	Means that all subsequent options hold for all the ranks.
	-rank.span	Means that all subsequent options hold for the span of ranks.
	-rank.head	Means that all subsequent options hold for the starting ranks.
	-rank.tail	Means that all subsequent options hold for the ending ranks.
	-threads	Set the number of solver threads on a specific (or all) rank.
	-read.threads	Use when memory is insufficient to read all the part in parallel (a significant amount of memory is needed to read a zone part, that is deallocated at the end of the read).
	-parts	Number of partitions.
	-load	Changes the load of a rank, by default all ranks have load 1.
	#	Load level for the current rank.
Examples	`-rank 1 -threads 1`	
	`-rank 2 -threads 8`	
	`-all.ranks -threads 48`	
	`-rank 2 -load 4`	

Table 9.6: Parallel rank option

9.6 Equation of State Options

The "Equation of State" input options section starts with the `--equation.of.state` directive, followed by a series of equation.of.state input options. Each section is used to declare a specific fluid with its equation of state and constitutive laws (*e.g.* Sutherland's law). There could be more than one section, one per fluid type. It is first assigned a name to be used by the user. The current version of **Arion** provides the means to solve either a single component perfect gas fluid or a multi-component gas under the assumption of chemical equilibrium or under non-equilibrium conditions. Listing 9.5 shows an example of an equation.of.state section.

```
--equation.of.state
   -name neq
   -type eos.mcg
   -ref.p 11.51
   -ref.T 234.2
```

Listing 9.5: Example of equation.of.state section

Name		
Syntax	`-name $fluid`	
Description	Name of flow medium.	
Parameters	$fluid	Flow medium.
Examples	`-name air`	

Table 9.7: equation.of.state name option

Type	
Syntax	`-type $eos.type.fluid`
Description	Choose the type of equation of state. Available types are listed in Table 9.9
Parameters	$eos.type.fluid Type of equation of state.
Examples	`-type eos.ideal.gas`

Table 9.8: equation.of.state type option

EOS Types	
EOS type	Description
eos.ideal.gas	Perfect gas (see Section 2.2.3 and Equation 2.10).
eos.table	Perfect gas under equilibrium assumption (required for multi-component.gas eqb system).
eos.mcg	Perfect gas under non-equilibrium or frozen chemistry assumption (required for multi-component.gas mcg or mcg.frozen system).
eos.stiffen.gas	Stiffened gas equation of state (A modified perfect gas model that can be used to describe condensed matter like liquids and solids under high pressure; to be used only with systems types that are described in Table 9.134).

Table 9.9: EOS types



R		
Syntax	`-R` #	
Description	Set the specific gas constant (applicable to the eos.ideal.gas type only). Default is 287.22 (air).	
Parameters	#	Specific gas constant.
Examples	`-R 287.22`	

Table 9.10: equation.of.state R option

Gamma		
Syntax	`-gamma #`	
Description	Set the heat capacity ratio (γ, applicable to the eos.ideal.gas type only). Default is 1.4 (air).	
Parameters	#	Heat capacity ratio.
Examples	`-gamma 1.4`	

Table 9.11: equation.of.state gamma option

Ref		
Syntax	`-ref.$ #`	
Description	Set reference properties. These reference values are relevant in particular for the limiters.	
Parameters	-ref.p	Reference pressure. Default is 1.
	-ref.T	Reference temperature. Default is 1.
	-ref.length	Reference length. Default is 1.
	#	Reference value.
Examples	`-ref.p 101325`	
	`-ref.length 0.5`	

Table 9.12: equation.of.state ref option

9.7 Molecular Options

The "Molecular" input options section that is used to evaluate the viscosity and thermal conductivity coefficients, starts with the `--molecular` directive, followed by a series of molecular input options. Each section is used to set the coefficients or to declare a specific type or constants for Sutherland's law. The coefficients can be evaluated in four different ways: The first, called molecular.const, sets the viscosity and thermal conductivity coefficients as constants, the second, called molecular.mcg, uses multi.component.gas molecular relations to calculate the transport properties. See the '`--multi.component.gas`' directive for more information, specifically the '-trns' option (Section 9.9, Tables 9.40, 9.41, and 9.42). The third way, called molecular.sutherland.air.1, uses the direct coefficients for μ and κ (see Equation 2.12), namely, Mu = Cmu1 * $T^{(3/2)}$ / (Cmu2 + T) and Ka = Cka1 * $T^{(3/2)}$ / (Cka2 + T). The fourth way, called molecular.sutherland.air.2, uses Equation 2.13, namely, Mu = Eta0 * (T0 - C0) / (T - C0) * (T / T0)$^{(3/2)}$ and Ka = Cp * Mu / Prandtl.

In there is no molecular section in the input file, the molecular options depend on the type of the relevant flow system. For ideal gas, the default is as if molecular.sutherland.air.1 is selected. For multi component gas, the default is as if molecular.mcg is selected. Listing 9.6 shows an example of the molecular section.

```
--molecular
    -name molecular1
    -type molecular.const
    -viscosity 1.78e-5
    -conductivity 2.48e-2
```

Listing 9.6: Example of the molecular section

Name		
Syntax	`-name $name`	
Description	Set the name of the molecular thermodynamic relations.	
Parameters	$name	Name of molecular thermodynamic relations.
Examples	`-name therm1`	

Table 9.13: Molecular name option

Type		
Syntax	`-type $type`	
Description	Set the type of the molecular thermodynamic relations.	
Parameters	$type	Type of the molecular thermodynamic relations. Available types are:
	molecular.const	Constant viscosity and thermal conductivity coefficients, see Tables 9.15 and 9.16
	molecular.mcg	Use multi.component.gas molecular relations to calculate the transport properties (see '`--multi.component.gas`' directive for more information, specifically at '-trns' option, Section 9.9, Tables 9.40, 9.41, and 9.42)
	molecular.sutherland.air.1	Use Equations 2.12 (see Section 2.2.4)
	molecular.sutherland.air.2	Use Equations 2.13 (see Section 2.2.4)
Examples	`-type molecular.const`	
	`-type molecular.mcg`	
	`-type molecular.sutherland.air.1`	
	`-type molecular.sutherland.air.2`	

Table 9.14: Molecular type option

Viscosity		
Syntax	`-viscosity #mu`	
Description	Set the viscosity coefficient.	
Parameters	#mu	Viscosity coefficient
Examples	`-viscosity 1.78e-5`	

Table 9.15: Molecular viscosity option

Conductivity		
Syntax	`-conductivity #ka`	
Description	Set the thermal conductivity coefficient.	
Parameters	#ka	Thermal conductivity coefficient
Examples	`-conductivity 2.48e-2`	

Table 9.16: Molecular conductivity option

Mu1		
Syntax	`-Mu1 #Cmu1`	
Description	Set the first coefficient for the Sutherland molecular viscosity formula; Default is 1.458e-06 (air, see Equation 2.12 and Table 2.1).	
Parameters	#Cmu1	First coefficient for the Sutherland formula
Examples	`-Mu1 1.458e-06`	

Table 9.17: Molecular Mu1 option

Mu2		
Syntax	`-Mu2 #Cmu2`	
Description	Set the second coefficient for the Sutherland molecular viscosity formula; Default is 110.3 (air, see Equation 2.12 and Table 2.1).	
Parameters	#Cmu2	Second coefficient for the Sutherland formula
Examples	`-Mu2 110.3`	

Table 9.18: Molecular Mu2 option

Ka1		
Syntax	`-Ka1 #Cka1`	
Description	Set the first coefficient for the Sutherland thermal conductivity formula; Default is 2.495e-03 (air, see Equation 2.12 and Table 2.1).	
Parameters	#Cka1	First coefficient for the Sutherland formula
Examples	`-Ka1 2.495e-03`	

Table 9.19: Molecular Ka1 option

Ka2		
Syntax	`-Ka2 #Cka2`	
Description	Set the second coefficient for the Sutherland thermal conductivity formula; Default is 194.0 (air, see Equation 2.12 and Table 2.1).	
Parameters	#Cka2	Second coefficient for the Sutherland formula
Examples	`-Ka2 194.0`	

Table 9.20: Molecular Ka2 option

Eta0	
Syntax	`-Eta0 #Eta0`
Description	Set η_0 for the Sutherland molecular viscosity formula; Default is 1.827×10^{-5} (air, see Equation 2.13 and Table 2.2).
Parameters	#Eta0 Coefficient for the Sutherland formula
Examples	`-Eta0 1.827e-5`

Table 9.21: Molecular Eta0 option

C0	
Syntax	`-C0 #C0`
Description	Set C_0 for the Sutherland molecular viscosity formula; Default is 120 (air, see Equation 2.13 and Table 2.2).
Parameters	#C0 Coefficient for the Sutherland formula
Examples	`-C0 120`

Table 9.22: Molecular C0 option

T0	
Syntax	`-T0 #T0`
Description	Set T_0 for the Sutherland molecular viscosity formula; Default is 291.15 (air, see Equation 2.13 and Table 2.2).
Parameters	#T0 Coefficient for the Sutherland formula
Examples	`-T0 291.15`

Table 9.23: Molecular T0 option

Prandtl	
Syntax	`-Prandtl #Prandtl`
Description	Set the Prandtl number; Default is 0.72 (air, see Equation 2.13 and Table 2.2).
Parameters	#Prandtl Prandtl number
Examples	`-Prandtl 0.72`

Table 9.24: Molecular Prandtl option

9.8 Tabulate Options

The "Tabulate" input options section starts with the `--tabulate` directive, followed by a series of tabulation input options. Tabulation is used in support of the chemical equilibrium model. Creating the eqb tables requires having a `--multi.component.gas` section (see Section 9.9). An example of a simple tabulate options section is found in Listing 9.7 followed by a description of all available options.

```
--tabulate
   -name tabular
   -type create
   -distribution linear
   -key ./tab/
   -pT.chemistry <
   -name p
   -n 400
   -min 20
   -max 2e5 > <
   -name T
   -n 400
   -min 230
   -max 8000 >
```

Listing 9.7: Example of the tabulate section

Name		
Syntax	`-name $name`	
Description	Set the name of the tabulation.	
Parameters	$name	Table name.
Examples	`-name chem`	

Table 9.25: Tabulate name option

Type		
Syntax	`-type $type`	
Description	Set tabulation type.	
Parameters	$type	Tabulation type, options are:
	create	Create new arion tables. Required: `-distribution` (see Table 9.27), `-key` (see Table 9.28), and `-pT.chemistry` as well (see Table 9.29).
	read	Read an existing arion tables. Required: `-key` as well (see Table 9.28).
	read.external	Read existing tabulation from non-arion engine. Required: `-key` as well (see Table 9.28).
Examples	`-type create`	

Table 9.26: Tabulate type option

Distribution		
Syntax	`-distribution $distribution`	
Description	Set the table distribution type.	
Parameters	$distribution	Distribution type, options are:
	linear	Linear table distribution.
	log	Logarithmic table distribution.
	tanh	Hyperbolic tangent table distribution (default).
Examples	`-distribution tanh`	

Table 9.27: Tabulate distribution option

Key		
Syntax	`-key $key`	
Description	Set the table path for saving or loading.	
Parameters	$key	Table path.
Examples	`-key /path/data/tables`	

Table 9.28: Tabulate key option

pT.chemistry		
Syntax	`-pT.chemistry`	
	< -name p -n #n -min #min -max #max >	
	< -name T -n #n -min #min -max #max >	
Description	Create all tables using the given pressure and Temperature data.	
Parameters	-name	Paramemter name (p or T).
	-n #n	Parameter resolution.
	-min #min	Parameter minimum value.
	-max #max	Parameter maximum value.
Examples	`-pT.chemistry`	
	`< -name p -n 400 -min 10000 -max 1e7 >`	
	`< -name T -n 600 -min 250 -max 700 >`	

Table 9.29: Tabulate pT.chemistry option

9.9 Multi Component Gas Options

The "Multi Component Gas" input options section starts with the `--multi.component.gas` directive, followed by a series of multi-component gas input options. An example of a simple multi-component gas options section is found in Listing 9.8 followed by a description of all available options.

```
--multi.component.gas
   -name air
   -composition O2=0.23, NO=0, N=0, O=0, N2=0.77 composition.end
   -react.db Dunn_Kang_5sp.dat
   -eqconst arrhenius
   -trns cantera
   -cantera.db gri30.yaml
   -cy.cutoff
```

Listing 9.8: Example of the multi-component gas section

Name		
Syntax	`-name` $name	
Description	Set the name of the mcg (multi component gas) object.	
Parameters	$name	Multi component gas object name.
Examples	`-name` air	

Table 9.30: Multi component gas name option

Composition	
Syntax	`-composition $composition composition.end`
Description	Set the composition mass fractions.
Parameters	$compostion — The composition in mass fraction. composition.end — End of the composition.
Examples	`-composition O2=0.23, NO=0, N=0, O=0, N2=0.77 composition.end`

Table 9.31: Multi component gas composition option

Composition molar	
Syntax	`-composition molar $composition composition.end`
Description	Set the composition molar fractions.
Parameters	$composition — The composition in mass fraction. composition.end — End of the composition.
Examples	`-composition molar O2=0.21, NO=0, N=0, O=0, N2=0.79 composition.end`

Table 9.32: Multi component gas composition molar option

cy.eq	
Syntax	`-cy.eq`
Description	Equilibrate and override mcg composition based on inlet conditions (for applicable systems). If no composition is supplied in the `--bc` section (see Section 9.11), the global composition is used for the equilibrium calculation. If a different composition is specified, only the species listed there will participate in the equilibrium mixing. Therefore, include all species that should be part of the equilibrium mixture, even if their concentration is set to zero.
Parameters	N/A
Examples	`-cy.eq`

Table 9.33: Multi component gas cy.eq option

cy.cutoff	
Syntax	`-cy.cutoff`
Description	Employ a cutoff to the mass fractions so that they are between 0 and 1.
Parameters	N/A
Examples	`-cy.cutoff`

Table 9.34: Multi component gas cy.cutoff option

Thrm.db	
Syntax	`-thrm.db $thrm.db`
Description	Set the path to the thermal database.
Parameters	$thrm.db Path to the thermal database.
Examples	`-thrm.db /path/data/thermo.inp`

Table 9.35: Multi component gas thrm.db option

Trns.db	
Syntax	`-trns.db $trns.db`
Description	Set the path to the transport database.
Parameters	$trns.db Path to the transport database.
Examples	`-trns.db /path/data/transport.dat`

Table 9.36: Multi component gas trns.db option

React.db	
Syntax	`-react.db $react.db`
Description	Set the path to the reactions database.
Parameters	$react.db Path to the reactions database.
Examples	`-react.db /path/data/reactions.dat`

Table 9.37: Multi component gas react.db option

Cantera.db	
Syntax	`-cantera.db $cantera.db`
Description	Set the path to the Cantera database file.
Parameters	$cantera.db Path to the Cantera database.
Examples	`-cantera.db /path/data/cantera_file.yaml`

Table 9.38: Multi component gas cantera.db option

Thrm		
Syntax	`-thrm $thrm`	
Description	Selects the method of calculating the thermodynamic properties.	
Parameters	$thrm	The method of calculating the thermodynamic properties. Possible options are:
	nasa	Use the NASA database to calculate thermodynamic properties (default).
	cantera	Use the cantera file to calculate thermodynamic properties.
Examples	`-thrm cantera`	
	`-thrm nasa`	

Table 9.39: Multi component gas thrm option

Trns		
Syntax	`-trns $trns`	
Description	Select the method of calculating the transport properties.	
Parameters	$trns	Transport properties, options are:
	wilke	Use Wilke's mixture rule to calculate the transport properties, and a constant Schmidt number for mass diffusivity (default). See Tables 9.41 and 9.42 for additional options.
	cantera	Use the Cantera mixture-averaged model to calculate the transport properties.
	cantera.multi	Use the Cantera multi-component model to calculate the transport properties.
Examples	`-trns cantera`	
	`-trns cantera.multi`	

Table 9.40: Multi component gas trns option

Viscosity		
Syntax	`-viscosity $trns`	
Description	Select the method for calculating the species viscosity (relevant only when wilke model is used, option are nasa or blottner).	
Parameters	$trns	Transport properties, options are:
	nasa	Use NASA 4 term curve fit (default).
	blottner	Use Blottner curve fit.
Examples	`-viscosity nasa`	
	`-viscosity blottner`	

Table 9.41: Multi component gas viscosity option

Conductivity		
Syntax	`-conductivity $trns`	
Description	Select the method for calculating the species thermal conductivity (relevant only when wilke model is used, option are nasa or eucken).	
Parameters	$trns	Transport properties, options are:
	nasa	Use NASA 4 term curve fit (default).
	eucken	Use Eucken formula (with Hirschfelder f_int fix).
Examples	`-conductivity nasa`	
	`-conductivity eucken`	

Table 9.42: Multi component gas conductivity option

Kinetics		
Syntax	`-kinetics $kins`	
Description	Select the method of calculating the reaction rates,.	
Parameters	$kins	Kinetics properties, options are:
	none	Disable chemical reactions.
	laminar	Use laminar model for chmeical reactions (default).
	cantera	Use Cantera to calculate the chemical reactions based on the reaction set in the Cantera file.
Examples	`-kinetics none`	
	`-kinetics cantera`	

Table 9.43: Multi component gas kinetics option

Eqconst		
Syntax	`-eqconst $eqconst`	
Description	Select the method of calculating the reverse reaction rates.	
Parameters	$eqconst	Method of calculating the reverse reaction rates, options are:
	arrhenius	Directly evaluate reverse reaction rates based on coefficients supplied with the -react.db option. If the reaction file contains the reverse rates, the code will automatically use them and override any other option.
	gibbs	calculate equilibrium constants for each reaction with Gibbs' free energy (default when no reverse rates are available).
Examples	`-eqconst gibbs`	

Table 9.44: Multi component gas eqconst option

Eq.engine		
Syntax	`-eq.engine $eq.engine`	
Description	Select the equilibrium solver.	
Parameters	$eq.engine	Equilibrium solver type, options are:
	internal	Use Arion internal solver (default).
	cantera	Use Cantera solver (usually much slower than the Arion internal solver).
Examples	`-eq.engine internal`	

Table 9.45: Multi component gas eq.engine option

Cantera Diagonal Jacobian	
Syntax	`-cantera.diagonal.jacobian`
Description	Use a diagonal approximation for Cantera reaction source terms. The diagonal entries are approximated using the species destruction rate divided by ρY_r.
Parameters	N/A.
Examples	`-cantera.diagonal.jacobian`

Table 9.46: Multi component gas cantera.diagonal.jacobian option

Sc		
Syntax	`-Sc #Sc`	
Description	Set the Schmidt number.	
Parameters	#Sc	Schmidt number (default is 0.7).
Examples	`-Sc 0.8`	

Table 9.47: Multi component gas Sc option

Sc.trb		
Syntax	`-Sc.trb #Sc`	
Description	Set the turbulent Schmidt number.	
Parameters	#Sc	Turbulent Schmidt number (default is 1.0).
Examples	`-Sc.trb 0.8`	

Table 9.48: Multi component gas Sc.trb option

Phase Transition	
Syntax	`-phase.transition <phase1> <phase2> [<phase3> ...]`
Description	Define instantaneous equilibrium transitions between condensed thermodynamic phases of the same material. Currently supported for finite-rate chemistry model only. Requires that the species appear in the composition list.
Parameters	<phase> Species name.
Examples	`-phase.transition AL2O3(a) AL2O3(L)` `-phase.transition SiO2(a-qz) SiO2(b-qz) SiO2(b-crt) SiO2(L)`

Table 9.49: Multi component gas phase transition option

9.10 Aeroelasticity

The aeroelasticity module in **Arion** is used to define and control the aeroelastic solution and to associate structural models with the corresponding CFD wall boundaries. The aeroelastic surface displacement is transferred to the volume grid using the elastic mesh-deformation system, which must be enabled when mesh deformation is required. The elastic mesh-deformation directives are described separately. This section describes the aeroelasticity options and aeroelastic components.

