Introduction to CFD for Hypersonic Flows with Eilmer

Rowan Gollan and Peter Jacobs

Day 1: 9 May 2018

The University of Queensland

Short Course Outline, Day 1

Morning: presentation & discussion

- Overview of Eilmer, a brief tour
- Theory & Formulation
- Implementation, Verification & Validation

Afternoon: hands-on activities

- Installing Eilmer on a laptop / workstation
- First example: cone in supersonic flow
  - introduction to Lua as input
  - pre-processing
  - running a simulation
  - post-processing
  - discussion of directory layout and files
- Second example: laminar flow over a flat plate
- Tips & tricks:
  - restarting a simulation
  - using a previously completed solution as initial state
- Third example: blunt wedge

Short Course Outline, Day 2

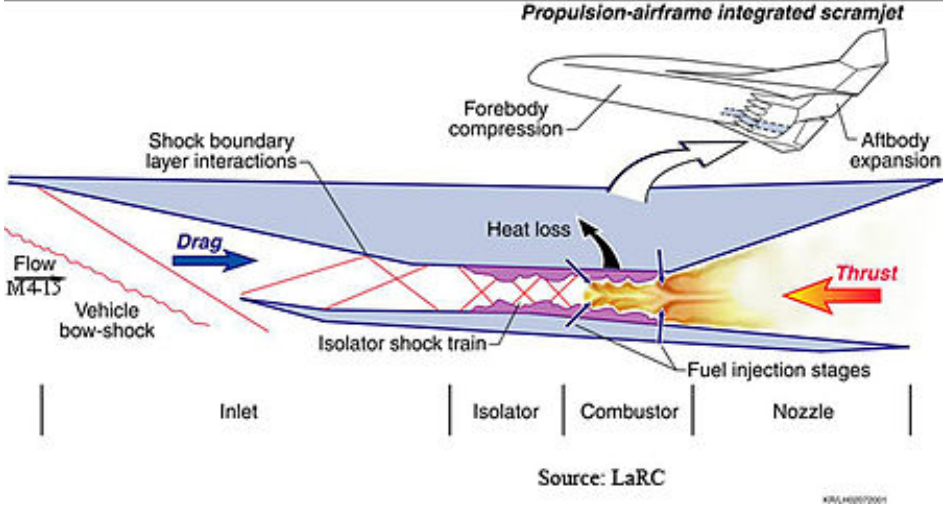
Morning: presentation & discussion

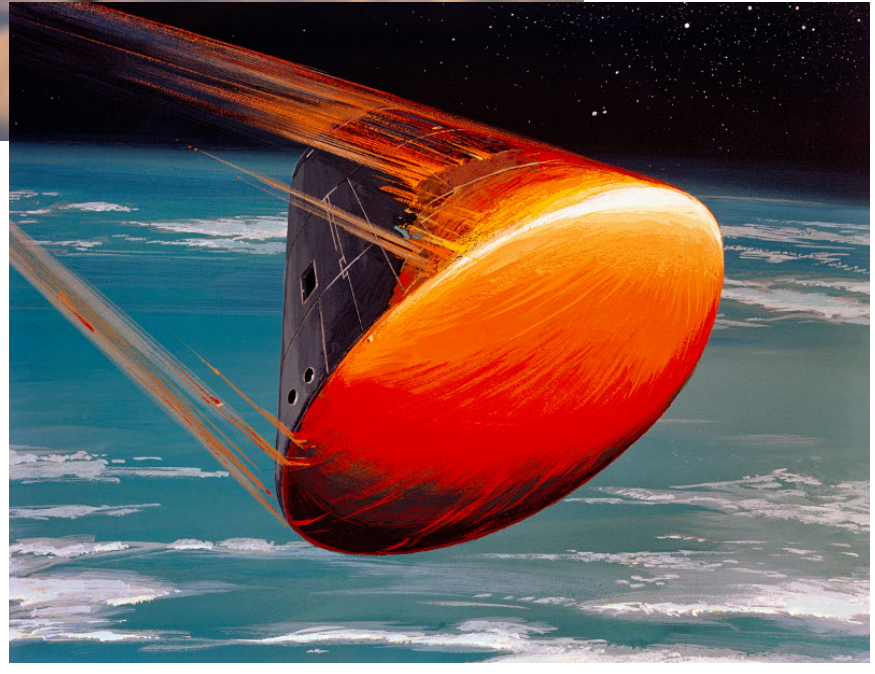
- Eilmer for simulation of hypersonic flows
- Advanced thermochemistry
- Turbulence modelling
- Parallel computing: small & large scale
- Advanced/experimental features
  - user-defined BCs and source terms
  - shock-fitting boundaries
  - moving grid
  - block-marching mode
  - wall functions for turbulence
  - GPU-acceleration of reacting flows
  - steady-state solver
  - adjoint solver for optimisation
  - state-specific chemistry coupled to flow

Afternoon: hands-on activities

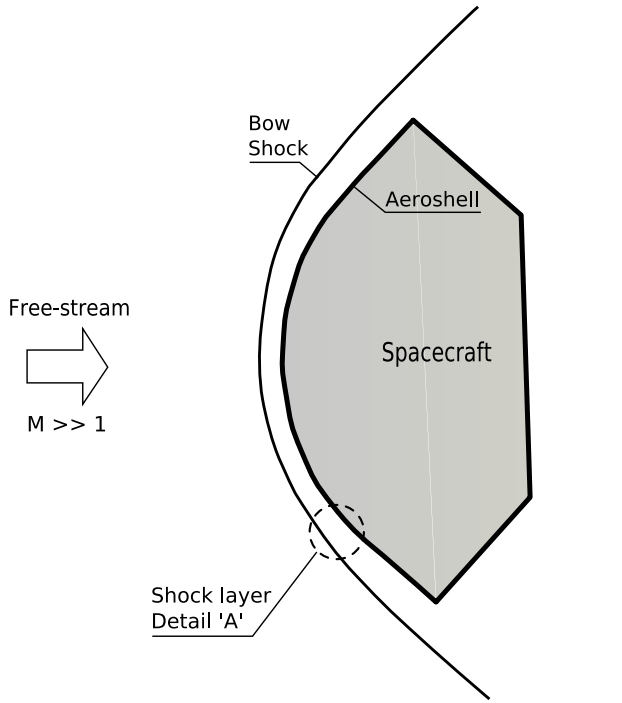
- Installing MPI version of Eilmer
- Working in 3D
  - importing grids from 3rd-party grid generation tools
  - partitioning structured and unstructured grids
- Fourth example: Reacting air flow over a sphere
- Fifth example: shock-wave boundary-layer interaction
- Tips & tricks:
  - running simulations in parallel
  - using the block-marching mode
- Bring-your-own examples

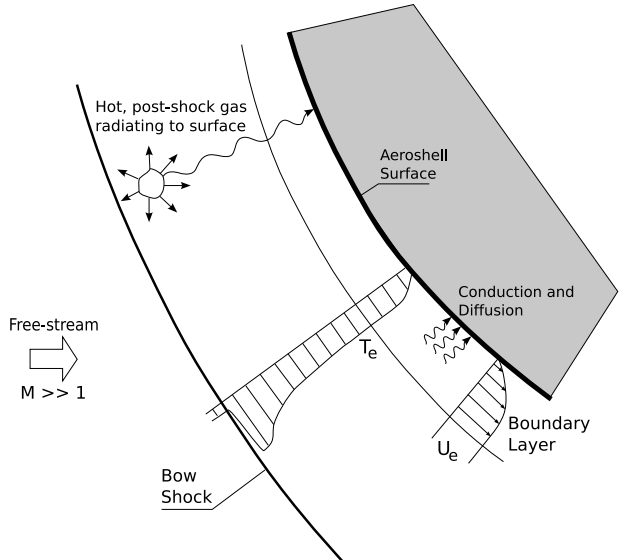
Why hypersonics?





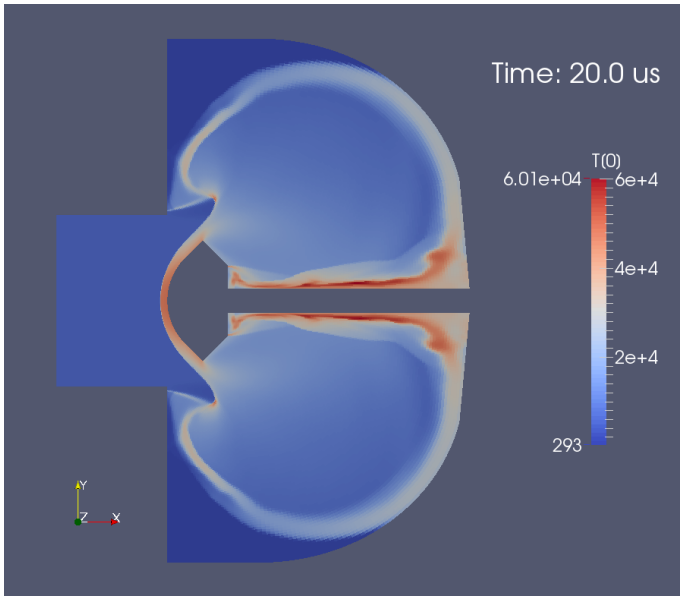
Why computer simulation of hypersonic flows?





- The flow physics are modelled reasonably well, but the interactions are complex.
- Physical experimentation provides insights but has limitations, such as: scaling (time and length), boundary conditions, quantification of uncertainties, expense
- Computer simulation complements physical experiments, and vice versa.
- Analysis via computer simulation (might) substitute when we don't have suitable experience.
- Computer analysis good for 'what-if' studies, and design.

Eilmer features – 1/2



- 2D/3D compressible flow simulation.
- Gas models include ideal, thermally perfect, equilibrium.
- Finite-rate chemistry.
- Multi-temperature and state-specific thermochemistry.
- Inviscid, laminar, turbulent (k-ω) flow.
- Solid domains with conjugate heat transfer in 2D.
- User-controlled moving grid capability, with shock-fitting method for 2D geometries.
- Dense-gas thermodynamic models and rotating frames of reference for turbomachine modelling.

