MB_CNS: A computer program for the simulation of transient compressible flows.

P. A. Jacobs

Department of Mechanical Engineering Report 10/96
The University of Queensland.

December 18, 1996

Abstract

The program *MB_CNS* is a CFD tool for the simulation of transient compressible flow in two-dimensional (planar or axisymmetric) geometries. It is based on a finite-volume formulation of the Navier-Stokes equations and has a shock-capturing capability through the use of a limited reconstruction scheme and an upwind-biased flux calculator. Subject to grid resolution and numerical diffusion issues, the code is capable of modelling flows that include shocks, expansions, shear layers and boundary layers.

The present code is a development of the single-block Navier-Stokes integrator *CNS4U* with the primary difference being the ability to handle a relatively complex flow geometry by decomposing it into several non-overlapping blocks.

This report describes the formulation of the code and a number of flow simulation examples. Further documentation is provided a set of hypertext pages and the source code itself.

Contents

Nomunclature, Units

A	: area, m^2
a	: sound speed, m/s
C_p, C_v	: specific heats, J/(kg.K)
E	: total specific energy, J/kg
e	: specific internal energy, J/kg
$\overline{F}$	: array of flux terms
f	: species mass fraction
h	: specific enthalpy, J/kg
$\hat{i}, \hat{j}$	: unit vectors for the cartesian coordinates
k	: coefficient of thermal conductivity
M	: Mach number
n	: direction cosine
$\hat{n}, \hat{p}$	: unit vectors for the cell interface
P	: point in the (x, y)-plane
p	: pressure, Pa
Q	: array of source terms
q	: heat flux, W/m^2
R	: gas constant, J/(kg.K)
r	: radial coordinate, m
r, s	: normalised coordinates
S	: control surface of the cell
T	: temperature, degree K
t	: time, s; independent parameter for the Bezier curves
U	: array of conserved quantities
u	: velocity, m/s
V	: cell volume, m^3
x, y, z	: cartesian coordinates, m
ρ	: density, kg/m^3
μ, λ	: first and second coefficients of viscosity, Pa.s
γ	: ratio of specific heats

Subscripts, Superscripts

i	: inviscid
is	: species index
L, R	: Left, Right
n	: normal to the cell interface
p	: tangent to the cell interface
v	: viscous
x, y, z	: coordinate directions
$*$	: intermediate state in the solution of the Riemann problem

1 Introduction

It is generally accepted [1] that Computational Fluid Dynamics (CFD) analysis will eventually replace wind tunnels. The program *MB_CNS* is a CFD tool for the simulation of transient compressible flow in two-dimensional (planar or axisymmetric) geometries. The code is intended primarily for the simulation of the transient flows experienced in shock tunnels and expansion tubes.

MB_CNS is based on a cell-centred finite-volume formulation of the Navier-Stokes equations and has a shock-capturing capability through the use of a limited reconstruction scheme and an upwind-biased flux calculator. The governing equations are expressed in integral form over arbitrary quadrilateral cells with the time rate of change of conserved quantities in each cell specified a summation of the fluxes (of mass, momentum and energy) through the cell interfaces. Subject to grid resolution and numerical diffusion issues, the code is capable of modelling flows that include shocks, expansions, shear layers and boundary layers.

The present code is a development of the single-block Navier-Stokes integrator *CNS4U* described in Ref. [1]. That original code was written with the intention of porting it to a distributed-memory parallel computer. It was written in C and with the data for the entire block of cells packaged a single data structure. The functions were then written so that they operated on the flow data contained within that data structure without the need for other information. Thus, extending the code to handle several blocks was relatively simple. The differences between the present code and the original include the following.

- The new code assumes that the flow domain is composed of a number of non-overlapping patches (or blocks). Hence, the name *MB_CNS* is an acronym for Multiple-Block Compressible Navier-Stokes solver.
- The geometry of the flow domain is specified as a Bezier polyline description of the block boundaries.
- The code now uses a flux-based update of the cell-averaged equations rather than explicitly using the cell-interface flow states.
- Alternative flux calculators are available. These include EFM [2] and AUSM [3].
- The gas may consist of several components (or species).

[1] Once upon a time I was told that this would happen, but I can't remember who told me.

- *MB_CNS* runs in parallel on a Silicon Graphics Power Challenge. It also runs on just about any computer with a C compiler.
- The code is packaged with a crude, but functional, postprocessing capability. This includes output in TECPLOT format, contour plots of flow quantities and extraction of flow data along cell-index directions.

The remainder of this report describes the formulation of the code and a number of examples of use. The second level of the documentation is the set of hypertext pages which include the details of unpacking and installing the code, a description of input data file formats and a guide to the content of the source files. The third (and final) level of documentation is the commented source code itself. The key to variable names is the set of data structure definitions in the file *mb_cns.h*. This file should be browsed before reading any of the C source files.

2 Formulation

The original ICASE report [1] describes the formulation of the governing equations, the flow field discretisation as a single block of cells and the time-stepping scheme used to integrate the discrete equations. Most aspects of the formulation (and the code modules implementing them) have remained essentially unchanged. Changes have been made in the way the cell information and the fluxes are computed and stored, the handling of multiple species, and, of course, the handling of multiple blocks. If some aspect of the code is not described here, there is a good chance that the description given in the ICASE report is still relevant.

2.1 Governing Equations

The starting point for the governing equations encoded within *MB_CNS* is the set Navier-Stokes equations which, in integral form, can be expressed as

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

where V is the cell's volume, S is the bounding (control) surface and $\hat{n}$ is the outward-facing unit normal of the control surface. For two-dimensional flow, V is the volume per unit depth in the z-direction and A is the area of the cell boundary per unit depth in z. The array of conserved quantities (per unit volume) is

$$U = \begin{bmatrix} \rho \\ \rho u_x \\ \rho u_y \\ \rho E \\ \rho f_{is} \end{bmatrix} \quad . \tag{2}$$

These elements represent mass density, x-momentum per volume, y-momentum per volume, total energy per volume and mass density of species is. The flux vector is divided into inviscid and viscous components and the inviscid component, in two dimensions, is

$$\overline{F}_i = \begin{bmatrix} \rho u_x \\ \rho u_x^2 + p \\ \rho u_y u_x \\ \rho E u_x + p u_x \\ \rho f_{is} u_x \end{bmatrix} \hat{i} + \begin{bmatrix} \rho u_y \\ \rho u_x u_y \\ \rho u_y^2 + p \\ \rho E u_y + p u_y \\ \rho f_{is} u_y \end{bmatrix} \hat{j} \quad . \tag{3}$$

The viscous component is

$$\overline{F}_v = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{yx} \\ \tau_{xx} u_x + \tau_{yx} u_y + q_x \\ \rho f_{is} \mu_{x,is} \end{bmatrix} \hat{i} + \begin{bmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ \tau_{xy} u_x + \tau_{yy} u_y + q_y \\ \rho f_{is} \mu_{y,is} \end{bmatrix} \hat{j} \quad , \tag{4}$$

where the viscous stresses are

$$\begin{aligned} \tau_{xx} &= 2\mu\frac{\partial u_x}{\partial x} + \lambda\left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y}\right) \quad , \\ \tau_{yy} &= 2\mu\frac{\partial u_y}{\partial y} + \lambda\left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y}\right) \quad , \\ \tau_{xy} = \tau_{yx} &= \mu\left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\right) \quad , \end{aligned} \tag{5}$$

and the viscous heat fluxes are

$$\begin{aligned} q_x &= k\frac{\partial T}{\partial x} + \rho\sum h_{is} f_{is}\mu_{x,is} \quad , \\ q_y &= k\frac{\partial T}{\partial y} + \rho\sum h_{is} f_{is}\mu_{y,is} \quad . \end{aligned} \tag{6}$$

Currently, the code convects species without considering their diffusion (*i.e.* $\mu_{x,is} = 0$, $\mu_{y,is} = 0$). For flow without heat sources or chemical effects, the source terms in Q are set to zero.

