MBCNS2 Example Book.

Mechanical Engineering Report 2006/02
P. A. Jacobs
Centre for Hypersonics
The University of Queensland.

February – December 2006

Preface

MBCNS2 is a small collection of programs for the simulation of transient two-dimensional (or axisymmetric) flows. It is part of the larger collection of compressible flow simulation codes found at `http://www.mech.uq.edu.au/cfcfd/`. This manual is a collection of example simulations: scripts, results and commentary. It may be convenient for new users of the code to identify an example close to the situation that they wish to model and then adapt the scripts for that example.

This report will be updated occasionally with new examples and commentary. As the simulation codes are improved, we will try to maintain compatibility so that older examples are not "broken", however, this backward compatibility will not be guaranteed.

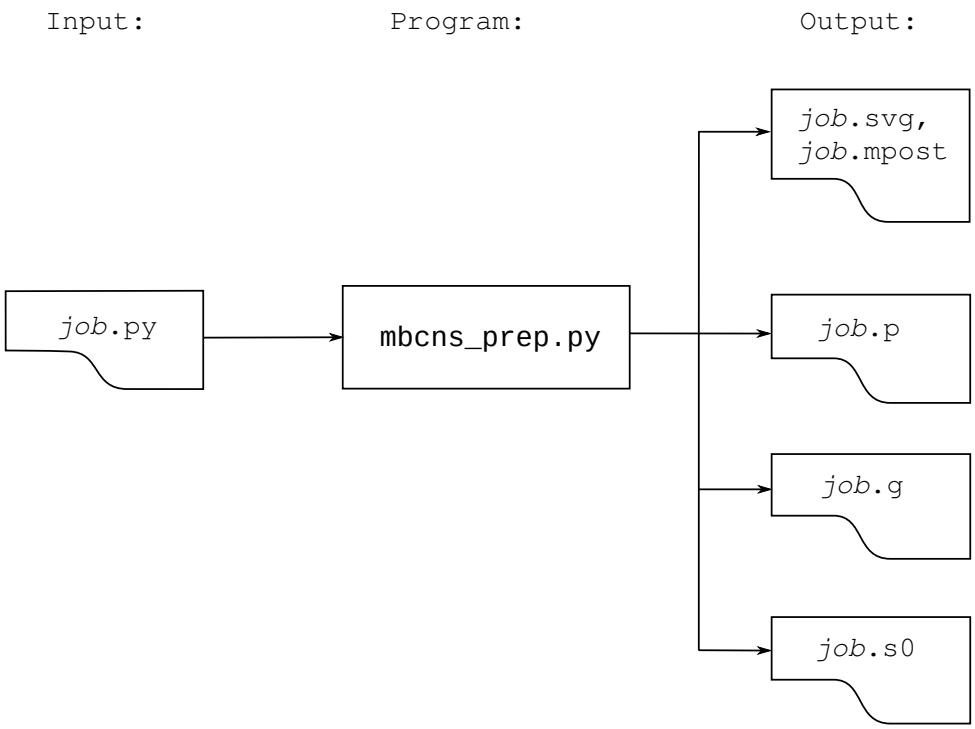
Contents

1 Running simulations

Setting up a simulation is mostly an exercise in writing a text-based description of your flow and its bounding geometry. This flow-specification file is presented to the preparation program as a Python source file, often with the extension ".py". Once you have prepared your flow specification file, the simulation data is generated in a number of stages:

1 Create the geometry definition, a grid and an initial flow solution (at $t = 0$) with the command.

```
$ mbcns_prep.py --job=job --do-svg --do-mpost
```



The italics word *job* should be replaced by whatever job name that you have chosen. That name is then used as a base to derive specific names for each of the files associated with the simulation. The files from the preparation stage are:

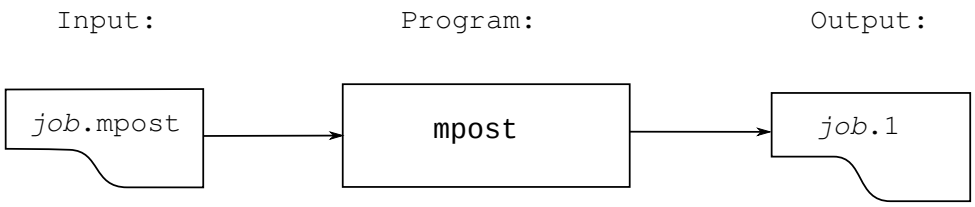
- *job*`.svg`: A rendering of the flow domain and boundary conditions in Scalable Vector Graphics (SVG) format.
- *job*`.mpost`: A rendering of the flow domain and boundary conditions in Metapost format.
- *job*`.p`: A database of flow simulation parameters in INI format. Parameters are specified, one per line, as *parameter-name* = *value*. A hierarchical structure is given to the set of parameters via named subsections in the file. Although you would probably never assemble one of these parameter files manually, it is sometimes convenient to alter a value or two and rerun a simulation without

invoking `mbcns_prep.py`.

- *job*`.g`: The grid of finite-volume cells that define the flow domain.
- *job*`.s0`: The initial flow state within each of the finite-volume cells.

2 Check the geometry definition (visually) by using Metapost to make a viewable postscript file containing labelled nodes, block boundary curves and blocks. Metapost is distributed as part of the TEX document preparation system. It is most likely already installed on your UNIX/Linux system and there is a stand-alone binary for Win32 systems.

```
$ mpost job.mpost
```



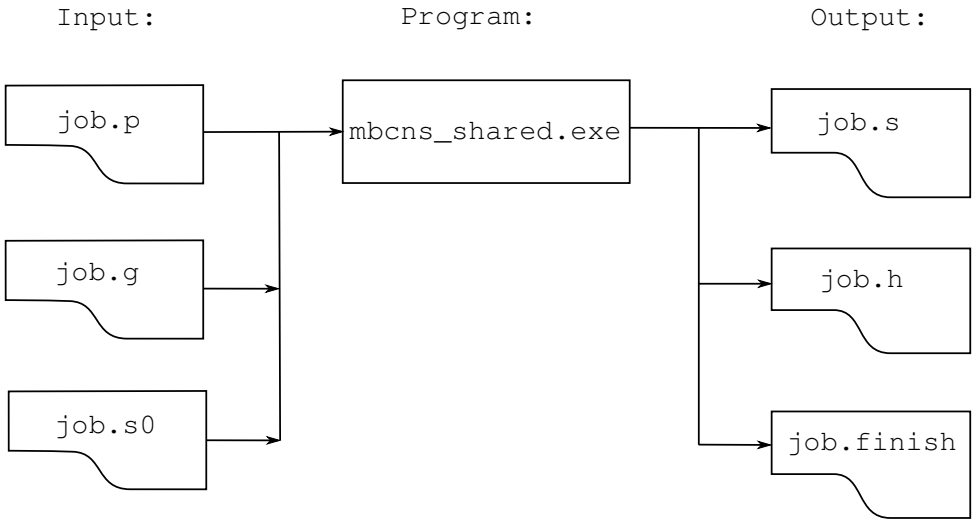
The postscript output is written to a file with `.1` as its extension. Alternatively, a drawing program (such as Inkscape) can be used to edit the Scalable Vector Graphics file.

3 Run the simulation code to produce flow data at subsequent times.

```
$ mbcns_shared.exe --job job --run
```

Alternatively, shorter command-line options may be used.

```
$ mbcns_shared.exe -f job -r
```



The output files are:

- *job*.s: The flow data for all cells at the times requested. As the simulation proceeds, whole-field solutions are added to the end of this file.
- *job*.h: Data at particular "history locations" and at times requested. This data is typically used to simulate the signals recorded by pressure and heat-trnasfer sensors mounted on model surfaces.
- *job*.finish: A little bit of information about the final time reached by the simulation. This may assist when automating some of the post-processing operations.

4 Extract subsets of the flow solution data for postprocessing. The specific command for this stage depends very much on what you want to do. Enter the program name alone to get some hints as to usage.

