Eilmer: an open-source multi-physics hypersonic flow solver

Nicholas N. Gibbons[a], Kyle A. Damm[a], Peter A. Jacobs[a], Rowan J. Gollan[a,*]

[a]*Centre for Hypersonics, School of Mechanical & Mining Engineering, The University of Queensland*

Abstract

This paper introduces Eilmer, a general-purpose open-source compressible flow solver developed at the University of Queensland, designed to support research calculations in hypersonics and high-speed aerothermodynamics. Eilmer has a broad userbase in several university research groups and a wide range of capabilities, which are documented on the project's website, in the accompanying reference manuals, and in an extensive catalogue of example simulations. The first part of this paper describes the formulation of the code: the equations, physical models, and numerical methods that are used in a basic fluid dynamics simulation, as well as a handful of optional multi-physics models that are commonly added on to do calculations of hypersonic flow. The second section describes the processes used to develop and maintain the code, documenting our adherence to good programming practice and endorsing certain techniques that seem to be particularly helpful for scientific codes. The final section describes a half-dozen example simulations that span the range of Eilmer's capabilities, each consisting of some sample results and a short explanation of the problem being solved, which together will hopefully assist new users in beginning to use Eilmer in their own research projects.

Keywords: Scientific Computing; Computational Fluid Dynamics; Hypersonics; Parallel Computing

PROGRAM SUMMARY

Program Title: Eilmer

CPC Library link to program files: (to be added by Technical Editor)

Developer's repository link: https://github.com/gdtk-uq/gdtk

Code Ocean capsule: (to be added by Technical Editor)

Licensing provisions(please choose one): GPLv3

Programming language: D, Lua

Supplementary material: https://gdtk.uqcloud.net

Nature of problem:

Eilmer solves the compressible Navier-Stokes equations with a particular emphasis on flows at hypersonic speeds. The code includes modelling for high-temperature gas effects such as chemical and vibrational nonequilibrium. Eilmer can be used for the simulation for unsteady and steady flows.

Solution method:

The code is implemented in D [1] and built on a finite-volume formulation that is capable of solving the Navier-Stokes equations in 2D and 3D computational domains, discretised with structured or unstructured grids. Grids may be generated using a built-in parametric scripting tool or imported from commercial gridding software. The inviscid fluxes are computed using the reconstruction-evolution approach. In structured-grid mode, reconstruction stencils up to fourth-order spatial accuracy are available. In unstructured-grid mode, least-squares reconstruction provides second-order spatial accuracy. A variety of flux calculators are available in the code. Viscous fluxes are computed with compact stencils with second-order spatial accuracy. For unsteady flows, explicit time-stepping with low-order RK-family schemes are available, along with a point-implicit Backward-Euler update scheme for stiff systems of equations. For steady flows convergence can be greatly accelerated Jacobian-free Newton-Krylov update scheme, which seeks a global minimum in the residuals using a series of large pseudo-timesteps. Domain decomposition is used for parallel execution using both shared memory and distributed memory programming techniques.

Additional comments:

Eilmer provides a programmable interface for pre-processing, post-processing and user run-time customisations. The programmable interface is enabled using a built-in embedded interpreter for the Lua programming language [2]. Run-time customisations include used-defined boundary conditions, source terms and grid motion.

References

[1] D Programming Language web page: https://dlang.org/

[2] Lua Programming Language web page: https://www.lua.org/

*Corresponding author.

E-mail address: r.gollan@uq.edu.au

1. Introduction

Hypersonic flight is a frontier of aerodynamic science that presents us with many challenges. Some arise from the physical complexity of the fluid mechanics involved: the presence of shockwaves, turbulence, thermochemical nonequilibrium, radiation, and other phenomena that prevents us from developing a simple theory for the flow around a hypersonic vehicle. Pushing from the other direction are problems of operational requirements: the engineering and practical considerations that must be solved to build an efficient flying machine that can survive the extreme conditions, and with enough mass to spare for a useful payload.

In spite of these challenges, recent successes in hypersonic flight have shown that the challenges can be overcome, thanks in part to the testing, simulation, and design tools accumulated by the research community over many decades of work. These tools include flight testing of expendable models, ground testing in hypersonic wind tunnels, and computer modelling and simulation, chiefly, the use of computational fluid dynamics (CFD).

The role of CFD in the study of hypersonic flows is now ubiquitous. CFD is used in support of ground testing, flight testing, and, increasingly, on its own as a numerical wind tunnel. In ground testing, CFD is used in the design and analysis of test articles, and in the design and analysis of the test facilities themselves. For example, hypersonic test facilities typically cannot directly measure everything about the test flow they produce, and rely on CFD to reconstruct the full flow conditions delivered during the experiment. Flight testing is performed infrequently due to the time, expense and risk involved. When flight testing is undertaken, CFD is used in the design stage and as part of proving flight-readiness. With advances in the accuracy and robustness of CFD algorithms and the power of computers, it is now possible to resolve all time and length scales of a wall-bounded turbulent hypersonic flow. When used in this manner, CFD becomes a numerical wind tunnel, and, as such, a tool for investigation of flow physics. This mode of CFD usage is quite different from engineering analysis, which aims to support decision making. This list is just a small sample of the roles for CFD in advancing hypersonic vehicle design.

The varied roles for CFD in the study of hypersonic flow present a challenge for the development of a CFD tool. The strategies for addressing this challenge fall broadly into two approaches: either develop a niche code to tackle one application; or develop a comprehensive code that has a broad range of capabilities but less specificity. The advantages of the niche code approach include: quick development time; limited feature set to verify and validate; use of the best algorithms and computer architectures for the application; and a relatively small surface area for code maintenance. The principal disadvantage of this approach, the niche bespoke code, is one of reusing the code for new applications. As such, the niche code approach can face limitations in versatility such as restricted sets of boundary conditions, limitations on geometric complexity, and limitations on physical modelling capability. In some cases, these codes cannot be easily modified for new applications because the code is so highly customised towards its original purpose.

At the other extreme, the strategy involves building a comprehensive code that serves multiple purposes. The advantages of comprehensive code development include: that developers and users learn one code base; synergies in code re-use for varied applications (such as I/O, message passing, geometry routines and gas modelling); and relative ease to add new modelling capability not originally envisaged. The disadvantages when contrasted to the niche code are: long development time; large modelling capability that requires verification and validation (see Kleb and Wood [1] for a discussion of this issue); trade-offs on algorithm choices against the need for generalisation; and a large surface area of code for maintenance.

These approaches to development fall on a continuum, and, likely, any comprehensive code began life with a niche purpose. We are not advocating one approach over the other here as each has their merits relative to circumstances. For example, in academia, the niche code approach makes a lot of sense in a world of short funding cycles. The niche code is also extremely useful when developing a completely novel algorithm. By contrast, the comprehensive code approach is better suited to government and industry institutions where funding time scales are typically longer. We present this view on simulation code development approaches to give context to the work presented in this paper. Here we describe the simulation code Eilmer. It is a comprehensive code for hypersonic multi-physics simulation that has been developed for 30 years and, principally, that development has been at The University of Queensland.

The advantages and disadvantages as described are mostly concerns for the developer. When viewed from a user's perspective, the benefits of a comprehensive code are clearer. These benefits are a reduction in the investment learning-curve time by using one tool for a variety of applications; and a larger community of users for

support. As mentioned earlier, a hypersonics engineer has to use CFD in a variety of situations at varied levels of modelling fidelity. So, there is good motivation for a comprehensive simulation code for hypersonic flows.

The purpose of this paper is to introduce Eilmer as a comprehensive hypersonic flow solver with multi-physics capability. The timing of this paper coincides with a version 4.0 release of the code. We wish to show that there are few open-source CFD tools with hypersonic multi-physics simulation capability. We also aim to demonstrate that Eilmer is supported for general use with extensive documentation. Furthermore, modelling capability can be augmented by users in two ways: by user-defined runtime customisations written in the scripting language Lua; or by direct modification to the source code.

There has been other recent activity in the development of open-source flow solvers for the compressible flow regime. We restrict our attention to open-source offerings since we believe these offer the best opportunity for code re-use and re-purpose. The STREAmS code [2] is a specialised solver for performing direct numerical simulation on wall-bounded compressible flows. The code is specialised both in terms of algorithms and computer architectures. It is designed for high-resolution calculations and achieves this by demonstrating excellent scaling on multi-GPU HPC platforms (up to 1024 GPUs). STREAmS is restricted to Cartesian domains. For treating geometric complexity, Romero et al. have released ZEFR [3] as an open-source code to simulate compressible viscous flows. ZEFR is also targeted towards high-resolution capability and the associated computational demands are met by developing for multi-GPU architectures, as for STREAmS. OpenS-BLI [4] is another open-source code that falls into the category of scale-resolving flow codes, accelerated by GPUs. OpenSBLI is actually a code-generation system to produce code for heterogeneous architectures on modern HPC platforms. In terms of algorithms, it is based on high-order finite differences on structured curvilinear grids, with both WENO and TENO schemes for shock-capturing. All of these codes simulate ideal gases. This is a limitation for application in the high-temperature hypersonic flow regimes. By contrast, the HTR solver [5] by Di Renzo et al. include modelling of thermochemical nonequilibrium effects typical in hypersonic flows. HTR is also aimed at very high-resolution calculations for DNS work, and is similarly accelerated by implementation for multi-GPU architectures. All of these codes could be considered quite niche in terms of application.

The hy2Foam code [6] is an example of an open-source hypersonic flow solver that is quite general purpose. Built on top of OpenFOAM, hy2Foam has access to numerical discretization algorithms to treat a variety of geometric complexity. The hy2Foam authors added physical modelling in terms of nonequilibrium thermochemical effects so that the code can be applied in the high-temperature flow regime. Another open-source project for multi-physics simulation in the hypersonic flow regime is SU2-NEMO [7]. This code, built as an extension to SU2 [8], is an unstructured solver so that objects with high geometric complexity can be simulated. It has modelling for high-temperature nonequilibrium effects, such as finite-rate chemistry and two-temperature thermal relaxation, which are required for providing engineering estimates of flow field quantities in high Mach number flight.

The Eilmer code, presented in this paper, is a comprehensive open-source code for multi-physics simulation of hypersonic flows fit for multiple purposes. We believe this range of physical modelling capability, combined with the open-source nature and active development of the project, makes a novel contribution to the academic CFD simulation community. Eilmer offers capabilities for time accurate simulation and accelerated steady-state analyses. Computational domains of considerable geometric complexity are handled with multiple-block structured or unstructured grids. For simulating high-temperature effects, the gas models include mixtures of thermally perfect gases, multi-temperature Boltzmann gases, and state-to-state models for extreme nonequilibrium. The code includes modelling of the nonequilibrium relaxation via finite-rate chemistry modules and thermal energy exchange models. Multi-physics simulation capability includes fluid-structure interactions, fluid-thermal interactions, and fluid-radiation coupling. Aerodynamic shape optimisation with many design parameters is enabled by an adjoint-solver. Most of these capabilities can be customised for special purpose via user-defined runtime controls.

In this paper, we present the Eilmer code showing what it offers to the community in terms of application and re-use. In Section 2, the core numerical formulation based on a finite-volume approach is presented. To build trust in the quality and longevity of the code, we discuss the development process in Section 3, including quality control via verification and validation. Section 4 presents the parallelisation algorithm used to run simulations on distributed computational hardware, and includes some test data to prove good scaling performance up to 1000 physical cores. We have mentioned that Eilmer supports a variety of use cases. In Section 5, we

present examples of those uses cases: engineering analysis, optimised aerodynamic design, and flow physics investigation. An unusual aspect of Eilmer as a research code is that it is used in the classroom; we include an example of student simulations in Section 5 also.

2. Formulation

The core of Eilmer is a set of numerical routines for solving mixed hyperbolic/parabolic partial differential equations (PDEs) that have been discretised on a grid of finite-volume cells. These PDEs are the compressible Navier-Stokes equations for viscous flow in two or three dimensions, augmented with various add-ons, simplifications, and optional extras for including additional physics. Rather than exhaustively document the solver's entire suite of capabilities, in this section we present the core flow equations and the foundational assumptions underlying them, followed by three of the most common multi-physics capabilities that are commonly used in hypersonics research calculations. Additional details about any of these topics can be found in the code's online documentation or the cited references.