The conservation equations are supplemented by the equation of state giving pressure as a function of density, specific internal energy and species mass fractions

$$p = p(\rho, e, f_{is}) \quad . \tag{7}$$

The coefficients of viscosity μ, λ and heat conduction k are also allowed to vary with the fluid state. See Appendix B for a description of the gas models implemented in the code.

2.2 Axisymmetric Geometries

For axisymmetric flow, the geometry is defined such that x-axis is the axis of symmetry and y is the radial coordinate. There are relatively minor changes to the governing equations which include:

- dA is now computed as interface area per radian;
- dV is now cell volume per radian;
- The shear stresses τ_{xx}, τ_{yy} have a extra term so that

$$\begin{aligned} \tau_{xx} &= 2\mu\frac{\partial u_x}{\partial x} + \lambda\left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{u_y}{y}\right) \quad , \\ \tau_{yy} &= 2\mu\frac{\partial u_y}{\partial y} + \lambda\left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{u_y}{y}\right) \quad , \end{aligned} \tag{8}$$

- and there is a pressure and shear-stress contribution to the radial momentum equation which can be expressed as an effective source term

$$Q = \begin{bmatrix} 0 \\ 0 \\ (p - \tau_{\theta\theta}) A_{xy}/V \\ 0 \\ 0 \end{bmatrix} \quad , \tag{9}$$

where A_{xy} is the projected area of the cell in the (x, y)-plane and

$$\tau_{\theta\theta} = 2\mu \frac{u_y}{y} + \lambda \left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{u_y}{y} \right) \quad . \tag{10}$$

2.3 Discretised Equations and Flux Calculation

The conservation equations are applied to straight-edged quadrilateral cells for which the boundary, projected onto the (x, y)-plane, consists of four straight lines. These lines (or cell interfaces) are labelled North, East, South and West and the integral equation is approximated as the algebraic expression

$$\frac{dU}{dt} = -\frac{1}{V} \sum_{NESW} (\overline{F}_i - \overline{F}_v) \cdot \hat{n} \; dA + Q \quad , \tag{11}$$

where U and Q now represent cell-averaged values. The code updates the cell-average flow quantities each time step by

1. applying inviscid boundary conditions or exchanging data at boundaries of each block as appropriate;

2. reconstructing (or interpolating) the flow field state on both sides of each interface;

3. computing the inviscid fluxes at interfaces as $(\overline{F}_i \cdot \hat{n})$ using a one-dimensional flux calculator such as a Riemann solver [4], the equilibrium-flux method [5, 2] or AUSM [3];

4. applying viscous boundary conditions at solid walls;

5. computing the viscous contribution to the fluxes as $(\overline{F}_v \cdot \hat{n})$; and finally

6. updating the cell-average values using equation (11).

This whole process will be applied in two stages if predictor-corrector time stepping is used.

When computing the inviscid fluxes at each interface, the velocity field is rotated into a local (n, p)-coordinate system with unit vectors

$$\begin{aligned} \hat{n} &= n_x \, \hat{i} + n_y \, \hat{j} \quad , \\ \hat{p} &= p_x \, \hat{i} + p_y \, \hat{j} \quad , \end{aligned} \tag{12}$$

normal and tangental to the cell interface respectively. We have chosen the tangential direction $p_x = -n_y$ and $p_y = n_x$. The normal and tangential velocity components

$$\begin{aligned} u_n &= n_x \, u_x + n_y \, u_y \quad , \\ u_p &= p_x \, u_x + p_y \, u_y \quad , \end{aligned} \tag{13}$$

are then used, together with the other flow properties either side of the interface, to compute the fluxes

$$\begin{bmatrix} F_{mass} \\ F_{n-momentum} \\ F_{p-momentum} \\ F_{energy} \\ F_{species-is} \end{bmatrix} = \begin{bmatrix} \rho u_n \\ \rho u_n u_n + p \\ \rho u_n u_p \\ \rho u_n E + p u_n \\ \rho u_n f_{is} \end{bmatrix} \quad , \tag{14}$$

in the local reference frame. These are then transformed back to the (x, y)-plane as

$$\overline{F} \cdot \hat{n} = \begin{bmatrix} F_{mass} \\ F_{x-momentum} \\ F_{y-momentum} \\ F_{energy} \\ F_{species-is} \end{bmatrix} = \begin{bmatrix} F_{mass} \\ F_{n-momentum} n_x + F_{p-momentum} p_x \\ F_{n-momentum} n_y + F_{p-momentum} p_y \\ F_{energy} \\ F_{species-is} \end{bmatrix} \quad . \tag{15}$$

2.4 Handling Multiple-Blocks and Parallelisation

The data arrays for each block are dimensioned such that there is a buffer region, two cells deep, around the active cells. The active cells completely define the flow domain covered by the block and the buffer region contains ghost cells which are used to hold a copy of the flow information from adjacent blocks or to implement the boundary conditions.

For a boundary common to two blocks, the ghost cells in the buffer region of each block overlap the active cells of the adjacent block. The only interaction that occurs between blocks is the exchange of boundary data, prior to the reconstruction phase of each time step. The exchange of cell-average data along the block boundaries takes place as a direct copy from the active-cell of one block to the ghost-cell of the other block. Thus, the cells along the common boundary of each block must match in both number and position. Some logic is used within the exchange routines to set the appropriate indexing direction for each boundary. Refer to the functions in *mb_exch.c* for more detail.

The information on the connections between block boundaries is stored in a (global) connectivity array. For each boundary on each block, this array stores the identity of the adjacent block and the name of the connecting boundary on the adjacent block. To keep the code simple, the two-way nature of the exchange is explicitly stored in the connectivity array. Thus, if the East boundary of block 0 is connected to the West boundary of block 1, the array stores the information for that relation as part of the data for block 0 *and* it also stores the corresponding (inverted) information as part of the data for block 1.

Except for this block to block communication (and the occasional checking of time step magnitudes), the rest of the calculation can be done independently for all blocks. Thus, the algorithm is fairly easy to implement on a MIMD parallel computer. A shared-memory machine, such as the Silicon Graphics Power Challenge, is a particularly simple host architecture.

The program is written as an (outer) time-stepping loop which does the calculation of each time step as a number of phases. For each phase, there is a loop which calls a function (or set of functions) to perform the same operations on each block. Only these loops (over the blocks) need to be flagged for parallelisation. The self-contained packaging of the data for each block ensures that there are no memory conflicts. Even the block-boundary data exchange, which involves copying from another block's active cells to the current block's ghost cells, can be parallelised without special coding on a shared memory machine. The only difference between the serial version of the code and the parallel version are a few compiler directives added to the *main()* function to indicate (to the *Power C Analyser*) that loops over the blocks are safe to run concurrently.

Parallelisation on a distributed machine is more involved. As part of his PhD project, Andrew McGhee [6] is adapting the code so that it runs on distributed memory computers (such as the IBM SP2 and the the Fujitsu VP3300). All of the data storage for a single block and the functions which process the data within a single block remain unchanged. However, in a Single-Program-Multiple-Data model, there are now several copies of the program running independently on separate processors. Each copy of the program handles the processing for a single block but, to exchange block-boundary data, must communicate with the other programs for adjacent blocks. The communication and synchronisation tasks are handled via a standard message passing library (MPI).