9.10.1 Aeroelasticity Options

The "Aeroelasticity Options" input section starts with the `--aeroelasticity.options` directive, followed by a series of input options. It is used to define global aeroelasticity options that control the aeroelastic solution. An example of a simple aeroelasticity options section is found in Listing 9.9 followed by a description of all available options.

```
--aeroelasticity.options
   -name ae.opt
   -start.from 1000
   -update.every 100
   -relax 1
   -convergence.rel.tol 1e-5
   -displacement.rel.tol 1e-5
   -plot plot_modes modes scale 10
   -log log_ae modal.solution generalized.force convergence
      max.displacement
```

Listing 9.9: Example of aeroelasticity options section

Relax	
Syntax	`-relax #value`
Description	Set the modal relaxation factor applied to each aeroelastic update.
Parameters	#value — Relaxation factor.
Examples	`-relax 0.9`

Table 9.50: Aeroelastic relax option

Start From	
Syntax	`-start.from #iteration`
Description	Set the solver iteration from which aeroelastic updates are enabled.
Parameters	#iteration — Solver iteration number.
Examples	`-start.from 1000`

Table 9.51: Aeroelastic start from option

Update Every	
Syntax	`-update.every #iterations`
Description	Set the number of solver iterations between consecutive aeroelastic updates.
Parameters	#iterations — Number of iterations between aeroelastic updates.
Examples	`-update.every 500`

Table 9.52: Aeroelastic update every option

Convergence Relative Tolerance	
Syntax	`-convergence.rel.tol #value`
Description	Set the relative tolerance for convergence of the modal fixed-point solution.
Parameters	#value Relative tolerance.
Examples	`-convergence.rel.tol 0.01`

Table 9.53: Aeroelastic convergence relative tolerance option

Convergence Absolute Tolerance	
Syntax	`-convergence.abs.tol #value`
Description	Set the absolute tolerance for convergence of the modal fixed-point solution.
Parameters	#value Absolute tolerance.
Examples	`-convergence.abs.tol 1e-3`

Table 9.54: Aeroelastic convergence absolute tolerance option

Displacement Relative Tolerance	
Syntax	`-displacement.rel.tol #value`
Description	Set the relative tolerance for convergence of the aeroelastic displacement.
Parameters	#value Relative tolerance.
Examples	`-displacement.rel.tol 0.01`

Table 9.55: Aeroelastic displacement relative tolerance option

Displacement Absolute Tolerance	
Syntax	`-displacement.abs.tol #value`
Description	Set the absolute tolerance for convergence of the aeroelastic displacement.
Parameters	#value Absolute tolerance.
Examples	`-displacement.abs.tol 1e-2`

Table 9.56: Aeroelastic displacement absolute tolerance option

Minimum Updates		
Syntax	`-min.updates #number`	
Description	Set the minimum number of aeroelastic updates before convergence is accepted.	
Parameters	#number	Number of minimum aeroelastic updates.
Examples	`-min.updates 5`	

Table 9.57: Aeroelastic minimum updates option

Maximum Updates		
Syntax	`-max.updates #number`	
Description	Set the maximum number of aeroelastic updates.	
Parameters	#number	Maximum number of aeroelastic updates.
Examples	`-max.updates 5`	

Table 9.58: Aeroelastic maximum updates option

Elastic Modes		
Syntax	`-modes.elastic #number`	
Description	Set the number of elastic modes to retain. Rigid-body modes are automatically excluded. A value of 0 retains all non-rigid modes.	
Parameters	#number	Number of elastic modes to retain; 0 selects all non-rigid modes.
Examples	`-modes.elastic 15`	

Table 9.59: Aeroelastic elastic modes option

Modes		
Syntax	`-modes #mode_ids`	
Description	Explicitly select physical elastic mode IDs and their retained order. This is an alternative to selecting modes by count with `-modes.elastic`.	
Parameters	#mode_ids	List of physical elastic mode IDs in the required retained order.
Examples	`-modes 3 5 9 6`	

Table 9.60: Aeroelastic modes option

Modes Rigid Tolerance		
Syntax	`-modes.rigid.tol #value`	
Description	Set the eigenvalue tolerance used to classify rigid-body modes. A mode with an eigenvalue less than or equal to this tolerance is treated as a rigid-body mode. For ZAERO FREE FORMAT input, eigenvalue = FREQ^2, where FREQ is the angular frequency in rad/s.	
Parameters	#value	Rigid body mode eigenvalue tolerance.
Examples	`-modes.rigid.tol 1e-5`	

Table 9.61: Aeroelastic modes rigid tolerance option



Modes Normalization		
Syntax	`-modes.normalization {mass.normalized | modal.mass #mass [#mass ...]}`	
Description	Set the modal normalization convention for PATRAN modal input. Valid values are mass.normalized and modal.mass. For mass.normalized, the structural modes are assumed to be mass-normalized. For modal.mass, the diagonal generalized modal masses must be specified following the normalization type, with one positive mass value for each retained elastic mode and in retained-mode order. This option is not required for ZAERO free format input, since the generalized modal masses are provided in the ZAERO file.	
Parameters	mass.normalized	Assume mass-normalized structural modes (default).
	modal.mass	Use explicitly supplied generalized modal masses for PATRAN modal input.
	#mass	Positive diagonal generalized modal mass corresponding to a retained elastic mode.
Examples	`-modes.normalization mass.normalized`	
	`-modes.normalization modal.mass 1.0 4.0 5.6`	

Table 9.62: Aeroelastic modes normalization option

Modes Eigenvalue Relative Tolerance		
Syntax	`-modes.eigenvalue.rel.tol #value`	
Description	Set the relative tolerance used when comparing retained modal eigenvalues, generalized masses and modal stiffnesses between elastic components.	
Parameters	#value	Relative consistency tolerance for modal properties between elastic components.
Examples	`-modes.eigenvalue.rel.tol 1e-10`	

Table 9.63: Aeroelastic modes eigenvalue relative tolerance option



9.10.1.1 Radial Basis Function Options

The following options control global numerical tolerances and diagnostics used by the radial basis function (RBF) mapping. A theoretical background of RBF mapping can be found in Section 7.4.2. The following options are intended for expert users.

RBF Singular Tolerance	
Syntax	`-rbf.singular.tol #value`
Description	Set the RBF singularity tolerance. (for use by the expert user).
Parameters	#value Singularity tolerance level.
Examples	`-rbf.singular.tol 1e-5`

Table 9.64: Aeroelastic RBF singular tolerance option

RBF Geometry Dimension Tolerance	
Syntax	`-rbf.geometry.dimension.tol #value`
Description	Set the tolerance used when determining the effective geometric dimension of the RBF source-point distribution. (for use by the expert user).
Parameters	#value Geometry-dimension tolerance level.
Examples	`-rbf.geometry.dimension.tol 1e-5`

Table 9.65: Aeroelastic RBF geometry dimension tolerance option

RBF Projection Duplicate Tolerance	
Syntax	`-rbf.projection.duplicate.tol #value`
Description	Set the tolerance used to identify duplicate source points after projection to a reduced-dimensional RBF coordinate representation. (for use by the expert user).
Parameters	#value — Projected-source duplicate-point tolerance level.
Examples	`-rbf.projection.duplicate.tol 1e-5`

Table 9.66: Aeroelastic RBF projection duplicate tolerance option

RBF Enable Mapping Warnings	
Syntax	`-rbf.enable.mapping.warnings`
Description	Enable additional RBF mapping warnings. (for use by the expert user).
Parameters	N/A.
Examples	`-rbf.enable.mapping.warnings`

Table 9.67: Aeroelastic RBF enable mapping warnings option

Export		
Syntax	`-export $file`	
Description	Export the aeroelastic state to the specified file, in addition to the state saved automatically with the CFD solution. The full path, including the file name, must be specified.	
Parameters	$file	Full path and file name of the aeroelastic state file.
Examples	`-export /Users/mypath/aeroelastic.state`	

Table 9.68: Aeroelastic export option

Import		
Syntax	`-import $file`	
Description	Import the aeroelastic state from the specified file, overriding the state loaded automatically with the CFD solution. The full path, including the file name, must be specified.	
Parameters	$file	Full path and file name of the aeroelastic state file.
Examples	`-import /Users/mypath/aeroelastic.state`	

Table 9.69: Aeroelastic import option

Plot		
Syntax	`-plot $plot [plot-options] ...`	
Description	Add aeroelastic quantities to a plot file.	
Parameters	$plot	Plot name.
	modes	Add structural mode shapes.
	scale #scale	Optional scale factor for mode-shape visualization.
Examples	`-plot aeroelastic modes`	
	`-plot aeroelastic modes scale 10`	

Table 9.70: Aeroelastic plot option

Log		
Syntax	`-log $log [log-options] ...`	
Description	Report aeroelastic quantities in a log file.	
Parameters	$log	Log name.
	modal.solution	Add the modal solution.
	generalized.force	Add the generalized forces.
Examples	`-log aeroelastic modal.solution`	
	`-log aeroelastic modal.solution generalized.force`	

Table 9.71: Aeroelastic log option



9.10.2 Aeroelasticity Component

The "Aeroelasticity Component" input section starts with the `--aeroelasticity.component` directive, followed by a series of input options. Each aeroelastic component of a certain geometry has one such directive. For example, a simple wing body configuration will have one directive for the body and one directive for the wing. Each component is associated with one or more CFD wall boundary condition(s). An elastic component reads structural modal data and maps it to the CFD surface. A rigidly-attached component inherits its rigid motion from the modal deformation of another aeroelastic component (the parent). The parent component must be defined prior to the rigidly-attached component. An example of a simple aeroelastic component options section is found in Listing 9.10 followed by a description of all available options.

```
--aeroelasticity.component
    -name Fus-Aero
    -nastran.grid.file ../data/grid_s_fus.dat
    -nastran.modes.file ../data/modes_s_fus.pch
    -bc WALL_FUS
    -mapping bs
```

Listing 9.10: Example of the aeroelastic component section

Name		
Syntax	`-name $name`	
Description	Set the component name.	
Parameters	$name	Aeroelastic component name.
Examples	`-name fuselage` `-name wing`	

Table 9.72: Aeroelastic component name option

BC		
Syntax	`-bc $bc_name [$bc_name ...]`	
Description	Set the name(s) of the CFD wall boundary condition(s) associated with the component.	
Parameters	$bc_name	Name of wall boundary condition.
Examples	`-bc fuselage_f fuselage_c fuselage_a` `-bc wing`	

Table 9.73: Aeroelastic component bc option

9.10.2.1 Structural Input Options

An elastic component may use either NASTRAN input or ZAERO FREE FORMAT input. The two structural input formats are alternatives for a given component.

NASTRAN Grid File		
Syntax	`-nastran.grid.file $file`	
Description	Set the NASTRAN grid file containing the structural reference geometry grid points.	
Parameters	$file	Name of grid file containing the structural reference geometry component grid points.
Examples	`-nastran.grid.file fuselage_grid_file`	

Table 9.74: Aeroelastic component nastran.grid.file option

NASTRAN Modes File		
Syntax	`-nastran.modes.file $file`	
Description	Set the NASTRAN modes file containing the structural reference geometry modes.	
Parameters	$file	Name of PCH file containing the structural reference geometry component modes.
Examples	`-nastran.modes.file fuselage_modes_file`	

Table 9.75: Aeroelastic component nastran.modes.file option

ZAERO Free Format File		
Syntax	`-zaero.free.format.file $file`	
Description	Set the ZAERO FREE FORMAT file containing structural coordinates, modes and generalized masses.	
Parameters	$file	Name of the ZAERO FREE FORMAT file.
Examples	`-zaero.free.format.file zaero_file`	

Table 9.76: Aeroelastic component zaero.free.format.file option

Points For Spline File		
Syntax	`-points.for.spline.file $file`	
Description	Optional file containing the structural grid points to use as mapping source points. If omitted, all structural grid points are used.	
Parameters	$file	Name of file containing the structural grid points to use as mapping source points.
Examples	`-points.for.spline.file spline_points_file`	

Table 9.77: Aeroelastic component points.for.spline.file option

9.10.2.2 Structural Transformation Options

Structural transformations provide an optional ordered transformation of the structural model into the CFD/global reference frame. Transformation operations are applied in the order in which they are specified, and changing the order can change the resulting transformation. The option may be specified multiple times for the same elastic component.

Note that structural transformations are applied before structural-to-CFD mapping and are not valid for rigid-attached components.

Structural Transform		
Syntax	`-structural.transform {rotate.axis.angle #axis_x #axis_y #axis_z #angle_deg #pivot_x #pivot_y #pivot_z | translate #dx #dy #dz }`	
Description	Rotate the structural model about an axis passing through the specified pivot point. The axis and pivot are specified in the current CFD/global reference frame. The rotation angle is specified in degrees and follows the right-hand rule. The rotation axis does not need to be normalized. Alternatively, translate the structural model by the specified vector in the current CFD/global reference frame.	
Parameters	rotate.axis.angle	Rotate structural model.
	#axis_x #axis_y #axis_z	Rotation axis.
	#angle_deg	Rotation angle in degrees.
	#pivot_x #pivot_y #pivot_z	Pivot point through which the rotation axis passes.
	translate	Translate structural model.
	#dx #dy #dz	Translation vector.
Examples	`-structural.transform rotate.axis.angle 0 0 1 9 0 0 0 0`	
	`-structural.transform translate 10 0 0`	

Table 9.78: Aeroelastic component structural transform option

9.10.2.3 Mapping Options

Mapping		
Syntax	`-mapping $type`	
Description	Set the structural-to-CFD mapping method type.	
Parameters	$type	The structural-to-CFD mapping method type. See Table 9.80 for mapping types.
Examples	`-mapping rbf.no.rot`	
	`-mapping ips`	

Table 9.79: Aeroelastic component mapping option

Mapping Types	
Mapping type	Description
rbf.no.rot	RBF mapping using translational modal data only
rbf.with.rot	Rotation-aware RBF mapping using translational and rotational modal data (default)
beam.1d	One-dimensional beam mapping
bs	Infinite beam spline (IBS) mapping
ips	Infinite plate spline (IPS) mapping

Table 9.80: Mapping types

9.10.2.4 Radial Basis Function Component Options

The following options control the radial basis function (RBF) mapping for the current aeroelastic component. These options are intended for expert users.

RBF Kernel		
Syntax	`-rbf.kernel $kernel`	
Description	Set the RBF kernel for the current component (for use by the expert user).	
Parameters	$kernel	Kernel type. Valid values are: polyharmonic.r3, polyharmonic.r1, r2.log.r.
Examples	`-rbf.kernel polyharmonic.r3`	

Table 9.81: Aeroelastic component RBF kernel option

RBF Coordinates		
Syntax	`-rbf.coordinates $coordinates`	
Description	Set the RBF coordinate representation for the current component (for use by the expert user).	
Parameters	$coordinates	Coordinate representation. Valid values are: auto, global.3d, global.xy, local.surface.2d, principal.axis.1d.
Examples	`-rbf.coordinates local.surface.2d`	

Table 9.82: Aeroelastic component RBF coordinates option

RBF Polynomial		
Syntax	`-rbf.polynomial $polynomial`	
Description	Set the RBF polynomial augmentation for the current component (for use by the expert user).	
Parameters	$polynomial	Polynomial type. Valid values are: auto, constant, linear.1d, linear.2d, linear.3d.
Examples	`-rbf.polynomial linear.2d`	

Table 9.83: Aeroelastic component RBF polynomial option

9.10.2.5 Rigid Attachment Options

Specifying `-attach.to` defines the current component as rigidly attached to another aeroelastic component.

Attach To		
Syntax	`-attach.to $component_name`	
Description	Set the name of the parent aeroelastic component. The parent component must be defined before the attached component.	
Parameters	$component_name	Parent component name.
Examples	`-attach.to fuselage` `-attach.to wing`	

Table 9.84: Aeroelastic component attach to option

When the parent component is itself rigid-attached, `-attach.by` is omitted and the child inherits the resolved attachment frame of its parent.

Attach By		
Syntax	`-attach.by {grid.id #grid_id | point #x #y #z}`	
Description	Attach the component to an exact structural grid point of the parent elastic component or at the specified point in the CFD/global reference coordinate system.	
Parameters	#grid_id	ID of a structural grid point in the parent elastic component.
	#x, #y, #z	Coordinates of the specified point in the CFD/global reference coordinate system.
Examples	`-attach.by grid.id 1001`	
	`-attach.by point 0.0 0.0 0.0`	

Table 9.85: Aeroelastic component attach by option

9.11 BC Options

Boundary conditions are set for each boundary cell, based on the name that has been assigned by the user (see Chapter 6). For each name, representing a group of boundary cells, the boundary conditions are set in two steps. First, assign the name, and second, assign the type. Each group starts with the `--bc` directive, followed by the name and type. An example of a simple bc options section is found in Listing 9.11 followed by a description of all available options.

```
--bc
    -name Wall
    -type noslip.wall
    -T 300
```

Listing 9.11: Example of the bc section

Name		
Syntax	`-name $name`	
Description	Name of the boundary as set in Pointwise.	
Parameters	$name	Name of boundary.
Examples	`-name WALL`	

Table 9.86: BC name option

Type	
Syntax	`-type $type`
Description	Type of boundary condition. See Chapter 5 and Table 9.88 for available types.
Parameters	$type Type of boundary condition.
Examples	`-type noslip.wall`

Table 9.87: BC type option

Boundary Condition Types

BC type	Description (see Chapter 5 for details)
impermeable.wall	Impermeable wall.
noslip.wall	No slip wall.
nichols.wall.function	Nichols and Nelson wall function (see Section 5.2.1).
automatic.wall.function	same as nichols.wall.function.
riemann.inlet	Riemann type inlet boundary conditions.
riemann.outlet	Riemann type outlet boundary conditions.
riemann	Riemann type outer boundary conditions (automatic detection between riemann.inlet, riemann.outlet, fixed, and extrap).
turkel.inlet	Turkel type inlet boundary conditions (for subsonic flow only).
turkel.outlet	Turkel type outlet boundary conditions (for subsonic flow only).
turkel	Turkel type outer boundary conditions (automatic detection between turkel.inlet, turkel.outlet, fixed, and extap).
inlet	Inlet type outer boundary conditions.
outlet	Pressure outlet type outer boundary conditions.
inout	Inout outer boundary conditions (automatic detection between inlet and outlet).
extrap	Zero order extrapolation.
fixed	Fixed boundary conditions.
stagnation.inlet	Stagnation inlet.
mass.flow.inlet	Mass flow inlet.
symmetry	Symmetry type conditions.
2d	Two dimensional domain conditions.
axis	Axis for axisymmetric model.