2.1. The Navier-Stokes Equations

Equation 1 summarises the fundamental set of conservation equations of compressible fluid flow, expressed in conservative, integral form. They consist of a vector of conserved quantities $\mathbf{U}$ that is tracked inside of a small volume V, which changes in time as each quantity flows through the surfaces of the volume, as determined by the fluxes $\mathbf{F}$, or generated internally by a source term $\mathbf{Q}$.

$$\frac{\partial}{\partial t} \int_V \mathbf{U} \, dV = - \oint_S (\mathbf{F}_i - \mathbf{F}_v) \cdot \hat{n} \, dA + \int_V \mathbf{Q} \, dV \quad (1)$$

Embodied in these equations are two foundational assumptions that underlie all simulations performed with Eilmer. The first is that the flow is modelled as a *continuum*: A field with values at each point in space rather than a discrete collection of molecules. The second is that the fields are *compressible*, in the sense that information about a disturbance must travel in waves at a finite speed from the point of the disturbance. The first assumption is valid when the particles making up the fluid are so small and so numerous that they form a well behaved statistical ensemble, though the assumption can break down in flows that are very cold, very

low density, or interacting with a very small object. The second assumption is appropriate to flows of any velocity, although a low speed flow is much more efficiently modelled by an incompressible formulation where the signal speed can be approximated as infinitely fast.

For a nonreacting flow in three-dimensions, with no additional physics present, the conserved quantities are the mass, x-momentum, y-momentum, z-momentum, and energy — or more precisely, the density per unit volume of these quantities.

$$\mathbf{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{bmatrix} \quad (2)$$

The inviscid flux vector $\mathbf{F}_i$ tracks the movement of conserved quantities due to bulk fluid motion, and consists of elements that are each in turn vectors in three-dimensional space.

$$\mathbf{F}_i = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho vu \\ \rho wu \\ \rho Eu + pu \end{bmatrix} \hat{i} + \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho wv \\ \rho Ev + pv \end{bmatrix} \hat{j} + \begin{bmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ \rho Ew + pw \end{bmatrix} \hat{k} \quad (3)$$

These fluxes are distinguished from the viscous fluxes, which arise from nano-scale disordered motion of the particles making up the fluid, and tend to involve derivatives of the primitive quantities:

$$\mathbf{F}_v = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{yx} \\ \tau_{zx} \\ \tau_{xx}u + \tau_{yx}v + \tau_{zx}w + q_x \end{bmatrix} \hat{i} + \begin{bmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ \tau_{zy} \\ \tau_{xy}u + \tau_{yy}v + \tau_{zy}w + q_y \end{bmatrix} \hat{j}$$

$$+ \begin{bmatrix} 0 \\ \tau_{xz} \\ \tau_{yz} \\ \tau_{zz} \\ \tau_{xz}u + \tau_{yz}v + \tau_{zz}w + q_z \end{bmatrix} \hat{k}$$

$$(4)$$

where the viscous stress tensor τ is composed from derivatives of the velocity vector, the viscosity μ and the volume (or bulk) viscosity λ.

$$\tau_{xx} = 2\mu\frac{\partial u}{\partial x} + \lambda\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right)$$

$$\tau_{yy} = 2\mu\frac{\partial v}{\partial y} + \lambda\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right)$$

$$\tau_{zz} = 2\mu\frac{\partial w}{\partial z} + \lambda\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right)$$

$$\tau_{xy} = \tau_{yx} = \mu\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \tag{5}$$

$$\tau_{yz} = \tau_{zy} = \mu\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}$$

$$\tau_{xz} = \tau_{zx} = \mu\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}$$

The code generally assumes that λ can be evaluated using Stokes' hypothesis as $\lambda = -\frac{2}{3}\mu$. The heat conduction vector q is modelled using Fourier's law and the gradient of the temperature T, with conductivity κ.

$$q_x = \kappa\frac{\partial T}{\partial x}$$

$$q_y = \kappa\frac{\partial T}{\partial y} \tag{6}$$

$$q_z = \kappa\frac{\partial T}{\partial z}$$

The transport properties μ and κ can be computed by any of several of different models depending the needs of the user. For compressible flows with simple thermochemistry Sutherland's approximation can be used, but for higher temperatures or complex mixtures of species the user can request the curve-fits from the NASA Glenn thermodynamic database [9], which includes a comprehensive database of individual species. For the thermal equation of state we generally assume perfect gas behaviour, meaning that the internal energy is function of temperature only. Under this assumption, the pressure p can be computed using the temperature, total density, and the specific gas constant R.

$$p = \rho R T \tag{7}$$

The last piece of the formulation is an expression for the total energy E, which consists of a kinetic energy component from the gas velocity and an internal energy e that is a function of temperature.

$$\rho E = \frac{1}{2}\rho\left(u^2 + v^2 + w^2\right) + \rho\, e(T) \tag{8}$$

The relationship between internal energy e and temperature can also be specified in different ways, including an ideal gas assumption that is fast but valid only for flow below $< 500K$, and a thermally perfect model with Glenn database curve-fits [9] that are usually good up to $\approx 20,000K$.

2.2. Optional Extras for Hypersonic Multi-physics

Hypersonic flows exhibit a wide range of complicated physical phenomena. High energies in the gas state can cause chemical reactions or nonequilibrium thermodynamics, both of which commonly progress on similar timescales to the flow itself and lead to time-dependant thermal behaviour that is difficult to model analytically. Another complication is turbulence: Three-dimensional, unsteady, fine scale fluctuations in the fluid properties that occur unavoidably at a high enough Reynolds number. These effects require modifying the fluid-mechanic formulation above into a coupled or "multi-physics" approach, with extra equations added to the set of hyperbolic PDEs. In this section we present the three most common extra physics models, starting with those needed for simulating flows with chemical reactions.

Both exothermic reactions like combustion and endothermic reactions like air dissociation are commonly simulated with Eilmer, which requires the introduction of an extra transport equation for the partial density of each chemical species ρ_s, where the index s runs across every species in a given reaction mechanism.

$$\mathbf{U} = \begin{bmatrix} \vdots \\ \rho_1 \\ \rho_2 \\ \vdots \\ \rho_s \end{bmatrix} \tag{9}$$

Each species is convected with the flow, diffused via concentration gradients, and created or destroyed by chemical reactions. These effects appear in the inviscid, viscous, and source vectors respectively.

$$\mathbf{F}_i = \begin{bmatrix} \vdots \\ \rho_1 u \\ \rho_2 u \\ \vdots \\ \rho_s u \end{bmatrix}\hat{i} + \begin{bmatrix} \vdots \\ \rho_1 v \\ \rho_2 v \\ \vdots \\ \rho_s v \end{bmatrix}\hat{j} + \begin{bmatrix} \vdots \\ \rho_1 w \\ \rho_2 w \\ \vdots \\ \rho_s w \end{bmatrix}\hat{k} \tag{10}$$

$$\mathbf{F}_v = \begin{bmatrix} \vdots \\ \rho D_1 \frac{\partial Y_1}{\partial x} \\ \rho D_2 \frac{\partial Y_2}{\partial x} \\ \vdots \\ \rho D_s \frac{\partial Y_s}{\partial x} \end{bmatrix} \hat{i} + \begin{bmatrix} \vdots \\ \rho D_1 \frac{\partial Y_1}{\partial y} \\ \rho D_2 \frac{\partial Y_2}{\partial y} \\ \vdots \\ \rho D_s \frac{\partial Y_s}{\partial y} \end{bmatrix} \hat{j} + \begin{bmatrix} \vdots \\ \rho D_1 \frac{\partial Y_1}{\partial z} \\ \rho D_2 \frac{\partial Y_2}{\partial z} \\ \vdots \\ \rho D_s \frac{\partial Y_s}{\partial z} \end{bmatrix} \hat{k} \quad (11)$$

In these expressions, Y_s and D_s are the mass fraction and mixture-averaged diffusion coefficient of species s. The diffusion of the species also introduces a new term into the energy transport equation, to account for the latent heat of formation tied up in the specific enthalpy of each species h_s. This alters the energy equation's entry in the viscous flux vector:

$$F_v^{\rho E} =$$
$$(\tau_{xx}u + \tau_{yx}v + \tau_{zx}w + q_x - \sum_s \rho h_s D_s \frac{\partial Y_s}{\partial x}) \hat{i}$$
$$+(\tau_{xy}u + \tau_{yy}v + \tau_{zy}w + q_y - \sum_s \rho h_s D_s \frac{\partial Y_s}{\partial y}) \hat{j} \quad (12)$$
$$+(\tau_{xz}u + \tau_{yz}v + \tau_{zz}w + q_z - \sum_s \rho h_s D_s \frac{\partial Y_s}{\partial z}) \hat{k}$$

The source term $\mathbf{Q}$ is usually zero for the fluid equations alone, although Eilmer does contain an axisymmetric mode that includes geometric source terms that account for the complications of a polar co-ordinate system. But the source terms are also used heavily in almost all of the multiphysics models. For chemical reactions they consist of the net mass density production rates for each species.

$$\mathbf{Q} = \begin{bmatrix} \vdots \\ M_1 \dot{\omega}_1 \\ M_2 \dot{\omega}_2 \\ \vdots \\ M_s \dot{\omega}_s \end{bmatrix} \quad (13)$$

These rates are assembled by summing over the reactions in a reaction mechanism r.

$$\dot{\omega}_s = \sum_r -\nu_{rs} \left[k_r^f \Pi_n [X_n]^{\nu_{rn}^f} - k_r^b \Pi_n [X_n]^{\nu_{rn}^b} \right] \quad (14)$$

where ν_{rs}^f and ν_{rs}^b are the forward and backward stoichiometric coefficients for the species s in reaction r, ν_{rs} is the net stoichiometric coefficient $\nu_{rs} = \nu_{rs}^f - \nu_{rs}^b$, $[X_n]$ is the molar concentration of species n, and k_r^f and

k_r^b are the forward and backward reaction tick rates for reaction r. (This nomenclature is similar to that presented by Anderson [10], which should be consulted for further detail).

Usually k^f is specified by an Arrhenius equation with coefficients A, n, and C curve-fitted from reaction rate data, each reaction in the set r having different coefficients.

$$k^f = AT^n e^{-C/T} \quad (15)$$

The backward rates k^b can also be specified in the same way, or they can be computed from the equilibrium coefficient for the given reaction K_{eq}

$$k^b = \frac{k^f}{K_{eq}} \quad ; \quad K_{eq} = \left(\frac{p^\circ}{R_u T} \right)^{\sum_s \nu_s} \times exp\left(\frac{\sum_s \nu_s g_s^\circ(T) M_s}{R_u T} \right) \quad (16)$$

where $g^\circ = g - T s^\circ$ is the Gibbs free energy per unit mass at standard pressure p°, and M_s is the molecular mass of species s. The final change for chemically reacting flows involves the energy equation and equation of state, which are assembled out of the partial densities, the species internal energies e_s, and the standard-state heats of formation $h_s^f = h_s(T = 298.15)$, with 298.15 Kelvin chosen as the standard temperature.

$$p = \sum_s \rho_s R_s T \quad (17)$$

$$\rho E = \frac{1}{2}\rho \left(u^2 + v^2 + w^2 \right) + \sum_s \rho_s e_s(T) + \sum_s \rho_s h_s^f \quad (18)$$

Another common phenomenon present in hypersonic flows is turbulence. Turbulence is the result of cascading instabilities in the flow that break down large disturbances in the flow into smaller ones, producing an unsteady fractal of swirling motion with a wide range of scales that is difficult to resolve on a finite-volume grid. A straightforward and reasonably successful modelling strategy to avoid this problem is Reynolds-Averaging, where the turbulent structures are time-averaged out of the simulation and accounted for with modified transport properties such as the turbulent viscosity μ_t and turbulent thermal conductivity k_t. Eilmer is equipped for Reynolds-Averaged Navier-Stokes (RANS) style turbulence modelling using various models that have been well-validated by the aerospace community, any of which add a number of extra transport equations to the formulation. As an example, the flux vectors for the one-equation Spalart-Allmaras model [11] are shown

below, which adds a conserved quantity for the turbulent viscosity $\rho\hat{v}$, an inviscid flux, a viscous flux, and a complicated source term. The two equation $k - \omega$ model [12] and the high-fidelity Improved Delayed Detached Eddy Simulation [13] technique are also available.

