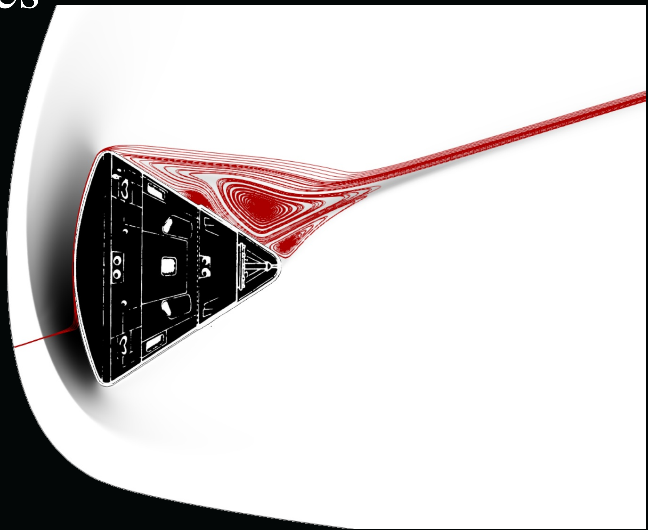
A Tutorial Guide to the Simulation of Hypersonic flows with Eilmer

Rowan Gollan, Kyle Damm, Nicholas Gibbons,
Lachlan Whyborn, Robert Watt and Peter Jacobs
Centre for Hypersonics @ The University of Queensland

29 March 2024
STO-AVT-358 Lecture Series and Workshop
at von Karman Institute for Fluid Dynamics

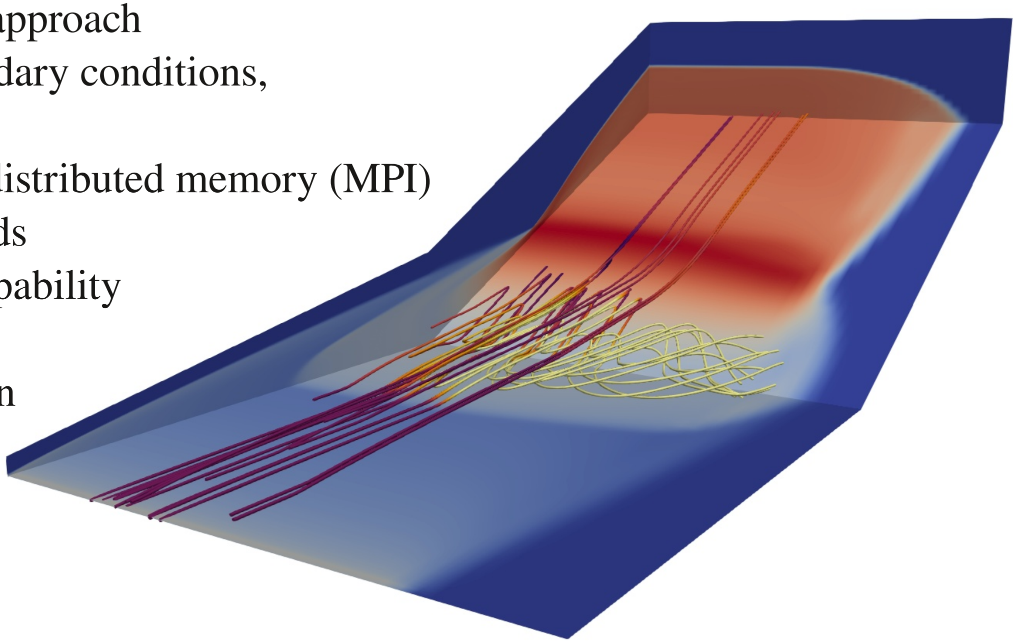


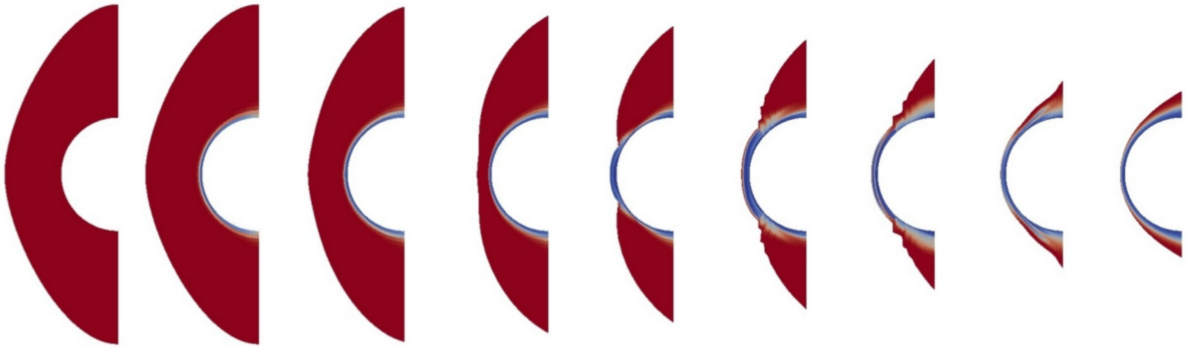
Workshop outline

1. Introduction to Eilmer
2. Getting started
3. Examples

Eilmer: features and capabilities

+ 2D/3D compressible flow simulation
+ cell-centred finite-volume, 2nd-order but with shock-capturing
+ Gas models include ideal, thermally perfect, multi-temperature and state-specific
+ Finite-rate chemistry and generalised thermal nonequilibrium modelling
+ Inviscid, laminar, turbulent flows
+ Solid domains with conjugate heat transfer
+ User-controlled moving grid capability
+ Boundary shock-fitting method for blunt body shock layers
+ Transient, time-accurate updates with Runge-Kutta family integrators
+ Steady-state accelerator using Newton-Krylov approach
+ User-defined customisations available for boundary conditions, source terms, and pre- and post-processing
+ Parallel computation using shared memory or distributed memory (MPI)
+ Multiple-block structured and unstructured grids
+ Native grid generation and 3rd-party import capability
+ Unstructured-mesh partitioning via Metis
+ Adjoint solver for efficient sensitivies evaluation

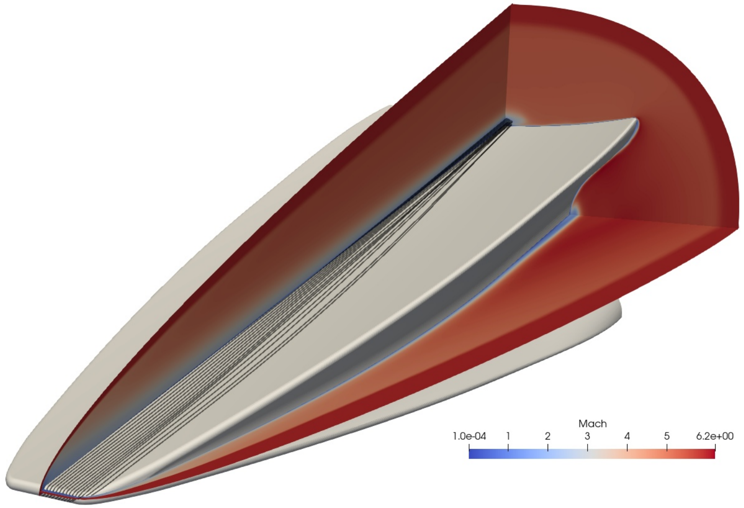


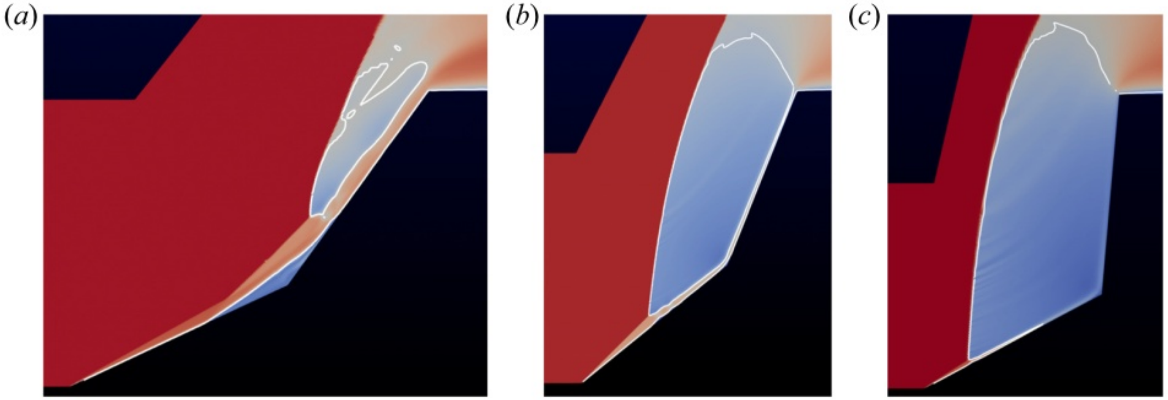


Eilmer: UQ's in-house hypersonic flow solver

THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

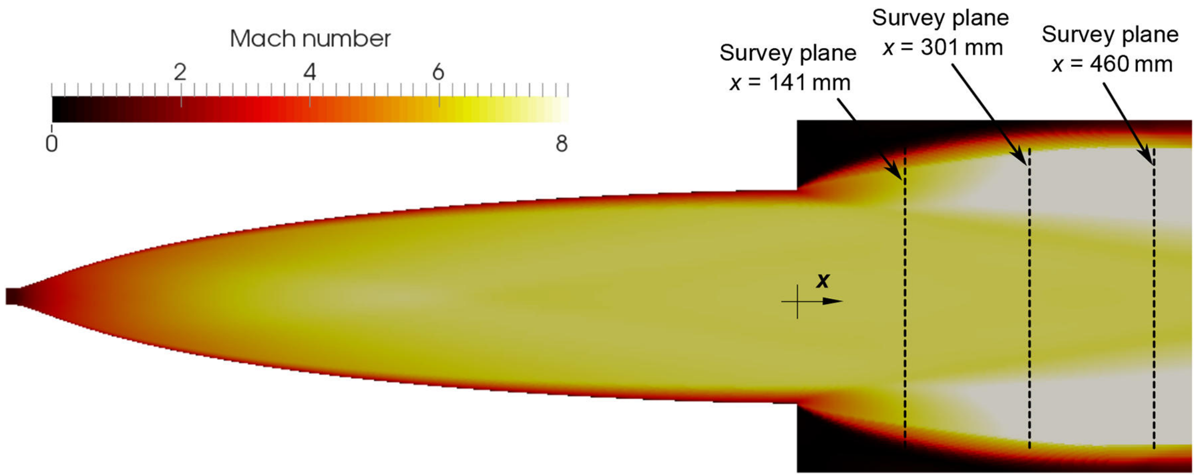
Experiment support
+ analysis of facility operation
+ analysis of test articles





Flow physics investigation
+ as a wind tunnel substitute

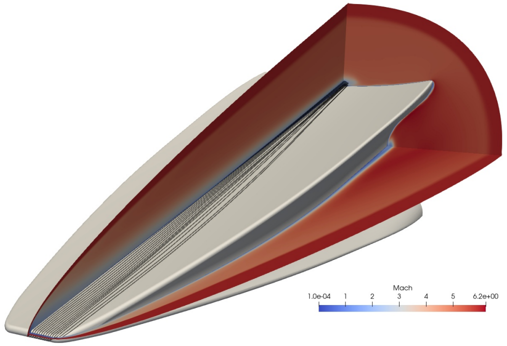
Flowpath design and optimisation
+ impulse facility nozzles
+ aerodynamic shape optimization



Gibbons, Damm, Jacobs & Gollan (2023) Computer Physics Communications, v 282, 108551

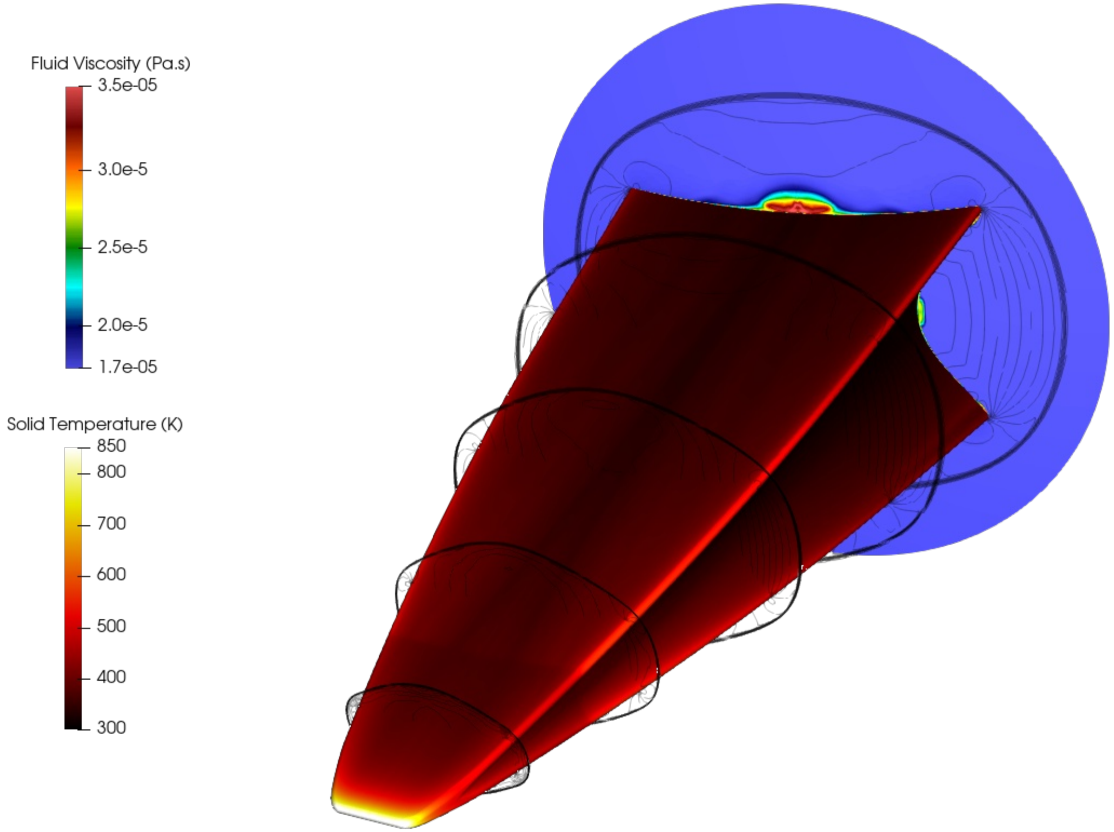
`Eilmer` for use in support of experiments

THE UNIVERSITY OF QUEENSLAND
AUSTRALIA



BoLT-II flight experiment (with ground testing)

+ simulations of flow field to help with sensor placement
+ simulations of coupled heat transfer in fluid/solid domains to inform hot-wall testing