Eilmer features – 2/2

- Transient, time-accurate, using explicit Euler, PC, RK updates.
- Alternate steady-state solver with implicit updates using Newton-Krylov method.
- Parallel computation via shared-memory on workstations, and using MPI on a cluster computer.
- Multiple block, structured and unstructured grids.
- Native grid generation and import capability.
- Unstructured-mesh partitioning via Metis.

- `en.wikipedia.org/wiki/Eilmer_of_Malmesbury`
- Gas model calculator and compressible flow relations.

Origins

- in the late 1980s, the state of the art for scramjet simulations involving reactive flow was JP Drummond *SPARK* code
- Flow solver component based on Bob McCormack's (1969) finite-difference shock-capturing technique.
- All configuration hard-coded into the Fortran source code and compiled to run on a Cray supercomputer.
- In the 1980s, a new CFD technology (upwind flux) was being developed by the applied mathematics people and parallel computing environments were being developed by the computer science people (cluster computers).
- Dec 1990: following a CFD lesson on the chalk-board from Bob Walters and Bernard Grossman, *cns4u* was started with the intention to be like SPARK but with new technology

Development of Eilmer

- 1993 built *sm3d*, a space-marching code for 3D scramjet flows
- 1995 through 1999: the postgrad years expanded scope of experimentation and application
- 1996: code reformulation around fluxes (frequent discussions with Mike Macrossan); all code still in C with a preprocessor having a little command interpreter built in.
- 1997: discovered scripting languages Tcl and Python
- May 2003: *scriptit.tcl* provided fully programmable environment for simulation-preparation.
- Aug 2004: *Elmer* began as a hybrid code using Python and C.
- Jun 2005: rewrite of *Elmer(2)* in C alone.
- Jul 2006: rewrite *Elmer2* in C++ and, in 2008, call it *Eilmer3*. Class-based implementation was easier to extend.

Eilmer contributors over the years

The Eilmer3 Code: User Guide and Example Book
2015 Edition

Mechanical Engineering Report 2015/07

Peter A. Jacobs* Rowan J. Gollan† Ingo Jahn‡ and Daniel F. Potter§

with contributions¶ from a cast of many, including:
Ghassan Al'Doori, Nikhil Banerji, Justin Beri, Peter Blyton, Daryl Bond, Arianna Bosco,
Djamel Boutamine, Laurie Brown, David Buttsworth, Wilson Chan, Sam Chiu,
Chris Craddock, Brian Cook, Jason Czapla, Kyle Damm, Andrew Dann,
Andrew Denman, Zac Denman, Luke Doherty, Elise Fahy, Antonia Flocco,
Delphine Francois, James Fuata, Nick Gibbons, David Gildfind, Richard Goozeé, Sangdi Gu,
Stefan Hess, Carolyn Jacobs, Chris James, Ian Johnston, Ojas Joshi, Xin Kang,
Rainer Kirchhartz, Sam Lamboo, Steven Lewis, Tom Marty, Matt McGilvray, David Mee,
Carlos de Miranda-Ventura, Luke Montgomery, Jan-Pieter Nap, Brendan O'Flaherty,
Andrew Pastrello, Paul Petrie-Repar, Jorge Sancho Ponce, Jason (Kan) Qin,
Deepak Ramanath, Andrew Rowlands, Michael Scott, Umar Sheikh, Sam Stennett,
Ben Stewart, Joseph Tang, Katsu Tanimizu, Pierpaolo Toniato,
Paul van der Laan, Tjarke van Jindelt, Anand Veeraragavan, Jaidev Vesudevan,
Han Wei, Mike Wendt, Brad (The Beast) Wheatley, Vince Wheatley,
Adriaan Window, Hannes Wojciak, Fabian Zander, Mengmeng Zhao

School of Mechanical and Mining Engineering,
The University of Queensland.

August 1, 2015

Eilmer4 – think big, but control the complexity

- Jun 2015+: rebuild in the D and Lua programming languages.
- Heather Muir worked on the unstructured-grid generator. based on the paving algorithm.

Eilmer – find out more

Website:

`cfcfd.mechmining.uq.edu.au/eilmer`

Source code repository:

`bitbucket.org/cfcfd/dgd`

Documentation in the Eilmer 4.0 guides:

- *Guide to the transient flow solver*
- *Guide to the basic gas models package*
- *Guide to the geometry package*
- *Formulation of the transient flow solver*
- *Reacting gas thermochemistry*

Theory: formulation, verification and validation

Overview: compressible flow CFD via finite volumes

- What we solve: governing equations for a viscous compressible flow
- How we solve: discretisation in space and time
- Global view of the update algorithm
- Convective fluxes: reconstruction-evolution approach, interpolation order, flux calculators, limiters
- Time integration and the CFL condition
- Diffusive fluxes
- Boundary conditions

What we solve

Integral form of a conservation law

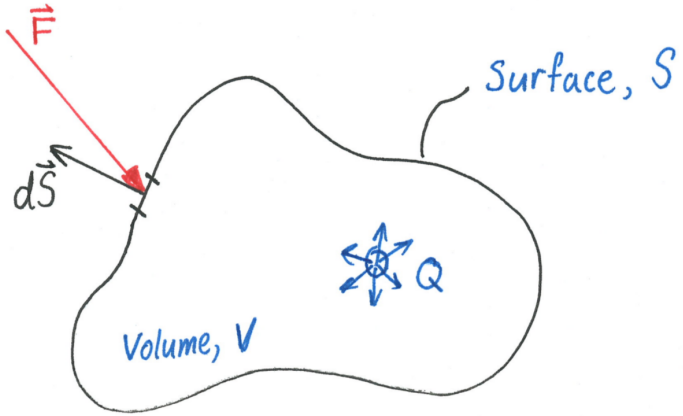
A general conservation law for quantity U is written in integral form as

$$\frac{\partial}{\partial t}\int_V U dV = -\oint_S \left(\vec{F}_c - \vec{F}_d\right)\cdot \hat{n}\, dA + \int_V Q dV\,, \qquad (1)$$

where S is the bounding surface and $\hat{n}$ is the outward-facing unit normal of the control surface.

What are the quantities U?

The conserved quantities in a compressible flow are **mass**, **momentum** and **energy**. In two dimensions, we can group these conservation equations with vector notation.



Integral form of conservation laws in vector form

For an ideal gas in two dimensions, the vector of conserved quantities is:

$$U = \begin{bmatrix} \rho \\ \rho u_x \\ \rho u_y \\ \rho E \end{bmatrix}, \tag{2}$$

with convective flux vector

$$\vec{F}_c = \begin{bmatrix} \rho u_x \\ \rho u_x^2 + p \\ \rho u_y u_x \\ \rho E u_x + p u_x \end{bmatrix} \hat{i} + \begin{bmatrix} \rho u_y \\ \rho u_x u_y \\ \rho u_y^2 + p \\ \rho E u_y + p u_y \end{bmatrix} \hat{j}, \tag{3}$$

and diffusive flux vector

$$\overline{F}_d = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{yx} \\ \tau_{xx} u_x + \tau_{yx} u_y + q_x \end{bmatrix} \hat{i} + \begin{bmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ \tau_{xy} u_x + \tau_{yy} u_y + q_y \end{bmatrix} \hat{j}. \tag{4}$$

The vector of sources Q is typically zero.

Expansion of diffusive fluxes

The shear stresses in 2D are:

$$\begin{aligned} \tau_{xx} &= 2\mu\frac{\partial u_x}{\partial x} + \lambda\left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y}\right), \\ \tau_{yy} &= 2\mu\frac{\partial u_y}{\partial y} + \lambda\left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y}\right), \\ \tau_{xy} = \tau_{yx} &= \mu\left(\frac{\partial u_x}{dy} + \frac{\partial u_y}{dx}\right), \end{aligned}$$

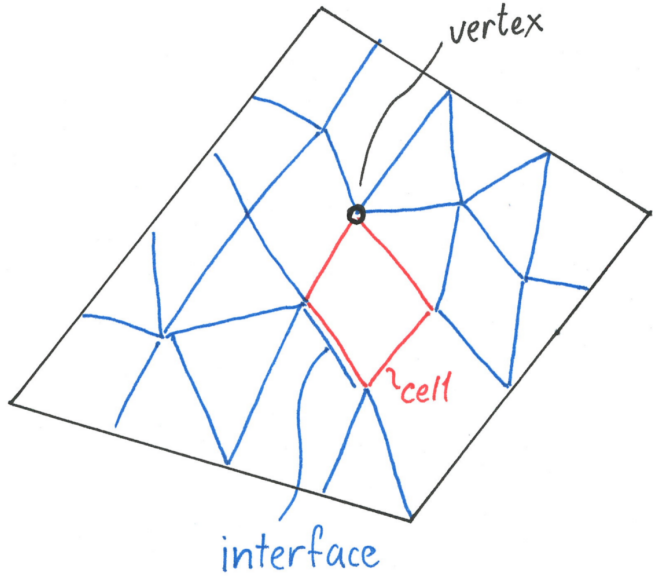
where the secondary viscosity coefficient λ is related to the primary (dynamic) viscosity coefficient μ via Stokes' hypothesis, $\lambda = -\frac{2}{3}\mu$.

The diffusive heat fluxes are:

$$\begin{aligned} q_x &= -k\frac{\partial T}{\partial x} \\ q_y &= -k\frac{\partial T}{\partial y}, \end{aligned}$$

where k is the thermal conductivity of the gas.