$$\mathbf{U} = \begin{bmatrix} \vdots \\ \rho\hat{v} \end{bmatrix} \qquad (19)$$

$$\mathbf{F}_i = \begin{bmatrix} \vdots \\ \rho\hat{v}u \end{bmatrix}\hat{i} + \begin{bmatrix} \vdots \\ \rho\hat{v}v \end{bmatrix}\hat{j} + \begin{bmatrix} \vdots \\ \rho\hat{v}w \end{bmatrix}\hat{k} \qquad (20)$$

$$\mathbf{F}_v = \begin{bmatrix} \vdots \\ \frac{\rho(v+\hat{v})}{\sigma}\frac{\partial\hat{v}}{\partial x} \end{bmatrix}\hat{i} + \begin{bmatrix} \vdots \\ \frac{\rho(v+\hat{v})}{\sigma}\frac{\partial\hat{v}}{\partial y} \end{bmatrix}\hat{j} + \begin{bmatrix} \vdots \\ \frac{\rho(v+\hat{v})}{\sigma}\frac{\partial\hat{v}}{\partial z} \end{bmatrix}\hat{k} \qquad (21)$$

$$\mathbf{Q} = \begin{bmatrix} \vdots \\ \rho c_{b1}(1-f_{t2})\hat{S}\hat{v} - \rho\left[c_{w1}f_w - \frac{c_{b1}}{\kappa^2}f_{t2}\right]\left(\frac{\hat{v}}{d}\right)^2 + \\ \frac{1}{\sigma}\frac{\partial}{\partial x_j}\left(\rho(v+\hat{v})\frac{\partial\hat{v}}{\partial x_j}\right) + \frac{c_{b2}}{\sigma}\rho\frac{\partial\hat{v}}{\partial x_i}\frac{\partial\hat{v}}{\partial x_i} \end{bmatrix} \qquad (22)$$

These equations introduce the kinematic viscosity v, and allow for the solution of the Spalart-Allmaras field variable $\hat{v}$ throughout the flow, using a large number of additional constants and functions that are defined in Allmaras et al. [11]. The actual turbulent viscosity μ_t is then computed as:

$$\mu_t = \overline{\rho}\hat{v}\frac{(\hat{v}/v)^3}{(\hat{v}/v)^3 + c_{v1}^3} \qquad (23)$$

and the turbulent conductivity κ_t and diffusivity D_t from the turbulent Prandtl number Pr_t and turbulent Schmidt number Sc_t.

$$\kappa_t = \frac{c_p\mu_t}{\mathrm{Pr}_t} \quad D_t = \frac{\mu_t}{\rho\mathrm{Sc}_t} \qquad (24)$$

These transport properties are then added to their laminar equivalents, and the sum is used in place of μ, κ, and D_s in the preceding expressions.

The last commonly encountered multi-physics model of interest is thermal nonequilibrium. This phenomenon occurs during rapid expansions or compressions of the flow, such as behind a shockwave, and affects the statistical distribution of the molecules making up the gas. At the molecular scale, the gas particles are rapidly churning through different internal states of translational velocity as they drift and collide, but the average properties of the entire collection stays more-or-less constant. This average state is well-described by a statistical function called the Boltzmann distribution, which is parameterised by the temperature. A sudden compression can instantaneously change the distribution of translational velocities, but the energy of the gas is also stored in other modes, such as rotation or vibration of the molecules, which may take a finite amount of time to catch up as energy is transferred from the translational mode via collisions. During this time the thermodynamic state of the gas is in a time-dependent state referred to as thermal-nonequilibrium, and the governing equations need to be modified to account for the different relationship between pressure, energy, and temperature.

The starting point for this modification is to include an extra transport equation for each additional energy mode beyond the first. These equations track the convection and diffusion of energy in each specific mode, and have source terms describing the energy exchange with the other modes via collisions. For the purposes of illustration, this section describes the commonly applied Two-Temperature model, in which the translational and rotational energies of all the species are combined into one thermal mode, and the vibration and electronic energies of all the species are combined into a second mode. The vibrational/electronic energy e_v is then governed by the following equations.

$$\mathbf{U} = \begin{bmatrix} \vdots \\ \rho e_v \end{bmatrix} \qquad (25)$$

$$\mathbf{F}_i = \begin{bmatrix} \vdots \\ \rho e_v u \end{bmatrix}\hat{i} + \begin{bmatrix} \vdots \\ \rho e_v v \end{bmatrix}\hat{j} + \begin{bmatrix} \vdots \\ \rho e_v w \end{bmatrix}\hat{k} \qquad (26)$$

$$\begin{aligned} \mathbf{F}_v = &\begin{bmatrix} \vdots \\ \kappa_v\frac{\partial T_v}{\partial x} - \sum_s\rho h_{vs}D_s\frac{\partial Y_s}{\partial x} \end{bmatrix}\hat{i} \\ +&\begin{bmatrix} \vdots \\ \kappa_v\frac{\partial T_v}{\partial y} - \sum_s\rho h_{vs}D_s\frac{\partial Y_s}{\partial y} \end{bmatrix}\hat{j} \\ +&\begin{bmatrix} \vdots \\ \kappa_v\frac{\partial T_v}{\partial z} - \sum_s\rho h_{vs}D_s\frac{\partial Y_s}{\partial z} \end{bmatrix}\hat{k} \end{aligned} \qquad (27)$$

$$\mathbf{Q} = \begin{bmatrix} \vdots \\ \sum_s Q^{vib-trans} + \sum_s Q^{elec-trans} + \sum_r Q_r^{vib-chem} \end{bmatrix} \qquad (28)$$

With the vibrational/electronic energies of each species lumped together, the source term in this case consists of a simple sum over all the energy exchange mechanisms. Exchanges between the translational energy and the vibrational modes are typically modelled

using the Landau-Teller relaxation formula, with a relaxation time for each species τ_s computed using the semi-empirical correlations of Millikan and White [14] or, optionally, by custom-fitted constants.

$$Q_s^{vib-trans} = \rho \frac{e_v - e_v^*(T)}{\tau_s} \qquad (29)$$

At higher velocities such as those experienced by reentering spacecraft, the flow may be ionised by the large amounts of energy present in the flow. The electrons freed by this process are sometimes assumed to have their own energy mode, or they may be lumped in with the vibrational/electronic mode as above. Energy exchange from collisions between the free-electrons translational energy and the heavy-particle translational energy can be modelled using an expression from Gnoffo et al. [15], which includes curve-fitted data for the electron/heavy collision frequencies ν_{es}.

$$Q_s^{elec-trans} = 2\rho_{e-} \frac{3}{2} R_u (T - T_v) \sum_s \frac{\nu_{es}}{M_s} \qquad (30)$$

where ρ_{e-} is the partial density of free electrons. The final term in expression 28 concerns energy exchanged between modes by chemical reactions. We have implemented a model based on the formulation in Knab et al. [16] to account for this effect, which can be written as follows by summing over the molecular dissociation reactions r,

$$Q_r^{vib-chem} =$$
$$k_r^f \Pi_n [X_n]^{v_{rn}^f} \times \left[\frac{R_u \Theta_s}{exp(\Theta_v / T^*) - 1} - \frac{D}{exp(D/R_u / T^*) - 1} \right]$$
$$-k_r^b \Pi_n [X_n]^{v_{rn}^b} \frac{1}{2} [D - \Theta_v R_u]$$
$$\qquad (31)$$

where Θ_v is the characteristic vibrational temperature of the relevant molecule, D is the dissociation energy liberated by the reaction, and k_r^f and k_r^b are the forward and backward chemical reaction rates, as defined in equation 14.

2.3. Numerical Discretisation

Together the equations presented in sections 2.1 and 2.2 form a coupled set of partial differential equations in three dimensions, which are not analytically solvable except for a few special cases. However, by replacing with the differential terms with their discrete equivalents, it is possible to produce powerful and nearly exact numerical solutions of the equations on a digital computer. The first step in this process is to recast the continuous integral form of the conservation equations into a form based on small but finite-sized polyhedra called cells.

$$\frac{\partial \mathbf{U}}{\partial t} = -\frac{1}{V} \sum_f^{faces} (\mathbf{F}_i - \mathbf{F}_v)_f \cdot \hat{n}_f A_f + \mathbf{Q} \qquad (32)$$

This form replaces the boundary integral with a sum over the faces of the cell f, and the fluxes with the equivalent averages over the face in question. Likewise the conserved quantities $\mathbf{U}$ and the source terms $\mathbf{Q}$ are the volume averages of the state inside the cell. The flow field thus consist of a collection of small cells, each containing numbers for the pressure, temperature, density, flow velocity, and any other quantity of interest within. These numbers are then advanced in time through a sequence of discrete steps by evaluating equation 32 over and over again, each step broken up into the following substeps.

1. Reconstruct the flow on both sides of each face
2. Compute the inviscid fluxes $\mathbf{F}_i$
3. Compute the gradients of the flow at each face
4. Compute the viscous fluxes $\mathbf{F}_v$
5. Compute the source term $\mathbf{Q}$ in each cell
6. Compute the residual $\mathbf{R} = \partial \mathbf{U}/\partial t$
7. Advance the conserved quantities in time

Eilmer performs the reconstruction substep differently depending on whether it is operating in structured or unstructured mode. When using a structured grid the cells are arranged into rectangular arrays of hexahedral elements, and each face is aware of its two adjacent cells on each side, labelled L_1, L_0, R_0, R_1 in figure 1. This allows the reconstruction algorithm to perform a pair of quadratic interpolations of each variable to the face, one on either side.

Figure 1: Reconstruction stencil for the structured grid interpolator.

The reconstruction of a variable q is then computed as follows, accounting for the possibly nonuniform spacing of the cells h,

$$q_L = q_{L_0} + \alpha_L \left[\Delta_{L+}(2h_{L_0} + h_{L_1}) + \Delta_{L-}h_{R_0} \right] s_L$$
$$q_R = q_{R_0} - \alpha_R \left[\Delta_{R+}h_{L_0} + \Delta_{R-}(2h_{R_0} + h_{R_1}) \right] s_R$$
$$\qquad (33)$$

with the Δ and α terms:

$$\Delta_{L-} = \frac{q_{L_0} - q_{L_1}}{1/2(h_{L_0} + h_{L_1})}, \quad \Delta_{R+} = \frac{q_{R_1} - q_{R_0}}{1/2(h_{R_0} + h_{R_1})}$$

$$\Delta_{L+} = \Delta_{R-} = \frac{q_{R_0} - q_{L_0}}{1/2(h_{R_0} + h_{L_0})} \quad (34)$$

$$\alpha_L = \frac{h_{L_0}/2}{h_{L_1} + 2h_{L_0} + h_{R_0}}, \quad \alpha_R = \frac{h_{R_0}/2}{h_{L_0} + 2h_{R_0} + h_{R_1}}$$

where s_L and s_R are slope limiters using the formulation of van Albada et al. [17], which enforces the conservation of total variation and includes a small constant $\epsilon = 1 \times 10^{-12}$ to prevent division by zero.

$$s_L = \frac{\Delta_{L-}\Delta_{L+} + |\Delta_{L-}\Delta_{L+}| + \epsilon}{\Delta_{L-}^2 + \Delta_{L+}^2 + \epsilon}$$

$$s_R = \frac{\Delta_{R-}\Delta_{R+} + |\Delta_{R-}\Delta_{R+}| + \epsilon}{\Delta_{R-}^2 + \Delta_{R+}^2 + \epsilon} \quad (35)$$

In unstructured mode (which may also use arrays of hexahedra but is designed to be agnostic as to the grid type), a cell-centred least-squares gradient calculation is performed at each cell (see equations 41-44), and the reconstruction is completed with linear extrapolation using the computed gradients. Figure 2 shows the cloud of cells used for the gradient calculation in each cell using dotted lines, and the two displacement vectors from the cell centers to the face mid-point $\vec{p}_L$ and $\vec{p}_R$.

Figure 2: Reconstruction stencil for the unstructured grid interpolator.