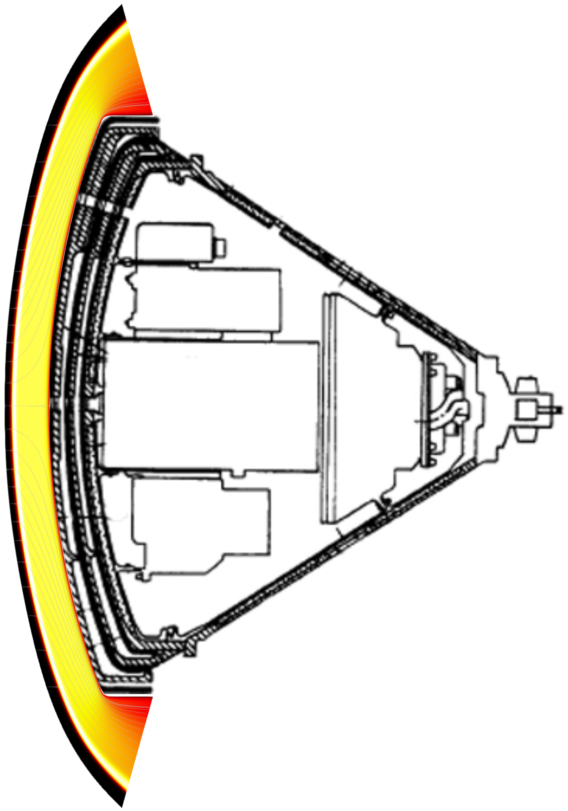


Damm, Gollan & Veeraragavan (2023) AIAA SciTech Paper 2023-0292

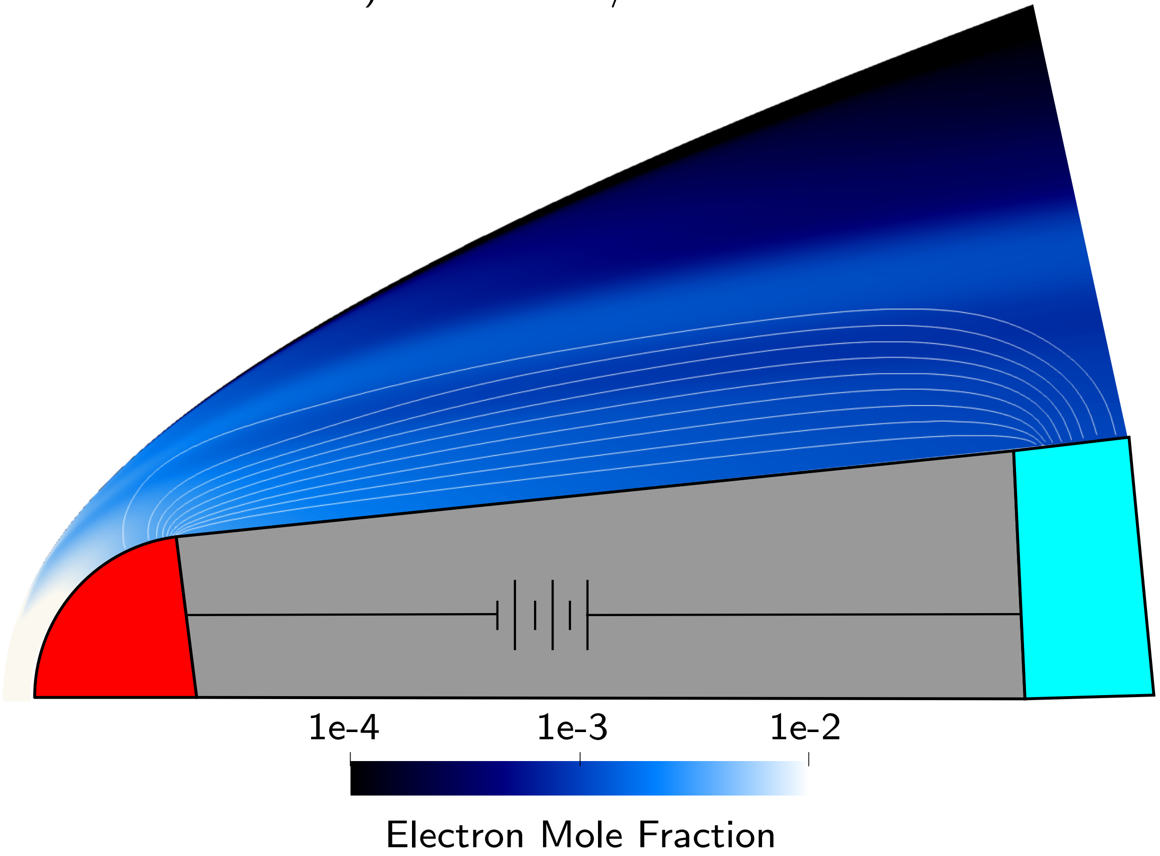
`Eilmer` for novel aero tech evaluation

- Eilmer is an Engineering-style flow simulation code
- As accurate as possible but still tractable for real-scale problems
- How to simulate ETC in Eilmer?

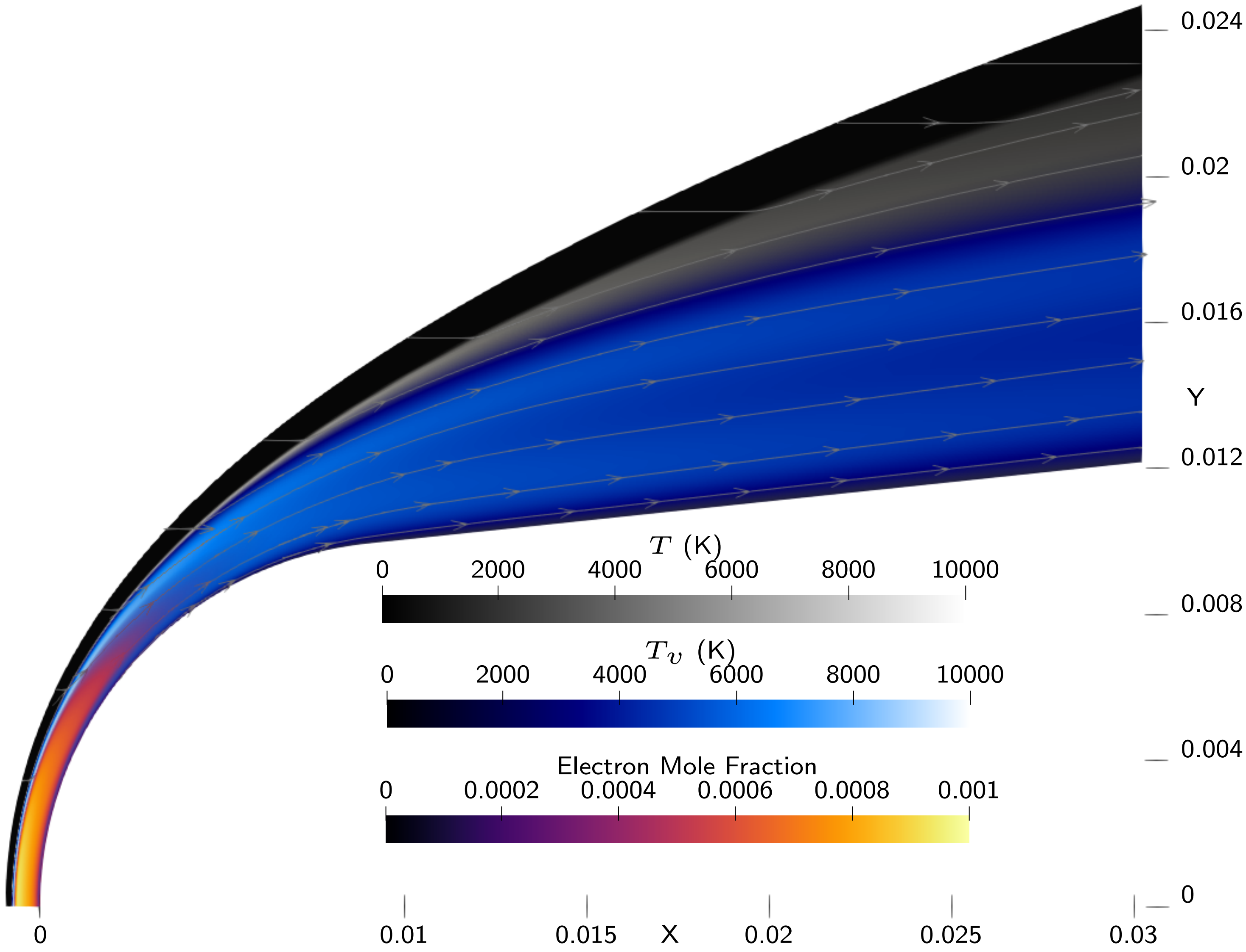
1.) High-temperature gas dynamics



2.) Electrostatic/fluid interactions

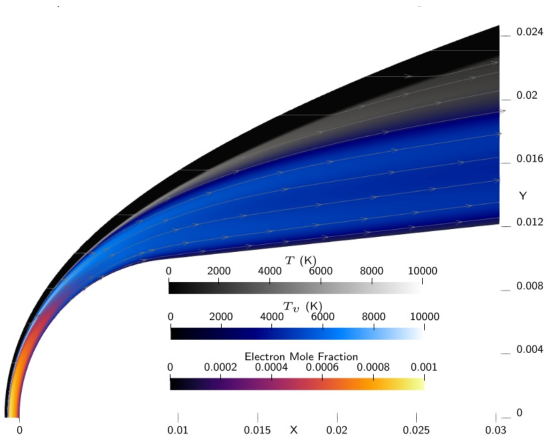


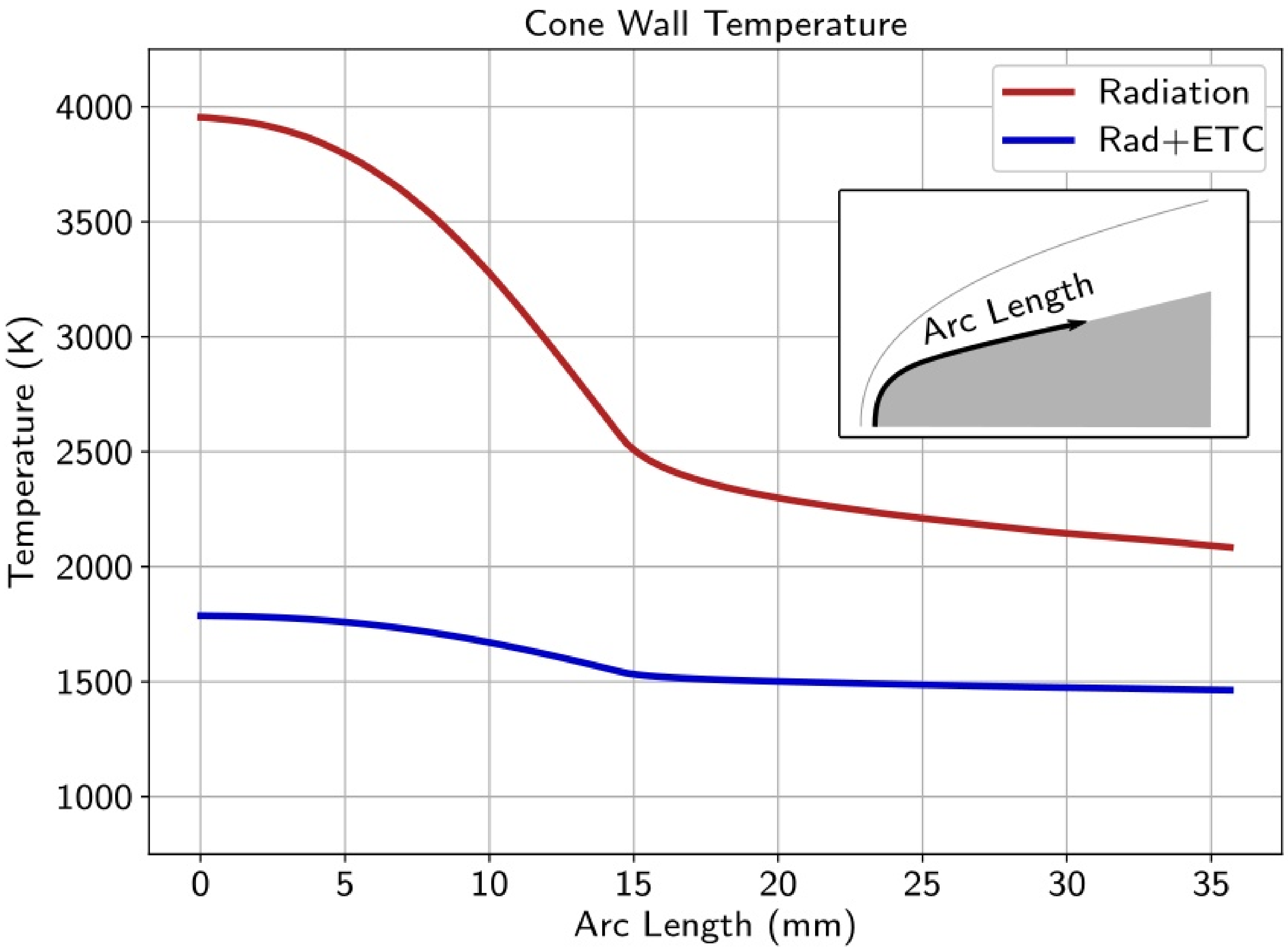
- Solution at v=6 km/s, h=35 km, on 84x120 cell nominal grid