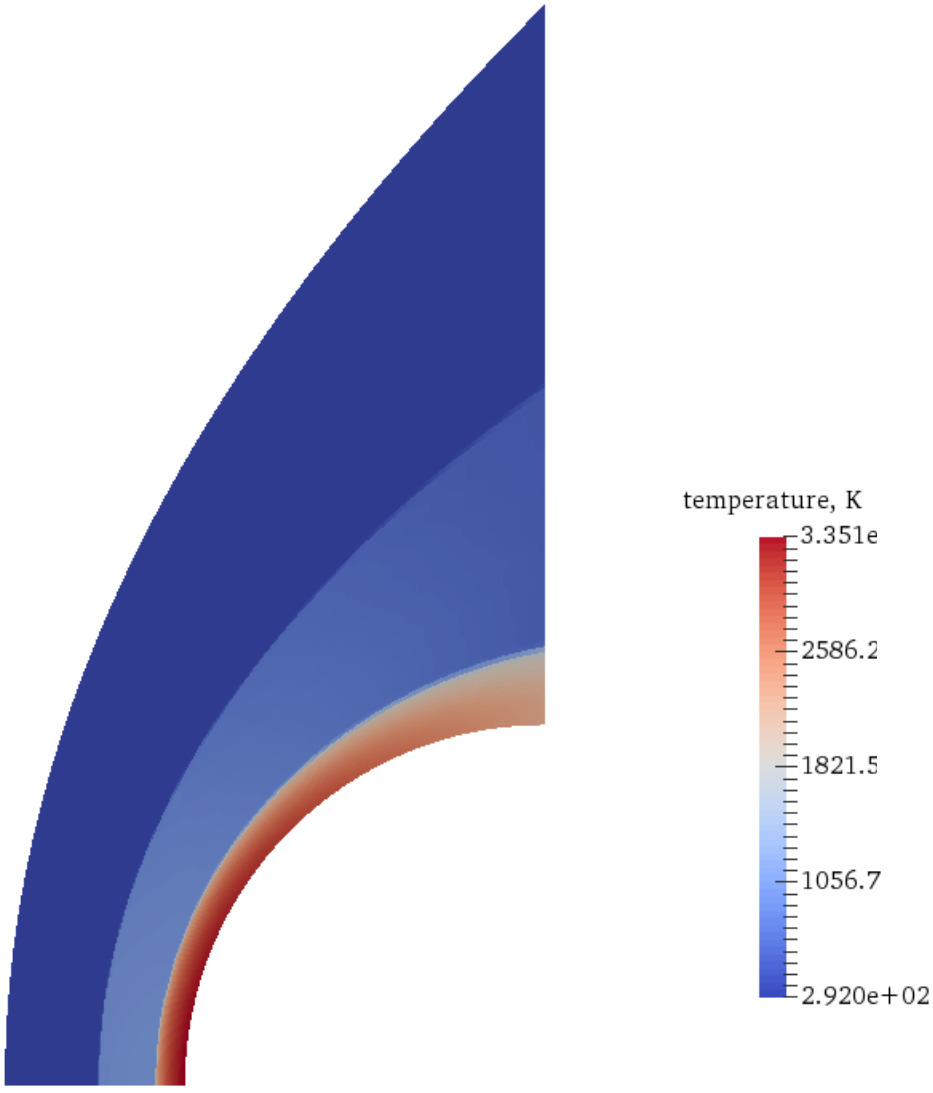
How we solve: discretisation in space, finite-volume method

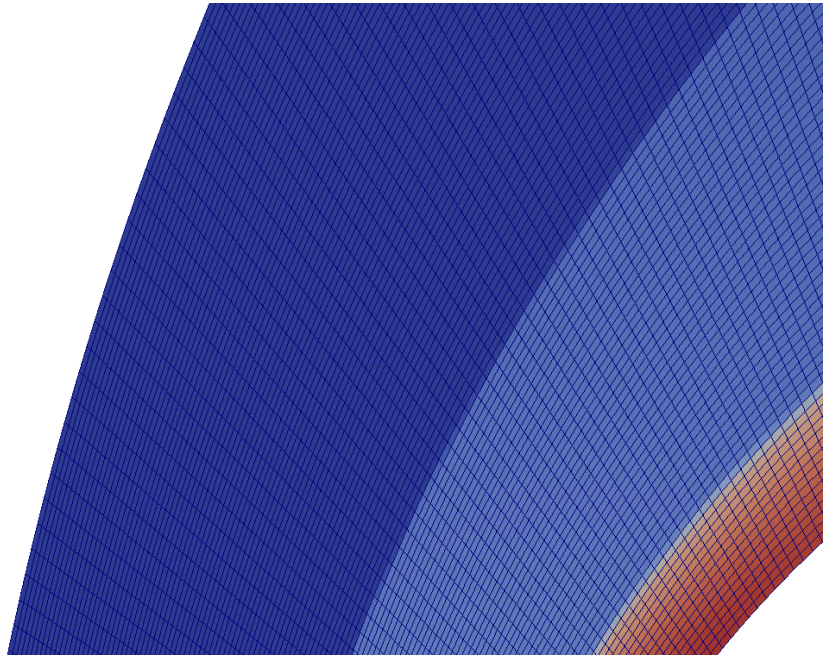


Some nomenclature

cells:	the finite volumes used to discretise the domain the flow solution is computed as average quantities in the cells
interfaces:	the edges (or faces, in 3D) of cells
vertices:	points in space forming the corners of cells

A flow domain with discretisation





Spatial discretisation: semi-discretised equations

$$\frac{dU}{dt} = -\frac{1}{V} \sum_{\text{cell-surface}} \left(\vec{F}_c - \vec{F}_d \right) \cdot \hat{n}\, dA + Q \tag{5}$$

- The right-hand side (RHS) of Eqn 5 is now discretised — it forms a set of algebraic expressions.
- If our finite volumes completely fill the domain *(they should!)*, we see that this discretisation is inherently conservative: flux out of one cell is balanced as a flux in to an adjoining cell.
- The complete system is a system of ordinary differential equations.
- This system can be integrated forward in time using numerical methods for ODEs. We typically stick to low-order time methods because of the complexity and computer memory required by evaluation of the RHS.
- More detail on how we compute $\sum_{\text{cell-surface}} \left(\vec{F}_c - \vec{F}_d \right) \cdot \hat{n}\, dA$ are presented later.
- But first, let's talk about using operator-splitting on this system.

Operator-splitting approach to update flow system

Operator-splitting, also known as timestep-splitting, is a solution technique whereby the system is 'split' and each part is updated in turn.

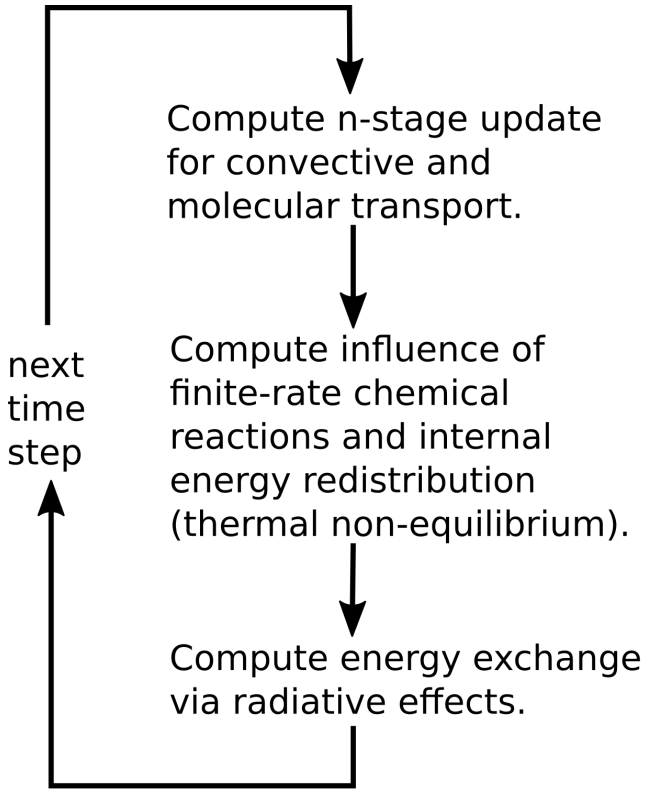
$$\int_{\Delta t} \frac{dU}{dt} dt = \int_{\Delta t} \left(\frac{dU}{dt}\right)_{\text{conv.}} dt + \int_{\Delta t} \left(\frac{dU}{dt}\right)_{\text{diff.}} dt \,, \tag{6}$$

where

$$\begin{aligned} \left(\frac{dU}{dt}\right)_{\text{conv.}} &= -\frac{1}{V} \sum_{\text{cell-surface}} \left(\vec{F}_c\right) \cdot \hat{n}\, dA + Q \,, \\ \left(\frac{dU}{dt}\right)_{\text{diff.}} &= -\frac{1}{V} \sum_{\text{cell-surface}} \left(-\vec{F}_d\right) \cdot \hat{n}\, dA \,. \end{aligned}$$

- advantage: use of the best numerical integration schemes based on the type of system
- disadvantage: the global solution is subject to the numerical instabilities that could arise from any of the individual split processes

Global view of update algorithm



for s=1 to n do:

- clear flux data
- apply pre-reconstruction action
- detect shock points
- reconstruct flow data at cell interfaces
- compute convective fluxes
- apply post-convective-flux action
- apply pre-spatial-derivative action
- compute spatial derivatves
- apply post-diffusion flux action
- add source terms, if any
- compute time derivatives of conserved quantities
- update cell-average conserved quantities for stage s
- decode conserved quantities to all flow quantities

Convective flux update: overview

The goal is to numerically approximate the convective fluxes across cell surfaces and update the cell quantity based on those approximations.

The convective flux update can be divided (broadly) into 4 steps. Steps 1 and 2 are focussed on the interfaces in the domain:

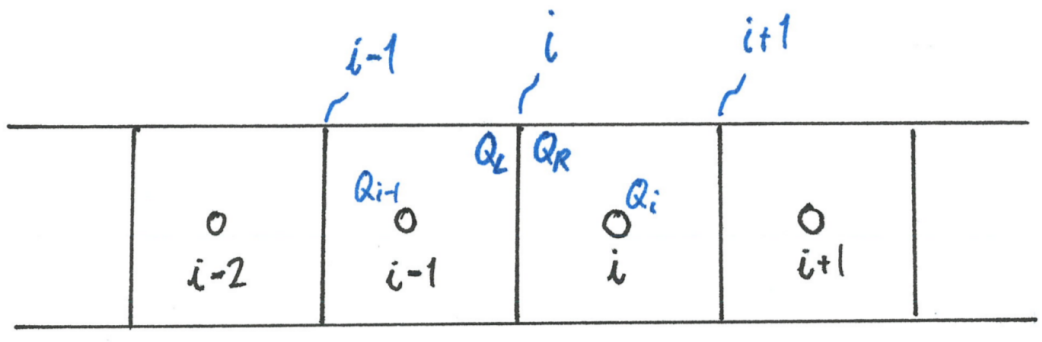
1. Reconstruction: use some form of interpolation to estimate the flow states either side of an interface.
2. Flux calculation: using the reconstructed flow states either side of an interface, compute the numerical flux through the interface.