   The flow solution data is cell-averaged data associated with cell centres. You may extract the flow data for all cells at a particular time using mbcns_post.exe and reformat it for a particular plotting program or you may extract data along single grid lines (using mbcns_profile.exe) in a form ready for display with GNU-Plot or for further calculation. See the shell scripts in the examples for ideas on what can be done. Since the output of this stage is always a text file, you may look at the head of the file for hints as to what data is present.

Currently, mbcns_prep.py is implemented as a Python program that has a library of classes specialized for constructing geometric regions and specifying flow conditions. Because your specification script, *job*.py, becomes part a part of that program when it runs, it is worth the effort to learn just enough Python to be dangerous. The Web site http://www.python.org is a good starting point for learning about the Python programming language.

After doing some initialization, mbcns_prep.py executes your script file and assembles the geometry and flow specification data into a form that can be given to the main simulation code mbcns_shared.exe[1]. The advantage of this approach is that you have the full capability of the Python interpreter available to you from within your script. You can perform calculations so that you can parameterize your geometry, for example, or you can use Python control structures to make repetitive definitions much more concise. Additionally, you may use Python comments and print statements to add documentation to the script file.

[1]The "shared" tag indicated that we are using the shared-memory version of the code. There is also a distributed-memory version, based on message passing (MPI). When using this program (mbcns_mpi.exe), you need to have multiple grid and solution files, one for each block within the flow domain. This is arranged with the --split command-line option for mbcns_prep.py.

Beginners may find that the `mbcns_console.tcl` program presents a convenient GUI where the processing stages appear explicitly in an "action" menu and the options for each of the programs are displayed in a tabbed-notebook arrangement. By laying out all of the options, `mbcns_console.tcl` provides an overview of the simulation process. Also, the commands that it issues to run the various stages can be seen in the text window that remains in the background. Experienced users will probably want to write shell (or Python) scripts to automate their simulations.

The following sections provide example input files and shell scripts for a number of simulations. These are intended to be starting points for your own simulations and should be studied together with the other manuals that can be found in the documentation section of the Compressible Flow CFD Group web site: `http://www.mech.uq.edu.au/cfcfd/`.

2 Mach 1.5 flow over a 20^o cone

This is a small (in both memory and run time) example that is useful for checking that the simulation and plotting programs have been built or installed correctly. Assuming that you have the program executable files built and accessible on your system's search `PATH`, try the following commands:

```
$ cd ~/cfcfd2/examples/mbcns2/cone20
$ ./cone20_run.sh
$ ./cone20_plot.sh
```

And, within a minute or so, you should end up with a number of files with various solution data plotted. The grid and initial solution are created and the time-evolution of the flow field is computed for 5 ms (with 1105 time steps being required). The commands invoke the shell scripts displayed in subsection 2.2.

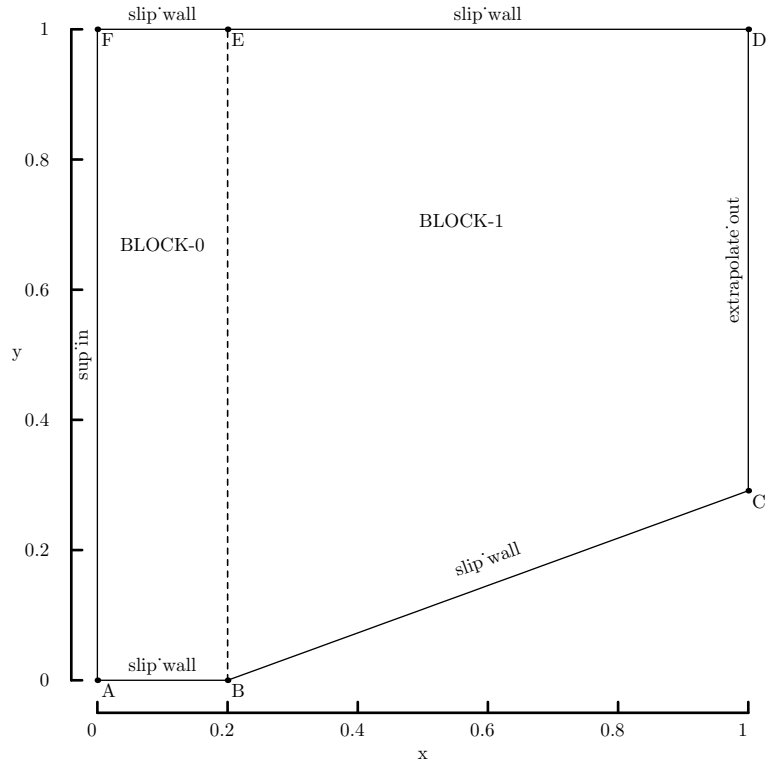


Figure 1: Schematic diagram of the geometry for a cone with 20 degree half-angle. This encapsulated postscript figure was generated from the Metapost file.

The free-stream conditions ($p_\infty = 95.84\,\text{kPa}$, $T_\infty = 1103\,\text{K}$ and $u_\infty = 1000\,\text{m/s}$) are related to the shock-over-ramp test problem in the original ICASE Report [1] and are set to give a Mach number of 1.5. From Chart 5 in Ref. [2], the expected steady-state shock

wave angle is 49^o and, from Chart 6, the pressure coefficent is

$$\frac{p_{cone-surface} - p_\infty}{q_\infty} \approx 0.387$$

and the dynamic pressure for the specified free stream is $q_\infty = \frac{1}{2}\rho_\infty u_\infty^2 \approx 151.38\,\text{kPa}$. Figure 4 shows the pressure coefficient estimated as

$$C_p = \frac{f_x - p_\infty A}{q_\infty A}$$

from the simulated axial force, f_x, written into the simulation log file and frontal area of the cone, A.

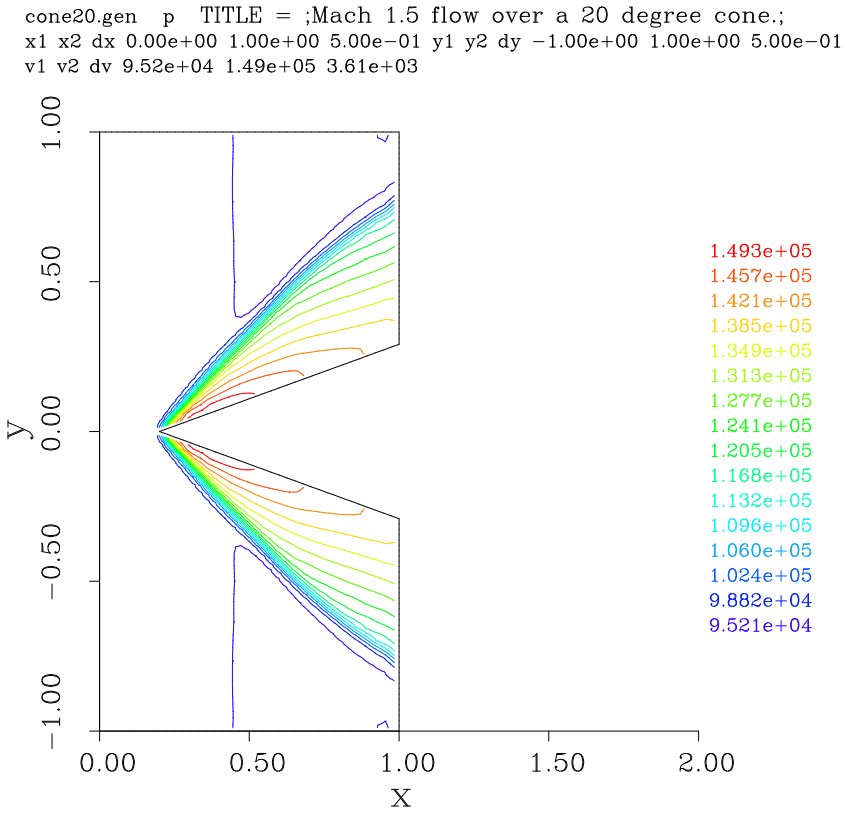


Figure 2: Pressure data for flow over a cone with 20 degree half-angle. The shock profile is not yet straight and the pressure field near the cone surface is not conically symmetric, although it would become more so if we continued the simulation.