Table 9.88: Boundary condition types



Mass.flow	
Syntax	`-mass.flow #mdot`
Description	Specifies the target mass flow rate [kg/s]. For an "outlet" boundary condition, the boundary pressure is adjusted iteratively to achieve the specified mass flow rate (see Section 5.3.6.1). The `-relax` (see Table 9.90) and `-delay` (see Table 9.91) options can be used with this boundary condition. For a "mass.flow.inlet" boundary condition, the stagnation temperature must also be specified (see Section 5.3.8). The stagnation temperature is specified using `-T` (see Table 9.109).
Parameters	#mdot Target mass flow rate [kg/s].
Examples	`-mass.flow 10`

Table 9.89: BC mass.flow option

Relax	
Syntax	`-relax #val`
Description	A relaxation parameter for the mass flow rate pressure adjustment (see Table 9.89). Use values smaller than 1 if the pressure changes too rapidly (see Section 5.3.6.1).
Parameters	#val Relaxation parameter.
Examples	`-relax 0.5`

Table 9.90: BC relax option

Delay		
Syntax	`-delay #val`	
Description	A delay mechanism for targeted mass flow rate, namely, activate the pressure adjustment mechanism after #val iterations for steady-state simulations or #val steps for unsteady simulations (see Table 9.89).	
Parameters	#val	Delay parameter.
Examples	`-delay 3000`	

Table 9.91: BC delay option

Reference.pressure		
Syntax	`-reference.pressure #ref`	
Description	Set the mean-flow reference pressure (for force calculation), used to override free-stream reference pressure.	
Parameters	#ref	Reference pressure.
Examples	`-reference.pressure 101325`	

Table 9.92: BC reference.pressure option

Composition	
Syntax	`-composition $composition composition.end`
Description	Override the composition mass fractions (for applicable systems, can specify only species fractions that are not zero).
Parameters	$compostion — The composition in mass fraction. composition.end — End of the composition.
Examples	`-composition O2=0.23, N2=0.77 composition.end`

Table 9.93: BC gas composition option

Composition Molar	
Syntax	`-composition molar $composition composition.end`
Description	Override the composition molar fractions (for applicable systems, can specify only species fractions that are not zero).
Parameters	$compostion — The composition in mass fraction. composition.end — End of the composition.
Examples	`-composition molar O2=0.21, N2=0.79 composition.end`

Table 9.94: BC gas composition molar option

cy.eq	
Syntax	`-cy.eq`
Description	Equilibrate and override mcg composition based on inlet conditions (for applicable systems).
Parameters	N/A
Examples	`-cy.eq`

Table 9.95: BC gas cy.eq option

deform.slip	
Syntax	`-deform.slip`
Description	Allow boundary nodes to slide tangentially along the boundary during mesh deformation. The local boundary normal is used to constrain the node displacement, preventing motion normal to the boundary.
Parameters	N/A
Examples	`-deform.slip`

Table 9.96: BC aeroelastic deformation slip option

9.11.1 BC Log Options

Log	
Syntax	`-log $log [log-options] ...`
Description	Report in log file. See Table 9.98 for log options list.
Parameters	$log Log name.
Examples	`-log coefficients cl cd`

Table 9.97: Boundary conditions log option

BC Log Options	
Log type	Description
wet.surface	Wet surface area.
p.center.sum	Pressure center sum.
p.center.x	Center of pressure x coordinate.
p.center.y	Center of pressure x coordinate.
p.center.z	Center of pressure x coordinate.
p.center.xFy	Center of pressure x coordinate.
p.center.xFz	Center of pressure x coordinate.
p.center.yFx	Pressure center sum(y * dFx) / Fx coordinate.
p.center.yFz	Pressure center sum(y * dFz) / Fz coordinate.
p.center.zFx	Pressure center sum(z * dFx) / Fx coordinate.
p.center.zFy	Pressure center sum(z * dFy) / Fy coordinate .
p.average	Average pressure on the boundary.



fx	X coordinate direction force.
fy	Y coordinate direction force.
fz	Z coordinate direction force.
fx.pressure	Force in x direction due to pressure.
fy.pressure	Force in y direction due to pressure.
fz.pressure	Force in z direction due to pressure.
fx.friction	Force in x direction due to friction.
fy.friction	Force in y direction due to friction.
fz.friction	Force in z direction due to friction.
lift	Lift force.
drag	Drag force.
mx	Moment about X coordinate direction.
my	Moment about Y coordinate direction.
mz	Moment about Z coordinate direction.
cfx	X coordinate direction force coefficient.
cfy	Y coordinate direction force coefficient.
cfz	Z coordinate direction force coefficient.
cmx	Moment coefficient about X coordinate direction.
cmy	Moment coefficient about Y coordinate direction.
cmz	Moment coefficient about Z coordinate direction.
cl	Lift coefficient.
cd	Drag coefficient.
mass.flow	Mass flow rate.

Table 9.98: BC log options

9.11.2 Wall Distance Calculations

The following directives pertain to wall type boundary conditions. Normally, one would like to calculate wall distance based on whether a specific boundary has been assigned wall conditions. However, sometimes it is required to override this default. Note that the wall distance is used only for turbulence models that require wall distance. The following two directives are exactly for that. They should be used along with a wall boundary condition.

Wall.distance	
Syntax	`-wall.distance`
Description	Explicitly set wall distance search for a specific boundary. Default: for all walls.
Parameters	N/A
Examples	`-wall.distance`

Table 9.99: BC wall.distance option

No.wall.distance	
Syntax	`-no.wall.distance`
Description	Explicitly disable wall distance search for a specific boundary. Default: for all boundaries other than walls.
Parameters	N/A
Examples	`-no.wall.distance`

Table 9.100: BC no.wall.distance option

9.11.3 Free-Stream Conditions Override

Mach		
Syntax	`-Mach #M`	
Description	Override the free-stream Mach number.	
Parameters	#M	Free-stream Mach number.
Examples	`-Mach 0.95`	

Table 9.101: BC Mach option

Reynolds		
Syntax	`-Reynolds #Re`	
Description	Override the free-stream Reynolds number. Requires temperature **AND** velocity (or Mach number) inputs.	
Parameters	#Re	Free-stream Reynolds number.
Examples	`-Reynolds 1e6`	

Table 9.102: BC Reynolds option

Velocity.magnitude		
Syntax	`-velocity.magnitude #V`	
Description	Override the free-stream velocity [m/s].	
Parameters	#V	Free-stream velocity [m/s].
Examples	`-velocity.magnitude 200`	

Table 9.103: BC velocity.magnitude option

Alpha		
Syntax	`-alpha #aoa`	
Description	Override the free-stream angle of attack [degrees].	
Parameters	#aoa	Free-stream angle of attack [degrees].
Examples	`-alpha 5`	

Table 9.104: BC alpha option

Beta		
Syntax	`-beta #beta`	
Description	Override the free-stream side slip angle [degrees].	
Parameters	#beta	Free-stream side slip angle [degrees].
Examples	`-beta 3`	

Table 9.105: BC beta option

U		
Syntax	`-u #u`	
Description	Override the free-stream x coordinate direction velocity [m/s].	
Parameters	#u	Free-stream x coordinate direction velocity [m/s].
Examples	`-u 150`	

Table 9.106: BC u option

V		
Syntax	`-v #v`	
Description	Override the free-stream y coordinate direction velocity [m/s].	
Parameters	#v	Free-stream y coordinate direction velocity [m/s].
Examples	`-v 20`	

Table 9.107: BC v option

W		
Syntax	`-w #w`	
Description	Override the free-stream z coordinate direction velocity [m/s].	
Parameters	#w	Free-stream z coordinate direction velocity [m/s].
Examples	`-w 10`	

Table 9.108: BC w option

T		
Syntax	`-T #T`	
Description	Override the free-stream temperature [K]. In the case of the stagnation inlet (see Section 5.4) this directive pertains to the free-stream stagnation temperature.	
Parameters	#T	Free-stream temperature [K].
Examples	`-T 300`	

Table 9.109: BC T option

p		
Syntax	`-p #p`	
Description	Override the free-stream pressure [Pa]. In the case of the stagnation inlet (see Section 5.4) this directive pertains to the free-stream stagnation pressure.	
Parameters	#p	Free-stream pressure [Pa].
Examples	`-p 105000`	

Table 9.110: BC p option

Trb.intensity		
Syntax	`-trb.intensity #ti`	
Description	Override the free-stream turbulent intensity (in absolute fraction, not percentage).	
Parameters	#ti	Free-stream turbulent intensity (in absolute fraction, not percentage). Default is 0.001.
Examples	`-trb.intensity 0.01`	

Table 9.111: BC trb.intensity option

Trb.dnu		
Syntax	`-trb.dnu #dnu`	
Description	Override the free-stream turbulent Spalart-Allmaras model variable ($\tilde{\nu}$).	
Parameters	#dnu	Free-stream turbulent Spalart-Allmaras model variable.
Examples	`-trb.dnu 0.1`	

Table 9.112: BC trb.dnu option



9.12 System Options

A system refers to a "System of Equations," *e.g.,* Euler or Reynolds Averaged Navier-Stokes equations. Other examples may include, turbulence model equations, various physical model equations, or a system of equations for the solution of elastic mesh deformation (required for aeroelastic flow simulations). Each system starts with the directive `--system`, followed by a series of options.

Certain systems require that an additional, complimentary system would be defined. For example, when simulating turbulent flows it is required to define a mean flow system that is consistent with the turbulence model selected and a second system to solve the actual turbulence model equations. An example of a simple system options section is found in Listing 9.12 followed by a description of all available options.

```
--system
   -name meanflow
   -type mcg.sst
   -time.step 0.1 5000 cfl.exponential 2
   -cell.gradient green.gauss.cell
   -face.gradient thin.layer
   -limiter mlp.3d
   -mlp.k 1
   -relax 0.4
   -plot plot velocity p T cy residual
   -log convergence cfl residual log.residual
   -table table Cp p qflux cy
```

Listing 9.12: Example of the system section

Name	
Syntax	`-name $system_name`
Description	Set the name of the system.
Parameters	$system_name Name of the system.

Examples	`-name mean.flow`
	`-name turbulence.model`
	`-name grid.movement`

Table 9.113: System name option

Type		
Syntax	`-type $type`	
Description	Type of equation set to be solved.	
Parameters	$type	Type of equation set. See Tables 9.134, 9.135, 9.136, 9.137, and 9.146 for equation system types.

Examples	`-type ideal.gas.mf.inviscid`
	`-type ideal.gas.mf.tnt`
	`-type turbulent.kw.sst`
	`-type eqb.viscous`
	`-type mcg.frozen.sa`
	`-type elastic.mesh.deform`	

Table 9.114: System type option

9.12.1 Flow Equations

This section describes the various options that are relevant to flow equations solvers. The flow equation types are described in Sections 9.12.1.2, 9.12.1.3, 9.12.1.4, and 9.12.1.5.

Cell.gradient		
Syntax	`-cell.gradient $method`	
Description	Method to calculate the cell center gradient that is required for high order approximations.	
Parameters	$method	Method for cell gradient calculation. See Table 9.116 for a list of available cell gradient methods.
Examples	`-cell.gradient green.gauss.node`	

Table 9.115: System cell.gradient option

Cell Gradient Types	
Cell Gradient type	Description
green.gauss.node	Calculate cell gradient based on the Green-Gauss Theorem using node reconstruction (default).
green.gauss.cell	Calculate cell gradient based on the Green-Gauss Theorem using cell center reconstruction.
least.square.1	Calculate cell gradient based on least square approximations using r (the distance) as the weight.
least.square.1.5	Calculate cell gradient based on least square approximations using $r^{1.5}$ as the weight.
least.square.2	Calculate cell gradient based on least square approximations using r^2 as the weight.

Table 9.116: Cell gradient types

Face Gradient		
Syntax	`-face.gradient $method`	
Description	Method to calculate the face gradient that is required for viscous fluxes calculations.	
Parameters	$method	Method for face gradient calculation. See Table 9.118 for a list of available face gradient methods.
Examples	`-face.gradient diamond`	

Table 9.117: System face.gradient option

Face Gradient Types	
Face Gradient type	Description
diamond	Calculate face gradient based on the Green-Gauss Theorem using a diamond-like shape (default).
thin.layer	Calculate face gradient based on the thin layer assumption.
hasselbacher	Calculate face gradient based on defect correction by Hasselbacher.

Table 9.118: Face gradient types

Limiter		
Syntax	`-limiter $limiter`	
Description	Type of limiter.	
Parameters	$limiter	Type of limiter. See Table 9.120 for limiter types.
Examples	`-limiter venka.2d`	

Table 9.119: Limiter option

Limiter Types	
Limiter type	Description
none	No limiter.
1st	First order.
venka.2d	Venkatakrishnan limiter (2D domain).
venka.3d	Venkatakrishnan limiter (3D domain).
mlp.2d	MLP-u2 limiter (2D domain).
mlp.3d	MLP-u2 limiter (3D domain).
mlp.pw	MLP.PW limiter (2D, 3D).
minmod	Minmod limiter (2D, 3D).

Table 9.120: Limiter types

Venka K		
Syntax	`-venka.k #k`	
Description	Change the Venkatakrishnan limiter coefficient. Default is 5.	
Parameters	#k	Venkatakrishnan limiter coefficient.
Examples	`-venka.k 5`	

Table 9.121: System venka.k option

Time Step		
Syntax	`-time.step #initial #iterations {cfl.linear | cfl.exponential | cfl.tangential | dt.linear | dt.exponential | dt.tangential } #final`	
Description	Set the CFL number or time step (first order single time stepping).	
Parameters	#initial	Initial CFL number or time step.
	#iterations	Number of iteration for which the CFL number or time step grows from #initial to #final.
	cfl.linear	Linear growth of the CFL number.
	cfl.exponentia	Exponential growth of the CFL number.
	cfl.tangential	Hyperbolic tangent growth of the CFL number.
	dt.linear	Linear growth of the time step.
	dt.exponentia	Exponential growth of the time step.
	dt.tangential	Hyperbolic tangent growth of the time step.
	#final	Final CFL number or time step.
Examples	`-time.step 5 50 cfl.linear 10`	

Table 9.122: System CFL option



Implicit Jacobi	
Syntax	`-implicit.jacobi`
Description	Use Jacobi instead of the default Gauss-Seidel.
Parameters	N/A
Examples	`-implicit.jacobi`

Table 9.123: System implicit.jacobi option

Convection No-slip Diagonal		
Syntax	`-convection.noslip.diagonal {normal | analytic | implicit.on.noslip}`	
Description	Select the way the diagonal convective jacobian is estimated on no-slip boundaries.	
Parameters	normal	The inviscid diagonal is evaluated directly based on the scheme.
	analytic	Use analytical Jacobian
	implicit.on.no	Implicit treatment of wall boundary conditions through the Jacobian.
Examples	`-convection.noslip.diagonal analytic`	

Table 9.124: System convection.noslip.diagonal option

Convection Impermeable. Diagonal		
Syntax	`-convection.impermeable.diagonal {normal | analytic | implicit.on.noslip}`	
Description	Selects the way the diagonal convective jacobian is estimated on impermeable boundaries.	
Parameters	normal	The inviscid diagonal is evaluated directly based on the scheme.
	analytic	Use analytical Jacobian
	implicit.on.no	Implicit treatment of wall boundary conditions through the Jacobian.
Examples	`-convection.impermeable.diagonal analytic`	

Table 9.125: System convection.impermeable.diagonal option

Convection Symmetry Diagonal		
Syntax	`-convection.symmetry.diagonal {normal | analytic | implicit.on.noslip}`	
Description	Selects the way the diagonal convective jacobian is estimated on symmetry boundaries.	
Parameters	normal	The inviscid diagonal is evaluated directly based on the scheme.
	analytic	Use analytical Jacobian
	implicit.on.no	Implicit treatment of wall boundary conditions through the Jacobian.
Examples	`-convection.symmetry.diagonal analytic`	

Table 9.126: System convection.symmetry.diagonal option

Damp	
Syntax	`-damp #damp-factor`
Description	Set damping to the convection Jacobian. Values that are less than unity increase stability. Not recommended for turbulence model equations. The recommended damping factor is 0.5-1.0. Default is 1.
Parameters	#damp-factor Damping factor.
Examples	`-damp 0.75`

Table 9.127: System damp option

Relax	
Syntax	`-relax #relax-factor`
Description	Set relaxation to the solution increment. The recommended relaxation factor is 0.5-1.0. Default is 1.
Parameters	#relax-factor Relaxation factor.
Examples	`-relax 0.75`

Table 9.128: System relaxation option

Iteration Convergence	
Syntax	`-iteration.convergence #criterion`
Description	Set the convergence criterion (log(Res/Res0)), default is ignored. For unsteady flows this would be the convergence of the whole simulation. For dual time this would be convergence of the iterations.
Parameters	#criterion Convergence criterion in terms of [log(Res/Res0)].
Examples	`-iteration.convergence -5`

Table 9.129: System iteration.convergence option

User Source	
Syntax	`-user.source $source_name`
Description	Adds a previously defined user source to this system. Multiple user sources are applied in the order they appear.
Parameters	#source_- User defined source name. name
Examples	`-user.source my_user_source_name`

Table 9.130: System user.source option

9.12.1.1 Turbulence Model Equations System Options

Realizability.trb.w	
Syntax	`-realizability.trb.w`
Description	Adds realizability condition for omega (applies only to k-ω-SST turbulence model).
Parameters	N/A
Examples	`-realizability.trb.w`

Table 9.131: System realizability.trb.w option

Realizability.trb.w.Smag	
Syntax	`-realizability.trb.w.Smag`
Description	Adds realizability condition for omega, based on the size of the strain-rate tensor (applies only to k-ω-SST turbulence model).
Parameters	N/A
Examples	`-realizability.trb.w.Smag`

Table 9.132: System realizability.trb.w.Smag option

Source.mpk		
Syntax	`-source.mpk #mpk`	
Description	Limit turbulence model production (applies only to k-ω turbulence models).	
Parameters	#mpk	Upper limit for production term.
Examples	`-source.mpk 5`	

Table 9.133: System source.mpk option

9.12.1.2 Mean Flow Equations for Ideal Gas

System Types

System type	Description
ideal.gas.mf.inviscid	Solve the Euler equations assuming inviscid, perfect gas flow.
ideal.gas.mf.viscous	Solve the Navier-Stokes equations assuming laminar perfect gas flow.
ideal.gas.mf.tnt	Solve the Navier-Stokes equations assuming turbulent ideal gas flow (using the k-ω-TNT turbulence model). This assumes an additional system for solving the k-ω-TNT turbulence model equations.
ideal.gas.mf.sst	Solve the Navier-Stokes equations assuming turbulent ideal gas flow (using the k-ω-SST turbulence model). This assumes an additional system for solving the k-ω-SST turbulence model equations.
ideal.gas.mf.sa	Solve the Navier-Stokes equations assuming turbulent ideal gas flow (using the Spalart-Allmaras turbulence model) This assumes an additional system for solving the Spalart-Allmaras turbulence model equation.

Table 9.134: Mean flow system types for ideal gas

9.12.1.3 Mean Flow Equations for Multi Component Gas under Chemical Equilibrium

System Types

System type	Description
eqb.inviscid	Solve the Euler equations assuming inviscid perfect gas flow under chemical equilibrium.
eqb.viscous	Solve the Navier-Stokes equations assuming laminar perfect gas flow under chemical equilibrium.
eqb.tnt	Solve the Navier-Stokes equations assuming turbulent perfect gas flow under chemical equilibrium (requires subsequent mean-flow k-ω-TNT model system).
eqb.sst	Solve the Navier-Stokes equations assuming turbulent perfect gas flow under chemical equilibrium (requires subsequent mean-flow k-ω-SST model system).
eqb.sa	Solve the Navier-Stokes equations assuming turbulent perfect gas flow under chemical equilibrium (requires subsequent mean-flow Spalart-Allmaras model system).

Table 9.135: Mean flow system types for multi component gas under chemical equilibrium

9.12.1.4 Mean Flow Equations for Multi Component Gas

System Types

System type	Description
mcg.inviscid	Solve the Euler equations assuming inviscid perfect gas flow under chemical non-equilibrium.
mcg.viscous	Solve the Navier-Stokes equations assuming laminar perfect gas flow under chemical non-equilibrium.
mcg.tnt	Solve the Navier-Stokes equations assuming turbulent perfect gas flow under chemical non-equilibrium (requires subsequent mean-flow k-ω-TNT model system).
mcg.sst	Solve the Navier-Stokes equations assuming turbulent perfect gas flow under chemical non-equilibrium (requires subsequent mean-flow k-ω-SST model system).
mcg.sa	Solve the Navier-Stokes equations assuming turbulent perfect gas flow under chemical non-equilibrium (requires subsequent mean-flow Spalart-Allmaras model system).
mcg.frozen.inviscid	Solve the Euler equations assuming inviscid perfect gas flow under frozen chemistry.

mcg.frozen.viscous	Solve the Navier-Stokes equations assuming laminar perfect gas flow under frozen chemistry.
mcg.frozen.tnt	Solve the Navier-Stokes equations assuming turbulent perfect gas flow under frozen chemistry (requires subsequent mean-flow k-ω-TNT model system).
mcg.frozen.sst	Solve the Navier-Stokes equations assuming turbulent perfect gas flow under frozen chemistry (requires subsequent mean-flow k-ω-SST model system).
mcg.frozen.sa	Solve the Navier-Stokes equations assuming turbulent perfect gas flow under frozen chemistry (requires subsequent mean-flow Spalart-Allmaras model system).