The left and right side face values are then reconstructed as follows.

$$q_L = q_{L_0} + s_L \times \nabla q_L \cdot \vec{p}_L$$

$$q_R = q_{R_0} + s_R \times \nabla q_R \cdot \vec{p}_R \quad (36)$$

Once again the s terms are slope limiters, by default the formulation of VenkataKrishnan [18], which considers the set of all the cells in the local cloud used to generate the gradients C and first computes the minimum and maximum values of the variable.

$$q_{max} = max(q \in C) \quad ; \quad q_{min} = min(q \in C) \quad (37)$$

The limiting function is based on a comparison between two differences in the variable q, the first being that predicted by the gradient extrapolated to the face $\Delta_{grad} = \nabla q \cdot \vec{p}$, and the second using either q_{max} or q_{min} for the whole cloud.

$$\Delta_{diff} = \begin{cases} q_{max} - q : \Delta_{grad} > 0 \\ q_{min} - q : \Delta_{grad} < 0 \end{cases} \quad (38)$$

The slope limiter is then computed for each face, and the final value is the minimum of the whole set of faces surrounding the cell.

$$\phi = \frac{(\Delta_{diff})^2 + 2\Delta_{diff}\Delta_{grad} + \epsilon}{(\Delta_{diff})^2 + 2(\Delta_{grad})^2 + \Delta_{diff}\Delta_{grad} + \epsilon} \quad (39)$$

$$s = min(\phi \in F) \quad (40)$$

Once reconstructed, a variety of options are available for computing the inviscid fluxes through a face, ranging from the very dissipative Equilibrium-Flux-Method [19] and Haenel [20] flux calculators to the less dissipative AUSMDV [21] scheme, as well as various hybrid options that adaptively switch to a dissipative scheme near a detected shock wave. The flux calculators used in Eilmer are mostly Godunov-type schemes, which consider both sides of the reconstructed interface as a kind of Riemann problem and solve approximately for the state in between to obtain the flux. This approach is attractive because it allows shockwaves and other discontinuities to be captured in the solution with no other special handling, while at the same time producing higher order spatial accuracy in smooth regions of the flow. The different schemes all make different assumptions and approximations in their formulation which leads to situational strengths and weaknesses, and users are encouraged to experiment with the available options to find a flux calculator that works well for their problem.

Although the viscous fluxes require less special handling than the inviscid ones, they do require a calculation of the gradients of the primitive variables at each of the cell faces. By default, this task is accomplished using a cell-centred least-squares gradient reconstruction that proceeds as follows.

Consider a first-order extrapolation from the center of a cell to a nearby point, say the centre of one of the faces. If the gradient of a variable q were known, the extrapolation would be computed as:

9

$$q_f = q_c + \nabla q \cdot \vec{dx_f} \qquad (41)$$

where q_c is the value of q at cell center, q_f is the value at the face center, and $\vec{dx_f}$ is the distance vector from the former to the latter. In the CFD solver however, the values of q_f and q_c are already known, specifically, q_f is computed as the average of the left and right state reconstructions on each side of the face. This means that the extrapolation will produce a value of q_f with some amount of error, which can be written as:

$$q_f - (q_c + \nabla q \cdot \vec{dx_f}) \qquad (42)$$

The least-squares gradient calculation proceeds by applying this formula to every face in a given cell, and then solving for the gradient that minimises the square of the error summed across the entire cloud of faces. Formulating this problem as a matrix equation, we begin by presenting the extrapolation step as:

$$\begin{bmatrix} w_0 dx_0 & w_0 dy_0 & w_0 dz_0 \\ w_1 dx_1 & w_1 dy_1 & w_1 dz_1 \\ w_2 dx_2 & w_2 dy_2 & w_2 dz_2 \\ w_3 dx_3 & w_3 dy_3 & w_3 dz_3 \\ w_4 dx_4 & w_4 dy_4 & w_4 dz_4 \\ w_5 dx_5 & w_5 dy_5 & w_5 dz_5 \end{bmatrix} \begin{bmatrix} \nabla q_x \\ \nabla q_y \\ \nabla q_z \end{bmatrix} = \begin{bmatrix} w_0(q_0 - q_c) \\ w_1(q_1 - q_c) \\ w_2(q_2 - q_c) \\ w_3(q_3 - q_c) \\ w_4(q_4 - q_c) \\ w_5(q_5 - q_c) \end{bmatrix} \qquad (43)$$

$$\mathbf{X} \cdot \nabla q = \mathbf{dq}$$

where we have assumed 3D flow with six faces attached to the cell. Eilmer also assigns different weights w to each face based on each one's inverse distance to cell center, effectively weighting nearer faces higher in the error minimisation. The gradient that best minimises the right hand side difference vector $\mathbf{dq}$ is then:

$$\nabla q = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{dq} \qquad (44)$$

An interesting property of this expression is that the design matrix $\mathbf{X}$ depends only on the cell geometry, which means that the complicated matrix $(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$ need only be computed once for each cell during startup, and then at each step of the calculation the gradients ∇q are evaluated cheaply using the current solutions difference vector. This exact same procedure is used in the unstructured solver to compute the inviscid reconstruction, however a cloud of nearby cells is used, and the weightings w are all set to one.

Once the cell-centred gradients are computed, they are then transferred to the faces by an averaging method developed by Haselbacher and Blazek [22], which considers the average gradient from both sides of the face

$\nabla w_{av} = \frac{1}{2}(\nabla w_L + \nabla w_R)$ and the alignment between the face normal vector $\hat{n}$ and the vector between the two cell centers $\vec{e}$, as shown in figure 3.

Figure 3: Vector definitions for Ref. [22] face gradient calculation.

The face averaged gradients ∇w_f are then computed using the following equation.

$$\nabla w_f = \nabla w_{av} - \frac{(\nabla w_{av} \cdot \hat{e} - (w_R - w_L)/|\vec{e}|)}{\hat{n} \cdot \hat{e}} \hat{n} \qquad (45)$$

This formula corrects for some amount of cell non-uniformity and is critically important for resolving viscous flows on unstructured grids, but it has also proven to be helpful for structured calculations as well. The extra dissipation appears to help stabilize the calculation especially in the boundary layer, where the gradients are often high and the cell aspect ratios large.

Computing the source terms $\mathbf{Q}$ is reasonably straightforward except for the chemical reactions, which are often numerically stiff and require special handling. When running Eilmer in transient (explicit) mode, the chemical state of each cell is decoupled from the main flow solver and advanced using an Ordinary Differential Equation (ODE) solver that integrates the chemical state through a number of subcycles for each timestep. For numerically stiff chemical reactions, such as those encountered in combusting flows, Mott's α-quasi-steady-state integrator [23] is available. For other cases, an adaptive Runge-Kutta-Fehlberg scheme [24] is used for integrating the chemical kinetic ODE, which computes both a fifth-order accurate and sixth-order accurate solution of the ODE and adjusts the substep size to keep the difference between the two within a tolerance. When running in implicit (steady-state) mode, no subcycling is performed and the source terms for the chemistry are calculated directly, relying on the inherent stability of the implicit update scheme to handle the chemical stiffness.

The final substep of the simulation main loop is to compute the rate of the change of the conserved variables (sometimes called the residual $\mathbf{R}$) using equation

10

32 and update the conserved variables using a time-stepping scheme.

$$\frac{\partial \mathbf{U}}{\partial t} = \mathbf{R} \qquad (46)$$

Eilmer has a number of options for these schemes, including a family of explicit integrators based on the the Runge-Kutta approach, where the residual is computed multiple times over a single step and the results blended together to achieve higher-order time accuracy. For example, a two-stage predictor corrector explicit scheme computes the change in conserved variable from timestep $n-1$ to n as follows.

$$\mathbf{U}^{n+1}_{pred} = \mathbf{U}^n + \mathbf{R}(\mathbf{U}^n)dt$$

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \left(\frac{1}{2}\mathbf{R}(\mathbf{U}^n) + \frac{1}{2}\mathbf{R}(\mathbf{U}^{n+1}_{pred}) \right)dt \qquad (47)$$

For some problems, it is possible to gain significant improvements in run time by using implicit time advancement schemes that are stable at large timesteps. Eilmer contains a point-implicit update scheme which is based on a Backward-Euler formulation where the change in conserved quantities $\Delta \mathbf{U}$ is computed using the unknown residual at time $n+1$. This residual may be unknown, but it can be approximated using a first-order linearisation that couples the updates at every point of the flow into a large linear system.

$$\frac{\Delta \mathbf{U}}{\Delta t} = \mathbf{R}^{n+1} = \mathbf{R}^n + \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}}\Delta \mathbf{U}^n \qquad (48)$$

Introducing the identity matrix $\mathbf{I}$, this can be rearranged into a large sparse matrix problem as follows.

$$\left[\frac{1}{\Delta t}\mathbf{I} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}} \right]\Delta \mathbf{U}^n = \mathbf{R}^n \qquad (49)$$

Equation 49 introduces the Jacobian matrix $\partial \mathbf{R}/\partial \mathbf{U}$, a measure of the sensitivity of the flow time derivatives to the conserved variables. To evaluate the Jacobian, Eilmer uses a complex-step finite differencing approach, which proceeds by perturbing each component of $\mathbf{U}$ by a small amount in the imaginary plane and comparing the resulting difference in $\mathbf{R}$.

$$\frac{\partial \mathbf{R}_i}{\partial \mathbf{U}_j} = \frac{Im(R_i(U_j + ih))}{h} \qquad (50)$$

The complex number differentiation is implemented using the D language's templating capability, and has proven to be a significant improvement over using real-valued finite differencing, which often experiences difficulties related to the choice of perturbation size, or analytic differentiation of the code, which would be a major maintenance and development burden.

An exact solution to the matrix problem in equation 49 would be expensive to compute and is not generally necessary, since for steady-state problems we are only interested in reducing $\mathbf{R}$ with each iteration. The point-implicit calculation thus approximates by setting all of the off-diagonal terms in equation 49's block matrix to zero, effectively uncoupling the cells from each other and turning the update problem into an isolated matrix problem for each cell that can be solved one-at-a-time. This produces a time advancement method with first-order accuracy that is stable at large timesteps, allowing the solver to rapidly approach a steady-state solution that would otherwise take an impractically large number of explicit steps. Eilmer also has an experimental solver specifically designed for steady flows, in which equation 49 is solved for $\mathbf{R} = 0$ using a series of pseudo-timesteps by a Jacobian-Free Newton-Krylov method.

3. Development Processes

Software is as important to modern scientific research as telescopes and test tubes.

Wilson et al. (2014) [25]

The earliest Eilmer code can be traced back to the early 1990's, when the era of modern CFD was inaugurated by the invention of parallel hardware and advances in applied mathematics like high-resolution flux calculators. In the years since, the code has evolved through multiple languages, rewrites, and paradigm shifts, and we have accumulated some lessons learned that may be of interest to readers of this article. This section describes the behind the scenes detail that goes into developing the code; the programming practices, verification and validation exercises, and testing protocols that we use to write new code and ensure it works properly. It follows in the spirit of articles such as Wilson et al. [25] that call for scientific computing to be more responsibly conducted and better scrutinised, noting as they do that the risk of mistakes in our field is high and the consequences potentially severe. We begin with the programming framework and discuss the reasons for our somewhat unusual choice of language and setup, then describe the various forms of quality control and testing the code is subjected to.

3.1. The D Programming Language

Eilmer is written in D: A high-level, multi-paradigm, statically-typed, compiled programming language intended to be a modernised revision of C++ with the same speed and flexibility. Though D maintains a familiar C-like syntax, it features substantially redesigned high-level features such as objects, templates, and automatic memory management, based on lessons learned from C++ and other popular languages. Specific features of D that have helped us include string mixins for automatically generated inline code, improved templating syntax with much clearer error messages when things go wrong, a module-based architecture that removes the need for header files, and improved object-orientated syntax with property functions that eliminates the need for verbose getters and setters on all of an object's public-facing attributes.

Eilmer itself was written in C++ during the 2006-2015 era of the code's lifetime, but the 2016-present implementation has greatly benefited from D's many improvements in terms of overall line count, complexity, and programmer-friendliness. Our project in particular is maintained part-time by a small team of researchers, and D's helpful error messages and reduced boilerplate has allowed us to make much better use of our limited development time. The code is also used by many students and faculty who are not professional programmers, and the shallower learning curve compared to C++ ensures they are able to make genuine contributions in their teaching assignments and thesis projects.