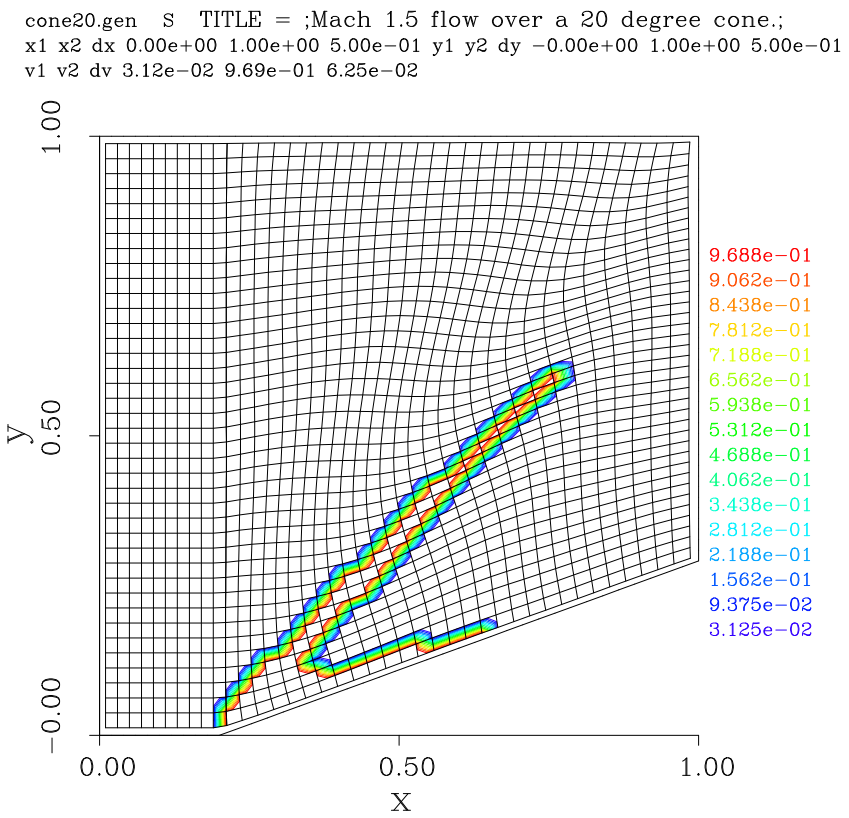


Figure 3: Shock-sensor data for flow over a cone with 20 degree half-angle. For the `adaptive` flux calculator, this sensor indicates the regions of the flow where the more dissipative scheme should be used.

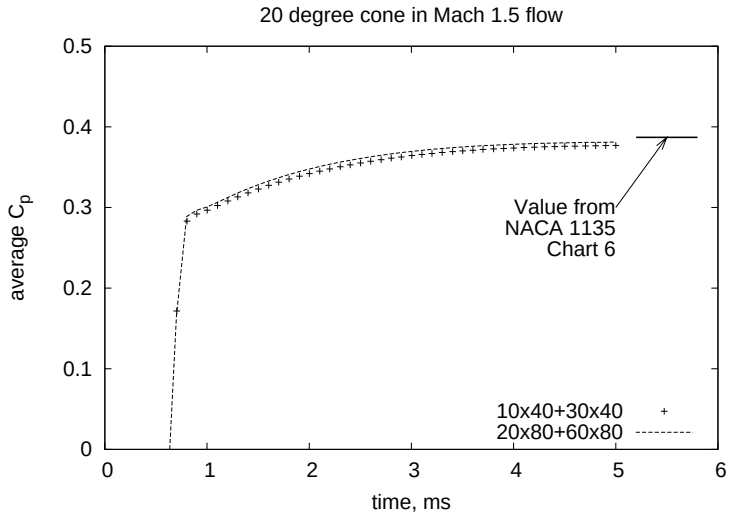


Figure 4: Evolution of the axial (drag) force for flow over a cone with 20 degree half-angle for two mesh resolutions.

2.1 .py file

```
## \file cone20.py
## \brief Test job-specification file for mbcns_prep.py
## \author PJ, 08-Feb-2005
##
## We have set this file up very much like the cone20.sit file
## so that users may more-easily see the correspondence between
## the Tcl and Python elements.

job_title = "Mach 1.5 flow over a 20 degree cone."
print job_title

# We can set individual attributes of the global data object.
gdata.title = job_title
gdata.case_id = 5
gdata.axisymmetric_flag = 1
gdata.stringent_cfl = 1  # to match the old mb_cns behaviour

# Accept defaults giving perfect air (gamma=1.4)
create_perf_gas()
gdata.set_gas_model()

# Define flow conditions
initial = FlowCondition(p=5955.0,  u=0.0,    v=0.0, T=304.0,  mf=[1.0,])
inflow  = FlowCondition(p=95.84e3, u=1000.0, v=0.0, T=1103.0, mf=[1.0,])

# Set up two quadrilaterals in the (x,y)-plane be first defining
# the corner nodes, then the lines between those corners and then
# the boundary elements for the blocks.
# The labelling is not significant; it is just to make the MetaPost
# picture look the same as that produced by the Tcl scriptit program.
a = Node(0.0, 0.0, label="A")
b = Node(0.2, 0.0, label="B")
c = Node(1.0, 0.29118, label="C")
d = Node(1.0, 1.0, label="D")
e = Node(0.2, 1.0, label="E")
f = Node(0.0, 1.0, label="F")

ab = Line(a, b); bc = Line(b, c) # lower boundary including cone surface
fe = Line(f, e); ed = Line(e, d) # upper boundary
af = Line(a, f); be = Line(b, e); cd = Line(c, d) # vertical lines

# Define the blocks, boundary conditions and set the discretisation.
nx0 = 10; nx1 = 30; ny = 40
blk_0 = Block2D(make_patch(fe, be, ab, af), nni=nx0, nnj=ny,
                fill_conditions=initial, label="BLOCK-0")
blk_1 = Block2D(make_patch(ed, cd, bc, be, "AO"), nni=nx1, nnj=ny,
                fill_conditions=[initial,], label="BLOCK-1",
                hcell_list=[(10,1)], xforce_list=[0,0,1,0])
identify_block_connections()
blk_0.bc_list[WEST] = SupInBC(inflow) # one way to set a BC
blk_1.set_BC(EAST, EXTRAPOLATE_OUT)   # another way

# Do a little more setting of global data.
gdata.viscous_flag = 1
gdata.flux_calc = ADAPTIVE
gdata.max_time = 5.0e-3  # seconds
gdata.max_step = 3000
gdata.dt = 1.0e-6
# gdata.dt = 4.0e-6
# gdata.fixed_time_step = True
gdata.dt_plot = 1.5e-3
gdata.dt_history = 10.0e-5

sketch.xaxis(0.0, 1.0, 0.2, -0.05)
sketch.yaxis(0.0, 1.0, 0.2, -0.04)
sketch.window(0.0, 0.0, 1.0, 1.0, 0.05, 0.05, 0.17, 0.17)
```