Steps 3 and 4 are from the cell's perspective:

3. Compute time derivatives: sum fluxes on bounding surfaces of cell to get the time rate of change of quantities U.
4. Update cell-centre values: using a numerical integration technique.

Convective flux update: reconstruction

We'd like to compute the flux through an interface based on the flow states either side of the interface, so we need a means of estimating those flow states. However, the finite volume formulation only gives us cell average information. The process of estimating *Left* and *Right* flow states from nearby cell averages is called **reconstruction**.



The simplest approach would be to take:

$$Q_L = Q_{i-1}; \quad Q_R = Q_i$$

This in fact leads to a scheme that is first order accurate in space.

```
config.interpolation_order = 1
```

Reconstruction on structured grids

Higher-order approaches involve use of more nearby cells leading to higher order spatial accuracy.

On structured grids for interface i, Eilmer's reconstruction makes use of cells:

$$[Q_{i-2}, Q_{i-1}, Q_i, Q_{i+1}]$$

The (unlimited) reconstruction is **3rd order accurate in space**.

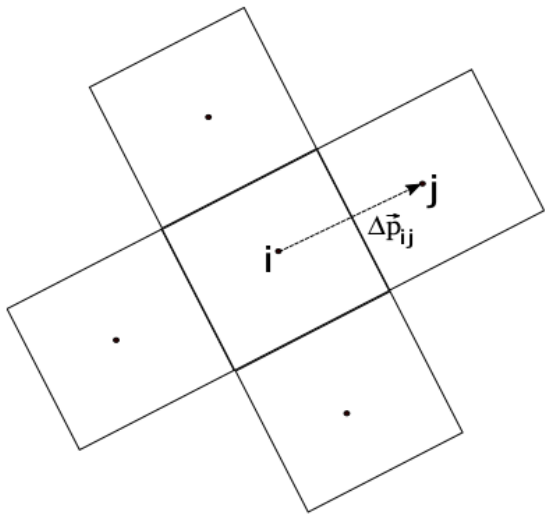
For robust shock-capturing, a limiter is applied. On structured grids we use the **van Albada limiter**. It puts a limit on the reconstructed values before passing them to the flux calculator.

```
config.interpolation_order = 2
```

Why 2 not 3? The mid-point integration used in the finite-volume has second order truncation error. Despite this, the use of 3rd-order reconstruction gives some benefit at no real extra computational expense.

Reconstruction on unstructured grids

A weighted least squares reconstruction is used based on neareset neighbour cells.



Reconstructed values require limiting.

Options are:

- Venkatakrishnan
- `min-mod`
- Barth
- van Albada
- MLP

```
config.interpolation_order = 2
```

A note on reconstruction and grid quality

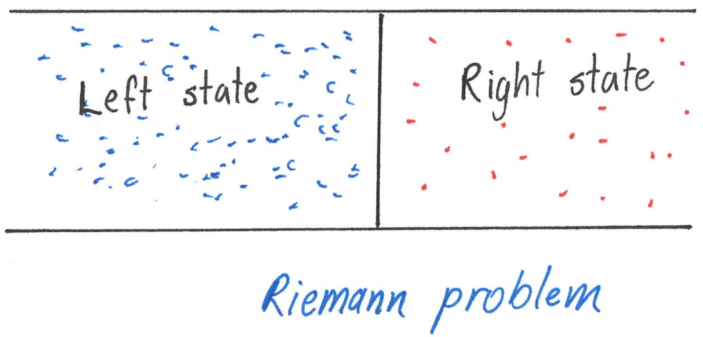
- Those with some CFD experience may be familiar with the link between the quality of the grid (smooth variations in cell size, mostly orthogonal cells) and the quality and robustness of the numerical solution. You may have even had someone else tell you: *"Improve your grid."*
- The principal link between the grid quality and the numerical solution quality is the reconstruction.
- Why? because of the assumptions built into the reconstruction methods.
- On structured grids, we've assumed that cells are (mostly) aligned in one direction. Some reconstruction schemes even assume that all cells are of equal width locally.
- On unstructured grids, we need to solve a least-squares problem. The geometric distribution of neighbouring cell influences the quality of the solution to the least-squares problem. We are essentially trying to fit a plane through the collection of nearby points.
- Typically, as our grids deviate from these assumptions, the quality of the solution degrades, particularly in regions of strong flow gradients.

Convective flux update: flux calculation

Given flow states either side of the interface, the **flux calculator** algorithm is required to estimate the numerical flux across the interface. There is an enormous variety of approaches to the problem of flux calculation. Flux calculation is the same on both structured and unstructured grids since we've reduced it to a Riemann problem.

Flux calculators available in Eilmer:

- AUSMDV
- AUSM-plus-up
- EFM
- HLLE
- adaptive-EFM-AUSMDV
- adaptive-HLLE-AUSMDV

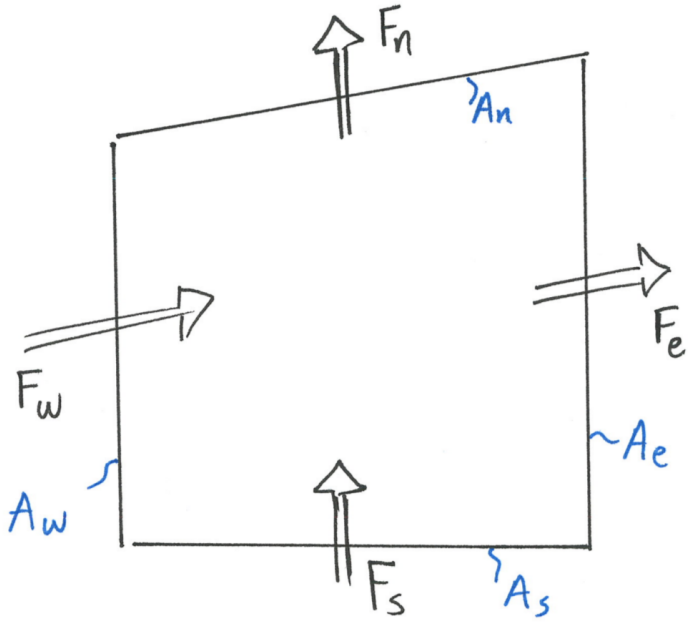


The adaptive calculators are designed for flows with strong shocks. A dissipative calculator is used near shocks (EFM or HLLE) and a low-dissipation calculator (AUSMDV) is used elsewhere.

Convective flux: computing time derivatives

Sum contribution from fluxes across bounding surfaces.

$$\left(\frac{dU}{dt}\right)_{\text{conv.}} = -\frac{1}{V}\left(-F_n A_n - F_e A_e + F_s A_s + F_w A_w\right)$$



This expression generalises for arbitrary cell types on 2D and 3D unstructured grids.

Convective flux update: time integration

The flow solution may be advanced in time using standard ODE solution methods.

Explicit techniques

- include forward Euler, predictor-corrector, multi-stage Runge-Kutta methods
- easy to program
- typically low memory requirements (compared to implicit)
- timestep size is limited by stability considerations

Implicit techniques

- stable for large timesteps timestep beyond explicit timestep limits, not necessarily accurate at large steps
- in practice, approximations in formulation put limits on the timestep for stability
- can be difficult to implement (influenced by choice of flux calculator)
- can have large memory requirements (scale as n^2 where n is number of equations to solve)

Convective flux update: CFL condition

The Courant-Friedrichs-Lewy (CFL) condition states that the numerical domain of dependence must wholly include the physical domain of dependence. This can be interpreted to a give a timestep limit Δt in terms of the cell size and the fastest characteristic wave in the flow:

$$\Delta t < \frac{\Delta x}{|\mathbf{u}| + a}$$

Depending on the chosen time integration scheme, the numerical domain of dependence might be larger or smaller than that by the CFL condition. Furthermore, the CFL condition does not account for nonlinearities, so we might want to be conservative in its application. We can modify the expression by a multiplier α to give an expression for the allowable timestep:

$$\Delta t_{\text{allowable}} < \alpha \frac{\Delta x}{|\mathbf{u}| + a}$$

In this form, α is commonly called the CFL value.

```
config.cfl_value = 0.4
```

More on the CFL value

It is a (reasonably) expensive operation to check the CFL condition. The $\Delta t_{\text{allowable}}$ must be checked at *every* cell and search performed across the flow domain to find the smallest value. This also forces a global synchronisation step when running in parallel.

