The Gas Dynamic Tool Kit – March of the Puffin

Peter Jacobs

The University of Queensland

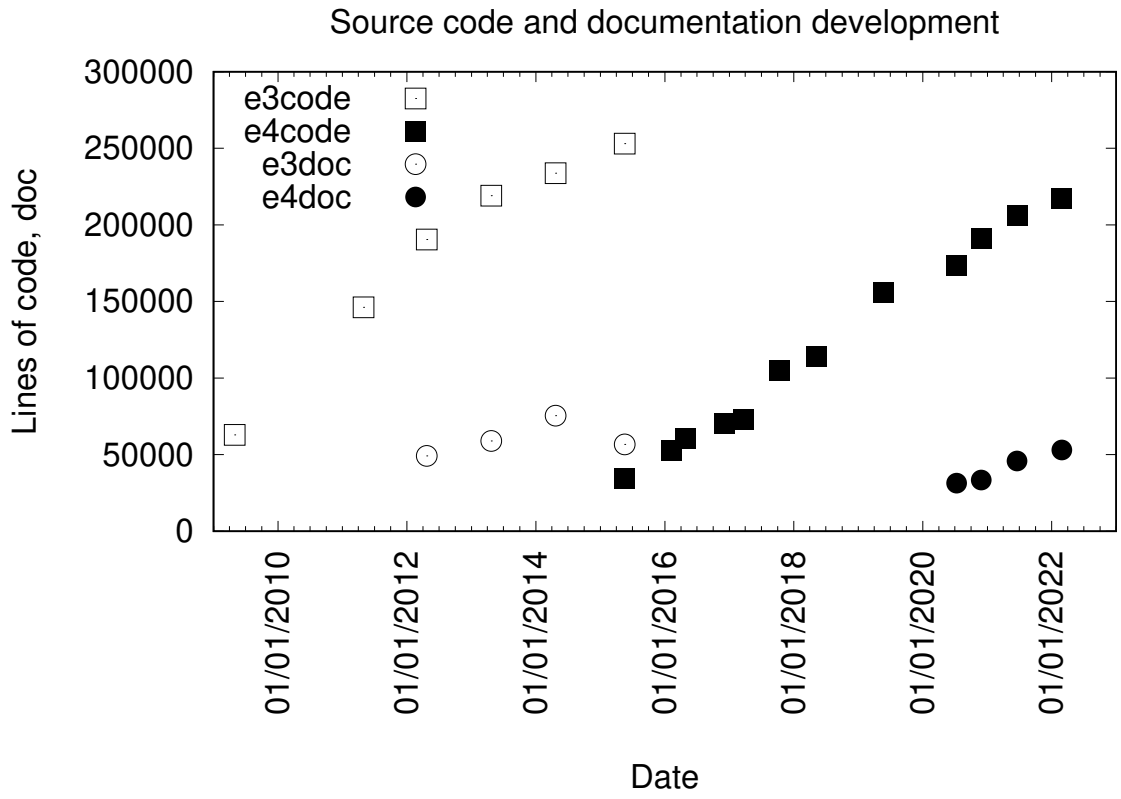
10 Mar 2022

Motivation for writing yet-another program

Puffin space-marching code

Sample applications

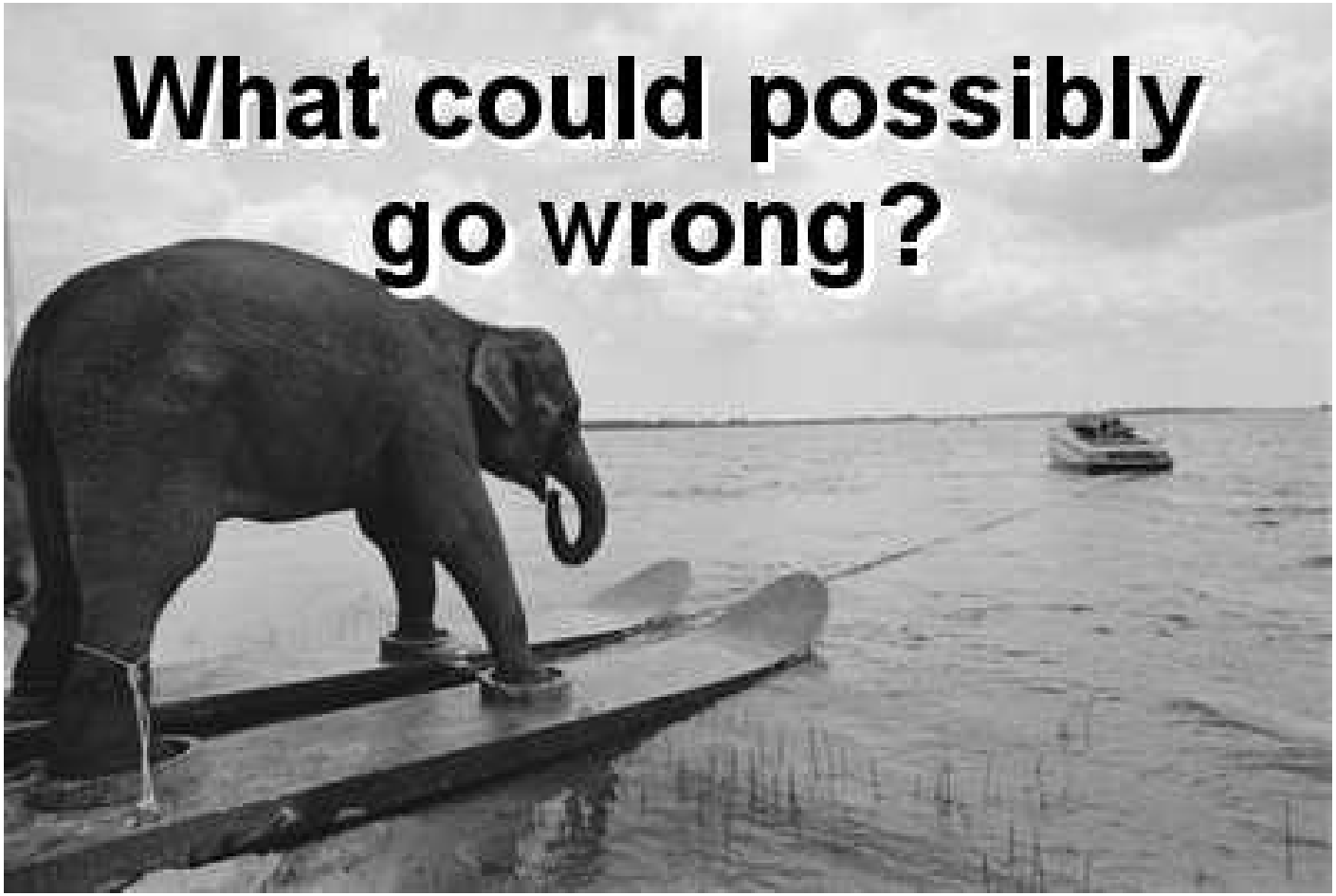
Progress of the GDTK code collection. Are we there yet?



According to Bill Gates:

- "Measuring programming progress by lines of code is like measuring aircraft building progress by weight."
- "There are no significant bugs in our released software that any significant number of users want fixed."