2.2 Shell scripts

```
#! /bin/sh
# cone20_run.sh
# exercise the Navier-Stokes solver for the Cone20 test case.
# It is assumed that the path is set correctly.

# Generate the Bezier, Input parameter and MetaPost files from the Script File.
# The MetaPost file provides us with a graphical check on the geometry
# specification and it is worth creating if metapost (mpost) is available.
# Also, generate the Grid and Initial Solution Files.
mbcns_prep.py --job=cone20.py --do-mpost --do-svg
mpost cone20.mpost

# Integrate the solution in time,
# recording the axial force on the cone surface.
#
# The following environment variables allow the shared-memory version
# of the code to use one thread for each of the two blocks.
# The stacksize requirements may increase as the code develops and
# more elements are added to the internal data structures.
export OMP_NUM_THREADS=1
export KMP_STACKSIZE=8m
time mbcns_shared.exe -f cone20 --run

# Extract the solution data and reformat.
# If no time is specified, the first solution found is output.
mbcns_post.exe -fp cone20.p -fg cone20.g -fs cone20.s -fo cone20 -generic

# Extract the average coefficient of pressure from the axial force
# records that were written to the simulation log file.
awk -f cp.awk mbcns_shared.log > cone20_cp.dat

echo "At this point, we should have a new solution"
echo "Run cone20_plot.sh next"
```

```
# cone20_plot.sh
# Pick up the reformatted data and make plots of:

# 1. Shock indicator
mb_cont.exe -fi cone20.gen -fo cone20_S.ps -var 9 -ps -colour \
            -xrange 0.0 1.0 0.5 -yrange -0.0 1.0 0.5 -mesh

# 2. Pressure.
mb_cont.exe -fi cone20.gen -fo cone20_p.ps -var 6 -ps -colour \
            -mirror -xrange 0.0 1.0 0.5 -yrange -1.0 1.0 0.5
mb_cont.exe -fi cone20.gen -fo cone20.gif -var 6 -gif -colour \
            -mirror -xrange 0.0 1.0 0.5 -yrange -1.0 1.0 0.5

# 3. The mesh alone.
mb_cont.exe -fi cone20.gen -fo cone20_mesh.ps -var 6 -ps -colour \
            -xrange 0.0 1.0 0.5 -yrange -0.0 1.0 0.5 -mesh -nocontours

# 4. The average coefficient of pressure on the cone surface.
#    We assume that the high-resolution data file is also available.
gnuplot <<EOF
set term postscript eps enhanced 20
set output "cone20_cp.ps"
set style line 1 linetype 1 linewidth 3.0
set title "20 degree cone in Mach 1.5 flow"
set xlabel "time, ms"
set ylabel "average C_p"
set xtic 1.0
set ytic 0.1
set yrange [0:0.5]
set key bottom right
set arrow from 5.2,0.387 to 5.8,0.387 nohead linestyle 1
```

```
set label "Value from\nNACA 1135\nChart 6" at 5.0,0.3 right
set arrow from 5.0,0.3 to 5.5,0.387 head
plot "cone20_cp.dat" using 1:2 title "10x40+30x40", \
     "cone20_cp_hi-res.dat" using 1:2 title "20x80+60x80" with lines
EOF
```

2.3 Notes

- Remember that long-format command-line options start with two dashes. For example `--job=cone20`. These double dashes are a little hard to distinguish in the shell scripts.

- Run time is approximately 30 seconds on an LG LS70 *Express* portable computer with an Intel Pentium-M 1.73Ghz processor.

- This cone20.py file really has full access to the Python interpreter on your system. Later examples will show how to use Python to write data files from within the input script. Be careful.

- Python is a dynamic language. It is easy to bind names to new objects within your script. Be careful that you do not rebind essential names that will be later used by the `mbcns_prep.py` program. Where this might happen in a non-obvious way is in the importing of foreign modules (to do something interesting in your script) with the command "from *module-name* import *".

- There is a shell script (`cone20_mpi.sh`) to run the MPI version of the simulation code for this example.

- Awk script for extracting x-force data from the simulation log file. New users might like to use an equivalent program written in Python.

```
# cp.awk
# Extract the simulation times and axial force values from the log file.
# The relevant lines in mb_cns.log start with the string "XFORCE"
# and are of the form:
#    XFORCE: TIME 2.049920e-04 BLOCK 1 BNDY 2 FX_P 1.840878e+03 FX_V 3.472884e-02
# Present the axial force as an average coefficient of pressure to
# compare with that obtained from NACA 1135.

BEGIN {
    p_inf = 95.84e3;  # Pa
    T_inf = 1103.0;   # K
    u_inf = 1000.0;   # m/s
    R = 287;          # J/kg.K
    r_base = 0.29118; # m
    rho_inf = p_inf / (R * T_inf);  # kg/m**3
    q_inf = 0.5 * rho_inf * u_inf * u_inf;  # Pa
    A = 3.14159 * r_base * r_base;  # m**2
    print "# time (ms)  Cp";
    print "# rho_inf= ", rho_inf, " q_inf= ", q_inf, " A= ", A
}

/XFORCE/ {
    # Select just the simulation time and the force on the cone surface.
```

```
    t = $3;  # in seconds
    f = $9;  # pressure force in Newtons
    # The coefficient of pressure is based on the difference
    # between the cone surface pressure and the free-stream pressure.
    Cp = (f / A - p_inf) / q_inf;
    print t*1000.0, Cp;
}
```

- The command: `$ mb_compare.sh cone20`
  will compare the newly-computed solution with a reference solution stored in compressed files in the `./reference` subdirectory. If all is well, you should get a report with zero differences for each of the files except the log-file. The log-file will almost certainly contain differences with respect to run times (or wall-clock times).

3 Flow of equilibrium-air over a sphere

This example is a good starting-point for the modelling of hypersonic flow over blunt bodies. It shows the use of arcs and the use of a look-up table as the equation of state for a gas in chemical equilibrium but it remains geometrically simple by using a single-block grid. Also, the .py file makes use of the Python language to parameterize the simulation's specification.

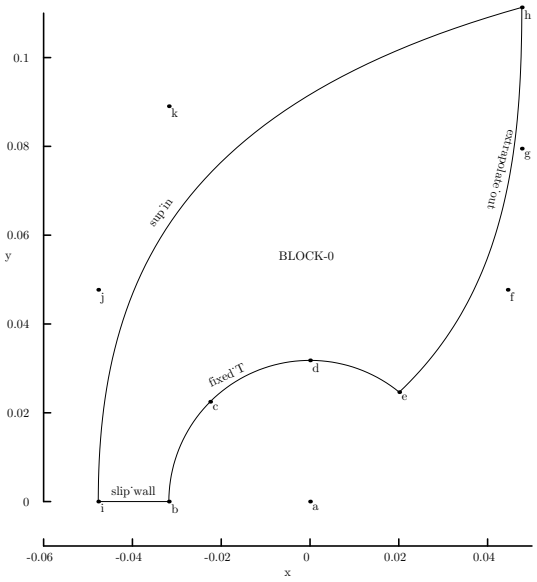


Figure 5: Schematic diagram of the geometry for a sphere wrapped by a single-block grid.

The free-stream condition ($p_\infty = 20\,\text{kPa}$, $T_\infty = 296\,\text{K}$, $u_\infty = 4.68\,\text{km/s}$) corresponds to Case 3 in Ref. [3] with $M_\infty = 13.6$. According to Sawada & Dendou [3], the air is close to being in chemical equilibrium and there is a very thin boundary layer. The results show that the inviscid simulation does indeed capture some of the high-temperature chemistry influence. Ideal stagnation temperature would be 11204 K whereas the simulated temperature along the stagnation line rises to only 6081 K. Secondly, the stand-off distance for an ideal gas is expected to be approximately 4.63 mm. In Fig. 7 the simulated shock stand-off distance is 2.66 mm near the stagnation point. This is within 3% of the experimental value obtained by D. Reda in Sandia's Ballistics Range (see [3]).

