NASA/TM—2019-220192

WRLES: Wave Resolving Large-Eddy Simulation Code, Theory and Usage

James R. DeBonis
Glenn Research Center, Cleveland, Ohio

May 2019

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program plays a key part in helping NASA maintain this important role.

The NASA STI Program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI Program provides access to the NASA Technical Report Server—Registered (NTRS Reg) and NASA Technical Report Server—Public (NTRS) thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counter-part of peer-reviewed formal professional papers, but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., "quick-release" reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

- E-mail your question to help@sti.nasa.gov

- Fax your question to the NASA STI Information Desk at 757-864-6500

- Telephone the NASA STI Information Desk at 757-864-9658

- Write to:
  NASA STI Program
  Mail Stop 148
  NASA Langley Research Center
  Hampton, VA 23681-2199

NASA/TM—2019-220192

WRLES: Wave Resolving Large-Eddy Simulation Code, Theory and Usage

James R. DeBonis
Glenn Research Center, Cleveland, Ohio

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

May 2019

Acknowledgments

The author would like to thank Dr. Seth Spiegel and Dr. Dennis Yoder for their thorough reviews and helpful comments. The code described herein was developed over many years under several programs and projects within the National Aeronautics and Space Administration's Aeronautics Research Mission Directorate. The development of this document was sponsored by the Transformative Aeronautics Concepts Program, Transformational Tools and Technologies Project.

This work was sponsored by the
Transformative Aeronautics Concepts Program.

Level of Review: This material has been technically reviewed by technical management.

Available from

NASA STI Program
Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
703-605-6000

This report is available in electronic form at http://www.sti.nasa.gov/ and http://ntrs.nasa.gov/

WRLES: Wave Resolving Large-Eddy Simulation Code, Theory and Usage

James R. DeBonis
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Abstract

A computational fluid dynamics code has been developed for large-eddy simulations (LES) of turbulent flow. The code uses high-order of accuracy and high-resolution numerical methods to minimize solution error and maximize the resolution of the turbulent structures. Spatial discretization is performed using explicit central differencing. The central differencing schemes in the code include 2nd- to 12th-order standard central difference methods as well as 7-, 9-, 11- and 13-point dispersion relation preserving schemes. Solution filtering and high-order shock capturing are included for stability. Time discretization is performed using multistage Runge-Kutta methods that are up to 4th order accurate. Several options are available to model turbulence including: Baldwin-Lomax and Spalart-Allmaras Reynolds-averaged Navier-Stokes turbulence models, and Smagorinsky, Dynamic Smagorinsky and Vreman sub-grid scale models for LES. This report presents the theory behind the numerical and physical models used in the code and provides a user's manual to the operation of the code.

Contents

List of Figures

List of Tables

Nomenclature

a_j	finite difference coefficients
b_j	filter coefficients
c	wave speed
e	internal energy per unit mass
e_t	total energy per unit mass
f	arbitrary function
f	force, $\mathrm{f} = m \cdot l/t^2$
k	thermal conductivity, or wavelength
l	length
l^T	turbulent length scale for synthetic eddy method
l_x^T, l_y^T, l_z^T	turbulent length scale in the x, y, and z directions for digital filtering
m	mass
$\dot{m}$	massflow rate
p	pressure
q_i	heat flux vector
r_i	shock sensor parameter
t	time
Δt	computational time step
$\Delta x, \Delta y, \Delta z$	grid spacing in coordinate directions
$u_i = [u, v, w]^T$	velocity vector
$x_i = [x, y, z]^T$	physical coordinate vector
C_1, C_2	constants in Sutherland's law
C_p	specific heat at constant pressure
C_v	specific heat at constant volume
CFL	Courant-Friedrichs-Lewy number
D	damping function
E, F, G	flux vectors
$G(\xi)$	filter function
J	Jacobian
M	Mach number
N	stencil half-width
Pr	Prandtl number
Pr_t	turbulent Prandtl number
Q	vector of conservation variables
R	specific gas constant
Re	Reynolds number
S_{ij}	strain rate tensor
T	temperature
δ_{ij}	Kronecker delta function
ℓ	turbulent length scale
η	Kolmogorov length scale
γ	ratio of specific heats
μ	dynamic viscosity
ν	kinematic viscosity
ρ	density

σ	filter strength
τ_{ij}	stress tensor
ω	wave number
$\xi_i = [\xi, \eta, \zeta]^T$	computational coordinate vector
Δ	filter width

superscripts

$\bar{}$	averaged quantity
$\hat{}$	Reynolds averaged, or filtered quantity
$\tilde{}$	Favre filtered quantity
T	turbulent (Reynolds-averaged Navier-Stokes)
sgs	subgrid scale
sc	shock capturing
$*$	numerical
$+$	dimensionless quantity based on friction velocity
$\prime$	strong conservation form

subscripts

min	minimum value
max	maximum value
t	derivative with respect to time, or turbulent quantity
x, y, z	derivative with respect to spatial coordinate
v	viscous
$wall$	wall value
ξ, η, ζ	derivative with respect to computational coordinate
0	stagnation/total condition
∞	freestream condition

Introduction

Computational fluid dynamics (CFD) technology is pervasive throughout the aerospace industry and is used in almost every facet of research and product development. Decades of research and development in the areas of numerical methods, physical modeling, grid generation, and computer science has produced the capability of solving highly complex flows in and about highly complex geometries for a wide range of flow speeds. Chemical kinetics, heat transfer, multiphase flow, and fluid-structure interactions are also routinely included in these calculations.

Most widely used production-type CFD codes solve the Reynolds-averaged Navier-Stokes (RANS) equations using second-order accurate finite volume based numerical methods. The RANS equations are closed using a turbulence model. The Spalart-Allmaras model [1] and Menter's shear stress transport (SST) model [2] are the most widely used. RANS methods solve for a time-averaged solution of the Navier-Stokes equations. The turbulence model provides the effect of the unsteady turbulent fluctuations on the time-averaged flow. These codes perform very well in many circumstances. However, they uniformly fail to accurately predict separated flows, and free shear flows. The limiting factor for these "standard" CFD codes is their reliance on the Reynolds-averaged equations and turbulence models. Many attempts have been made to improve RANS turbulence models with little success.

The most promising approach to improve CFD predictions of turbulent flows is to employ scale-resolving simulations. Scale-resolving simulations do away with the Reynolds-averaged equations and directly solve the time accurate equations for the turbulent motion. These simulations are classified into direct numerical simulation (DNS) where all the turbulent scales are computed, and large-eddy simulation (LES) where the large energy-containing scales are computed and the smaller dissipative scales are modeled. The key to scale-resolving simulations is the accurate resolution of the turbulent motion on the computational grid. Turbulent motion contains eddies comprised of a wide range of scales/sizes, and the smallest scales require computational grids much finer than those used in RANS calculations. The CFD codes developed for RANS computations are not well suited for DNS and LES calculations. Their second-order accurate numerics require an inordinate amount of grid points to resolve a turbulent eddy and contain a relatively significant dissipative error, which destroys the turbulent motion we strive to capture.

This report describes a computational fluid dynamics code that was purpose-built for LES and DNS calculations. The code employs high-order of accuracy finite-difference numerical methods which can resolve fine-scale turbulent motion using less grid points than "standard" codes. The code has been successfully applied to round jets [3–7], compressible mixing layers [8, 9], the Taylor-Green vortex [10], a supersonic boundary layer [11], cavity flow [12, 13], turbulent channel flow (unpublished), and flow over a circular cylinder (unpublished). The numerical methods were chosen to minimize dissipative error to preserve and accurately convect the turbulence through the domain. The code can perform DNS, LES, RANS, and hybrid RANS/LES computations, and sub-grid scale and turbulence models are available for LES and RANS calculations. This code uses block structured computational grids with point-matched, overlapping boundaries. Part I of the report describes the governing equations and numerical methods used in the code. Part II describes usage of the code, including grid requirements, input parameters, boundary conditions, and recommended practices.

Part I

Theory

1 Governing Equations

The equations governing turbulent compressible fluid flows are presented below. The fluid is assumed to be a continuum, which implies that the smallest scales of interest are much larger than the scales of molecular motion.

1.1 The Navier-Stokes Equations

The most fundamental set of equations considered are the three-dimensional Navier-Stokes equations. As presented below, they express the conservation of mass, momentum, and energy for an unsteady compressible fluid in tensor form using Cartesian coordinates (x,y,z).

The continuity equation expresses the conservation of mass.

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_i}\left(\rho u_i\right) = 0 \tag{1}$$

The momentum equation expresses Newton's second law. It relates the time rate of change of momentum to the forces applied.

$$\frac{\partial}{\partial t}\left(\rho u_i\right) + \frac{\partial}{\partial x_j}\left(\rho u_i u_j + \delta_{ij} p\right) = \frac{\partial \tau_{ij}}{\partial x_j} \tag{2}$$

The stress tensor is defined as

$$\tau_{ij} = \mu\left(2S_{ij} - \tfrac{2}{3}\delta_{ij} S_{kk}\right) \tag{3}$$

and the strain rate tensor is

$$S_{ij} = \frac{1}{2}\left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j}\right) \tag{4}$$

Sutherland's law is used to model the viscosity

$$\mu = \frac{C_1 T^{\frac{3}{2}}}{C_2 + T} \tag{5}$$

The constants, C_1 and C_2, are dependent on the gas being modeled.

Conservation of energy expresses the first law of thermodynamics. It relates the time rate of change of energy to the amount of heat added and the work done.

$$\frac{\partial}{\partial t}\left(\rho e_t\right) + \frac{\partial}{\partial x_i}\left(\rho u_i e_t + u_i p\right) = \frac{\partial}{\partial x_i}\left(u_j \tau_{ij} - q_i\right) \tag{6}$$

The total energy of the fluid, e_t, is defined from the internal energy, $e = c_v T$

$$e_t = e + \tfrac{1}{2} u_i u_i \tag{7}$$

The heat flux vector is represented by

$$q_i = -k \frac{\partial T}{\partial x_i} \tag{8}$$

The system of equations is closed using the equation of state for a perfect gas.

$$p = \rho R T \tag{9}$$

1.2 Flux Vector Form

For CFD, it is helpful to recast the equations in flux vector form. This puts all of the equations in the same form for easy discretization and solution.

The flux vector form of the three-dimensional Navier-Stokes equations is

$$\frac{\partial Q}{\partial t} + \frac{\partial}{\partial x}(E - E_v) + \frac{\partial}{\partial y}(F - F_v) + \frac{\partial}{\partial z}(G - G_v) = 0 \tag{10}$$

where

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho e_t \end{bmatrix}$$

$$E = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ (\rho e_t + p)\, u \end{bmatrix} \qquad E_v = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ u\tau_{xx} + v\tau_{xy} + w\tau_{xz} - q_x \end{bmatrix}$$

$$F = \begin{bmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ (\rho e_t + p)\, v \end{bmatrix} \qquad F_v = \begin{bmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yz} \\ u\tau_{yx} + v\tau_{yy} + w\tau_{yz} - q_y \end{bmatrix}$$

$$G = \begin{bmatrix} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + p \\ (\rho e_t + p)\, w \end{bmatrix} \qquad G_v = \begin{bmatrix} 0 \\ \tau_{zx} \\ \tau_{zy} \\ \tau_{zz} \\ u\tau_{zx} + v\tau_{zy} + w\tau_{zz} - q_z \end{bmatrix}$$

For inviscid simulations, the Euler equations are obtained by setting the viscous flux vectors, E_v, F_v, and G_v to zero.

1.3 Equations for Turbulent Simulations

1.3.1 Direct Numerical Simulation

Direct numerical simulation (DNS) is the solution of the Navier-Stokes equations in which all scales of turbulent motion are captured in the simulation. This means that the equations are solved in a time-accurate manner on a grid that contains enough grid points so that the numerical scheme can resolve all scales of turbulent motion everywhere in the domain. The smallest scales of turbulence were first estimated by Kolmogorov [14] and bear his name. The Kolmogorov scales are a strong function of Reynolds number, $\eta/\ell_0 = Re^{-3/4}$. As Reynolds number becomes large, the range of scales that must be simulated becomes very large. Consequently, the number of grid points for a high Reynolds number simulation is prohibitive. Choi and Moin provide estimates for the grid requirements of scale resolving simulations [15]. For the foreseeable future, DNS is limited to low Reynolds number flows. Where this technique can be applied, excellent results are possible since there are no additional assumptions or modeling necessary.

1.3.2 Reynolds-Averaged Navier-Stokes

In a Reynolds-averaged Navier-Stokes (RANS) analysis the equations of motion are averaged over a long time period, τ. For a compressible flow a density weighted, Favre averaging, is used.

$$\hat{f} = \frac{1}{\overline{\rho}\tau}\int_t^{t+\tau} \rho f dt \tag{11}$$

The resulting equations are

$$\frac{\partial \overline{\rho}}{\partial t} + \frac{\partial}{\partial x_i}\left(\overline{\rho}\hat{u}_i\right) = 0 \tag{12}$$

$$\frac{\partial}{\partial t}(\overline{\rho}\hat{u}_i) + \frac{\partial}{\partial x_j}(\overline{\rho}\hat{u}_i\hat{u}_j + \delta_{ij}\overline{p}) = \frac{\partial}{\partial x_j}\left(\overline{\tau}_{ij} + \tau_{ij}^T\right) \tag{13}$$

$$\frac{\partial}{\partial t}\left(\overline{\rho}\hat{e}_t\right) + \frac{\partial}{\partial x_i}\left(\overline{\rho}\hat{u}_i\hat{e}_t + \hat{u}_i\overline{p}\right) = \frac{\partial}{\partial x_i}\left[\left(\hat{u}_j\overline{\tau}_{ij} + \hat{u}_j\tau_{ij}^T\right) - \left(\overline{q}_i + q_i^T\right)\right] \tag{14}$$

$$\overline{p} = \overline{\rho}R\hat{T} \tag{15}$$

The unclosed terms from the RANS momentum and energy equations are the Reynolds stress, τ_{ij}^T, (eqn. 13) and the turbulent heat flux vector, q_i^T, (eqn. 14). In linear eddy viscosity models, the

Boussinesq approximation is used to relate the turbulent Reynolds stress to the mean rate of strain tensor through a turbulent (or eddy) viscosity, μ_t.

$$\tau_{ij}^{T} = \mu_t \left(2\hat{S}_{ij} - \tfrac{2}{3}\delta_{ij}\hat{S}_{kk} \right) \tag{16}$$

The rate of strain tensor, $\hat{S}_{ij}$, is given by:

$$\hat{S}_{ij} = \frac{1}{2}\left(\frac{\partial \hat{u}_j}{\partial x_i} + \frac{\partial \hat{u}_i}{\partial x_j} \right) \tag{17}$$

Similarly, the turbulent heat flux is related to the temperature gradient through the turbulent viscosity and the turbulent Prandtl number, Pr_t, is typically treated as a constant.

$$q_j^{T} = -\mu_t \frac{C_p}{Pr_t}\frac{\partial \hat{T}}{\partial x_j} \tag{18}$$

In a RANS simulation, the affect of the turbulent motion on the averaged flow is accounted for by the turbulent viscosity. The turbulent viscosity is computed in the turbulence model based on the local averaged flow quantities. There are numerous turbulence models in the literature of varying sophistication, each with its own advantages and drawbacks. The resulting averaged flowfield is generally assumed to be the time-mean flowfield. In these instances, the solution is converged to a steady state. However, since the time period over which the equations are averaged is ambiguous, this does not have to be the case. Unsteady simulations can be run, but the separation between the unsteady motion captured by the Navier-Stokes equations and the turbulent motion captured by the turbulence model should be clear. Unsteady RANS (URANS) can be used successfully where there is a large-scale unsteadiness in the mean flow that is of a much lower frequency than the random turbulent motion. If this separation of scales is not distinct, the validity of the solution will be questionable.

Baldwin-Lomax Model The Baldwin-Lomax model [16] is a simple algebraic mixing-length model designed for wall bounded flows (boundary layers). Its inclusion in this code is intended for modeling regions of small-scale turbulence where LES is not amenable. It is not intended for accurate turbulent simulations (advanced turbulence models in production-type RANS codes are more appropriate). Some examples include: the flow upstream of a cavity, nozzle boundary layers and attached flow over an airfoil.

The model computes eddy viscosity for two layers: one near the wall - the inner layer and one away from the wall - the outer layer.

$$\mu_t = \begin{cases} (\mu_t)_{\mathsf{inner}}\,, & y \le y_{\mathsf{crossover}} \\ (\mu_t)_{\mathsf{outer}}\,, & y > y_{\mathsf{crossover}} \end{cases} \tag{19}$$

Here, the y-coordinate is assumed to be normal to the wall. The crossover point, $y_{\mathsf{crossover}}$, is the normal distance from the wall at which the two values for μ_t are equal. The inner layer is computed from

$$(\mu_t)_{\mathsf{inner}} = \rho\ell^2|\omega| \tag{20}$$

where

$$\ell = \kappa y\left[1 - e^{\left(-y^+/A^+\right)}\right] \tag{21}$$

where y^+ is the nondimensional distance from the wall, based on the law of the wall, and $|\omega|$ is the magnitude of the vorticity vector. This outer layer is computed from

$$(\mu_t)_{\mathsf{outer}} = \rho\alpha C_{\mathsf{CP}} F_{\mathsf{wake}} F_{\mathsf{Kleb}}(y) \tag{22}$$

where

$$F_{\mathsf{wake}} = min\left(y_{\mathsf{max}}F_{\mathsf{max}}, C_{\mathsf{wake}}y_{\mathsf{max}}U^2_{\mathsf{dif}}/F_{\mathsf{max}}\right) \tag{23}$$

and F is a function based on the normal distance from the wall and the vorticity magnitude

$$F(y) = y|\omega|\left[1 - e^{\left(-y^+/A^+\right)}\right] \tag{24}$$

The Klebanoff intermittency factor is

$$F_{\mathsf{Kleb}}(y) = \left[1 + 5.5\left(\frac{C_{\mathsf{Kleb}}y}{y_{\mathsf{max}}}\right)^6\right]^{-1} \tag{25}$$

F_{max} is the maximum value of F, and y_{max} is location where the maximum value of F is achieved. U_{dif} is the difference between the maximum and minimum velocity in the boundary layer profile. The coefficients are $\kappa = 0.40$, $A^+ = 26$, $\alpha = 0.0168$, $C_{\mathsf{CP}} = 1.6$, $C_{\mathsf{Kleb}} = 0.3$, and $C_{\mathsf{wake}} = 1.0$.