Table 9.136: Mean flow system types for multi component gas

9.12.1.5 Turbulence Model Equations

System Types

System type	Description
turbulent.kw.tnt	Solve the k-ω-TNT turbulence model equations.
turbulent.kw.tnt.xles	Solve the k-ω-TNT-XLES hybrid turbulence model equations (see Section 3.7.1).
turbulent.kw.sst	Solve the k-ω-SST turbulence model equations.
turbulent.kw.sst.sas	Solve the k-ω-SST-SAS turbulence model equations.
turbulent.kw.sst.iddes	Solve the k-ω-SST-IDDES hybrid turbulence model equations (see Section 3.7).
turbulent.sa	Solve the Spalart-Allmaras turbulence model equation.
turbulent.sa.iddes	Solve the Spalart-Allmaras hybrid turbulence model equation (see Section 3.7).

Table 9.137: Turbulence models system types

9.12.2 Flow Equations System Outputs

Log	
Syntax	`-log $log [log-options] ...`
Description	Report in log file. See Tables 9.140 and 9.141 for log options list.
Parameters	$log Log name.
Examples	`-log coefficients cl cd`

Table 9.138: System log option

Plot	
Syntax	`-plot $plot [plot-options] ...`
Description	Functions to include in FV-UNS or CGNS file. See Table 9.142, 9.143, 9.144, and 9.145 for plot functions list.
Parameters	$plot Plot name.
Examples	`-plot NACA0012 r p velocity residual`

Table 9.139: System plot option

9.12.2.1 System Log Options

General System Log Options

Log type	Description
cfl	System CFL.
dt	System Δt.
time	Time.
residual	System residual.
log.residual	System log of the residual.
minimal.timestep	Minimal time step (for non sp.petsc).
orig(minimal.timestep)	Minimal time step original cell index (for non sp.petsc).
cell(minimal.timestep)	Minimal time step zonal cell index (for non sp.petsc).
zone(minimal.timestep)	Minimal time step zone index (for non sp.petsc).
orig(res)	Original (ranked grid) cell index of the maximum residual.
primal(res)	Primal (input grid) cell index of the maximum residual.
cell(res)	Max residual zonal cell number.
zone(res)	Max residual zone number.
uns(res)	Max residual uns name.
petsc.iters	Petsc iteration number.
petsc.residual	Petsc residual.

Table 9.140: General system log options



Turbulence Model System Log Options	
Log type	Description
max(mt)	Max of the turbulence viscosity.
cell(mt)	Max turbulent viscosity cell number.
zone(mt)	Max turbulent viscosity zone number.
xyz(mt)	Adds maximal turbulent viscosity cell center coordinates.

Table 9.141: Turbulence model system log options

9.12.2.2 System Plot Functions

General System Plot Functions

Plot function	Description
residual	Residual field.
relax	Relaxation field.
cfl	CFL.

Table 9.142: General system plot functions

Mean Flow System Plot Functions

Plot function	Description
velocity	Velocity vector field.
{u | v | w }	Velocity vector field components.
dudx, dudy, dudz	Cell centered u-velocity derivatives.
dvdx, dvdy, dvdz	Cell centered v-velocity derivatives.
dwdx, dwdy, dwdz	Cell centered w-velocity derivatives.
r	Density field.
p	Pressure field.
Cp	Pressure coefficient.

Cf	Skin-friction coefficient.
Cf.signed	Signed skin-friction coefficient.
T	Temperature field.
{lim.r | lim.epsilon | lim.u | lim.v | lim.w}	Limiter fields.
wall.distance	Wall distance field.
qflux	Normal heat flux.
total.enthalpy	Total enthalpy.
total.temperature	Total temperature.
total.pressure	Total pressure.
enthalpy.conductance	Enthalpy conductance.
mach	Mach number.
Tw	Magnitude of $\bar{\tau}_{wall}$.
Tw.vec	The vector $\bar{\tau}_{wall}$.
Tw{x | y | z }	X | Y | Z direction component of $\bar{\tau}_{wall}$.
cfl	CFL.
q.criterion	Q criterion.
lambda2	λ_2.
eos.gamma	Heat capacity ratio ($\gamma = \frac{c_p}{c_v}$).
eos.R	Gas constant.

Table 9.143: Mean flow system plot functions

Turbulence Model System Plot Functions

Plot function	Description
mt	Turbulent viscosity field (where applicable).
trb.k	Turbulent kinetic energy field (where applicable).
trb.w	Turbulent specific dissipation rate field (where applicable).
{trb.lim.k | trb.lim.w}	Turbulence model limiters (where applicable).
yplus	y^+ (where applicable).
rans.les	RANS / LES flag (where applicable).
hybrid.diss.factor	Central/upwind dissipation factor (where applicable).
resolved.reynolds.stress	Resolved stress (where applicable).
modeled.reynolds.stress	Modeled stress (where applicable).
integral.length	The turbulence integral length scale (where applicable).
cells.per.eddy	Cells per eddy (where applicable).

Table 9.144: Turbulence model system plot functions

Multi Component Gas Plot Functions

Plot function	Description
cy	Species mass fraction.
react.fraction	Cell reaction volume fraction (available for PaSR kinetics model).

Table 9.145: Multi component gas plot functions

9.12.3 Elastic Mesh-Deformation System

The elastic mesh-deformation system propagates prescribed boundary displacements into the CFD volume mesh by treating the mesh as an artificial elastic solid.

For aeroelastic calculations, the system may be defined as, for example,

```
--system
   -name deform
   -type elastic.mesh.deform
   -deform.from.ref
   -deform.stiffness.distribution volume
```

Listing 9.13: Example of aeroelastic calculations system section

System Types

System type	Description
elastic.mesh.deform	Deform the mesh based on the solution of the elasticity equations as described in Section 7.7.

Table 9.146: Deformation system types

Deform From Reference Mesh	
Syntax	`-deform.from.ref`
Description	Evaluate the mesh deformation relative to the undeformed reference mesh rather than incrementally from the current mesh configuration. This formulation is recommended for aeroelastic calculations because the prescribed aeroelastic displacement is defined relative to the reference aerodynamic shape.
Parameters	N/A.
Examples	`-deform.from.ref`

Table 9.147: Elastic mesh deformation from the reference mesh

Mesh Stiffness Distribution		
Syntax	`-deform.stiffness.distribution {constant | volume | wall.distance}`	
Description	Select the spatial distribution of the relative elastic mesh stiffness. The default distribution is `volume`.	
Parameters	constant	Use uniform mesh stiffness, $E = 1.$.
	volume	Increase the mesh stiffness as cell volume decreases.
	wall.distance	Increase the mesh stiffness as the wall distance decreases, preferentially protecting cells close to the wall. For inviscid simulations each wall boundary condition participating in the wall-distance calculation must include the `wall.distance` directive.
Examples	`-deform.stiffness.distribution constant`	
	`-deform.stiffness.distribution volume`	
	`-deform.stiffness.distribution wall.distance`	

Table 9.148: Elastic mesh-deformation stiffness distribution

Maximum Mesh Stiffness		
Syntax	`-deform.stiffness.max $value`	
Description	Set the maximum relative mesh stiffness. The limit is applied after evaluation of the selected stiffness distribution and stiffness power. By default, no maximum stiffness limit is imposed.	
Parameters	$value	Maximum relative mesh stiffness.
Examples	`-deform.stiffness.max 1000`	

Table 9.149: Elastic mesh-deformation maximum stiffness

Mesh Stiffness Power		
Syntax	`-deform.stiffness.power $value`	
Description	Set the exponent applied to the mesh stiffness distribution. Increasing the power increases the contrast between stiff and compliant regions of the mesh.	
Parameters	$value	Exponent applied to the relative mesh stiffness.
Examples	`-deform.stiffness.power 1.0`	

Table 9.150: Elastic mesh-deformation stiffness power

Wall-Distance Stiffness Offset		
Syntax	`-deform.stiffness.offset $value`	
Description	Set the wall-distance offset distance used with the `wall.distance` stiffness distribution. Increasing the offset distance extends the highly stiffened region away from the wall.	
Parameters	$value	Wall-distance stiffness offset distance.
Examples	`-deform.stiffness.offset 0.001`	

Table 9.151: Elastic mesh-deformation wall-distance stiffness offset

9.12.3.1 Solution using PETSc

The elastic mesh-deformation equations are solved using PETSc. PETSc options for this system are specified by prepending `sys_elastic_mesh_deform_` to the corresponding PETSc option. For example,

```
-sys_elastic_mesh_deform_ksp_type gmres
-sys_elastic_mesh_deform_ksp_rtol 1e-10
```

Listing 9.14: Example of aeroelastic solver PETSc options

For aeroelastic calculations, a relative linear-solver tolerance of approximately 10^{-9}–10^{-10} is recommended. See Section 7.7 for discussion of the mesh-deformation formulation and solution accuracy.

Important notes regarding the use of PETSc toward the solution of the elastic mesh deformation equations. First, if no specific options are prescribed, **Arion** automatically applies an optimized set of PETSc options. If one or more are set by the user, then none of the optimized set of options is applied but rather certain default PETSc options are applied. And last, the options should be added under the PETSc section (see Section 9.14).

9.13 Solve Options

The Solve input options section starts with the `--solve` directive, followed by a series of solve options. An example of a simple solve options section is found in Listing 9.15 followed by a description of all available options.

```
--solve
    -eos neq
    -p 278
    -T 251
    -Mach 5
    -alpha 0
    -beta 0
    -trb.intensity 0.0035
    -trb.mt 0.1
    -time.march implicit.petsc
    -convection.flux ausm.dv
    -convection.jacobian  steger
    -spatial.order 2
    -iterations 30000
    -save 1000
    -time.step.reduction 4
    -save.path ./save/
    -log convergence iter exec.time
    -reference.longitudinal 4.4
```

Listing 9.15: Example of the solve section

Convection Flux		
Syntax	`-convection.flux $method`	
Description	Set the convection flux approximation method.	
Parameters	$method	Flux approximation method. See Table 9.153 for available convection flux types.
Examples	`-convection.flux hllc.roe`	

Table 9.152: Solve convection.flux option

Convection Flux Types	
Convection flux type	Description
hllc.roe	HLLC Roe.
hllc.davis	HLLC Davis.
ausm	AUSM.
ausm.up	AUSM$^+$-up.
ausm.dv	AUSM-DV.
slau2	SLAU2.
steger	Steger-Warming (for chemical equilibrium only).

Table 9.153: Convection flux types

Convection Jacobian	
Syntax	`-convection.jacobian $jacobian`
Description	Set the convection Jacobian. See Table 9.155 for available convection Jacobian types.
Parameters	$jacobian Jacobian.
Examples	`-convection.jacobian hllc.roe`

Table 9.154: Solve convection.jacobian option

Convection Jacobian Types	
Convection Jacobian type	Description
hllc.roe	HLLC Roe.
hllc.davis	HLLC Davis.
van.leer	van Leer.
steger	Steger-Warming Flux Vector Splitting.
rossow.low.mach	Low Mach number Rossow.
matrix.free	Matrix free.

Table 9.155: Convection Jacobian types

Spatial Order		
Syntax	`-spatial.order #`	
Description	Set the convection flux spatial order. Available options are either 1 for first order or 2 for second order.	
Parameters	#	Convection flux approximation order.
Examples	`-spatial.order 2`	

Table 9.156: Solve spatial.order option

Sweeps		
Syntax	`-sweeps #`	
Description	Set the number of sweeps within an iteration.	
Parameters	#	Number of sweeps. Can be any even number. Default is 6.
Examples	`-sweeps 4`	

Table 9.157: Solve sweeps option

Axisymmetric	
Syntax	`-axisymmetric`
Description	Set an axisymmetric simulation. The 2-D grid must be positioned in the $x-z$ plane and extruded by one node into the y-direction. Here, x represents the axis of rotation and z corresponds to the radius of rotation.
Parameters	N/A
Examples	`-axisymmetric`

Table 9.158: Solve axisymmetric option

Time March		
Syntax	`-time.march $method`	
Description	Set the time marching method. See Table 9.160 for a list of available time marching methods.	
Parameters	$method	Time marching method.
Examples	`-time.march implicit.pgs`	

Table 9.159: Solve time.march option

Time Marching Methods

Time marching method	Description
explicit.euler	Explicit Euler time marching scheme.
explicit.rk3	Third order Runge-Kutta explicit time marching scheme.
explicit.rk4	Fourth order Runge-Kutta explicit time marching scheme.
explicit.rk5	Fifth order Runge-Kutta explicit time marching scheme.
implicit.pgs	Implicit point Gauss-Seidel time marching scheme.
implicit.b2.pgs	Second order implicit B2 time marching scheme.
implicit.heun.R.pgs	Second order Runge-Kutta implicit point Gauss-Seidel time marching scheme.
implicit.rk3.R.pgs	Third order Runge-Kutta implicit point Gauss-Seidel time marching scheme.
implicit.rk4.R.pgs	Fourth order Runge-Kutta implicit point Gauss-Seidel time marching scheme.
implicit.rk5.R.pgs	Fifth order Runge-Kutta implicit point Gauss-Seidel time marching scheme.
implicit.rk3.R.petsc	Krylov-based Implicit (Runge-Kutta R, 3rd order).
implicit.rk4.R.petsc	Krylov-based Implicit (Runge-Kutta R, 4th order).
implicit.b2.petsc	Krylov-based Implicit (B2).

Table 9.160: Time marching methods



Time.accurate.explicit		
Syntax	`-time.accurate.explicit #steps #time-step`	
Description	Use 3rd order Runge-Kutta TVD (see Section 4.4.1.3) or Explicit Euler as a time-accurate method. Requires to set the time marching method (explicit.euler or explicit.rk3) as in Table 9.160.	
Parameters	#steps	Number of steps.
	#time-step	Time step.
Examples	`-time.march explicit.rk3`	
	`-time.accurate.explicit 1000 0.0001`	

Table 9.161: Solve time.accurate.explicit option

Iterations		
Syntax	`-iterations #`	
Description	Set the number of iterations. For unsteady flows this would be the number of iterations of the whole simulation. For dual time this would be the maximum iterations of the current time step.	
Parameters	#	Number of iterations.
Examples	`-iterations 1000`	

Table 9.162: Solve iterations option

Time.step.reduction	
Syntax	`-time.step.reduction #attempts`
Description	Set time the number of consecutive CFL reduction attempts when non-physical values are detected in the solution.
Parameters	#attempts Number of attempts.
Examples	`-time.step.reduction 3`

Table 9.163: Solve time.step.reduction option

Time.step.reduction.value	
Syntax	`-time.step.reduction.value #value`
Description	The minimal CFL/DT allowable for the reduction.
Parameters	#value Minimal allowable CFL/DT.
Examples	`-time.step.reduction.value 0.1`

Table 9.164: Solve time.step.reduction.value option

Dual.time		
Syntax	`-dual.time #steps #time-step [1st.order | 2nd.order | auto.order]`	
Description	Set dual time stepping simulation.	
Parameters	#steps	Number of physical time steps.
	#time-step	Physical time step (Δt).
	1st.order	First order temporal accuracy.
	2nd.order	Second order temporal accuracy.
	auto.order	Automatic temporal accuracy.
Examples	`-dual.time 200 0.0001`	

Table 9.165: Solve dual.time option

Global.minimal.timestep	
Syntax	`-global.minimal.timestep`
Description	Forces all the cells to use the global minimal time step that was calculated.
Parameters	N/A
Examples	`-global.minimal.timestep`

Table 9.166: Solve global.minimal.timestep option

Save	
Syntax	`-save #`
Description	Set the intermittency of output of restart files.
Parameters	# Number of steps between output.
Examples	`-save 10`

Table 9.167: Solve save option

Save.path	
Syntax	`-save.path $path`
Description	Set the path for output of log and restart files.
Parameters	$path Path of log and restart files.
Examples	`-save.path ./save/`

Table 9.168: Solve save.path option

Save.sequential	
Syntax	`-save.sequential`
Description	Adds iteration signature to the saves.
Parameters	N/A
Examples	`-save.sequential`

Table 9.169: Solve save.sequential option

Load.path	
Syntax	`-load.path $path`
Description	Set the path for restart files.
Parameters	$path Path of restart files.
Examples	`-load.path ./load/`

Table 9.170: Solve load.path option

Load.sequence	
Syntax	`-load.sequence #iteration`
Description	Load a specific iteration.
Parameters	#iteration Iteration for load.
Examples	`-load.sequence 1000`

Table 9.171: Solve load.sequence option

EOS		
Syntax	`-eos $eos`	
Description	Set the equation of state.	
Parameters	$eos	Equation of state. Currently, only perfect gases are supported.
Examples	`-eos air`	

Table 9.172: Solve eos option

Molecular		
Syntax	`-molecular $molecular`	
Description	Set the name of the molecular viscosity and thermal conductivity.	
Parameters	$molecular	Molecular relations name.
Examples	`-molecular air-suther`	

Table 9.173: Solve molecular option

P		
Syntax	`-p #`	
Description	Set the free stream pressure.	
Parameters	#	Free stream pressure.
Examples	`-p 101325`	

Table 9.174: Solve p option

T	
Syntax	`-T #`
Description	Set the free stream temperature.
Parameters	# Free stream temperature.
Examples	`-p 288`

Table 9.175: Solve T option

Mach	
Syntax	`-Mach #`
Description	Set the free stream Mach number.
Parameters	# Free stream Mach number.
Examples	`-Mach 0.8`

Table 9.176: Solve Mach option

Reynolds	
Syntax	`-Reynolds #Re`
Description	Set the free-stream Reynolds number. Requires temperature **AND** velocity (or Mach number) inputs.
Parameters	#Re Free-stream Reynolds number.
Examples	`-Reynolds 1e6`

Table 9.177: Solve Reynolds option

Velocity.magnitude		
Syntax	`-velocity.magnitude #`	
Description	Set the free stream velocity magnitude.	
Parameters	#	Free stream velocity magnitude.
Examples	`-velocity.magnitude 200`	

Table 9.178: Solve velocity magnitude option

U		
Syntax	`-u #`	
Description	Set the free stream X coordinate direction velocity component.	
Parameters	#	Free stream X coordinate direction velocity component.
Examples	`-u 200`	

Table 9.179: Solve u option

V		
Syntax	`-v #`	
Description	Set the free stream Y coordinate direction velocity component.	
Parameters	#	Free stream Y coordinate direction velocity component.
Examples	`-v 0`	

Table 9.180: Solve v option

W		
Syntax	`-w #`	
Description	Set the free stream Z coordinate direction velocity component.	
Parameters	#	Free stream Z coordinate direction velocity component.
Examples	`-w 20`	

Table 9.181: Solve w option

Alpha		
Syntax	`-alpha #`	
Description	Set the free stream angle of attack.	
Parameters	#	Free stream angle of attack in degrees.
Examples	`-alpha 10`	

Table 9.182: Solve alpha option

Beta		
Syntax	`-beta #`	
Description	Set the free stream side slip angle.	
Parameters	#	Free stream side slip angle in degrees.
Examples	`-beta 0`	

Table 9.183: Solve beta option

Trb.intensity		
Syntax	`-trb.intensity #`	
Description	Set the free stream turbulence intensity.	
Parameters	#	Free stream turbulence intensity (in absolute fraction, not percentage).
Examples	`-trb.intensity 0.01`	

Table 9.184: Solve trb.intensity option

Trb.mt		
Syntax	`-trb.mt #mt`	
Description	Set the free stream turbulence viscosity.	
Parameters	#mt	Free stream turbulence viscosity (in absolute fraction from the molecular viscosity, not percentage).
Examples	`-trb.mt 0.01`	

Table 9.185: Solve trb.mt option

Prandtl.trb	
Syntax	`-Prandtl.trb #Pr_t`
Description	Sets the turbulent Prandtl number (Default is 0.9).
Parameters	#Pr_t Turbulent Prandtl number.
Examples	`-Prandtl.trb 0.9`

Table 9.186: Solve Prandtl.trb option

Reference.velocity	
Syntax	`-reference.velocity #ref`
Description	Set the reference velocity. Default is the mean flow free stream velocity.
Parameters	#ref Reference velocity.
Examples	`-reference.velocity 300`

Table 9.187: Solve reference.velocity option

Reference.mach	
Syntax	`-reference.mach #ref`
Description	Set the reference Mach number. Default is the mean flow free stream Mach number.
Parameters	#ref Reference Mach number.
Examples	`-reference.mach 0.75`

Table 9.188: Solve reference.mach option

Reference.pressure	
Syntax	`-reference.pressure #ref`
Description	Set the mean-Flow reference pressure, used to override free-stream pressure.
Parameters	#ref Reference pressure.
Examples	`-reference.pressure 101325`

Table 9.189: Solve reference.pressure option

Reference.velocity.trb	
Syntax	`-reference.velocity.trb #ref`
Description	Set the reference velocity for the turbulence model (used to override the free-stream velocity).
Parameters	#ref Free-stream velocity [m/s].
Examples	`-reference.velocity.trb 210`

Table 9.190: Solve reference.velocity.trb option

Reference.mach.trb	
Syntax	`-reference.mach.trb #ref`
Description	Set the reference Mach number for the turbulence model (used to override the free-stream Mach number).
Parameters	#ref Free-stream Mach number.
Examples	`-reference.mach.trb 0.7`

Table 9.191: Solve reference.mach.trb option

Reference.longitudinal		
Syntax	`-reference.longitudinal #ref`	
Description	Set the reference length for Reynolds number evaluation. Unused by the current version.	
Parameters	#ref	Reference length.
Examples	`-reference.length 1.5`	

Table 9.192: Solve reference.length option

Ref		
Syntax	`-ref.$ # { # # }`	
Description	Set the reference length, area, and reference point for force and moment coefficient calculations.	
Parameters	-reference.area	Reference area (scalar).
	-reference.len	Reference length (vector).
	-reference.point	Reference point (vector).
	#	Reference value.
Examples	`-reference.len 0.325 0.1 0.325`	
	`-reference.point 1 0 0`	

Table 9.193: Solve ref option

Plot	
Syntax	`-plot [plot-options] ...`
Description	Add to the FV-UNS or CGNS file. See Table 9.195 for plot options list.
Parameters	N/A
Examples	`-plot rank owner orig`

Table 9.194: Solve plot option

Solve Plot Options	
Log type	Description
rank	Add the rank to the fvuns or cgns plot file.
part	Add the part to the fvuns or cgns plot file.
owner	Add ownership to the fvuns or cgns plot file.
orig	Add original node index to the fvuns or cgns plot file.