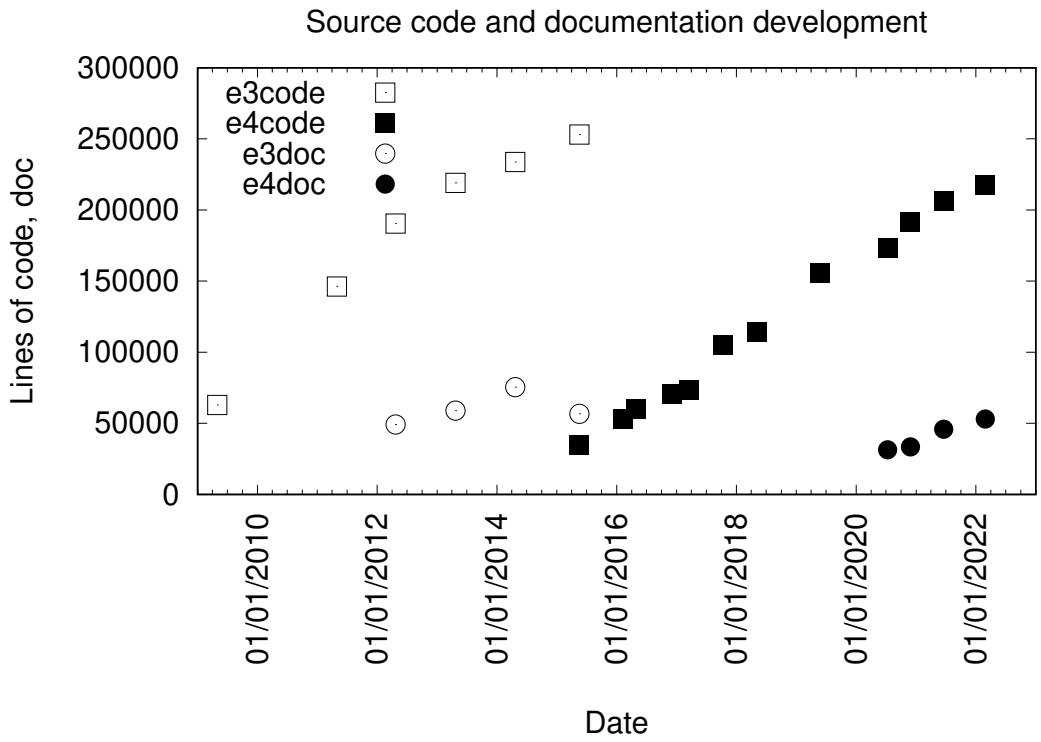
So, let's write a new code



The only sensible thing to do over the summer holiday.

Motive 1: The code is getting heavy.

Margaret Hamilton (1969) with Apollo Guidance Computer source code, assembly, ≈11,000 pages.



At 60 lines per page, the Eilmer4 code weighs in as a 4200 page document.

How much better can we do?

Compare lines of code in similar functions.

Eilmer4 simcore.d

- init_simulation()
  681 lines
- integrate_in_time()
  409 lines

Puffin marching_calc.d

- init_calculation()
  66 lines
- relax_slice_to_steady_flow()
  35 lines

This is looking promising. I would prefer to read and try to understand 1 page of code than 11 pages.

Motive 2: Replace the SEAGULL code

VOL. 14, NO. 5, MAY 1976 AIAA JOURNAL

Shock Fitting Method for Complicated Two-Dimensional Supersonic Flows

Manuel D. Salas*
NASA Langley Research Center, Hampton, Va.

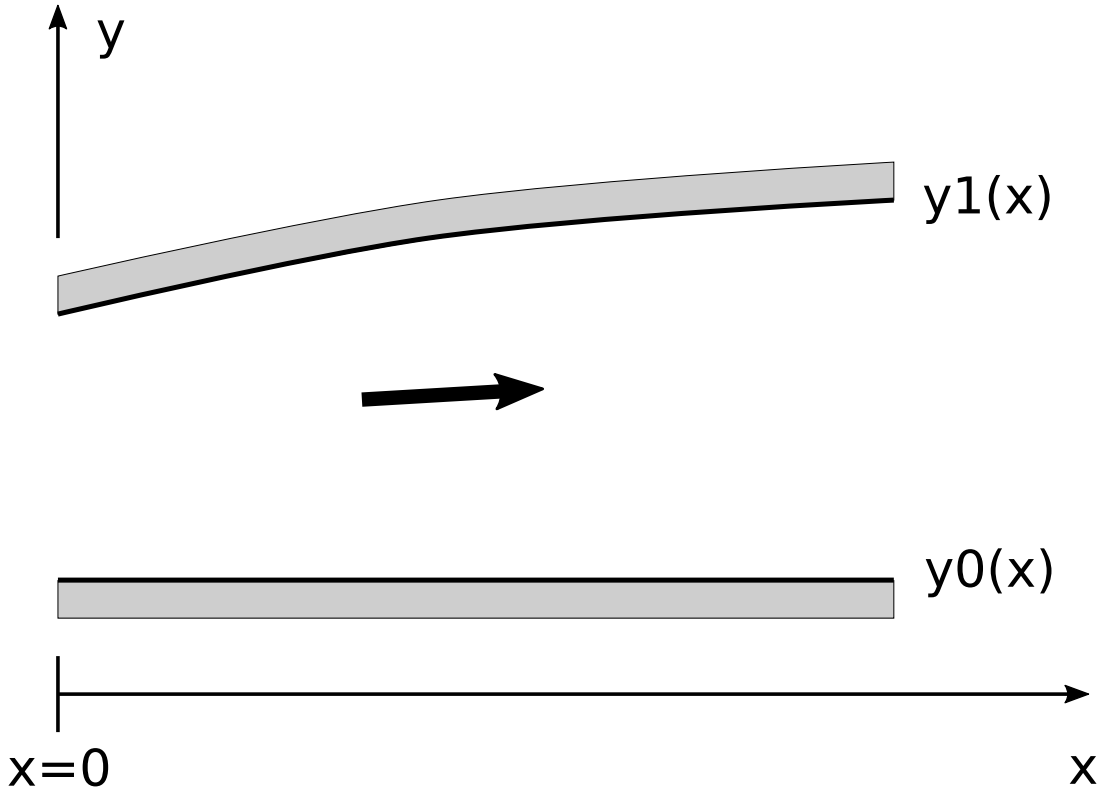
- 2D, inviscid, supersonic flow of an ideal gas.
- Floating-shock-fitting that tracks the shocks explicitly.
- More than 4000 lines of FORTRAN code.
- It's getting difficult to build into Rowan's inlet-design tools.

The Puffin space-marching code

- Space-marching, for speed, so that we can use it inside a design loop.
- Shock-capturing, for convenience.
- Inviscid, supersonic 2D flow, to limit the scope of the code.
- Complex thermochemistry comes for free with GDTK.
- Design goal: simple enough code to write in a week.
- Note that Eilmer remains our main simulation code. Puffin is our training code.

https://www.audubon.org/field-guide/bird/atlantic-puffin#photo7

Stream tube – top-level object



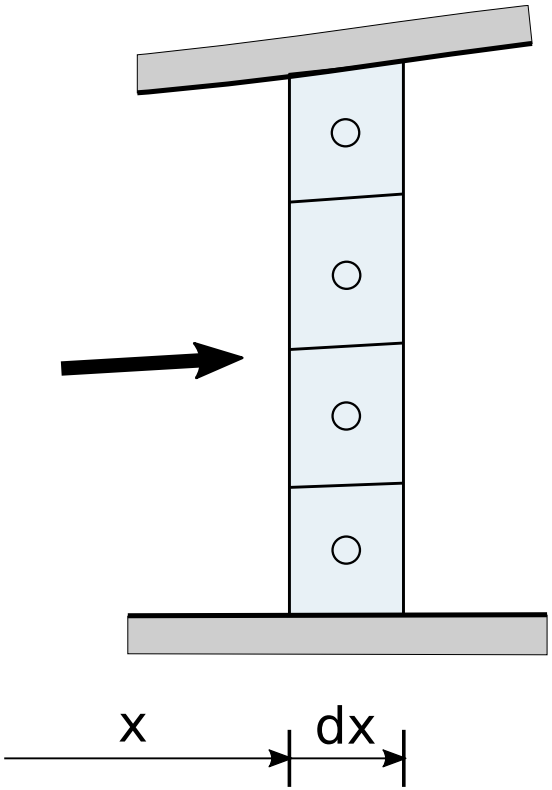
- Assume a dominant supersonic flow in the positive x-direction.
- Start with a specified flow at $x = 0$.
- Work from left to right, solving the steady-flow solution, one thin (dx) slice at a time.