3.1 .py file

```
# file: ss3.py
#
# Sphere in equilibrium air modelling Case 3 from
#     K. Sawada & E. Dendou (2001)
#     Validation of hypersonic chemical equilibrium flow calculations
#     using ballistic-range data.
#     Shock Waves (2001) Vol. 11, pp 43--51
```

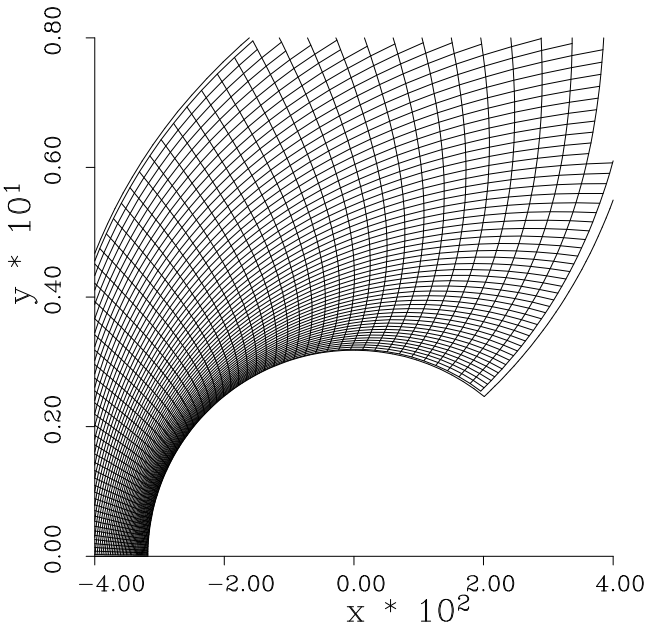


Figure 6: Mesh for flow over a sphere.

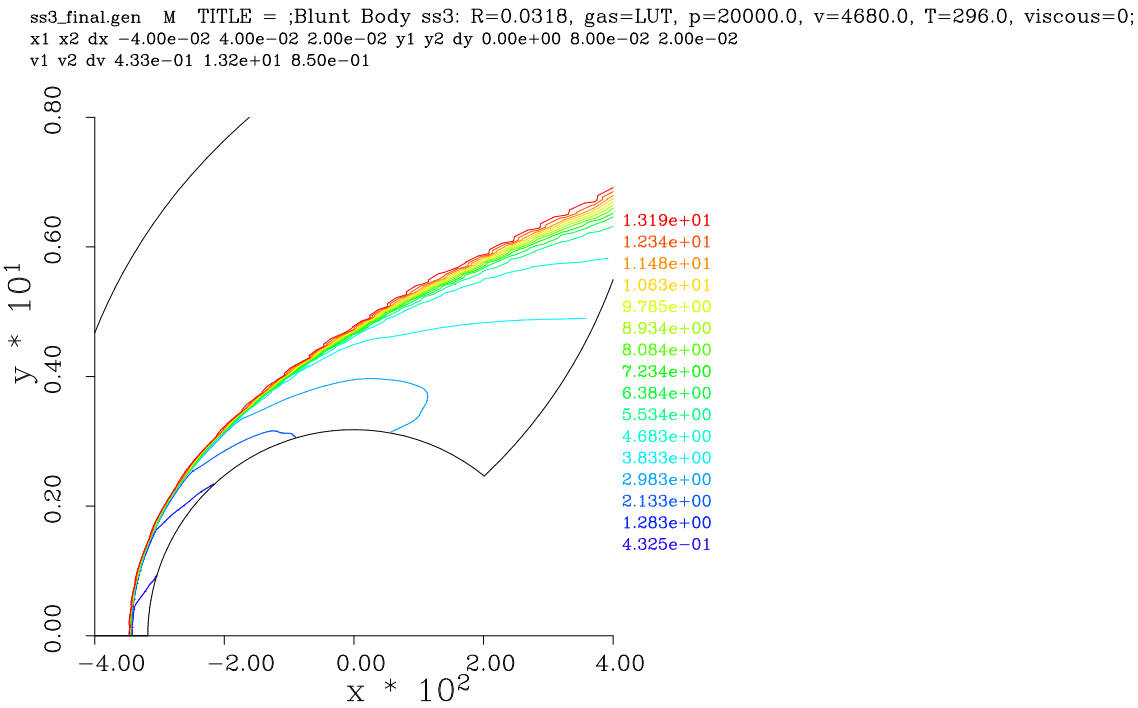


Figure 7: Mach number data for equilibrium-air flow over a sphere.

```
#
# Experimental shock stand-off distance is 2.59mm
# Sawada & Dendou CFD value:               2.56mm
#
# This script derived from rbody, 22-Jan-2004.
# and the Python version: ss3.py, 04-Apr-2005, 10-Aug-2006, 27-Nov-2006
# PJ
#
# The grid is a bit wasteful because the shock lies close to
# the body for equilibrium air, however, this grid layout
# (as used in rbody) allows us to play with perfect-gas models
# without hitting the inflow boundary with the shock.

# The following JOB name is used to build file names at the end.
JOB = "ss3"

# Radius of body
R = 31.8e-3                     # m
T_body = 296.0                  # surface T, not relevant for inviscid flow
body_type = "sphere"            # choose between "cylinder" and "sphere"

# Free-stream flow definition
p_inf = 20.0e3                  # Pa
T_inf = 296.0                   # degrees K
u_inf = 4.68e3                  # flow speed, m/s

# For equilibrium chemistry, use the look-up-table.
gdata.set_gas_model( "LUT", "lut.dat" )
inflow = FlowCondition(p=p_inf, u=u_inf, v=0.0, T=T_inf, mf=[1.0,])
initial = FlowCondition(p=0.3*p_inf, u=0.0, v=0.0, T=T_inf, mf=[1.0,])

# Job-control information
do_viscous = 0                  # flag for viscous/inviscid calc
nn = 60                         # grid resolution, both ix and iy
t_final = 10.0 * R / u_inf      # allow time to settle at nose
t_plot = t_final / 1.0          # plot only once
TitleText = "Blunt Body " + JOB + ": R=" + str(R) + ", gas=" + str(gdata.gas_name) + \
            ", p=" + str(p_inf) + ", v=" + str(u_inf) + ", T=" + str(T_inf) + \
            ", viscous=" + str(do_viscous)
gdata.title = TitleText
gdata.case_id = 0
if do_viscous:
    gdata.viscous_flag = 1
    gdata.viscous_delay = t_plot
if body_type == "sphere":
    gdata.axisymmetric_flag = 1
gdata.flux_calc = ADAPTIVE
gdata.max_time = t_final
gdata.max_step = 400000
gdata.dt = 1.0e-8
gdata.cfl = 0.40
gdata.dt_plot = t_plot
gdata.dt_history = 1.0e-6

# Begin geometry details...
# Note that mbcns_prep.py has already imported the math module.
deg2rad = math.pi / 180.0
alpha1 = 135.0 * deg2rad
alpha2 = 50.8 * deg2rad
# The node coordinates are scaled with the body radius.
# The labels are not required but make the MetaPost plot
# look a little like the plot produced by scriptit.tcl.
a = Node(0.0, 0.0, label="a")
b = Node(-1.0 * R, 0.0, label="b")
c = Node(math.cos(alpha1) * R, math.sin(alpha1) * R, label="c")
d = Node(0.0, R, label="d")
e = Node(math.cos(alpha2) * R, math.sin(alpha2) * R, label="e")
f = Node(1.4 * R, 1.5 * R, label="f")
g = Node(1.5 * R, 2.5 * R, label="g")
h = Node(1.5 * R, 3.5 * R, label="h")
i = Node(-1.5 * R, 0.0, label="i")
j = Node(-1.5 * R, 1.5 * R, label="j")
k = Node(-1.0 * R, 2.8 * R, label="k")
```

```
east0  = Polyline([Arc(b, c, a), Arc(c, d, a), Arc(d, e, a)])
north0 = Bezier([e, f, g, h,]); north0.reverse()
south0 = Line(i, b)
west0  = Bezier([i, j, k, h,])

print "ss3: block to be defined."
cluster_functions = [RobertsClusterFunction(0, 1, 1.2),
                     RobertsClusterFunction(1, 0, 1.1),
                     RobertsClusterFunction(0, 1, 1.2),
                     RobertsClusterFunction(1, 0, 1.1)]
boundary_conditions = [ExtrapolateOutBC(), FixedTBC(T_body),
                       SlipWallBC(), SupInBC(inflow)]

blk_0 = Block2D(psurf=make_patch(north0, east0, south0, west0),
                fill_conditions=initial,
                nni=nn, nnj=nn,
                cf_list=cluster_functions,
                bc_list=boundary_conditions,
                label="BLOCK-0", hcell_list=[(nn,1)])

# Some hints to scale and place the sketch.
# If you change the radius, you'll probably have to adjust the axes.
sketch.xaxis( -0.060, 0.050, 0.020, -0.010)
sketch.yaxis(  0.0,   0.110, 0.020, 0.0)
sketch.window(-1.5*R, 0.0, 1.5*R, 3.0*R, 0.05, 0.05, 0.15, 0.15)
```