2.5 Block-Boundary Specification and Grid Generation

The domain covered by each block is discretized as a structured grid of quadrilateral cells with grid points internal to the block being obtained from the boundary data by

transfinite interpolation (or a linear Coons' surface [7]). Referring to Fig. 1, internal points $P = [x,\ y]^T$ are calculated as

$$\begin{aligned} P(r,s) &= (1-s)\ P_S(r) + s\ P_N(r) + (1-r)\ P_W(s) + r\ P_E(s) \\ &\quad -(1-s)(1-r)\ P_S(0) - (1-s)r\ P_S(1) \\ &\quad -s(1-r)\ P_N(0) - s\ r\ P_N(1) \quad , \end{aligned} \tag{16}$$

where $P_N(r),\ P_S(r),\ 0 \le r \le 1$ are the North and South boundary curves respectively and $P_W(s),\ P_E(s),\ 0 \le s \le 1$ are the West and East boundary curves. For consistency, the end points of the curves must coincide such that $P_S(0) = P_W(0)$, $P_S(1) = P_E(0)$, $P_N(0) = P_W(1)$ and $P_N(1) = P_E(1)$.

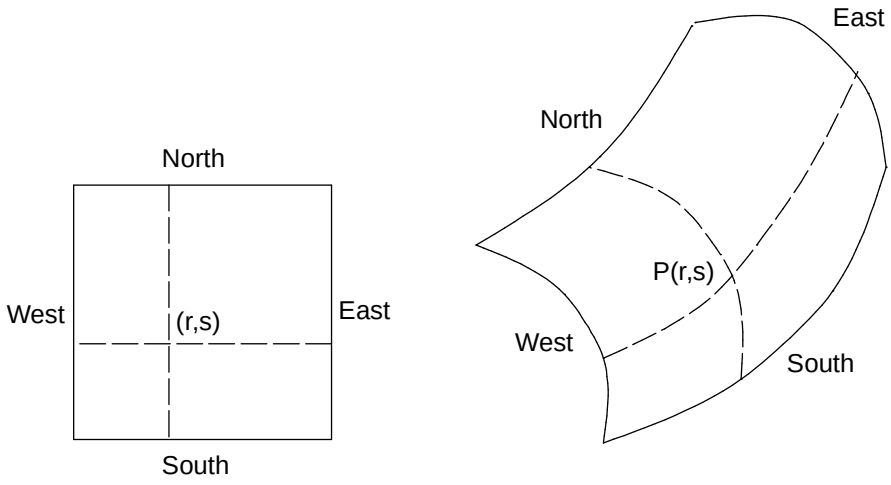

Figure 1: Interpolation of internal grid points from the boundary curves.

Although this method is fast and can easily accommodate grid clustering via one of Robert's stretching transformations [8], it requires "well behaved" boundary curves in order to produce grids of reasonable quality. If grid smoothness is required and the grid is not clustered, an iterative (Laplacian) grid smoother is available as an option in the grid preparation program. This function was provided by Andrew McGhee.

Following Ref. [9], each boundary curve is defined as a Bezier polyline consisting of n_p Bezier segments of degree 3. Segments are defined by four control points $P_0 = [x_0, y_0]^T$, P_1, P_2 and P_3 and points on the curve are given by

$$P(t) = (1-t)^3\ P_0 + 3(1-t)^2 t\ P_1 + 3(1-t)t^2\ P_2 + t^3\ P_3 \quad , \tag{17}$$

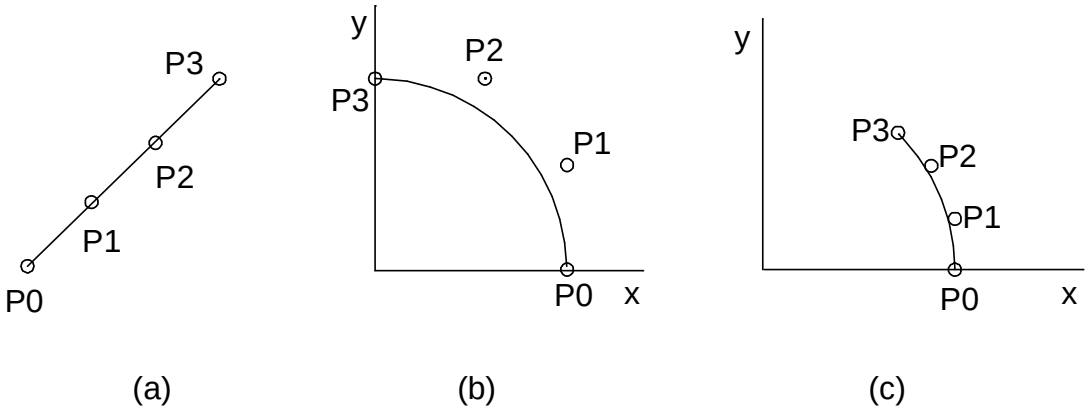

Figure 2: Frequently used curves with Bezier control points: (a) straight line; (b) quarter circle; (c) 45 degree sector.

where $0 \leq t \leq 1$. When the boundary curve consists of several Bezier segments, the parameter (r or s) is scaled with the (approximate) lengths of the segments so that points distributed uniformly in parameter space transform to uniformly distributed points in (x, y)-space. See the code module *bezier.c* for further details.

This form of boundary definition has been chosen for the convenience of modelling arbitrary shapes. Some frequently used curves are shown in Fig. 2. For the straight segment

$$P_1 = P_0 + \frac{1}{3}(P_3 - P_0) \quad , \quad P_2 = P_0 + \frac{2}{3}(P_3 - P_0) \quad , \tag{18}$$

The quarter circle can be modelled approximated as

$$P_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \ , \ P_1 = \begin{bmatrix} 1 \\ k \end{bmatrix} \ , \ P_2 = \begin{bmatrix} k \\ 1 \end{bmatrix} \ , \ P_3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \ , \ k = \frac{4}{3}(\sqrt{2} - 1) \ , \tag{19}$$

with a maximum error of 0.027% (See Ref. [10], p177). For modelling flows around spherically blunted cones and cylinders it is also convenient to model a 45 degree sector as

$$P_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \ , \ P_1 = \begin{bmatrix} 1 \\ k \end{bmatrix} \ , \ P_2 = \begin{bmatrix} \frac{(1+k)}{\sqrt{2}} \\ \frac{(1-k)}{\sqrt{2}} \end{bmatrix} \ , \ P_3 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \ , \ k = 0.265216 \ , \tag{20}$$

which has a maximum error of 0.00042%. These control points have been selected to give the correct locations and tangents at the segment end points and to give the correct location of the midpoint of the segment (*i.e.* at $t = 0.5$).

3 Examples of Use

Although the following examples exercise various capabilities of *MB_CNS*, they are not so much a demonstration of the validity of the code as they are a starting point for applying the code to new problems.

3.1 One-Dimensional Helium-Air Shock Tube

This is the smallest example (in terms of the computational time required to obtain a solution) and so is a good place to start to see if *MB_CNS* has been installed correctly. The input parameter and Bezier files are supplied as part of the source code package. Look for the files in the *mb_cns/examples/sod2* directory.

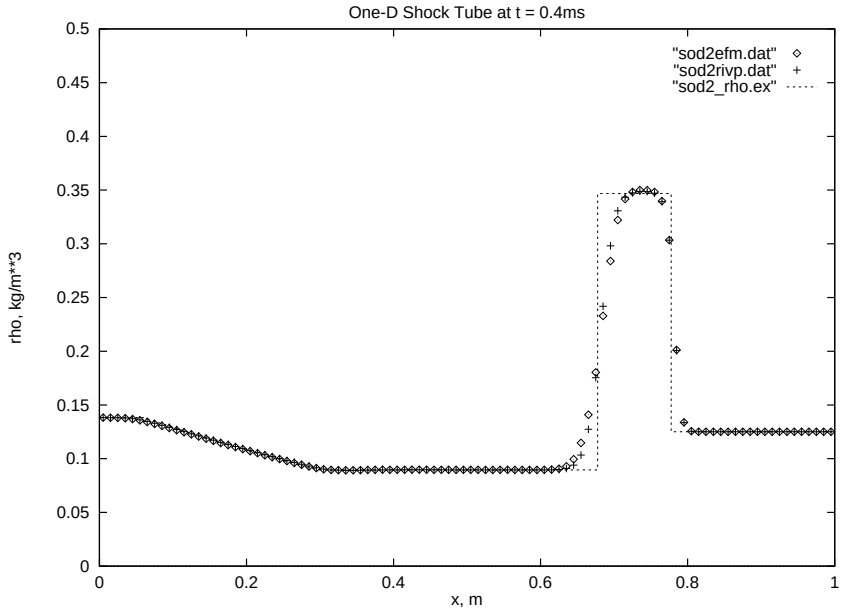

Figure 3: Density profiles for the one-dimensional shock tube problem. Symbols correspond to the discrete solutions based on the EFM and Riemann-solver flux calculators. The dashed line indicates the exact solution.