Overview of the calculation process

Given a supersonic inflow at $x = 0$ and a bounding streamtube:

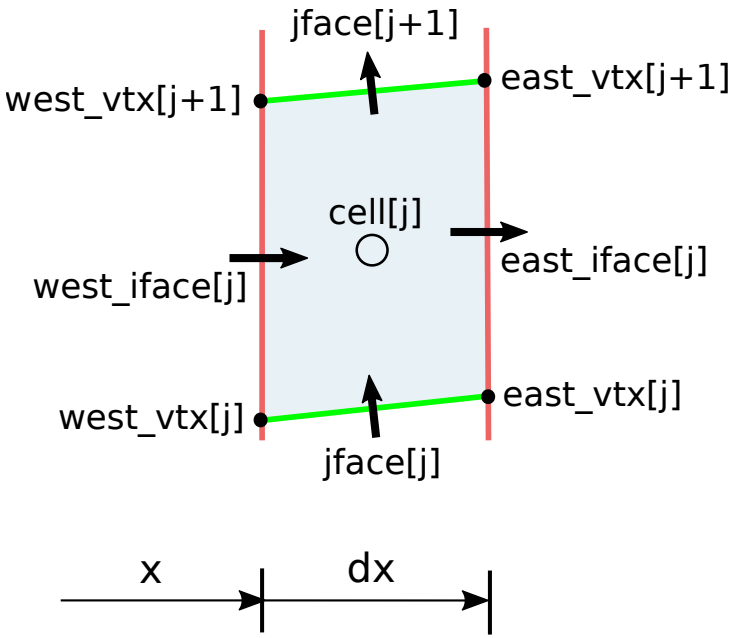
1. Discretize a thin slice of the domain, width dx, into an array of cells.
2. Copy the supersonic flow from the left into the cells.
3. Relax to steady flow within the slice of cells while applying the boundary conditions at the stream-tube walls.
4. Maybe print the flow data for the slice.
5. Move on to position $x + dx$
6. If $x < xmax$, go back to step 1.
7. We are done.

Slice of finite-volume cells – component objects



- Each slice is discretized into an array of finite-volume cells.
- Cell-average flow data is associated with the cell centre.
- Supersonic flow so, as an initial guess, copy the flow data from upstream (left to right).
- It takes time for information to propagate across the slice from the boundaries.

Cells, faces, vertices and indexing



vertex: geometric location

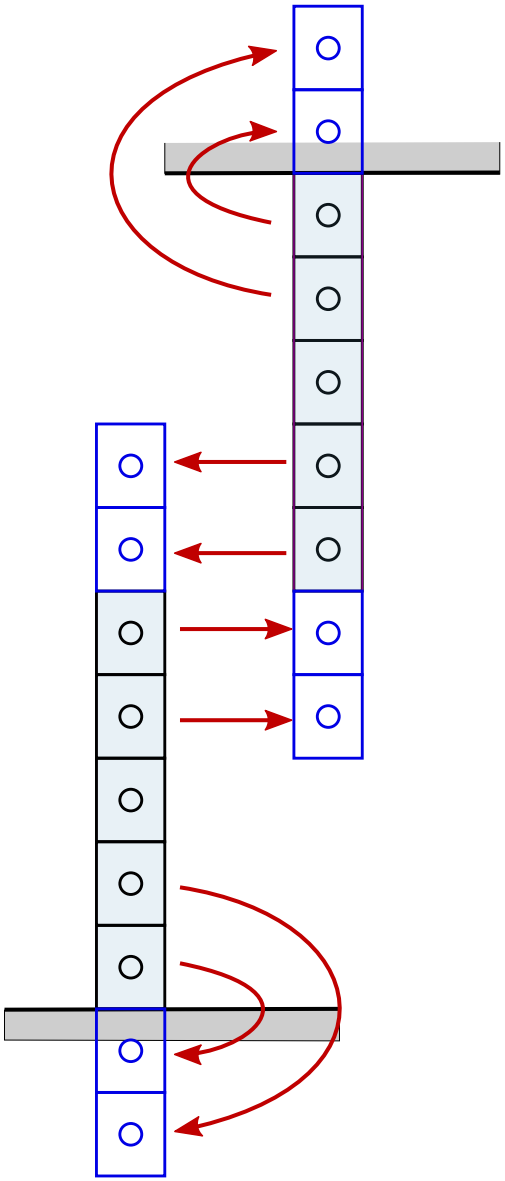
face:

- Defined by vertices at ends.
- We calculate fluxes of mass, momentum and energy across each face.

cell:

- Defined by bounding faces.
- Flow state associated with cell centre.
- Each finite-volume cell is the basic unit over which the gas-dynamic equations are applied.

Boundary conditions, multiple streams



The picture at left shows the cells for two stream tubes. They are at equal x-positions even though they are shown offset.

- Each stream tube has two "ghost" cells associated with each end of the slice.
- Boundary conditions are implemented by copying appropriate data into the ghost cells. Red arrows indicate data transfer.
- An adjacent stream interacts through a simple copy of the flow data.
- There is a fixed ordering of the streams.
- Interaction with a solid wall involves reflection of the flow data in the frame of the boundary face.

Gas-dynamic equations

Conservation statements for mass, momentum and energy determine what happens next:

$$\frac{\partial}{\partial t}\int_V U dV = -\oint_S \overline{F}_c \cdot \hat{n}\, dA + \int_V Q dV \ ,$$

For a single chemical species, the array (U)of conserved quantities and associated flux vectors (F) are

$$U = \begin{bmatrix} \rho \\ \rho v_x \\ \rho v_y \\ \rho E \end{bmatrix} \ , \ \overline{F}_c = \begin{bmatrix} \rho v_x \\ \rho v_x^2 + p \\ \rho v_y v_x \\ \rho E v_x + p v_x \end{bmatrix} \hat{i} + \begin{bmatrix} \rho v_y \\ \rho v_x v_y \\ \rho v_y^2 + p \\ \rho E v_y + p v_y \end{bmatrix} \hat{j} \ ,$$

Conserved quantities in the U vector are per unit volume.
Total specific energy is $E = u + \frac{1}{2}v^2$.
The source term for axisymmetric flow is $Q = [0, 0, pA_{xy}/V, 0]$.

Flux calculators

To compute the flux of mass, momentum and energy at each face, we work in the local coordinate frame of the face, $\hat{n}$ is the unit normal, $\hat{t}$ is the corresponding unit tangent.

We usually think of the flux calculator as incorporating some approximate solution to the Riemann problem for the interaction of a left (L) flow state and a right (R) flow state, with the flux values being estimated at the initial location of the interface.

The flux for a uniform supersonic flow is

$$\overline{F}_c = \begin{bmatrix} \dot{m} \\ \dot{m}\, v_n + p \\ \dot{m}\, v_t \\ \dot{m}\,(E + p/\rho) \end{bmatrix} \hat{n}$$

where $\dot{m} = \rho\, v_n$ is the mass flux.

Time integration

- Code extract from function relax_slice_to_steady_flow().
- Work on a single slice of cells in each stream tube.