Spalart-Allmaras Model The Spalart-Allmaras (SA) model [1] is a very popular one-equation turbulence model that provides good results for a wide range of applications. The SA model solves a transport equation for the turbulent viscosity. Several variants of the standard SA model have been developed. The Negative SA model allows for negative values of the SA variable $\tilde{\nu}$ and allows stable solutions on under-resolved grids and during transients; desirable features in high-order simulations. A conservative formulation of the Negative SA model, based on reference [17] is implemented here.

The turbulent viscosity transport equation is given by

$$\frac{\partial}{\partial t}(\rho\tilde{\nu}) + \frac{\partial}{\partial x_j}(\rho u_j\tilde{\nu}) - \rho\left(P - D\right) - \frac{1}{\sigma}\frac{\partial}{\partial x_j}\left[\rho(\nu + \tilde{\nu}f_n)\frac{\partial\tilde{\nu}}{\partial x_j}\right] - \frac{c_{b2}}{\sigma}\rho\frac{\partial\tilde{\nu}}{\partial x_k}\frac{\partial\tilde{\nu}}{\partial x_k} + \frac{1}{\sigma}(\nu + \tilde{\nu}f_n)\frac{\partial\rho}{\partial x_k}\frac{\partial\tilde{\nu}}{\partial x_k} = 0 \tag{26}$$

where $\nu = \mu/\rho$. For the negative version of the model, the following terms are dependent on the sign of the turbulent viscosity

$$P = \begin{cases} c_{b1}\left(1 - c_{t3}\right) S\tilde{\nu} & \text{if } \mu_t < 0 \\ c_{b1}\left(1 - f_{t2}\right) \tilde{S}\tilde{\nu} & \text{if } \mu_t \geq 0 \end{cases} \quad (27) \qquad D = \begin{cases} -c_{w1}\left[\frac{\tilde{\nu}}{d}\right]^2 & \text{if } \mu_t < 0 \\ \left(c_{w1} f_w - \frac{c_{b1}}{\kappa^2} f_{t2}\right)\left(\frac{\tilde{\nu}}{d}\right)^2 & \text{if } \mu_t \geq 0 \end{cases} \quad (28)$$

$$f_n = \begin{cases} \frac{c_{n1}+\chi^3}{c_{n1}-\chi^3} & \text{if } \mu_t < 0 \\ 1 & \text{if } \mu_t \geq 0 \end{cases} \quad (29) \qquad \mu_t = \texttt{max}\left(0, \rho\tilde{\nu} f_{v1}\right) \quad (30)$$

The remaining terms are

$$c_{w1} = \frac{c_{b1}}{\kappa^2} + \frac{1 + c_{b2}}{\sigma} \quad (31) \qquad \chi = \frac{\tilde{\nu}}{\nu} \quad (32)$$

$$f_{t2} = c_{t3} e^{\left(-c_{t4}\chi^2\right)} \quad (33) \qquad f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3} \quad (34)$$

$$f_{v2} = 1 - \frac{\chi}{1 + \chi f_{v1}} \quad (35) \qquad S = |\bar{\omega}| \quad (36)$$

$$|\tilde{S}| = S + \frac{\tilde{\nu}}{\kappa^2 d^2} f_{v2} \quad (37) \qquad g = r + c_{w2}\left(r^6 - r\right) \quad (38)$$

$$r = min\left(\frac{\tilde{\nu}}{\tilde{S}\kappa^2 d^2}, r_{lim}\right) \quad (39) \qquad f_w = g\left[\frac{1 + c_{w3}^6}{g^6 + c_{w3}^6}\right]^{1/6} \quad (40)$$

The variable d represents the minimum distance to the wall. The constants are $\sigma = \frac{2}{3}$, $c_{t3} = 1.2$, $c_{t4} = 0.5$, $\kappa = 0.41$, $c_{b1} = 0.1355$, $r_{lim} = 10.0$, $c_{w2} = 0.3$, $c_{w3} = 2.0$, $c_{b2} = 0.622$, $c_{v1} = 7.1$, and $c_{n1} = 16.0$. The SA model equation is treated as a sixth equation in the Navier-Stokes set and solved with the others in the same manner.

1.3.3 Large-Eddy Simulation

The basis of large-eddy simulation (LES) is the separation of the large- and small-scale turbulent fluctuations. The large-scale turbulence is computed directly using the Navier-Stokes equations. The small scale turbulence is modeled. Since the large scales carry the vast majority of the turbulent kinetic energy, the technique offers promise for accurate turbulent simulations. The small-scale turbulence serves to dissipate the large scales and is isotropic. This fact implies that a simple model can be constructed to impose the effect of the small scales on the rest of the flow.

To separate the large (resolved) and small (unresolved) scales, the equations are filtered. A spatial

filter G with a filter width Δ is used.

$$\bar{f} = \int_{-\infty}^{\infty} G(x-\xi) f(\xi) d\xi \tag{41}$$

The overbar represents the resolved, filtered, or large-scale portion of the function. Commonly used filter functions are a box filter, a Gaussian filter, or a spectral cutoff filter [18]. Typically, it is not necessary to know the exact form of the filter function G, but only that it exists. In practice, the solution is not explicitly filtered. It is assumed that the numerically computed solution is a filtered representation of the exact solution.

Several constraints are imposed on the filter function.

$$\begin{aligned}
&1) \quad && G(-\xi) = G(\xi) \\
&2) \quad && \int_{-\infty}^{\infty} G(\xi) d\xi = 1 \\
&3) \quad && G(\xi) \to 0 \quad \text{as} \quad |\xi| \to \infty \\
&4) \quad && G(\xi) \quad \text{is small outside} \quad \left(-\tfrac{\Delta}{2}, \tfrac{\Delta}{2}\right)
\end{aligned}$$

These constraints are necessary to insure that the filter function will commute with the derivative.

$$\overline{\frac{\partial f}{\partial x}} = \frac{\partial \bar{f}}{\partial x} \tag{42}$$

Favre (density) weighting is used in the filtering process. This allows for convenient recovery of terms corresponding to the unfiltered equations.

$$\widetilde{f} = \frac{\overline{\rho f}}{\bar{\rho}} \tag{43}$$

The filtering process is applied to the continuity, momentum, and energy equations. The resulting equations are comprised of resolved and unresolved terms. The resolved terms in the filtered equations directly correspond, in form, to the unfiltered equations.

The resulting LES expressions for conservation of mass, momentum, and energy are:

$$\frac{\partial \overline{\rho}}{\partial t} + \frac{\partial}{\partial x_i}\left(\overline{\rho}\tilde{u}_i\right) = 0 \tag{44}$$

$$\frac{\partial}{\partial t}(\overline{\rho}\tilde{u}_i) + \frac{\partial}{\partial x_j}(\overline{\rho}\tilde{u}_i\tilde{u}_j + \delta_{ij}\overline{p}) = \frac{\partial}{\partial x_j}\left(\overline{\tau}_{ij} + \tau_{ij}^{sgs}\right) \tag{45}$$

$$\frac{\partial}{\partial t}\left(\overline{\rho}\tilde{e}_t\right) + \frac{\partial}{\partial x_i}\left(\overline{\rho}\tilde{u}_i\tilde{e}_t + \tilde{u}_i\overline{p}\right) = \frac{\partial}{\partial x_i}\left[\left(\tilde{u}_j\overline{\tau}_{ij} + \tilde{u}_j\tau_{ij}^{sgs}\right) - \left(\overline{q}_i + q_i^{sgs}\right)\right] \tag{46}$$

$$\overline{p} = \overline{\rho} R \tilde{T} \tag{47}$$

As with the turbulent Reynolds stress in the RANS momentum equation, the sub-grid scale stress is related to the rate of strain tensor by a turbulent viscosity that will be determined by a model.

$$\tau_{ij}^{sgs} = \mu_t \left(2\widetilde{S}_{ij} - \tfrac{2}{3}\delta_{ij}\widetilde{S}_{kk} \right) \tag{48}$$

The rate of strain tensor, $\widetilde{S}_{ij}$, is given by:

$$\widetilde{S}_{ij} = \frac{1}{2}\left(\frac{\partial \tilde{u}_j}{\partial x_i} + \frac{\partial \tilde{u}_i}{\partial x_j} \right) \tag{49}$$

Similarly, the turbulent heat flux vector is related to the temperature gradient through the turbulent viscosity and the turbulent Prandtl number, Pr_t, typically treated as a constant.

$$q_j^{sgs} = -\mu_t \frac{C_p}{Pr_t}\frac{\partial \widetilde{T}}{\partial x_j} \tag{50}$$

The unclosed terms from the LES momentum and energy equations are the sub-grid scale stress, τ_{ij}^{sgs}, (eqn. 45) and the sub-grid scale heat flux vector, q^{sgs}, (eqn. 46). In a traditional LES approach, a sub-grid model would be used to compute τ_{ij}^{sgs} and q^{sgs}, analogous to a RANS turbulence model. However, many practitioners forego sub-grid modeling, relying on the dissipation implicit in the numerical scheme to dissipate the energy at the small scales. This approach is sometimes called Implicit LES (ILES).

In LES, the fact that the small-scale turbulence is not resolved significantly reduces the number of grid points required for a simulation. But, it is important to note that the grid requirements for LES are still high, and increasing the grid resolution reduces the reliance on the sub-grid modeling and its associated errors and assumptions.

Smagorinsky Model The Smagorinsky sub-grid model [19] is included in the code. It is a simple but very popular and widely used model for the sub-grid scale stress tensor. It is an eddy viscosity model where the sub-grid scale stress tensor is modeled as an eddy viscosity multiplying the resolved strain-rate tensor.

$$\mu_t = \bar{\rho} C \Delta^2 |\widetilde{S}| \tag{51}$$

The magnitude of the the strain rate tensor is given by $|\widetilde{S}| = \sqrt{2S_{ij}S_{ij}}$ and the coefficient C is a user defined constant with a default value of 0.01.

The filter width Δ is chosen to be a characteristic length of the computational grid. Since the grid is not uniform, this length varies widely over the grid. The filter width at each location is defined as the cubed root of the volume associated with each grid point.

To maintain a laminar sub-layer in wall bounded regions, the effect of the sub-grid model must be diminished near the wall. To accomplish this, a Van Driest damping function [20] is used to scale

the effect of the sub-grid terms,

$$f_{VD} = \left[1 - e^{\left(-y^+/A^+\right)}\right] \tag{52}$$

where y^+ is the inner variable distance, and the constant A^+ is set to 26.

Dynamic Smagorinsky Model The coefficient in the Smagorinsky model is set *a priori* and is used uniformly throughout the domain. Dynamic sub-grid scale models remove this limitation by automatically computing the coefficient based on the local flow conditions. The dynamic model assumes the relationship between the resolved field and the sub-grid field is the same as the relationship between the resolved field at a larger test-filter width and the resolved field at the grid-filter width. The test-filter width is typically twice the grid-filter width. The stress-strain relationship at the larger scale is compared to the grid scale to determine the coefficient's value. A compressible version of Lilly's dynamic model [21] is implemented here. The stress tensor representing the flow's scales between the test-filter width and grid-filter width is

$$L_{ij} = -\widehat{\bar{\rho}\tilde{u}_i\tilde{u}_j} + \frac{1}{\hat{\bar{\rho}}}\left(\widehat{\bar{\rho}\tilde{u}_i}\widehat{\bar{\rho}\tilde{u}_j}\right) \tag{53}$$

Where $\widehat{()}$ is the test filter and $\tilde{u}_i$ is the grid-filtered velocity. We define

$$M_{ij} = 2\widehat{\Delta^2}\hat{\bar{\rho}}|\hat{\tilde{S}}|\hat{\tilde{S}}_{ij} - 2\Delta^2\widehat{\bar{\rho}|\tilde{S}|\tilde{S}_{ij}} \tag{54}$$

where $\tilde{S}$ is the grid filtered traceless strain rate tensor. The coefficient is obtained from

$$C = \frac{1}{2}\left(\frac{L_{ij}M_{ij}}{M_{ij}M_{ij}}\right) \tag{55}$$

The sub-grid turbulent viscosity is then computed as in the Smagorinsky model (eqn. 51). The dynamic procedure can produce negative values of the coefficient, which cause instability in the solution. This is normally managed by averaging the value of C in a homogeneous flow-direction (i.e. spanwise in a boundary layer or shear layer), or by limiting the coefficient to positive values. By default, WRLES performs averaging in the ζ-/k-direction. Averaging in a different direction, or limiting requires a relatively easy modification of the source code.

The additional filtering procedures used make the dynamic model computationally expensive. To minimize the expense, the dynamic Smagorinsky model is only called on the first Runge-Kutta stage of a time step, and the resulting turbulent viscosity is used for all of the stages in the time step.

Vreman Model Vreman [22] developed a simple algebraic model that addresses some of the perceived deficiencies of the Smagorinsky model. Vreman's model is less dissipative than the

Smagorinsky model and the turbulent viscosity automatically vanishes near the wall and in transitional regions without the ad hoc use of the Van Driest damping function. The model is given by

$$\mu_t = \bar{\rho} C_{\mathtt{vr}} \sqrt{\frac{B_\beta}{\alpha_{ij}\alpha_{ij}}} \tag{56}$$

where

$$\alpha_{ij} = \frac{\partial \tilde{u}_j}{\partial x_i} \qquad\qquad \beta_{ij} = \Delta^2 \alpha_{mi}\alpha_{mj}$$

and

$$B_\beta = \beta_{11}\beta_{22} - \beta_{12}^2 + \beta_{11}\beta_{33} - \beta_{13}^2 + \beta_{22}\beta_{33} - \beta_{23}^2$$

The coefficient in Vreman's model is related to the Smagorinsky constant, $C_{\mathtt{vr}} \approx 2.5C$.

2 Numerical Method

The governing equations presented above are solved numerically using finite-difference-based numerical methods. These numerical schemes will be presented in terms of a one-dimensional model equation for simplicity

$$\frac{\partial q}{\partial t} + \frac{\partial f}{\partial x} = 0 \tag{57}$$

where $f = f(q)$.

2.1 Spatial Discretization

The spatial derivatives are computed using one of several finite difference stencils. For the Navier-Stokes equations, these stencils act on the flux vectors and are represented by the **D** operator in equation 69. Both standard nth-order central differences and several dispersion relation preserving stencils are implemented . Central differencing of the flux terms was chosen for this code because of its non-diffusive properties. Standard central difference schemes are typically written as follows. The second-order stencil is

$$\left.\frac{\partial f}{\partial x}\right|_i = \frac{f_{i+1} - f_{i-1}}{2(\Delta x)} \tag{58}$$

The fourth-order stencil is

$$\left.\frac{\partial f}{\partial x}\right|_i = \frac{-f_{i+2} + 8f_{i+1} - 8f_{i-1} + f_{i-2}}{12(\Delta x)} \tag{59}$$

The size of the differencing stencil is increased to increase the order of accuracy. The central difference stencil can be written for an arbitrary stencil half-width, N, as

$$\left.\frac{\partial f}{\partial x}\right|_i = \sum_{j=1}^{N} \frac{1}{\Delta x}[a_j(f_{i+j} - f_{i-j})] \tag{60}$$

This code is written to accommodate stencils up to $N = 6$. The total number of points in the central difference stencil is $2N + 1$.

2.1.1 Standard Stencils

Standard central difference stencils are designed to minimize the truncation error for a given stencil size. The order of the truncation error is related to the width of the stencil, $O\left[(\Delta x)^{2N}\right]$. WRLES supports central difference stencils from 2nd- to 12th-order. This formal order of accuracy is the traditional measure of a numerical scheme. An alternative way to examine the error in a numerical scheme is Fourier analysis. For DNS and LES analyses, accurate resolution and convection of the turbulent structures is critical. We can use Fourier analysis to examine the resolution and convection of waves using central differencing [23]. As mentioned previously, central difference schemes do not produce dissipation error. However, they do produce dispersion errors that affect the speed of traveling waves. For central difference schemes, the error in the phase speed can be expressed as a function of the normalized angular wave number ($k\Delta x$), where k is the wave number and Δx is the grid spacing.

$$\frac{c^*}{c} = \frac{2}{k\Delta x} \sum_{j=1}^{N} a_j sin\left[j\left(k\Delta x\right)\right] \tag{61}$$

The normalized angular wave number can be used as a measure of the resolution of a wave. The number of points resolving one wavelength is $\frac{2\pi}{k\Delta x}$. When $k\Delta x$ is small, the resolution is high, and vice versa.

The results of equation 61 for the standard central difference schemes are plotted in figure 1. For all schemes, the error in wave speed increases with decreasing resolution, with the lower-order schemes only behaving accurately for very low values of $k\Delta x$. This plot clearly indicates that 2nd-order schemes are a very poor choice for LES and accurate resolution of turbulent structures requires very fine grids or high-order methods.