3.2 Shell scripts

```
#! /bin/sh
# file: ss3_setup_lut.sh

cp ~/cfcfd2/lib/cea_tables/cea_table_air.txt ./cea_table.txt
cea_to_binary.exe -extrapolate
mv cea_lut.dat lut.dat

echo "We should now have a Look-Up-Table for air"
```

```
# ss3_run_py.sh
# Shell script to set up and run Sawada & Dendou's sphere case 3.
# Use the newer Python script interpreter.
# PJ, 04-Apr-05
# updated 18-Jul-06 by RJG, 22-Jul-06 by PJ

# For a clean start
mbcns_prep.py --job=ss3.py --do-mpost --do-svg
mpost ss3.mpost

# The main event
time mbcns_shared.exe --job=ss3 --run
```

```
# ss3_post.sh
#
TFINAL=67.0e-6

mbcns_profile.exe -fp ss3.p -fg ss3.g -fs ss3.s -fo ss3_stag_line.data \
    -t $TFINAL -yline 0 0 1
awk -f locate_shock.awk ss3_stag_line.data > ss3.result
```

3.3 Notes

- The `ss3_setup_lut.sh` script assumes a "standard" location for the cfcfd2 directories in order to find the files for the look-up-table gas model. Yes, that is a single dash for the `-extrapolate` command-line option to the `cea_to_binary.exe` program. The text form of the look-up-table has been generated as a regular array of sample points over ranges of density and temperature. When reformatting the table to have a regular array of data points over density and internal-energy, this option allows the program to extrapolate when necessary. When this option is not given, the binary table covers smaller ranges of density and internal-energy that fall completely within the original text data.

- This simulation reaches a final time of 67.95 μs in 4548 steps and, on a Pentium-M 1.73 Ghz system, this takes 5 min, 6 s of CPU time. This timing will be a bit sensitive to the state of the code because the large data structures appear to be causing a lot of cache misses. If we cut down on the amount of storage for each cell and reduce the size of the temporary arrays, we can achieve significant reductions in the CPU time.

- Awk script for extracting the shock location from the stagnation-line flow data.

```
# locate_shock.awk

BEGIN {
    p_old = 0.0;
    x_old = -2.0;   # dummy position
    y_old = -2.0;
    p_trigger = 2.0e6; # something midway between free stream and stagnation
    shock_found = 0;
}

$1 != "#" { # for any non-comment line, do something
    p_new = $7;
    x_new = $1;
    y_new = $2;
    # print "p_new=", p_new, "x_new", x_new, "y_new", y_new
    if ( p_new > p_trigger && shock_found == 0 ) {
        shock_found = 1;
        frac = (p_new - p_trigger) / (p_new - p_old);
        x = x_old + frac * (x_new - x_old);
        y = y_old + frac * (y_new - y_old);
        print "shock-location= ", x, y
    }
    p_old = p_new;
    x_old = x_new;
    y_old = y_new;
}

END {
    if ( shock_found == 0 ) {
        print "shock not located";
    }
    print "done."
}
```

4 Hypersonic flow of ideal air over a blunt wedge

This example is a partial solution to the CFD exercise for the MECH4470 class in 2004. Because the original specification was given in nondimensional form, an arbitrary 10 mm nose radius has been selected for the inviscid simulation. This is also a reasonable size for a possible wind tunnel experiment. The free-stream condition was specified as having a Mach number of 5 and the gas was specified as ideal air. Choosing particular values of $p_\infty = 100\,\text{kPa}$, $T_\infty = 100\,\text{K}$, lead to a free-stream velocity of $u_\infty = 1002\,\text{m/s}$ and a dynamic pressure of $q_\infty = 1.75\,\text{MPa}$.

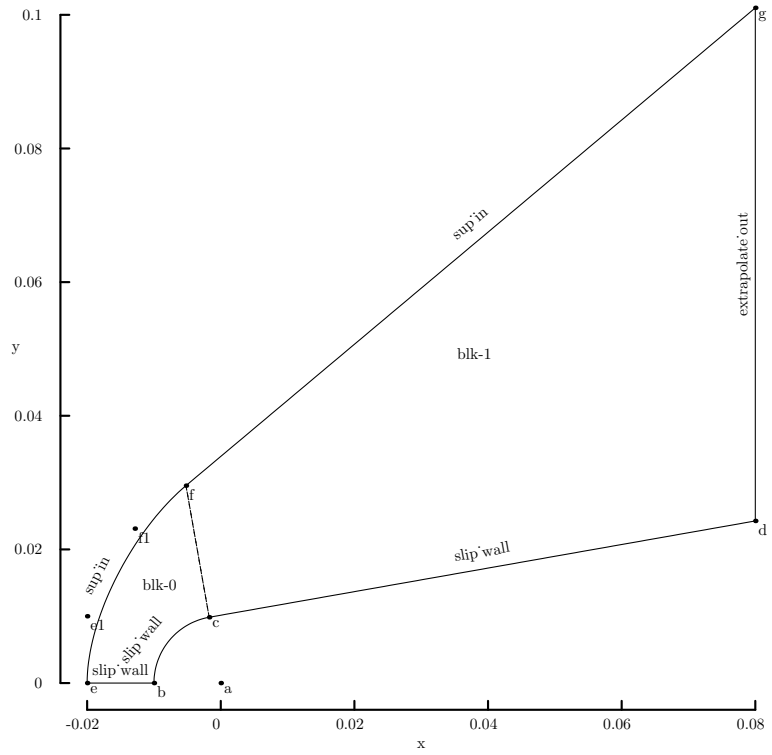


Figure 8: Schematic diagram of the geometry for the blunted 10 degree wedge.

The simulation is started with low-pressure conditions throughout the flow domain and free-stream conditions applied to the inflow boundary (the west boundary of blk-1 and the north boundary of blk-1). The flow data is allowed to evolve until $t_{final} = 399\,\mu\text{s}$, which corresponds to a particle of the free-stream travelling 40 nose radii. The axial force (shown in Fig.10) is seen to settle to a value of 28590 N in that time. This corresponds to a drag coefficient of 0.674.

The surface pressure (shown normalised in Fig. 11) has been extracted from the solution file by `mbcns_profile.exe` by selecting the east-most line of cells of the first block and the south-most line of cells of the second block. The selected data is filtered by an Awk script to produce the normalised data (and the Newtonian reference data) as plotted.