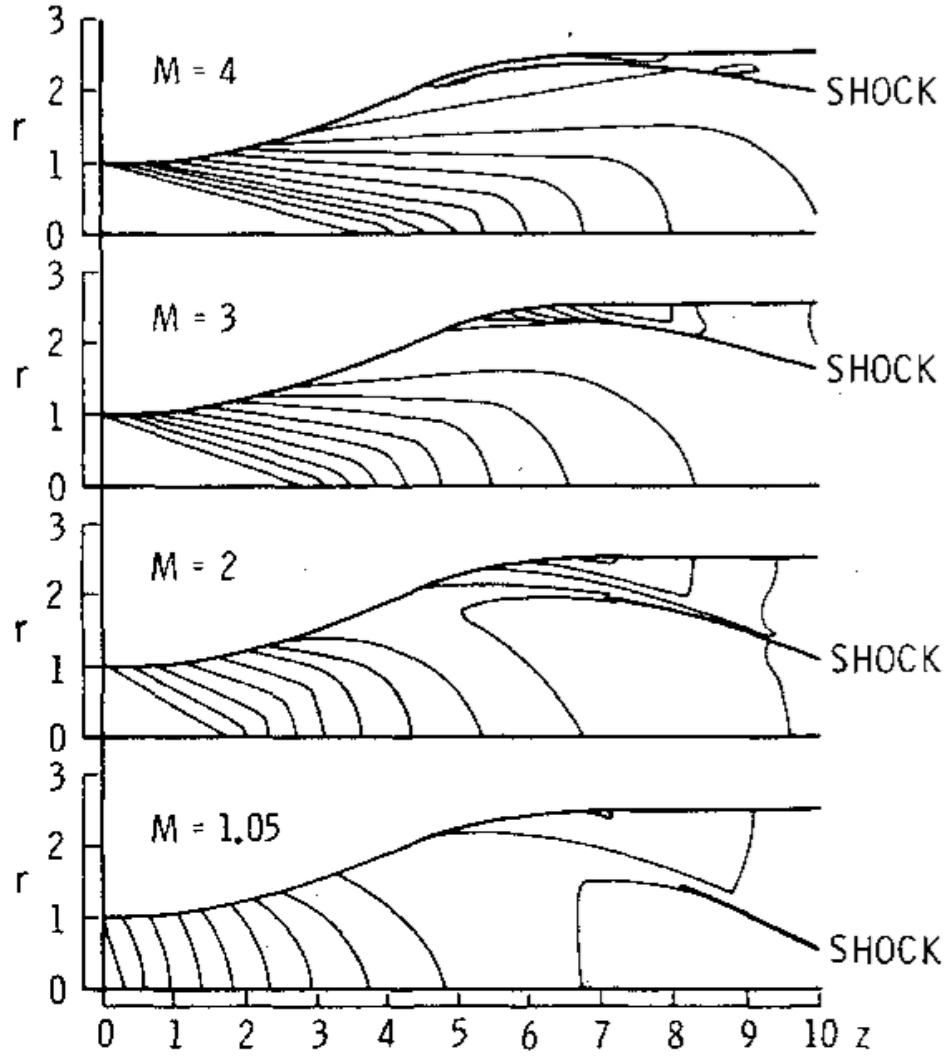
```
foreach (k; 0 .. Config.max_step_relax) {
    // 1. Predictor (Euler) step..
    apply_boundary_conditions(xmid);
    foreach (st; streams) { st.mark_shock_cells(); }
    foreach (st; streams) { st.predictor_step(dt); }
    if (Config.t_order > 1) {
        apply_boundary_conditions(xmid);
        foreach (st; streams) {
            st.corrector_step(dt);
            st.transfer_conserved_quantities(2, 0);
        }
    } else {
        // Clean-up after Euler step.
        foreach (st; streams) {
            st.transfer_conserved_quantities(1, 0);
        }
    }
    // 3. [TODO] measure residuals overall
    // break early, if residuals are small enough
}
```

Space marching

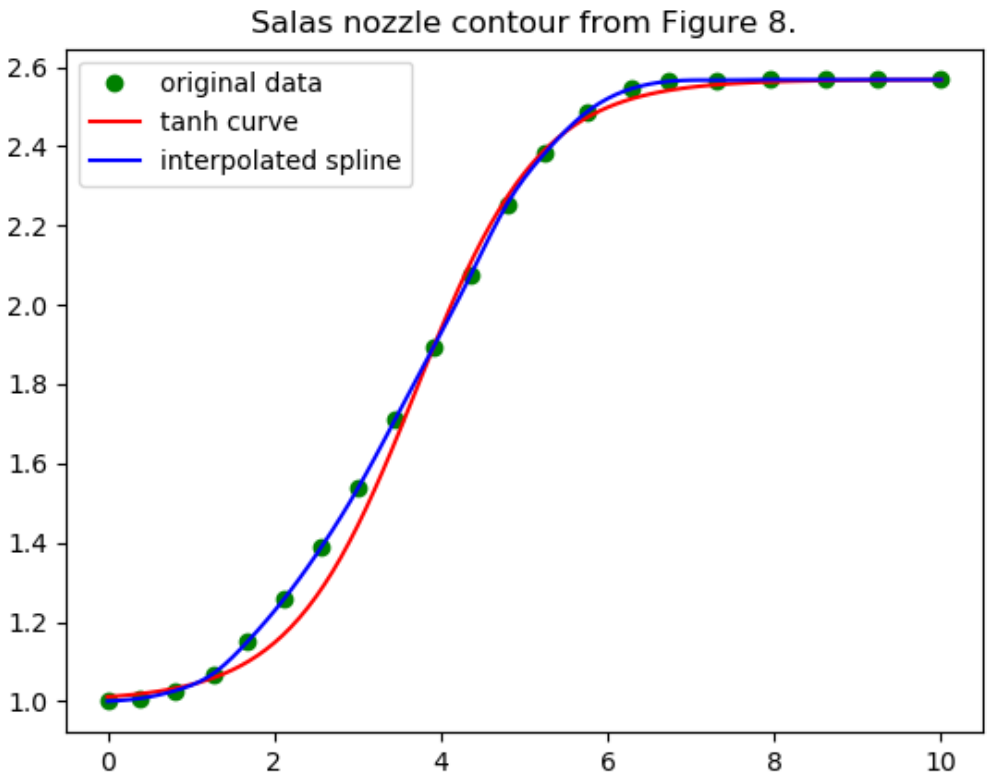
```
while (progress.x < Config.max_x || progress.step < Config.max_step) {
    // 1. Set size of space step.
    if (progress.dx < Config.dx) { progress.dx *= 1.2; }
    progress.dx = min(Config.dx, progress.dx);
    // 2. Take a step.
    int attempt_number = 0;
    bool step_failed;
    do {
        ++attempt_number;
        step_failed = false;
        try {
            foreach (st; streams) { st.set_up_slice(progress.x + progress.dx); }
            relax_slice_to_steady_flow(progress.x + 0.5*progress.dx);
        } catch (Exception e) {
            writefln("Step failed e.msg=%s", e.msg);
            step_failed = true;
            progress.dx *= 0.2;
        }
    } while (step_failed && (attempt_number <= 3));
    if (step_failed) {
        throw new Exception("Step failed after 3 attempts.");
    }
    //
    // 3. Prepare for next spatial step.
    foreach (st; streams) { st.shuffle_data_west(); }
    progress.x += progress.dx;
    progress.step++;
```

Salas' 2D nozzle expansion

Fig. 8 Parametric study of a nozzle, showing shock waves and isobars.