Eilmer for novel aero tech evaluation

- Computed wall temperatures for v=6 km/s, h=35 km case



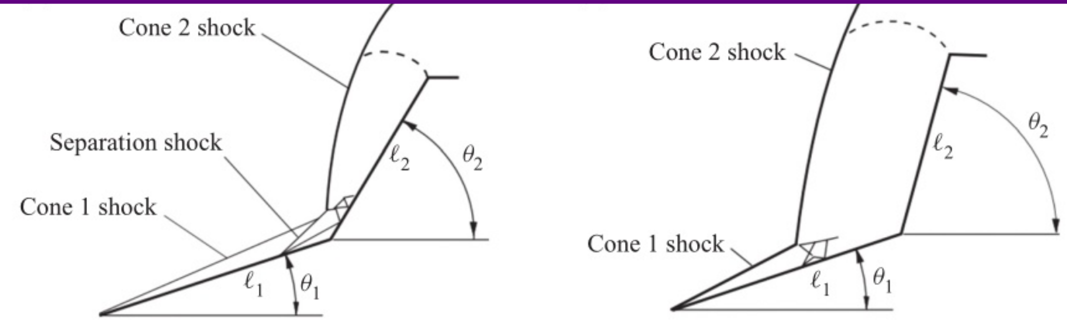


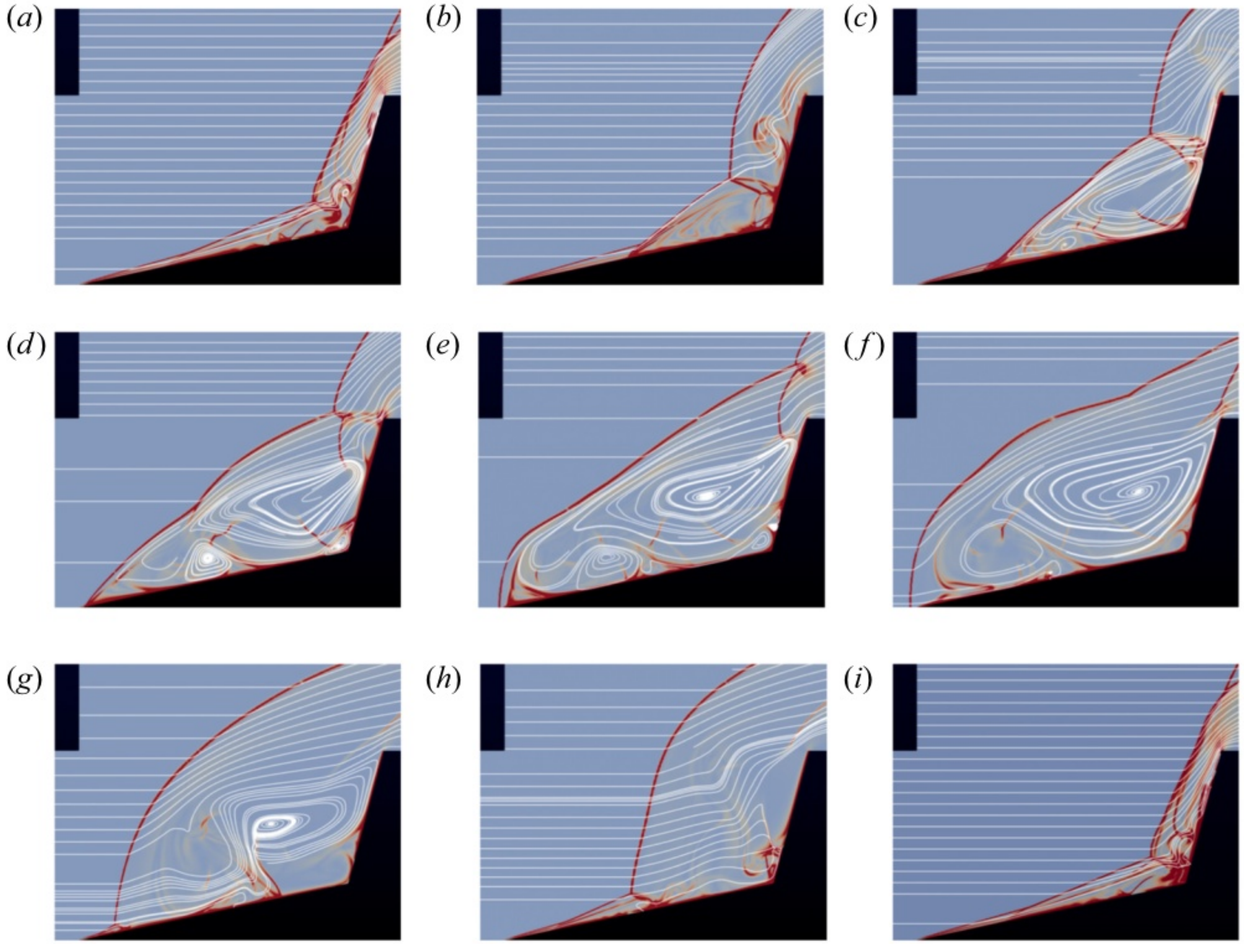
`Eilmer` for flow physics investigation

THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

Steady and unsteady flow over double-cones

+ 300+ simulations to sweep geometric parameter space and flow conditions
+ aim to determine unsteadiness boundaries





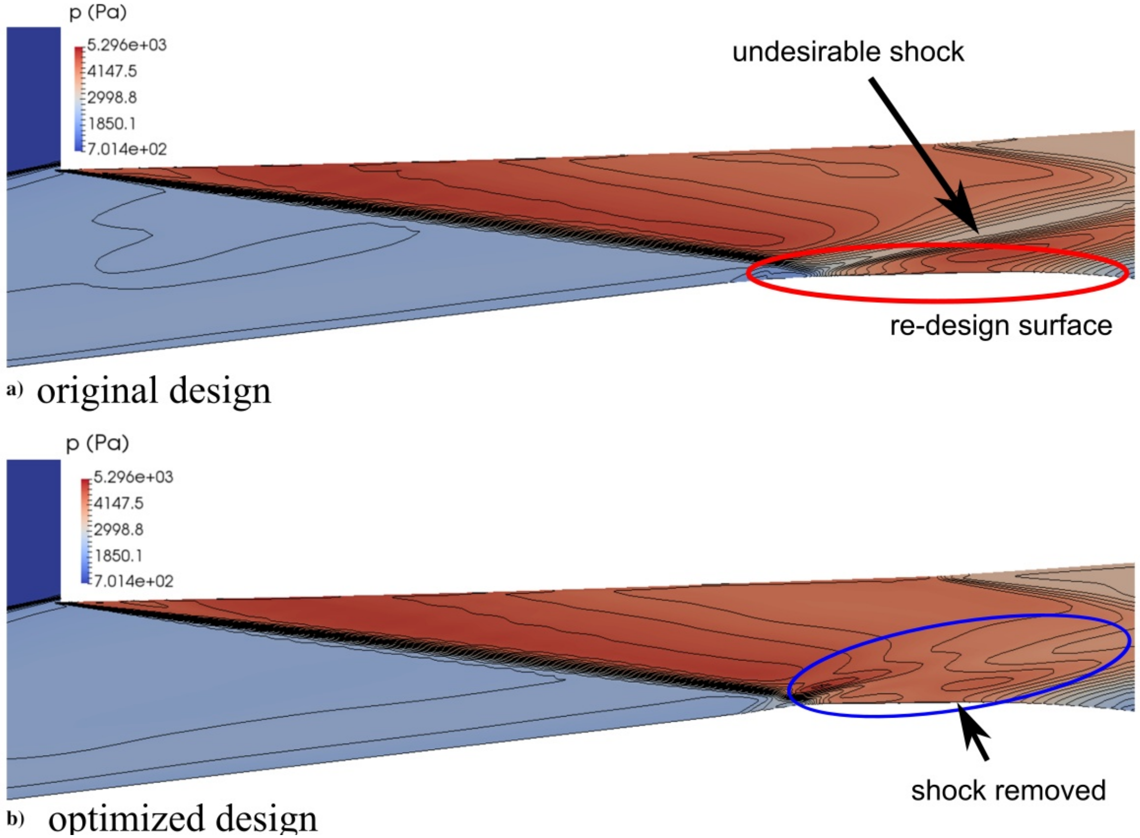
Hornung, Gollan & Jacobs (2021) Journal of Fluid Mechanics, v.916

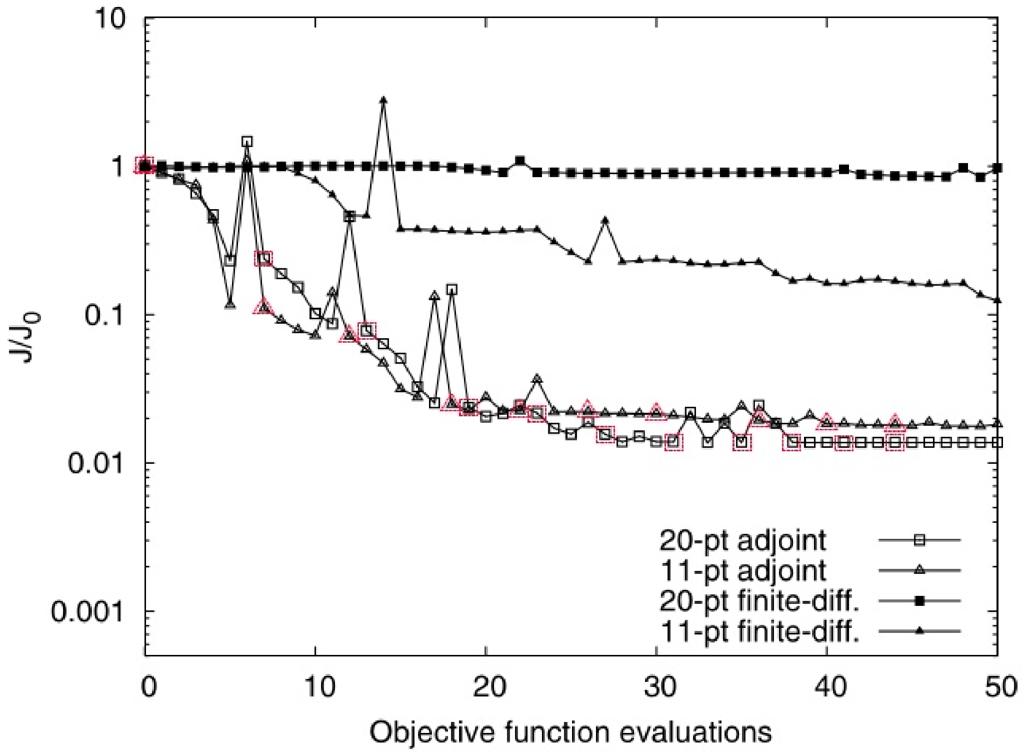
Eilmer for flowpath design

THE UNIVERSITY OF QUEENSLAND AUSTRALIA

Efficient multi-parameter aerodynamic shape optimization

+ in-built state-of-the-art adjoint solver, allowing "open-box" optimization





Optimization target: minimum deviation from design pressure at inlet throat

Damm et al. (2020) AIAA Journal, 58(6)

Eilmer: a brief history

- cns4u began life in 1990 (Peter Jacobs, ICASE NASA Langley)
- mb_cns started in mid 1990s (Peter Jacobs + grad students at UQ)
- mbcns2 in early 2000s in C++, generalised thermochemistry modelling
- elmer/elmer2 in early 2000s, 3D code in C with Python
- merge mbcns2 and elmer2 --> Eilmer3 in November 2008, C++ core
- Eilmer4 written in D programming language began in 2014
- 2024: release of Eilmer5

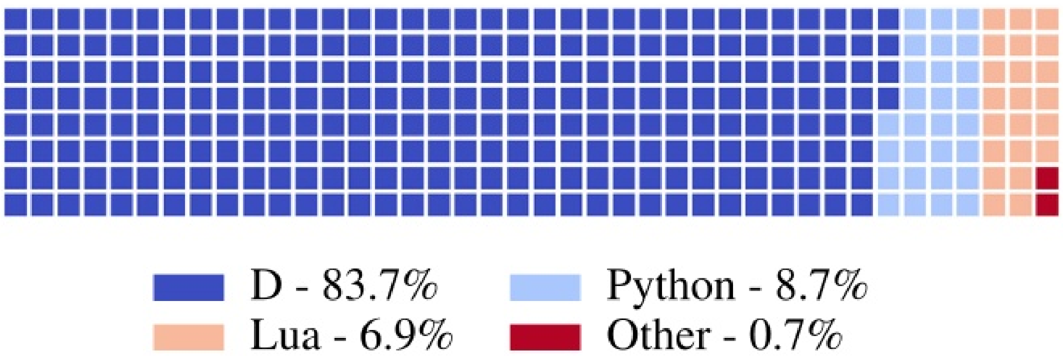
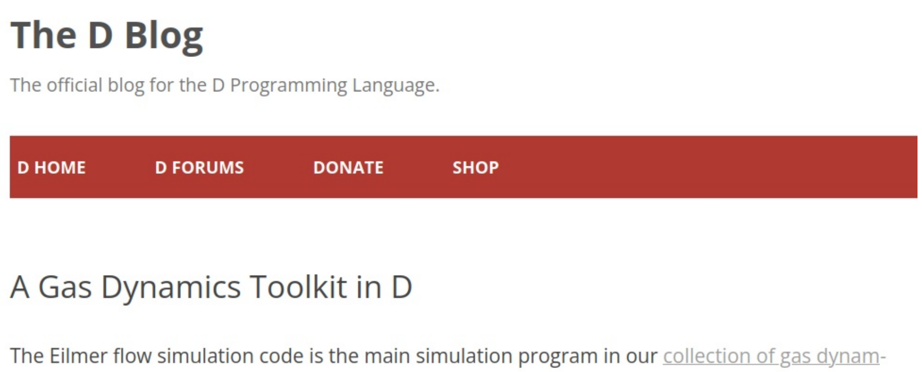


Fig. 4. Division of languages within Eilmer as of April 2022.