Figure 4 shows the present division of D and other languages present in the repository as a waffle plot, where the area or number of squares corresponds to the proportion of each language.

Figure 4: Division of languages within Eilmer as of April 2022.

Like many programs, Eilmer uses Lua as an internal scripting language for configuration and run-time customisation. This is facilitated by D's C library bindings, which are used to pass data in and out of a Lua interpreter that is co-compiled and linked into the main executable. The source code for this interpreter is a small amount of lightweight C code that is carried around in our repository, part of a general philosophy that tries to minimise external dependencies that the user has to obtain before compiling the code. This philosophy is a lesson learned from the previous incarnation of the code, which suffered from a degree of churn and rot in the many open-source libraries and toolkits it relied on. The current implementation requires only on a handful of standard Linux development libraries, a D compiler, and OpenMPI for parallelisation, with a few optional extra features available if the foreign-function libraries for Python and Ruby are installed. In this manner it is compatible with a wide range of High-Performance Computing (HPC) systems and should remain so even as hardware and software trends shift in the future.

3.2. Verification and Quality Assurance

Modern scientific computing distinguishes two ways of vetting that software is fit for purpose. Verification (*"are we solving the equations right"*) involves checking that the program is correctly achieving the intentions of its programmers. An incorrectly spelled variable, an unintended logical comparison, or a mistake in a mathematical formula are all examples of problems that are in the realm of verification. The second activity, Validation (*"are we solving the right equations"*) is the subject of the section 3.3.

The first technique applied to the verification of Eilmer will be familiar to software engineers: unit tests. A core principle of good software design is to break a program up into encapsulated modules, each with a single clearly defined responsibility. To ensure that the smallest subcomponents of the code are performing correctly, each can be tested in isolation by being given a small test problem and comparing the output to a known good value.

Extensive unit tests are available for Eilmer's subcomponents, including the modules for handling gas behaviour, chemical kinetics, geometry, and numerical methods/linear algebra. For example, the gas tests can be invoked using:

```
$ cd src/gas
$ make test
```

This will compile a small executable for each of the relevant gas models and use them to compute a set of sample problems, such as the internal energy of a mixture given a pressure, temperature, and composition. The output of each calculation is then compared to a known good value, producing a summary of the outcomes and a success message if all is well.

Another technique of verification used within Eilmer is to compare an entire simulation against an exact solution computed using an analytic technique. Although the equations of fluid mechanics are, in general, too complex to be solved by these techniques, for some special cases exact analytical solutions are available. One example of this procedure applied to Eilmer is shown in figure 5, a solution of the detonation wave problem proposed by Powers and Aslam [26] in a paper calling for more widespread and rigorous verification of numerical codes.

The problem consists of a curved ramp that drives an oblique detonation wave through the oncoming flow, sustained by a simplified one-step chemical reaction that activates once a temperature threshold has been reached. The reaction rates and the curved ramp surface are designed to ensure that the angle of wave is fixed at 45°, and exact solutions are available for postshock flow conditions along streamlines through the shocked gas. Figure 5 shows the solver's computed solution, with a regression line fitted to the cells where the shock is detected. Even on a relatively coarse grid this agrees well with the analytic solution, and further refinements show good results for the postshock flowfield as well, evidence that the reaction rates and the chemistry/flow solver coupling has been implemented correctly.

Computational physics calculations can also be verified using the Method of Manufactured Solutions. This technique takes advantage of an interesting feature of the partial differential equations used in continuum mechanics codes, which typically have the following form.

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}(\mathbf{U})}{\partial x} + \frac{\partial}{\partial x}\frac{\partial \mathbf{V}(\mathbf{U})}{\partial x} = 0 \qquad (51)$$

In general, any given $\mathbf{U}$ will not be a solution to these equations, and putting it through the numerical machinery for calculating the derivatives would give a nonzero remainder on the right hand side. The essential insight of the Method of Manufactured solutions is to add an artificial source term to the equations that precisely matches this remainder.

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}(\mathbf{U})}{\partial x} + \frac{\partial}{\partial x}\frac{\partial \mathbf{V}(\mathbf{U})}{\partial x} = \frac{\partial \mathbf{F}(\mathbf{U}_{mms})}{\partial x} + \frac{\partial}{\partial x}\frac{\partial \mathbf{V}(\mathbf{U}_{mms})}{\partial x} \qquad (52)$$

By choosing an arbitrary solution $\mathbf{U}$ that is analytically differentiable, the source term on the right hand side can be evaluated exactly. One can then run the numerical solver with this manufactured source term in every cell, which acts to drive the numerical solution toward $\mathbf{U}_{mms}$, assuming that the numerical machinery that approximates the spatial derivatives has been implemented correctly. The discrepancy between the final value arrived at by the solver and the manufactured solution is then an estimate of the discretisation error in the numerical algorithm, which should be small if all is well.

Eilmer has several manufactured solution test cases included in its examples directory and others that are published in various places, which provide verification coverage across both structured [27], and unstructured [28] grids, as well as solid/fluid conjugate heat transfer flows[29]. A new example of a manufactured solution, this one for a viscous flow with the Spalart-Allmaras turbulence model, is shown in figure 6. (The manufactured solution itself is documented in Appendix 1.)

Merely converging the the manufactured solution is a necessary but not sufficient condition for verification. A key characteristic of the numerical methods used in high-resolution CFD solvers is that they are higher order: the discretisation error should not merely reduce as the grid is refined, but the reduction will be nonlinear, generally second order $O(dx^2)$. This property can be checked by computing the total error of multiple MMS cases with different levels of refinement, and plotting the results on a logarithmic scale as in figure 7. The test shows the expected nonlinear drop, with the slope of each line indicated by an annotation for each variable.

Figure 5: Oblique detonation wave, mass fraction of species B.

Figure 6: Colour maps of numerical solution to MMS test case. Exact solution shown in white contour lines.

3.3. Validation and Reality Checks

Verification and Validation are the oft-confused twin pillars of responsible scientific computing. The previous section has dealt with Verification: Are we solving the equations correctly? This subsection takes up Validation: Are we solving the correct equations?

Validation involves checking a program's correspondence with the aspects of reality it is intended to simulate. Comparing numerical predictions against experimental data is the gold-standard activity, but other valuable work includes checking the validity of assumptions in the formulation and establishing boundaries where missing physics models are required. In contrast to Verification, Validation is not so much the activity of a single person but a shared responsibility of the entire research community, as it relates to the shared pool of physical models we have developed and our knowledge of their limitations.

In this regard Eilmer benefits from a long history of collaboration with experimental hypersonics. The ancestry of the code can be traced back to cns4u, a single block Navier-Stokes integrator written by Peter Jacobs in the early 90's for simulating hypersonic wind tunnels. Testing against measurements taken in an expansion tube revealed that the program could predict the large scale wave processes reasonably well, as measured

Figure 7: Error norms and orders of accuracy from 3D MMS cases, with 16^3, 32^3, and 64^3 cells

by pressure sensors on the inside of the tunnel walls and a rake of pitot-pressure sensors in the freestream [30]. Simulations of wind tunnel operation continue today using the modern incarnation of the code, for example in Jewell et al. [31] who performed transient 2D calculations of the AFRL Mach 6 Ludwieg tube, and Gildfind et al. [32] simulations of the large X3 expansion tube. Both authors report good agreement with pressure measurements taken inside the respective facilities, though the expansion tunnel behaviour is complex and not perfectly captured for various reasons of approximation.

A common use for hypersonic wind tunnels is to generate experimental data solely for the purpose of validating CFD. Basic shapes such as ramps, cavities, and spheres are typically used for this purpose, outfitted with pressure and heat transfer sensors that measure the state of the flow touching the model. An example is the hollow flared-cylinder model tested by Holden and Wadhams [33] in the CUBRC shock tunnel facilities, part of a large library of experimental results intended to be used for validation studies. The flow over the out-

14

Figure 8: Temperature colour map of hollow cylinder flare solution. Dimensions in mm.

side of the cylinder features a separation and a viscous shock interaction, shown in figure 8, as computed by an Eilmer simulation that is available in the examples directory online. The predicted heat load and pressure on the walls compare quite closely to the experimental measurements (see figure 9), and are in line with the predictions of other hypersonics codes reported in MacLean and Holden [34].

Similar exercises have been reported in Sun et al. [35], in which multiple codes (including Eilmer) are compared on a cone-shock interaction problem; Park et al. [36] who matched pressure and heat transfer results on a high-enthalpy cylinder flow; and Damm [28], who documents four different validation tests including a double cone, shock interaction, and a turbulent boundary layer, all non-reacting. An additional reacting flow exercise can be found in Hoste et al. [37]. In general, these exercises generate reasonably good agreement to pressure measurements, though heat transfer is sometimes harder to match due to the larger margins of error encountered in the measurements. This problem has been underlined by multi-code validation exercises such as Candler [38], where consistent problems matching certain kinds of experimental data have been found, especially in high enthalpy flows. Interestingly, Ray et al. [39] have suggested that the major cause of these problems could be that the inflow conditions given to the CFD solvers are often inaccurate, due to the large number of approximations made in the facility flow models that generate the predictions of what the conditions are. Better numerical simulations of hypersonic facil-

ities (such as those described in an earlier paragraph) are an obvious place to look for answers to this problem.

More direct measurements of a hypersonic flow can be obtained using optical techniques such as Schlieren photography. A handful of experiments (such as Lobb [40] and Nonaka et al. [41]) have used this technique to visualise the shockwave created by a blunt hypersonic object as it flies through a ballistic range, a useful arrangement that avoids the problem of reconstructing the test flow from a wind tunnel. The distance between the object's leading edge and its bow shock (as detected by the Schlieren) is called the shock standoff distance, which can be used for code validation if the precise shape of the shock can be reconstructed from the image using a fitting technique.

A collection of these exercises, performed using Eilmer, is reported in Gollan and Jacobs [42], with more detailed analysis undertaken by Zander et al. [43] for higher enthalpy flows. In general, the code performed well, predicting the shock shape to within the range of experimental error, as long as thermal and chemical nonequilibrium are accounted for. Though these results affirm the presence of nonequilibrium effects caused by high temperatures, the data are currently too noisy to distinguish which of the many reaction schemes and relaxation models should be preferred.

Further insight into questions like this can be gained using finer grained optical diagnostics, such as emission spectroscopy, to interrogate the bright light emitted by a hot hypersonic flow and separate it into fre-

Figure 9: Surface pressure and heat transfer results from hollow cylinder flare solution. Experiments by Ref. [33]

database to produce reasonable predictions of the visible wavelength range, although a number of lines generated by the complex C_2 and CN radiators were missing, with particular trouble again in the ultraviolet.

More interestingly (at least for validation purposes), it is possible to take spectral information measured in a real flow and work backwards to compute other properties such as temperature or composition. One example of this technique was developed by Potter et al. [46], who performed least-squares fitting of an artificial spectrum (generated by the *Photaura* radiation calculator) to a real spectrum generated by a CO_2-N_2 flow over a cylinder in an expansion tube. The fitting produced a temperature and composition that best matched the measured radiation, which could be compared to an Eilmer CFD result. Although the error bars of the resulting temperatures were fairly large ($\approx 10 - 20\%$), the results were good enough to make some judgements about some of the different CFD models present in the literature. This same technique was later used by Banerji et al. [47], this time in an Earth-like atmosphere, to diagnose an abundance of $N2^+$ that was causing overprediction of radiation in the ultraviolet range, and by Gu et al. [48] to show that the existing energy relaxation rates of CO_2 are far too slow in an expanding flow, such as behind the shoulder of a reentry capsule. Even the spectral lines themselves can be used for code validation. Liu et al. [49] measured the width of Hydrogen lines in a simulated gas-giant entry and used the amount of Stark broadening to estimate the concentration of free electrons in the shock layer. These measurements confirmed some suspected problems in the ionisation rate models that are available in the literature, and the authors were able to modify an existing model to produce an Eilmer calculation that showed good agreement with their measurements.