The flow domain is a rectangle $0 \leq x \leq 1.0$m, $0 \leq y \leq 0.1$m and consists of two blocks, each with 50×2 cells. Block 0 covers $0 \leq x \leq 0.5$m and initially contains helium with conditions

$$\rho = 0.1382 \text{ kg/m}^3, \quad u_x = 0, \quad u_y = 0, \quad e = 1.085 \times 10^6 \text{ J/kg},$$

$$p = 100 \text{ kPa}, \quad T = 348.3 \text{ K}.$$

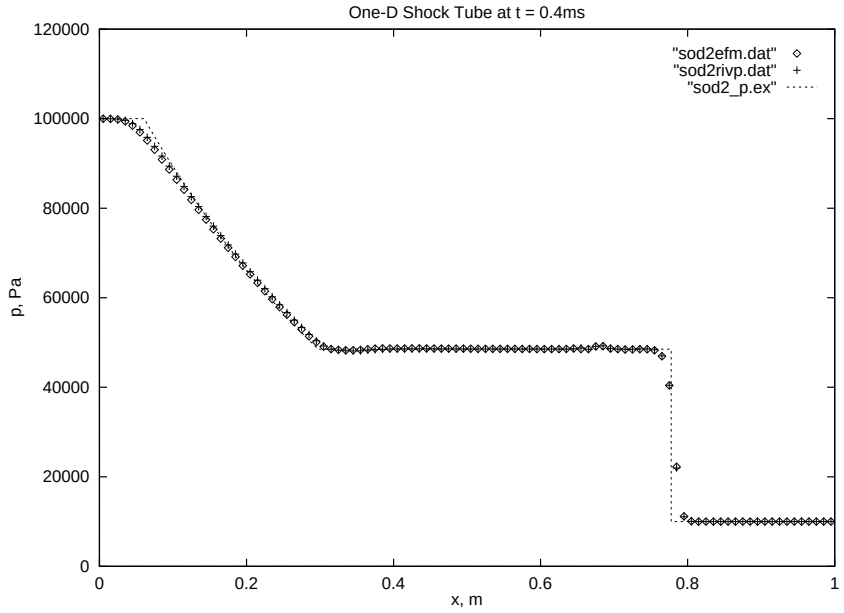

Figure 4: Pressure profiles for the one-dimensional shock tube problem. Symbols correspond to the discrete solutions based on the EFM and Riemann-solver flux calculators. The dashed line indicates the exact solution.

Block 1 covers the right half of the domain and initially contains air with conditions

$$\rho = 0.125 \text{ kg/m}^3, \quad u_x = 0, \quad u_y = 0, \quad e = 2.0 \times 10^5 \text{ J/kg},$$

$$p = 10 \text{ kPa}, \quad T = 287.5 \text{ K}.$$

The boundary conditions are that the East boundary of block 0 is connected to the West boundary of block 1 and all other boundaries are solid (reflective) walls.

At $t = 0$, the hypothetical diaphragm between the blocks is removed and the high pressure helium expands into the right half of the domain, compressing the air via a shock wave. Figures 3, 4 and 5 show the density, pressure and mass-fraction profiles along the domain at $t \approx 0.4$ms. High-order (MUSCL) reconstruction and a CFL number of 0.5 have been used during the integration of the flow equations. The solutions for both the Riemann-solver flux calculator and the EFM flux calculator are compared with the exact solution. Although the grid resolution is coarse, both methods have captured the shock reasonably well. The contact surface, however, is spread over 6-8 cells because of the numerical diffusion. Of the two flux calculators, EFM is computationally cheaper (as shown in Table 1) but is more diffusive. The strong diffusive nature of the EFM flux calculator is not clear in this one-dimensional calculation but it is more apparent when calculating two-dimensional flows with viscous boundary layers.

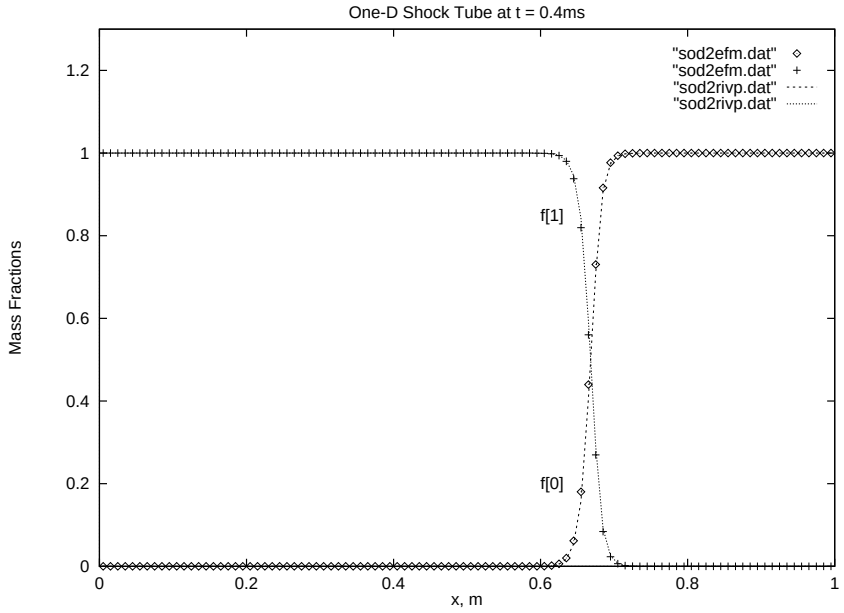

Figure 5: Mass fraction profiles for the one-dimensional shock tube problem. Symbols correspond to the discrete solution based on the EFM flux calculator and the lines indicate the solution based on the Riemann-solver flux calculator. Species 0 is air and species 1 is helium.

3.2 Viscous Flow Along a Cylinder

This case (*mb_cns/examples/cyl50*) computes the flow for a supersonic laminar boundary layer growing along a hollow cylinder. It was used in the original report to verify the implementation of the viscous and axisymmetric terms in the code.

The flow geometry consists of a hollow cylinder, 1.0m long with radius 0.005m, aligned with the x-axis. The flow domain, discretised by a single-block grid, is defined by a quadrilateral with corners (1.0, 0.005), (1.0, 0.7), (0.0, 0.06), (0.0, 0.005). This region is shaped to capture the weak leading-edge-interaction shock while concentrating cells near the cylinder surface for the early part of the boundary layer development. The grid consists of 50×50 cells which are clustered toward the leading edge of the cylinder and (even more strongly in the y-direction) toward the cylinder surface.