Source: *Gibbons et al. (2023), CPC 282:108551*

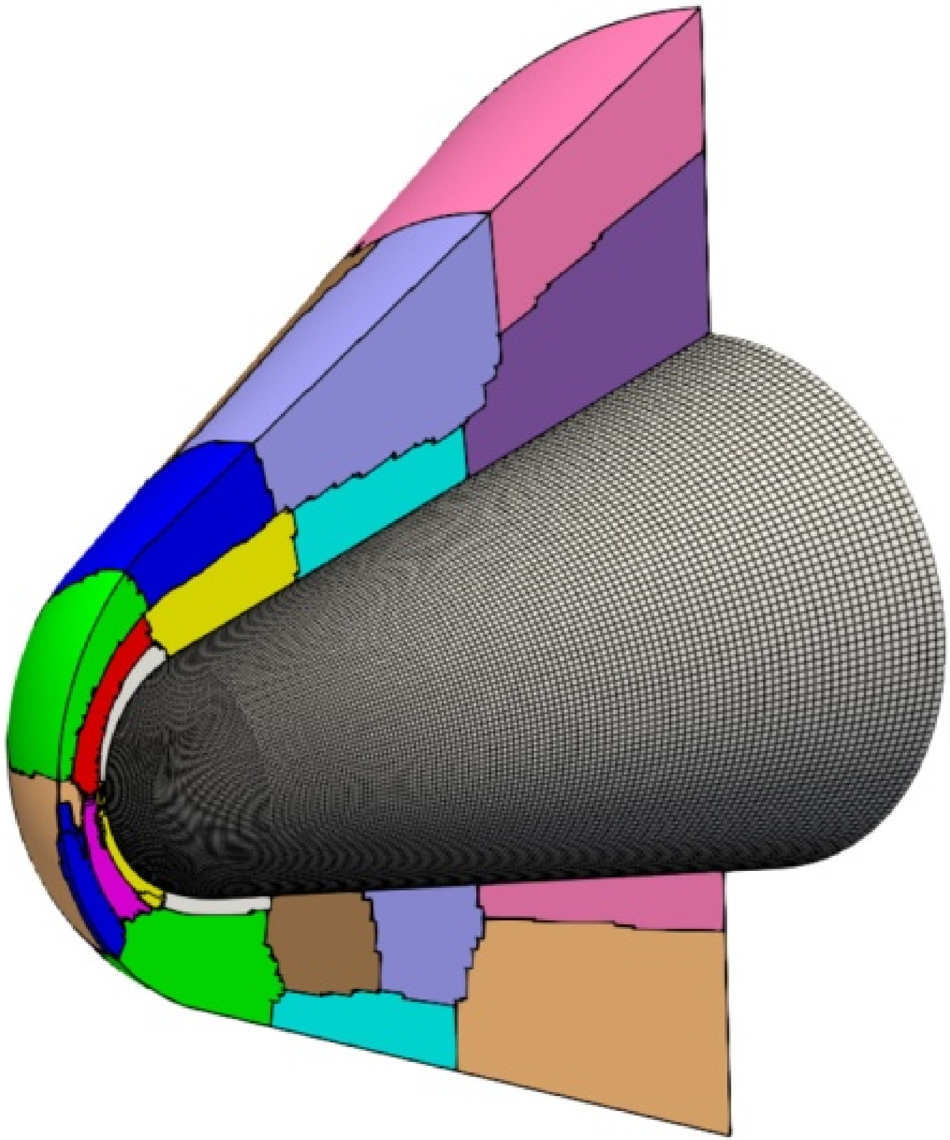


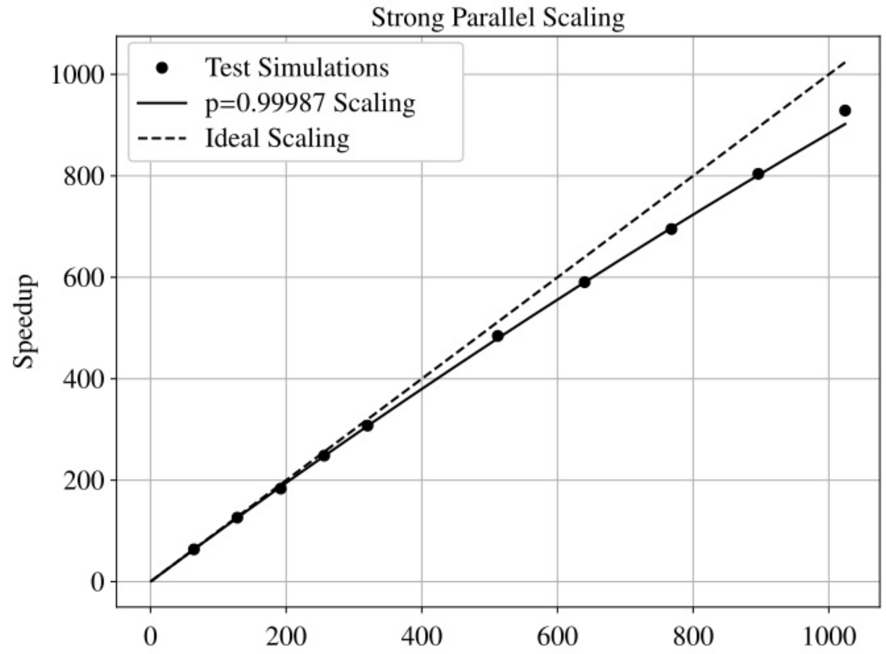
On the choice of D programming language

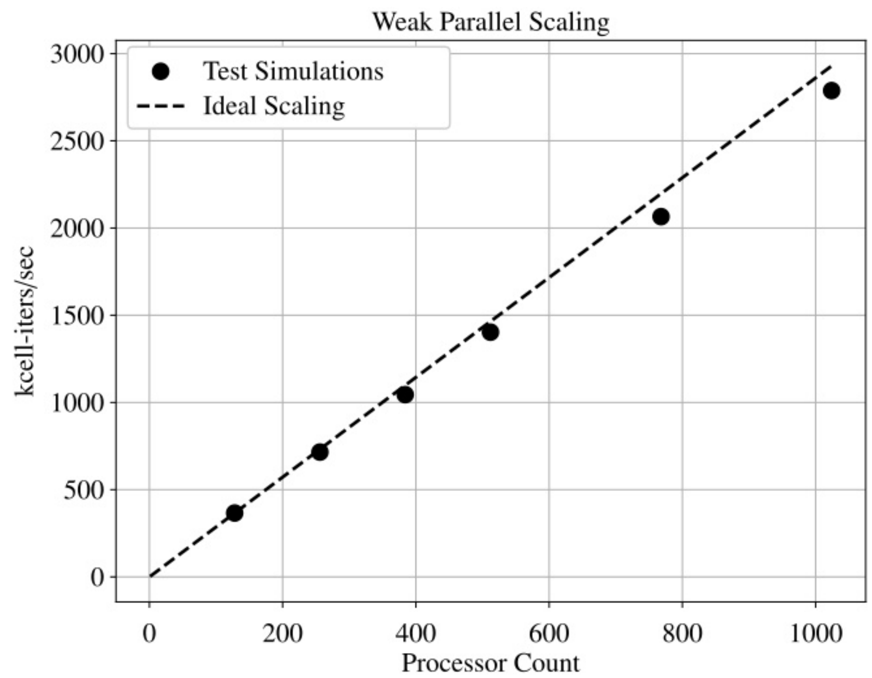
- template programming
- good error messages from compiler
- comprehensive standard library
- simple and direct linkage to C libraries
- fast compilation and good optimizing compilers
- a garbage collector but one that allows control

Eilmer: parallel performance

THE UNIVERSITY OF QUEENSLAND AUSTRALIA



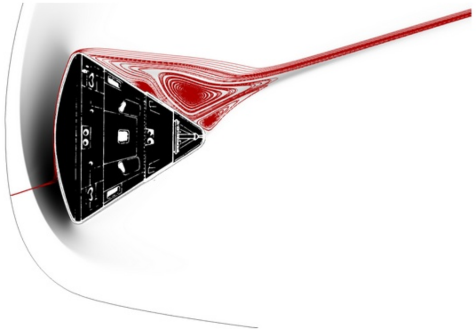




Getting started with Eilmer

Eilmer documentation and more

GDTk Docs Blog

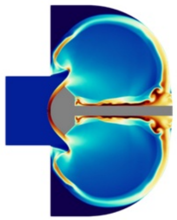


Gas Dynamics Toolkit

GDTk is a collection of software for doing gas dynamics, from simple desktop calculations through to simulations on supercomputers

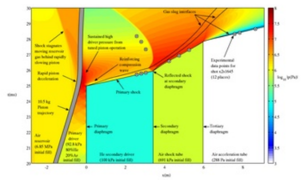
Get started

Open-source GPL3 Licensed. Github repository



Eilmer

2D/3D CFD code for compressible flows.



L1d

Quasi-1D simulator for impulse facilities

Impulse Facility Estimators

State-to-state estimator for flow processes in impulse facilities including: Pitot, ESTCN, and NENZF1d.

Documentation

Head here for the project docs!

https://gdtk.uqcloud.net

Preparing your compute environment

1. Install the LLVM D compiler (latest available)
2. Install prerequisite packages

Software	linux, Debian-family	linux, RedHat-family	macOS
build environment	`build-essential`	"C Development Tools and Libraries"	xcode
LLVM D compiler	— *on linux, download latest is recommended* —		`ldc`
			`dub`
Fortran compiler	`gfortran`	`gcc-gfortran`	`gcc`
	`gfortran-multilib`		
git	`git`	`git`	`git`
readline	`libreadline-dev`	`readline-devel`	`readline`
ncurses	`libncurses5-dev`	`ncurses-devel`	`ncurses`
OpenMPI	`libopenmpi-dev`	`openmpi-devel`	`open-mpi`
plotutils	`libplot-dev`	`plotutils-devel`	`plotutils`
Paraview	— *on linux, download latest is recommended* —		`paraview` (as cask)
gnuplot	`gnuplot`	`gnuplot`	`gnuplot`
Python	— *on linux, system default is fine* —		`python`
Pandas	`python-pandas`	`python-pandas`	`pandas` (via pip3)
matplotlib	`python-matplotlib`	`python-matplotlib`	`matplotlib` (via pip3)
sed	— *on linux, system default is fine* —		`gnu-sed`

Download, build and install

```
$ cd
$ git clone https://github.com/gdtk-uq/gdtk.git gdtk
$ cd gdtk/src/lmr
$ make install
```

Environment variables

```
export DGD_REPO=${HOME}/gdtk

export DGD=${HOME}/gdtkinst

export PATH=${PATH}:${DGD}/bin:${HOME}/opt/ldc2-1.37.0-linux-x86_64/bin

export DGD_LUA_PATH=${DGD}/lib/?.lua

export DGD_LUA_CPATH=${DGD}/lib/?.so
```

3. Examples

1. **Supersonic inviscid flow over a cone**
2. **Hypersonic viscous flow over convex ramp**
3. **Multi-temperature reacting air flow over a sphere**

Workflow and command-line interface

1. Pre-processing
2. Running a simulation
3. Post-processing

CLI:

```
$ noun action options/arguments
```

```
$ lmr prep-grid

$ lmr prep-grid --job=grid.lua

$ lmr help

$ lmr help -a

$ lmr help prep-grid

$ lmr version
```

Supersonic inviscid flow over a cone

gdtk/examples/lmr/2D/sharp-cone-20-degrees/sg-minimal

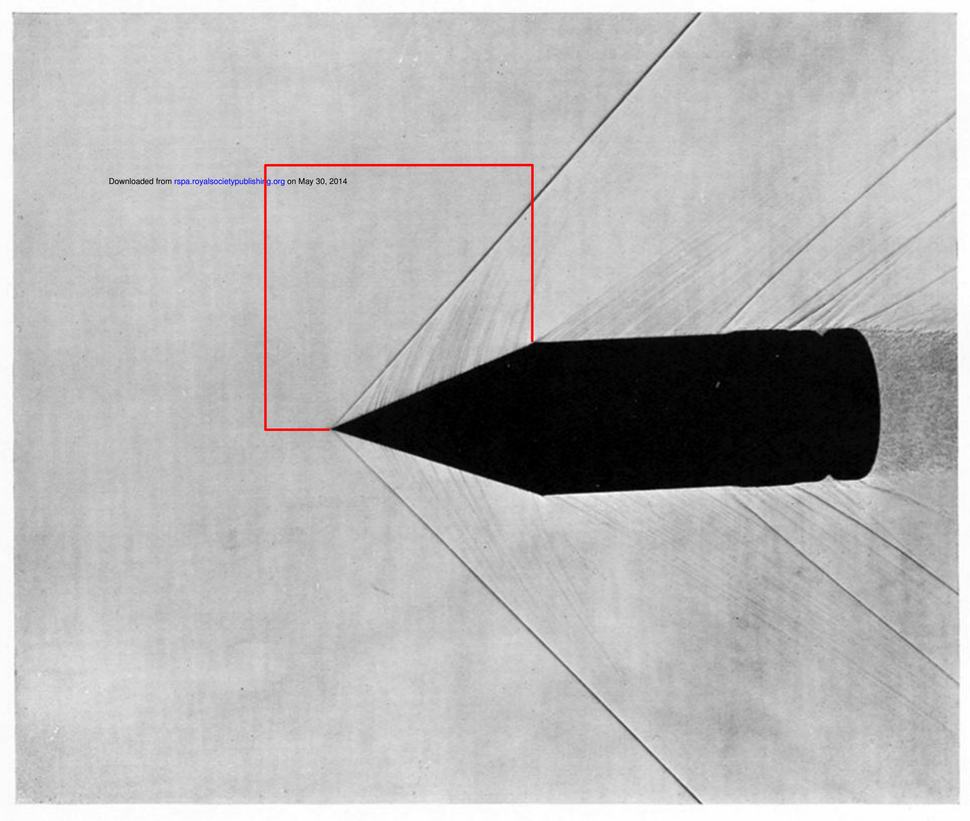


FIG. 3. $U/a = 1{\cdot}576$

Source: Maccoll (1937)

You will learn how to:

- prepare a simple gas model
- generate a two-block domain with structured grids
- configure settings for the transient solver
- post-process to generate VTK files

Supersonic inviscid flow over a cone

gdtk/examples/lmr/2D/sharp-cone-20-degrees/sg-minimal

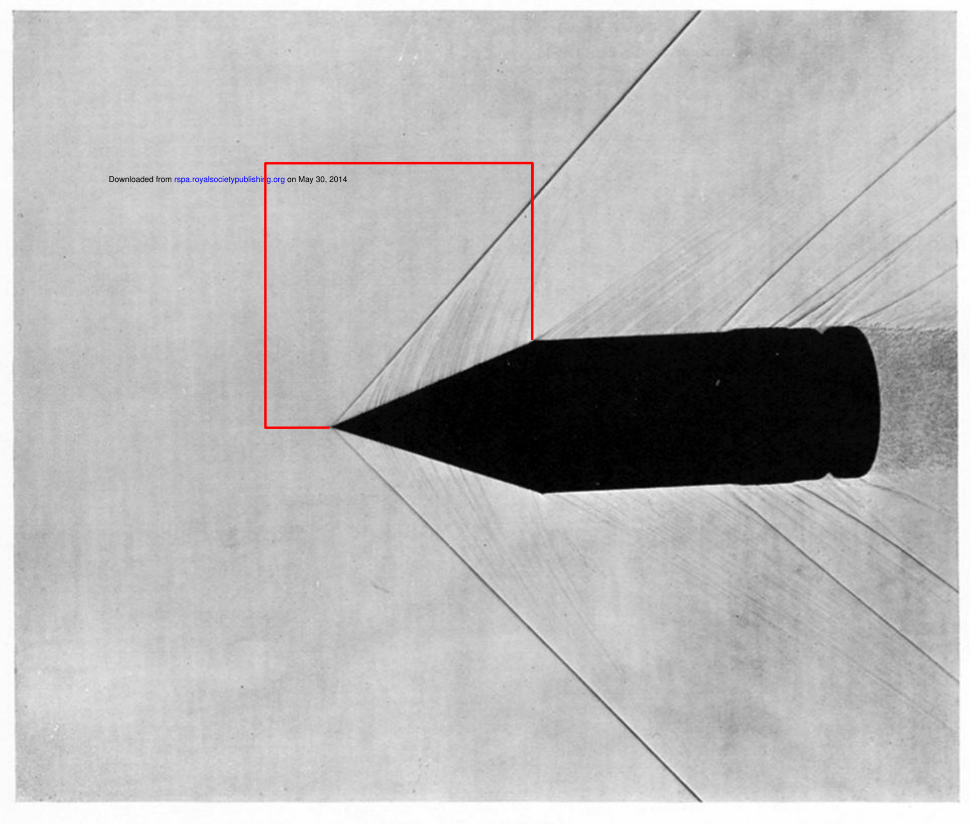
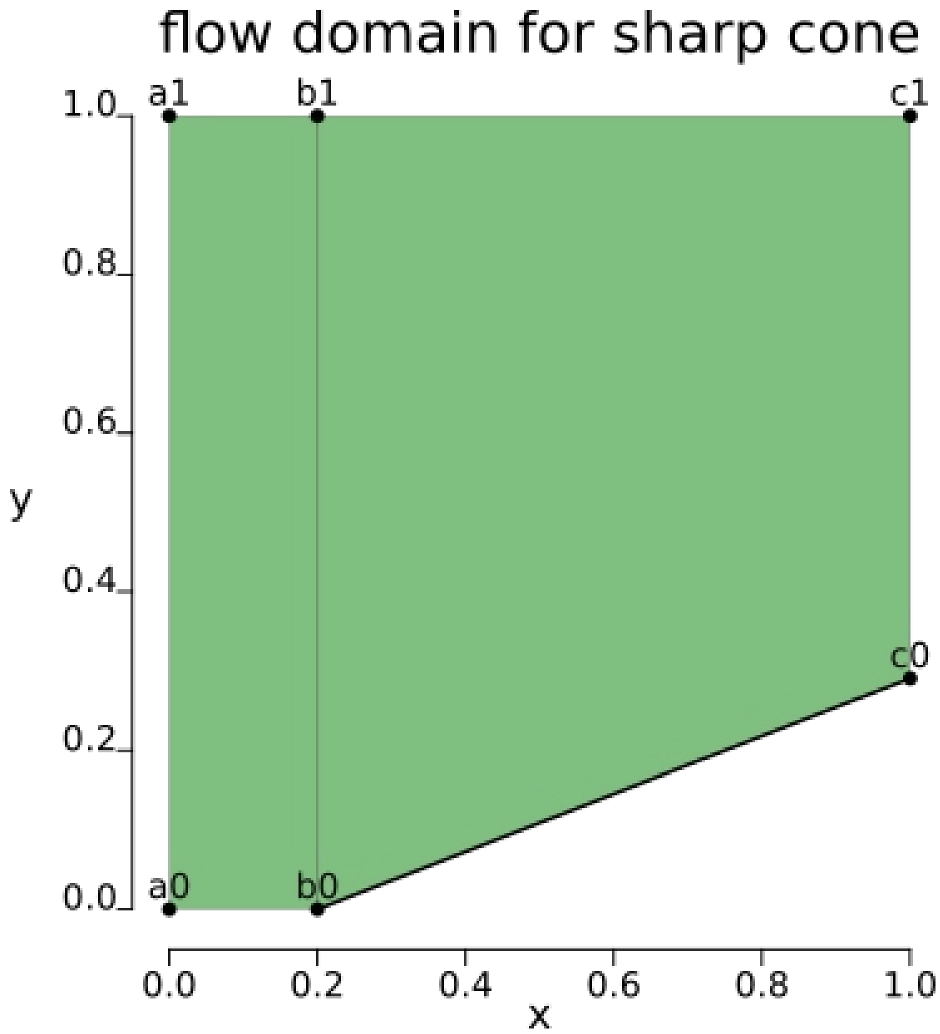