Table 9.195: Solve plot options

Table	
Syntax	`-table [table-options] ...`
Description	Add to table output. See Table 9.197 for table options list.
Parameters	N/A
Examples	`-table rank`

Table 9.196: Solve table option

Solve Table Options	
Log type	Description
rank	Add the rank to the table output.
face.area	Add the face area to the table output (only applicable when face output is enabled via `-loc.face`, see Table 9.300).

Table 9.197: Solve table options

Log	
Syntax	`-log $log [log-options] ...`
Description	Report in log file. See Table 9.199 for log options list.
Parameters	$log Log name.
Examples	`-log progress-logs iter step time`

Table 9.198: Solve log option

Solve Log Options

Log type	Description
iter	Iteration number.
step	Step number.
pos	Position # (iteration for steady state, step for dual time simulation).
time	Physical time for dual time simulation.
exec.time	Cumulative execution time.
iter.avg.time	iteration average time.
iteration.time	Iteration time.
step.time	Step time.
petsc.mem.usage	Petsc memory usage in mebibyte.
petsc.mem.malloc	Petsc memory malloc in mebibyte.

Table 9.199: Solve log options

Source.mom

Syntax	`-source.mom #fx #fy #fz`
Description	Set a constant momentum source term of the form r × [fx, fy, fz] Where r is the local density.
Parameters	#fx #fy #fz External acceleration components.
Examples	`-source.mom 1.0 1.0 1.0`

Table 9.200: Solve source.mom option

Hyb.diss.min	
Syntax	`-hyb.diss.min #value`
Description	Set the minimum dissipation value for the hybrid upwind–central convection scheme used in LES/DES simulations (default value is 0.050000). Note that setting this value to 1 results in a purely upwind scheme.
Parameters	#value Minimum dissipation value.
Examples	`-hyb.diss.min 0.5`

Table 9.201: Solve hyb.diss.min option

Hyb.diss.type	
Syntax	`-hyb.diss.type $type`
Description	Sets the scheme for determining the dissipation factor in the hybrid upwind–central convection scheme. Note that multiple schemes may be active at the same time, and the final dissipation value is taken as the maximum over all active schemes. The options are listed in Table 9.203. Can use multiple types and the maximum between all prevails. Note that the IDDES $\tilde{f}_d$ is automatically activated when using the IDDES scheme.
Parameters	$type Dissipation factor type.
Examples	`-hyb.diss.type ducros`

Table 9.202: Solve hyb.diss.type option

Hybrid Dissipation Factor Types	
Dissipation Type	Description
iddes	IDDES.
ducros	Ducros shock sensor.
travin	Travin shock sensor.

Table 9.203: Hybrid Dissipation Factor Types

Grid.length.scale	
Syntax	`-grid.length.scale $gridLengthScale`
Description	Set the grid length scale for the hybrid RANS/LES simulation. The options are listed in Table 9.205.
Parameters	$type Dissipation factor type.
Examples	`-grid.length.scale sla`

Table 9.204: Solve grid.length.scale option

Grid Length Scale Types	
Dissipation Type	Description
sla	Shear-Layer-Adapted grid length scale (see Section 3.7.3.2).
max.distance	Maximum distance between cell and it's neighbors.
volume	Cell volume.
iddes	IDDES length scale ???

Table 9.205: Grid Length Scale Types

Gravitation.acceleration		
Syntax	`-gravitation.acceleration #gx #gy #gz`	
Description	Set the gravitation acceleration [m/sec^2], used for weight calculation in the dof calculations (default is [0.00, 0.00, -9.81]).	
Parameters	#gx	Gravitation acceleration in the x-coordinate direction.
	#gy	Gravitation acceleration in the y-coordinate direction.
	#gz	Gravitation acceleration in the z-coordinate direction.
Examples	`-gravitation.acceleration 0.0 0.0 -19.62`	
	`-gravitation.acceleration 0.0 -9.81 0.00`	

Table 9.206: Solve gravitation.acceleration option

2d.tolerance		
Syntax	`-2d.tolerance #tolerance`	
Description	Tolerance for two-dimensional detection.	
Parameters	#tolerance	Set the tolerance.
Examples	`-2d.tolerance 1e-8`	

Table 9.207: Solve 2d.tolerance option

Geometric.tolerance	
Syntax	`-geometric.tolerance #tolerance`
Description	Geometric tolerance where needed.
Parameters	#tolerance Set the tolerance.
Examples	`-geometric.tolerance 2e-7`

Table 9.208: Solve geometric.tolerance option

9.13.1 Wall Distance Evaluation

Wall distance evaluation is implemented by searching through a BVH (bounding volume hierarchy) tree. It is always exact in the sense that it calculates the distance from a given cell center to a wall boundary face surface. The tree can have various properties (number of children per trunk), the search can be conducted from the tree root or from the last leaf found. Furthermore, the boundary volume can be expressed in double or single precision (it does not affect the distance calculation which is always in double precision). The algorithm results can be verified against a different calculation using BVH or against brute force (when all options are searched). On top of the search, there is a load balancing algorithm which can still search from different ranks in case there is a significant load imbalance. In its current version, the wall distance evaluation is significantly faster than the previous versions.

Wall.distance		
Syntax	`-wall.distance {exact | brute.force} [load.balance.rma | load.balance.async | no.load.balance] [BVH4 | BVH8] [from.root | from.leaf] [triangulated | not.triangulated] [double | float] [arion.exact.center | fast.exact.center | hiorder.exact]`	
Description	Set the wall distance algorithm.	
Parameters	exact	Fast BVH (bounding volume hierarchy) tree search.
	brute.force	Sequential search on all options.
	load.balance.rma	Load balance between ranks (processes) using Remote Memory Access (default).
	load.balance.asyn	Load balance between ranks (processes) using asynchronous MPI messages.
	no.load.balance	No load balancing.
	BVH4/BVH8	BVH tree number of children per trunk (default BVH4).
	from.root/from.le	Search from tree root or from last searched leaf (default from.root).
	triangulated/not.t	Convert quad boundary faces to triangles (default: not.triangulated).
	double/float	Keep TREE boundary extents in double or single precision (default double).
	fast.exact.center/a	Algorithms for calculating the distance from a point to a polygon, maintained for backward compatibility (default fast.exact.center).
Examples	`-wall.distance exact`	

Table 9.209: Solve wall distance option



9.13.2 Line Search

Line.search iters	
Syntax	`-line.search iters #iterations`
Description	Set the number of line.search iterations default is 0 (no line search iterations).
Parameters	#iterations Number of line search iterations.
Examples	`-line.search iters 1`

Table 9.210: Solve line search iterations option

Line.search min.dq	
Syntax	`-line.search min.dq #dq`
Description	Set the minimal relaxation factor that the line search can use (optional, default is 0.1).
Parameters	#dq Minimal relaxation factor.
Examples	`-line.search min.dq 1e-2`

Table 9.211: Solve line search minimum dq option

Line.search poly.order		
Syntax	`-line.search poly.order #order`	
Description	Set the polynomial order of the back-traced line search curve-fitting, can be 2 or 3 (optional, default is 3).	
Parameters	#order	Order of back-traced line search curve-fitting polynomial.
Examples	`-line.search poly.order 3`	

Table 9.212: Solve line search polynomial order option

Line.search beta		
Syntax	`-line.search beta #beta`	
Description	Set the convergence check criterion constant in the line search algorithm (optional, default is 1.000000e-04).	
Parameters	#beta	Line search convergence criterion constant.
Examples	`-line.search beta 1e-3`	

Table 9.213: Solve line search beta option

Line.search flavor		
Syntax	`-line.search flavor [meanflow | full | min]`	
Description	Set the line search flavor, *i.e.*, how it is applied to the system of equations. Options are: meanflow, full, and min. (default is full).	
Parameters	meanflow	Apply line search to the mean flow system only.
	full	Apply line search to all systems.
	min	Apply line search to all systems and use the minimum dq value.
Examples	`-line.search flavor min`	

Table 9.214: Solve line search flavor option

9.14 PETSc Toolkit

The PETSc input option section starts with the `--petsc` directive, followed by a series of petsc options. An example of a simple petsc options section is found in Listing 9.16 followed by a description of all available options. For advanced usage of the PETSc package the user is referred to the command line help or the online PETSc manual.

```
--petsc
   -ksp_norm_type unpreconditioned
   -ksp_max_it 200
   -ksp_rtol 2e-2
   -ksp_atol 1e-30
   -ksp_type fgmres
   -pc_type asm
   -sub_pc_type ilu
   -sub_pc_factor_levels 2
   -sub_pc_factor_fill 2
   -sub_ksp_type preonly
```

Listing 9.16: Example of the petsc section

KSP_view	
Syntax	`-ksp_view`
Description	View internal Krylov solver iterations.
Parameters	N/A
Examples	`-ksp_view`

Table 9.215: Petsc ksp_view option

KSP_monitor_true_residual	
Syntax	`-ksp_monitor_true_residual`
Description	Monitor true (un-preconditioned) residuals.
Parameters	N/A
Examples	`-ksp_monitor_true_residual`

Table 9.216: Petsc ksp_monitor_true_residual option

KSP_norm_type		
Syntax	`-ksp_norm_type [type]`	
Description	Select type of norm used to determine convergence of Krylov solver.	
Types	none	Use no norm calculations.
	preconditioned	Use the preconditioned residual norm.
	unpreconditioned	Use the unpreconditioned residual norm.
	natural	Use the norm induced by the linear operator.
Examples	`-ksp_norm_type none`	

Table 9.217: Petsc ksp_norm_type option

KSP_max_it		
Syntax	`-ksp_max_it #`	
Description	Set maximum number of Krylov solver iterations.	
Parameters	#	Maximum number of Krylov iterations.
Examples	`-ksp_max_it 20`	

Table 9.218: Petsc ksp_max_it option

KSP_rtol		
Syntax	`-ksp_rtol #`	
Description	Set relative tolerance threshold for convergence (reduction in orders of magnitude).	
Parameters	#	Relative tolerance threshold for convergence.
Examples	`-ksp_rtol 2e-2`	

Table 9.219: Petsc ksp_rtol option

KSP_atol		
Syntax	`-ksp_atol #`	
Description	Set absolute tolerance threshold for convergence (absolute size of residual, use when residuals approach machine accuracy).	
Parameters	#	Aelative tolerance threshold for convergence.
Examples	`-ksp_atol 1e-10`	

Table 9.220: Petsc ksp_atol option

KSP_type	
Syntax	`-ksp_type [type]`
Description	Select Krylov solver (Linear Solvers and Krylov Methods (KSP)).
Types	Selected type list can be found in Table 9.222. For a complete list the user is referred to Petsc relevant user guide page.
Examples	`-ksp_type fgmres`

Table 9.221: Petsc ksp_type option

KSP Types	
Type	Description
gmres	Generalized Minimal Residual method.
fgmres	Flexible Generalized Minimal Residual method.

Table 9.222: Petsc KSP Types

KSP_gmres_restart	
Syntax	`-ksp_gmres_restart #`
Description	Set GMRES restart length (increase to improve convergence on account of memory and cpu time, default: 30).
Parameters	# GMRES restart length.
Examples	`-ksp_gmres_restart 30`

Table 9.223: Petsc ksp_gmres_restart option

PC_type	
Syntax	`-pc_type [type]`
Description	Set pre-conditioner (for additional information the reader is referred to Petsc relevant user guide page).
Types	Selected type list can be found in Table 9.226. For a complete list the user is referred to Petsc list of pre-conditioners.
Examples	`-pc_type asm`

Table 9.224: Petsc pc_type option

Sub_pc_type	
Syntax	`-sub_pc_type [type]`
Description	Select method used to invert the sub-domains created by the additive Schwartz pre-conditioner.
Types	Selected type list can be found in Table 9.226. For a complete list the user is referred to Petsc list of pre-conditioners.
Examples	`-sub_pc_type ilu` `-sub_pc_type preonly`

Table 9.225: Petsc sub_pc_type option

PC Types	
Type	Description
asm	Additive Schwarz.
ilu	Incomplete LU
preonly	Disable Krylov solver in computing inverse of asm sub-domains.

Table 9.226: Petsc PC Types

Sub_pc_factor_levels		
Syntax	`-sub_pc_factor_levels #`	
Description	Set sub-domains pre-conditioner factor level.	
Parameters	#	Sub-domains pre-conditioner factor level.
Examples	`-sub_pc_factor_levels 2`	

Table 9.227: Petsc sub_pc_factor_levels option

Sub_pc_factor_fill		
Syntax	`-sub_pc_factor_fill #`	
Description	Indicate the amount of fill you expect in the factored matrix (for the sbu-domain).	
Parameters	#	Amount of fill you expect in the factored matrix.
Examples	`-sub_pc_factor_fill 1`	

Table 9.228: Petsc sub_pc_factor_fill option

9.15 Domain Options

The Domain input options section starts with the `--domain` directive, followed by a series of domain options. It is used to define a volumetric domain. An example of a simple domain options section is found in Listing 9.17 followed by a description of all available options.

```
--domain
   -name domain
   -type box x.min 1.02 x.max 2.1 y.min -0.205 y.max 0.205 z.min
      -0.345 z.max -0.238
   -init init
```

Listing 9.17: Example of the domain section

Name	
Syntax	`-name $name`
Description	Set the name of the domain.
Parameters	$name Domain name.
Examples	`-name my_domain_name`

Table 9.229: Domain name option

Init	
Syntax	`-init $init_name`
Description	Set the name of the init (see Section 9.16, Table 9.234).
Parameters	$init_name Init name (see Table 9.234).
Examples	`-init my_init_name`

Table 9.230: Domain init option

Type	
Syntax	`-type $type $options`
Description	Set the type of the domain. Currently available types: 'box' and 'cylinder'. The options vary depending on the type. See Table 9.232 for specific options and examples for the 'box' shape and Table 9.233 for specific options and examples for the 'cylinder' shape.
Parameters	$type Domain type. $options Options specific to the type. See Tables 9.232 and Table 9.233 for the available options.
Examples	`-type box x.min 0 x.max 5 y.min -2 y.max 2 z.min -1 z.max 1` `-type cylinder r.min 0.1 r.max 0.7 c1 0 0 0 c2 10 0 0`

Table 9.231: Domain type option

Box		
Syntax	`-type box {[x.min #val] [x.max #val] [y.min #val] [y.max #val] [z.min #val] [z.max #val]}`	
Description	Set a domain of type 'box'. The domain 'box' is a 3-D rectangular cuboid defined by min / max value on each axis. If one axis limits is not defined, then it is assumed to be $+\infty$ or $-\infty$ (depending on the context).	
Parameters	box	3D rectangular cuboid.
	x.min	Set x min.
	x.max	Set x max.
	y.min	Set y min.
	y.max	Set y max.
	z.min	Set z min.
	z.max	Set z max.
	#val	Value.
Examples	`-type box x.min 0 x.max 5 y.min -2 y.max 2 z.min -1 z.max 1`	
	`-type box x.min 0 y.min -3 z.min -5`	

Table 9.232: Domain box type option

Cylinder		
Syntax	`-type cylinder [r.min #val] [r.max #val] [c1 #val.x #val.y #val.z] [c2 #val.x #val.y #val.z]`	
Description	Set a domain of type 'cylinder'. The domain 'cylinder' is a 3D hollow cylinder defined by min radius / max radius (r.min and r.max, respectively) and 2 points of the center of the two bases (c1 and c2, respectively). If r.min is not defined it is assumed to be zero (and then the cylinder is filled rather than hollow). If one of the other limits is not defined, then it is assumed to be $+\infty$ or $-\infty$ (depending on the context).	
Parameters	cylinder	3D hollow cylinder.
	r.min	Set r min.
	r.max	Set r max.
	#val	Set the radius value.
	c1	Set beginning base of cylinder.
	c2	Set ending base of cylinder.
	#x.val	Set x value.
	#y.val	Set y value.
	#z.val	Set z value.
Examples	`-type cylinder r.min 0.1 r.max 0.7 c1 0 0 0 c2 10 0 0`	
	`-type cylinder r.min 0.1 r.max 0.7 c1 0 0 0`	

Table 9.233: Domain cylinder type option

9.16 Init Options

The Init input options section starts with the `--init` directive, followed by a series of domain options. It is used to define a volumetric domain. An example of a simple init options section is found in Listing 9.18 followed by a description of all available options.

```
--init
    -name init
    -u 60
    -p 300000
```

Listing 9.18: Example of the init section

Name	
Syntax	`-name $name`
Description	Set the name of the init.
Parameters	$name Init name.
Examples	`-init my_init_name`

Table 9.234: Init name option

T	
Syntax	`-T #T`
Description	Override the free-stream temperature [K].
Parameters	#T Temperature.
Examples	`-T 300`

Table 9.235: Init T option

P	
Syntax	`-p #p`
Description	Override the free-stream pressure [pa].
Parameters	#p pressure.
Examples	`-p 67000`

Table 9.236: Init p option

U	
Syntax	`-u #u`
Description	Override the free-stream x direction velocity [m/s].
Parameters	#u velocity.
Examples	`-u 150`

Table 9.237: Init u option

V	
Syntax	`-v #v`
Description	Override the free-stream y direction velocity [m/s].
Parameters	#v velocity.
Examples	`-v 0`

Table 9.238: Init v option

W	
Syntax	`-w #w`
Description	Override the free-stream z direction velocity [m/s].
Parameters	#w velocity.
Examples	`-w 10`

Table 9.239: Init w option

Composition	
Syntax	`-composition $composition composition.end`
Description	Override the free-stream mcg composition (for applicable systems) mass fraction (can specify only species fractions that are not zero).
Parameters	$compostion The composition in mass fraction. composition.end End of the composition.
Examples	`-composition O2=0.23, NO=0, N=0, O=0, N2=0.77 composition.end`

Table 9.240: Init composition option

Composition Molar		
Syntax	`-composition molar $composition composition.end`	
Description	Override the free-stream mcg composition (for applicable systems) molar fraction (can specify only species fractions that are not zero).	
Parameters	$compostion	The composition in molar fraction.
	composition.end	End of the composition.
Examples	`-composition molar O2=0.21, N2=0.79 composition.end`	

Table 9.241: Init composition molar option

9.17 Library Options

The libraries input options section starts with the `--lib` directive, followed by a series of library options. The libraries option is used to dynamically load a library.