3.4. Continuous Integration Testing

Any quality control process is only as good as the version of the code it makes use of. Eilmer is a large, constantly evolving codebase with many contributions from collaborators, and we have developed a continuous testing program to keep the code in working order as we add and change things. The core of this process is a set of test simulations (fifty-six at the time of this writing) that can be automatically run and post-processed by a set of scripts written in the Ruby scripting language. The simulations are typically coarsened or scaled down versions of the tests discussed in the prior sections, including simple validation exercises, MMS verification tests, flows with analytic solutions, and a handful of

quencies. In theory, one can even take the temperatures and composition computed by a CFD simulation, input them into a radiation program that accounts for all the relevant lines, transitions, and atomic processes, and compare the resulting artificial spectrum to a real one measured in an experiment. In practice, these calculations are fiendishly complicated and often have discrepancies that are difficult to diagnose, given the many steps between the input and output. Studies such as Fahy et al. [44] have used Eilmer and the PARADE radiation database to compute the radiation from the 2010 Hayabusa probe reentry, finding reasonably good agreement with measurements in the near-infrared frequencies (mostly oxygen and nitrogen lines), but poor agreement in the ultraviolet range. Similar work by Banerji et al. [45] considered a Venus-like CO_2-N_2 atmosphere, using Eilmer and the NEQAIR radiation

other building block simulations that exercise major features of the code.

At the end of each test the associated script extracts a handful of diagnostic numbers and compares them to a known good outcome; this may be the number of iterations taken by the solver, or the residuals at the final step, or the drag on a flat plate, or the root-mean-square error between the numerical solution and its analytical counterpart. A too-large difference in any diagnostic is reported as a failed test, which typically indicates some kind of bug or regression in the code from the prior commit.

Building a test suite with good coverage is helpful for quality control reasons, but we have found the biggest advantage is that we can do *continuous* testing, using an automated system that tests every published commit and reports via email if something is wrong. This system was implemented in the last quarter of 2020 and has been monitoring our repository ever since. A record of its activity through early 2022 is shown in figure 10.

Figure 10: Runtime history of Eilmer continuous testing system.

The execution time of the entire suite has dropped from over an hour to about forty-five minutes throughout 2021 because of optimisations and changes to the roster of tests, and the period includes 509 successful runs, 16 issues with the build process, and 40 failures of the automated tests for the entire period. Many of these issues arose from compiler changes, platform specific glitches, and other obscure problems that are hard for an isolated developer to detect on their own, and it is fair to say we were surprised at how helpful the automated testing system turned out to be. Continuous testing can be easily implemented in a Python script and then run as a cron job using a server or virtual cloud machine, and we highly recommend it to the authors of other scientific and research codes.

4. Parallel Scaling

Fluid dynamics problems can require large amounts of computation time. Like most modern scientific codes, Eilmer can mitigate this problem by breaking a simulation up into a number of independent pieces and executing them in parallel, using multi-core CPUs or groups of them joined by a high-speed network. The core parallelisation strategy we use is *block decomposition*, in which the fluid domain to be simulated is divided into a jigsaw puzzle of non-overlapping blocks, either manually by specifying the split points in a structured grid, or automatically using the external graph partitioning tool METIS [50]. Figure 11 is an example of a 3D blunt cone partitioned using the automated procedure.

Figure 11: Block decomposition for a viscous blunt cone mesh using METIS, nblocks=48

Each block consists of cell elements that contain an average state of the fluid variables $\mathbf{U}$ throughout a small region of space. The change in state over a small step in time $\Delta\mathbf{U}$ is computed using the fluxes of each variable $\mathbf{F}$, which depend on both the cell in question and also its neighbour, due to the reconstruction process discussed in section 2.3. This spatial dependency makes parallelising the solver a nontrivial problem at the boundaries where one block meets another.

The main way that this problem is solved is by surrounding each block in a layer of virtual or "ghost" cells that are not part of the actual simulation, and instead are simply filled with flow that matches the conditions in each neighbouring block. The required data is passed back and forth between the different processors working on each block, using standardised MPI (Message Passing Interface) library functions, along with Reduce and Barrier operations to synchronise control flow between the different processors. In this way, each processor marches independently through the same algorithm but working only on its designated region of the flow.

In this section, we present the results of some scaling experiments to benchmark the code's parallel performance and ensure the implementation is scaling efficiently to the large number of cores that are sometimes required for timely fluid dynamics research. Parallel efficiency can be assessed by using a simple model of task scheduling where an algorithm is split into a Serial fraction s and a Parallel fraction p, where $p + s = 1$. The expected time taken T_n for a task running on n processes is:

$$T_n = sT_1 + \frac{p}{n}T_1 \tag{53}$$

The parameters s and p are not actually constants, but they can be estimated for a typical simulation that is representative of the large parallel calculations our users perform on supercomputers. This section presents the results of timing tests on such a case, the $5°$ blunt cone depicted in figure 11, which uses a 2 million cell 3D viscous mesh immersed in a thermally perfect air flow at Mach 3. The exact same simulation is performed multiple times on different numbers of processors, and the average time per iteration and estimated speedup is shown in table 1.

In a perfectly parallel calculation the speedup would exactly follow the number of processors, but the table shows speed-up degrading as more and more cores are added, an unavoidable reality of each one having to duplicate the serial fraction s of the algorithm. The above data can be used to curve-fit values of s and p from equation 53, resulting in a parallel efficiency of $p = 0.99957$ and the estimated performance shown in figure 12.

These tests confirm that the parallelisation is scaling efficiently up to ≈ 1000 processors, *for this particular problem*. Note however that this will not be true in general. Specifically, the efficiency p is a function of the number of cells in the mesh, and a smaller simulation than the 2 million cell test problem described here will be expected to have a lower p value and much faster degradation in the speed-up as the number of cores is increased. Users are encouraged to perform a pair of scaling simulations like in the above example for their own simulations whenever parallel efficiency is important, such as when using a shared cluster machine with metered usage.

Figure 12: Parallel scaling performance with $p = 0.99957$ on blunt cone problem.

5. Example Applications

This section showcases a handful of representative simulations that cover most of the common applications that Eilmer has been used for in the past. They range from pure research in fluid dynamics to applied calculations of heat transfer and pressure in an engineering problem, with aerodynamic shape optimisation and hypersonic facility design somewhere in between. The final example is slightly different — it consists of a simple blunt body flow calculation performed by our students each year in the University of Queensland's computational fluid dynamics course. Taken together, the examples give a broad outline of what Eilmer is capable of, though the range future applications is limited only by the imagination and, of course, by the available compute time.

5.1. Minimal Working Example

The very first simulation that most users of Eilmer will run is the classic sharp-nosed $20°$ cone, a basic test case that consists of a Mach 1.5 freestream and an oblique shockwave in ideal, inviscid, air. Figure 13 shows the entire Lua configuration file required to set up the simulation, including building the grid, setting the gas model and boundary conditions, and configuring the solver.

n Processors	48	96	144	288	576	768	960
time/iter	1.00	0.506	0.343	0.191	0.109	0.076	0.065
speed-up	48.0	95.0	140.1	250.6	439.7	632.16	741.6

Table 1: Results of parallel scaling tests using 3D blunt cone simulation

18

```
-- 1. Flow domain and grids, dimensions in metres.
config.axisymmetric = true
--
a0 = {x=0.0, y=0.0};    a1 = {x=0.0, y=1.0}
b0 = {x=0.2, y=0.0};    b1 = {x=0.2, y=1.0}
c0 = {x=1.0, y=0.29118}; c1 = {x=1.0, y=1.0}
--
quad0 = CoonsPatch:new{p00=a0, p10=b0, p11=b1, p01=a1}
quad1 = AOPatch:new{p00=b0, p10=c0, p11=c1, p01=b1}
--
grid0=StructuredGrid:new{psurface=quad0,niv=11,njv=41}
grid1=StructuredGrid:new{psurface=quad1,niv=31,njv=41}
--
-- 2. Gas model and flow states.  SI units.
setGasModel('ideal-air-gas-model.lua')
initial = FlowState:new{p=5955.0, T=304.0} -- Pa K
inflow = FlowState:new{p=95.84e3, T=1103.0,
                       velx=1000.0} -- Pa K m/s
--
-- 3. Fluid blocks, initial flow, boundary conditions.
blk0=FluidBlock:new{grid=grid0,initialState=inflow}
blk1=FluidBlock:new{grid=grid1,initialState=initial}
identifyBlockConnections()
blk0.bcList['west']=InFlowBC_Supersonic:new{
                                     flowState=inflow}
blk1.bcList['east']=OutFlowBC_Simple:new{}
--
-- 4. Simulation parameters.
config.max_time = 5.0e-3   -- seconds
config.max_step = 3000
config.dt_plot = 1.5e-3
```

Figure 13: Sharp-nosed 20° cone configuration script

Figure 14: Sharp-nosed 20° cone results: Mach number

Before running the simulation, this script is passed to a preprocessor that produces Eilmer-native grid and flow solution files, as well as a verbose machine-readable configuration file (in JSON format) with all of the available solver settings, including default values for any that are not specified in the input script. The actual simulation then proceeds using these preprocessed files and not the user-facing Lua input script, eventually producing the flowfield shown below.

This multi-stage workflow is partially a legacy of the code's early history, when stand-alone grid generation tools did not exist and users had to specify their own flow domain in precise detail, but the process continues to have many benefits. The built-in grid generation is useful for making parametric grids, and the programmable input allows the user to compute interesting and complicated things in their setup process, such as a spatially varying initial flowfield or boundary conditions that depend on an in-situ compressible flow calculation. The verbose configuration files are also useful for documentation and reproducibility purposes, essentially serving as an automatically produced log of the simulation settings used in any research work.

5.2. Free Piston Driver

The earliest calculations done with Eilmer's predecessors involved simulating expansion tunnels, large machines which are used to generate controlled hypersonic flow for laboratory experiments. These facilities typically use a heavy piston to compress a large slug of driver gas, which eventually bursts through a thick steel diaphragm and drives an incident shockwave through the facility, heating and pressurising the test gas that will eventually flow over the model.

Numerical simulations are used extensively to configure the tunnels before an experiment and to interpret the results afterward, and the modern Eilmer code has many capabilities of interest to experimenters, including a moving mesh transient solver which is the subject of this example. Figure 15 shows the results of a simple calculation where a piston is driven at constant velocity through an axisymmetric tube. The motion drives a shockwave through the gas with an analytically calculable velocity that compares well to the numerical results.

19

Figure 15: Free piston driver results: Temperature (K) and X-velocity (m/s)

Figure 16: Vibrational/Electronic temperature flowfield over a scaled Apollo capsule. Capsule geometry from Ref. [52], Figure 1.

5.3. Re-Entry Capsule

High fidelity CFD is also used extensively in the design of the re-entry capsules that return people and equipment from space. These capsules are equipped with a sacrificial ablative heat-shield that absorbs some of the ferocious heating applied to the spacecraft's windward flank, burning away in layers and ejecting the absorbed energy away from the vehicle. Designing these heat shields requires a reasonably precise estimate of the peak and integrated heat transfer rates, since a too-thin layer of ablative material will not protect the vehicle properly, but a too-thick one will take up precious mass that could have been used for the payload. For these reasons, a large amount of research has gone into developing sophisticated engineering models for re-entry flows and providing experimental datasets they can be validated against. The application in this section is one such validation exercise, a subscale Apollo capsule model at conditions matching the second Project FIRE flight test from May 1965, specifically the 1636 second trajectory point at 71 kilometres altitude, where the model was travelling at 11.2 kilometres per second. The flow is modelled using a viscous, two-temperature, 11-species air model based on Gnoffo et al. [15], with reactions and thermal relaxation rates from Park [51]. A 2D axisymmetric structured grid with 100x257 elements is used, with geometric clustering applied to the wall to achieve a first cell height of 2 μm, enough to grid-converge the wall heat transfer.

The biggest modelling uncertainty in the problem comes from the impact of surface chemistry. Figure 17 shows the predicted heat transfer rates for two nearly identical simulations, one with a non-catalytic

wall where the wall-normal species density gradient is set to zero, and a second with a super-catalytic wall where the actual mass fractions are specified to be the same as the freestream. (This is effectively the same as a fully catalytic wall, given the surface temperature at these conditions is set to 810 K). At the stagnation point, Eilmer predicts a wall heat transfer rate of 185.6 W/cm^2 for the non-catalytic simulation and 306.6 W/cm^2 for the catalytic one, which neatly brackets the actual convective heating rate of 264.5W/cm^2, computed by subtracting the measured radiative heating from the total rate.

Figure 17: Wall heat transfer data from simulation of Project FIRE, flight II.