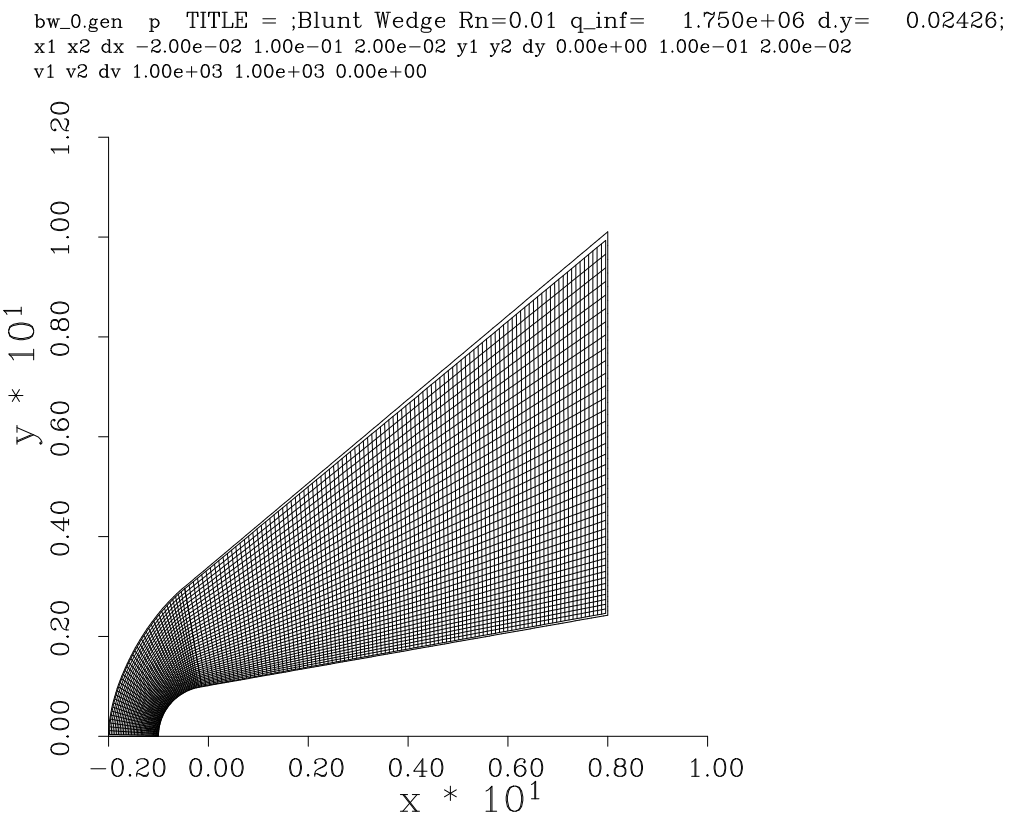


Figure 9: Mesh for the blunt wedge exercise.

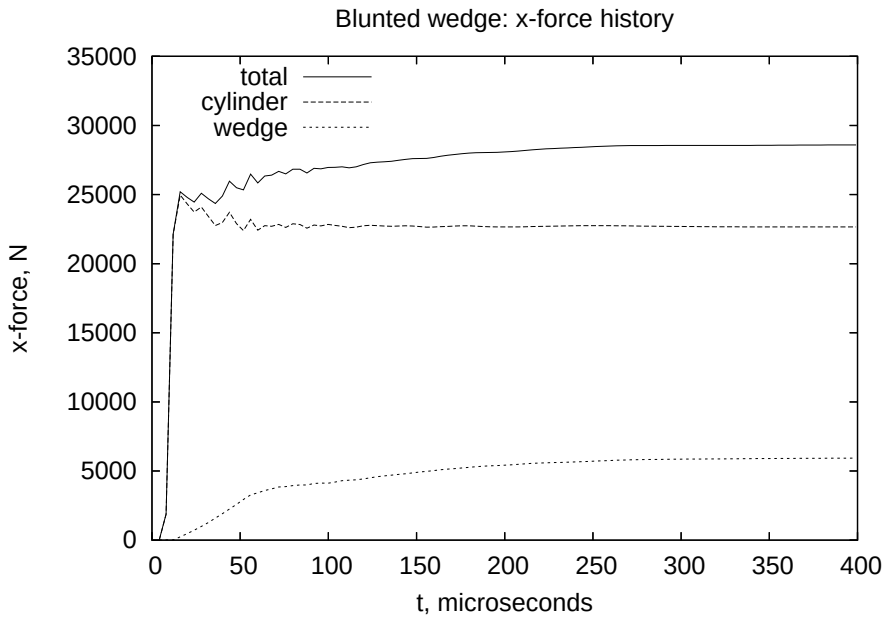


Figure 10: History of the axial forces for the blunt-wedge exercise.

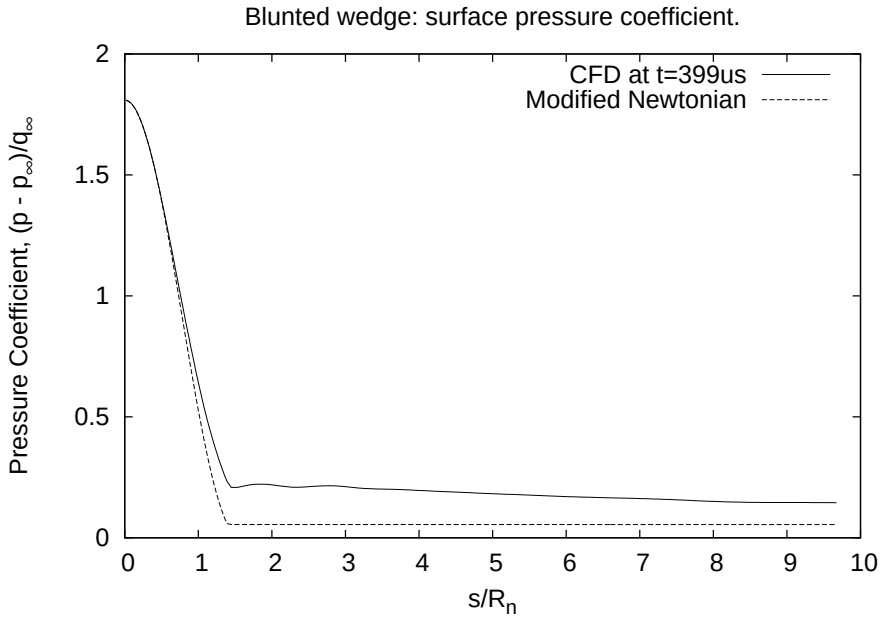


Figure 11: Surface pressure coefficient data for the blunt-wedge exercise.

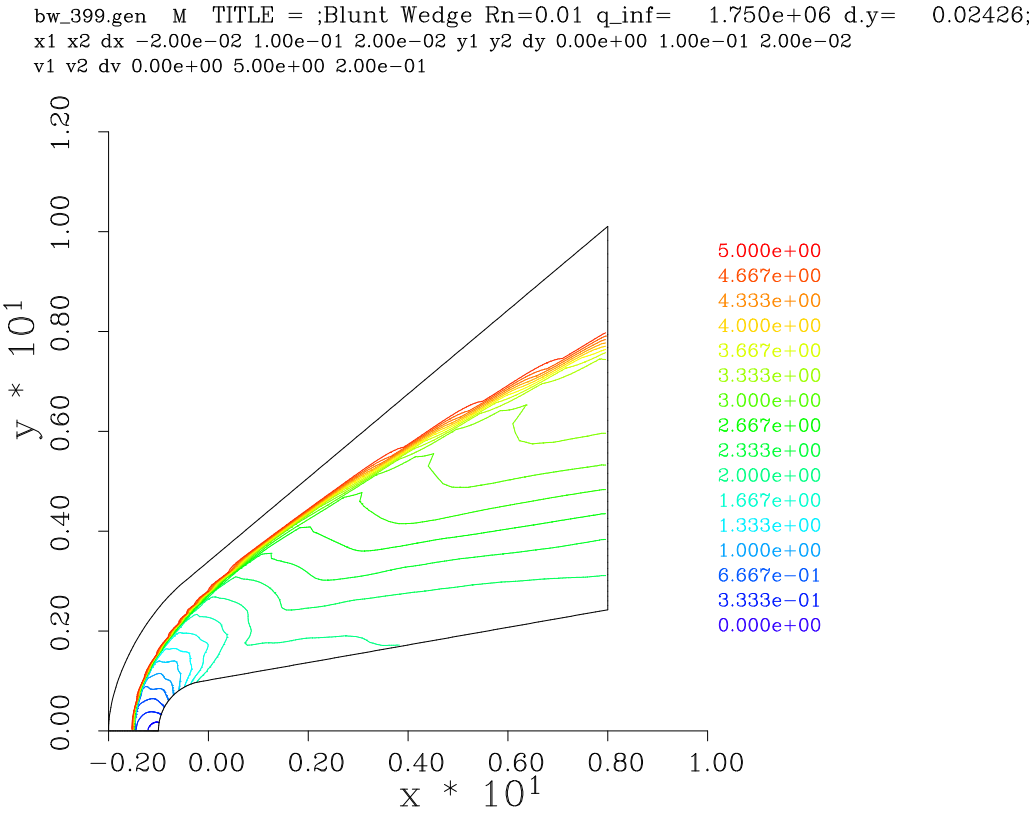


Figure 12: Mach number data for the blunt-wedge exercise.