Name		
Syntax	`-name $name`	
Description	Set the name of the library.	
Parameters	$name	Library name.
Examples	`-name my_lib`	

Table 9.242: Libraries name option

Prefix		
Syntax	`-prefix $path`	
Description	Set the full library path.	
Parameters	$path	Full path to the library.
Examples	`-prefix /users/my_user/my_path/lib.so` `-prefix ./lib.so`	

Table 9.243: Libraries prefix option

9.18 User Defined Source Options

The user source section starts with the `--user.source` directive, followed by a series of user.source options. The user.source directive is used to define source terms by the user. It must be accompanied by a `-user.source` option under a certain system section (see Section 9.12). An detailed example of how the user defined source options are utilized is given in Section 9.29.

Name	
Syntax	`-name $name`
Description	Set the user source object name.
Parameters	$name Object name.
Examples	`-name heat_source`

Table 9.244: User source name option

Lib	
Syntax	`-lib $library_name`
Description	Name of the library object that contains the user source implementation (see Section 9.17 for the library options).
Parameters	$library_name Library name.
Examples	`-lib heat_source_lib`

Table 9.245: User source lib option

Ctor	
Syntax	`-ctor $name`
Description	Name of the constructor callback function in the library.
Parameters	$ctor Constructor name.
Examples	`-ctor my_ctor`

Table 9.246: User source ctor option

Dtor	
Syntax	`-dtor $name`
Description	Name of the destructor callback function in the library.
Parameters	$ctor Destructor name.
Examples	`-dtor my_dtor`

Table 9.247: User source dtor option

RHS	
Syntax	`-rhs $name`
Description	Name of the RHS callback function in the library.
Parameters	$name RHS function name.
Examples	`-rhs my_rhs`

Table 9.248: User source rhs option

LHS	
Syntax	`-lhs $name`
Description	Name of the optional LHS/Jacobian callback function in the library.
Parameters	$name LHS function name.
Examples	`-lhs my_lhs`

Table 9.249: User source lhs option

9.19 Var Options

The definition of a new function is conducted using the `--var` directive, followed by a series of var options. The `--var` directive is useful where there is a need for a varying scalar function (time or space dependent). The options are a constant function, a linear function, a trigonometric function, and a tabulated function. Table 9.250 describes the usage of the directive and its options. See Section 9.19.1 for examples and a detailed explanation of the usage.

var		
Syntax	`--var {[-name] $name} [id #id] {[const] #c | linear.({t|x|y|z}) #a #b | {sin|cos|tanh}.({t|x|y|z}) #a #b #c #d | table.{t|x|y|z} [#x #y]... table.end}`	
Description	Add a function.	
Parameters	$name	var name.
	#id	Identification number (for internal use).
	const	Constant function.
	linear	Linear function.
	sin, cos, tanh	Trigonometric functions.
	#a, #b, #c, #d	Constants.
	t, x, y, z	Independent variables.
Examples	`--var -name new_const const 1`	
	`--var -name linear_function linear.t 2 -1`	
	`--var -name trigonometric_function tanh.t 3 5 7 9`	
	`--var pressure table.z 0 101325 1000 89874.6 2000 79495.2 table.end`	
	See Section 9.19.1 for more examples and a detailed explanation of the usage.	

Table 9.250: Var directive and its options

9.19.1 Var Examples

9.19.1.1 Constant Function

Directive	Function
`--var const_function const 5`	const_function = 5

9.19.1.2 Linear Function

Directive	Function
`--var -name line linear.t 1 2`	line.t = 1 + 2 * t

9.19.1.3 Trigonometric Function

Directive	Function
`--var -name trig sin.x 1 2 3 4`	trig(x) = 1 + 2 * [sin(3x) + 4]

9.19.1.4 Tabulated Function

A tabulated function assumes linear variation between table entries. The directive: `--var -name my.table table.y 0 4 1 3 2 0 3 -6 table.end` translates into the table [y, my.table(y)] whose entries are: (0, 4), (1, 3), (2, 0), (3 ,-6).

9.20 Vertex Option

The definition of a vertex is conducted using the `--vertex` directive, followed by a series of vertex options. Table 9.251 describes the usage of the directive.

vertex	
Syntax	`--vertex {-name $name} {[-coordinates] #x #y #z} [-moves.with] [-coordinates.of]`
Description	Add a 3D vertex. The coordinates can be either in observer frame or a certain UNS name. The vertex can move along with a UNS. Vertex that moves with a UNS can be defined directly from a UNS but they can not be referenced outside the UNS.
Parameters	$name Vertex name.
Examples	`--vertex -name nozzle -coordinates 0.4 0 0`

Table 9.251: Vertex option

9.21 UNS Options

The UNS input options section starts with the `--uns` directive, followed by a series of uns options. An example of a simple uns options section is found in Listing 9.19 followed by a description of all available options.

```
--uns
    -name airfoil
    -prefix.cgns ../grid/phdp_uns_1em4
    -log convergence cl cd cmy
```

Listing 9.19: Example of the uns section

Name		
Syntax	`-name $name`	
Description	Name of flow case.	
Parameters	$name	Flow case name.
Examples	`-name naca0012_laminar`	

Table 9.252: UNS name option

Prefix		
Syntax	`{-prefix | -prefix.starcd | -prefix.cgns} $prefix`	
Description	Set prefix of the flow case.	
Parameters	prefix	Used for manual conversion (conversion and decomposition has been conducted using star2metis or cgns2metis).
	prefix.starcd	Use original Star-CD export grid files (conversion and decomposition is conducted).
	prefix.cgns	Use original CGNS export grid files (conversion and decomposition is conducted).
	$prefix	Prefix of flow case.
Examples	`-prefix ./naca0012/NACA0012-Str-Laminar-Rey500`	

Table 9.253: UNS prefix option

Block		
Syntax	`-block #block`	
Description	Select block from within a multi-block Stat-CD file.	
Parameters	#block	Star-CD file block number.
Examples	`-block 2`	

Table 9.254: UNS block option

Base	
Syntax	`-base #base`
Description	Use only the specified base within the CGNS file. The user is referred to the CGNS format for further details.
Parameters	#base Base number.
Examples	`-base 1`

Table 9.255: UNS base option

Zone	
Syntax	`-zone #zone`
Description	Use only the specified zone within the CGNS file. The user is referred to the CGNS format for further details.
Parameters	#zone Zone number.
Examples	`-zone 5`

Table 9.256: UNS zone option

CRC	
Syntax	`-crc $signature`
Description	Use an explicit signature.
Parameters	#signature Signature.
Examples	`-crc`

Table 9.257: UNS crc option

Key	
Syntax	`-key $prefix`
Description	Save/load a key file. The file contains pre-calculated wall distance. If the file exists, the code reads it prior to execution. If the file does not exist, the code generates one.
Parameters	$prefix Prefix of the key file.
Examples	`-key ./NACA0012-Str-Laminar-Rey500`

Table 9.258: UNS key option

Geometric.tolerance	
Syntax	`-geometric.tolerance #tolerance`
Description	Geometric tolerance where needed (overrides global values).
Parameters	#tolerance Set the tolerance.
Examples	`-geometric.tolerance 2e-7`

Table 9.259: UNS geometric.tolerance option

Offset		
Syntax	`-offset #x #y #z`	
Description	Offset the grid.	
Parameters	#x	Offset in the x direction.
	#y	Offset in the y direction.
	#z	Offset in the z direction.
Examples	`-offset 5 0 0`	

Table 9.260: UNS offset option

Scale		
Syntax	`-scale #x #y #z`	
Description	Scale the grid.	
Parameters	#x	Scale in the x direction.
	#y	Scale in the y direction.
	#z	Scale in the z direction.
Examples	`-scale 0.001 0.001 0.001`	

Table 9.261: UNS scale option

Rotate [rad]	
Syntax	`-rotate.{x|y|z}.rad #angle`
Description	Rotate the grid around an axis at an angle [rad] (can concatenate rotations) .
Parameters	#angle Rotation angle [rad].
Examples	`-rotate.x.rad 0.1` `-rotate.y.rad 0.1`

Table 9.262: UNS rotate option [rad]

Rotate [deg]	
Syntax	`-rotate.{x|y|z}.deg #angle`
Description	Rotate the grid around an axis at an angle [deg] (can concatenate rotations).
Parameters	#angle Rotation angle [deg].
Examples	`-rotate.x.deg 10` `-rotate.z.deg -10`

Table 9.263: UNS rotate option [deg]

Vertex	
Syntax	`{-<o>vertex | -vertex<o> | -vertex<o>:m | -<b>vertex | -vertex<b> | -vertex<b>:m} $name #x #y #z`
Description	Set named vertex in observer frame coordinates (<o>) or body frame coordinates (<b>) that moves along with the uns. The uns vertices are guaranteed to be 'known' iff they are defined prior to be referenced.
Parameters	$name — Vertex name. #x, #y, #z — Vertex coordinates.
Examples	`-<o>vertex v1 0 0 0`

Table 9.264: UNS vertex option

Log	
Syntax	`-log $log [log-options] ...`
Description	Report in log file. See Tables 9.266 and 9.271 for log options list.
Parameters	$log — Log name.
Examples	`-log coefficients cl cd`

Table 9.265: UNS log option

UNS Log Options

Log type	Description
wet.surface	Wet surface area.
p.center.sum	Center of pressure.
p.center.x	Center of pressure x coordinate.
p.center.y	Center of pressure x coordinate.
p.center.z	Center of pressure x coordinate.
p.center.xFy	Center of pressure x coordinate.
p.center.xFz	Center of pressure x coordinate.
p.center.yFx	Pressure center sum(y * dFx) / Fx coordinate.
p.center.yFz	Pressure center sum(y * dFz) / Fz coordinate.
p.center.zFx	Pressure center sum(z * dFx) / Fx coordinate.
p.center.zFy	Pressure center sum(z * dFy) / Fy coordinate .
fx	X coordinate direction force.
fy	Y coordinate direction force.
fz	Z coordinate direction force.
fx.pressure	Force in x direction due to pressure.
fy.pressure	Force in y direction due to pressure.
fz.pressure	Force in z direction due to pressure.
fx.friction	Force in x direction due to friction.
fy.friction	Force in y direction due to friction.
fz.friction	Force in z direction due to friction.
lift	Lift force.
drag	Drag force.

mass.flow	Mass flow rate.
mx	Moment about X coordinate direction.
my	Moment about Y coordinate direction.
mz	Moment about Z coordinate direction.
cfx	X coordinate direction force coefficient.
cfy	Y coordinate direction force coefficient.
cfz	Z coordinate direction force coefficient.
mx.pressure	Adds moments around x axis (of this bc) induced by pressure.
my.pressure	Adds moments around y axis (of this bc) induced by pressure.
mz.pressure	Adds moments around z axis (of this bc) induced by pressure.
mx.friction	Adds moments around x axis (of this bc) induced by friction.
my.friction	Adds moments around y axis (of this bc) induced by friction.
mz.friction	Adds moments around z axis (of this bc) induced by friction.
cfx.pressure	Adds force coefficient in x direction (of this bc) induced by pressure.
cfy.pressure	Adds force coefficient in y direction (of this bc) induced by pressure.
cfz.pressure	Adds force coefficient in z direction (of this bc) induced by pressure.
cfx.friction	Adds force coefficient in x direction (of this bc) induced by friction.

cfy.friction	Adds force coefficient in y direction (of this bc) induced by friction.
cfz.friction	Adds force coefficient in z direction (of this bc) induced by friction.
cmx.pressure	Adds moments coefficient around x axis (of this bc) induced by pressure.
cmy.pressure	Adds moments coefficient around y axis (of this bc) induced by pressure.
cmz.pressure	Adds moments coefficient around z axis (of this bc) induced by pressure.
cmx.friction	Adds moments coefficient around x axis (of this bc) induced by friction.
cmy.friction	Adds moments coefficient around y axis (of this bc) induced by friction.
cmz.friction	Adds moments coefficient around z axis (of this bc) induced by friction.
cmx	Moment coefficient about X coordinate direction.
cmy	Moment coefficient about Y coordinate direction.
cmz	Moment coefficient about Z coordinate direction.
cl	Lift coefficient.
cd	Drag coefficient.

Table 9.266: UNS log options

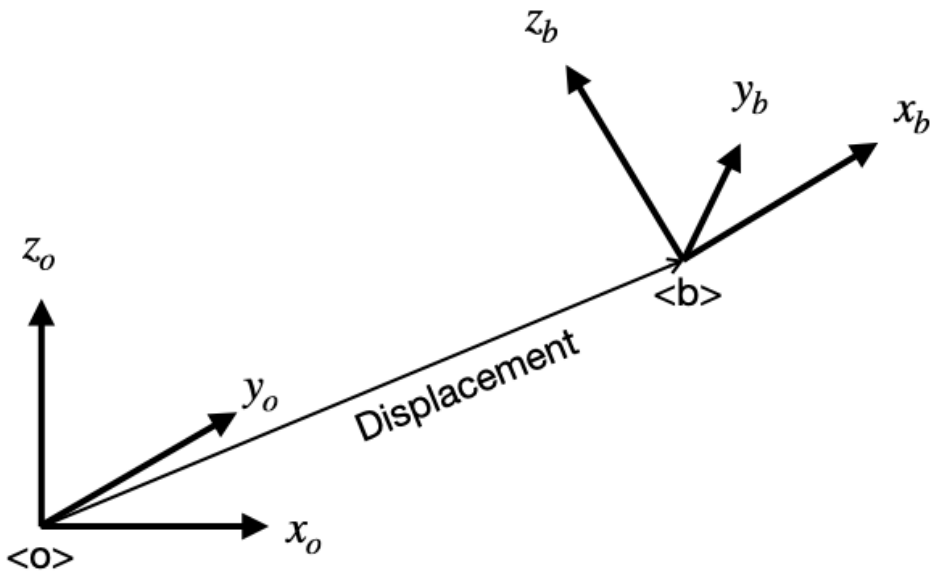


Figure 9.1: Observer and body reference frames

9.21.1 Motion Options

This section describes the directive to control motion. The current version of **Arion** supports the motion that is controlled by the equations of motion resulting from aerodynamics, inertial, and prescribed forces and moments acting on a certain body. A body is defined as the collection of surfaces in a certain UNS. The equations of motion can be overridden by the use of explicit values for acceleration, velocity, and position. The motion parameters are controlled using the `--var` directive (it is strongly recommended to use the `--var` directive, see Section 9.19). Using `--var` enables first and second derivatives. Therefore, setting the position using vars automatically sets the velocity and acceleration. In what follows a certain terminology is adopted. For example the option "<o>initial.body.origin:m" is translated as follows: The '<o>' refers to observer reference frame ('<b>' refers to body reference frame, '<p>' refers to "parent" reference frame), 'initial.body.origin' is the actual option and ':m' refers to the dimensions (if SI system is used it can be omitted). Note that '<o/cg>' refers to the center of gravity in the observer reference frame. Figure 9.1 shows the observer and body reference frames.

Initial Position		
Syntax	`-dof <o>initial.body.origin:m #x #y #z`	
Description	Set the initial body frame origin in observer frame, it does NOT move the grid, just point to where the body axes origin are (start here).	
Parameters	#x #y #z	Origin of the body reference frame in observer frame.
Examples	`-dof <o>initial.body.origin:m 5.5 0 0`	

Table 9.267: UNS DOF initial position option

Initial Orientation		
Syntax	`-dof {<o>initial.body.orientation:rad | <o>initial.body.orientation:deg} #`ϕ `#`θ `#`ψ	
Description	Set the initial body frame axes orientation in observer frame, it does NOT move the grid nor its orientation, it just sets the axes.	
Parameters	$\#\phi$ $\#\theta$ $\#\psi$	Origin of the body reference frame orientation in observer frame.
Examples	`-dof <o>initial.body.orientation:deg 45 90 45`	

Table 9.268: UNS DOF initial orientation option

From/Until		
Syntax	`-dof [from #t] [until #t] {dof instruction}`	
Description	Declare dof order, where from and until indicate the time span of the order effectiveness.	
Parameters	#t	Starting or ending time.
	instruction	Instruction, see Table 9.270 for a list of supported instructions.
Examples	`-dof from 0.1 instruction` (from Table 9.270)	
	`-dof until 0.6 instruction` (from Table 9.270)	
	`-dof from 0.2 until 0.4 instruction` (from Table 9.270)	

Table 9.269: UNS DOF from/until option

DOF-Instructions

Center of gravity	Description
<b>cg:m #x #y #z	Center of gravity in body reference frame where #x, #y, and #z are the coordinates thereof.
Mass and Inertia	**Description**
<o>mass:kg #m	Set the body mass where #m is the mass. Acts at the center of gravity.
{<b>inertia:kg*sq.m | <o/cg>inertia:kg*sq.m} #Ixx #Ixy #Ixy #Iyy #Iyz #Izz	Moment of inertia tensor where #Ixx, #Ixy, #Ixy, #Iyy, #Iyz, and #Izz are the inertia tensor components.
Displacement, Velocity, and Acceleration	**Description**
<p>displacement:m #dx #dy #dz	Override displacement where #dx #dy, and #dz are the displacement components.
<p>velocity:m/s #u #v #w	Override velocity where #u, #v, and #w are the velocity components.
<p>acceleration:m/sq.s #u.dot #v.dot #w.dot	Override the acceleration where #u.dot, #v.dot, and #w.dot are the acceleration vector components.

Angular orientation, velocity, and acceleration	Description
{<p>euler.angle:rad | <p>euler.angle:deg } #ϕ #θ #ψ	Override the initial Euler angles where #ϕ, #θ, and #ψ are the angles.
{<b>angular.velocity:rad/sec | <b>angular.velocity:deg/sec } #p #q #r	Override angular velocity where #p, #q, and #r are the angular velocity components.
{<b>angular.acceleration:rad/sq.sec | <b>angular.acceleration:deg/sq.sec } #pdot #qdot #rdot	Override angular acceleration where #pdot, #qdot, and #rdot are the angular acceleration components.
Forces and Moments	**Description**
<b>explicit.force:N #fx #fy #fz {$vertex | {#x #y #z}}	Set an explicit force in body reference frame where #fx, #fy, and #fz are the force components and $vertex or #x, #y, and #z is the point of action
<b>explicit.pure.moment:Nm #Mx #My #Mz	Set an explicit moment where #Mx, #My, and #Mz are the component components.
aerodynamic.force	Add all of the aerodynamic forces of current uns.
aerodynamic.force.of $uns $bc	Add the aerodynamic forces of certain $bc of a certain $uns.

<o>weight	Add weight of current uns.
<o>weight.of $uns	Add weight of a certain $uns.

Table 9.270: DOF instrcutions

DOF Log Options

Log type	Description
<p>dof.phi	ϕ-euler angle in parent frame.
<p>dof.theta	θ-euler angle in parent frame.
<p>dof.psi	ψ-euler angle in parent frame.
<b>dof.p	p - angular velocity in body frame.
<b>dof.q	q - angular velocity in body frame.
<b>dof.r	r - angular velocity in body frame.
<b>dof.p.dot	dp/dt - angular acceleration in body frame.
<b>dof.q.dot	dq/dt - angular acceleration in body frame.
<b>dof.r.dot	dr/dt - angular acceleration in body frame.
dof.all.angular	All angualr information.
<p>dof.dx	x - translational offset in parent frame.

<p>dof.dy	y - translational offset in parent frame.
<p>dof.dz	z - translational offset in parent frame.
<p>dof.u	u - translational velocity in observer frame.
<p>dof.v	v - translational velocity in observer frame.
<p>dof.w	w - translational velocity in observer frame.
<p>dof.u.dot	du/dt - translational acceleration in observer frame.
<p>dof.v.dot	dv/dt - translational acceleration in observer frame.
<p>dof.w.dot	dw/dt - translational acceleration in observer frame.
dof.all.translational	All translational information.
<b>dof.cg, <b>dof.cg.x, <b>dof.cg.y, <b>dof.cg.z	Center of gravity in body frame.
<o>dof.cg, <o>dof.cg.x, <o>dof.cg.y, <o>dof.cg.z	Center of gravity in observer frame.
<o>dof.mass	Mass.