2.1.2 Dispersion Relation Preserving Stencils

Tam & Webb [24] and Tam & Shenn [25] originated a new family of schemes referred to as dispersion relation preserving (DRP) schemes. In these schemes, some of the coefficients in the stencil are optimized to reduce the dispersion error of the scheme rather than the truncation error. The original DRP scheme used a 7-point stencil. Bogey and Bailly developed a family of DRP schemes

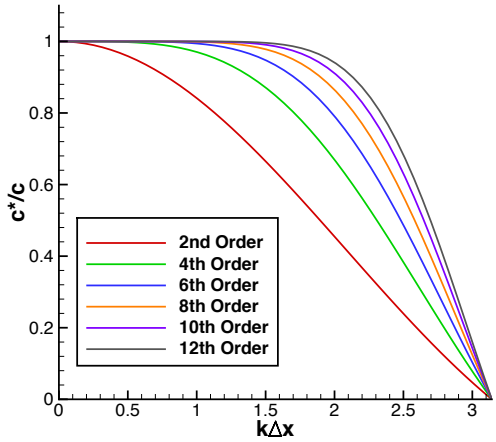


Figure 1: Standard central difference schemes' preservation of wave speed

that use 9-, 11-, and 13- point stencils [26]. All of these schemes are formally fourth-order accurate. Five points are required for a fourth-order accurate scheme. The additional points in the stencils are used to improve the dispersion characteristics of the schemes and the range of wave numbers where phase error is low increases with the stencil size. The wave speed preserving abilities of DRP schemes is compared to the standard schemes in figure 2. Schemes using the same number of points

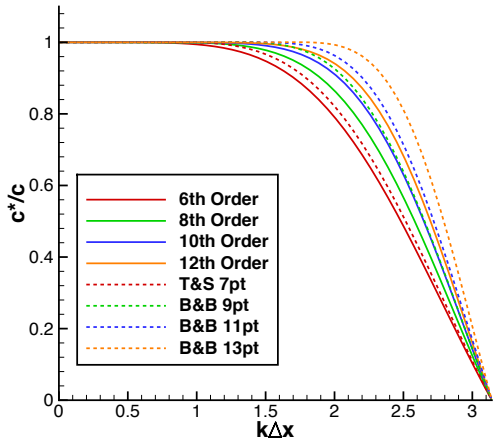


Figure 2: Comparison of standard and dispersion relation preserving schemes' preservation of wave speed

in the stencil are represented by the same color; solid lines indicate standard schemes and dashed lines indicate DRP schemes. It is clear from this plot that the DRP schemes provide accurate convection speeds for a wider range of wave resolutions than the standard schemes for the same amount of computational work. The coefficients for the central-difference schemes implemented in WRLES and used in equation 60 are given in table 1

Table 1: Central differencing stencils for spatial discretization

	coefficient, a_j					
Scheme	1	2	3	4	5	6
2nd order	1/2					
4th order	8/12	-1/12				
6th order	45/60	-9/60	1/60			
8th order	672/840	-168/840	32/840	-3/840		
10th order	2100/2520	-600/2520	150/2520	-25/2520	2/2520	
12th order	23760/27720	-7425/27720	2200/27720	-495/27720	72/27720	-5/27720
Tam & Shen's DRP	0.770882380518	-0.166705904415	0.020843142770			
Bogey & Bailly's 9pt DRP	0.841570125482	-0.244678631765	0.059463584768	-0.007650904064		
Bogey & Bailly's 11pt DRP	0.872756993962	-0.286511173973	0.090320001280	-0.020779405824	0.002484594688	
Bogey & Bailly's 13pt DRP	0.907646591371	-0.337048393268	0.133442885327	-0.045246480208	0.011169294114	-0.001456501759

2.2 Filtering and Artificial Dissipation

The central differencing stencils used for the spatial discretization introduce a dispersive error into the solution. This error manifests itself in two ways. First, waves that are not resolved with enough grid points have their speeds altered. This error can be minimized by using the dispersion relation preserving stencils. Second, new high wave number waves are introduced into the solution. If unchecked, these errors can grow and cause the analysis to become unstable and eventually fail. In addition, when finite difference stencils encounter a discontinuity in the solution, such as a shock wave, they produce unphysical oscillations (ringing) and overshoots near the sharp discontinuity. This is know as the Gibbs phenomenon. Artificial dissipation, added to the solution, damps the unwanted waves and maintains a stable computation.

2.2.1 Solution Filtering

The high wave number waves introduced by the dispersive error destabilize the solution. To ensure a stable solution without adversely affecting the resolving properties of the scheme, solution filtering is used. This is a low-pass filter that leaves the low-wavenumber well-resolved structures untouched and removes the high-wavenumber unresolved structures that cause instability. The filter adds dissipation and care must be taken to insure that it does not adversely affect the solution. The filter must be properly matched to the differencing scheme, so that the filter removes only those waves that are not accurately computed. In WRLES, a coefficient multiplying the filter has been added. By selecting $0 \leq \sigma \leq 1$, the dissipation can be further minimized. As shown in equation 62, the filter is implemented similarly to the finite differencing scheme and is applied sequentially in each computational direction to each field of solution variables at every iteration.

$$\hat{f}_i = f_i + \sigma \sum_{j=-N}^{N} b_j f_{i+j} \tag{62}$$

The damping of the filter as a function of normalized wave number is given by

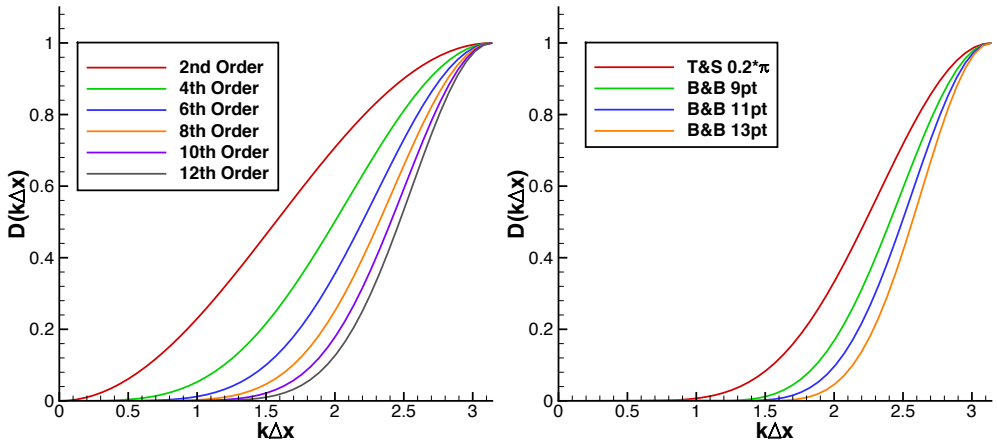
$$D\left(k\Delta x\right) = b_0 + \sigma \left[2 \sum_{j=1}^{N} b_j cos\left(jk\Delta x\right) \right] \tag{63}$$

Kennedy and Carpenter [27] developed a set of solution filters from 2nd- to 12th-order. These are implemented in WRLES and are intended to be used with the standard central difference schemes. In practice, a filter two orders higher than the differencing scheme can be used (e.g. a 4th-order central difference scheme and a 6th-order filter). Bogey & Bailly [26] and Tam & Shen [25] created filters to match the resolving properties of their DRP stencils, and they are also included. Figure 3 shows the filter damping functions for the standard and DRP based filters.

Table 2 lists all the available filters and their coefficients.

Table 2: Coefficients for the spatial filters

	coefficient, b_j						
Scheme	0	1	2	3	4	5	6
2nd order	2/4	-1/4					
4th order	6/16	-4/16	1/16				
6th order	20/64	-15/64	6/64	-1/64			
8th order	70/256	-56/256	28/256	-8/256	1/256		
10th order	252/1024	-210/1024	120/1024	-45/1024	10/1024	-1/1024	
12th order	924/4096	-792/4096	495/4096	-220/4096	66/4096	-12/4096	1/4096
Tam & Shen ($\sigma = 0.3\pi$) DRP	0.327698660846	-0.235718815308	0.086150669577	-0.014281184692			
Tam & Shen ($\sigma = 0.2\pi$) DRP	0.303058855153	-0.231695528614	0.098470572424	-0.018304471386			
Bogey & Bailly's 9pt DRP	0.243527493120	-0.204788880640	0.120007591680	-0.045211119360	0.008228661760		
Bogey & Bailly's 11pt DRP	0.234810479761700	-0.199250131285813	0.120198310245186	-0.049303775636020	0.012396449873964	-0.001446093078167	
Bogey & Bailly's 13pt DRP	0.190899511506	-0.171503832236	0.123632891797	-0.069975429105	0.029662754736	-0.008520738659	0.001254597714



(a) Kennedy & Carpenter filters (b) Dispersion relation preserving filters

Figure 3: Damping function of the solution filters

2.2.2 Shock Capturing

For weak shock waves, the dissipation from the filter may be enough to minimize the Gibbs phenomenon appearing around shock waves. For stronger shocks, additional artificial dissipation can be added to reduce the oscillations. The method used here was developed by Bogey, de Cacqueray, and Bailly [28] and is adapted from Jameson's classic method [29]. It uses a shock sensor function to identify the points in the domain where the shock wave is located and then applies an adaptive second-order filter around the discontinuity.

The shock-filtered variable is

$$f_i^{\text{sc}} = f_i - \left(\sigma^{sc}_{i+\frac{1}{2}} D^{sc}_{i+\frac{1}{2}} - \sigma^{sc}_{i-\frac{1}{2}} D^{sc}_{i-\frac{1}{2}} \right) \tag{64}$$

where the damping functions are computed as

$$D^{sc}_{i+\frac{1}{2}} = \sum_{j=1-n}^{n} c_j f_{i+j} \qquad\qquad D^{sc}_{i-\frac{1}{2}} = \sum_{j=1-n}^{n} c_j f_{i-j}$$

where $n = 2$, $c_{1-j} = -c_j$ and, $c_1 = -0.210383$ and $c_2 = 0.039617$.

The filtering strength, $0 \leq \sigma^{sc}_{i\pm\frac{1}{2}} \leq 1$, varies within the flowfield and is a maximum near shocks and a minimum away from them. It is based on a pressure gradient parameter that senses the location of shock waves

$$dp_i = \frac{-p_{i+1} + 2p_i - p_{i-1}}{4} \tag{65}$$

The magnitude of dp_i is computed as

$$|dp_i| = \frac{1}{2}\left[(dp_i - dp_{i+1})^2 + (dp_i - dp_{i-1})^2\right] \tag{66}$$

and the shock sensor parameter is

$$r_i = \frac{|dp_i|}{p_i^2} + \epsilon \tag{67}$$

where ϵ is a small number. The filter strength is computed based on r_i and a user defined threshold value, r_{th}, that sets the sensitivity of the sensor.

$$\sigma_i^{sc} = \frac{1}{2}\left(1 - \frac{r_{th}}{r_i} + \left|1 - \frac{r_{th}}{r_i}\right|\right) \tag{68}$$

and

$$\sigma_{i+\frac{1}{2}}^{sc} = \left(\sigma_{i+1}^{sc} + \sigma_i^{sc}\right) \qquad \sigma_{i-\frac{1}{2}}^{sc} = \left(\sigma_i^{sc} + \sigma_{i-1}^{sc}\right)$$

2.3 Time Discretization

Runge-Kutta schemes are a popular family of time-advancement schemes that offer high-order temporal accuracy. The multi-stage framework of Runge-Kutta allows a great deal of flexibility. The number of stages and the associated coefficients can be varied to adjust the order of accuracy, stability, and error properties. The number of stages in the scheme must be equal to or greater than the desired order of accuracy. Several researchers [30–33] have developed alternative Runge-Kutta schemes that have a lower dispersion error than the standard scheme leading to greater stability and accuracy. To accomplish this, additional stages are required, which provide a means to impose the additional constraints necessary to minimize the error. Berland, Bogey and Bailly [34], and Allampalli, Hixon, Nallasamy and Sawyer [35] developed schemes that are stable at higher CFL numbers.

All of the schemes in WRLES are based on a general M-stage 2-N storage (2 solution vectors are stored) formulation given by

$$df_m = \alpha_m df_{m-1} + \Delta t \mathbf{D}\left(f_{m-1}\right) \tag{69a}$$

$$f_m = f_{m-1} + \beta_m df_m \tag{69b}$$

for $m = 1 \ldots M$, and where $f_0 = f^n$ represents the solution at the current time step (t) and $f_M = f^{n+1}$ represents the solution at the new time step ($t + \Delta t$). The coefficient α_1 is typically set to zero for the algorithm to be self-starting; a solution at the previous time step $t - \Delta t$ is not required. The operator $\mathbf{D}$ is the spatial finite difference operator as described in section 2.1.

Eleven different schemes are implemented in the code, ranging from first- to fourth-order accuracy and from one to seven Runge-Kutta stages. Table 3 lists the schemes, order of accuracy and number of stages. The coefficients for the schemes, α_m and β_m, are given in table 4.

Table 3: Low-dispersion Runge-Kutta schemes implemented in the code

Scheme		Number of Stages	Order of Accuracy
Euler forward	EF 1-1	1	1
Second order	SO 2-2	2	2
Williamson [36]	WN 3-3	3	3
Gottlieb & Shu, TVD [32]	GS 3-3	3	3
Carpenter & Kennedy [30]	CK 4-3	4	3
Carpenter & Kennedy [30]	CK 5-4	5	4
Stanescu & Habashi [33]	SH 5-4	5	4
Stanescu & Habashi [33]	SH 6-4	6	4
Berland et al [34]	BBB 6-4	6	4
Allampalli et al [35]	HALE-RK 6-4	6	4
Allampalli et al [35]	HALE-RK 7-4	7	4

For time accurate simulations, a global time step size, Δt, is used. The time step is either computed based on a CFL number or is specified directly by the user. The time step based on CFL is computed using

$$\Delta t = CFL \cdot \Delta t_{CFL} \tag{70}$$

where the characteristic time based on the inviscid CFL condition [37] is given by

$$\Delta t_{CFL} = \left(\frac{|u|}{\Delta x} + \frac{|v|}{\Delta y} + \frac{|w|}{\Delta z} + a\sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2}} \right)^{-1} \tag{71}$$

If the CFL number is specified, Δt is computed at every grid point and the minimum value in the domain is used to advance the solution.

Local time stepping is available for steady state analyses or to quickly advance the solution from initial conditions. In this technique, the CFL number is specified and Δt is computed and stored at each grid point. This local time step is used to advance the solution at different rates at each point. To improve stability on poor quality grids, WRLES filters the time step with a second-order Kennedy & Carpenter filter. This ensures that the time step varies smoothly between neighboring points, minimizing oscillations in the solution.

Table 4: Coefficients for Low-Dispersion Runge-Kutta scheme

	α_i						
Scheme	1	2	3	4	5	6	7
EF 1-1	0.0						
SO 2-2	0.0	-1.0					
WN 3-3	0.0	-5/9	-153/128				
GS 3-3	0.0	-2.91549252453890	-0.935419705297567				
CK 4-3	0.0	-1.0	-1.0	-1.0			
CK 5-4	0.0	$\frac{-567301805773}{1357537059087}$	$\frac{-2404267990393}{2016746695238}$	$\frac{-3550918686646}{2091501179385}$	$\frac{-1275806237668}{842570457699}$		
SH 5-4	0.0	-0.691306507590891	-2.65515560104995	-0.814768857645745	-0.668658730443832		
SH 6-4	0.0	-0.491957542000342	-0.894626417580752	-1.552667803218557	-3.407797355404573	-1.07426404175980	
BBB 6-4	0.0	-0.737101392796	-1.634740794341	-0.744739003780	-1.469897351522	-2.813971388035	
HALE-RK 6-4	0.0	-0.691750960670	-1.727127405211	-0.694890150986	-1.039942756197	-1.531977447611	
HALE-RK 7-4	0.0	-0.647900745934	-2.704760863204	-0.460080550118	-0.500581787785	-1.906532255913	-1.45

	β_i						
Scheme	1	2	3	4	5	6	7
EF 1-1	1.0						
SO 2-2	1.0	0.5					
WN 3-3	1/3	15/16	8/15				
GS 3-3	0.924574	0.287713036318675	0.626538109512740				
CK 4-3	1/3	3/4	2/3	1/4			
CK 5-4	$\frac{1432997174477}{9575080441755}$	$\frac{5161836677717}{13612068292357}$	$\frac{1720146321549}{2090206949498}$	$\frac{3134564353537}{4481467310338}$	$\frac{2277821191437}{14882151754819}$		
SH 5-4	0.1	0.75	0.7	0.47931331770131	0.31039285385376		
SH 6-4	0.145309585177875	0.465379788883625	0.467539741872758	0.779527988100590	0.357432717815297	0.15	
BBB 6-4	0.032918605146	0.823256998200	0.381530948900	0.200092213184	1.718581042715	0.27	
HALE-RK 6-4	0.122000000000	0.477263056358	0.381941220320	0.447757195744	0.498614246822	0.186648570846	
HALE-RK 7-4	0.117322146869	0.503270262127	0.233663281658	0.283419634625	0.540367414023	0.371499414620	0.136670099385

2.4 Generalized Curvilinear Coordinates

To enable flow solutions for a wide range of geometries the Navier-Stokes equations are solved using generalized curvilinear coordinates [38]. This enables the computational grids to be fitted to complex shapes and allows for grid stretching.

2.4.1 Coordinate Transformation

Easy and efficient implementation of the numerical scheme is done by working in a computational domain. This domain is rectangular and consists of equally spaced grid points. The computational domain (ξ, η, ζ) is mapped from the physical domain (x, y, z) using the following transformation

$$\begin{aligned} \xi &= \xi(x, y, z) \\ \eta &= \eta(x, y, z) \\ \zeta &= \zeta(x, y, z) \end{aligned} \tag{72}$$

Derivatives in cartesian coordinates are computed using the chain rule

$$\begin{aligned} \frac{\partial}{\partial x} &= \xi_x \frac{\partial}{\partial \xi} + \eta_x \frac{\partial}{\partial \eta} + \zeta_x \frac{\partial}{\partial \zeta} \\ \frac{\partial}{\partial y} &= \xi_y \frac{\partial}{\partial \xi} + \eta_y \frac{\partial}{\partial \eta} + \zeta_y \frac{\partial}{\partial \zeta} \\ \frac{\partial}{\partial z} &= \xi_z \frac{\partial}{\partial \xi} + \eta_z \frac{\partial}{\partial \eta} + \zeta_z \frac{\partial}{\partial \zeta} \end{aligned} \tag{73}$$

The terms ξ_x, ξ_y, ξ_z, η_x, η_y, η_z, ζ_x, ζ_y, and ζ_z are the grid metrics associated with the transformation. The grid metrics are computed numerically using the following method.

$$\begin{aligned} \xi_x &= J(y_\eta z_\zeta - y_\zeta z_\eta) \\ \xi_y &= J(x_\zeta z_\eta - x_\eta z_\zeta) \\ \xi_z &= J(x_\eta y_\zeta - x_\zeta y_\eta) \\ \eta_x &= J(y_\zeta z_\xi - y_\xi z_\zeta) \\ \eta_y &= J(x_\xi z_\zeta - x_\zeta z_\xi) \\ \eta_z &= J(x_\zeta y_\xi - x_\xi y_\zeta) \\ \zeta_x &= J(y_\xi z_\eta - y_\eta z_\xi) \\ \zeta_y &= J(x_\eta z_\xi - x_\xi z_\eta) \\ \zeta_z &= J(x_\xi y_\eta - x_\eta y_\xi) \end{aligned} \tag{74}$$

The Jacobian of the transformation, J, is defined as

$$J = \frac{1}{x_\xi(y_\eta z_\zeta - y_\zeta z_\eta) - x_\eta(y_\xi z_\zeta - y_\zeta z_\xi) + x_\zeta(y_\xi z_\eta - y_\eta z_\xi)} \tag{75}$$

The terms x_ξ, x_η, x_ζ, y_ξ, y_η, y_ζ, z_ξ, z_η, and z_ζ represent differentiation of the physical coordinate with respect to the computational coordinate (subscript) and are computed using the same finite difference formula as specified for the solution of the equations. The calculation is easily done in computational space because the grid is evenly spaced in the ξ, η, and ζ directions.