FIG. 3. $U/a = 1{\cdot}576$



Ideal air gas model

gdtk/examples/lmr/2D/sharp-cone-20-degrees/sg-minimal

gas model input file

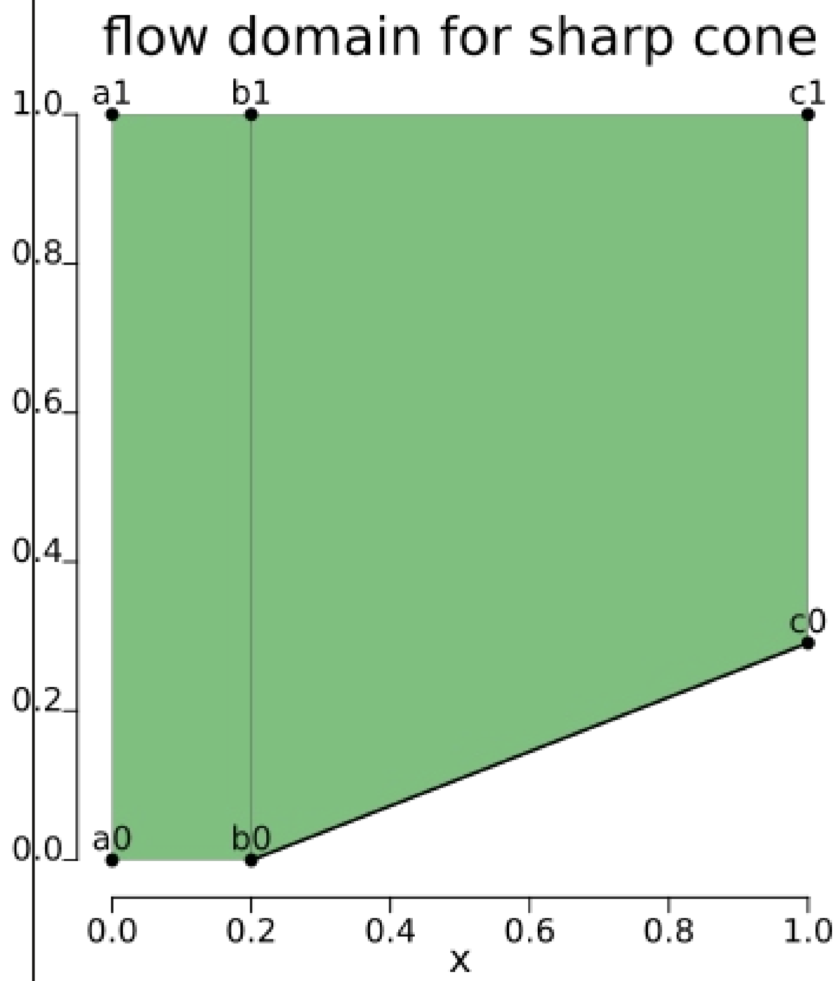
```
model = "IdealGas"
species = {'air'}
```

```
$ lmr prep-gas -i ideal-air.lua -o ideal-air.gas
```

Grid description and generation

gdtk/examples/lmr/2D/sharp-cone-20-degrees/sg-minimal


```
-- grid.lua
print("Set up geometry and grid for a Mach 1.5 flow over a
20 degree cone.")
--
-- 1. Geometry
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}
--
-- 2. Grids
grid0 = registerFluidGrid{
   grid=StructuredGrid:new{psurface=quad0, niv=11, njv=41},
   fsTag="inflow",
   bcTags={west="inflow"}
}
grid1 = registerFluidGrid{
   grid=StructuredGrid:new{psurface=quad1, niv=31, njv=41},
   fsTag="initial",
   bcTags={east="outflow"}
}
identifyGridConnections()
```




```
$ lmr prep-grid --job=grid.lua
```

Simulation description

gdtk/examples/lmr/2D/sharp-cone-20-degrees/sg-minimal


```
-- transient.lua
print("Set up transient solve of Mach 1.5 flow over a 20 degree cone.")
--
-- 0. Assume that a previous processing has step set up the grids.
--
-- 1. Domain type, gas model and flow states
config.solver_mode = "transient"
config.axisymmetric = true
setGasModel('ideal-air.gas')
initial = FlowState:new{p=5955.0, T=304.0} -- Pa, degrees K
inflow = FlowState:new{p=95.84e3, T=1103.0, velx=1000.0}
flowDict = {initial=initial, inflow=inflow}
--
-- 2. Fluid blocks, with initial flow states and boundary conditions.
-- Block boundaries that are not otherwise assigned a boundary condition
-- are initialized as WallBC_WithSlip.
bcDict = {
   inflow=InFlowBC_Supersonic:new{flowState=inflow},
   outflow=OutFlowBC_Simple:new{}
}
--
makeFluidBlocks(bcDict, flowDict)
--
-- 3. Simulation parameters.
config.max_time = 5.0e-3  -- seconds
```

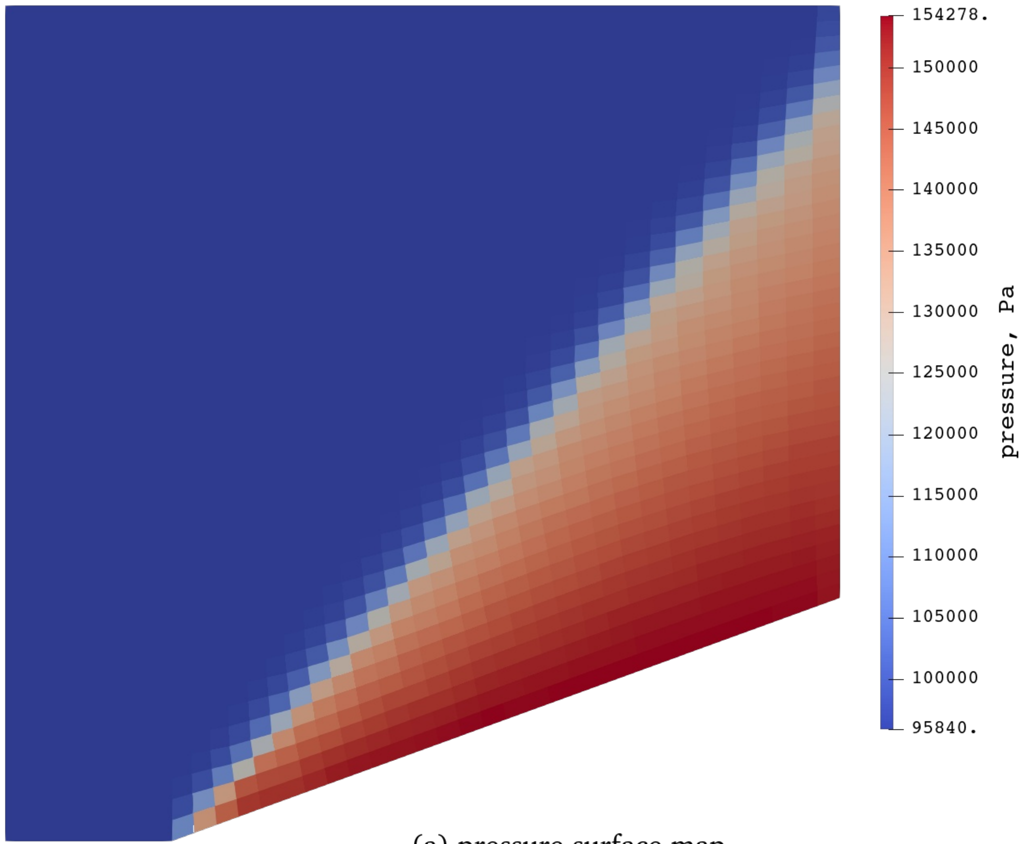
```
$ lmr prep-sim --job=transient.lua
```

Ready to run

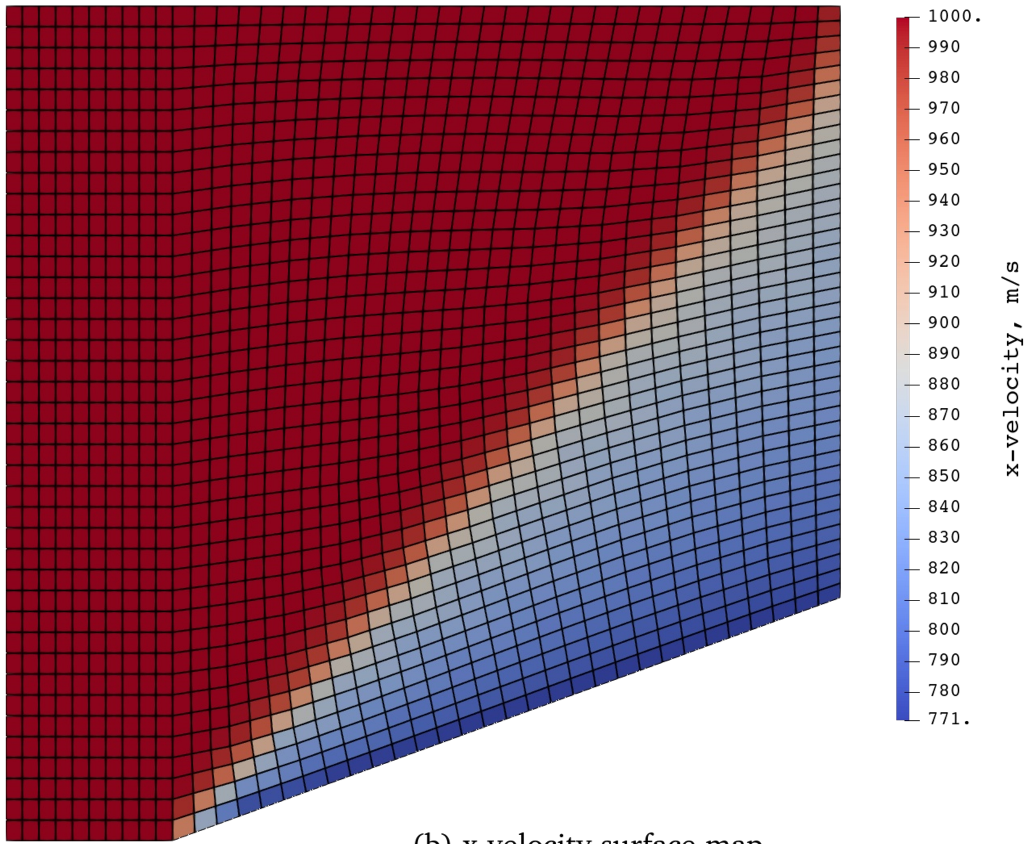
`gdtk/examples/lmr/2D/sharp-cone-20-degrees/sg-minimal`


```
$ lmr run
```

```
$ lmr snapshot2vtk
```



(a) pressure surface map



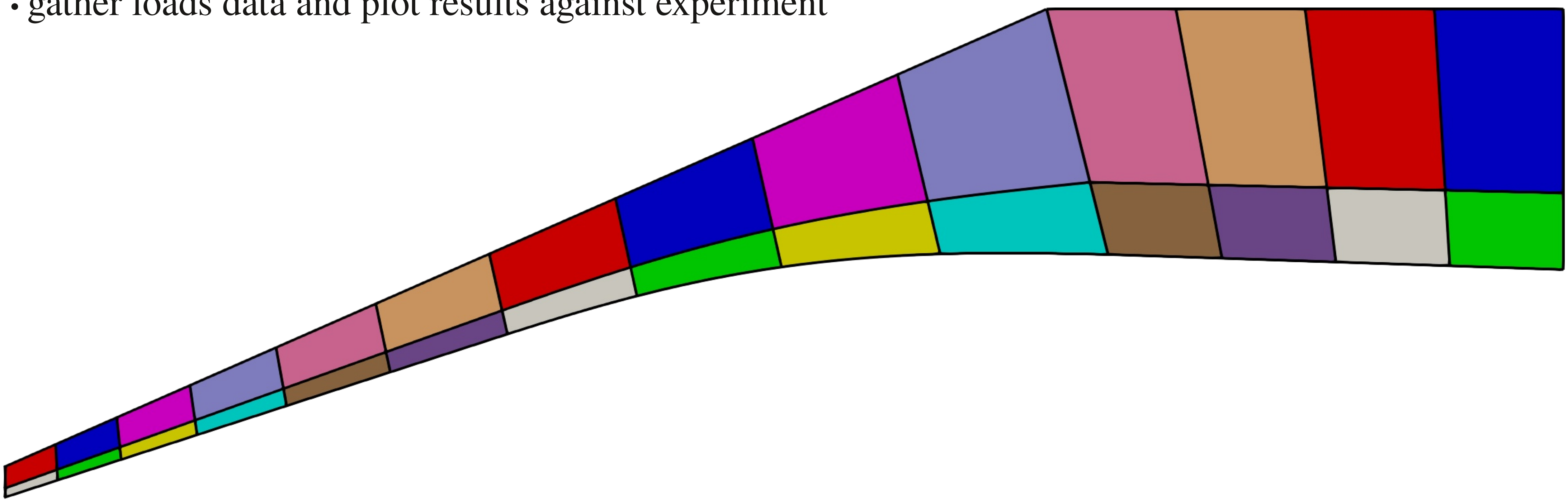
(b) x-velocity surface map

Hypersonic viscous flow over a convex ramp

`gdtk/examples/lmr/2D/convex-ramp`

You will learn how to:

- prepare a nonequilibrium gas model
- use Lua to define a custom path to represent the ramp
- request surface loads as output during the simulation
- run the solver in steady mode
- gather loads data and plot results against experiment