<b>inertia, <b>ixx, <b>ixy, <b>ixz, <b>iyy, <b>iyz, <b>izz	Inertia tensor.
dof.all.inertia	All mass and inertia information.
<o>F, <p>F, <b>F	Force summation either in observer frame, parent frame, or body frame.
<b>M, <o>M	Moment summation either in body frame, or observer frame.
dof.all.force	All forces information.

Table 9.271: DOF log options

9.22 Point Cloud Data

The point cloud data input options section starts with the `--pcd` directive, followed by a series of point cloud data options. This section describes the point cloud data options. A simple example of the point cloud section is brought in Listing 9.20. The reader is referred to Section 6.3 for a brief description of the adaptation method that is implemented in the **Arion** code.

```
--pcd
  -name mach-sensor
  -prefix mach-cycle-0
  -mach
  -power 1
  -k 1
  -percentage 2.5
  -decay 0.3
  -spacing 0.5
```

Listing 9.20: Example of a pcd section

Name		
Syntax	`-name $name`	
Description	Set the name of the point cloud data.	
Parameters	$name	Point cloud data name.
Examples	`-name pcd1`	

Table 9.272: PCD name option

Prefix	
Syntax	`-prefix $prefix`
Description	The point cloud data plot-file will be saved using the path prefix with '.fvuns' suffix.
Parameters	$prefix Point cloud data prefix.
Examples	`-prefix ./pcfd/pcd`

Table 9.273: PCD prefix option

Mach	
Syntax	`-mach`
Description	Set the grid adaptation criterion to Mach number.
Parameters	N/A
Examples	`-mach`

Table 9.274: PCD mach option

Pressure	
Syntax	`-pressure`
Description	Set the grid adaptation criterion to pressure.
Parameters	N/A
Examples	`-pressure`

Table 9.275: PCD pressure option

Density	
Syntax	`-density`
Description	Set the grid adaptation criterion to density.
Parameters	N/A
Examples	`-density`

Table 9.276: PCD density option

Velocity.magnitude	
Syntax	`-velocity.magnitude`
Description	Set the grid adaptation criterion to velocity magnitude.
Parameters	N/A
Examples	`-velocity.magnitude`

Table 9.277: PCD velocity.magnitude option

Power	
Syntax	`-power #power`
Description	Set the sensor power value (p). See Equation 6.1.
Parameters	#power Sensor power value.
Examples	`-power 1`

Table 9.278: PCD power option

K	
Syntax	`-k #k`
Description	Set the sensor constant (K). See Equation 6.1.
Parameters	#k Sensor constant.
Examples	`-k 1`

Table 9.279: PCD K option

Percentage	
Syntax	`-percentage #percentage`
Description	Set the refinement percentage (refine the top percentage% of the cells according to the value of sensor).
Parameters	#percentage Percentage of refined cells.
Examples	`-percentage 2`

Table 9.280: PCD percentage option

Spacing	
Syntax	`-spacing #spacing`
Description	Set the spacing.
Parameters	#decay Spacing (used by Pointwise to refine the mesh).
Examples	`-spacing 0.5`

Table 9.281: PCD spacing option

Decay	
Syntax	`-decay #decay`
Description	Set the decay.
Parameters	#decay Decay (used by Pointwise to refine the mesh).
Examples	`-decay 0.3`

Table 9.282: PCD decay option

9.23 Log Control Options

Log control options are used to set up the log outputs. For example, the options are used to set up the log file name. The log control options section starts with the `--log` directive, followed by a series of options that are described in the following tables. It is important to note that the number of logs (and thus generated log files) is unlimited. An example of a simple log options section is found in Listing 9.21 followed by a description of all available options.

```
--log
   -name convergence
   -prefix ./logs/convergence
   -per iter
   -screen
```

Listing 9.21: Example of the log section

Name		
Syntax	`-name $name`	
Description	Set the name of the log. This name is referred to by the specific `-log` options. It can be the predefined keywords: 'stdout', 'screen', or 'display': where it will be only shown on the screen. It can also be: 'stderr' where it will be only directed to the standard error stream.	
Parameters	$name	Log name.
Examples	`-name convergence`	

Table 9.283: Log name option

Prefix	
Syntax	`-prefix $prefix`
Description	The log file path and name prefix. The '.log' suffix is added to obtain the actual log file name.
Parameters	$prefix Prefix.
Examples	`-prefix ./logs/convergence`

Table 9.284: Log prefix option

Per	
Syntax	`-per $type`
Description	Set when the log is printed based on the $type. The possibilities are: iter, produced every iteration (virtual time step); step, produced every step (physical time step); or pos, produced every 'pos' (depending on the simulation type).
Parameters	$type Type (iter, step, or pos).
Examples	`-per pos`

Table 9.285: Log per option

Stdout	
Syntax	`-stdout` | `-screen` | `-display`
Description	Adds echo to the standard output as well as to the log file. One can also use the -screen or -display (the outcome is exactly the same).
Parameters	N/A
Examples	`-screen`

Table 9.286: Log stdout option

Stderr	
Syntax	`-stderr`
Description	Adds echo to the standard error as well as to the log file.
Parameters	N/A
Examples	`-stderr`

Table 9.287: Log stderr option

9.24 Plot Control Options

Plot control options are used to set up the plot outputs. The plot control options section starts with the `--plot` directive, followed by a series of options that are described in the following tables. An example of a simple plot options section is found in Listing 9.22 followed by a description of all available options.

```
--plot
   -name plot
   -prefix ./restart/plot
   -interval 100
   -overwrite
   -cgns
```

Listing 9.22: Example of the plot section

Name		
Syntax	`-name $name`	
Description	Set the name of the plot. This name is referred to by the specific `-plot` options.	
Parameters	$name	Plot name.
Examples	`-name Alpha10`	

Table 9.288: Plot name option

Prefix	
Syntax	`-prefix $prefix`
Description	The plot file path and name prefix. The actual file number is composed from the prefix, and the $name as described in Table 9.252. The '.fvuns' or '.cgns' suffix is added to the plot file name. If the computational domain has been decomposed, the files are split as well and the number of the partition is added to the file name.
Parameters	$prefix Prefix.
Examples	`-prefix ./plots/`

Table 9.289: Plot prefix option

Interval	
Syntax	`-interval #interval`
Description	Set the interval (of iterations/steps) to create the plot files.
Parameters	#interval Plot interval.
Examples	`-interval 100`

Table 9.290: Plot interval option

Sequential Overwrite	
Syntax	`-sequential` | `-overwrite`
Description	Include position stamp in the plot files (or overwrite, without position stamp, default).
Parameters	N/A
Examples	`-sequential`

Table 9.291: Plot sequential/overwrite option

SIDS.Names	
Syntax	`-sids.names`
Description	Use SIDS-standard variable names for output variables (*e.g.*, 'Density', 'VelocityX' etc.).
Parameters	N/A
Examples	`-sids.names`

Table 9.292: Plot sids.names option

FVUNS/CGNS	
Syntax	`-fvuns | -cgns`
Description	Select the file format to use, default is: -cgns.
Parameters	N/A
Examples	`-cgns` `-fvuns`

Table 9.293: Plot fvuns/cgns option

BC		
Syntax	`-bc {walls | all}`	
Description	Include only boundaries in the cgns file (not applicable for fvuns files).	
Parameters	walls	Add only boundaries that are designated as walls to plot file.
	all	Add all boundaries to plot file.
Examples	`-bc wall` `-bc all`	

Table 9.294: Plot bc option

9.25 Table Control Options

Table control options are used to set up the table outputs. The table control options section starts with the `--table` directive, followed by a series of options that are described in the following tables. An example of a simple table options section is found in Listing 9.23 followed by a description of all available options.

```
--table
   -name table
   -prefix ./tables/table
   -interval 1000
   -plane.bc 0 -1e-4 0 0 -1 0 Wall
```

Listing 9.23: Example of the table section

Name		
Syntax	`-name $name`	
Description	Set the name of the table. This name is referred to by the specific `-table` options.	
Parameters	$name	Table name.
Examples	`-name Alpha10`	

Table 9.295: Table name option

Prefix	
Syntax	`-prefix $prefix`
Description	The table file path and name prefix. The '.tbl' suffix is added to obtain the actual table file name.
Parameters	$prefix Prefix.
Examples	`-prefix ./tables/surface-pressure`

Table 9.296: Table prefix option

Interval	
Syntax	`-interval #interval`
Description	Set the interval (of iterations/steps) to create the table files.
Parameters	#interval Table interval.
Examples	`-interval 100`

Table 9.297: Table interval option

Sequential/overwrite	
Syntax	`-sequential` | `-overwrite`
Description	Include position stamp in the table files (or overwrite, without position stamp, default).
Parameters	N/A
Examples	`-sequential`

Table 9.298: Table sequential/overwrite option

Horizontal/vertical	
Syntax	`-horizontal` |`-vertical`
Description	The table type (vertical is the default).
Parameters	N/A
Examples	`-horizontal`

Table 9.299: Table horizontal/vertical option

Loc Face	
Syntax	`-loc.face`
Description	Output table values at face centers (default is at grid nodes).
Parameters	N/A
Examples	`-loc.face`

Table 9.300: Table loc face option

Point	
Syntax	`-point #x #y #z`
Description	Create a table from a point where [x, y, z] is the point.
Parameters	#x #y #z Point.
Examples	`-point 0 0 0`

Table 9.301: Table point option

Ray		
Syntax	`-ray #x #y #z #vx #vy #vz`	
Description	Create a table along a predefined ray.	
Parameters	#x #y #z	Ray origin.
	#vx #vy #vz	Ray direction.
Examples	`-ray 0 0 0 1 1 1`	

Table 9.302: Table ray option

Plane		
Syntax	`-plane #x #y #z #nx #ny #nz`	
Description	Create a table from a plane where [x, y, z] is a point on the plane and [nx, ny, nz] is the normal vector to the plane.	
Parameters	#x #y #z	Plane origin.
	#nx #ny #nz	Plane normal direction.
Examples	`-plane 0 0 0 0 0 1`	

Table 9.303: Table plane option

BC		
Syntax	`-bc $bc`	
Description	Create the table from a boundary condition.	
Parameters	$bc	Boundary condition name.
Examples	`-bc STRUT`	

Table 9.304: Table bc option

Plane.bc		
Syntax	`-plane.bc #x #y #z #nx #ny #nz $bc-name`	
Description	Create a table along an intersection between a plane and a boundary surface.	
Parameters	#x #y #z	Plane.bc origin.
	#nx #ny #nz	Plane.bc normal direction.
	$bc-name	The boundary surface name as assigned by the `--bc -name` directive (see Section 9.11, Table 9.86).
Examples	`-plane.bc 0 0 0 0 1 0 WALL`	

Table 9.305: Table plane.bc option

9.26 Post Option

The post directive is utilized to read a solution and output all logs, plots, and tables without any flow calculations. The directive can be simply added to the input file. See Table 9.306 for the syntax.

Post	
Syntax	`--post`
Description	Output all the logs, plots, and tables and quit.
Parameters	N/A.
Examples	`--post`

Table 9.306: Post directive

9.27 Converge Options

The converge input options section starts with the `--converge` directive, followed by the `-name` option. Once a convergence option has been named, it is used within the `--solve` (see Section 9.13), `--uns` (see Section 9.21) , or `--bc` sections (see Section 9.11). See Table 9.308 for the syntax. The options that are set in the relevant sections are used to determine simulation convergence other than residual convergence. To achieve convergence, **all** of the criteria of a certain section must be met. Multiple criteria convergence sections are evaluated independently. If one of the criteria convergence sections is met, the simulation is considered converged. For iteration based convergence only iteration duration is considered. Likewise, for time-accurate convergence, only time-span is considered.

Name		
Syntax	`-name $name`	
Description	Set the convergence criterion name.	
Parameters	$name	Name of the convergence criterion.
Examples	`-name coefficients`	

Table 9.307: Converge name option

Converge		
Syntax	`-converge $name $criterion {{amplitude.minmax #min #max} | {amplitude.range #range}} {{duration.i #iters} | {duration.t #time-span}}`	
Description	Set a convergence criterion.	
Parameters	$name	Name of the convergence criterion.
	$criterion	Convergence criterion type (see Table 9.309 for criteria types).
	amplitude.minmax	Set minimum and maximum amplitude.
	#min	Minimum amplitude.
	#max	Maximum amplitude.
	amplitude.range	Set amplitude range of convergence criterion.
	#range	Amplitude range of convergence criterion.
	duration.i	Set the number of consecutive iterations that meet the criterion.
	#iters	Number of consecutive iterations to meet the criterion.
	duration.t	Set the time span that meets the criterion (for time-accurate simulations).
	#time-span	Time span to meet the criterion.

Examples	`-converge coefficients cl amplitude.range 0.01 duration.i 1000`
	
	`-converge coefficients cd amplitude.range 0.001 duration.i 1000`	

Table 9.308: Converge converge option



Convergence Criteria Types

Convergence Criterion Type	Description (see Table 9.308)
f	Total force.
fx	Force in the x coordinate direction.
fy	Force in the y coordinate direction.
fz	Force in the z coordinate direction.
lift	Lift force.
drag	Drag force.
cf	Total force coefficient.
cfx	Force coefficient in the x coordinate direction.
cfy	Force coefficient in the y coordinate direction.
cfz	Force coefficient in the z coordinate direction.
cl	Lift coefficient.
cd	Drag coefficient.
m	Moment magnitude.
mx	Moment about the x coordinate direction.
my	Moment about the y coordinate direction.
mz	Moment about the z coordinate direction.
cm	Moment coefficient.
cmx	Moment coefficient about the x coordinate direction.
cmy	Moment coefficient about the y coordinate direction.
cmz	Moment coefficient about the z coordinate direction.

Table 9.309: Convergence criteria types



9.28 Run-time Options

Arion provides the capability to control the run while the application is running using the run-time options. This can be achieved by creating an ascii text file named 'arion.ins' containing one of the following run-time options:

9.28.1 Run-time Options Examples

The examples in Table 9.311 provide the complete Unix/Linux command to create the 'arion.ins' file with the required content. Alternatively, the user may use an editor to create the file and enter the content. Note, once **Arion** identifies the existence of the file 'arion.ins' and reads its content, the file is erased.

Run-time Options

Option	Action
stop	Stop and save a restart.
kill	Immediately stop the run (even in dual-time mode) and save a restart.
plot	Dump the FVUNS or CGNS file as requested by the user at the next available opportunity.
table	Dump all tables requested by the user.
save	Save a restart.
time.step [$system] #cfl	Change to the specified constant CFL or virtual time step of all the systems or the specified $system.
dt #dt	Change to the specified physical time step (for dual time simulations).
iterations #iterations	Change the maximum number of iterations.
iteration.convergence #convergence	Change convergence criterion.
iteration.convergence $system #convergence	Change the convergence criterion of a systems.
steps #steps	Change the number of steps.

Table 9.310: Run-time options

Run-time Options Examples

Command	Result
echo stop > arion.ins	Stop and save a restart.
echo plot > arion.ins	Dump the FVUNS or CGNS file as requested by the user at the next available opportunity.
echo save > arion.ins	Save a restart.
echo time.step 50.0 > arion.ins	Change the CFL or time step to 50.0 (all systems).
echo time.step turbulent.kw.tnt 30.0 > arion.ins	Change the CFL or virtual time step for the k-ω-TNT turbulence model equations to 30.0.
echo dt 0.001	Change the physical time step to 0.001

Table 9.311: Run-time options examples

9.29 User Source Library Interface

The **Arion** solver supports user-defined source terms through an external dynamically loaded shared library. This mechanism allows the user to add custom source terms to the governing equations without recompiling the CFD solver. The required user-source interface files, including `user_source.h`, are supplied with the Arion release files.

A user source may contribute to the residual right-hand side, and may optionally provide a local Jacobian contribution for implicit source treatment.

9.29.1 Input Definition

A user source is configured in three steps:

1. Define the user source object using a `--user.source` block (see Section 9.18).
2. Define the shared library using a `--lib` block (see Section 9.17).
3. Attach the user source to a solver system using the `-user.source` option inside the relevant `--system` block (see Section 9.12).

The `--user.source` block defines the source object and the names of the functions exported by the user library:

```
--user.source
   -name heat_source
   -lib heatlib
   -ctor ctor
   -dtor dtor
   -rhs source
   -lhs jac
```

Listing 9.24: Example of a simple **Arion** user source section

The `--lib` block defines the dynamic library that is used by the source:

```
--lib
    -name  heatlib
    -prefix  ./heat.so
```

The source is then attached to a system using the `-user.source` option:

```
--system
    -user.source  heat_source
```

The name that is used by `-user.source` must match a previously defined `--user.source -name`. The library name that is used by `--user.source -lib` must match a previously defined `--lib -name`.

Several user sources may be attached to the same system:

```
--system
    -user.source  heat_source_1
    -user.source  heat_source_2
    -user.source  heat_source_3
```

When several user sources are attached to the same system, they are applied in the same order in which they appear under the `--system` block.

9.29.2 Exported Library Functions

The shared library must export C functions with the signatures expected by the solver.

The constructor is called once during source initialization:

```
int ctor(void** user_ctx, const struct user_source_api_s* api);
```

The destructor is called when the source is destroyed:

```
int dtor(void* user_ctx);
```

The RHS source callback is called during residual evaluation:

User Source Input Options	
Option	Description
`--user.source`	Begins a user-source definition block
`-name`	Name of the user source object
`-lib`	Name of the library object used by this source
`-ctor`	Constructor function exported by the library
`-dtor`	Destructor function exported by the library
`-rhs`	RHS source function exported by the library
`-lhs`	Optional Jacobian/LHS function exported by the library

Table 9.312: User source input options

Library Input Options	
Option	Description
`--lib`	Begins a dynamic-library definition block
`-name`	Name of the library object
`-prefix`	Path to the shared library file

Table 9.313: Library input options

System Connection Option	
Option	Description
`-user.source`	Adds a previously defined user source to the current system

Table 9.314: System connection option

```
int source(
    void* user_ctx,
    const struct user_source_sim_s* sim,
    uint32_t cell,
    const real_t* const vars[],
    real_t* rhs);
```

The Jacobian callback is optional:

```
int jac(
    void* user_ctx,
    const struct user_source_sim_s* sim,
    uint32_t cell,
    const real_t* const vars[],
    real_t* J);
```

The callback functions return nonzero on success and zero on failure. If any callback returns zero, the solver treats this as a fatal user-source error and aborts the current run. The solver does not silently skip the source, retry the callback, or replace the contribution by zero.

9.29.3 Runtime Flow

The user source is initialized and used in the following order:

1. The solver loads the shared library.
2. The solver resolves the exported symbols, such as `ctor`, `dtor`, `source`, and optionally `jac`.
3. The solver fills the API structure, including `api->abi_version`.
4. The solver calls the user constructor `ctor()`.
5. Inside `ctor()`, the user library checks `api->abi_version` against `USER_SOURCE_ABI_VERSION`.
6. Inside `ctor()`, the user library requests variable and equation indices by name.
7. The solver dynamically registers the requested variable names.
8. The solver binds the requested variable names to actual solver arrays.
9. During residual evaluation, the solver calls the RHS callback.
10. During implicit assembly, if available, the solver calls the Jacobian callback.
11. At shutdown, the solver calls `dtor()` for sources whose constructors completed successfully.

9.29.4 User Context

The `user_ctx` pointer is owned by the user library. It may be used to store resolved variable indices, equation indices, constants, lookup tables, or other user-defined data.

For example:

```
typedef struct
{
    int var_T;
    int eq_energy;

    real_t q0;
```

```
} heat_ctx_t;
```

The user library may allocate this structure in `ctor()` and free it in `dtor()`.

A simple source may also ignore `user_ctx` and store data in static or global variables inside the user library. This is valid, but `user_ctx` is safer when several source instances use the same shared library.

9.29.5 Variable Lookup

The user library does not access solver arrays by hard-coded locations. Instead, it asks the solver for variable indices by name during `ctor()`.

```
ctx->var_T = api->get_var_index(api->solver_ctx, "T");
```

The returned index is local to the current user source. During the RHS callback, the variable value is accessed through the `vars[]` array:

```
T = vars[ctx->var_T][cell];
```

The user library must not assume a fixed order for `vars[]`. It must always use the indices returned by `get_var_index()`.