The free stream is a uniform supersonic flow of air, modelled as a perfect gas with conditions

$$\rho = 0.00404 \text{ kg/m}^3, \quad u_x = 597.3 \text{ m/s}, \quad u_y = 0, \quad e = 1.592 \times 10^5 \text{ J/kg},$$

$$T = 222 \text{ K}, \quad p = 257 \text{ Pa}, \quad M = 2.$$

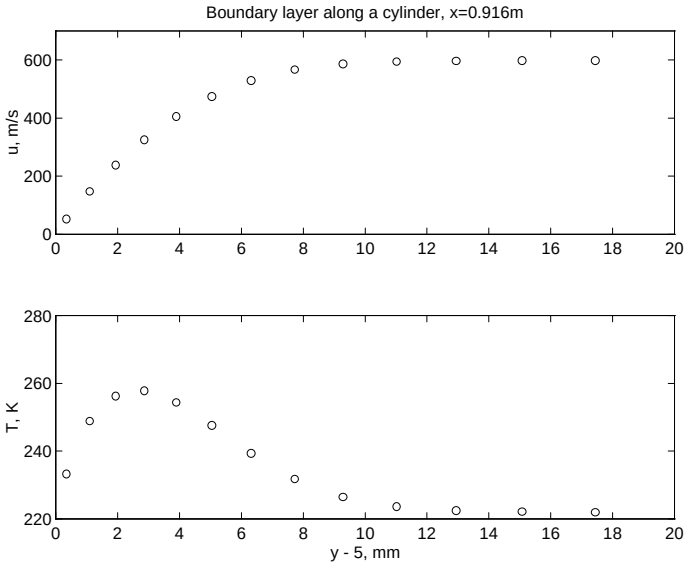

Figure 6: Velocity and temperature profiles at $x = 0.916$ m for viscous flow along a cylinder.

This free stream condition is applied to the West and North boundaries, the East boundary is a supersonic outflow boundary and the South boundary (along the cylinder surface) is a no-slip boundary with temperature fixed at $T = 222$ K. The Reynolds number at the end of the plate is 1.65×10^5.

Initially, the flow throughout the block is set at the same conditions as the free stream and the governing equations are integrated in time with the Euler time-stepping scheme with a CFL number of 0.8. The Riemann solver flux calculator is used together with high-order reconstruction. Figure 6 shows the x-velocity and temperature profiles through the boundary layer at $x = 0.916$m, 48 cells from the leading edge of the cylinder.

This case requires a fairly large computational effort. On a DEC Alphastation (model 250 4/266), the CPU time required is 5.45 hours for 131480 time steps. This gives the CPU time per cell per Euler time-step as 60μs. The CPU time required for predictor-corrector time stepping would be approximately twice this value.

3.3 Inviscid Flow Over a Cone

As well as being a test case that requires less CPU time, supersonic flow over a cone (*mb_cns/examples/cone20*) also provides a more interesting set of data for flow visualisation. Figure 7 shows the pressure contours for $M = 1.5$ flow over 20^o cone a short time

after the starting shock has left the flow domain.

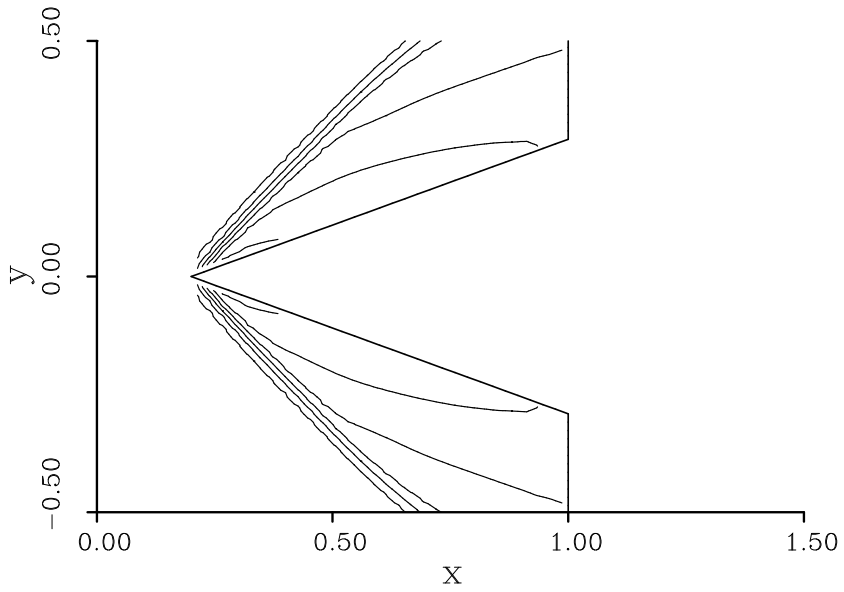

Figure 7: Pressure contours ($0 \leq p \leq 150$kPa, $\Delta p = 10$kPa) for supersonic $M = 1.5$ flow over a 20^o cone. Distances are in metres.

The flow domain consists of two blocks. Block 0, discretised by 10×40 cells, covers the flow domain upstream of the cone vertex while block 1, with 30×40 cells, covers the part of flow domain adjacent to the cone surface. The cone apex is located at $x = 0.2$m and its base is at $x = 1.0$m.

The free stream is a uniform supersonic flow of air, modelled as a perfect gas with conditions

$$\rho = 0.3028 \text{ kg/m}^3, \quad u_x = 1000 \text{ m/s}, \quad u_y = 0, \quad e = 7.913 \times 10^5 \text{ J/kg},$$

$$T = 1103 \text{ K}, \quad p = 95.84 \text{ kPa}, \quad M = 1.5$$

This free stream condition is applied to the West boundary of block 0. The East boundary of block 1 is an extrapolation (supersonic outflow) boundary and the East boundary of block 0 is connected to the West boundary of block 1. All South and North boundaries are inviscid (reflective) walls. The inflow conditions are applied at $t = 0$ when the conditions throughout the interior of the domain are

$$\rho = 0.0682 \text{ kg/m}^3, \quad u_x = 0 \text{ m/s}, \quad u_y = 0, \quad e = 2.183 \times 10^5 \text{ J/kg}.$$

Thus, the flow starts with the propagation of a shock through the flow domain and over the cone. The simulation here has been stopped at time step 300 with $t \approx 1.25$ms, shortly

after the starting shock has exited through the East boundary of block 1. The flow has not reached steady state as indicated by the curved shock attached to the apex of the cone. In the steady-state limit, this shock will be straight and be at an angle of $\beta \approx 49^o$ to the x-axis.

Table 1: CPU times per cell, per predictor-corrector time-step required for various computers and test cases.

Computer	OS & Compiler	Test Case	μs/cell/p-c-step RIVP fluxes	μs/cell/p-c-step EFM fluxes
Toshiba T2130CS	OS/2, GNU C	sod2	1230	1000
		cone20	767	554
Pentium-133 Clone	OS/2, GNU C	cone20	181	
		hemi2	195	156
DEC Alphastation	OSF1, DEC C	cone20	100	64
Model 250 4/266		hemi2		81
		hemi2_short	113	82
SUN Ultra SPARC 1	Solaris, GNU C	cone20	77	62
		hemi2_short	86	70
SUN Ultra SPARC 2	Solaris, GNU C	cone20	56	
SGI Power Challenge	IRIX, Power C			
1 R8000 processor		hemi2_short	90	77
6 processors		hemi2_short	19.1	16.5
IBM SP2	MPL, xlc			
1 processor		Apollo	90	
4 processors		Apollo	24	

Because, the resolution is coarse and the number of time steps is limited to 300 (in the parameter file supplied with the source code package), this case is a convenient time trial. On a Toshiba Satellite laptop computer (Model T2130CS with Intel 486 processor) the run time is approximately 266 seconds (EFM flux calculator) when using the GNU C compiler and the OS/2 operating system. The time per cell per predictor-corrector time step is thus 554μs. CPU times for other computers (and a variety of test cases) are shown in Table 1. Note that the figures in this table are somewhat "rubbery". Cache size, temporary array size and compiler optimisations all seem to play important (and confounding) roles. At the time of writing this report, the Riemann-solver flux calculator

used large temporary arrays and was coded to suit vector-processing computers. However, it was actually used as a scalar function by the rest of the code. The penalty for this (measured relative to the EFM flux calculator) depended on the computer.

3.4 Transient Flow Over a Heat-Flux Probe

This example (*mb_cns/examples/hemi2*) was motivated by the need to know the transient flow conditions near the stagnation point of a heat-flux probe for a short time after the arrival of a shock wave [11]. The probe is essentially a spherically-blunted cylinder of diameter 10 mm.