Best guess for nozzle profile



x	y
0	1
0.3944	1.0072
0.8121	1.0267
1.2645	1.0691
1.6821	1.151
2.1114	1.2563
2.566	1.3892
3.0046	1.5376
3.4571	1.7136
3.9095	1.8930
4.3619	2.0772
4.7912	2.2536
5.2436	2.3821
5.7541	2.4873
6.2877	2.5455
6.7401	2.564
7.3086	2.567
7.9466	2.568
8.6079	2.568
9.2459	2.568
10	2.568

Puffin input script – preamble

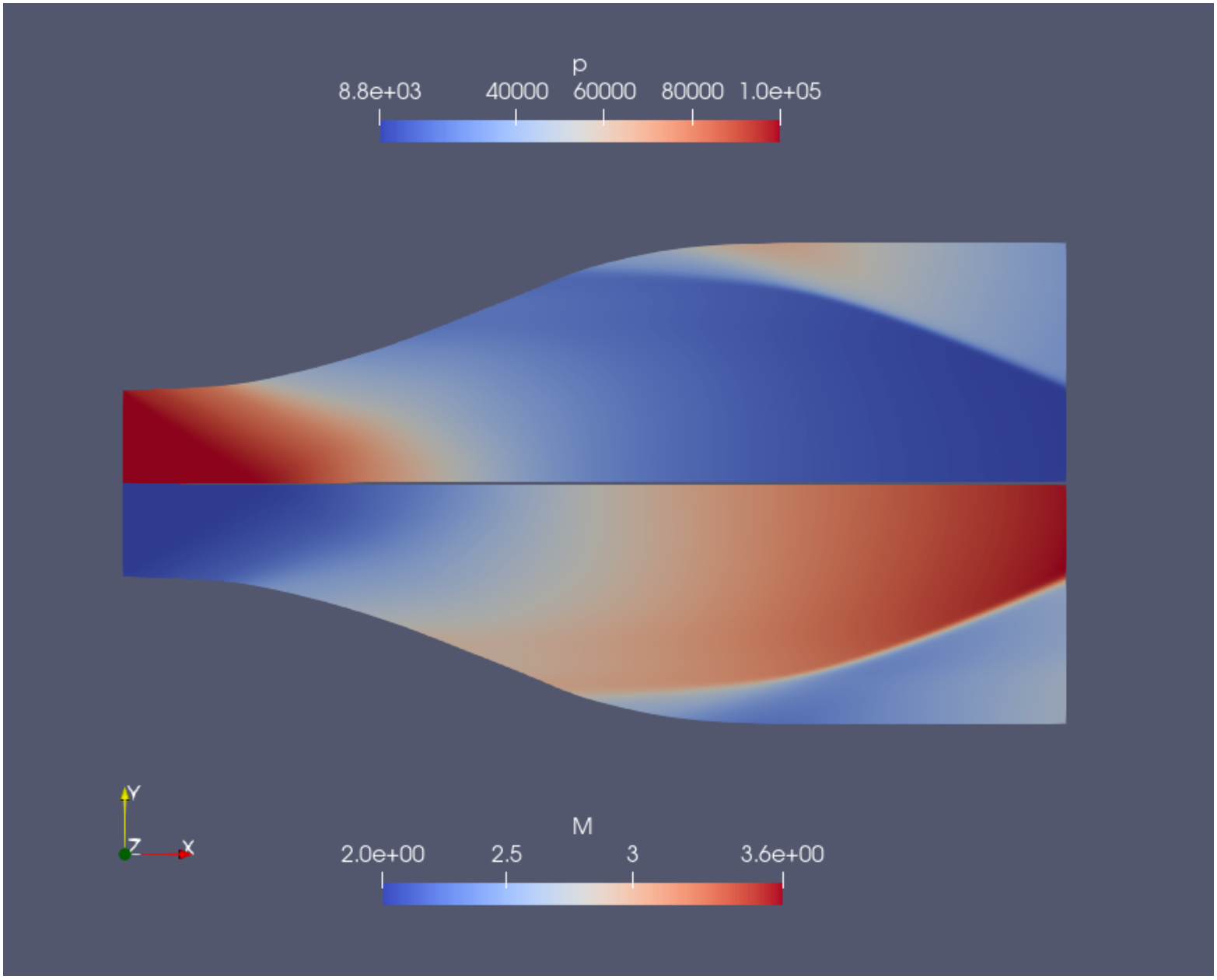
```
# Salas' axisymmetric nozzle example.
# PJ 2022-02-03

init_gas_model("ideal-air-gas-model.lua")
gas1 = GasState(config.gmodel)
gas1.p = 100.0e3
gas1.T = 300.0
gas1.update_thermo_from_pT()
gas1.update_sound_speed()
M1 = 2.0
V1 = M1 * gas1.a
print("V1=", V1)

config.max_step_relax = 40
```

- Input script is written in Python.
- You have full access to all of the GDTK gas models.
- Puffin is not yet complete (relaxation calculation does not yet identify steady flow)

Puffin input script – stream tube definition

```
import csv
xs = []; ys = []
with open('salas-nozzle-path.tsv', 'r') as tsvf:
    tsv_data = csv.reader(tsvf, delimiter='\t')
    for row in tsv_data:
        if row[0] == 'x': continue
        xs.append(float(row[0]))
        ys.append(float(row[1]))

from eilmer.spline import CubicSpline
upper_y = CubicSpline(xs, ys)
def lower_y(x): return 0.0
def lower_bc(x): return 0
def upper_bc(x): return 0

config.max_x = xs[-1]
config.dx = config.max_x/1000

st1 = StreamTube(gas=gas1, velx=V1, vely=0.0,
                 y0=lower_y, y1=upper_y, bc0=lower_bc, bc1=upper_bc,
                 ncells=80)
```

Puffin solution



Salas' scramjet flowpath

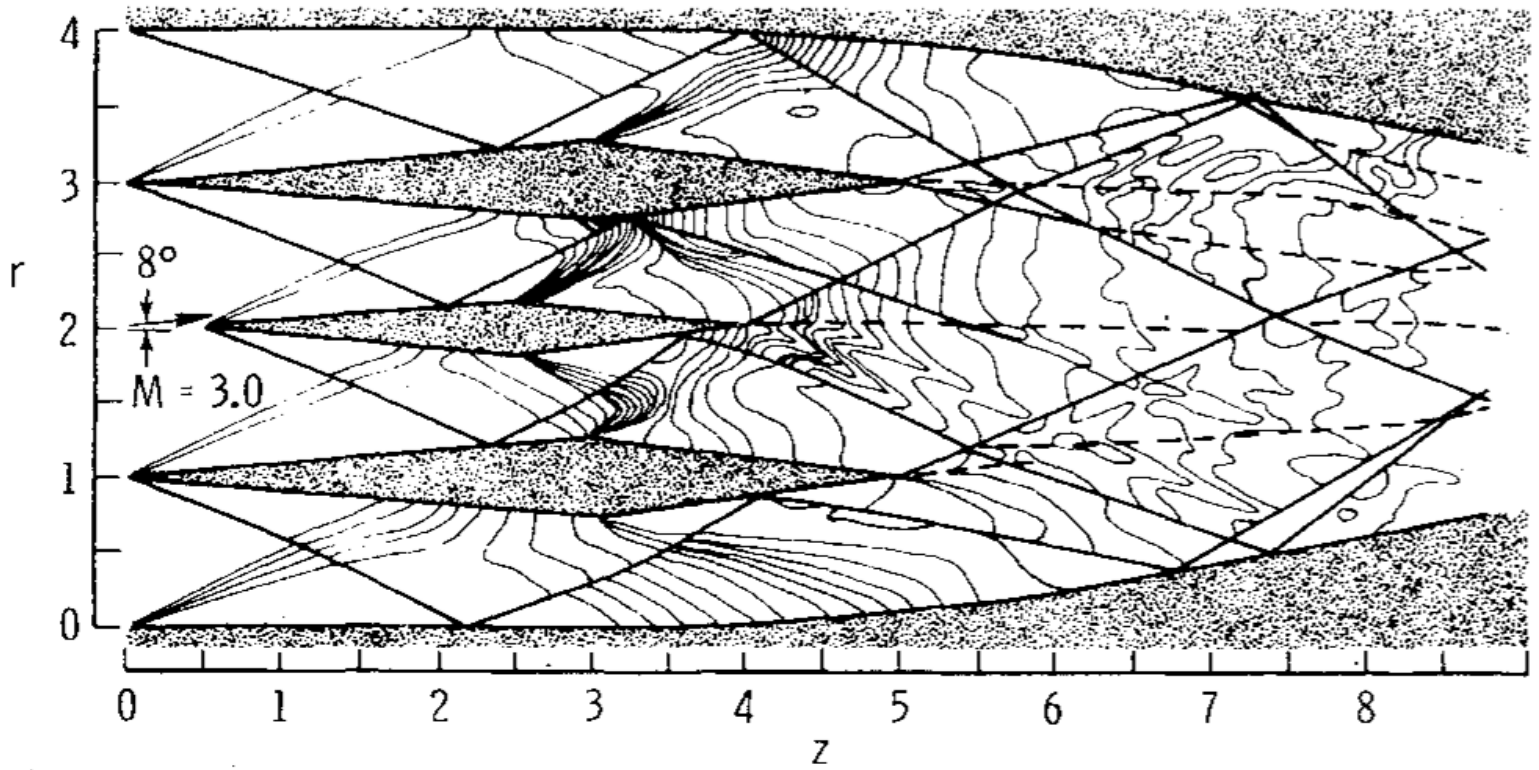


Fig. 11 Flowfield for a simulated scramjet, showing shock waves, vortex sheets and isobars.