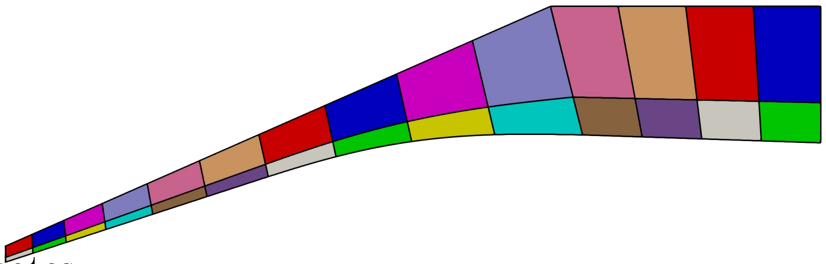
gas model input file

```
model = "TwoTemperatureGas"
species = {'N2', 'O2', 'N', 'O', 'NO'}
```

```
$ lmr prep-gas -i air-5sp-gas-model.lua -o air-5sp-2T.lua
```

Custom path to represent ramp

gdtk/examples/lmr/2D/convex-ramp



- initially straight at 18° until $x = 3$ inches;
- then a faired section defined by

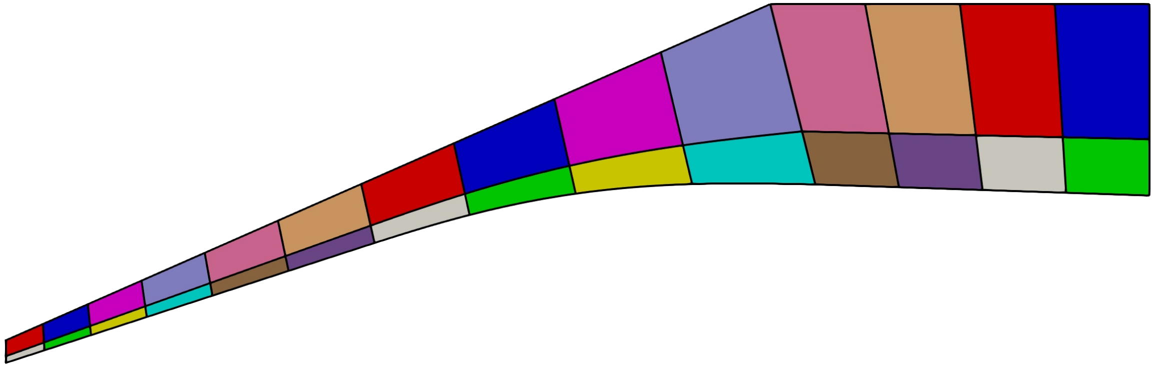
$$g = 0.0026s^4 - 0.0211s^3$$

  where s and g are the local coordinates, in inches, rotated 18° to the x, y coordinates;
- the fairing second derivative is zero at both joining points: $s = 0$ and $s = 4.058$; and
- the final straight section is at -1.9° in the (x, y) plane, away from the free-stream flow direction.

```
function ramp(t)
   local alpha = 18.0*math.pi/180.0 -- angle of initial straight section
   local sin18 = math.sin(alpha)
   local cos18 = math.cos(alpha)
   local tan18 = math.tan(alpha)
   local x_join_inch = 3.0
   local y_join_inch = x_join_inch * tan18
   local L1 = x_join_inch/cos18 -- length of initial straight section
   local L2 = 4.14677 -- length of fairing (computed via maxima)
   local t2 = (L1+L2) * t
   local x_inch, y_inch
   if t2 < L1 then
      x_inch = t2 * cos18
      y_inch = t2 * sin18
   else
      s = (t2 - L1)/L2 * 4.0577
      g = 0.0026 * math.pow(s,4) - 0.0211 * math.pow(s,3)
      x_inch = x_join_inch + s * cos18 - g * sin18
      y_inch = y_join_inch + s * sin18 + g * cos18
   end
   return {x=x_inch*m_per_inch, y=y_inch*m_per_inch, z=0.0}
end
```

Requesting loads as output

gdtk/examples/lmr/2D/convex-ramp



```
bcDict = {
   inflow=InFlowBC_Supersonic:new{flowState=inflow},
   outflow=OutFlowBC_FixedPT:new{p_outside=p_inf/5, T_outside=T_inf},
   noslipwall=WallBC_NoSlip_FixedT:new{Twall=T_wall, group='loads'},
}
--
makeFluidBlocks(bcDict, flowDict)
mpiDistributeBlocks{ntasks=8}
--
NewtonKrylovGlobalConfig{
   ...
   ...
   write_loads = true,
   steps_between_loads_update = 20,
}
```

Running with MPI

`gdtk/examples/lmr/2D/convex-ramp`

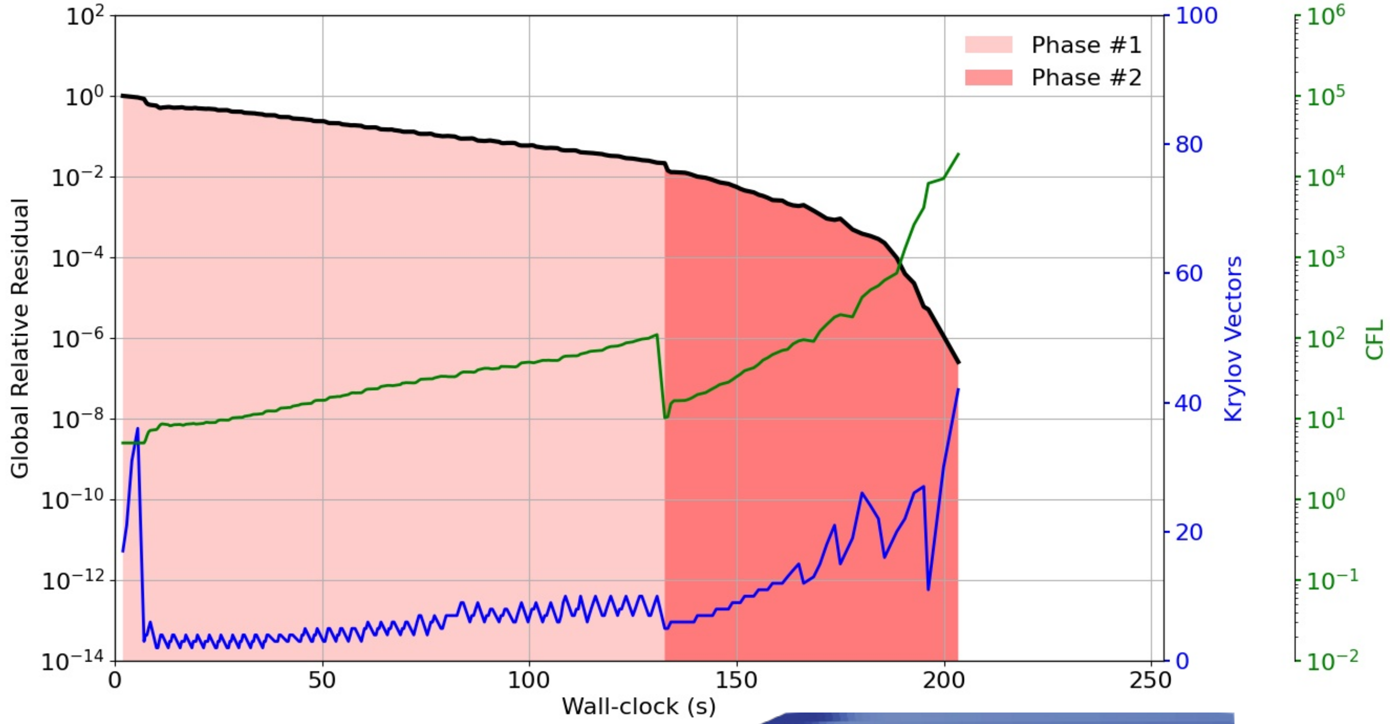
```
$ mpirun -np 8 lmrZ-mpi-run
```

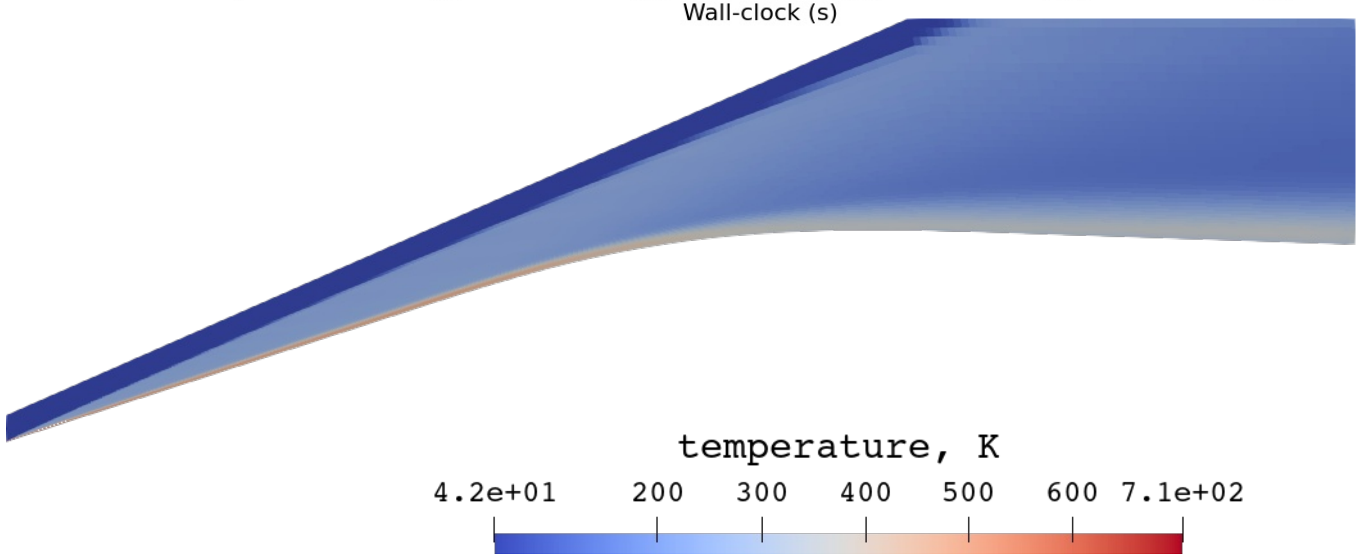
On my laptop with 4 physical cores:

```
$ mpirun --use-hwthread-cpus -np 8 lmrZ-mpi-run
```

Convergence and visualisation

gdtk/examples/lmr/2D/convex-ramp

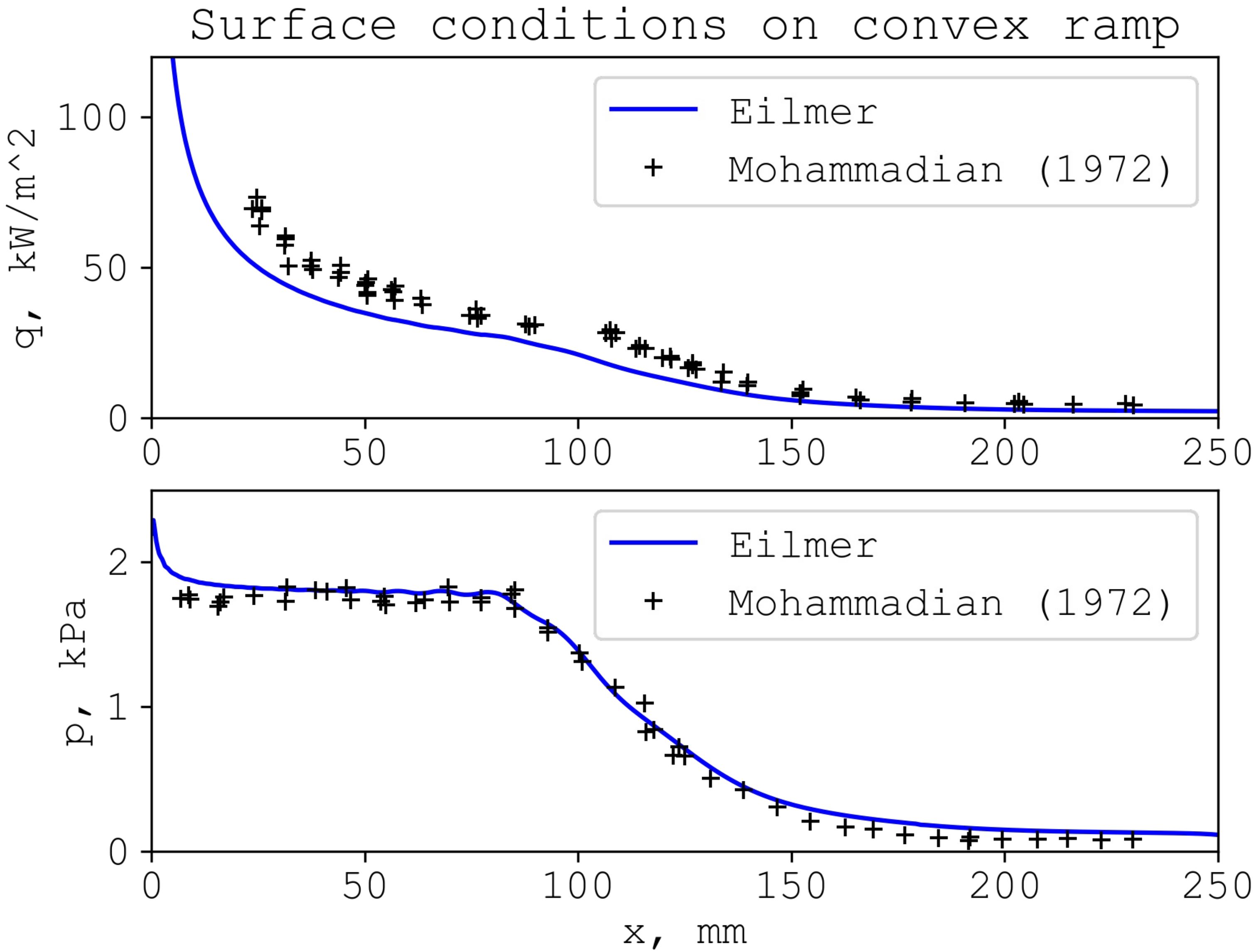




Heat loads and surface pressure distribution

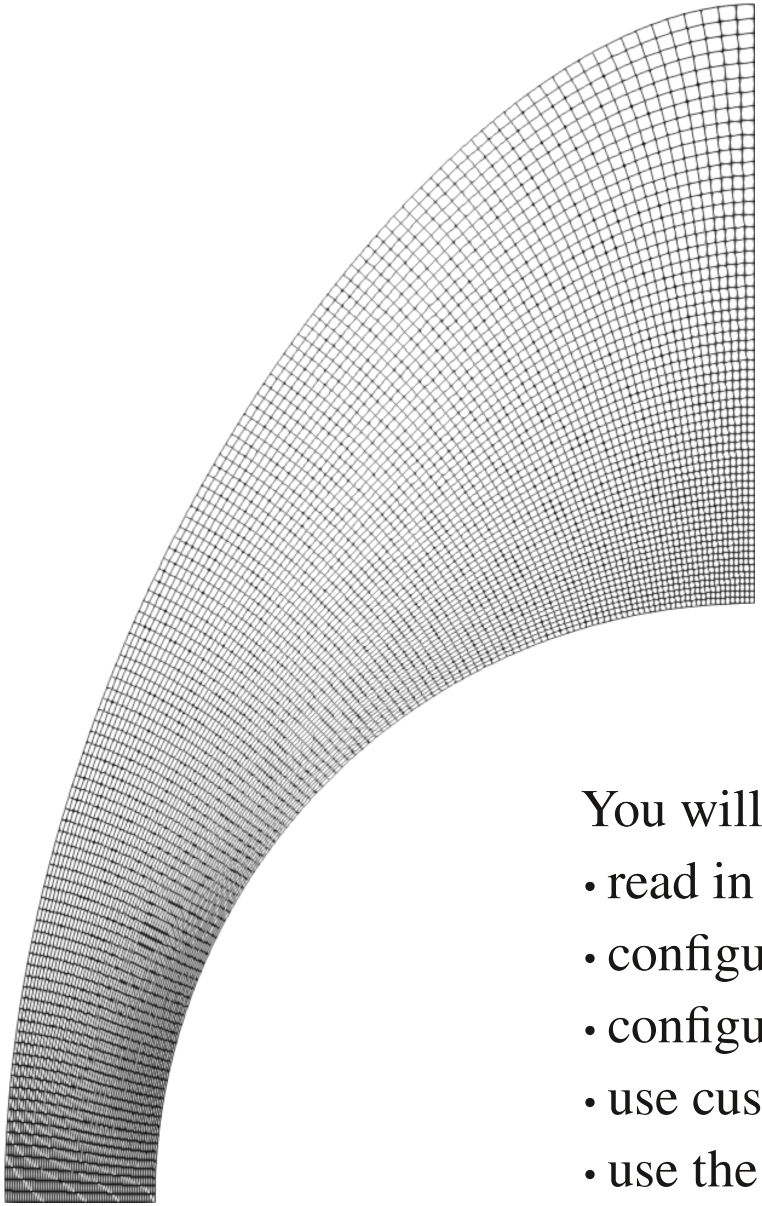
gdtk/examples/lmr/2D/convex-ramp


```
$ python3 plot_surface_conditions.py
```