2.4.2 Strong Conservation Form of the Governing Equations

Most CFD solvers working in generalized curvilinear coordinates solve the strong conservation law form of the Navier-Stokes equations developed by Vinokur [39].

$$\frac{\partial Q'}{\partial t} + \frac{\partial}{\partial \xi}(E' - E'_v) + \frac{\partial}{\partial \eta}(F' - F'_v) + \frac{\partial}{\partial \zeta}(G' - G'_v) = 0 \tag{76}$$

where

$$Q' = \frac{Q}{J}$$

$$E' = \frac{1}{J}(\xi_x E + \xi_y F + \xi_z G) \qquad E'_v = \frac{1}{J}(\xi_x E_v + \xi_y F_v + \xi_z G_v)$$
$$F' = \frac{1}{J}(\eta_x E + \eta_y F + \eta_z G) \qquad F'_v = \frac{1}{J}(\eta_x E_v + \eta_y F_v + \eta_z G_v)$$
$$G' = \frac{1}{J}(\zeta_x E + \zeta_y F + \zeta_z G) \qquad G'_v = \frac{1}{J}(\zeta_x E_v + \zeta_y F_v + \zeta_z G_v)$$

2.4.3 Chain Rule Form of the Governing Equations

Alternatively, the chain rule form of the governing equations can also be used

$$\begin{aligned} &\frac{\partial Q}{\partial t} + \xi_x \frac{\partial}{\partial \xi}(E - E_v) + \eta_x \frac{\partial}{\partial \eta}(E - E_v) + \zeta_x \frac{\partial}{\partial \zeta}(E - E_v) \\ &\quad + \xi_y \frac{\partial}{\partial \xi}(F - F_v) + \eta_y \frac{\partial}{\partial \eta}(F - F_v) + \zeta_y \frac{\partial}{\partial \zeta}(F - F_v) \\ &\quad + \xi_z \frac{\partial}{\partial \xi}(G - G_v) + \eta_z \frac{\partial}{\partial \eta}(G - G_v) + \zeta_z \frac{\partial}{\partial \zeta}(G - G_v) = 0 \end{aligned} \tag{77}$$

This form of the equations is a weak conservation form. Hixon, Shih, Dong, and Mankbadi [40] showed that in practice, the chain rule form of the equations is more accurate than the strong conservation form when the metric terms are computed numerically. Both forms of the equations are available in the code.

Part II

Usage

1 Overview

The WRLES code operates in a manner similar to most other CFD codes. The main inputs to the code are the grid, previous solution (if applicable), and an input file specifying both code options and boundary conditions. It is typically run in batch mode on a highly parallelized computing platform. The code's main output is a new solution file. Most simulations performed with WRLES are time accurate simulations so the new solution is an instantaneous one. The code has the ability to time average any quantity that can be computed from the instantaneous solution and this result can also be output.

The code was originally written in Fortran 77 and subsequently ported to Fortran 90. A mix of both standards can be seen in the code. New routines are written using modern Fortran. Fortran 90 dynamic memory allocation is used throughout. Users should expect to perform at least some code modification; specifying time-averaged variables and other custom output options, at a minimum. Care has been taken to make sure the source code is easy to read and easily changed.

The following sections provide the necessary information to set up and run the code. Simple examples are provided.

1.1 Requirements

WRLES is routinely run on basic multi-core Linux workstations, Macintosh computers, and NASA's Pleiades multi-node, multi-core massively parallel supercomputer. There are only two requirements for running WRLES: a Fortran compiler and the Message Passing Interface (MPI) libraries. OpenMP capability in the Fortran compiler is desirable for multi-core machines, but is not required. The MPI libraries are required even when running on a single compute node. WRLES employs a minimum of 2 MPI processes.

1.2 Setting Parameters

A Fortran module, `parameters.f`, contains parameter statements that are necessary for compiling the code. These parameters control the precision of real variables (single or double), the size of the buffers used in MPI communication, and the size of the arrays used for time averaging and post-processing.

Double precision is recommended and set as the default, but single precision can be selected by commenting out the double precision parameters and uncommenting the single precision parame-

ters.

The buffer sizes should not be changed unless the code is modified and additional variables must be passed in MPI communication. The size of the buffers is determined by the number of integers and real numbers contained in the buffer. When adding or subtracting variables in the buffer(s), one should simply modify the number of integers or reals in the parameter statement(s).

The final two parameters will most likely be modified for every new case that is computed. These parameters set the number of variables that are being time averaged, and the number of variables being post-processed. All time-averaged and post-processed variables are assumed to be real numbers. The user must modify the time averaging, `tavew.f`, and post processing, `postw.f`, subroutines for their needs and then set the parameters `nvartav` and `nvarpost` to ensure that they are sized correctly.

1.3 Compiling

The source code is distributed as a tar file comprised of individual subroutines. WRLES is compiled using a makefile which is included in the distribution. Prior to compiling the code, the user must set, in this makefile, the compiler name, compiler options, and loader options specific to their system. Two examples that work with common system configurations are below. Users are encouraged to experiment with the compiler options on their system to optimize the computational speed.

Many MPI installations use the `mpif90` command to automatically compile the Fortran code and link in the MPI libraries. For these systems the following example should be used.

```
FC=mpif90
FFLAGS=-O2 -openmp
LDFLAGS=-openmp
```

The `-openmp` option allows parallel processing at the do loop level, within an MPI process. The `-O2` sets the compiler optimization level.

For systems without the `mpif90` command, the Fortran compiler must be specified (ifort in this example), and the MPI libraries must be linked in with the loader flag `-lmpi`.

```
FC=ifort
FFLAGS=-O2 -openmp
LDFLAGS=-openmp -lmpi
```

Compilation is performed by issuing the command *make*. The command *make clean* will remove the old executable and object files. A `tar` file can be generated with the command *make tar*. A single source code file can be generated with the command *make cat*. Note that the files `parameters.f`, `data.f` and `dataturbin.f` affect the data structures throughout the code. Modifying one of these files forces a recompilation of all of the subroutines.

1.4 Units

The code solves a dimensional form of the equations. Hence, a consistent set of units must be used. The quantities in the input file must match the units of the grid file. The default system is the British Imperial System (BIS); slugs, feet, degrees Rankine, and seconds. The metric system (SI system) can be selected through the input file; kilograms, meters, degrees Kelvin, and seconds. Table 5 shows the two standard systems of units. Alternatively, the individual constants necessary to define the measurement system may be specified in the input file. Section 3 will have further details on specifying the inputs. The constants affected by a change in units are the specific gas constant and the constants for Sutherland's law to compute the kinematic vicosity.

Table 5: Units for the BIS and SI systems

Quantity	BIS	SI
mass, m	slugs	kg
length, l	ft	m
temperature, T	R	K
time, t	s	s
force, f	lbs	N
pressure, p	psf (lb/ft^2)	Pa (N/m^2)

2 Files

WRLES employs numerous files in the process of running a simulation. There are two files that are always required by the code: the computational grid and the input file. These files are inputs to the code and must be generated externally. The other files contain data computed within the code and are output. In many cases, these files can be read back in, to restart the solution process from the previous code execution.

The input file, which is a text file, selects the equation set to be solved, the numerical methods used, and the boundary conditions for the simulation. The input file will be discussed in section 3.

The grid and solution data use the NASA PLOT3D standard. The format of the PLOT3D based files being input and output is determined by the input variables `ifin` and `ifout` respectively. A value of 1 is a formatted (ASCII) file. A value of 2 is a Fortran unformatted file. The precision of the files is determined by the setting in the source file `parameters.f`. The default is double precision.

Table 6 lists the Fortran unit numbers for the various files input and output by WRLES.

Table 6: File unit numbers and formats

File	Unit Number Input	Unit Number Output	Format
input	1		ASCII text file
output		stdout	ASCII text file
grid	10	20	PLOT3D grid
solution	11	21	PLOT3D solution
turbulent variables, algebraic models	12	22	PLOT3D function
turbulent variables, Spalart-Allmaras model	13	23	PLOT3D function (2 variables)
time-averaged file	15	25	PLOT3D function (`nvartav` variables)
turbulent inflow boundary data	$100 + nb$*		ASCII data file (see section 4.1.2)
grid metrics and Jacobian		27	PLOT3D function (10 variables)
grid cell size		28	PLOT3D function (3 variables)
post-processed data		31	PLOT3D function (`nvarpost` variables)

*nb = grid block number

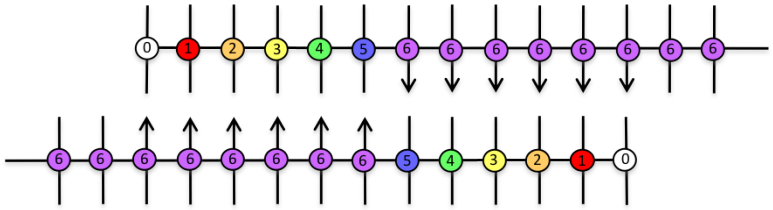


Figure 4: Stencil half widths, N, at overlapping block boundary

2.1 Computational Grids

WRLES uses three-dimensional blocked structured grids. The grids are specified using the NASA PLOT3D file format with the three-dimensional and multi-block options [41]. The code requires that the interface between grid blocks must be point matched and overlapping. In other words, the grid blocks will share a number of common grid planes. The number of overlapping grid planes depends on the largest stencil size, finite-difference or filter, used in the calculation. The number of shared grid planes is $2N$, where N is the stencil half width. This is one less than the total stencil size. This requirement ensures that every grid point in the domain is computed using a full central difference and filter stencil. For example, if the 13-point DRP scheme and matching filter is selected, the stencil half-width is $N = 6$ and the interfacing grid blocks must share 12 ($2N$) common planes. If a 4th-order central difference ($N = 2$) is used in conjunction with a 6th-order filter ($N = 3$ points in the stencil), then the grid blocks must share 6 grid planes. This overlapping requirement preserves the order of accuracy of the scheme at grid interfaces without impacting computational efficiency. A schematic showing the overlap requirement and the resulting stencil half widths at each point for a 13-point stencil is given in figure 4.

High-order schemes are very sensitive to grid quality. Care should be taken to maximize grid

smoothness and minimize grid stretching. Bogey and Bailly recommend stretching rates no more than 2- or 3-percent [26] for their optimized schemes. Abrupt changes in curvature should be avoided, and curved meshes must be adequately resolved to avoid anomalies in the grid metrics. In other words the grid resolution must be such that the grid metrics can be accurately represented by a polynomial of the order of the numerical scheme. This is particularly important when there is curvature near boundaries.

A large-eddy simulation's success hinges on the resolution of the turbulent structures in the flow. These structures must be well resolved in all three-dimensions. In addition to the need for many more grid points compared to a RANS simulation, there are grid topology matters that must be addressed. Grid topologies that are successful in Reynolds-averaged Navier-Stokes computations are often poor choices for LES. RANS grids often have high cell aspect ratios, with much finer resolution in the wall normal direction than the streamwise direction. LES grids require much finer resolution in the streamwise and spanwise directions.

WRLES performs parallel processing at the grid-block level. Each block is processed by a separate MPI process, usually on a separate CPU. The grid must be split into blocks prior to running WRLES. For efficient computations, all blocks should contain a similar number of grid points, so that each processor performs the same amount of work. If the work is not balanced across the processors, some processes must wait for the larger grid blocks to finish, resulting in inefficiencies. Splitting grid blocks can be done when generating the grid. Alternatively, a stand-alone program was developed to split the grid in an efficient manner and generate the appropriate input file. Details of this procedure are in section 6.

The grid file must be in PLOT3D file format: three-dimensional, multi-block, Fortran unformatted (default) [41]. The grid must be dimensioned in units that match those specified in the input file (the default is feet), and it must be in a right-handed coordinate system. It is read on Fortran unit 10. A sample Fortran code that writes the proper grid format is below.

```
  write(20) nblk
  write(20) (imax(n),jmax(n),kmax(n),n=1,nblk)
  do n = 1,nblk
    write(20) (((x(i,j,k,n),i=1,imax(n)),j=1,jmax(n)),k=1,kmax(n)),
&             (((y(i,j,k,n),i=1,imax(n)),j=1,jmax(n)),k=1,kmax(n)),
&             (((z(i,j,k,n),i=1,imax(n)),j=1,jmax(n)),k=1,kmax(n))
  end do
```

The variables are described in table 7. WRLES writes out an iblanked grid file on unit 20. This file contains an iblank array as a fourth variable. The iblank array is an integer array which contains ones and zeros; ones for active points and zeros for those points which are set as holes in the input file. The iblanked file is not necessary for WRLES, but can be useful for plotting.

2.2 Flow Solution

The flowfield used by WRLES is an instantaneous solution if global time stepping is used, and a quasi-steady solution if local time-stepping is used. The file is in PLOT3D solution file format:

Table 7: Variables in PLOT3D grid, solution, and function files

Variable	Description	Type	Dimension
`nblk`	number of grid blocks	integer	1
`imax(n)`	maximum i-index for block `n`	integer	`nblk`
`jmax(n)`	maximum j-index for block `n`	integer	`nblk`
`kmax(n)`	maximum k-index for block `n`	integer	`nblk`
`fsmach`	freestream Mach Number	real	1
`alpha`	angle of attack	real	1
`re`	Reynolds number	real	1
`time`	time	real	1
`timeave0`	total time of averaging	real	1
`x(i,j,k)`	x-coordinate	real	`imax(n) x jmax(n) x kmax(n)`
`y(i,j,k)`	y-coordinate	real	`imax(n) x jmax(n) x kmax(n)`
`z(i,j,k)`	z-coordinate	real	`imax(n) x jmax(n) x kmax(n)`
`q(i,j,k,l)`	solution vector	real	`imax(n) x jmax(n) x kmax(n) x 5`
`func(i,j,k,l)`	function	real	`imax(n) x jmax(n) x kmax(n) x nvar(n)`
`qavg(i,j,k,l)`	time-averaged variable	real	`imax(n) x jmax(n) x kmax(n) x nvartav(n)`

three-dimensional, multi-block, Fortran unformatted (default) [41]. If starting a new computation, the user can set the variable `iinit = 1` in the input file. No solution file will be read and the solution will be initialized to the freestream values specified in the input file. If starting from a previous solution, this solution is read on Fortran unit 11. The final solution is written on unit 21. A sample Fortran code that reads the solution file is below.

```
      read(11) nblk
      read(11) (imax(n),jmax(n),kmax(n),n=1,nblk)
      do n = 1,nblk
        read(11) fsmach,alpha,re,time
        read(11) (((q(i,j,k,l,n),i=1,imax(n)),j=1,jmax(n)),
     &                            k=1,kmax(n)),l=1,5)
      end do
```

Note that `fsmach`, `alpha` and `re` are specified by the PLOT3D standard but are not used in the WRLES code. These variables can be set to any value, but it is recommended that they be set for their intended purpose, freestream Mach number, angle of attack and Reynolds number, in order to avoid any confusion. The `time` variable represents the elapsed time in the simulation. If global time stepping is being used, the value for `time` will be zero. The solution variables are dimensional and the units must match those specified in the input file. Table 8 shows the solution file variables with the generic units: mass (m), length (l), and time (t).

2.3 Function Files

Other files may be used by the code depending on the options selected. Most of these files use the PLOT3D function file format [41].

Table 8: Dimensional units for solution variables

variable	quantity	dimensions
Q_1	ρ	m/l^3
Q_2	ρu	$m/(l^2 \cdot t)$
Q_3	ρv	$m/(l^2 \cdot t)$
Q_4	ρw	$m/(l^2 \cdot t)$
Q_5	ρe_t	$m/(l \cdot t^2)$

```
  read(unit) nblk
  read(unit) (imax(n),jmax(n),kmax(n),nvar(n),n=1,nblk)
  do n = 1,nblk
    read(unit) (((func(i,j,k,l,n),i=1,imax(n)),j=1,jmax(n)),
&                                   k=1,kmax(n)),l=1,nvar(n))
  end do
```

2.3.1 Algebraic Turbulence data

If an algebraic turbulence model or sub-grid scale model is used, the code will produce a function file which contains the turbulent viscosity, μ_t. Strictly speaking, this file is not necessary at startup, as the turbulent viscosity is recomputed before it is used in solving the Navier-Stokes equations. However, WRLES attempts to read the file at startup in case of situations where it will be needed such as a post-processing only run.. If the file is not there, the code will continue after writing a warning message. This file is read on unit 12, and written on unit 22.

2.3.2 Spalart-Allmaras Turbulence data

When the Spalart-Allmaras turbulence model is selected, the code will read and write two variables to a function file: the variable solved in the SA equation, $\rho\tilde{\nu}$, and the resulting turbulent viscosity, μ_t. If the SA model is selected, but the data file does not exist, the turbulence will be initialized to the freestream value. This file is read on unit 13 and written on unit 23.