The computed values are also in good agreement with the simulation results presented in Hash et al. [53], a code comparison between NASA Ames's DPLR code,

NASA Langley's LAURA, and the University of Minnesota's US3D, all using very similar physical models to those implemented in Eilmer.

5.4. Aerodynamic Shape Optimisation

A single CFD solution of the flow around an object can predict values of lift, drag and heat transfer that an engineer can use in designing a supersonic aircraft. But one of the most interesting frontier of numerical research involves coupling the solver with a numerical optimisation technique to produce *automated* designs that maximise performance subject to constraints. This subsection is devoted to an example of this workflow using Eilmer, taken from Kyle Damm's PhD thesis [28]. The exercise is a simple proof-of-concept optimisation of a slender axisymmetric body, which is known to have a curved shape that minimises the total wave drag. The body shape is controlled by a 20-point Bézier curve with fixed ends, and the open-source optimisation toolkit DAKOTA [54] is used to shift the Bézier control points around until a minimum drag configuration is found.

Figure 18: Before (blue) and after (red) axisymmetric wedge subjected to optimisation for minimum wave drag.

Figure 18 shows the before and after results of the optimisation, with the initial shape (a right circular cone) shown in blue, the optimised curve shown in red, and the steady flowfield of each state shown as the black-on-white colourmap. The optimiser has induced a subtle curvature to the surface and reduced the wave drag by approximately 7%, using 66 iterations of the optimisation loop. Crucially, the gradient of the objective function with respect to the Bézier control points is evaluated using an adjoint method that is built into the flow solver. Adjoint methods are crucial to CFD-based optimisation, as they allow the objective function gradients to be computed with just one CFD simulation, regardless of how many design variables are present. A more complete explanation of this methodology can be found in Damm et al. [55], which applies the same method to a hypersonic vehicle inlet and presents more detail about the adjoint solver and the simulations involved.

5.5. Double Cone Flow Physics Investigation

Another popular application for flow simulation codes is doing fundamental research in theoretical fluid mechanics. Numerical simulations have a number of advantages over brick-and-mortar wind tunnels, and at least in laminar flow they can usually be relied on to generate exact and often quite complex solutions of the Navier-Stokes equations. An example is the recent paper of Hornung et al. [56], where Eilmer was used to solve the high Mach-number flow over a large number of double-cones with different angles, producing either a stable or unstable shock structure in each case depending on the geometry. An example of the stable structure is shown in figure 19.

Figure 19: Steady flow from Ref. [56], $\theta_1 = 40°$, $\theta_2 = 70°$, condition A.

This figure is a numerical shadowgraph showing the density gradients over a double cone with angles of $\theta_1 = 40°$ and $\theta_2 = 70°$, with a flow Mach number of 7.7 and a composition of pure N_2. The shock structure is macroscopically steady, consisting of a conical oblique shock attached to the tip of the first cone and a complex triple point where it intersects with a supersonic jet and a small separation close to the wall.

Decreasing the angle of the first ramp strengthens the second shock in comparison to the first one, driving the separation point forward and potentially causing it to break free from the wall. At $\theta_1 = 0°$ the geometry would essentially be a blunt body with a detached bow shock, but at intermediate values of θ_1 the shock is neither attached nor detached but instead oscillates violently back

and forth. A single frame of this process is shown in figure 20, taken from a simulation with $\theta_1 = 10°$ and $\theta_2 = 70°$, but otherwise using the same flow conditions as figure 19. The shock structure is in the act of surging forward, producing a large region of chaotic separated flow, and will shortly detach from the tip of the first cone and then reverse direction, eventually crashing back against the wall and beginning a new cycle of unsteadiness. The simulation captures multiple cycles of this unsteadiness using Eilmer's time accurate transient solver, and Hornung et al. [56] use many simulations over a wide parameter sweep to infer a boundary in θ_1 vs. θ_2 space where the instability begins.

Figure 20: Unsteady flow from Ref. [56], $\theta_1 = 10°$, $\theta_2 = 70°$, condition A.

5.6. Eilmer in the Classroom

Throughout its history Eilmer has been used for both research and teaching at a handful of different universities. Many postgraduate students use the code in their research projects, and the codebase is full of small contributions from students who have have needed some cutting-edge simulation capability and managed to implement it themselves. This is one of the major advantages of using open-source software for academic research, and sparing students the fearsome learning curve of C++ was one of the key drivers for our adoption of the D programming language for the code's 2016-era rewrite.

At The University of Queensland, Eilmer is also used in our undergraduate and masters level course in computational fluid dynamics. This last example is a simple validation exercise performed by our final year mechanical and aerospace engineering students each year, in which they must match the experimentally measured shock standoff around a blunt cone immersed in a Mach 4 test flow generated by a small shock tunnel. The students do the experiments, collect high-speed Schlieren video of the models, build grids, run the simulations, and are responsible for extracting the shape of the shockwaves and accounting for the inevitable complications that arise in any real experiment.

Figure 21: Steady flow Schlieren image from 2021 MECH4480 student experiments

Figure 21 is an example of the experimental data, a single Schlieren image of a 15*mm* radius cone with a half angle of 20°, extracted during the approximately one millisecond window of steady test time. A circle with a radius of 53 pixels has been fitted to the body, which can be used to determine the scale of the image and the position of the model. A cubic Bézier curve has been fitted to the shock shape, which suggests an experimental shock standoff distance of $\approx 2.2mm$.

Figure 22 is the same data but with a shock-fitted thermally perfect simulation of the experiment.

Figure 22: Shock fitted simulation temperature field overlayed on Schlieren results

The simulation predicts a shock standoff of $\approx 2.4mm$ and is a reasonably good match to the data close to the stagnation point, though the simulated shape wanders above the measurements further downstream. This is most likely due to the unmodelled divergence in the nozzle's core-flow, which is assumed to be uniform in the

simulation to avoid introducing undue complexity into the problem. In fact, handling this kind of mismatch between experiment and simulation is precisely what makes the exercise a great learning experience for the students.

6. Conclusions

This paper has been written to fulfil multiple purposes. For the broad aerospace research community, it gives a brief overview of the open-source compressible flow simulation codes available, both niche and comprehensive, and explains how Eilmer fits into the landscape. For users and future developers of the code, it lays out the basic mathematical principles and governing equations that are solved to produce the simulation data they rely on. A good understanding of these principles is critical for developers, and even casual users should have some appreciation of the code's fundamentals to avoid wandering into areas where their results may be invalid. For the developers of other scientific codes, we have tried to document the programming principles, testing tools, and Verification and Validation exercises that go on behind-the-scenes of the project. We have found that a small investment in diligent software engineering in the present can avoid large amounts of extra work in the future, and wish to join other advocates of adopting programming best-practice ideas in scientific computing. For users who work with HPC facilities we have included a section on parallelisation, with an example of a scaling exercise that should serve as a blueprint for others to follow. This should be particularly helpful to those applying for scarce supercomputing time, whose gatekeepers are increasingly anxious to know that their allocations are being used efficiently. Finally, for the general user who wants to know what Eilmer is capable of, we have included a handful of interesting example simulations that cover theoretical and applied fluid mechanics research, hypersonic engineering calculations, and teaching exercises.

All of these purposes are really subgoals of the main goal of this paper, which is to share our project with the world. One of the great triumphs of the modern internet is the open-source software movement, which has strived to make computer programs that are *free*: Both in the sense of *free beer* (they do not cost any money), and free as in *freedom* (the source code is available for anyone to download, inspect, and modify). This philosophy overlaps with the broader mission of scientific research, which aims to discover information about the world that can be distributed to the benefit of everyone, and the proliferation of open-source research codes is

the result of a harmonious match between the two ideas. By publishing both our code and this paper, we hope to join in this quiet revolution, and by helping others advance the march of scientific progress, to play a small part in a brighter future.

References

[1] B. Kleb, W. Wood, CFD: A castle in the sand?, in: 34th AIAA Fluid Dynamics Conference and Exhibit, AIAA-2004-2627, 2004.

[2] M. Bernardini, D. Modesti, F. Salvadore, S. Pirozzoli, STREAmS: A high-fidelity accelerated solver for direct numerical simulation of compressible turbulent flows, Computer Physics Communications 263 (2021) 107906.

[3] J. Romero, J. Crabill, J. Watkins, F. Witherden, A. Jameson, ZEFR: A GPU-accelerated high-order solver for compressible viscous flows using the flux reconstruction method, Computer Physics Communications 250 (2020) 107169.

[4] D. Lusher, S. Jammy, N. Sandham, OpenSBLI: Automated code-generation for heterogeneous computing architectures applied to compressible fluid dynamics on structured grids, Computer Physics Communications (2021) 108063.

[5] M. Di Renzo, L. Fu, J. Urzay, HTR solver: An open-source exascale-oriented task-based multi-GPU high-order code for hypersonic aerothermodynamics, Computer Physics Communications 255 (2020) 107262.

[6] V. Casseau, D. E. Espinoza, T. J. Scanlon, R. E. Brown, A two-temperature open-source CFD model for hypersonic reacting flows, part two: multi-dimensional analysis, Aerospace 3 (2016) 45.

[7] W. T. Maier, J. T. Needels, C. Garbacz, F. Morgado, J. J. Alonso, M. Fossati, SU2-NEMO: An open-source framework for high-mach nonequilibrium multi-species flows, Aerospace 8 (2021) 193.

[8] T. D. Economon, F. Palacios, S. Copeland, T. W. Lucaczyk, J. J. Alonso, SU2: An open-source suite for multiphysics simulation and design, AIAA Journal 54 (2016) 828–846.

[9] B. J. McBride, M. J. Zehe, S. Gordon, NASA Glenn Coefficients for Calculating Thermodynamic Properties of Individual Species, Technical Report 211556, National Aeronautics and Space Administration, 2002.

[10] J. D. Anderson, Jr., Hypersonic and High-Temperature Gas Dynamics, AIAA, 2000.

[11] S. R. Allmaras, F. T. Johnson, P. Spalart, Modifications and clarifications for the implementation of the Spalart-Allmaras turbulence model, in: Seventh International Conference on Computational Fluid Dynamics, Big Island, Hawaii, 2012.

[12] D. C. Wilcox, Turbulence Modelling for CFD, second ed., DCW Industries, Inc., 2002.

[13] M. Shur, P. Spalart, M. Strelets, A. Travin, A hybrid RANS-LES approach with delayed-DES and wall modelled LES capabilities, International Journal of Heat and Fluid Flow 29 (2008) 1638–1649.

[14] R. C. Millikan, D. R. White, Systematics of vibrational relaxation, Journal of Chemical Physics 39 (1963) 3209–3213.

[15] P. A. Gnoffo, R. N. Gupta, J. L. Shinn, Conservation Equations and Physical Models for Hypersonic Air Flows in Thermal and Chemical Nonequilibirum, Technical Report 2867, National Aeronautics and Space Adminisation, 1989.

[16] O. Knab, H. H. Fruhauf, E. W. Messerschmid, Theory and validation of a physically consistent coupled vibration-chemistry-

vibration model, Journal of Thermophysics and Heat Transfer 9 (1995). doi:10.2514/3.649.

[17] G. D. van Albada, B. van Leer, W. W. Roberts, A comparative study of computational methods in cosmic gas dynamics, Astronomy and Astrophysics 108 (1982) 76–84. doi:10.1007/978-3-642-60543-7_6.

[18] V. VenkataKrishnan, Convergence to steady state solutions of the euler equations on unstructured grids with limiters, Journal of Computational Physics 118 (1995) 120–130. doi:10.1006/jcph.1995.1084.

[19] M. N. Macrossan, The equilibrium flux method for the calculation of flows with non-equilibrium chemical reactions, Journal of Computational Physics 80 (1989) 204–231. doi:10.1016/0021-9991(89)90095-8.

[20] D. Haenel, R. Schwane, G. Seider, On the accuracy of upwind schemes for the solution of the Navier-Stokes equations, in: 8th Computational Fluid Dynamics Conference, 1105, Honolulu, HI, USA, 1987. doi:10.2514/6.1987-1105.

[21] Y. Wada, M.-S. Liou, A flux splitting scheme with high-resolution and robustness for discontinuities, in: 32nd AIAA Aerospace Sciences Meeting and Exhibit, AIAA-94-0083, Reno, Nevada, 1994. doi:10.2514/6.1994-83.