Multi-temperature reacting air flow over a sphere

gdtk/examples/lmr/2D/sphere-nonaka



To read in a GridPro grid, add to `grid.lua`:

```
gproGrid = "gridpro/sphere-orig.grd"
grids = importGridproGrid(gproGrid, 1.0)
```

You will learn how to:

- read in a 3rd-party grid
- configure chemical reactions and energy exchange mechanisms
- configure boundary shock-fitting mode
- use custom post-processing to extract shock shape
- use the slicing tool to get stagnation line data

gas model input file

```
model = "TwoTemperatureGas"
species = {'N2', 'O2', 'N', 'O', 'NO'}
```

```
$ lmr prep-gas -i air-5sp-gas-model.lua -o air-5sp-2T.lua
```

reactions input file

```
...
-- Park 's' parameter
s = 0.5

SUBSET_SELECTION = '5-species-air'

Reaction{
   'O2 + M <=> O + O + M',
   fr={'Park', A=3.610e+18, n=-1.00, C=59400.00, s=s},
   br={'Arrhenius', A=3.010e+15, n=-0.50, C=0.0},
   label='r1',
   efficiencies={O2=9.0, N2=2.0, O=25.0, N=1.0, NO=1.0,
       ['NO+']=0.0, ['O2+']=0.0, ['N2+']=0.0, ['O+']=0.0,
       ['N+']=0.0, ['e-']=0.0}
}
...
```

```
$ lmr prep-reactions -g air-5sp-2T.gas -i GuptaEtAl-air-reactions-2T.lua \
  -o air-5sp-6r-2T.chem
```

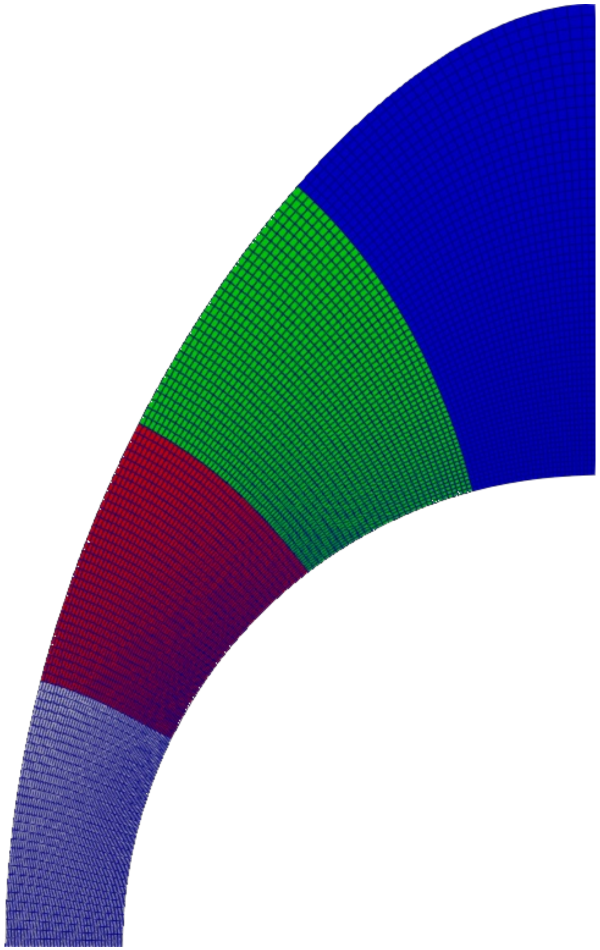
energy exchange input file

```
Mechanism{
   "(*molcs) ~~ (*heavy)",
   type = "V-T",
   rate = "Landau-Teller",
   relaxation_time = {"ParkHTC", submodel={"Millikan-White"}}
}
```

```
$ lmr prep-energy-exchange -g air-5sp-2T.gas -r air-5sp-6r-2T.chem \
  -i air-energy-exchange.lua -o air-VT.exch
```

Configure boundary shock-fitting

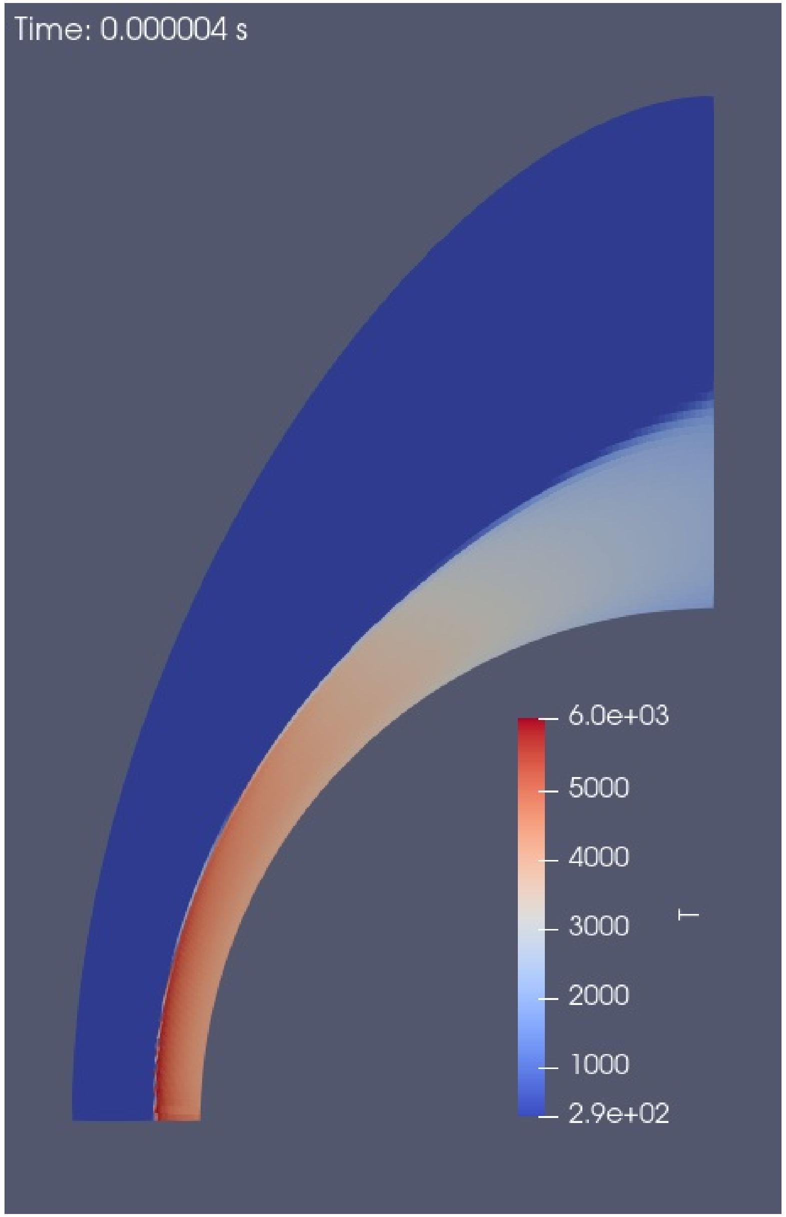
gdtk/examples/lmr/2D/sphere-nonaka

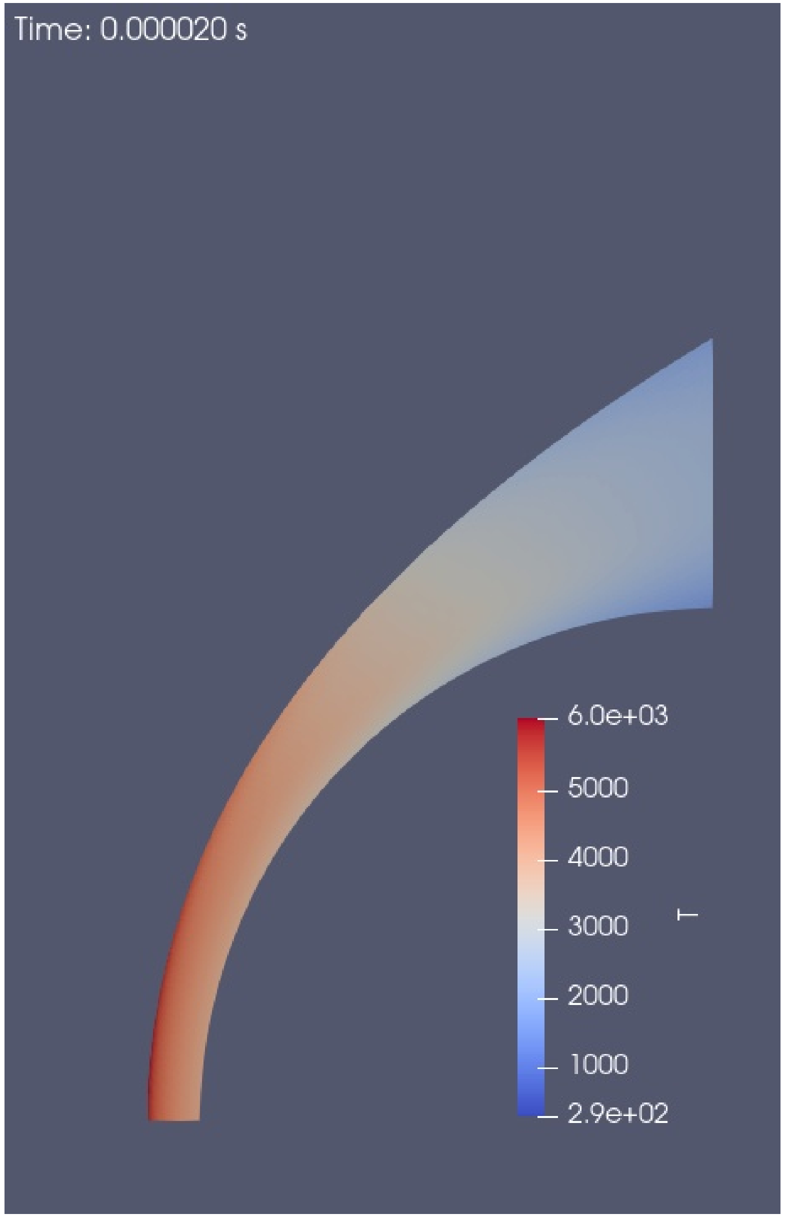


```
...
config.grid_motion = "shock_fitting"
config.gasdynamic_update_scheme = "moving_grid_2_stage"
config.shock_fitting_delay = body_flow_time
...
...
grids = importGridproGrid(gproGrid, 1.0)
registerFluidGridArray{
   grid=grids[1],
   nib=1, njb=njb,
   fsTag='initial',
   shock_fitting=true,
   bcTags={west='inflow_sf', north='outflow', east='wall'}
}
--
bcDict = {
   inflow_sf=InFlowBC_ShockFitting:new{flowState=inflow},
   outflow=OutFlowBC_Simple:new{},
   wall=WallBC_WithSlip0:new{}
}
...
```