2.3.3 Time-Averaged Data

If time averaging is activated, the code will output a file containing the time-averaged variables. The file is a slightly modified version of a PLOT3D function file. The only difference from the standard format, shown above, is the addition of a real variable on the first line, which holds the amount of time accumulated in the averaging process.

```
      read(15) nblk,timeave0
      read(15) (imax(n),jmax(n),kmax(n),nvartav(n),n=1,nblk)
      do n = 1,maxblk
        read(15) (((qavg(i,j,k,l,n),i=1,imax(n)),j=1,jmax(n)),
     &                                k=1,kmax(n)),l=1,nvartav(n))
      end do
```

Most programs will ignore this additional variable and read the file as a standard PLOT3D function file. The file is read on Fortran unit 15 and written on unit 25. The number of variables, `nvartav` and the specific variables that are output are specified by the user in the source files `parameters.f` and `tavew.f`, respectively. See section 5.1 for more details.

2.3.4 Grid Metrics

It is sometimes useful to view the grid metrics (section 2.4.1) in order to ascertain grid quality and to debug new code. Whereas most grid generation software has this feature built-in, the metrics are typically computed with second-order numerics. WRLES can output the metrics and Jacobian computed with the scheme currently chosen for the computation. The high-order results can be quite different for highly curved or stretched grids. The variables ξ_x, ξ_y, ξ_z, η_x, η_y, η_z, ζ_x, ζ_y, ζ_z, and J are written on unit 27 when the variable `imet` is set to `1` in the namelist.

2.3.5 Grid Cell Size

The internally computed cell length in each coordinate direction, Δx, Δy, and Δz, can be output on unit 28 by setting the variable `ics` to `1` in the namelist.

2.3.6 Post-Processed Data

The post-processing routines can be modified to the user's needs, and the information here only applies to the routines that come with the standard distribution of WRLES. When post processing is activated, a PLOT3D function file with 5 variables will be written on Fortran unit 31. The file contains the x, y, and z components of vorticity, dilatation, and the magnitude of the strain rate tensor.

3 Code Options

The inputs necessary for running the code are divided into two sections. The first section is for the inputs to select the various options available in the code. These options are divided into three categories: the numerical scheme, physical models, and input/output. The second section specifies the boundary conditions, unused grid regions (holes), and the exit zones.

The inputs in the first section select how the solver is run and what type of simulation is run. They also determine what type of processing is performed on the data and how it is output. These options are specified through a Fortran namelist. The code has default values for all the options. Any variable specified in the namelist overrides the default value. When the code initializes it writes out the values of all code inputs to ensure the user is aware of the options that are selected. The reader can refer to the theory portion of this document, section I, for details on how the input parameters and might affect the results.

The first line of the input file contains the start of the namelist,"`&inputs`". The various options are specified on subsequent lines and the end of the namelist is signaled with a "`/`". The variables, their possible values and their descriptions are found in tables 9, 10, and 11, for the numerical scheme, physical models, and input/output, respectively. Note that the default Fortran convention for integer (variables beginning with letters `i` through `n`) and real variables (all other variables) is used. In general, each integer variable has a predefined list of values that can be specified. Choosing an undefined value of an integer variable will lead to unpredictable code behavior. Real variables are generally physical constants or model coefficients and can take on any value. However, straying too far from the default or physically acceptable value may also lead to unpredictable code behavior.

The structure of the namelist is

```
$inputs
  ninter = 10
  acfl   = 1.0
  isch   = 43
  iorder = 13
  idmp   = 123
  sigma  = 0.5
  ivisc  = 1
  iturb  = 1
/
```

The variables in the namelist are described in the following sections.

3.1 Numerical Scheme Inputs

`niter`, integer: The number of iterations to be run. Make sure that the amount of CPU time is sufficient to complete this number of iterations. Otherwise, the job will terminate without writing the output.

`acfl`, real: Controls the time step. A positive number indicates that the value represents the CFL number. A negative number indicates that the absolute value of the number represents the physical time step, Δt. Because this is an explicit code, stable values for the CFL number are typically less than one. Although, values around 1.5 have been achieved with Berland's scheme. In practice, if local time stepping is selected, the CFL number must be reduced even further. When choosing a physical time step, it is recommended to first run with a stable CFL number and use the code

output to determine a reasonable Δt.

`ltstep`, integer (0 or 1): Selects global or local time stepping. The CFL number is used to compute a time step at every grid point. For `ltstep` = 0, global time stepping is used; the minimum value of the time step in the domain is used to advance the solution everywhere. For `ltstep` = 1 local time stepping is used; the time step at each grid point is used to advance the solution at that point. Local time stepping has proven to be less stable. The local time step is filtered to avoid large disparities at adjacent points.

`isch`, integer: Runge-Kutta scheme. This variable selects the particular Runge-Kutta scheme to be used for time advancement. See table 9 for available schemes.

`iorder`, integer: Selects the spatial differencing scheme. Even values 2 through 12 select the standard central difference stencils of the given order. Odd values greater than 100 select a DRP scheme where the total number of points in the differencing stencil is $2N + 1 =$ `iorder` $- 100$. See table 9 for available schemes.

`iconsv`, integer (0 or 1): Selects the form of the Navier-Stokes equations. A value of 0 selects the chain-rule form. A value of 1 selects the strong conservation form. The strong conservation form runs faster and is recommended for all calculations unless differencing across the centerline in a cylindrical grid is used. For the strong conservation form, the scaling of the solution vector, Q, by the Jacobian is done within the Runge-Kutta loop. Outside of the Runge-Kutta loop, in portions of the code where the user interacts with the data, the solution vector is not scaled by the Jacobian. This includes subroutines responsible for time-averaging, post-processing, boundary conditions, and output.

`idmp`, integer: Selects the filtering/damping scheme. Even values 2 through 10 select Kennedy and Carpenter's filters of the given order. Values 31 and 32 select Tam and Shenn's [25] DRP filters. Odd values greater than 100 select a Bogey and Bailly DRP filter where the total number of points in the filter stencil is $2N + 1 =$ `iorder` $- 100$. See table 9 for details.

`ndmp`, integer: The frequency of filtering/damping. A positive number filters the solution every `ndmp` iterations. A negative number filters the solution every |`ndmp`| Runge-Kutta steps. Filtering the solution once every iteration is typically the best practice. However, `ndmp` can be chosen to vary the amount of numerical dissipation in the solution.

`sigma`, real: The coefficient, σ multiplying the filtering function (equation 62). The value should be $0 \leq \sigma \leq 1$. This variable allows additional control of the amount of numerical dissipation in the solution.

`ishcap`, integer (0 or 1): A value of 1 activates the shock capturing dissipation scheme.

`scrth`, real: The threshold parameter in the shock capturing scheme (equation 68). This parameter determines where the additional dissipation is applied. Smaller values increase the shock sensor's sensitivity, increasing dissipation; larger values decrease it, reducing dissipation.

`iinit`, integer (0 or 1): Solution initialization. A value of 1 initializes the solution at startup to

the freestream Mach number, pressure and temperature specified in the input file. When restarting from a previous solution `iinit`, should be set to zero.

3.2 Physical Modeling Inputs

`ivisc`, integer (0 or 1): Viscous terms. A value of one includes the viscous terms in the Navier-Stokes equations. Setting `ivisc` = 0 creates an inviscid solution.

`iturb`, integer: Turbulence/sub-grid model selector. This variable activates the various RANS turbulence models and LES sub-grid scale models. See table 10 for the various options. A value of zero indicates that no model is active. When no model is active, the type of solution produced will be highly dependent on the resolution of the grid and can vary from laminar flow to a turbulent simulation using implicit LES, to a turbulent flow using direct numerical solution.

`csmg`, real: Smagorinsky/Vreman model coefficient. The coefficient determining the strength of the Smagorinsky sub-grid scale model (see equation 51). This coefficient is also used to set the strength of the Vreman model. The Vreman model coefficient is 2.5 times the value set here.

`prt`, real: Turbulent Prandtl number. This variable is used to relate the turbulent heat flux to the turbulent momentum transfer in RANS and LES sub-grid scale modeling. The value is dependent on the type of flow. See [42] for typical values.

`ibled`, integer: Baldwin-Lomax model boundary layer edge parameter. Limits how far from the wall the Baldwin-Lomax turbulence model will search for the boundary layer edge. This helps avoid spurious results when multiple walls or significant vorticity outside the boundary layer are present in the domain. This variable is expressed in the maximum number of grid points away from the wall to search for the boundary layer edge.

`iblpk`, integer: Baldwin-Lomax model peak selection. Determines which peak (1st, 2nd, etc.) in the F function (equation 24) to select as the boundary layer edge. Spurious peaks in the F function are known to occur very near the wall. Setting `iblpk` to n will ignore the first $n-1$ peaks. Note that typical values for `iblpk` are 2 or 3.

`fsmach`, real: Freestream Mach number. The freestream Mach number is used in several places throughout the code: code initialization, boundary conditions, etc. When a distinct freestream is not present in the flow, this value should be set to a Mach number representative of the bulk flow.

`aoa`, real: Angle of Attack. This variable specifies the angle of the flow relative to the x-axis in the x-y plane. The variable is used in the following boundary conditions: characteristic freestream, subsonic inflow, and supersonic inflow.

`pinf`, real: Freestream pressure. The freestream pressure is used in several places throughout the code: code initialization, boundary conditions, etc. When a distinct freestream is not present in the flow, this value should be set to a pressure representative of the bulk flow.

Table 9: Inputs for the numerical scheme (default values in italics)

Variable	Definition	Value(s)	Description
`niter`	number of iterations	*1*	
`acfl`	time step control	*1.0*	for `acfl` > 0, CFL number
			for `acfl` < 0, $\Delta t = |$`acfl`$|$, time step (s)
`ltstep`	local time stepping	*0*	*off*
		1	on
`isch`	Runge-Kutta temporal scheme	11	Euler forward
		22	2nd-order Runge-Kutta
		33	*Williamson's 3-3 scheme*
		332	Gottlieb & Shu's TVD scheme
		43	Carpenter & Kennedy's 4-3 scheme
		54	Carpenter & Kennedy's 5-4 scheme
		542	Stanescu & Habashi's 5-4 scheme
		64	Stanescu & Habashi's 6-4 scheme
		1064	Allampalli's HALE-RK 6-4 scheme
		1074	Allampalli's HALE-RK 7-4 scheme
		2064	Berland's 6-4 scheme
`iorder`	spatial scheme	2	2nd-order central difference
		4	*4th-order central difference*
		6	6th-order central difference
		8	8th-order central difference
		10	10th-order central difference
		12	12th-order central differnece
		107	Tam & Shenn's 7-point DRP scheme
		109	Bogey & Bailley's 9-point DRP scheme
		111	Bogey & Bailley's 11-point DRP scheme
		113	Bogey & Bailley's 13-point DRP scheme
`iconsv`	conservation form	*0*	*chain-rule form*
		1	strong conservation form
`idmp`	order of filter	2	Kennedy & Carpenter's 2nd-order filter
		4	Kennedy & Carpenter's *4th-order filter*
		6	Kennedy & Carpenter's 6th-order filter
		8	Kennedy & Carpenter's 8th-order filter
		10	Kennedy & Carpenter's 10th-order filter
		12	Kennedy & Carpenter's 12th-order filter
		31	Tam & Shenn's DRP filter ($\sigma = 0.3\pi$)
		32	Tam & Shenn's DRP filter ($\sigma = 0.2\pi$)
		109	Bogey & Bailley's 9-point DRP filter
		111	Bogey & Bailley's 11-point DRP filter
		113	Bogey & Bailley's 13-point DRP filter
`ndmp`	frequency of filter (iterations or Runge-Kutta stages)	*1*	for `ndmp` > 0, every `ndmp` iterations
			for `ndmp` < 0, every $|$`ndmp`$|$ Runge-Kutta stages
`sigma`	filter coefficient	*0.25*	$0 \leq \sigma \leq 1.0$
`ishcap`	shock capturing	*0*	*off*
		1	on
`scrth`	shock threshold	*1.0E-05*	Threshold for shock capturing detection
`iinit`	initial solution	0	read in initial solution
		1	*initialize to freestream values*

`tinf`, real: Freestream temperature. The freestream temperature is used in several places throughout the code: code initialization, boundary conditions, etc. When a distinct freestream is not present in the flow, this value should be set to a temperature representative of the bulk flow.

`gam`, real: Ratio of specific heats. This sets the variable $\gamma = \frac{c_p}{c_v}$.

`iunits`, integer (1 or 2): Sets the system of units used in the simulation. A value of 1 selects British Imperial units: feet, slugs, degrees Rankine. A value of 2 selects the metric system: meters, kilograms, degrees Kelvin.

`rgas`, real: Specific gas constant.

`csuth1`, real: First Sutherland's law constant. A coefficient in Sutherland's law which determines the viscosity as a function of temperature. See equation 5.

`csuth2`, real: Second Sutherland's law constant. A coefficient in Sutherland's law which determines the viscosity as a function of temperature. See equation 5.

`pr`, real: Prandtl number.

`plfac`, real: Pressure limiter factor. The code has the capability to force a lower bound on the static pressure to aid in stability during the initial transients of a solution. By setting `plfac` to a non zero value, the code will force the pressure to not go below $\texttt{plfac} \cdot p_\infty$. This is done by recomputing the density to satisfy the Navier-Stokes equations. The value of `plfac` should be between 0 and 1, and it should be set low enough that pressure limiting is only activated during initial transients.

`ittlim`, integer (0 or 1): Total temperature limiting. It has been found that for flows with very large temperature gradients, the total temperature in small regions of the domain can sometimes grow unchecked as a form of an instability. By setting $\texttt{itlim} = 1$ the total temperature in the solution will be limited to the value specified by `ttlim`. The density is adjusted to satisfy the Navier-Stokes equations.

`ttlim`, real: Total temperature limit. The maximum total temperature allowable in the solution. When used, set the value to at least 110 percent of the expected value. See `ittlim` above for more information.

3.3 Input/Output Inputs

`ifin`, integer: Grid and solution input file format. A value of 1 indicates that the grid and solution files to be read in are formatted Plot3D files. A value of 2 indicates that they are Fortran unformatted Plot3D files.

`ifout`, integer: Grid and solution output file format. A value of 1 indicates that the grid and solution files are to be written as formatted Plot3D files. A value of 2 indicates that they are to be

Table 10: Inputs for physical modeling (default values in italics)

Variable	Definition	Value(s)	Description
`ivisc`	viscous terms	0	off
		1	*on*
`iturb`	turbulence model	*0*	*none*
		1	LES - Smagorinsky
		2	LES - Dynamic Smagorinsky
		3	RANS - Baldwin-Lomax
		4	Hybrid RANS - LES using Baldwin-Lomax
		5	LES - Vreman's model
		6	RANS - Spalart-Allmaras
`csmg`	Smagorinsky coefficient	*0.01*	
`prt`	turbulent Prandtl number	*0.9*	
`ibled`	Baldwin-Lomax edge limiter	*25*	
`iblpk`	Baldwin-Lomax peak selection	*1*	
`fsmach`	freestream Mach number	*0.5*	
`aoa`	angle of attack	*0.0*	degrees
`pinf`	freestream pressure	*2116.8* $\frac{lbs}{ft^2}$	units: $\frac{\mathrm{f}}{l^2}$
`tinf`	freestream temperature	*530.0* R	units: T
`gam`	ratio of specific heats	*1.4*	
`iunits`	system of units	*1*	*British Imperial System (BIS)*
		2	Système International (SI)
`rgas`	specific gas constant	*1716.0* $\frac{ft \cdot lb}{slug \cdot R}$	units: $\frac{l \cdot \mathrm{f}}{m \cdot T}$
`csuth1`	Sutherland's law constant 1	*2.27* $\cdot$ *10*$^{-8}$ $slug/(ft \cdot s \cdot R^{1/2})$	units: $m/(l \cdot t \cdot T^{1/2})$
`csuth2`	Sutherland's law constant 2	*198.72* R	units: T
`pr`	Prandtl number	*0.72*	
`plfac`	pressure limiter factor	*0.1*	$0.0 \leq$ `plfac` ≤ 1.0
`ittlim`	total temperature limiter	*0*	*off*
		1	on
`ttlim`	total temperature limit	∞	

written as Fortran unformatted Plot3D files.

`nprnt`, integer: Output file write frequency. The code can write several pieces of data to the output file in order to check on the solution's progress and stability. These items include the total accumulated solution time along with the values and locations of the minimum time step, minimum pressure, maximum total temperature, and maximum Spalart-Allmaras variable. The variable `nprnt` is the frequency, in terms of the number of iterations, that this data is printed. WRLES typically runs for hundreds of thousands of iterations and setting the frequency greater than one can reduce the output file size.

`icp`, integer (0 or 1): Checkpoint file. Setting this variable to 1, writes a solution on Fortran unit 21 every `ncp` iterations.

`ncp`, integer: Checkpoint file frequency. The frequency of writing checkpoint files, in iterations.

`iffs`, integer (0, 1, or 2): Flowfield save. Setting this variable to 1 or 2 writes files on sequential Fortran unit numbers beginning with the value specified in the `isav` input parameter. A value of 1 uses a write frequency based on `nffs` iterations. A value of 2 uses a write frequency based on `tffs` seconds (approximate).

`nffs`, integer: Frequency of flowfield save in iterations.

`tffs`, real: Frequency of flowfield save in solution time (seconds).

`isav`, integer: Flowfield save Fortran unit number. The initial Fortran unit number for the flowfield save. If time averaging is active, the time-averaged file will be saved on unit `isav` + 10000. If the Spalart-Allmaras model is active, the turbulence model data will be saved on unit `isav` + 20000. Each subsequent file is incremented by one. To prevent overwriting existing files, `isav` must be changed when restarting WRLES from a previous solution.

`itav`, integer (0, 1, or 2): Time averaging. Setting this variable to 1 or 2 activates time averaging of the flowfield. A value of 1 uses a time-averaging frequency based on `ntav` iterations. A value of 2 uses a time-averaging frequency based on `ttav` seconds (approximate).

`ntav`, integer: Frequency of time averaging in iterations.

`ttav`, real: Frequency of time averaging in solution time (seconds).

`itres`, integer (0 or 1): Time reset. Setting this value to 1 resets the solution time to zero when the code starts. Be careful to change this variable back to zero before starting another run.

`iudat`, integer (0, 1, or 2): Write user defined data. Setting this variable to 1 or 2 activates writing user defined data. A value of 1 writes user data at a frequency based on `nudat` iterations. A value of 2 writes the data at a frequency based on `tudat` seconds (approximate).