In Eilmer, we only check the CFL condition after a certain number of timesteps have passed. The default is every 10 steps. Users may select to change how frequently the CFL value is checked.

```
config.cfl_count = 10
```

Recommended CFL values based on update scheme

Euler	$<$	0.4
predictor-corrector	$<$	0.7
3-stage Runge-Kutta	$<$	1.3

Diffusive flux update

The steps for performing the diffusive flux update are:

1. Spatial derivative estimate: required for shear stress and heat conduction terms
2. Assemble fluxes at interfaces
3. Compute time derivatives
4. Update cell-centre values

Spatial derivatives for diffusive fluxes

In Eilmer, in 2D on structured grids the recommended spatial derivative method is to use the **divergence theorem** based around **vertices**.

```
config.spatial_deriv_calc = 'divergence'
config.spatial_deriv_locn = 'vertices'
```

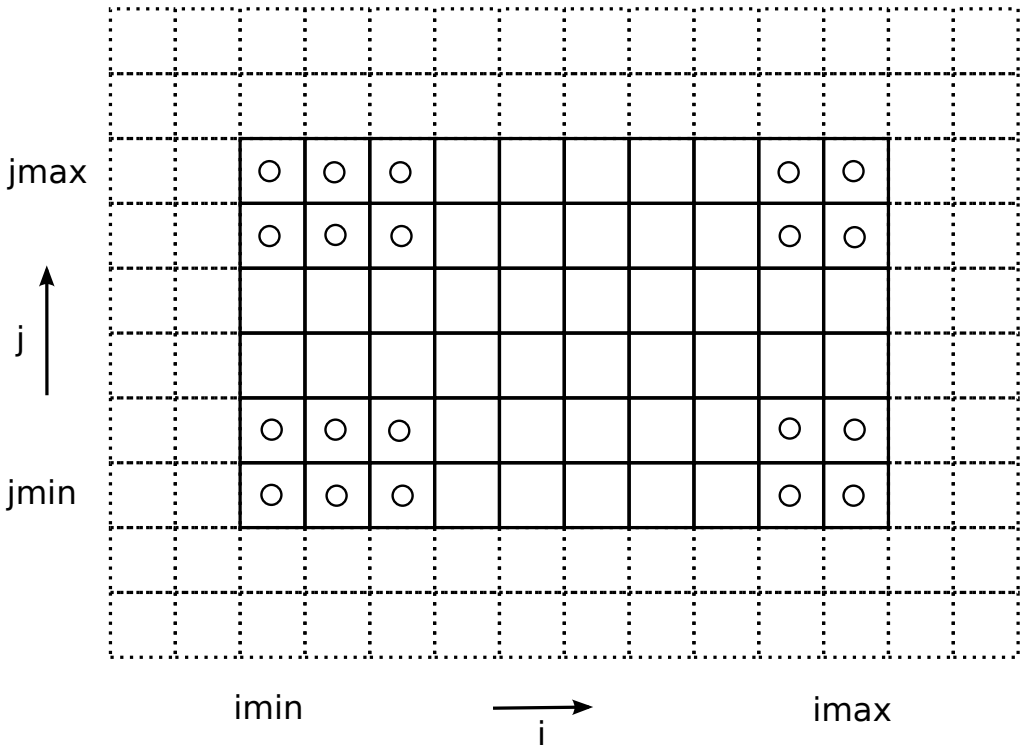
For all other cases, 2D unstructured, 3D structured or unstructured, the choices are[1]:

- unweighted least squares
- 2-point derivative estimate

[1] These can be used on 2D structured also, but we recommend the divergence theorem.

Boundary conditions

At the edge of the domain, we need to supply some information about how the flow at the boundary behaves. Sometimes it is useful to extend the domain with a few imaginary cells and fill them with appropriate flow conditions. In this way, we can use the regular reconstruction/flux calculation method to get the effect of the boundary conditions. These imaginary cells are called **ghost cells** in Eilmer. They are also used at the boundaries of block connections as a buffer to transfer flow data.

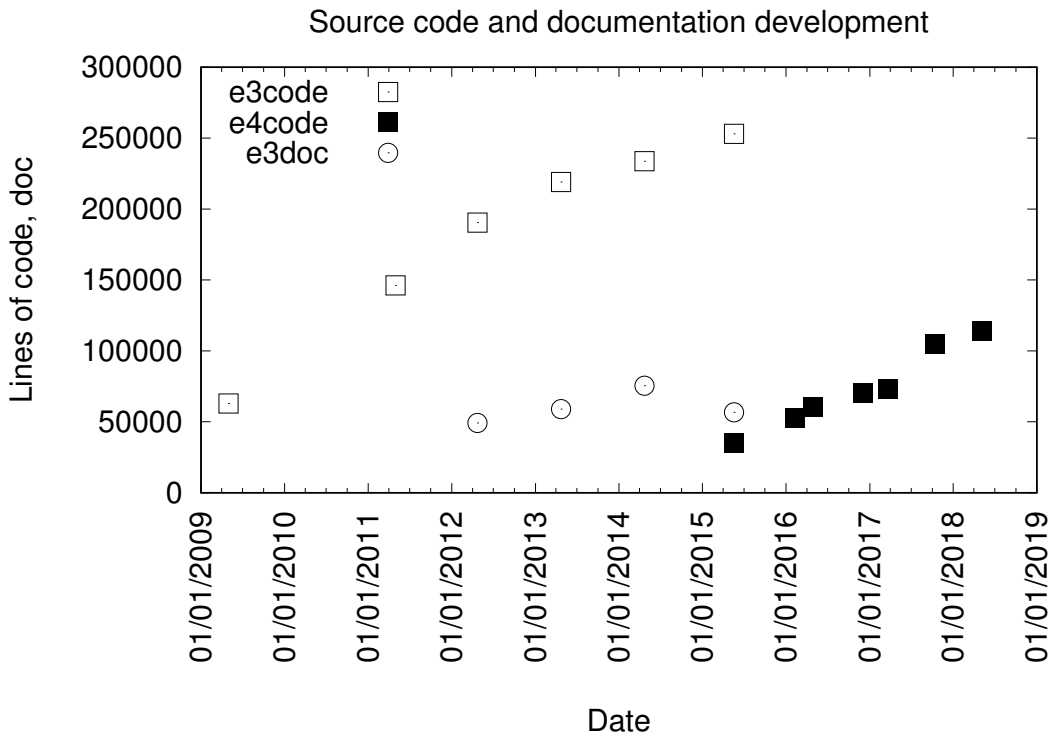


In other cases, it is more useful to specify the flow condition at the boundary interface directly (a Dirichlet condition). Other times, we know something about the flux at the boundary interfaces and so we set those values (a Neumann condition).

Implementation, Verification & Validation

Implementation

D language data storage and solver, with embedded Lua interpreters for preprocessing, user-controlled run-time configuration in boundary conditions and source terms and thermochemical configuration.



At 60 lines per page, the Eilmer4 code is equivalent to a 1200 page document.

Implementation: packages and modules

```
dgd
|-- extern
|   |-- lua-5.1.4
|   `-- zlib-1.2.3
`-- src
    |-- eilmer
    |-- extern
    |-- gas
    |-- gasdyn
    |-- geom
    |-- grid_utils
    |-- kinetics
    |-- moc
    |-- nm
    |-- quodas
    `-- util
```

Quality Control: Verification & Validation

To quote Blottner:

> *Verification: are we solving the equations right?*
> *Validation: are we solving the right equations?*

Verification:

- comparison to exact solutions
- comparison to manufactured solutions
- order of accuracy test
- code-to-code comparison

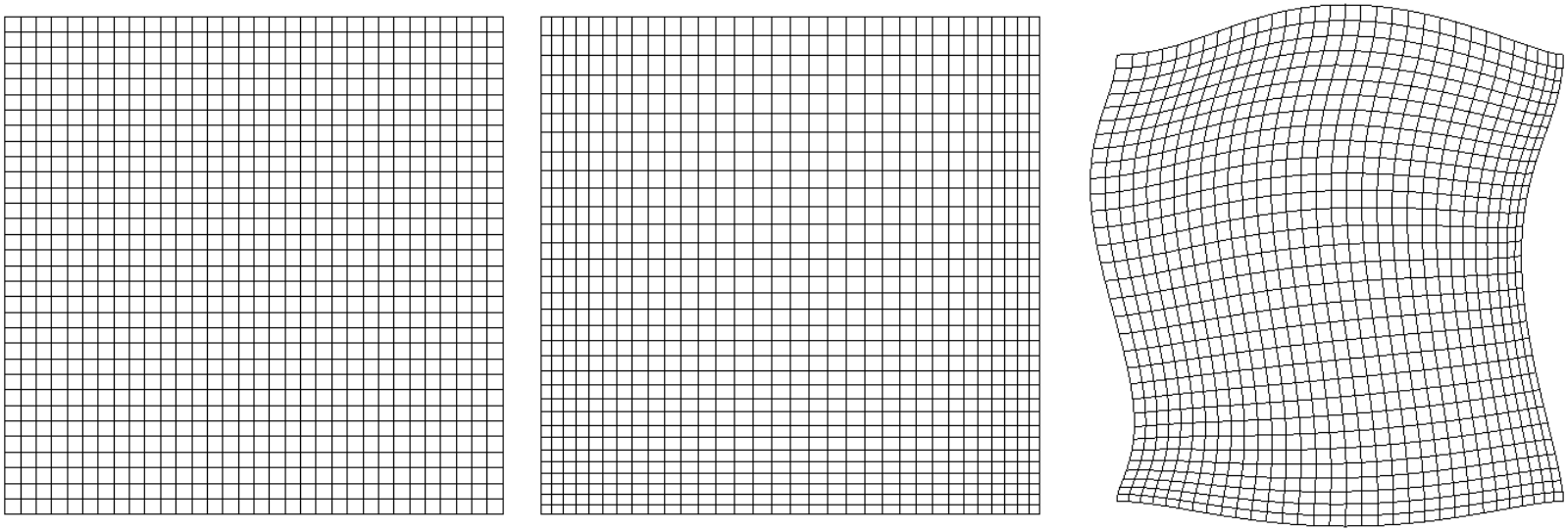
Validation:

- are our models of flow physics correct
- comparison to experimental measurements