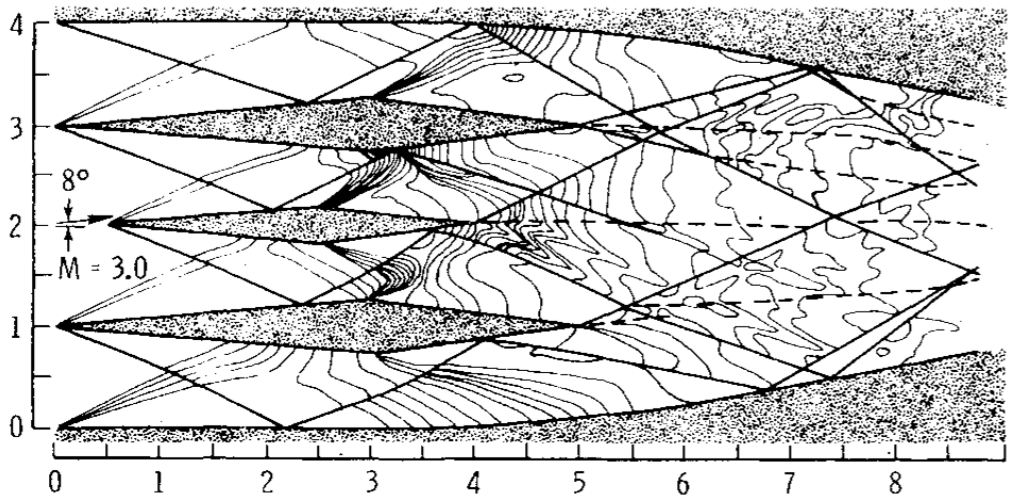
Defining the edges of the stream tubes

```
from math import sin, cos, radians
V1x = V1 * cos(radians(8.0))
V1y = V1 * sin(radians(8.0))

config.max_step_relax = 40

from eilmer.geom.xpath import XPath
path_0 = XPath().moveto(0,0).lineto(1,0).lineto(2,0)
path_0.lineto(3,0).lineto(4,0.02).lineto(5,0.1)
path_0.lineto(6,0.244).lineto(7,0.425).lineto(8,0.625).lineto(9,0.825)
path_1a = XPath().moveto(0,1).lineto(3,0.75).lineto(5,1).lineto(9,1.4)
path_1b = XPath().moveto(0,1).lineto(3,1.25).lineto(5,1).lineto(9,1.4)
path_2a = XPath().moveto(0,2).lineto(0.5,2).lineto(2.5,1.8).lineto(4,2).lineto(9,2)
path_2b = XPath().moveto(0,2).lineto(0.5,2).lineto(2.5,2.2).lineto(4,2).lineto(9,2)
path_3a = XPath().moveto(0,3).lineto(3,2.75).lineto(5,3).lineto(9,2.6)
path_3b = XPath().moveto(0,3).lineto(3,3.25).lineto(5,3).lineto(9,2.6)
path_4 = XPath().moveto(0,4).lineto(1,4).lineto(2,4)
path_4.lineto(3,4).lineto(4,4-0.02).lineto(5,4-0.1)
path_4.lineto(6,4-0.244).lineto(7,4-0.425).lineto(8,4-0.625).lineto(9,4-0.825)
```

- XPath can be built incrementally.
- Other segments: bezier2to, bezier3to
- XBezier also available

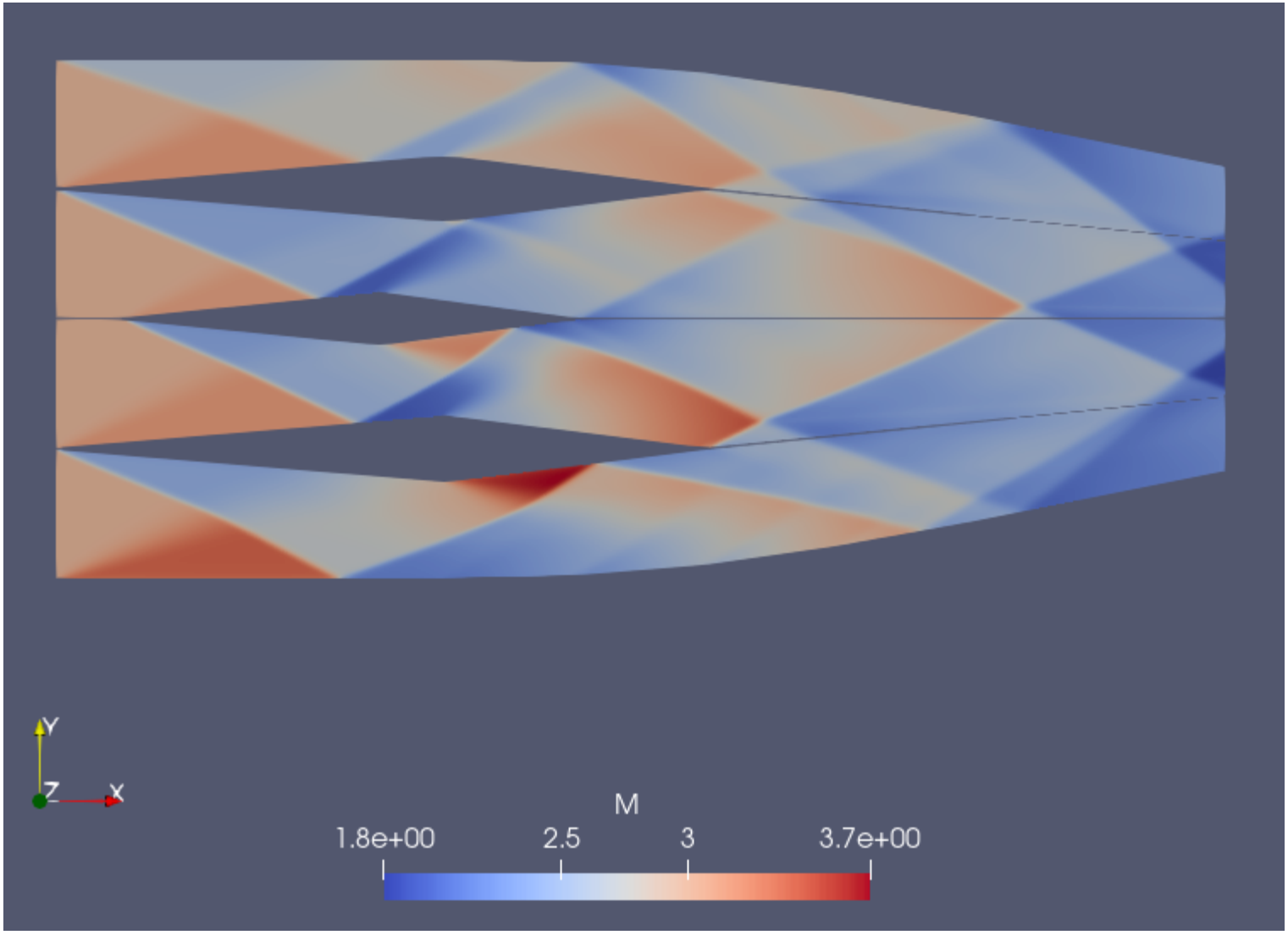
Defining the boundary conditions and stream tubes

```
def bc_0(x): return 0
def bc_1(x):
    if x < 5.0: return 0
    return 1
def bc_2(x):
    if x < 0.5: return 1
    if x < 4.0: return 0
    return 1
def bc_3(x):
    if x < 5.0: return 0
    return 1
def bc_4(x): return 0
```