`nudat`, integer: Frequency of writing user data in iterations.

`tudat`, real: Frequency of writing user data in solution time (seconds).

`iudsav`, integer: Initial Fortran unit number for the user data. The file unit number is incremented by one for each write.

`itavin`, integer (0 or 1): Initialize time averaging. Forces the reinitialization of the time-averaging file. Be careful to change this variable back to zero before starting another run.

`itt`, integer (0 or 1): Max. total temperature output. Setting this variable to 1 outputs the maximum total temperature and its location in the solution.

`isa`, integer (0 or 1): Max. Spalart-Allmaras variable output. Setting this variable to 1 outputs the maximum Spalart-Allmaras variable and its location in the solution.

`ipost`, integer (0 or 1): Post processing data. Setting this variable to 1 activates the post-processing routines at the end of a run.

`imet`, integer (0 or 1): Metric and Jacobian output. Setting this variable to 1 writes the grid transformation metrics and Jacobian to Fortran unit 27.

`ics`, integer (0 or 1): Cell size output. Setting this variable to 1 outputs the size of every grid cell in each coordinate direction to Fortran unit 28.

4 Grid Based Specifications

Following the line terminating the namelist, a series of lines specifying information related to the grid begins. Three types of data are provided: boundary conditions, hole points, and exit zones. This data is grouped by block and all three pieces of information are provided for a grid block before the next grid block's information begins.

4.1 Boundary Conditions

Boundary conditions are available to simulate a wide variety of flows. Because the width of the central difference stencil is reduced as the boundary is approached, thus reducing the resolution, most of the boundary conditions do not attempt to maintain high resolution. This is true for wall, inflow and outflow boundaries. High resolution is maintained for periodic, cylindrical centerline and block interface boundaries. The following section, Sec. 4.1.1 explains how to specify the boundaries in the input file. Sections 4.1.2, 4.1.3, 4.1.4, and 4.1.5 provide descriptions of the inflow/outflow boundaries, surface boundaries, axis boundaries, and periodic and interface boundaries respectively. Table 14 summarizes the boundary conditions and the information needed to specify them in the input file.

Table 11: Inputs for input/output specification (default values in italics)

Variable	Definition	Value(s)	Description
`ifin`	grid and solution read file format	1	formatted
		2	*Fortran unformatted*
`ifout`	grid and solution write file format	1	formatted
		2	*Fortran unformatted*
`nprnt`	output file write frequency	*1*	number of iterations
`icp`	write a checkpoint file	*0*	*off*
		1	on
`ncp`	frequency of checkpointing	*0*	number of iterations
`iffs`	save the flowfield	*0*	*off*
		1	on, save every `nffs` iterations
		2	on, save every `tffs` seconds
`nffs`	flowfield save frequency, iterations	*0*	number of iterations
`tffs`	flowfield save frequency, time	*0*	seconds
`isav`	unit number for first flowfield file	*1000*	Fortran unit number
`itav`	time averaging	*0*	*off*
		1	on, average every `ntav` iterations
		2	on, average every `ttav` seconds
`ntav`	time averaging frequency, iterations	*0*	number of iterations
`ttav`	time averaging frequency, time	*0*	seconds
`itres`	reset time to zero	*0*	*no*
		1	yes
`iudat`	call user data subroutine	*0*	*off*
		1	on, call every `nudat` iterations
		2	on, call every `tudat` seconds
`nudat`	user data frequency, iterations	*0*	number of iterations
`tudat`	user data frequency, time	*0*	seconds
`iudsav`	unit number for first user data file	*5000*	Fortran unit number
`itavin`	initialize time averaging	*0*	*off*
		1	on
`itt`	print max. total temperature	*0*	*off*
		1	on
`isa`	print max. Spalart-Allmaras variable	*0*	*off*
		1	on
`ipost`	post-process data	*0*	*off*
		1	on
`imet`	output grid metrics	*0*	*off*
		1	on
`ics`	output cell size	*0*	*off*
		1	on

4.1.1 Boundary Specification

The first line is not read and can be used for comments. The second line contains the number of boundary conditions for this block (a minimum of six is required). The subsequent lines contain the boundary condition specifications. Table 12 describes the variables in the specification. A sample follows.

```
Start of Boundary Conditions
maxbc
iside  icnst  ibcmin1  ibcmax1  ibcmin2  ibcmax2  ibc  iint  var1*  var2*  var3*  var4*
.
.
.
iside  icnst  ibcmin1  ibcmax1  ibcmin2  ibcmax2  ibc  iint  var1*  var2*  var3*  var4*
```

First, the grid plane is identified through the `iside`, `icnst`, `ibcmin1`, `ibcmax1`, `ibcmin2`, and `ibcmax2` variables. The boundary specifier `iside` tells the code 1) which computational direction is normal to the boundary, 2) whether the boundary is a constant i, j, or k plane, and 3) if it is a minimum or maximum boundary. For a minimum boundary, one must increase the grid index (from the value specified by `icnst`) to move into the domain. For a maximum boundary, one must decrease the grid index to move into the domain. For example, an inflow plane represented by $i = 1$ results in `iside` $= 1$ and `icnst` $= 1$. An upper wall represented by the plane at $j = 99$ results in `iside` $= 4$ and `icnst` $= 99$. The order of the varying indices is always i before j, before k. For example if j is the constant index, the first varying index is i and the second is k. Table 13 provides more detail.

Following the variables that define the grid plane, the boundary condition, `ibc`, the interface number `iint`, and the auxilary variables `var1` to `var4` are listed. For boundaries that serve as interfaces between blocks, the variable `iint` must be a unique integer that is shared with its corresponding interface in another block. For all other boundaries, this variable must be specified (typically set to zero), but is not used. The auxilary variables do not need to be listed unless the boundary condition requires them. See the input files in sections 8.1 and 8.2 for examples of how to specify the boundary conditions.

4.1.2 Inflow and Outflow Boundaries

Subsonic Inflow(s)
This boundary specifies the total pressure and total temperature of the flow. The incoming flow is assumed to be moving in the positive x-direction. Flow angle with respect to the y-direction can be specified through the freestream angle of attack, `aoa`. There are two different subsonic inflow boundary conditions that vary in the way they handle the outgoing wave. The first option (`ibc` $= 1$) simply extrapolates the axial velocity. The second (`ibc` $= 7$) extrapolates the outgoing characteristic, and is the recommended option. This formulation performs a Newton iteration on the static temperature and can have stability issues during startup transients. If this occurs, one can start the calculation with `ibc` $= 1$ and switch to `ibc` $= 7$ after the startup transients have been eliminated.

Table 12: Variables in boundary condition specification (* as needed)

Variable	Description
integers	
`maxbc`	number of boundary conditions specified
`iside`	i/j/k, and min/max boundary indicator (see Table 13)
`icnst`	value of the constant index of the boundary
`ibcmin1`	minimum of the first varying index of the boundary
`ibcmax1`	maximum of the first varying index of the boundary
`ibcmin2`	minimum of the second varying index of the boundary
`ibcmax2`	maximum of the second varying index of the boundary
`ibc`	number representing boundary condition type
`iint`	block interface number
reals	
`var1`	variable necessary for boundary condition specification*
`var2`	variable necessary for boundary condition specification*
`var3`	variable necessary for boundary condition specification*
`var4`	variable necessary for boundary condition specification*

Table 13: `iside` boundary indicator

`iside`	description	1st varying index	2nd varying index
`1`	i minimum boundary	j	k
`2`	i maximum boundary	j	k
`3`	j minimum boundary	i	k
`4`	j maximum boundary	i	k
`5`	k minimum boundary	i	j
`6`	k maximum boundary	i	j

Supersonic Inflow(s)
These boundaries fix the conservation variables based on the following: `ibc` = 2 uses the freestream conditions (M_∞, p_∞, and T_∞) specified in the namelist, and `ibc` = 8 uses the Mach number, static pressure and static temperature in the individual boundary condition specification, `var1`, `var2`, and `var3`. The incoming flow is assumed to be moving in the positive x-direction. Flow angle with respect to the y-direction can be specified through the freestream angle of attack.

Farfield Characteristic Boundary
This boundary imposes the freestream conditions, specified in the namelist, at a theoretical point infinitely far from the domain. Incoming and outgoing characteristic waves are used to determine the conditions at the boundary. This boundary should be placed far away from the primary area of interest. For best results, the boundary should not be parallel to the freestream flow.

Subsonic Outflow
The static pressure is held to the specified value on this boundary. The other variables are extrapolated from the interior of the domain. Care must be taken so that reverse flow does not reach this boundary. In such a case the boundary becomes unstable.

Supersonic Outflow
All variables are extrapolated from the interior of the domain.

Mixed Outflow
This boundary condition computes the local Mach number and applies the appropriate subsonic or supersonic outflow conditions. Static pressure must be specified for the subsonic portion.

Subsonic Outflow/Inflow
This boundary is intended as a more robust subsonic outflow condition for cases where a transient reverse flow condition may occur. The normal velocity at the boundary is computed. For regions of outflow, the standard subsonic outflow boundary is applied. Where there is inflow a simplified subsonic inflow boundary is applied. The total pressure and temperature of the flow must be

Table 14: Boundary condition numbering and required inputs

Boundary Condition	ibc	Auxilliary Variable 1	2	3	4
subsonic inflow, extrapolate velocity	1	p_0	T_0		
subsonic inflow, extrapolate characteristic	7	p_0	T_0		
supersonic inflow, uses freestream values from the namelist	2				
supersonic inflow, specified values	8	M	p	T	
characteristic boundary, uses freestream values from the namelist	3				
subsonic outflow, static pressure specified	11	p			
supersonic outflow	12				
mixed subsonic/supersonic outflow	13	p			
subsonic inflow/outflow	14	p_0	T_0	p	
synthetic eddy turbulent inflow	16	u_{conv}	l^T		
digital filtering (uniform grid) turbulent inflow	18	u_{conv}	l_x^T	l_y^T	l_z^T
digital filtering (computational grid) turbulent inflow	19	u_{conv}	l_x^T	l_y^T	l_z^T
inviscid (slip) wall	21				
viscous wall, adiabatic	22				
viscous wall, isothermal	23	T_{wall}			
viscous wall with turbulent boundary layer, adiabatic	24				
viscous wall with turbulent boundary layer, isothermal	25	T_{wall}			
axis at j_{min}, average over k	31				
axis at j_{min}, average over i	32				
axis at j_{min}, difference across	33				
periodic/overlap, 11-point stencils	40				
periodic/overlap, 12-point stencils	41				
periodic/overlap, 3-point stencils	42				
periodic/overlap, 5-point stencils	44				
periodic/overlap, 7-point stencils	46				
periodic/overlap, 9-point stencils	48				
block interface, 11-point stencils	50				
block interface, 13-point stencils	51				
block interface, 3-point stencils	52				
block interface, 5-point stencils	54				
block interface, 7-point stencils	56				
block interface, 9-point stencils	58				

specified for the inflow portion. Static pressure must be specified for the outflow portion. Care must be taken when using this condition, and it recommended that the problem be formulated in such a way as to remove the reverse flow at the boundary.

Synthetic Eddy Method Turbulent Inflow

The synthetic eddy method (SEM) provides a means to produce quasi-realistic turbulence at an inflow boundary [43]. This allows the user to forego expensive computations that include transition regions or recycling/rescaling to produce the turbulence. The boundary condition produces a time-varying velocity field that replicates a mean flow and Reynolds stress profile specified by the user. The method works by producing a random distribution of synthetic eddies, which are convected through a volume surrounding the inflow plane. The influence of the eddies on each point of the boundary is computed at each time step.

The SEM boundary condition is only written for an i-minimum (`iside` = 1) plane, with the flow in the positive x-direction. The flow profile and Reynolds stresses should vary in the j-coordinate. If there is a wall, it must be on the j_{min} or j_{max} boundary. The specified profile replicated at every k-plane. The convective velocity and the length scale of the synthetic eddies are specified as auxiliary variables (`var1` and `var2` respectively). The profile is specified in a formatted file that is read on Fortran unit $100 + nb$, where nb is equal to the number of the grid block where the boundary is located. In the file, there must be one line for each j-location. Each line contains the following variables: y-coordinate - y, average axial velocity - $\overline{u}$, axial Reynold normal stress - $< uu >$, transverse Reynolds normal stress - $< vv >$, spanwise Reynolds normal stress - $< ww >$, primary Reynolds shear stress - $< uv >$, average density - $\overline{\rho}$, and average temperature - $\overline{T}$. The file is read by the following Fortran code.

```
semunit = 100 + nb
do j=jmin,jmax
  read(semunit,*) y(j),u_avg(j),r11(j),r22(j),r33(j),r12(j),rho_avg(j),t_avg(j)
end do
```

For boundary layers, it has been found that convective speeds on the order of the edge velocity and length scales of $\delta/10$ to $\delta/20$ provide good results.

Digital Filtering Turbulent Inflow

Digital filtering provides an alternative means to SEM to produce turbulent structures at an inflow boundary [44]. This method digitally filters a random signal to produce the turbulent structures. Digital filtering allows for the specification of different length scales in each direction. The widths of the filters are set to the turbulent length scales, which results in eddies of the desired size. The convective velocity and three length scales (l_x^T, l_y^T, and l_z^T) are set in the boundary condition specification as auxiliary variables `var1` - `var4` respectively. There are two options for digital filtering inflow. In the first, `ibc` = 18, the generation of the turbulent structures is done on a uniform grid and the structure sizes are inherently uniform across the inflow plane. The results are then interpolated onto the computational grid. Here, the input flow profile does not need to match the computational grid. In the second version, `ibc` = 19, the digital filtering is done on the computational grid and the digital filter widths are locally scaled to provide uniform structures. Here, the input flow profile must be matched to the computational grid. In either case, the input file has the same format and is read on the same Fortran unit as the SEM inputs described above.

4.1.3 Surface Boundaries

Inviscid (Slip) Wall
Flow tangency is enforced on the boundary. This condition can be used as a solid wall condition for an inviscid simulation, or as a plane of symmetry.

Viscous (No-Slip) Walls
The no-slip condition for viscous flow is applied. The velocity at the wall is set to zero and the pressure gradient normal to the wall is set to zero. This boundary is used for laminar flow or for DNS and LES simulations when no RANS turbulence model is applied. `ibc` = 22 imposes an adiabatic wall (no heat transfer at the wall). `ibc` = 23 imposes an isothermal (constant temperature) condition at the wall.

Viscous (No-Slip) Wall with Turbulent Boundary Layer
This boundary condition imposes the no-slip condition at the wall and uses a RANS-based turbulent viscosity in the boundary layer adjacent to the wall. The specific RANS model is specified using the `iturb` variable. By using separate boundary types for no-slip walls and walls with turbulent boundary layers, transition can be crudely modeled for a RANS calculation. Wall distance for the turbulence model is computed along grid lines emanating from the boundary. An adiabatic wall is selected with `ibc` = 24, and an isothermal wall is selected with `ibc` = 25.

4.1.4 Axis Boundaries for Cylindrical Grids

The following boundaries are for meshes with an o-grid topology, where one computational surface is collapsed to form the axis. The WRLES code is only setup to handle axis boundaries on the `jmin` surface.

Averaging
This condition averages each of the flow variables around the axis and applies this average to the axis. There are two options depending on which computational coordinate varies around the axis : one averages over the k-index (`ibc` = 31) and the other averages over the i-index (`ibc` = 32). The collapsed surface that stems from this approach produces very small grid cells, resulting in very small time steps. This may be overcome with the next option.

Differencing Across the Axis
The centerline differencing condition (`ibc` = 33) computes the derivative across the axis centerline allowing the equations to be solved on this surface. This boundary condition eliminates the averaging condition and improves the time step restriction due to the small grid cells. The formulation is taken from Hixon [40], and the concept is illustrated in figure 5. The grid is constructed with a cylindrical void on the centerline to avoid a collapsed surface. The cell spacing is set such that the diameter of the void is equal to the adjacent radial grid spacing. Then, a stencil is constructed in such a manner that the points on the opposite side of the void are used.

4.1.5 Periodic and Block Interface Conditions

Periodic/Overlap
These boundary conditions are used for overlapping boundaries and periodic boundaries found within the same block. Examples of overlapping boundaries of this type are the connections (sometimes called branch cuts) in o-grid and c-grid topologies. To simulate periodicity in instances where the boundaries reside in different grid blocks, block interface boundaries should be use. In order to maintain accuracy across the boundaries, the grid must be overlapped. For a differencing scheme with N points in the stencil, the overlapped portion of the grid must share $N-1$ common planes. For example, if you are using the 11-point DRP scheme, the grid must have 10 common overlapped planes. Standard scheme stencils are one greater than the order of accuracy; a 4th-order scheme uses a 5-point stencil. Note that these boundary conditions simply copy the data to the corresponding plane on the opposite boundary. If periodicity is required on a semi-cylindrical geometry, the routines must be modified to deal with the radial and azimuthal velocity components.

Block Interface
These boundary conditions are used for passing data between grid blocks using MPI. The block boundaries must overlap to maintain the numerical scheme's accuracy. For a differencing scheme with N points in the stencil, the overlapped portion of the grid must share $N-1$ common planes. For example, if you are using the 11-point DRP scheme, the grid must have 10 common overlapped planes. Standard scheme stencils are one greater than the order of accuracy; a 4th-order scheme uses a 5-point stencil. See section 2.1 and figure 4 for more details. Block interfaces are numbered with each corresponding pair of interface boundaries sharing a unique number.

4.2 Hole Points

To represent complex geometries with block structured grids, it is sometimes necessary to place grid points in regions where there is no fluid flow. These points are not computed in the solver and are usually referred to as hole points. Hole points are specified following the block's boundary conditions. The first line following the last block boundary condition is for comments. The second line lists the number of holes in the block. If there are no hole points in the block, a zero must be specified. The following lines (if any) list the minimum and maximum i, j, and k indices for each hole.

```
Start of Hole Specification
maxhol
ihmin ihmax  jhmin jhmax  khmin khmax
.
.
.
ihmin ihmax  jhmin jhmax  khmin khmax
```

Table 15 describes the variables in the specification.