Verification for invisicd flows

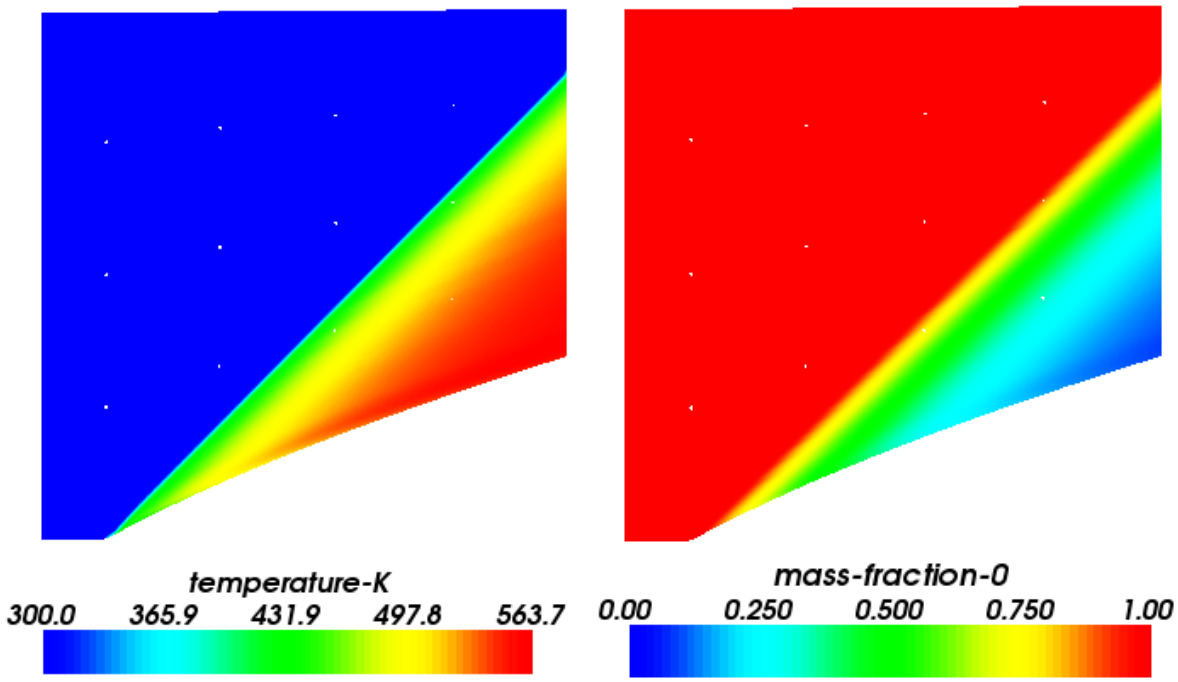
For inviscid flows, we have verified the code using:

- Roy's manufactured solution for supersonic invisicd flow. Demonstrated 2nd order spatial accuracy on regular, stretched, and distorted grids.
- Power & Stewarts's exact solution for an oblique detonation wave. Demonstrated 1st order spatial accuracy in the presence of shocks.



Verification: oblique detonation wave

The flow problem is an oblique detonation wave which is supported by a curved wedge surface. The analytical solution for this problem was first presented by Powers & Stewart (AIAA J. 1992), and first employed for verification by Powers & Aslam (AIAA J. 2006).



In order to make the problem analytically tractable, the reaction mechanism for the detonation is simplified. The reaction is a one-step reaction that proceeds once an ignition temperature is reached.

The Powers and Aslam gas model

- Two species A, B, with reaction of A to B proceeding at rate

$$\frac{d\rho_B}{dt} = \alpha\, \rho_A\, H(T - T_i)$$

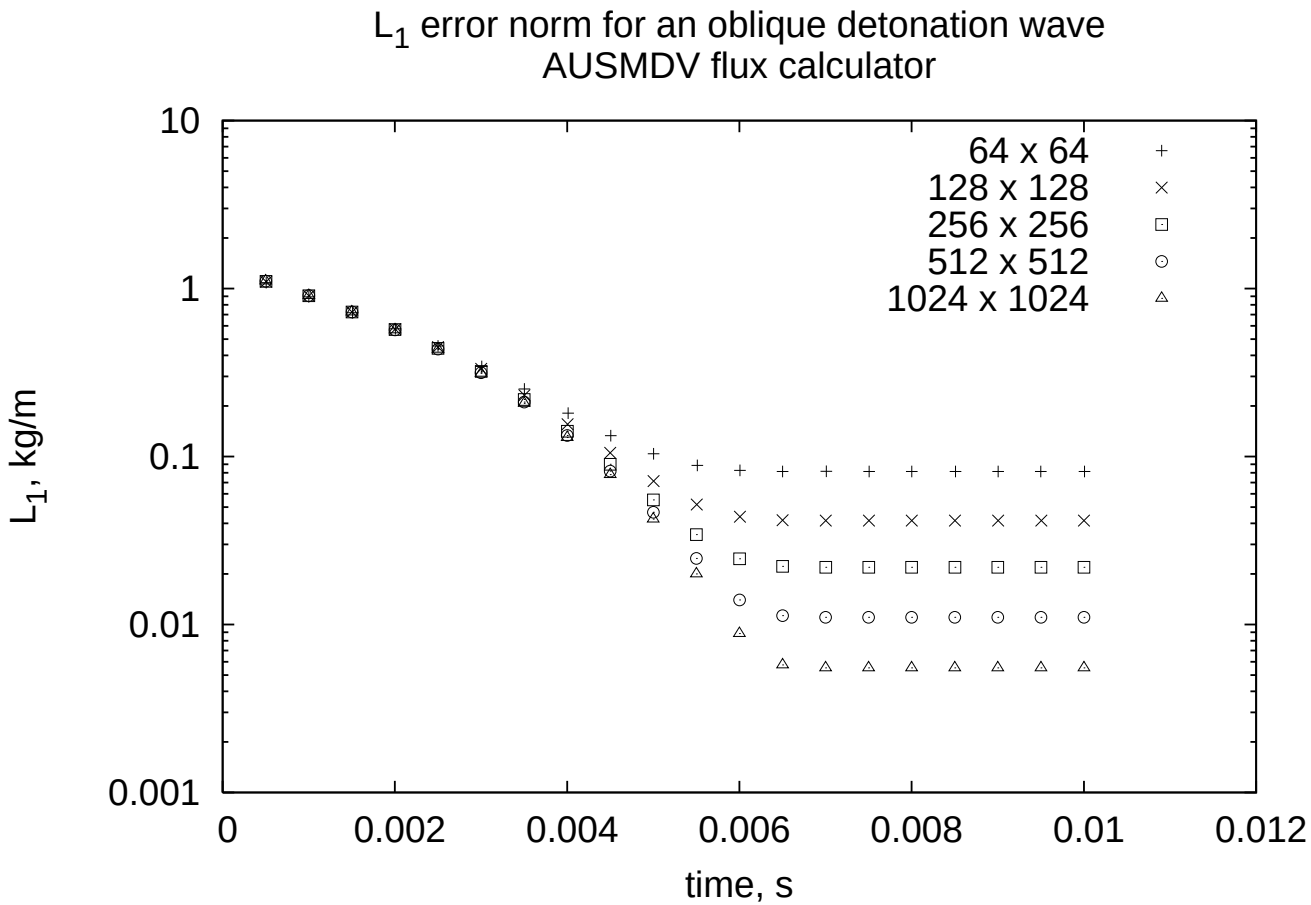
  with rate constant $\alpha = 0.001\,\mathrm{s}^{-1}$
- Reaction progress variable is mass fraction of B: $\lambda = Y_B = \frac{\rho_B}{\rho}$
- $Y_A = 1 - Y_B$
- Equation of state for internal energy:

$$u = \frac{1}{\gamma - 1}\frac{p}{\rho} - \lambda q = C_v T - Y_B q$$

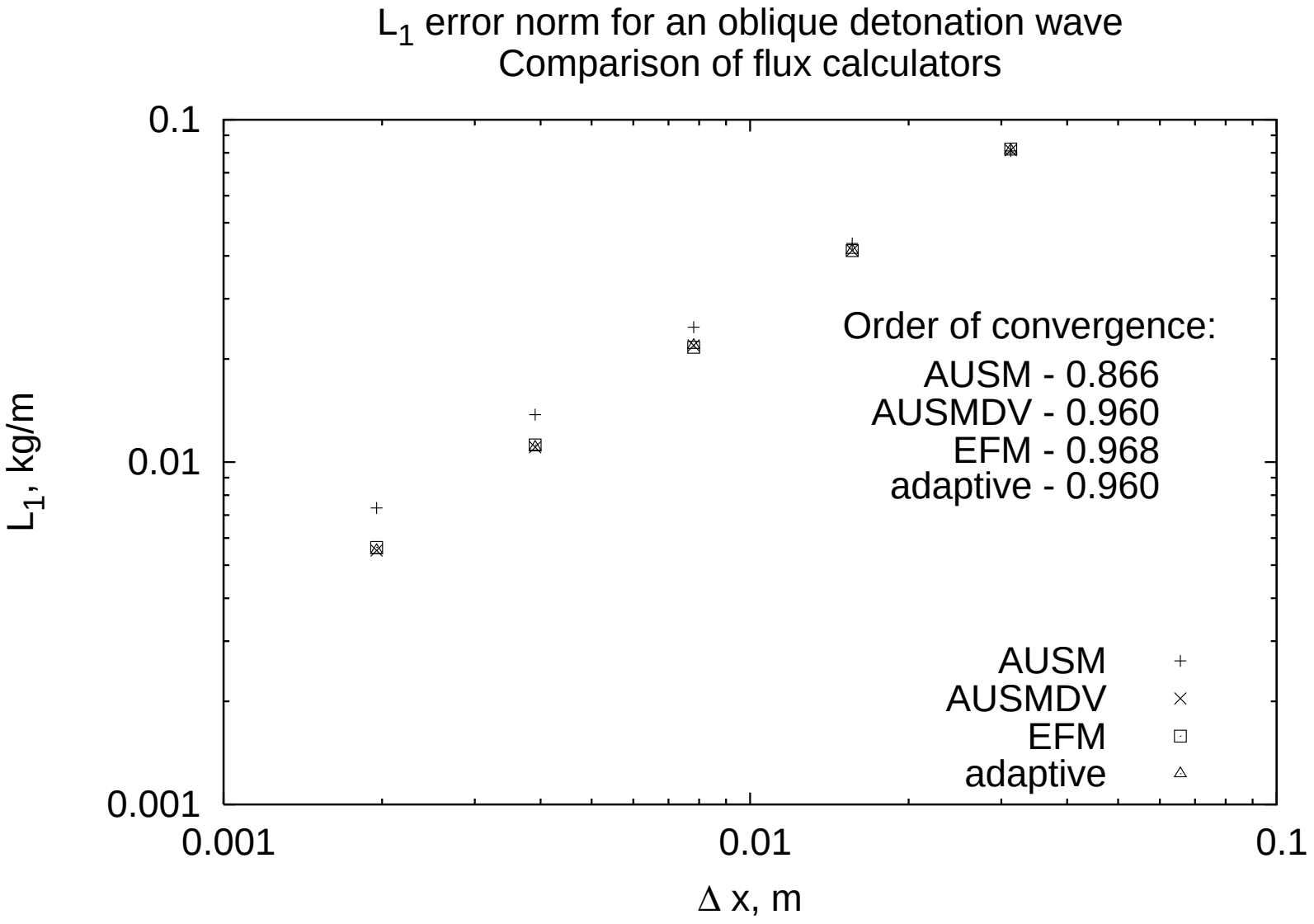
  with heat of reaction $q = 300000\,\mathrm{J/kg}$ and ratio of specific heats $\gamma = 6/5$.
- Pressure: $p = \rho R T$, with gas constant $R = 287\,\mathrm{J/kg.K}$