Initially, the gas (air) surrounding the probe is stationary with pressure 101 kPa and temperature 293 K. Flow over the probe is started by the arrival of a shock wave with a speed of 640 m/s (and shock Mach number of 1.865). For the simulations discussed here, the air is modelled as a perfect gas with $R = 287$ J/(kg.K) and $\gamma = 1.4$. Thus, the initial density is $\rho = 1.201$ kg/m^3 and specific internal energy is $e = 2.102 \times 10^5$ J/kg.

The simulation makes use of multiple blocks to decompose the flow domain around the probe into relatively simple regions that can be easily handled with the algebraic grid generator in *MB_PREP*. The use of multiple blocks also enables us to use a parallel computer to reduce the "wall clock" time needed for a particular calculation.

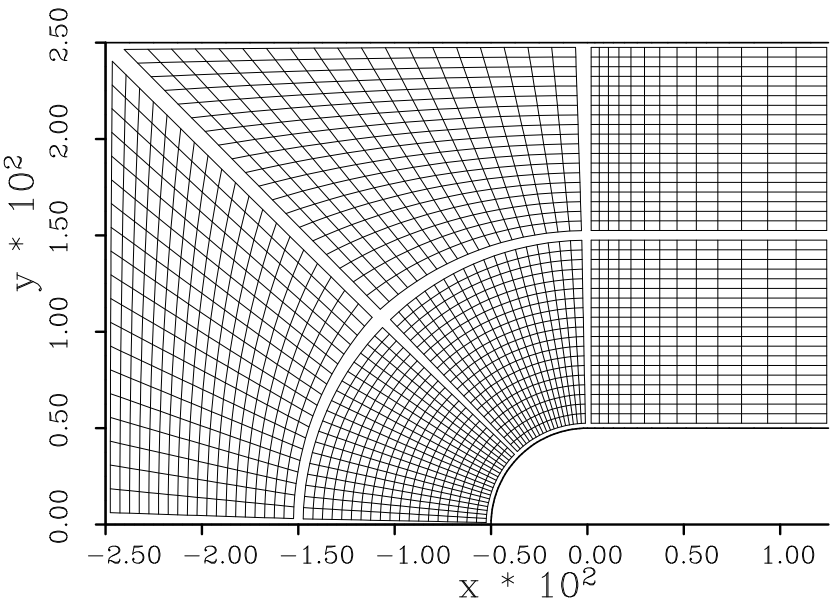

Figure 8: Low resolution grid (with 20×20 cells per block) for the heat-transfer probe. Distances are in metres. The lines, which look like cell boundaries, actually join adjacent cell centres.

Figure 8 shows a coarse grid discretisation of the flow domain around the spherical nose of the probe. The centre of curvature for the nose is located at the origin. Blocks 0 and 1 cover the part of the domain adjacent to the spherical nose, out to a radius of 15 mm. Block 2 extends this part of the domain downstream, along the cylindrical surface, to $x = +25$ mm. To allow a convenient simulation of the incident (planar) shock, block 3 extends the upstream part of the domain to a planar (inflow) surface at $x = -25$ mm. Blocks 4 and 5 complete the domain by providing a cylindrical outer boundary at $y = 25$ mm. Since only the inviscid flow near the nose of the probe is of interest, the (poor) quality of the grid in other regions of the flow domain has been ignored.

Slip boundary conditions are applied along the surfaces of the probe and along the $y = 25$ mm boundary. A simple extrapolation boundary condition is used at the downstream $x = +25$ mm boundary. Since the simulation was stopped before the shock reached this downstream boundary, this choice of boundary condition is not critical. The simulation starts at $t = 0$ with post-shock conditions

$$\rho = 2.957 \text{ kg/m}^3, \quad e = 3.324 \times 10^5 \text{ J/kg}, \quad u_x = 380.0 \text{ m/s},$$

$$p = 393.1 \text{ kPa}, \quad T = 463.2 \text{ K},$$

being applied at the upstream $x = -25$ mm boundary. Note that these inflow conditions are subsonic with $M = 0.881$ but, because the post-shock flow conditions are matched to the incident shock strength, there is no problem in applying these inflow boundary conditions (at least until an upstream-travelling wave strikes the boundary).

Figure 9 shows the development of the flow around the nose of the probe at three instants after the shock reaches the tip of the probe. This arrival can be seen as the peak in both pressure and temperature at $t = 31\mu$s in Figure 10 which shows the histories of pressure and temperature for the cell nearest the stagnation point. The conditions are momentarily like refected-shock conditions but, over a period of 20μs, they decay to Pitot conditions.

The grid resolution used for this calculation was 80×80 cells in each of the 6 blocks. The simulation took 2836 time steps to reach a final time of $t = 80\mu$s and, using the EFM flux calculator, required 8814 seconds of CPU time on a DEC Alphastation. Comparisons of CPU time required for other machines are included in Table 1.

With a few blocks and roughly equal computational effort required for all blocks, this case should perform reasonably well on a parallel computer such as the SGI Power Challenge. Table 2 gives the CPU times required to run a shortened version of this case

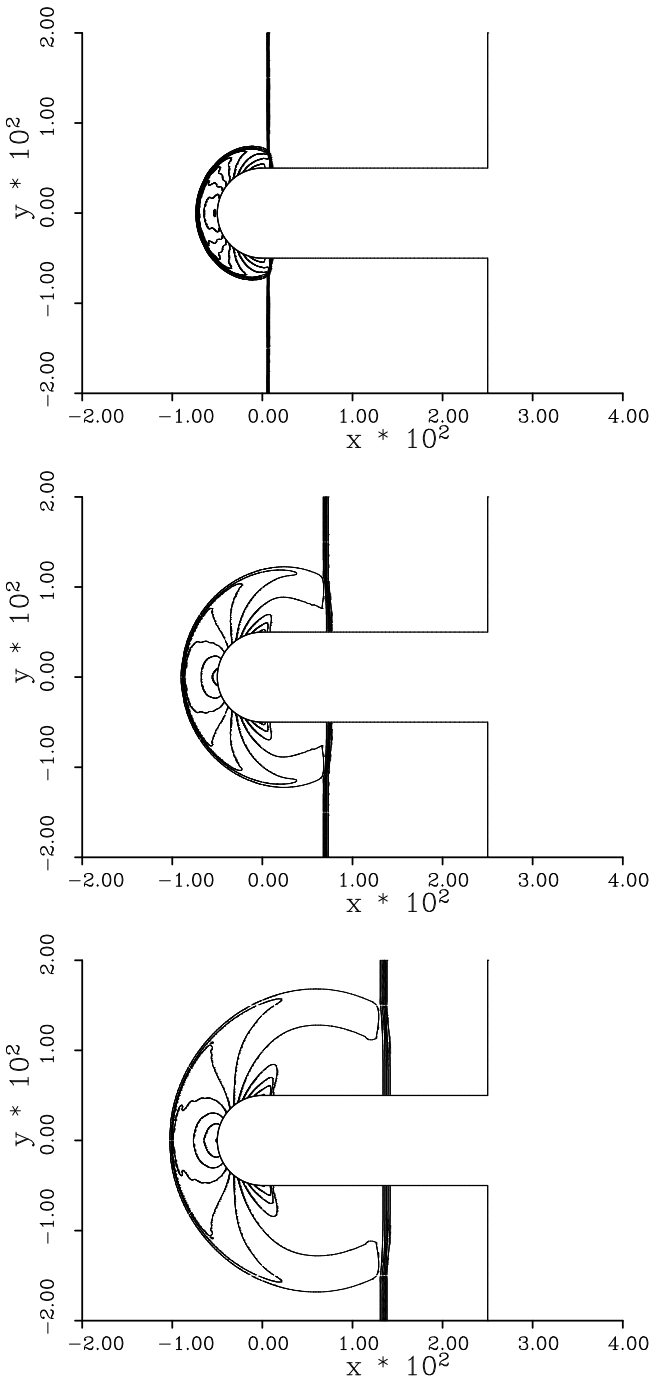

Figure 9: Pressure contours ($100 \leq p \leq 800$kPa, $\Delta p = 50$kPa) for transient flow over a heat-transfer probe at times $t = 40\mu$s, 50μs and 60μs. Grid resolution is 80×80 cells in each of the six blocks. Distances are in metres.