[22] A. Haselbacher, J. Blazek, Accurate and efficient discretization of Navier-Stokes equations on mixed grids, AIAA Journal 38 (2000) 2094–2102. doi:10.2514/2.871.

[23] D. R. Mott, E. S. Oran, B. van Leer, A quasi-steady-state solver for the stiff ordinary differential equations of reaction kinetics, Journal of Computational physics 164 (2000) 407–428. doi:10.1006/jcph.2000.6605.

[24] R. J. Gollan, The Computational Modelling of High-Temperature Gas Effects with Application to Hypersonic Flows, Ph.D. thesis, The University of Queensland, School of Mechanical and Mining Engineering, St Lucia, QLD 4072, 2008.

[25] G. Wilson, D. A. Aruliah, C. T. Brown, N. P. C. Hong, M. Davis, R. T. Guy, S. H. D. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, B. Waugh, E. P. White, P. Wilson, Best practices for scientific computing, PLoS Biology 12 (2014) 1–7. doi:10.1371/journal.pbio.1001745.

[26] J. M. Powers, T. D. Aslam, Exact solution for multidimensional compressible reactive flow for verifying numerical algorithms, AIAA Journal 44 (2006). doi:10.2514/1.14404.

[27] R. J. Gollan, P. A. Jacobs, About the formulation, verification and validation of the hypersonic flow solver Eilmer, International Journal for Numerical Methods in Fluids 73 (2013) 19–57. doi:10.1002/fld.3790.

[28] K. A. Damm, Adjoint-Based Aerodynamic Design Optimisation in Hypersonic Flow, Ph.D. thesis, The University of Queensland, School of Mechanical and Mining Engineering, St Lucia, QLD 4072, 2020. doi:10.14264/uql.2020.207.

[29] A. Veeraragavan, J. Beri, R. Gollan, Use of the method of manufactured solutions for the verification of conjugate heat transfer solvers, Journal of Computational Physics 307 (2016) 308–320. doi:10.1016/j.jcp.2015.12.004.

[30] P. A. Jacobs, Numerical simulation of transient hypervelocity flow in an expansion tube, Computers and Fluids 23 (1994) 77–101. doi:10.1016/0045-7930(94)90028-0.

[31] J. S. Jewell, C. C. Huffman, T. J. Juliano, Transient startup simulations for a large Mach 6 quiet Ludwieg tube, in: 55th AIAA Aerospace Sciences Meeting, AIAA-2017-0743, Grapevine, Texas, 2017. doi:10.2514/6.2017-0743.

[32] D. E. Gildfind, P. A. Jacobs, R. G. Morgan, W. Y. K. Chan, R. J. Gollan, Scramjet test flow reconstruction for a large-scale expansion tube, part 2: axisymmetric CFD analysis, Shock Waves 28 (2018) 899–918. doi:10.1007/s00193-017-0786-9.

[33] M. S. Holden, R. P. Wadhams, A database of aerothermal measurements in hypersonic flow in building block experiments for CFD validation, in: 41st AIAA Aerospace Sciences Meeting and Exhibit, AIAA-2003-1137, Reno, Nevada, 2003. doi:10.2514/6.2003-1137.

[34] M. MacLean, M. Holden, Validation and comparison of WIND and DPLR results for hypersonic, laminar problems, in: 42nd AIAA Aerospace Sciences Meeting and Exhibit, AIAA-2004-529, Reno, Nevada, 2004. doi:10.2514/6.2004-529.

[35] M. Sun, T. Saito, P. A. Jacobs, E. V. Timofeev, K. Ohtani, K. Takayama, Axisymmetric shock wave interaction with a cone: a benchmark test, Shock Waves 15 (2005) 313–331. doi:10.1007/s00193-005-0258-5.

[36] G. Park, S. L. Gai, A. J. Neely, Base flow of circular cylinder at hypersonic speeds, AIAA Journal 54 (2016). doi:10.2514/1.J054270.

[37] J.-J. O. E. Hoste, V. Casseau, M. Fossati, I. J. Taylor, R. J. Gollan, Numerical modeling and simulation of supersonic flows in propulsion systems by open source solvers, in: 21st International Space Planes and Hypersonic Systems and Technologies Conference, AIAA-2017-2411, Xiamen, China, 2017. doi:10.2514/6.2017-2411.

[38] G. V. Candler, Next-generation CFD for hypersonic and aerothermal flows, in: 22nd AIAA Computational Fluid Dynamics Conference, AIAA-2015-3048, Dallas, Texas, 2015. doi:10.2514/6.2015-3048.

[39] J. Ray, S. Kieweg, D. Dinzl, V. G. Weirs, B. Freno, M. Howard, T. Smith, I. N. ad G. V. Candler, Estimation of inflow uncertainties in laminar hypersonic double-cone experiments, AIAA Journal 58 (2020). doi:10.2514/1.J059033.

[40] R. K. Lobb, Experimental measurement of shock detachment distance on spheres fired in air at hypervelocites, in: W. C. Nelson (Ed.), The High Temperature Aspects of Hypersonic Flow, volume 68, 1962, pp. 519–527. doi:10.1016/B978-1-4831-9828-6.50031-X.

[41] S. Nonaka, H. Mizuno, K. Takayama, C. Park, Measurement of shock standoff distance for sphere in ballistic range, Journal of Thermophysics and Heat Transfer 14 (2000) 225–229. doi:10.2514/2.6512.

[42] R. J. Gollan, P. Jacobs, On the validation of a hypersonic flow solver using measurements of shock detachment distance, in: 18th Australiasian Fluid Mechanics Conference, Launceston, Tasmania, 2012.

[43] F. Zander, R. J. Gollan, P. A. Jacobs, R. G. Morgan, Hypervelocity shock standoff on spheres in air, Shock Waves 24 (2014) 171–178. doi:10.1007/s00193-013-0488-x.

[44] E. J. Fahy, D. R. Buttsworth, R. J. Gollan, P. A. Jacobs, R. G. Morgan, C. M. James, Experimental and computational fluid dynamics study of Hayabusa reentry peak heating, Journal of Spacecraft and Rockets 58 (2021) 1–14. doi:10.2514/1.A34863.

[45] N. Banerji, P. Leyland, E. Fahy, R. Morgan, Venus entry flow over a decomposing aeroshell in X2 expansion tube, Journal of Thermophysics and Heat Transfer 32 (2018) 292–302. doi:10.2514/1.T5172.

[46] D. F. Potter, T. Eichmann, A. Brandis, R. Morgan, P. A. Jacobs, T. J. McIntyre, Simulation of radiating CO_2-N_2 shock layer experiments at hyperbolic entry conditions, in: 40th AIAA Thermophysics Conference, AIAA-2008-3933, Seattle, Washington, 2008. doi:10.2514/6.2008-3933.

[47] N. Banerji, P. Leyland, E. Fahy, R. Morgan, Earth reentry flow over a phenolic aeroshell in the X2 expansion tube, Journal of Thermophysics and Heat Transfer 32 (2018) 414–428. doi:10.2514/1.T5255.

[48] S. Gu, R. G. Morgan, T. J. McIntyre, A. M. Brandis, An experimental study of CO_2 thermochemical nonequilibrium, AIAA

Journal 60 (2022). doi:`10.2514/1.J061037`.

[49] Y. Liu, C. M. James, R. G. Morgan, P. A. Jacobs, R. J. Gollan, T. J. McIntyre, Electron number density measurements in a Saturn entry condition, AIAA Journal 60 (2022). doi:`10.2514/1.J060560`.

[50] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal on Scientific Computing 20 (1999) 359–392. doi:`10.1137/S1064827595287997`.

[51] C. Park, Review of chemical-kinetic problems of future NASA missions, I: Earth entries, Journal of Thermophysics and Heat Transfer 7 (1993). doi:`10.2514/3.431`.

[52] D. S. Liechty, C. O. Johnston, M. J. Lewis, Comparison of DSMC and CFD solutions of Fire II including radiative heating, in: 42nd AIAA Thermophysics Conference, AIAA-2011-3494, Honolulu, Hawaii, 2011. doi:`10.2514/6.2011-3494`.

[53] D. Hash, J. Olejniczak, M. Wright, D. Prabhu, M. Pulsonetti, B. Hollis, P. Gnoffo, M. Barnhardt, I. Nompelis, G. Candler, FIRE II calculations for hypersonic nonequilibrium aerothermodynamics code verification: DPLR, LAURA, and US3D, in: 45th Aerospace Sciences Meeting and Exhibit, AIAA-2007-605, Reno, Nevada, 2007. doi:`10.2514/6.2007-605`.

[54] B. Adams, L. Bauman, W. Bohnhoff, K. Dalbey, M. Ebeida, J. Eddy, M. Eldred, P. Hough, K. Hu, J. Jakeman, Dakota: A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Etc: Version 6 User's Manual, Technical Report SAND2014-4633, Sandia National Lab, 2015.

[55] K. A. Damm, R. J. Gollan, P. A. Jacobs, M. K. Smart, S. Lee, E. Kim, C. Kim, Discrete adjoint optimization of a hypersonic inlet, AIAA Journal 58 (2020). doi:`10.2514/1.J058913`.

[56] H. G. Hornung, R. J. Gollan, P. A. Jacobs, Unsteadiness boundaries in supersonic flow over double cones, Journal of Fluid Mechanics 916 (2021) 1–23. doi:`10.1017/jfm.2021.203`.

Appendix 1: Manufactured Solution for 3D Viscous, Turbulent Flow

$$\rho = 1 - \frac{\sin\left(\frac{9\pi y}{20L}\right)}{10} + \frac{\sin\left(\frac{4\pi z}{5L}\right)}{10} + \frac{3\sin\left(\frac{\pi xz}{2L^2}\right)}{25} + \frac{3\cos\left(\frac{3\pi x}{4L}\right)}{20} + \frac{2\cos\left(\frac{13\pi xy}{20L^2}\right)}{25} + \frac{\cos\left(\frac{3\pi yz}{4L^2}\right)}{20}$$

$$u = 70 + 7\sin\left(\frac{\pi x}{2L}\right) - 4\sin\left(\frac{9\pi xz}{10L^2}\right) - 15\cos\left(\frac{17\pi y}{20L}\right) - 10\cos\left(\frac{2\pi z}{5L}\right) + 7\cos\left(\frac{3\pi xy}{5L^2}\right) + 4\cos\left(\frac{4\pi yz}{5L^2}\right)$$

$$v = 90 - 5\sin\left(\frac{4\pi x}{5L}\right) + 5\sin\left(\frac{3\pi xz}{5L^2}\right) + 10\cos\left(\frac{4\pi y}{5L}\right) + 5\cos\left(\frac{\pi z}{2L}\right) - 11\cos\left(\frac{9\pi xy}{10L^2}\right) - 5\cos\left(\frac{2\pi yz}{5L^2}\right)$$

$$w = 80 + 10\sin\left(\frac{9\pi y}{10L}\right) - 12\sin\left(\frac{2\pi xy}{5L^2}\right) + 5\sin\left(\frac{3\pi xz}{4L^2}\right) - 10\cos\left(\frac{17\pi x}{20L}\right) + 12\cos\left(\frac{\pi z}{2L}\right) + 11\cos\left(\frac{4\pi yz}{5L^2}\right)$$

$$p = 100,000 + 20,000\sin\left(\frac{17\pi z}{20L}\right) + 10,000\sin\left(\frac{4\pi xz}{5L^2}\right) + 20,000\cos\left(\frac{2\pi x}{5L}\right) + 50,000\cos\left(\frac{9\pi y}{20L}\right) - 25,000\cos\left(\frac{3\pi xy}{4L^2}\right) - 10,000\cos\left(\frac{7\pi yz}{10L^2}\right)$$

$$\hat{v} = 1 + \frac{4\sin\left(\frac{4\pi z}{5L}\right)}{5} - \frac{3\sin\left(\frac{3\pi xz}{5L^2}\right)}{5} + \frac{6\cos\left(\frac{7\pi x}{20L}\right)}{25} - \frac{3\cos\left(\frac{2\pi y}{5L}\right)}{10} + \frac{3\cos\left(\frac{\pi xy}{2L^2}\right)}{4} + \frac{\cos\left(\frac{\pi yz}{4L^2}\right)}{2}$$