For example, if the user library requests:

```
ctx->var_x = api->get_var_index(api->solver_ctx, "x");
ctx->var_T = api->get_var_index(api->solver_ctx, "T");
```

then the solver may bind:

```
vars[0] -> cell-center x-coordinate array
vars[1] -> temperature array
```

The actual order is determined dynamically by the requested variables.

`get_var_index()` returns a nonnegative local index on success. A negative return value means that the requested variable name is not available for the current solver

configuration. The user library must check the returned index before using it.

9.29.6 Available Variables

The currently available variable names are listed in Table 9.315.

Example:

```
int var_T;
int var_mu;
int var_Y_O2;

var_T    = api->get_var_index(api->solver_ctx, "T");
var_mu   = api->get_var_index(api->solver_ctx, "mu");
var_Y_O2 = api->get_var_index(api->solver_ctx, "Y.O2");
```

Then inside `source()`:

```
const real_t T    = vars[var_T][cell];
const real_t mu   = vars[var_mu][cell];
const real_t Y_O2 = vars[var_Y_O2][cell];
```

Species mass fractions are accessed using the prefix `Y.` followed by the species name used in the chemistry mechanism. For example, `Y.O2`, `Y.H2`, and `Y.H2O` request the mass fractions of oxygen, hydrogen, and water vapor, respectively.

9.29.7 Equation Lookup

The user library asks for equation indices by name:

```
ctx->eq_energy = api->get_eq_index(api->solver_ctx, "energy");
```

The returned index is used to add source terms into `rhs[]`:

```
rhs[ctx->eq_energy] += qdot;
```

Available User-Source Variables

Name	Description
`x`	Cell-center x coordinate
`y`	Cell-center y coordinate
`z`	Cell-center z coordinate
`volume`	Cell volume
`wall.distance`	Distance from nearest wall
`mt`	Turbulent dynamic viscosity, μ_t
`rho`	Density
`T`	Temperature
`p`	Pressure
`u`	Velocity component in the x direction
`v`	Velocity component in the y direction
`w`	Velocity component in the z direction
`H`	Total enthalpy
`c`	Speed of sound
`mu`	Molecular dynamic viscosity, μ
`Y.<species>`	Species mass fraction for species `<species>`
`trb.k`	Turbulent kinetic energy, k
`trb.w`	Specific dissipation rate, ω
`trb.dnu`	Spalart–Allmaras model variable, $\tilde{\nu}$

Table 9.315: Available user-source variables



Equation indices must not be hard-coded inside the user library. The equation ordering may change between flow models, chemistry models, and solver configurations.

`get_eq_index()` returns a nonnegative local index on success. A negative return value means that the requested equation name is not available for the current solver configuration. The user library must check the returned index before using it.

9.29.8 Available Equations

The user library requests equation indices by name using `get_eq_index()`.

The available equation names are listed in Table 9.316.

Available User-Source Equations

Name	Availability	Description
`rho`	Single-fluid modes	Continuity equation
`rhoU`	All supported modes	x-momentum equation
`rhoV`	All supported modes	y-momentum equation
`rhoW`	All supported modes	z-momentum equation
`rhoE`	All supported modes	Total energy equation
`rhoY.<species>`	Nonequilibrium chemistry only	Species conservative equation
`trb.rhok`	k-ω turbulence model	Turbulent kinetic energy equation
`trb.rhow`	k-ω turbulence model	turbulent specific dissipation rate equation
`trb.rhodnu`	Spalart–Allmaras model	Spalart–Allmaras model equation

Table 9.316: Available user-source equations

The equation `mass` is available for single-fluid modes, including ideal gas, equilibrium chemistry, and frozen chemistry. It is not available in nonequilibrium chemistry mode.

In nonequilibrium chemistry mode, the conservative species equations are used instead. A species equation is accessed using the prefix `massY.` followed by the species name used in the chemistry mechanism.

Example usage:

```
int eq_energy;
int eq_H2;

eq_energy = api->get_eq_index(api->solver_ctx, "energy");
eq_H2     = api->get_eq_index(api->solver_ctx, "massY.H2");
```

Then inside `source()`:

```
rhs[eq_energy] += qdot;
rhs[eq_H2]     += source_H2;
```

9.29.9 RHS Convention

The RHS callback computes the user-source contribution for the current cell. The solver initializes one local `rhs[]` array to zero for the cell, then passes the same local array through all user sources attached to the system. After all user sources have been called, the solver adds the final local `rhs[]` array to the system RHS/residual.

Therefore, if several user sources are attached to the same system, they are evaluated in the same order in which they appear under the `--system` block:

```
--system
    -user.source  source_a
    -user.source  source_b
    -user.source  source_c
```

The evaluation order is:

```
source_a
source_b
source_c
```

The callback may either set or increment entries in the received `rhs[]` array.

Incrementing an entry adds to the current value:

```
rhs[eq_energy] += qdot;
```

Setting an entry overwrites the current value:

```
rhs[eq_energy] = qdot;
```

Setting is not the same as adding. If an earlier user source already wrote to `rhs[eq_energy]`, then using `=` in a later source overwrites that previous contribution. Using `+=` preserves the previous value and adds to it.

9.29.10 Jacobian Convention

If an implicit user source is provided, the Jacobian callback computes the local derivatives of the user-source contribution for the current cell.

The Jacobian callback follows the same accumulation convention as the RHS callback. The solver initializes one local `J[]` array to zero for the cell, then passes the same local array through all Jacobian callbacks attached to the system. After all user-source Jacobian callbacks have been called, the solver adds the final local `J[]` array to the system Jacobian/LHS matrix.

The local Jacobian is stored in C row-major form:

```
J[row * sim->n_eqs + col] += value;
```

where `row` is the equation receiving the source term, and `col` is the equation variable being differentiated with respect to.

Example:

```
const int e = ctx->eq_energy;

J[e * sim->n_eqs + e] += dS_dE;
```

As with the RHS callback, `=` overwrites the current local value, while `+=` adds to it. For a constant heat source, the derivative is zero, so the Jacobian callback may leave `J[]` unchanged and return success.

9.29.11 Thread Safety

The solver may call `source()` and `jac()` concurrently for different cells. After `ctor()` returns, data stored in `user_ctx` should be treated as read-only during residual and Jacobian evaluation, unless the user library provides its own thread synchronization.

The callback functions must not write to shared global or static data during per-cell evaluation unless that access is thread-safe.

9.29.12 ABI Versioning

The user-source C interface is versioned by `USER_SOURCE_ABI_VERSION`, which is defined in `user_source.h`. The solver stores its supported ABI version in `api->abi_version` before calling the user constructor.

The user library must be compiled against a compatible `user_source.h` header supplied with the release files and must check the ABI version inside `ctor()`:

```
if(api->abi_version != USER_SOURCE_ABI_VERSION)
    return 0;
```

For the current interface, `ctor()` is the ABI compatibility gate: it is the first user callback that receives the solver API structure. If the ABI version does not match, `ctor()` must return zero.

If `ctor()` returns zero, the solver treats this as a fatal user-source initialization error. The solver must not call `source()` or `jac()` for that source. The user library is responsible for freeing any resources it allocated before returning failure from `ctor()`.

The solver calls `dtor()` only for a source whose constructor completed successfully.

The ABI version must be increased whenever an ABI-visible part of the interface changes, including callback signatures, structure layout, structure field meaning, scalar type convention, array layout convention, or callback accumulation semantics.

9.29.13 Minimal Heat Source Example

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

#include "user_source.h"

/* Internal plugin context */
typedef struct
{
    int var_x;

    int eq_energy;
} heat_ctx_t;

int ctor(void** user_ctx, const struct user_source_api_s* api)
{
    heat_ctx_t* ctx;

    if(user_ctx == NULL || api == NULL)
        return 0;

    if(api->abi_version != USER_SOURCE_ABI_VERSION)
    {
        printf("heat.c: bad ABI version\n");
        return 0;
    }
```

```
    ctx = (heat_ctx_t*)calloc(1, sizeof(heat_ctx_t));
    if(ctx == NULL)
        return 0;

    ctx->var_x = api->get_var_index(api->solver_ctx, "x");

    ctx->eq_energy = api->get_eq_index(api->solver_ctx, "energy");

    if(ctx->var_x < 0)
    {
        printf("heat.c: variable x not found\n");
        free(ctx);
        return 0;
    }

    if(ctx->eq_energy < 0)
    {
        printf("heat.c: equation rhoE not found\n");
        free(ctx);
        return 0;
    }

    *user_ctx = ctx;

    printf("heat.c: ctor ok\n");

    return 1;
}

int dtor(void* user_ctx)
{
    free(user_ctx);
```

```
    printf("heat.c: dtor ok\n");

    return 1;
}

int source(
    void* user_ctx,
    const struct user_source_sim_s* sim,
    uint32_t cell,
    const double* const vars[],
    double* rhs)
{
    heat_ctx_t* ctx = (heat_ctx_t*)user_ctx;

    double x = vars[ctx->var_x][cell];

    if(x > 2.0 && x < 2.9)
        rhs[ctx->eq_energy] += 1e5;

    return 1;
}

int jac(
    void* user_ctx,
    const struct user_source_sim_s* sim,
    uint32_t cell,
    const double* const vars[],
    double* J)
{
    // no implicit treatment of heat source

    return 1;
```

```
}
```

Bibliography

[1] Chul Park. Review of chemical-kinetic problems of future NASA missions. I - Earth entries. *Journal of Thermophysics and Heat Transfer*, 7(3):385–398, January 1993.

[2] Leonardo C Scalabrin. *Numerical Simulation of Weakly Ionized Hypersonic Flow Over Reentry Capsules*. PhD thesis, University of Michigan, 2007.

[3] Peter A Gnoffo, Roop N Gupta, and Judy L Shinn. Conservation equations and physical models for hypersonic air flows in thermal and chemical nonequilibrium, 1989.

[4] Alexander Burcat and Branko Ruscic. Third Millenium Ideal Gas and Condensed Phase Thermochemical Database for Combustion with Updates from Active Thermochemical Tables. Technical Report TAE 960, 2005.

[5] C R Wilke. A viscosity equation for gas mixtures. *Journal of Chemical Phys*, 18:517, 1950.

[6] F G Blottner, M Johnson, and M Ellis. Chemically Reacting Viscous Flow Program For Multi-Component Gas Mixtures. Technical report, Sandia National Laboratories (SNL), Albuquerque, NM, and Livermore, CA (United States), January 1971.

[7] Bonnie J Mcbride, Sanford Gordon, and Martin A Reno. Coefficients for calculating thermodynamic and transport properties of individual species. Technical Report NASA TM-4513, 1993.

[8] Joseph Oakland Hirschfelder and Charles F Curtiss. *Molecular theory of gases and liquids*. Wiley, 1964.

[9] Roop N Gupta, Jerrold M Yos, and Richard A Thompson. A review of reaction rates and thermodynamic and transport properties for the 11-species air model for chemical and thermal nonequilibrium calculations to 30000 K. 1989.

[10] Sanford Gordon and Bonnie J Mcbride. Computer Program for Calculation of Complex Chemical Equilibrium Compositions and Applications. Part 1: Analysis, 1994.

[11] Michael G Dunn and Sang-Wook Kang. Theoretical and Experimental Studies of Reentry Plasmas. NASA Langley Research Center, 1973.

[12] Chul Park. *Nonequilibrium hypersonic aerothermodynamics*. John Wiley & Sons, New York, January 1989.

[13] P. R. Spalart and S. R. Allmaras. A one-equation turbulence model for aerodynamic flows. 1992. AIAA paper 92 - 0439.

[14] Jack R. Edwards and Suresh Chandra. Comparison of eddy viscosity-transport turbulence models for three-dimensional, shock-separted flow fields. *AIAA Journal*, 34(4):756–763, 1996.

[15] Steven R Allmaras and Forrester T Johnson. Modifications and clarifications for the implementation of the spalart-allmaras turbulence model. In *Seventh international conference on computational fluid dynamics (ICCFD7)*, pages 1–11, 2012.

[16] CL Rumsey, J-R Carlson, TH Pulliam, and PR Spalart. Improvements to the quadratic constitutive relation based on nasa juncture flow data. *AIAA Journal*, 58(10):4374–4384, 2020.

[17] Philippe R Spalart and Christopher L Rumsey. Effective inflow conditions for turbulence models in aerodynamic calculations. *AIAA journal*, 45(10):2544–2553, 2007.

[18] F. R. Menter. Zonal two equation $k-\omega$ turbulence models for aerodynamic flows. In *24rd AIAA Fluid Dynamics Conference*, Orlando, FL, July 1993. AIAA paper 93 - 2906.

[19] Yu Egorov and F Menter. Development and application of sst-sas turbulence model in the desider project. In *Advances in Hybrid RANS-LES Modelling: Papers contributed to the 2007 Symposium of Hybrid RANS-LES Methods, Corfu, Greece, 17-18 June 2007*, pages 261–270. Springer, 2008.

[20] C. Kok, Johan, H. S. Doi, B. Oskam, and H. van der Van. Extra-large eddy simulation of massively separated flows. In *42th AIAA Aerospace Sciences Meeting & Exhibit*, Reno, NV, January 2004. AIAA paper 2004 - 264.

[21] Mikhail L. Shur, Philippe R. Spalart, Mikhail Kh. Strelets, and Andrey K. Travin. A hybrid LES-RANS approach with delayed-DES and wall-modelled LES capabilities. *International Journal of Heat and Fluid Flow*, 29(6):1638–1649, 2008.

[22] Mikhail S Gritskevich, Andrey V Garbaruk, and Florian R Menter. Fine-tuning of ddes and iddes formulations to the k-ω shear stress transport model. *Progress in Flight Physics*, 5:23–42, 2013.

[23] Mikhail L Shur, Philippe R Spalart, Mikhail Kh Strelets, and Andrey K Travin. An enhanced version of des with rapid transition from rans to les in separated flows. *Flow, turbulence and combustion*, 95(4):709–737, 2015.

[24] Ami Harten, Peter D. Lax, and Bram Van Leer. On upstream differencing and godunov-type schemes for hyperbolic conservation laws. *SIAM Review*, 25(1):35–61, 1983.

[25] E. F. Toro, M. Spruce, and W. Spears. Restoration of the contact surface in the hll riemann solver. *Shock Waves*, 4(4):25–34, 1994.

[26] P. Batten, M. A. Leschziner, and U. C. Goldberg. Average-state Jacobians and implicit methods for compressible viscous and turbulent flows. *Journal of Computational Physics*, 137(1):38–78, 1997.

[27] B. Einfeldt, C. D. Munz, P. L. Roe, and B. Sjogreen. On Godunov-type methods near low densities. *Journal of Computational Physics*, 92(2):273–295, 1991.

[28] P. L. Roe. Approximate riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, 43(2):357–372, 1981.

[29] SF Davis. Simplified second-order godunov-type methods. *SIAM Journal on Scientific and Statistical Computing*, 9(3):445–473, 1988.

[30] Meng-Sing Liou and Christopher J. Steffen Jr. A new flux splitting scheme. *Journal of Computational Physics*, 107(1):23–39, 1993.

[31] Meng-Sing Liou. A sequel to AUSM: $AUSM^+ - up$. *Journal of Computational Physics*, 129:364–382, 1996.

[32] Meng-Sing Liou. A sequel to AUSM, part II: $AUSM^+ - up$ for all speeds. *Journal of Computational Physics*, 214:137–170, 2006.

[33] Yasuhiro Wada and Meng-Sing Liou. A Flux Splitting Scheme with High-resolution and Robustness for Discontinuities. In *32th AIAA Aerospace Science Meeting and Exhibit*, page 23, Reno, NV, 1994.

[34] Culbert B Laney. *Computational Gasdynamics*. Cambridge University Press, May 1998.

[35] Bernard Parent. Positivity-preserving high-resolution schemes for systems of conservation laws. *Journal of Computational Physics*, 231(1):173–189, 2012.

[36] M R Hestenes and E Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49, 1952.

[37] J K Reid. On the method of conjugate gradients for the solution of large sparse systems of linear equations. *Pro. the Oxford conference of institute of mathmatics and its applications*, 5(4):231–254, 1971.

[38] Youcef Saad and Martin H Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[39] D A Knoll and D E Keyes. Jacobian-free Newton–Krylov methods: a survey of approaches and applications. *Journal of Computational Physics*, 193:357–397, 2004.

[40] P McHugh and D E Knoll. Inexact newton's method solutions to the incompressible navier-stokes and energy equations using standard and matrix-free implementations. In *11th Computational Fluid Dynamics Conference*, page 3332, 1993.

[41] Marian Nemec and David W Zingg. Newton-krylov algorithm for aerodynamic design using the navier-stokes equations. *AIAA journal*, 40(6):1146–1154, 2002.

[42] H Alcin, B Koobus, O Allain, and A Dervieux. Efficiency and scalability of a two-level Schwarz algorithm for incompressible and compressible flows. *International Journal for Numerical Methods in Fluids*, 72(1):69–89, May 2013.

[43] T K Sengupta, A Dipankar, and A K Rao. A new compact scheme for parallel computing using domain decomposition . *Journal of Computational Physics*, pages 654–677, 2007.

[44] G Wang and Danesh K Tafti. Performance Enhancement on Microprocessors with Hierarchical Memory Systems for Solving Large Sparse Linear Systems:. *The International Journal of High Performance Computing Applications*, 13:63–79, 1999.

[45] Xiao-Chuan Cai and Marcus Sarkis. A Restricted Additive Schwarz Preconditioner for General Sparse Linear Systems. *SIAM Journal on Scientific Computing*, 21:792–797, 1999.

[46] John E Dennis Jr and Robert B Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. SIAM, 1996.

[47] Robert H Nichols and CC Nelson. Wall function boundary conditions including heat transfer and compressibility. *AIAA journal*, 42(6):1107–1114, 2004.

[48] Roy R. Craig, Jr. *Structural Dynamics: An Introduction to Computer Methods*. John Wiley & Sons, 1981.

[49] Sahar Shpitz and Yuval Levy. The ember material response solver: Theoretical and user's manual. Technical Report 2025-15, Israeli CFD Center, Caesarea, Israel, November 2025.

[50] Charbel Farhat, Michel Lesoinne, and Patrick Le Tallec. Load and motion transfer algorithms for fluid/structure interaction problems with non-matching discrete interfaces: Momentum and energy conservation, optimal discretization and application to aeroelasticity. *Computer Methods in Applied Mechanics and Engineering*, 157(1–2):95–114, 1998.

[51] ZONA Technology, Inc. *ZAERO User's Manual*, version 9.3 edition.

[52] G. T. S. Done. Interpolation of mode shapes: A matrix scheme using two-way spline curves. *Aeronautical Quarterly*, 16(4):333–349, 1965.

[53] Armin Beckert and Holger Wendland. Multivariate interpolation for fluid-structure-interaction problems using radial basis functions. *Aerospace Science and Technology*, 5(2):125–134, 2001.

[54] T. C. S. Rendall and C. B. Allen. Unified fluid–structure interpolation and mesh motion using radial basis functions. *International Journal for Numerical Methods in Engineering*, 74(10):1519–1559, 2008.

[55] Regine Ahrem, Armin Beckert, and Holger Wendland. Recovering rotations in aeroelasticity. *Journal of Fluids and Structures*, 23(6):874–884, 2007.

[56] William P. Rodden and Erwin H. Johnson. *MSC/NASTRAN Aeroelastic Analysis User's Guide*. The MacNeal-Schwendler Corporation, Los Angeles, version 68 edition, 1994.

[57] Robert L. Harder and Robert N. Desmarais. Interpolation using surface splines. *Journal of Aircraft*, 9(2):189–191, 1972.

[58] Carlos A. Felippa, K. C. Park, and Charbel Farhat. Partitioned analysis of coupled mechanical systems. *Computer Methods in Applied Mechanics and Engineering*, 190(24–25):3247–3270, 2001.

[59] Keith Stein, Tayfun E. Tezduyar, and Richard Benney. Mesh moving techniques for fluid–structure interactions with large displacements. *Journal of Applied Mechanics*, 70(1):58–63, 2003.