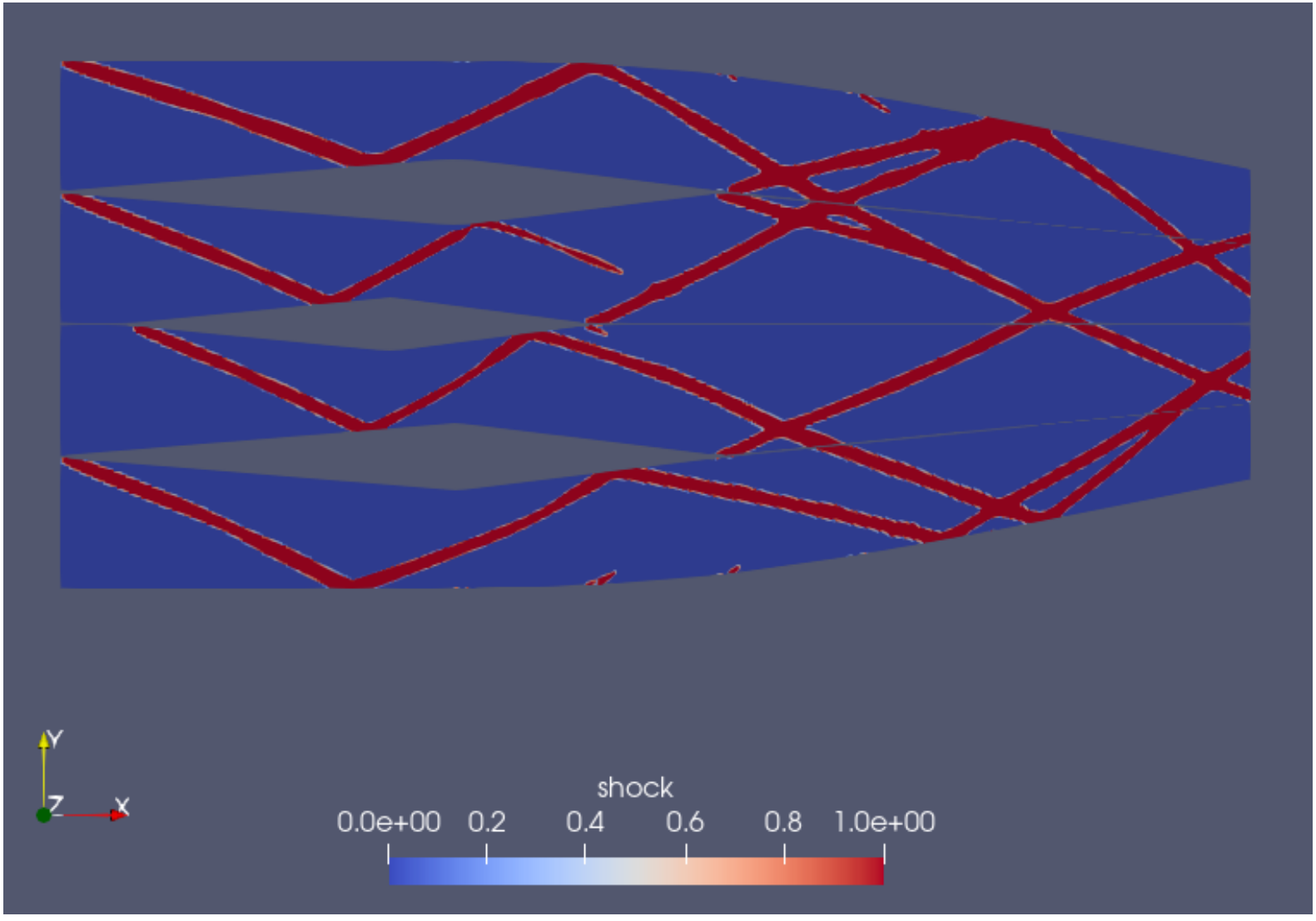



```
config.max_x = path_0.xs[-1]
config.dx = config.max_x/1000

st0 = StreamTube(gas=gas1, velx=V1x, vely=V1y,
                 y0=path_0, y1=path_1a, bc0=bc_0, bc1=bc_1, ncells=60)
st1 = StreamTube(gas=gas1, velx=V1x, vely=V1y,
                 y0=path_1b, y1=path_2a, bc0=bc_1, bc1=bc_2, ncells=60)
st2 = StreamTube(gas=gas1, velx=V1x, vely=V1y,
                 y0=path_2b, y1=path_3a, bc0=bc_2, bc1=bc_3, ncells=60)
st3 = StreamTube(gas=gas1, velx=V1x, vely=V1y,
                 y0=path_3b, y1=path_4, bc0=bc_3, bc1=bc_4, ncells=60)
```