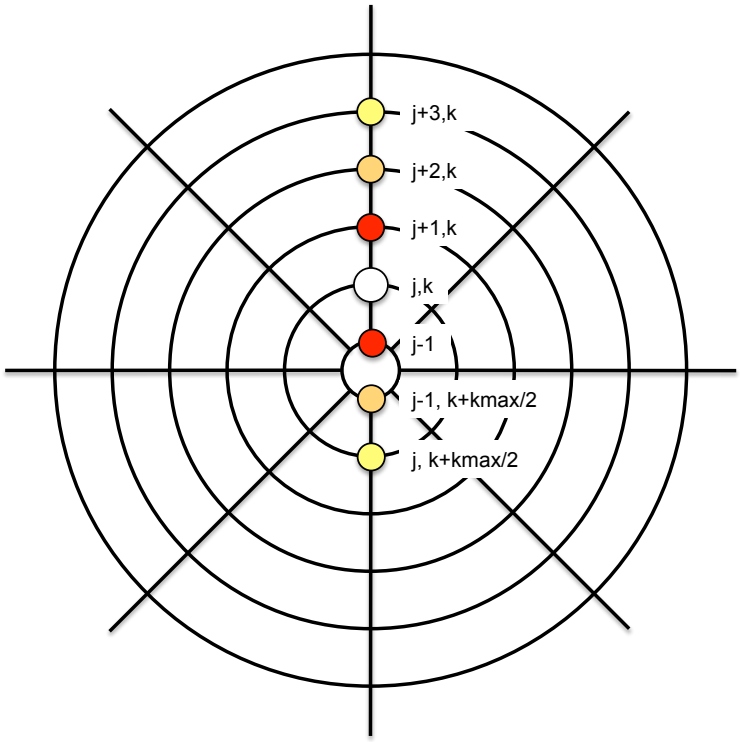


Figure 5: Differencing across singularity created by o-grid axis

Table 15: Variables in hole specification

Variable	Description
`ihmin`	minimum i-index of the hole
`ihmax`	maximum i-index of the hole
`jhmin`	minimum j-index of the hole
`jhmax`	maximum j-index of the hole
`khmin`	minimum k-index of the hole
`khmax`	maximum k-index of the hole

4.3 Exit Zones

Exit zones are regions of the grid adjacent to a boundary where additional work is done to damp outgoing waves that would otherwise reflect from the boundary and contaminate the interior solution. Typically in these zones, the grid spacing is gradually increased as the boundary is approached. This reduced resolution damps the waves. Additional dissipation is added through source terms or additional solution filtering. The dissipation is scaled such that it is zero at the beginning of the zone and a maximum at the boundary. Applying exit zones is not an exact science and some experimentation in damping parameters may be necessary. The exit zones are specified similarly to the hole points and follow them in each grid block's specifications. The first line is a comment line. The second line lists the number of exit zones. If no exit zones are required for this block, a zero must be used. For each exit zone a pair of lines are input; the first specifying the damping characteristics, and the second specifying the grid indices.

```
Start of Exit Zone Specification
maxez
ceza, cezb, ieztype, iezside
iezmin iezmax  jezmin jezmax  kezmin, kezmax
.
.
.
ceza, cezb, ieztype, iezside
iezmin iezmax  jezmin jezmax  kezmin, kezmax
```

Table 16 describes the variables in the specification. The source-term-type damping adds a source into the governing equations to force the flow towards the freestream values. The filtering-type damping applies additional filtering in the exit zone. The filtering appears to be more robust and has been used for the bulk of WRLES simulations. The strength of the damping is gradually increased over the distance of the exit zone until it is full strength at the boundary. The strength varies as $a(\Delta s)^b$, where a is a coefficient of damping, Δs is the normalized distance from the boundary and b is the exponent that ramps up the damping.

5 Data Processing

Scale-resolving simulations produce massive amounts of data. High-quality simulations contain tens or hundreds of million of data points per time step. WRLES has several mechanisms to collect and process this data within the code. The following sections describe these data processing options. All of these routines require alteration of the relevant subroutines in order to tailor the results to the problem at hand. It is important to note that the `implicit none` directive is used in all routines. If a new variable is created, it must be declared either in the subroutine (recommended) or in the data module.

Table 16: Variables in exit zone specification

Variable	Description
`ceza`	coefficient of damping
`cezb`	exponent of damping
`ieztype`	type of damping 1 = source terms 4 = 4th order filter 6 = 6th order filter
`iezside`	same as iside for boundary conditions
`iezmin`	minimum i-index of the exit zone
`iezmax`	maximum i-index of the exit zone
`jezmin`	minimum j-index of the exit zone
`jezmax`	maximum j-index of the exit zone
`kezmin`	minimum k-index of the exit zone
`kezmax`	maximum k-index of the exit zone

5.1 Time Averaging

Analysis of turbulent flows involves computing time averages of the various flow quantities. These averages are typically used to compute mean quantities, turbulence intensities, and higher order moments. Time averaging is controlled by the `itav` variable. Setting it to `1` will compute the time averages every `ntav` iterations. Setting it to `2` will compute the time averages every `ttav` seconds. Note that for this time-based sampling frequency, the code will perform the averaging when the solution time since the last average is greater than or equal to `ttav`. It will not adjust the time step in order to hit the sampling frequency exactly. It is recommended that the user exactly control the time step (Δt) by setting `acfl < 0`, and specifying the sampling frequency using the number of iterations.

Sample code showing the data needed to compute turbulence intensities, namely the averaging of the three velocity components and their squares, is presented below. First, the number of variables to be time averaged must be set in the `parameters.f` module. The following line must be modified to set `nvartav` to the correct value.

```
integer, parameter :: nvartav = 6
```

Next, the subroutine `tavew.f` must be modified to compute the variables to be averaged and then insert them into the running time-averaged array. Below, the three velocity components, u, v, and w, are computed from the solution vector. Then, the variable to be time averaged is multiplied by the factor `b` and summed into one component of the `qavg` array, which is multiplied by the factor `a`. The factors `a` and `b` are the time-weighted contributions of the previous and current flow values, respectively.

```
u             = q(i,j,k,2)/q(i,j,k,1)
v             = q(i,j,k,3)/q(i,j,k,1)
w             = q(i,j,k,4)/q(i,j,k,1)
qavg(i,j,k,1) = a*qavg(i,j,k,1) + b*u
qavg(i,j,k,2) = a*qavg(i,j,k,2) + b*v
qavg(i,j,k,3) = a*qavg(i,j,k,3) + b*w
qavg(i,j,k,4) = a*qavg(i,j,k,4) + b*u**2
qavg(i,j,k,5) = a*qavg(i,j,k,5) + b*v**2
qavg(i,j,k,6) = a*qavg(i,j,k,6) + b*w**2
```

The code will output a PLOT3D function file containing the time-averaged quantities for the entire flowfield. If a previous time-averaged file is found at startup, the code will continue averaging the current solution. If no file is found, the code will initialize a new time-averaged solution. Initializing a new time averaging file can also be forced by setting the input variable `itavin` to `1`.

5.2 Periodic Output

WRLES has the ability to periodically write files based on iteration or solution time. These files can be used to produce time histories, averages, or check point files.

5.2.1 Flowfield

The entire flowfield can be output by setting the variable `iffs`. Similar to time averaging, a value of `1` will output the flowfield every `nffs` iterations and a value of `2` will output the flowfield every `tffs` seconds of solution time. As with time averaging, the time increment is not exact. The initial solution file is written on the Fortran unit number specified by `isav`. Each subsequent file is incremented by one. If time averaging is activated, the averaged data is output on `10000+isav`. If the Spalart-Allmaras turbulence model is activated, the turbulence data is output on unit `20000+isav`. Note that if a new run is started, the initial unit number reverts to `isav`, so old files will be overwritten unless `isav` is adjusted.

The solution can also be periodically written on Fortran unit numbers as a checkpoint. In this instance the last solution is overwritten at the new time. To activate this feature `icp` is set to `1`. The frequency of the checkpoint save is specified by iteration only using `ncp`.

5.2.2 User Defined Data

Additional periodic output of user defined data is also possible. This option was created to enable saving time histories to produce spectra and other time dependent data. This feature is activated by setting the variable `iudat`. Similar to time averaging, a value of `1` will output the flowfield every `nudat` iterations and a value of `2` will output the flowfield every `tudat` seconds of solution time. As with time averaging, the time increment is not exact. The initial solution file is written on the Fortran unit number specified by `iudsav`.

The user must modify the `udatsavem.f` and `udatsavew.f` subroutines to suit their needs. In most cases, modifying these routines requires some knowledge of the Message Passing Interface (MPI). Routines ending in an `m` are run on the master process and do not have direct access to the data on the worker nodes. Routines ending in a `w` are run on the worker nodes. The data computed on the worker nodes must be passed to the master node via the MPI interface. Here it can be combined with the other nodes' data and output. An MPI buffer, `udbuf`, is set up for this purpose. The size of the buffer can be altered by changing the following statement in the `parameters.f` module.

```
integer, parameter :: udbufsize = 9*realsize + 5*intsize
```

The variables `realsize` and `intsize` specify the sizes of the real and integer numbers in WRLES. Setting the buffer to the correct size is done by simply changing the multiplier on these variables to the correct number of those variables in the buffer.

5.3 Post-Processing

The post-processing routines offer the user the ability to process the solution using the same numerics and framework as the flow solver. Most commercial post-processing software utilize low-order numerics to compute derivatives, which will not accurately reflect the flowfield produced by

the high-order differencing in WRLES. Boundary condition data is also readily available.

The subroutine `postw.f` performs the post-processing operations on each worker node. It can be modified to suit the user's needs. The array `post(i,j,k,n)` is intended to store the post-processed data at each grid point. The number of variables in the array (the last index) is specified in the `parameters.f` module by modifying the following line.

```
integer, parameter :: nvarpost = 5
```

This array is written as a PLOT3D function file. The released version of the code contains a `postw.f` routine that computes the velocity derivatives using the same differencing scheme as selected for the flow solver and outputs an array of vorticity, dilatation and magnitude of the strain rate tensor.

6 Parallelization

The code utilizes a hybrid parallelization paradigm which incorporates MPI and OpenMP. The standard mode of operation is to decompose the domain into multiple grid blocks. Each block is run on a separate compute node and data for the block interfaces is passed between nodes using MPI. On each node, parallelization over multiple processors and/or cores can be accomplished with OpenMP. Here, the work done within a Fortran `do` loop is divided across the available threads. Typically the user would specify one OpenMP thread per available core. The use of OpenMP where multiple processors or cores share the same memory eliminates the need to communicate data between blocks, increasing efficiency. Please note that MPI is required to run the code even if parallel computations are not necessary. The number of MPI processes required is equal to the number of grid blocks plus one. The first process is deemed the "master" and performs I/O and synchronization. The other processes are deemed the workers and perform the majority of the computations. For efficiency, it is desirable to have one MPI process per available processor, but this is not necessary.

For efficiency the work on each processor should be approximately equal; each grid block should have a similar number of points. Creating a WRLES input file for a highly parallelized blocked grid structure is not a trivial task, as the boundary conditions for each block must be specified manually, including the interfaces to the other blocks. It is recommended that the grid be generated in as few blocks as possible. This simplifies the task of generating the initial input file. A stand-alone Fortran code, `autosplit.f` is available to split an existing grid and corresponding input file into additional blocks for parallel processing.

The splitting code operates in two modes; automatic and manual. In both modes the code splits a grid and input, and has the option to split either a solution or function file. In automatic mode, `autosplit` partitions the grid based on the a user supplied input for the number of processors, and boundary overlap requirement. The partitioning logic is rudimentary and may not be optimal. These options are entered on the command line. It is recommended that the user experiment with the number of processors in the input, as small changes can significantly affect the efficiency of the partitioning. In manual mode, selected on the command line, the code reads a user supplied file specifying the partitioning. The file format for the user specified partitioning is below. The first

line contains: `nblocks` - the number of blocks in the new partitioned grid, `noverlap` - the number of grid planes that are overlapped between blocks, and `isolution` - an indicator to select split grid only (0), split grid and solution (1), or split grid and function file (2). Note that the grid and input file must always be supplied. The subsequent `nblocks` lines contain: `noldblock` - the number of the original block to be split, and the i, j, k minimum and maximum indices of the original block that make up the new partition.

```
nblocks      noverlap        isolution
noldblock    imin    imax    jmin    jmax    kmin    kmax
.
.
.
noldblock    imin    imax    jmin    jmax    kmin    kmax
```

Autosplit outputs a file that can be used with the code, unautosplit. This code reads that file, and the partitioned grid and solution/function file (optional) and recombines them into the original block structure. Table 17 lists the Fortran unit numbers for the various files input and output by autosplit.

Table 17: `autosplit` file unit numbers and formats

	Unit Number		
File	Input	Output	Format
original WRLES input file	1		ASCII text file
manual partitioning input	4		ASCII text file
original grid file	10		PLOT3D grid
original solution/function file	11		PLOT3D solution/function file
partitioned WRLES input file		2	ASCII text file
unautosplit input file		3	ASCII text file
partitioned grid file		20	PLOT3D grid
partitioned solution/function file		21	PLOT3D solution/function file

The OpenMP parallelization is done by splitting the work of the Fortran `do` loops across the available processors/cores. In most instances the outermost computational grid index, the k-index, is parallelized using the OpenMP compiler directives. The work in the inner nested loops is included and hence maximizes the work for each thread. The number of threads is specified via the environment variable `OMP_NUM_THREADS`. OpenMP is not necessary for compiling and/or running the code. If OpenMP is not used during the compilation process the OpenMP parallelization directives are ignored. An example of a parallelized loop is

```
!$OMP PARALLEL DO PRIVATE(k,j,i)
      do k=1,mk(nb)
      do j=1,mj(nb)
      do i=1,mi(nb)
        dudxi(i,j,k,1,1) = (cm1*(u(i+1,j,k,1)-u(i-1,j,k,1))
     &                     +cm2*(u(i+2,j,k,1)-u(i-2,j,k,1))
     &                     +cm3*(u(i+3,j,k,1)-u(i-3,j,k,1))
     &                     +cm4*(u(i+4,j,k,1)-u(i-4,j,k,1))
     &                     +cm5*(u(i+5,j,k,1)-u(i-5,j,k,1))
     &                     +cm6*(u(i+6,j,k,1)-u(i-6,j,k,1)))
      end do
      end do
      end do
```

7 Running

The following C-shell script provides a basic example of how to run WRLES. This script must be adapted to your specific computational platform. Note that not all files are required for a particular run.

```
#!/bin/csh
#
# set environment variable to allocate number of OpenMP threads
#
setenv OMP_NUM_THREADS 5
#
# link input files to fortran units
#
ln input fort.1
ln grid_in.x fort.10
ln soln_in.q fort.11
ln alg_turb_in.fun fort.12
ln sa_turb_in.fun fort.13
ln tavg_in.fun fort.15
ln turb_bc.dat fort.101
#
# run wrles
#
mpirun -np 2 wrles > output
#
# give output files descriptive names
#
mv fort.20 grid_out.x
mv fort.21 soln_out.q
mv fort.22 alg_turb_out.fun
mv fort.23 sa_turb_out.fun
mv fort.25 tavg_out.fun
mv fort.27 metrics.fun
mv fort.28 cellsize.fun
#
# remove input file fortran units
#
rm -f fort.1
rm -f fort.10
rm -f fort.11
rm -f fort.12
rm -f fort.13
rm -f fort.15
rm -f fort.16
```

8 Examples

Two basic examples are provided to show the user how to set up a typical input file.

8.1 Flat Plate

A sample input file is given for a simple three-dimensional flat plate case. The freestream Mach number is 0.2 and the freestream static pressure and temperature are 2116.8 psf and 519 R. A schematic of the computational boundaries is shown in figure 6. The grid for this case is a single block that has the maximum dimensions of 101x61x31 for the i-, j-, and k-directions, respectively. The plate surface is represented by the j-min boundary ($j = 1$). A short section of inviscid wall is specified upstream of the plate to simulate a freestream flow, and the plate starts at $i = 21$. Spanwise periodicity is set up by specifying periodic boundaries for the k-min and -max boundaries. The inflow at $i = 1$ is set to hold total pressure and total temperature constant (subsonic inflow boundary). The outflow is set to hold the static pressure constant (subsonic outflow). A freestream characteristic-based boundary is set on the j-max boundary. This boundary is angled relative to the freestream velocity. This removes any ambiguity as to the flow direction at the boundary, which can cause numerical problems.