Oblique detonation wave results – 1/2

$$L_1 = \sum_{i=1}^{N_i} \sum_{j=1}^{N_j} \left| \rho_{n,ij} - \rho_{e,ij} \right| \Delta x \Delta y$$



Oblique detonation wave results – 2/2



Verification for viscous flows

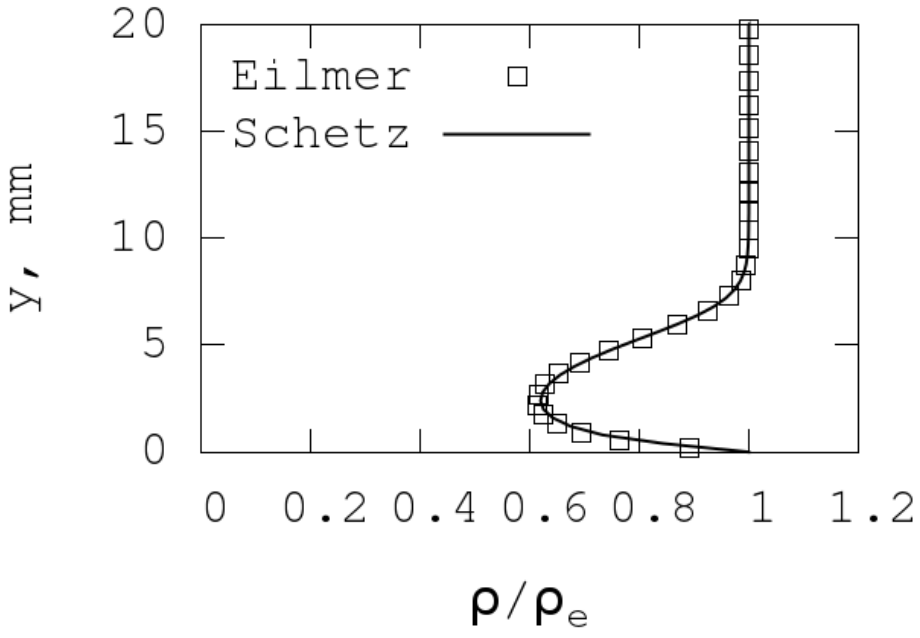
For laminar viscous flows, we have verified the code using:

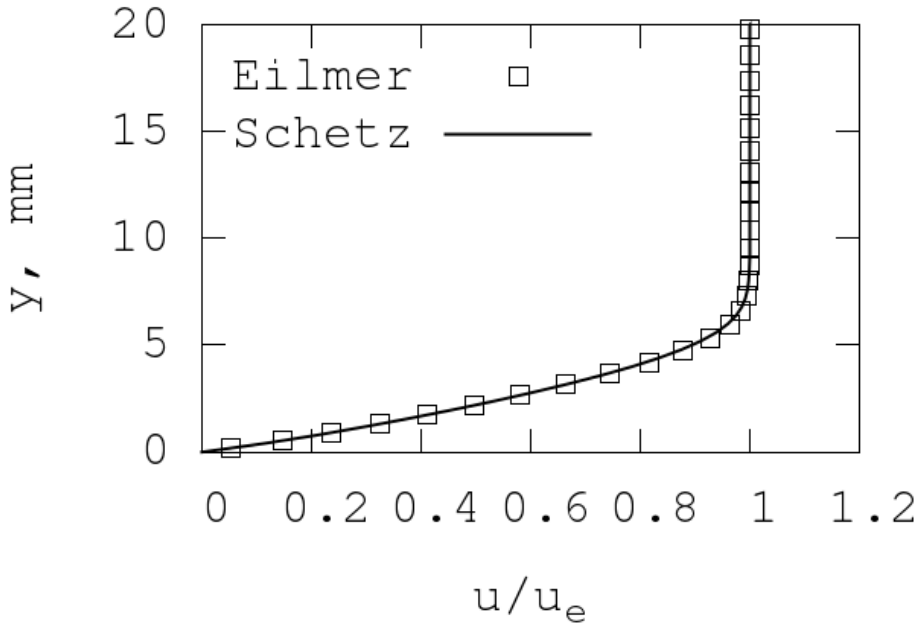
- Roy's manufactured solution for subsonic viscous flow. Demonstrated 2nd order spatial accuracy on regular, stretched, and distorted grids.
- Code-to-code comparison with a boundary layer code to compute the laminar supersonic flow over a flat plate.

For turbulent viscous flows, we have verified the code using:

- Roy's manufactured solution for the $k - \omega$ turbulence model, with modifications to suit the Wilcox 2006 version of the turbulence model.

Verification: laminar flow over a flat plate





$$M_\infty = 4.0; p_\infty = 101.3\,\text{kPa}; T_\infty = 300\,\text{K}$$

Validation

Validation involves comparing the numerical solution to the physical model to experimental measurements. This gives us an indication of how good the Euler and Navier-Stokes flow models are for the problems of interest.

We have performed a good deal of validation of Eilmer over the years and have given a special emphasis on hypersonic flows. We encourage users to validate the flow solver on well-regarded test problems when they are using the flow solver in a new area of investigation.

We will look at two validation cases that deal with ideal gases:

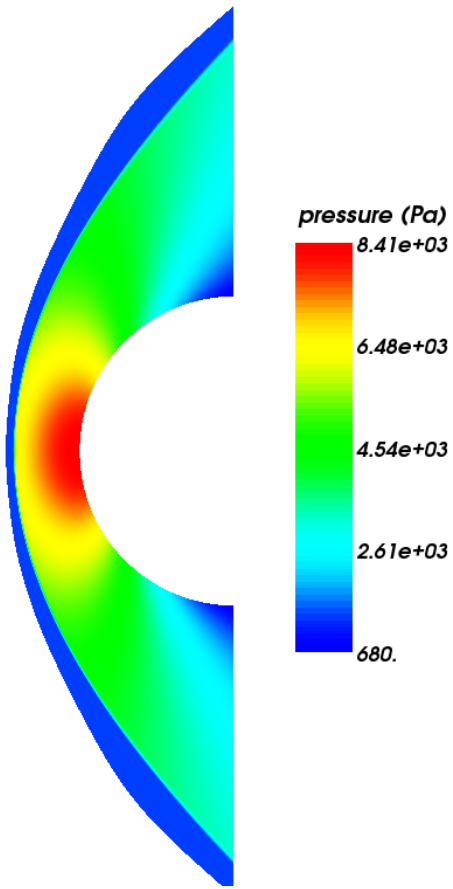
- Flow of Noble gases over spheres
- Heat transfer on a cylinder-flare configuration

Validation: spheres fired into Noble gases – 1/2

Schwartz & Eckerman tested their measurement technique of shock detachment distance by firing ball bearings into noble gases for which a sound theoretical model existed.

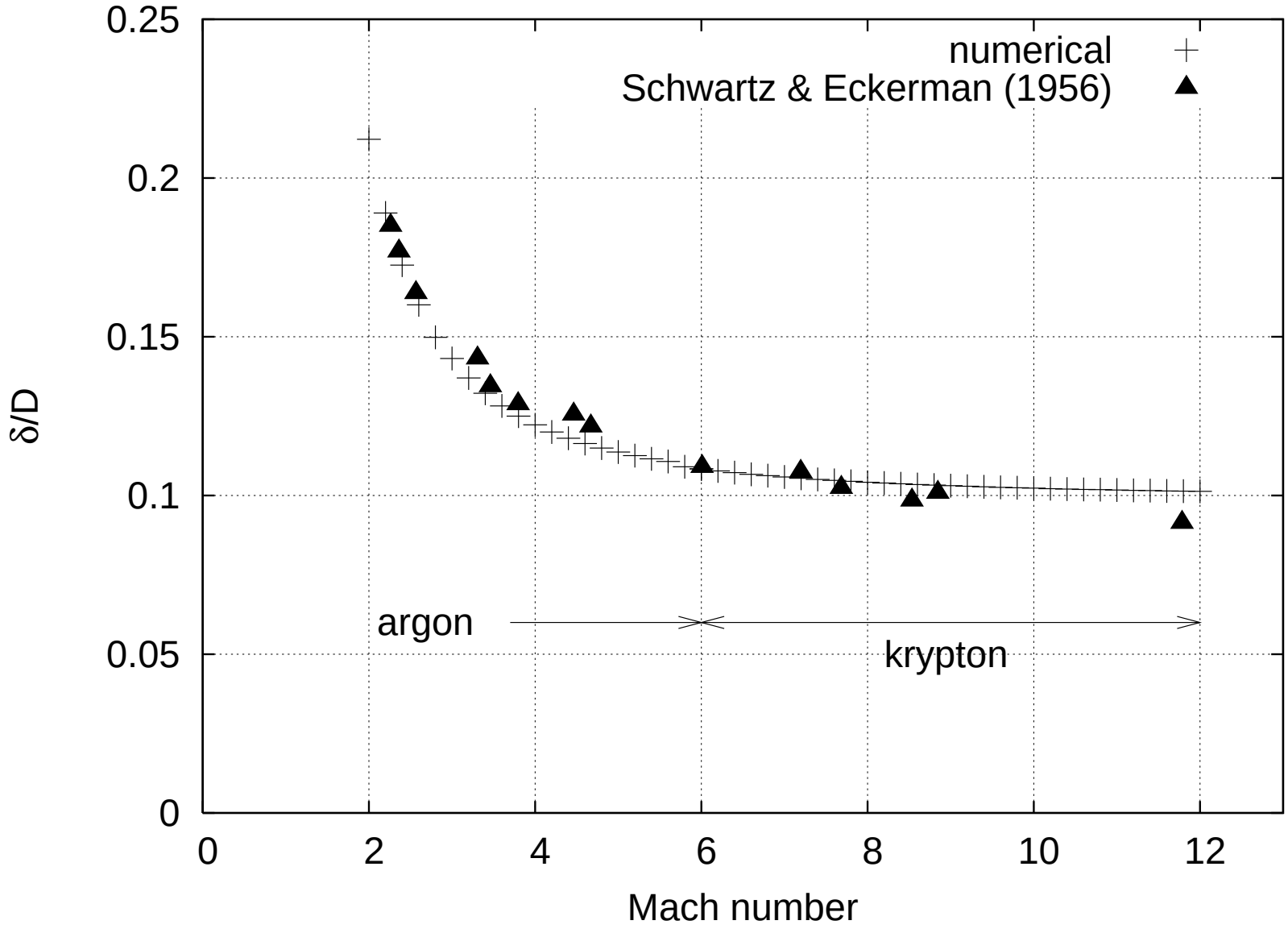
Argon flows	
M_∞	2 – 6
T_∞	300 K
p_∞	10, 100, 200 mmHg