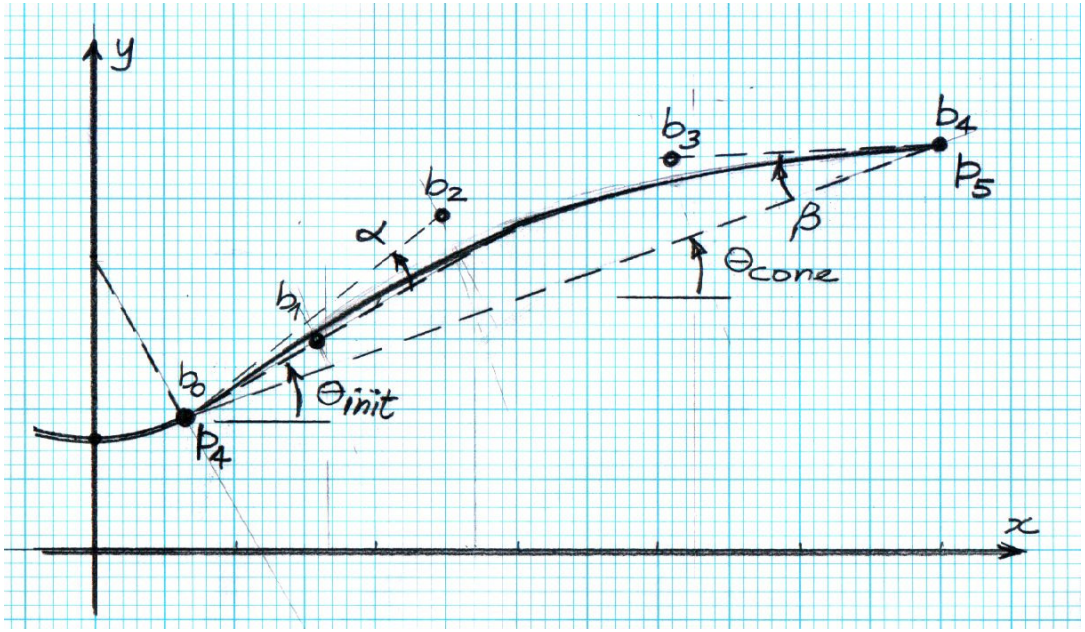
Puffin solution – Mach number



Puffin solution – shock detector

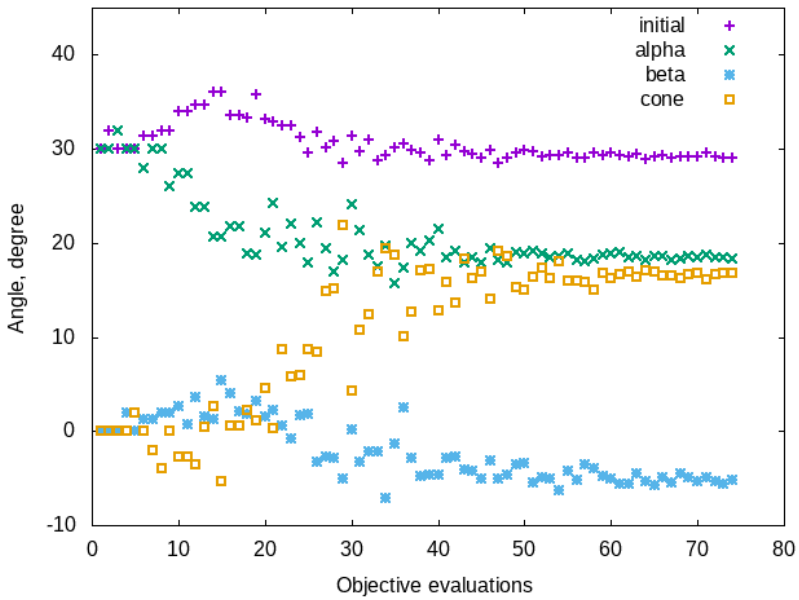


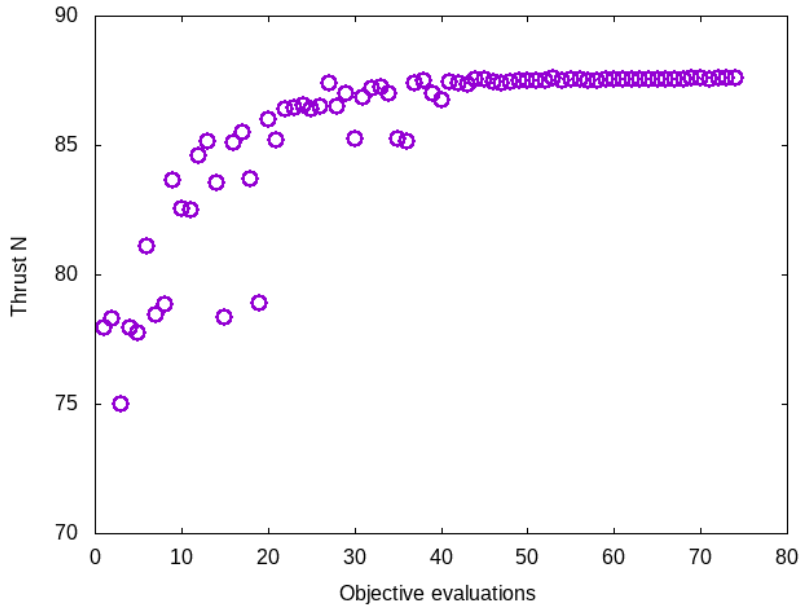
Optimisation study of a bell nozzle



- Eilmer4 case study at https://gdtk.uqcloud.net/docs/eilmer/technical-notes/
- Puffin run time 0.7 second per solution (with debug code) overall run time approximately 35 seconds.
- Eilmer4 run times are 85 seconds and 6035 seconds.

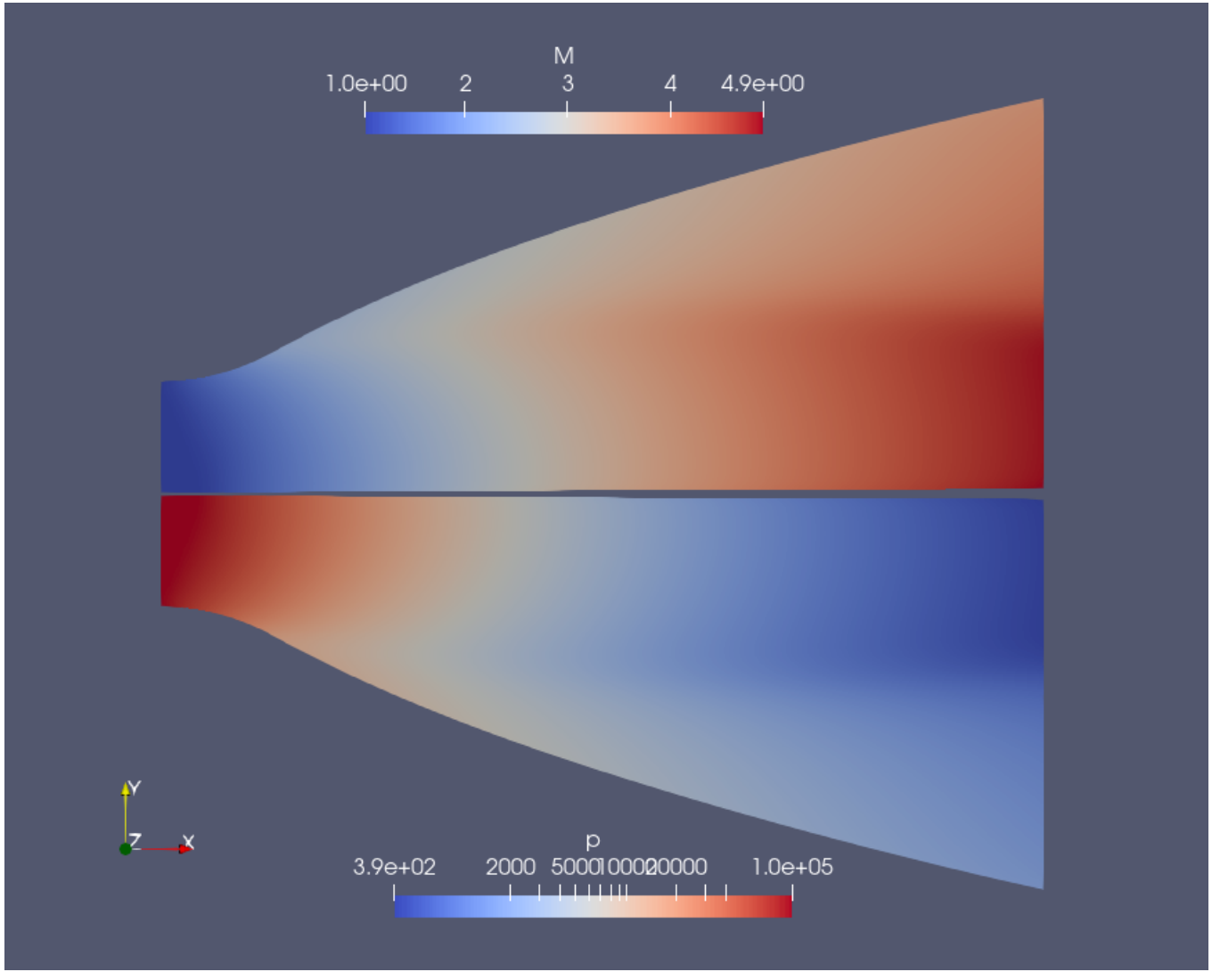
Optimisation progress – searching for maximum thrust





- Thrust is estimated from the pressure in the line of cells nearest the wall.

Flow field for maximum thrust



Are we there yet?

- So far, 90% of the work is done.
  - Core space-marching solver with a selection of flux calculators.
  - Flexible boundaries.
  - Complete preparation code built in Python.
  - Simple post-processing in Python, also.
  - Simple optimization code in Python, with parallel processing mode.
- There is about 90% yet to be done.
  - Convergence check for detecting steady flow in a slice.
  - Optimization and parallel loops over stream tubes.
  - Examples with chemically-reacting flow.
  - Documentation.
- Source code is available in the Gas Dynamics Toolkit, https://github.com/gdtk-uq/gdtk/tree/master/src/puffin
- The equivalent transient-flow code (Lorikeet) should be done in a month.