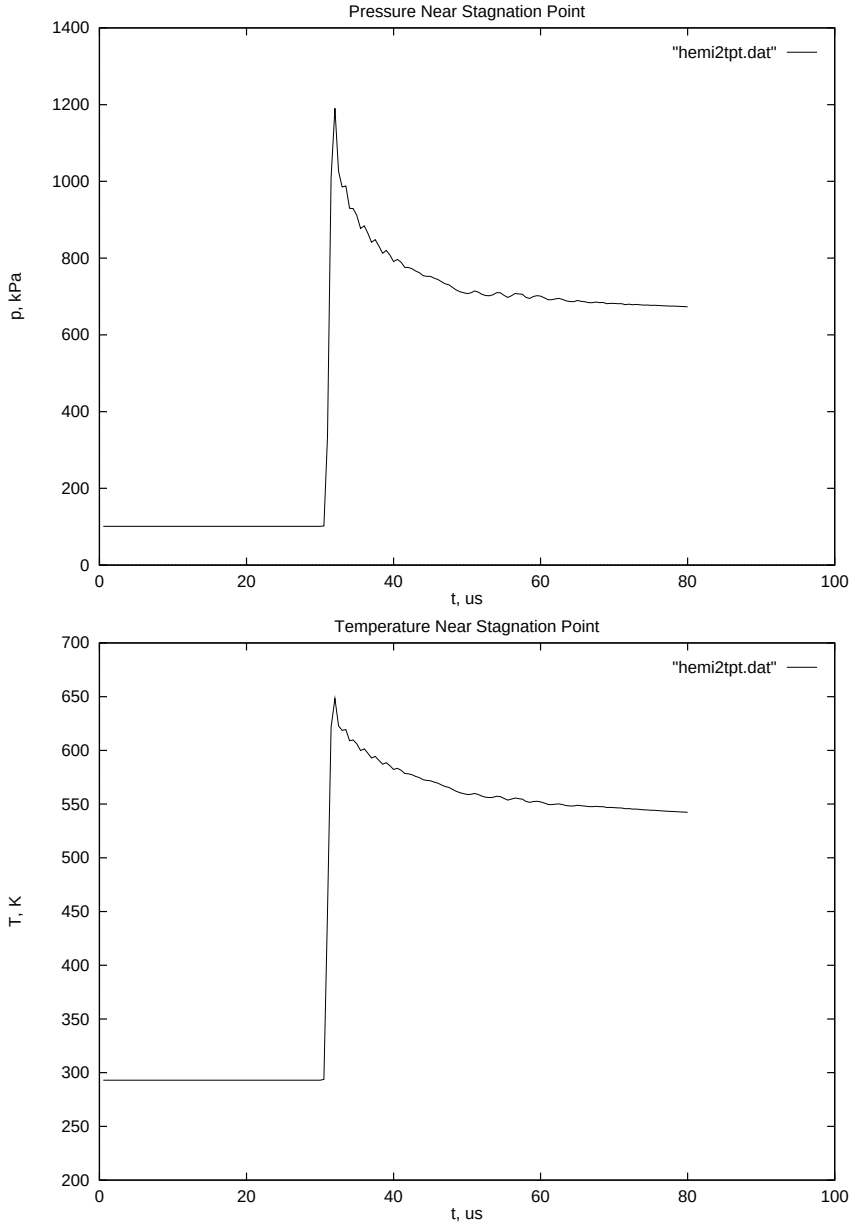

Figure 10: Pressure and temperature histories for the cell nearest the stagnation point for transient flow over a heat-transfer probe.

which stops after 100 time-steps (see *mb_cns/examples/hemi2/hemi2_short.p*). Since the system software would return only the total CPU time used by all threads (in contrast to the CPU time for each individual thread), the speedup achieved with n parallel processors is estimated as

$$s = \frac{n \times \mathrm{CPU-for-1-thread}}{\mathrm{CPU-for-n-threads}} \quad ,$$

and efficiency as

$$\eta = \frac{\mathrm{CPU-for-1-thread}}{\mathrm{CPU-for-n-threads}} \quad .$$

The table shows that, when the work is evenly distributed between the processors, reasonable parallel performance can be obtained. The column of most interest to the pragmatic user is labelled as "Wall-Clock Time". This the the *real* time that one needs to wait to get a result.

Table 2: Performance of the code on the SGI Power Challenge for 100 steps of the simulation of flow over the heat-flux probe.

Threads	Total CPU Time (seconds)	Wall-Clock Time (seconds)	Speedup	Efficiency %
1	304	324	–	–
1	297	313	–	–
2	319	154	1.86	93
3	350	113	2.54	85
3	333	107	2.68	89
4	441	109	2.70	–
5	524	104	2.83	–
6	378	62	4.71	79
6	381	62	4.68	78

4 Concluding Remarks & Acknowledgements

The code is generally available in machine readable form and requires only a C compiler in a suitable computing environment (e.g. UNIX or OS/2 with a few megabytes of memory). Further details on installing the code and running the examples are provided in the HTML files accompanying the source code.

The original (single-block) code was started approximately six years ago (December, 1991) and was based on the quasi-one-dimensional upwinding technology available at the time. Much of the code is now starting to show its age and the flow simulation technology has not kept pace with recent developments. The wish list for future developments include:

- a finite-rate chemistry module[12];
- a robust multi-dimensional flux calculator that has low computational cost;
- parallel code [6] that is based on the MPI standard for distributed-memory computers;
- better grid generation and geometry management software to generate smoothly varying, nearly-orthogonal grids and to eliminate the tedium of generating parameter files and Bezier geometry files by hand.

This code has used ideas from many sources and, because a number of years have passed, it is difficult to remember who said what and when. In particular, I would like to thank Bernie Grossman for those initial lessons on shock-capturing methods on the blackboard at ICASE. Also, at ICASE and NASA Langley: Bob Walters, James Quirk, John Korte, Jeff White, Phil Drummond and Jeff Scroggs. Since 1992, Michael Macrossan, Mike Wendt and Masa Takahashi have provided advice while the graduate students in CFD (Chris Craddock, Paul Petrie, Andrew McGhee and Ian Johnston) have provided algorithm and code contributions.

Financial support, especially for the parallelisation work, has been provided by a University of Queensland New Staff Grant. Parallel computing support has been provided by the Prentice Centre and the High Performance Computing Unit at The University of Queensland.

References

[1] P. A. Jacobs. Single-block Navier-Stokes integrator. ICASE Interim Report 18, 1991.

[2] M. N. Macrossan. The equilibrium flux method for the calculation of flows with non-equilibrium chemical reactions. *Journal of Computational Physics*, 80(1):204–231, 1989.

[3] M. S. Liou and C. J. Steffen. A new flux splitting scheme. NASA Technical Memorandum 104404, 1991.

[4] P. A. Jacobs. An approximate Riemann solver for hypervelocity flows. *A.I.A.A. Journal*, 30(10):2558–2561, 1992.

[5] D. I. Pullin. Direct simulation methods for compressible inviscid ideal-gas flow. *Journal of Computational Physics*, 34(2):231–244, 1980.

[6] A. M. McGhee and P. A. Jacobs. Parallel computation of hypervelocity flow. In R. L. May and A. K. Easton, editors, *Computational Techniques and Applications: CTAC95*, pages 549–556. World Scientific, 1996.