Krypton flows	
M_∞	6 – 12
T_∞	300 K
p_∞	10, 100, 200 mmHg



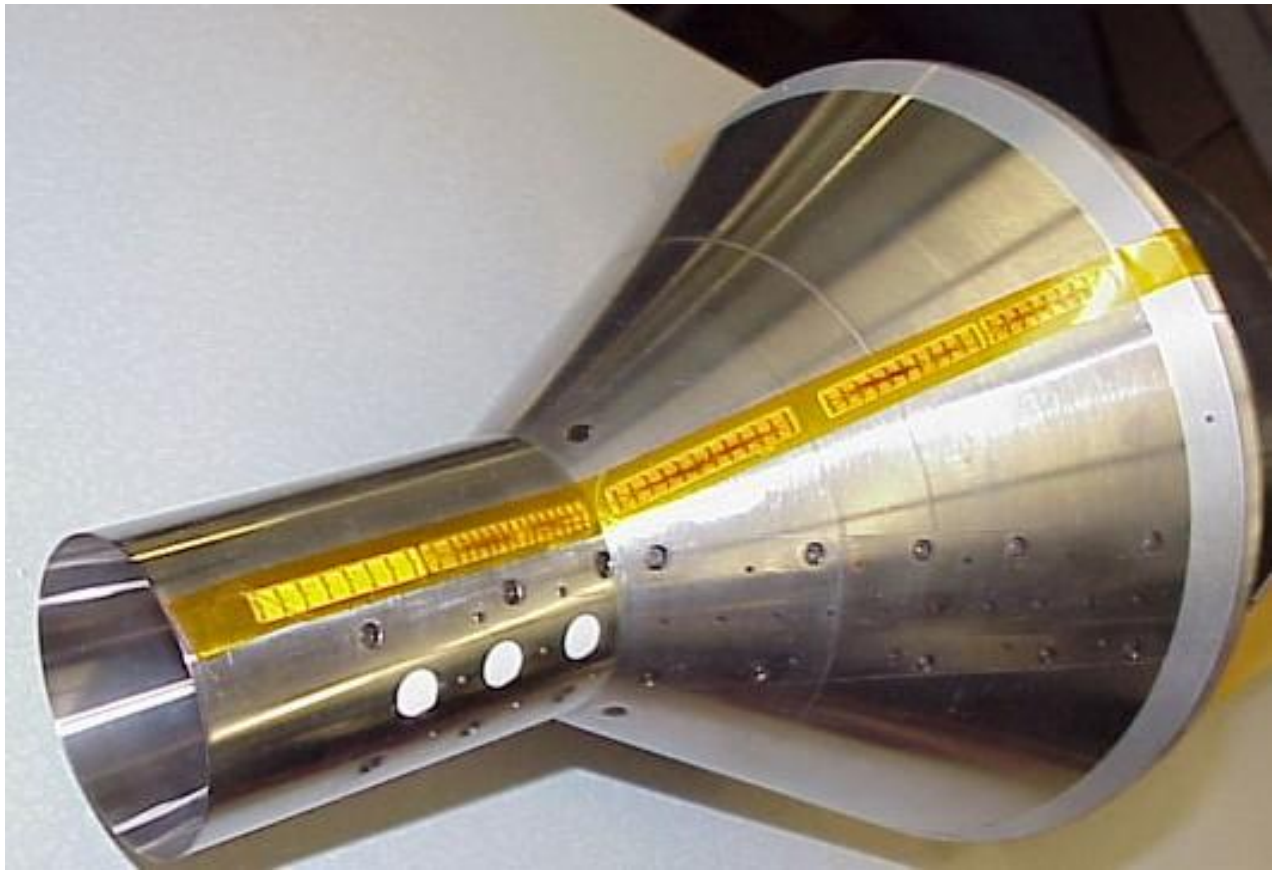
Argon flow, M = 2.0, p = 10.0 mm Hg

Validation: spheres fired into Noble gases – 1/2

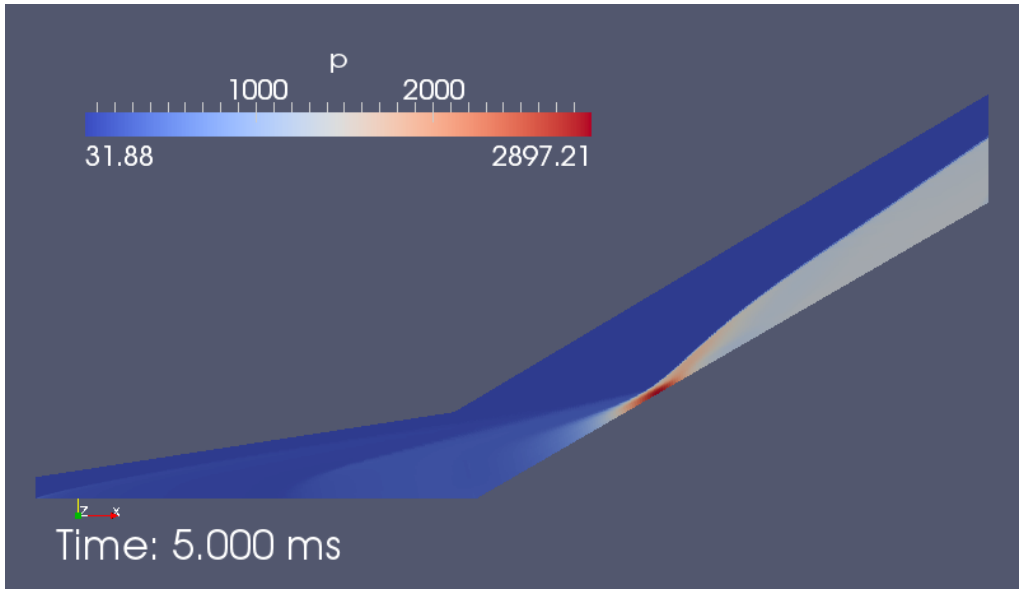


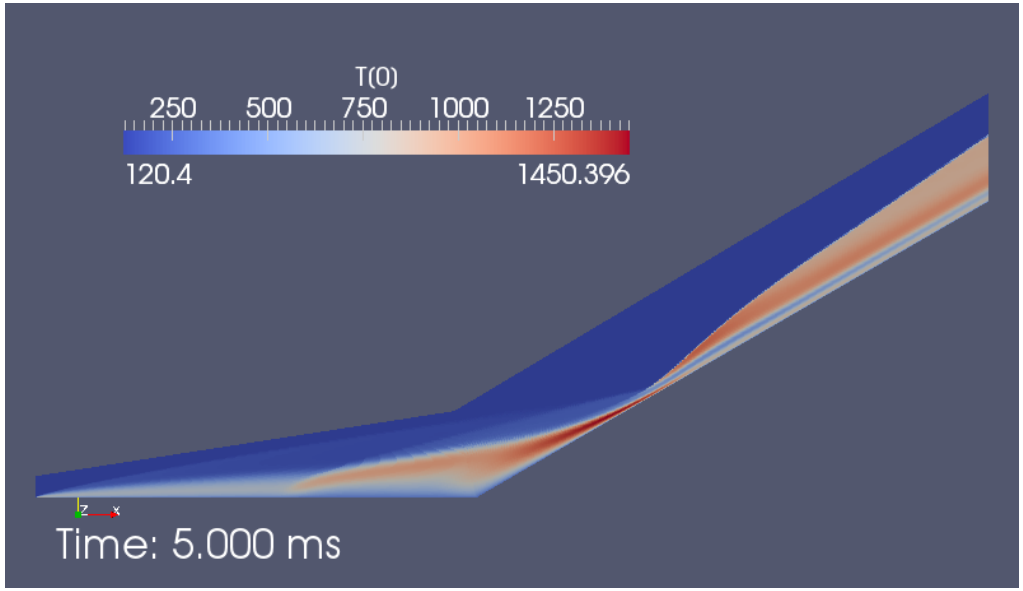
Validation: laminar flow over a cylinder-flare – 1/3

Experiments performed at CUBRC in the LENS Shock Tunnel and reported by Holden et al. This configuration is designed to give rise to a separation bubble.



Validation: laminar flow over a cylinder-flare – 2/3





Validation: laminar flow over a cylinder-flare – 3/3