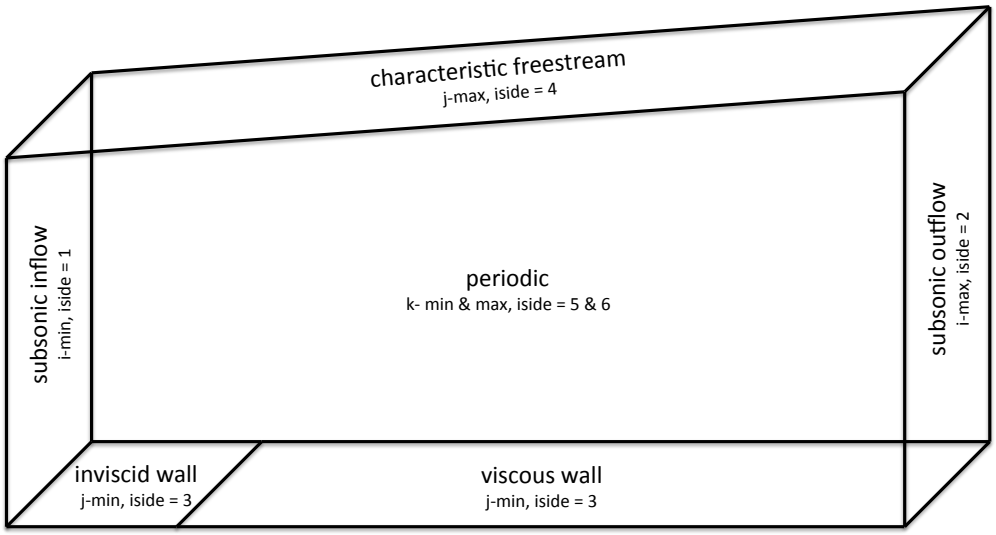


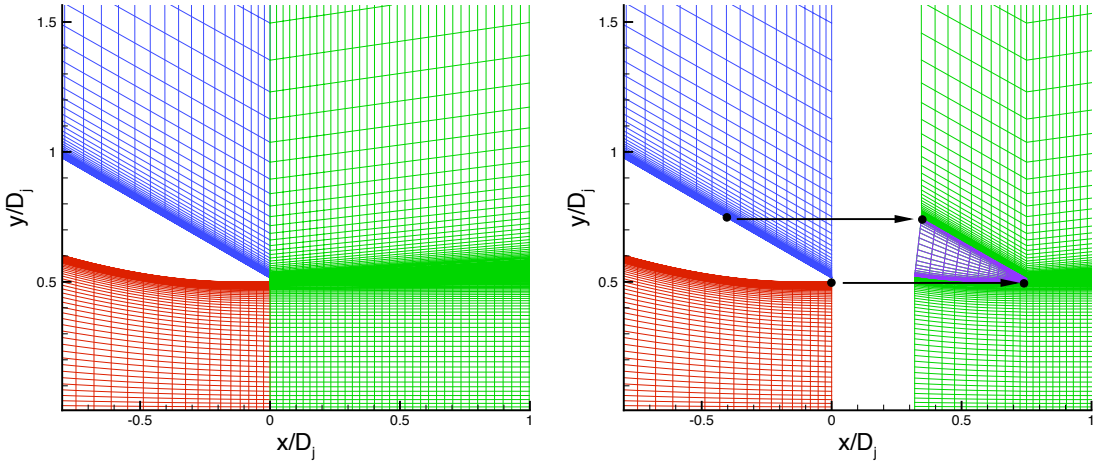
Figure 6: Flat plate schematic

```
$inputs
 niter  = 100
 acfl   = 0.7
 isch   = 43
 iorder = 13
 idmp   = 113
 sigma  = 0.25
 fsmach = 0.2
 pinf   = 2116.8
 tinf   = 519.00
 iinit  = 1
/
Boundary Conditions
7
1    1    1   61    1   31   7   0  2176.7  519.15
2  101    1   61    1   31  11   0  2116.8
3    1    1   20    1   31  21   0
3    1   21  101    1   31  22   0
4   61    1  101    1   31   3   0
5    1    1  101    1   61  41   0
6   31    1  101    1   61  41   0
Holes
0
Exit Zones
0
```

8.2 Nozzle

The following case is a more complex example representative of the cases typically run with WRLES. The case represents a jet flow emanating from a round nozzle. The grid consists of three grid blocks: the internal nozzle flow, the freestream inflow over the nozzle, and the jet region downstream of the nozzle. The overlapping grid requirement complicates the grid blocking topology, as there are multiple ways to represent the geometry. This example shows the recommended practice which avoids any ill-defined interface problems (figure 7). It is recommended that the downstream block be constructed to overlap into the two upstream blocks. This overlapped region must include the empty space between the two upstream blocks. The grid points that fall in the empty space are designated as hole points and are not computed. Figure 7 shows a detail of this region. The overlapped region, hole points (in purple), and coincident points can be seen in the exploded view.

For this case, the sizes of the nozzle, freestream inflow and jet region blocks are 45x55x132, 60x65x132, and 435x148x132, respectively. The overlapped blocks share 12 common planes. The nozzle lip is modeled with 30 points in the j-direction. The grids form a cylindrical shape using an o-grid topology. The centerline is treated using a type 33 boundary, which computes finite differences across the centerline. An exit zone that uses a sixth-order filter is located over the last 26 points of the domain in block 3. More details on this approach using WRLES can be found in reference [6].



(a) Nozzle grid detail

(b) Exploded view of nozzle grid

Figure 7: Nozzle grid blocking scheme

```
$inputs
 niter  = 100000
 acfl   = 0.7
 isch   = 43
 iconsv = 0
 iorder = 13
 idmp   = 123
 sigma  = 0.25
 ivisc  = 1
 fsmach = 0.05
 pinf   = 2116.8
 tinf   = 519.0
 iinit  = 0
/
Block   1 - Internal Nozzle
6
1    1    1   55    1  132    1   0  2332.7  941.40
2   45    1   55    1  132   51   1
3    1    1   45    1  132   33   0
4   55    1   45    1  132   22   0
5    1    1   45    1   55   41   0
6  132    1   45    1   55   41   0
 Holes
0
 Exit Zones
0
 Block   2 - Freestream Inflow
6
1    1    1   65    1  132    1   0  2120.5  519.26
2   60    1   65    1  132   51   2
3    1    1   60    1  132   22   0
4   65    1   60    1  132    3   0
5    1    1   60    1   65   41   0
6  132    1   60    1   65   41   0
Holes
0
Exit Zones
0
Block   3 - Jet Region
10
1    1    1   55    1  132   51   1
4   55    1   12    1  132   22   0
1   12   55   84    1  132   22   0
3   84    1   12    1  132   22   0
1    1   84  148    1  132   51   2
3    1    1  435    1  132   33   0
4  148    1  435    1  132    3   0
5    1    1  435    1  148   41   0
6  132    1  435    1  148   41   0
2  435    1  148    1  132   11   0  2116.8
Holes
1
1   11   56   83    1  132
Exit Zones
1
0.100 1.000  6  2
410  435    1  148    1  132
```

9 Recommended Practices

This section contains useful advice on how to increase the likelihood of a successful simulation. This is based on the author's experience running the WRLES code for several applications, but is by no means a definitive set of rules.

1. The accuracy of a scale-resolving turbulent simulation is directly tied to the resolution of the computational grid. Extremely fine grids must be used in order to resolve the turbulent structures and the cascade of turbulent energy from the large scales down to the dissipative scales. Grid refinement should be done as a matter of course. Choi and Moin [15] provide a means of estimating the required number of grid points based on the Reynolds number of the flow. In addition, Georgiadis, Rizzetta, and Fureby [45] provide the following grid spacings in wall units for well resolved boundary layers. For large-eddy simulation

$$50 \leq \Delta x^+ \leq 150, \quad \Delta y^+ < 1, \quad 15 \leq \Delta z^+ \leq 40 \tag{78}$$

   and for direct numerical simulation

$$10 \leq \Delta x^+ \leq 20, \quad \Delta y^+ < 1, \quad 5 \leq \Delta z^+ \leq 10 \tag{79}$$

2. Grid stretching and grid skewness should be kept to an absolute minimum. Grid stretching, the increase in size of adjacent cells, should be less than 5 percent in regions of interest.

3. The grid should be generated in as few blocks as possible in order to simplify the generation of the input file. Use the `autosplit.f` utility to partition the grid and input file for execution on parallel computational platforms.

4. WRLES is written to handle central differencing schemes with stencils up to 13 points. If a lower-order/resolution scheme is selected, the full 13-point stencil is still computed with the outer coefficients set to zero. Hence, there is no computational speed to be gained with the lower order schemes. However, the grid overlap necessary for block to block communication is less. This may improve code efficiency, but it has not been quantified. For this reason, the DRP schemes or the 10th- or 12th-order standard schemes are suggested to maximize resolution and minimize error. For the DRP schemes, their matching solution filter should be selected. For the standard schemes, the filter of the same order, or one order higher should be chosen. The filter coefficient should be initially set to one.

5. When starting a simulation, select the time step by specifying the CFL number. For most of the schemes in the code, a stable CFL number should be in the range of 0.3 to 1.2. The CFL is highly dependent on the flowfield and grid. The CFL may need to be set to a lower value in the initial stages while startup transients are being convected out of the domain. However, if the maximum stable CFL number is too low after the initial transients have cleared, consider refining or smoothing the grid. If the initial transients cause numerical instabilities that cannot be successfully bypassed, consider using a solution from a RANS code as the initial conditions. Also, the boundary conditions can be modified to reduce the initial transients. For example, when a subsonic inflow boundary is used, the total pressure could be slowly adjusted upward to alleviate a large gradient at the inflow that will propagate downstream. The minimum static pressure written to the output file is a good indicator of the solution's

stability. Rapid drops in pressure typically indicate the solution is diverging or "blowing up." The location of the minimum pressure, also written to the output, can help the user diagnose the cause of the instability.

6. Once a stable CFL number has been found, the filter coefficient should be systematically halved and evaluated until the minimum stable value is found. This minimizes the numerical dissipation in the solution, a key to a good LES.

7. Once a stable solution has been demonstrated with the current CFL and filter coefficient, and all transients have been convected out of the computational domain, time averaging and other data extraction can begin.

8. When restarting a simulation from a previous run, the files output from the previous run are used as inputs to the new run. The solution file, Spalart-Allmaras turbulence file, and time averaging file must be copied or moved to their input file counterparts, or input Fortran unit numbers.

9. The variables `iinit`, `itres`, and `itavin` must be used with extra care. These variable reinitialize the solution, solution time, and time averaging, respectively. They must be removed or set to zero before the next run, otherwise they will continue to perform the reinitialize function.

References

[1] Spalart, P. R. and Allmaras, S. R., "A One-Equation Turbulence Model for Aerodynamic Flows," AIAA Paper 92-439, 1992.

[2] Menter, F. R., "Zonal Two Equation k - ω Turbulence Models for Aerodynamic Flows," AIAA Paper 93-2906, 1993.

[3] DeBonis, J. R. and Scott, J. N., "A Large-Eddy Simulation of a Turbulent Compressible Round Jet," *AIAA Journal*, Vol. 40, No. 5, 2002, pp. 1346–1354.

[4] DeBonis, J. R., "A Large-Eddy Simulation of a High Reynolds Number Mach 0.9 Jet," AIAA Paper 2004-3025, 2004.

[5] DeBonis, J. R., "An Examination of the Spatial Resolution Requirements for LES of a Compressible Jet," *Quality and Reliability of Large-Eddy Simulation II*, edited by M. V. Salvetti, B. Geurts, H. Meyers, and P. Sagaut, Springer, 2010, pp. 329–338.

[6] DeBonis, J. R., "A High-Resolution Capability for Large-Eddy Simulation of Jet Flows," AIAA Paper 2010-5023, 2010.

[7] DeBonis, J. R., "Prediction of Turbulent Temperature Fluctuations in Hot Jets," *AIAA Journal*, Vol. 56, No. 8, 2018, pp. 3097–3111.

[8] Mankbadi, M., Georgiadis, N., and DeBonis, J., "Comparison of High-Order and Low-Order Methods for Large-Eddy Simulation of a Compressible Shear Layer," AIAA Paper 2015-2939, 2015.

[9] Mankbadi, M., DeBonis, J., and Georgiadis, N., "Large-Eddy Simulation of a Compressible Mixing Layer and the Significance of Inflow Turbulence," AIAA Paper 2017-0316, 2017.

[10] DeBonis, J., "Solutions of the Taylor-Green Vortex Problem Using High-Resolution Explicit Finite Difference Methods," AIAA Paper 2013-0382, 2015.

[11] Mankbadi, M., Vyas, M., DeBonis, J., and Georgiadis, N., "Evaluation of Inflow Turbulence Methods in Large-Eddy Simulations of a Supersonic Boundary Layer," AIAA Paper 2018-3404, 2018.

[12] Caraballo, E., Samimy, M., and DeBonis, J., "Low Dimensional Modeling of Flow for Closed-Loop Flow Control," AIAA Paper 2003-0059, 2003.

[13] Samimy, M., Debiasi, M., Caraballo, E., Ozbay, H., Efe, M., Yuan, X., DeBonis, J., and Myatt, J., "Development of Closed-loop Control for Cavity Flows," AIAA Paper 2003-4258, 2003.

[14] Kolmogorov, A. N., "Local Structure of Turbulence in Incompressible Viscous Fluid for Very Large Reynolds Number," *Doklady Akademiya Nauk SSSR*, Vol. 30, 1941, pp. 299–303.

[15] Choi, H. and Moin, P., "Grid-Point Requirements for Large Eddy Simulation: Chapman's Estimates Revisited," *Physics of Fluids*, Vol. 24, No. 1, January 2012, pp. 011702–011705.

[16] Baldwin, B. S. and Lomax, H., "Thin Layer Approximation and Algebraic Model for Separated Turbulent Flows," AIAA Paper 78-257, 1978.

[17] Allmaras, S. R., Johnson, F., and Spalart, P. R., "Modifications and Clarifications for the Implementation of the Spalart-Allmaras Turbulence Model," ICCFD Paper ICCFD7-1902, 2012, Seventh International Conference on Computational Fluid Dynamics (ICCFD7).

[18] Nelson, C. C., *Simulations of Spatially Evolving Compressible Turbulence Using a Local Dynamic Subgrid Model*, Ph.D. thesis, Georgia Institute of Technology, 1997.

[19] Smagorinsky, J., "General Circulation Experiments with the Primitive Equations, Part I: The Basic Experiment," *Monthly Weather Review*, Vol. 91, 1963, pp. 99–152.

[20] Van Driest, E. R., "On Turbulent Flow Near a Wall," *Journal of the Aeronautical Sciences*, Vol. 23, 1956, pp. 1007–1011.

[21] Lilly, D. K., "A Proposed Modification of the Germano Subgrid-Scale Closure Method," *Physics of Fluids A*, Vol. 4, No. 3, 1992, pp. 633–635.

[22] Vreman, A., "An Eddy-Viscosity Subgrid-Scale Model for Turbulent Shear Flow: Algebraic Theory and Applications," *Physics of Fluids*, Vol. 16, No. 10, 2004, pp. 3670–3681.

[23] Vichnevetsky, R. and Bowles, J. B., *Fourier Analysis of Numerical Approximations of Hyperbolic Equations*, Society for Applied and Industrial Mathematics, 1982.

[24] Tam, C. K. W. and Webb, J. C., "Dispersion Relation-Preserving Finite Difference Schemes for Computational Aeroacoustics," *Journal of Computational Physics*, Vol. 107, 1993, pp. 262–281.

[25] Tam, C. K. W. and Shen, H., "Direct Computation of Nonlinear Acoustic Pulses Using High Order Finite Difference Schemes," AIAA Paper 93-4325, 1993.

[26] Bogey, C. and Bailly, C., "A Family of Low Dispersive and Low Dissipative Explicit Schemes for Flow and Noise Computations," *Journal of Computational Physics*, Vol. 194, 2004, pp. 194–214.

[27] Kennedy, C. A. and Carpenter, M. H., "Comparison of Several Numerical Methods for Simulation of Compressible Shear Layers," NASA TP 3484, 1997.

[28] Bogey, C., de Cacqueray, N., and Bailly, C., "A Shock-Capturing Methodology Based on Adaptative Spatial Filtering for High-Order Non-Linear Computations," *Journal of Computational Physics*, Vol. 228, 2009, pp. 1447–1465.

[29] Jameson, A., Schmidt, W., and Turkel, E., "Numerical Solutions of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time-Stepping Schemes," AIAA Paper 81-1259, 1981.

[30] Carpenter, M. H. and Kennedy, C. A., "Fourth-Order 2N-Storage Runge-Kutta Schemes," NASA TM 109112, 1994.

[31] Hu, F. Q., Hussaini, M. Y., and Manthey, J. L., "Low-Dissipation and Low-Dispersion Runge-Kutta Schemes for Computational Acoustics," *Journal of Computational Physics*, Vol. 124, 1996, pp. 177–191.

[32] Gottlieb, S. and Shu, C. W., "Total Variation Diminishing Runge-Kutta Schemes," NASA CR 201591, 1996.

[33] Stanescu, D. and Habashi, W. G., "2N-Storage Low Dissipation and Dispersion Runge-Kutta Schemes for Computational Acoustics," *Journal of Computational Physics*, Vol. 143, No. 12, 1998, pp. 674–681.

[34] Berland, J., Bogey, C., and Bailly, C., "Low-Dissipation and Low-Dispersion Fourth-Order Runge-Kutta Algorithm," *Computers and Fluids*, Vol. 35, 2006, pp. 1459–1463.

[35] Allampalli, V., Hixon, R., Nallasamy, M., and Sawyer, S., "High-Accuracy Large-Step Explicit Runge-Kutta (HALE-RK) Schemes for Computational Aeroacoustics," *Journal of Computational Physics*, Vol. 228, 2009, pp. 3837–3850.

[36] Williamson, J. H., "Low-Storage Runge-Kutta Schemes," *Journal of Computational Physics*, Vol. 35, 1980, pp. 48–56.

[37] MacCormack, R. W., *Numerical Solution of the Interaction of a Shock Wave with a Laminar Boundary Layer*, Vol. 8 of *Lecture Notes in Physics*, Springer-Verlag, 1971, pp. 151–163, Proceedings of the Second International Conference on Numerical Methods in Fluid Dynamics.

[38] Anderson, D. A., Tannehill, J. C., and Pletcher, R. H., *Computational Fluid Mechanics and Heat Transfer*, Taylor & Francis, 1984.

[39] Vinokur, M., "Conservation Equations of Gas-Dynamics in Curvilinear Coordinate Systems," *Journal of Computational Physics*, Vol. 14, 1974, pp. 105–125.

[40] Hixon, R., Shih, S. H., Dong, T., and Mankbadi, R. R., "Evaluation of Generalized Curvilinear Coordinate Transformations Applied to High-Accuracy Finite-Difference Schemes," AIAA Paper 98-0370, 1998.

[41] Walatka, P., Buning, P., Pierce, L., and Elson, P., "PLOT3D User's Manual," NASA TM 101067, 1990.

[42] Reynolds, A., *Turbulent Flows in Engineering*, John Wiley & Sons, 1974, pp. 249–254.

[43] Jarrin, N., Benhamadouche, S., Laurence, D., and Prosser, R., "A Synthetic-Eddy-Method for Generating Inflow Conditions for Large-Eddy Simulations," *International Journal of Heat and Fluid Flow*, Vol. 27, No. 4, 2006, pp. 585–593.

[44] Xie, Z. and Castro, I., "Efficient Generation of Inflow Conditions for Large Eddy Simulation of Street-Scale Flows," *Flow, Turbulence and Combustion*, Vol. 81, 2008, pp. 449–470.

[45] Georgiadis, N., Rizzetta, D., and Fureby, C., "Large-Eddy Simulation: Current Capabilities, Recommended Practices, and Future Research," AIAA Paper 2009-948, 2009.