[7] D. F. Rogers and J. A. Adams. *Mathematical Elements for Computer Graphics (2nd ed).* McGraw-Hill, New York, 1990.

[8] D. A. Anderson, J. C. Tannehill, and R. H. Pletcher. *Computational Fluid Mechanics and Heat Transfer.* Hemisphere Publishing Corporation, New York, 1984.

[9] J. J. Quirk. An alternative to unstructured grids for computing gas dynamic flows around arbitrarily complex two-dimensional bodies. *Computers and Fluids*, 23(1):125–142, 1993.

[10] F. Yamaguchi. *Curves and surfaces in computer aided geometric design.* Springer-Verlag.

[11] D. R. Buttsworth and T. V. Jones. A transient thin film heat flux gauge with finite film thickness. Department of Engineering Science, Unpublished Report, University of Oxford, Oxford, UK., 1996.

[12] C. S. Craddock. A quasi-one-dimensional space-marching flow solver with finite rate chemical effects. Department of Mechanical Engineering Report 7/96, The University of Queensland, Brisbane, October 1996.

[13] J. J. Gottlieb and C. P. T. Groth. Assessment of Riemann solvers for unsteady one-dimensional inviscid flows of perfect gases. *Journal of Computational Physics*, 78(2):437–458, 1988.

[14] R. C. Reid, J. M. Prausnitz, and B. E. Poling. *The Properties of Gases and Liquids, 4th Ed.* 1987.

[15] S. Srinivasan, J. C. Tannehill, and K. J. Weilmuenster. Simplified curve fits for the thermodynamic properties of equilibrium air. NASA Reference Publication 1181, 1987.

[16] R. K. Prabhu and W. D. Erickson. A rapid method for the computation of equilibrium chemical composition of air to 15000 k. Technical Paper 2792, NASA, 1988.

A Approximate Riemann Solver

The Riemann solver [4] originally used in *CNS4U* initially assumed isentropic (rarefaction or compression) waves and then, if (at least) one of the pressure ratios was larger than 10, applied a Newton iteration based on the strong-shock equations. This allowed the flux calculator to be used for very strong shocks while still allowing the code to be vectorised. However, it was later found that switching the calculation procedure in this fashion would sometimes cause a sudden jump in the profiles of bow shocks over bluff bodies. To eliminate this behaviour, at the cost of an increase in computational effort, the Newton iterations were changed to work with the shock equations for shocks of arbitrary strength and were applied at lower pressure ratios.

The changes affect only the second stage of the solver so that, whenever the stage-1 estimate for the intermediate pressure p^* is "large" (*i.e.* $p^*/p_L > 1.5$ or $p^*/p_R > 1.5$), this estimate is updated using 4 Newton-Raphson steps of the form

$$p^*_{k+1} = p^*_k - F_k \left(\frac{dF_k}{dp^*} \right)^{-1} \quad . \tag{21}$$

The function $F_k(p^*)$ is constructed as the difference between the Left and Right estimates of the intermediate velocity

$$\begin{aligned} F_k(p^*) &= u^*_L(p^*_k) - u^*_R(p^*_k) \quad , \\ &= [u_L - f(p^*_k, p_L, a_L, \gamma_L)] - [u_R + f(p^*_k, p_R, a_R, \gamma_R)] \quad . \end{aligned} \tag{22}$$

The function f depends on whether the wave is a shock or rarefaction [13]

$$\begin{aligned} f(p^*, p, a, \gamma) &= \frac{a}{\gamma} \left[\frac{p^*}{p} - 1 \right] \frac{1}{\beta} \qquad , p^* \geq p \quad , \\ f(p^*, p, a, \gamma) &= \frac{2a}{\gamma - 1} [\delta - 1] \qquad , p^* < p \quad , \end{aligned} \tag{23}$$

where

$$\begin{aligned} \beta &= \left[\frac{(\gamma+1)}{2\gamma} \frac{p^*}{p} + \frac{\gamma - 1}{2\gamma} \right]^{\frac{1}{2}} \quad , \\ \delta &= \left[\frac{p^*}{p} \right]^{\frac{(\gamma-1)}{2\gamma}} \quad , \end{aligned} \tag{24}$$

have been used to keep the expressions in equation (23) relatively simple. The derivatives df/dp^* are

$$\begin{aligned} f'(p^*, p, a, \gamma) &= \frac{(\gamma+1)a}{4\gamma^2 p} \left[\frac{p^*}{p} + \frac{3\gamma - 1}{\gamma + 1} \right] \frac{1}{\beta^3} \qquad , p^* \geq p \quad , \\ f'(p^*, p, a, \gamma) &= \frac{a}{\gamma p^*} \delta \qquad , p^* < p \quad . \end{aligned} \tag{25}$$

B Gas Properties

A number of gas models are included in the module *gas.c*. These include a number of perfect gases, a couple of perfect gas mixtures, and models for air and nitrogen in chemical equilibrium. Access is provided by the routine *EOS()* which which uses values for ρ, e and f_{is} to compute the other thermodynamic properties T, p, a, and the viscous transport coefficients μ and k.

B.1 Perfect Gas Models

Air, modelled as a perfect gas, has an equation of state

$$p = \rho\ e\ (\gamma - 1) \qquad \text{Pa} \quad , \tag{26}$$

where $\gamma = C_p/C_v = 1.4$ is the ratio of specific heats, density ρ is given in kg/m^3 and specific internal energy e is given in J/kg. Temperature and speed of sound are also determined as

$$T = \frac{e}{C_v} \quad {}^{\circ}\text{K} \quad , \tag{27}$$

$$a = (\gamma R T)^{\frac{1}{2}} \quad \text{m/s} \quad , \tag{28}$$

where

$$\begin{aligned} R &= 287\ \text{J/(kg.K)}, \\ C_v &= \frac{R}{(\gamma - 1)} = 717.5\ \text{J/(kg.K)}, \text{ and} \\ C_p &= C_v + R = 1004.5\ \text{J/(kg.K)}\ . \end{aligned}$$

The viscous transport coefficients are computed as

$$\mu = 1.458 \times 10^{-6} \frac{T^{3/2}}{(T + 110.4)} \quad \text{Pa.s} \quad , \tag{29}$$

$$\lambda = \frac{-2}{3}\mu \quad \text{Pa.s} \quad , \tag{30}$$

$$k = \frac{\mu C_p}{Pr} \quad \text{W/(m.K)} \quad , \tag{31}$$

where $Pr = 0.72$ is the Prandtl number.

The other perfect gas models available include nitrogen, helium and argon. See the code module *gas.c* for details.

B.2 Mixtures of Two Perfect Gases

The thermodynamic properties of a mixture of two perfect gases are computed using effective gas constants and specific heats

$$R = f_a\ R_a + f_b\ R_b \quad , \tag{32}$$

$$C_v = f_a\ C_{va} + f_b\ C_{vb} \quad , \tag{33}$$

$$C_p = f_a\ C_{pa} + f_b\ C_{pb} \quad , \tag{34}$$

where f_a and f_b are the mass fractions of the gas components. Viscosity is estimated using Wilke's method with the Herning-Zipperer approximation as described in Chapter 9 of Ref. [14]

$$\mu = \alpha_a\ \mu_a + \alpha_b\ \mu_b \quad , \tag{35}$$

$$\alpha_a = \left[1 + \frac{f_b}{f_a}\left(\frac{M_a}{M_b}\right)^{\frac{1}{2}}\right]^{-1} \quad , \tag{36}$$

$$\alpha_b = 1 - \alpha_a \quad , \tag{37}$$

where M_a and M_b are the molecular weights of components a and b respectively. The thermal conductivity of the mixture is calculated using the same weighting scheme (see Ch.10 of Ref. [14]).

B.3 Models with Equilibrium Chemistry

Air, in chemical equilibrium is modelled using the curve fits from in Ref. [15] while a model for nitrogen alone has been extracted from the procedure described in Ref. [16].