Run simulation and visualise

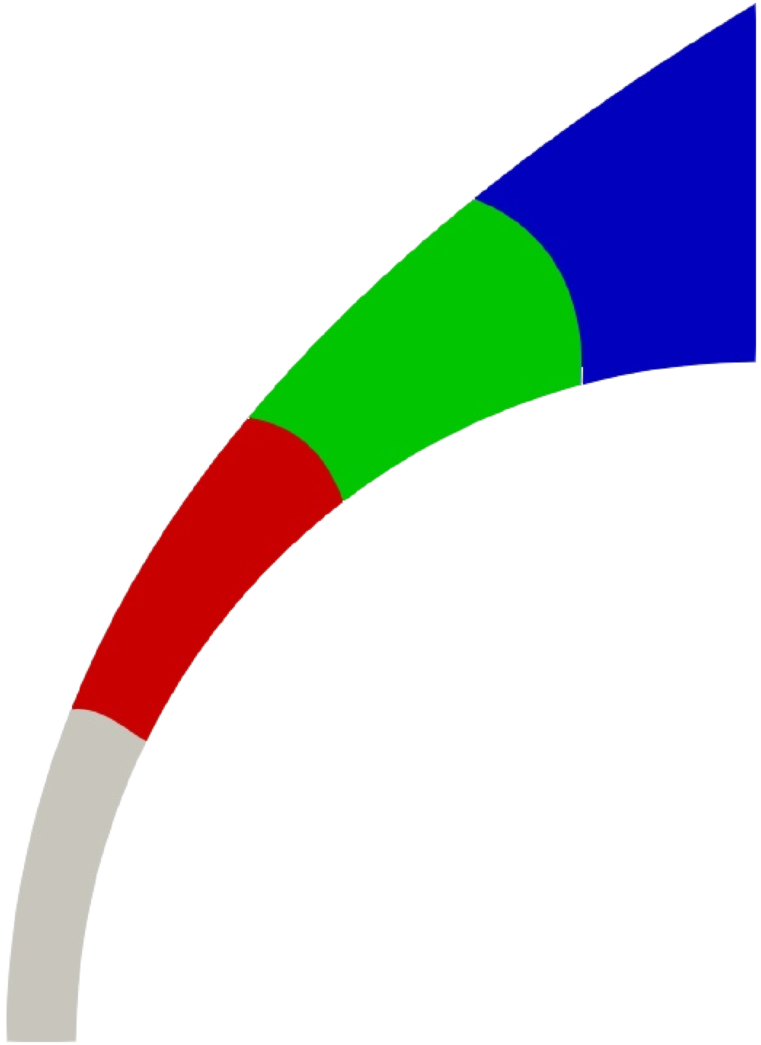
gdtk/examples/lmr/2D/sphere-nonaka





Custom post-processing: extract shock shape

gdtk/examples/lmr/2D/sphere-nonaka

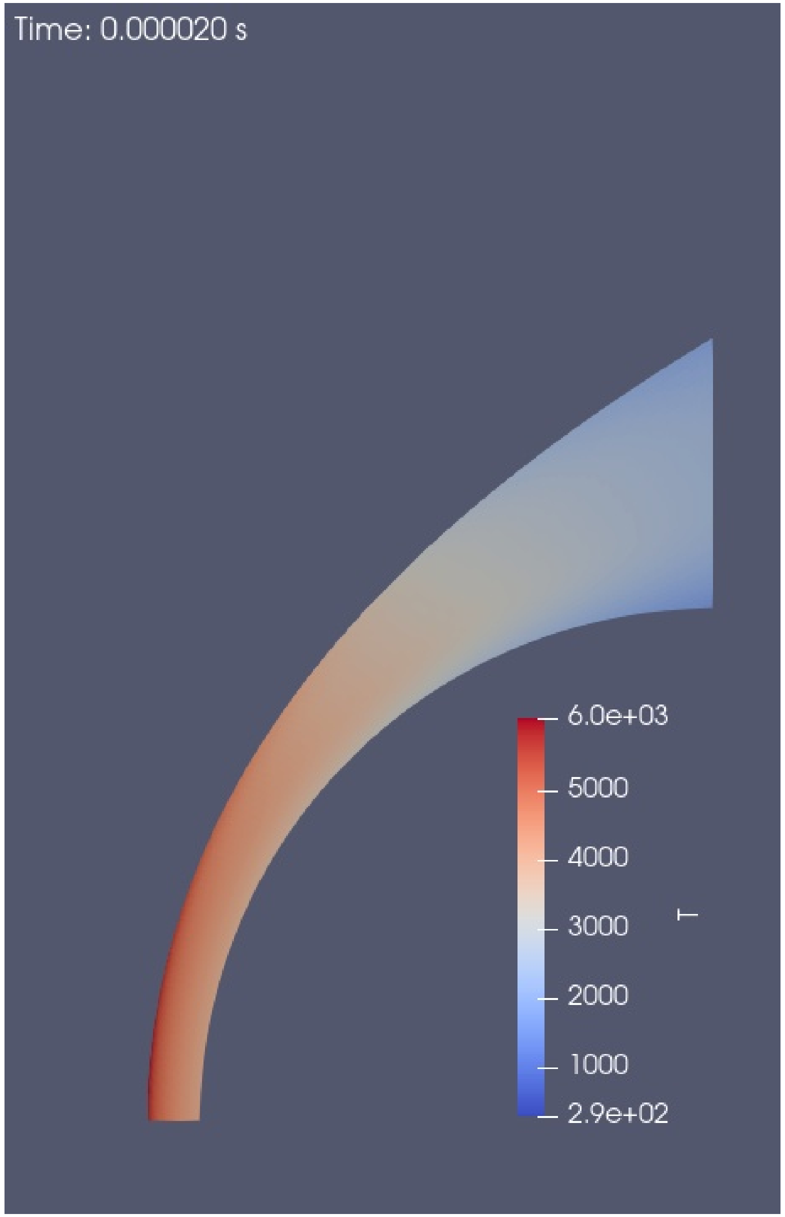


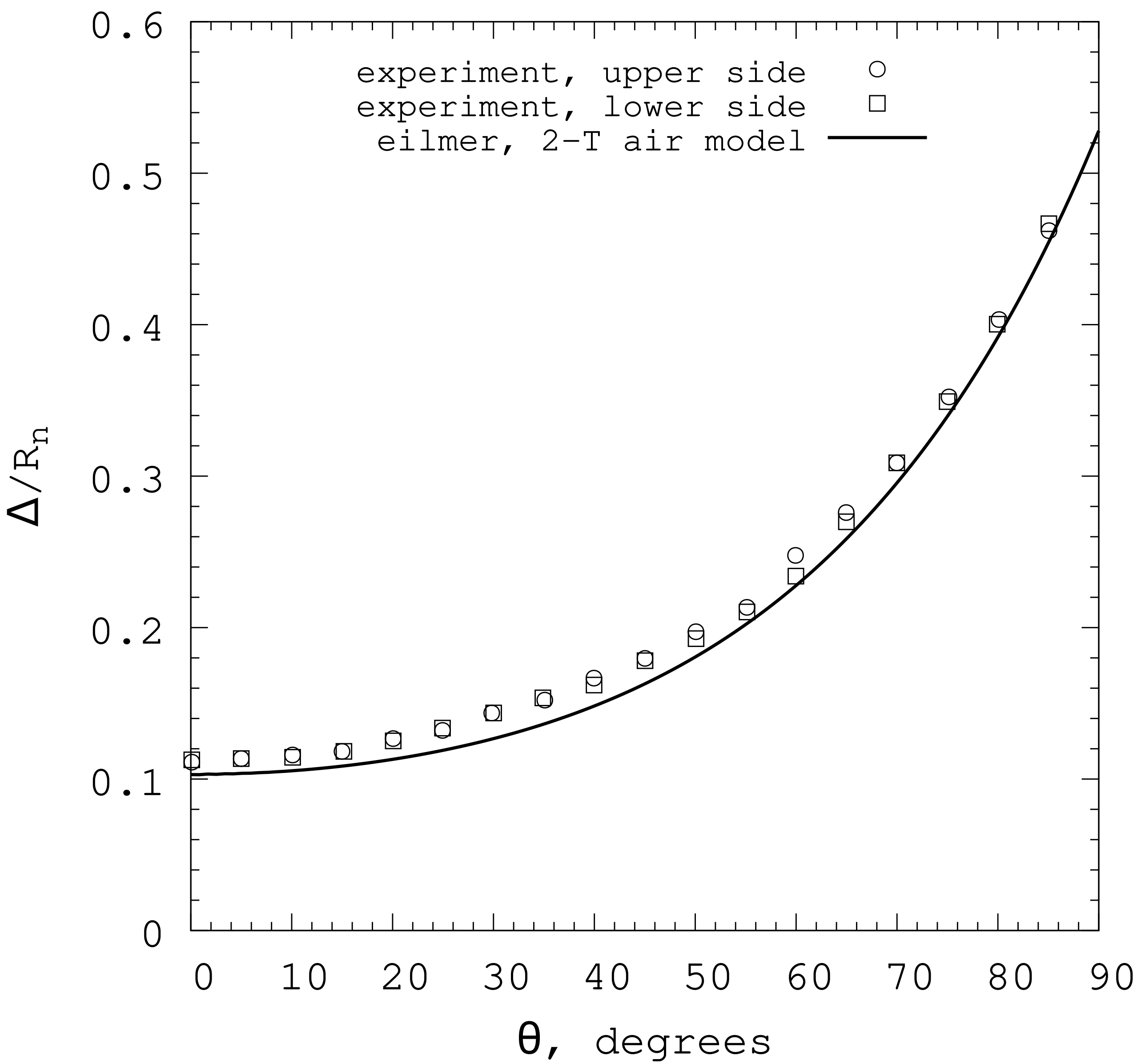
```
-- Pick up flow solution at final time
fsol = FlowSolution:new{dir=".",
snapshot="last", nBlocks=4}
f = io.open("shock-shape.dat", 'w')
f:write("#  x     y    theta    d/R\n")
for ib=0,3 do
   jstart = 1
   if ib == 0 then
      jstart = 0
   end
   for j=jstart,nyVtxsPerBlock-1 do
      vtx = fsol:get_vtx{ib=ib, i=0, j=j}
      x = vtx.x - R
      y = vtx.y
      theta = deg(atan2(y, -x))
      d = sqrt(x*x + y*y) - R
      d_R = d/R
      f:write(string.format("%20.12e %20.12e
%20.12e %20.12e\n", x, y, theta, d_R))
   end
end
f:close()
```

```
$ lmr custom-script --job=shock-shape.lua
```

Shock shape comparison to experiment

gdtk/examples/lmr/2D/sphere-nonaka

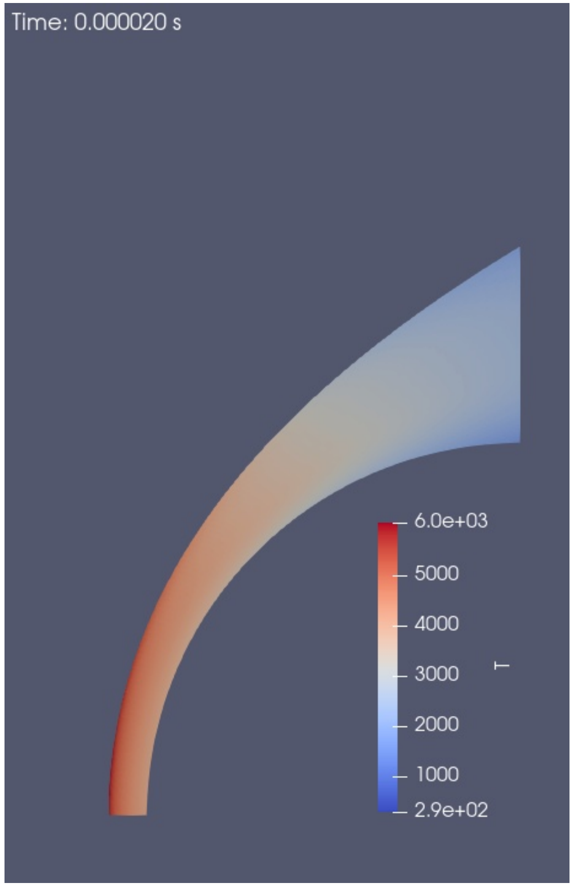


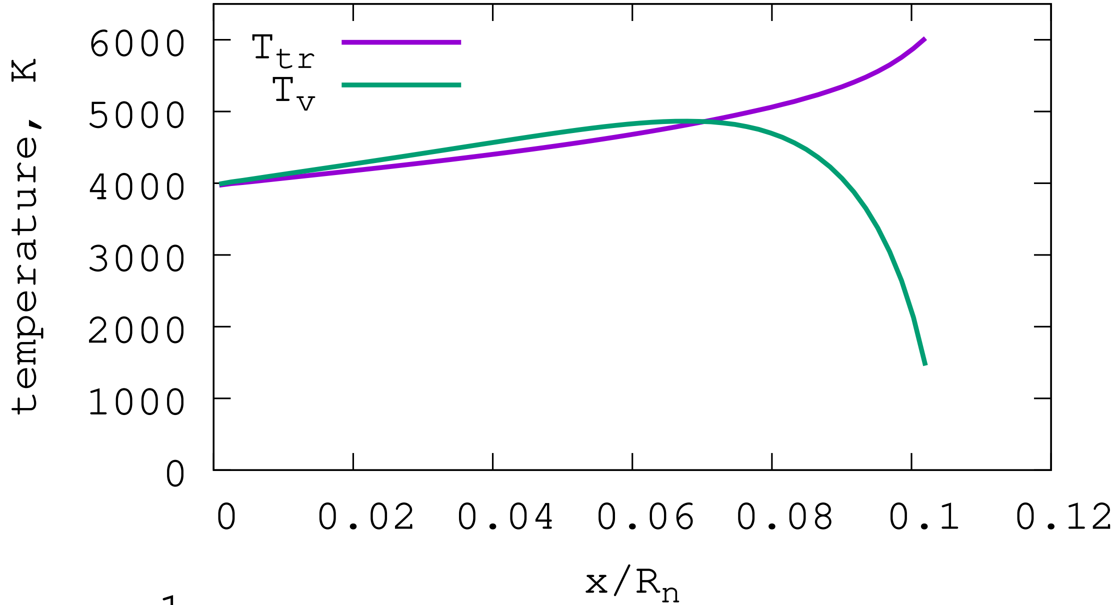


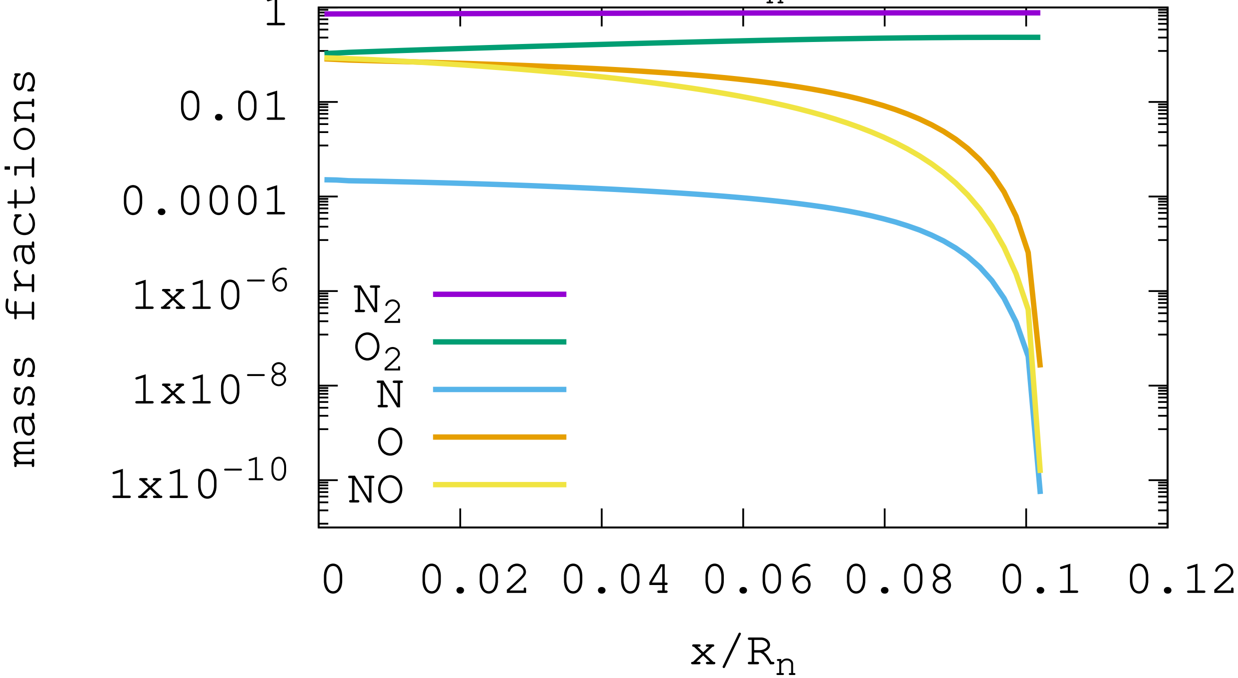
Extracting stagnation line data

gdtk/examples/lmr/2D/sphere-nonaka


```
$ lmr slice-flow --final --slice-list="0,:,0,0" --names="T,T-vibroelectronic" \
--output=stagnation-profile-T.dat
```







Catalogue of Examples

User Guides

Reference Manual

Technical Notes

Github issue tracker