4.1 .py file

```
# bw.py
# MECH4470/CFD Exercise: Hypersonic flow over a blunt wedge.
# PJ, 07-Dec-2006

from math import sqrt, sin, cos, tan, pi

# Default gas model is ideal air
create_perf_gas()
gdata.set_gas_model()

# Free stream
g_gas = 1.4                     # Ideal Air
R_gas = 287.0
M_inf = 5.0                     # Specified Mach number
p_inf = 100.0e3                 # Select a static pressure
T_inf = 100.0                   # and a temperature
a_inf = sqrt(T_inf * R_gas * g_gas) # determine sound speed
u_inf = M_inf * a_inf                   # and velocity
# Also, handy to know dynamic pressure for nondimensionalization
# of the pressures and drag forces.
q_inf = 0.5 * g_gas * p_inf * M_inf * M_inf
print "Free-stream velocity, u_inf=", u_inf
print "    static pressure, p_inf=", p_inf
print "    dynamic pressure, q_inf=", q_inf
free_stream = FlowCondition(p=p_inf, u=u_inf, v=0.0, T=T_inf, mf=[1.0,])
# For transient simulation, we start with a low pressure.
initial = FlowCondition(p=1000.0, u=0.0, v=0.0, T=100.0, mf=[1.0,])

# Geometry
Rn = 10.0e-3                    # radius of cylindrical nose
xEnd = 8.0 * Rn                 # downstream extent of wedge
alpha = 10.0 / 180.0 * pi       # angle of wedge wrt free stream
delta = 10.0e-3                 # offset for inflow boundary

# First, specify surface of cylinder and wedge
a = Node(0.0, 0.0, label='a') # Centre of curvature for nose
b = Node(-Rn, 0.0, label='b')
c = Node(-Rn*sin(alpha), Rn*cos(alpha), label='c')
bc = Arc(b, c, a)
# Down-stream end of wedge
d = Node(xEnd, c.y+(xEnd-c.x)*tan(alpha), label='d')
print "height at end of plate yd=", d.y
cd =Line(c, d)

# Outer-edge of flow domain has to contain the shock layer
# Allow sufficient for shock stand-off at the stagnation line.
R2 = Rn + delta
e = Node(-R2, 0.0, label='e')
# The shock angle for a 10 degree ramp with sharp leading edge
# is 20 degrees (read from NACA 1135, chart 2),
# however, the blunt nose displaces the shock a long way out
# so we allow some more space.
# We need to set the boundary high enough to avoid the shock
R3 = Rn + 2.0 * delta
f = Node(-R3*sin(alpha), R3*cos(alpha), label='f')
# Now, put in intermediate control points so that we can use
# cubic Bezier curve for the inflow boundary around the nose
# and a straight line downstream of point f.
e1 = Node(e.x, delta, label='e1')
alpha2 = 40.0 / 180.0 * pi
f1 = Node(f.x-delta*cos(alpha2), f.y-delta*sin(alpha2), label='f1')
ef = Bezier([e, e1, f1, f])
g = Node(xEnd, f.y+(xEnd-f.x)*tan(alpha2), label='g')
fg = Line(f,g)

# Define straight-line segments between surface and outer boundary.
eb = Line(e, b); fc = Line(f, c); dg = Line(d, g)
```

```
# Define the blocks using the path segments.
# Note that the EAST face of region0 wraps around the nose and
# that the NORTH face of region0 is adjacent to the WEST face
# of region1.
region0 = make_patch(fc, bc, eb, ef)
cf = fc.copy(); cf.reverse() # common boundary but opposite sense
region1 = make_patch(fg, dg, cd, cf)
cluster0 = RobertsClusterFunction(0, 1, 1.2)
cluster1 = RobertsClusterFunction(1, 0, 1.2)
nni0 = 40
nnj0 = 40
nni1 = 100
blk_0 = Block2D(region0, nni=nni0, nnj=nnj0,
                cf_list=[cluster0,None,cluster0,None],
                fill_conditions=initial,
                xforce_list=[0,1,0,0])
blk_1 = Block2D(region1, nni=nni1, nnj=nnj0,
                cf_list=[None,cluster1,None,cluster1],
                fill_conditions=initial,
                xforce_list=[0,0,1,0])
identify_block_connections()
blk_0.bc_list[WEST] = SupInBC(free_stream)
blk_1.bc_list[NORTH] = SupInBC(free_stream)
blk_1.bc_list[EAST] = ExtrapolateOutBC()


# We can set individual attributes of the global data object.
job_title = "Blunt Wedge Rn=" + str(Rn)
job_title += (" q_inf=%12.3e" % q_inf) + (" d.y=%10.5f" % d.y)
print job_title
gdata.title = job_title
gdata.viscous_flag = 1
gdata.flux_calc = ADAPTIVE
gdata.max_time = 40.0 * Rn / u_inf
print "Final time=", gdata.max_time
gdata.max_step = 5000
gdata.dt = 1.0e-8
gdata.dt_plot = gdata.max_time
gdata.dt_history = gdata.max_time / 100.0
HistoryLocation(b.x-0.001, b.y) # just in front of the stagnation point

sketch.xaxis(-0.020, 0.080, 0.020, -0.004)
sketch.yaxis(0.0, 0.100, 0.020, -0.004)
sketch.scales(1.5, 1.5)
```

4.2 Shell scripts

```
# bw_prep.sh
#
mbcns_prep.py --job=bw --do-mpost --do-svg
mpost bw.mpost

mbcns_post.exe -fp bw.p -fg bw.g -fs bw.s0 -fo bw_0 -generic

XMIN=-20.0e-3
XMAX=100.0e-3
YMIN=0.0
YMAX=100.0e-3
TIC=20.0e-3
mb_cont.exe -fi bw_0.gen -fo bw_0_mesh.ps -ps -var 6 -mesh \
        -xrange $XMIN $XMAX $TIC -yrange $YMIN $YMAX $TIC
```

```
# bw_run.sh
#
time mbcns_shared.exe --job=bw --run
mv mbcns_shared.log bw.mbcns_shared.log
echo "Done"
```

```
# bw_post.sh

TIMES="399"
XMIN=-20.0e-3
XMAX=100.0e-3
YMIN=0.0
YMAX=100.0e-3
TIC=20.0e-3

for TME in $TIMES
do
    mbcns_post.exe -fp bw.p -fg bw.g -fs bw.s -fo bw_$TME -t $TME.0e-6 -generic

    mb_cont.exe -fi bw_$TME.gen -fo bw_"$TME"_p.ps -ps -var 6 -colour \
        -xrange $XMIN $XMAX $TIC -yrange $YMIN $YMAX $TIC
    mb_cont.exe -fi bw_$TME.gen -fo bw_"$TME"_M.ps -ps -var 7 -colour \
        -levels 0.0 5.0.0 0.2 \
        -xrange $XMIN $XMAX $TIC -yrange $YMIN $YMAX $TIC
    mb_cont.exe -fi bw_$TME.gen -fo bw_"$TME"_sonic.ps -ps -var 7 -colour \
        -levels 0.0 1.0 0.2 \
        -xrange $XMIN $XMAX $TIC -yrange $YMIN $YMAX $TIC
done
```

```
# bw_force.sh

# Plot the surface pressure on the wedge
TME=399
NX=40
mbcns_profile.exe -fp bw.p -fg bw.g -fs bw.s -fo bw_surface.dat \
    -t $TME.0e-6 -xline 0 0 $NX -yline 1 1 1
awk -f surface_pressure.awk bw_surface.dat > bw_surface_p_coeff.dat

gnuplot <<EOF
set term postscript eps enhanced 20
set output "bw_surface_pressure.eps"
set title "Blunted wedge: surface pressure coefficient."
set xlabel "s/R_n"
set ylabel "Pressure Coefficient, (p - p_{/Symbol \245})/q_{/Symbol \245}"
set yrange [0.0:2.0]
plot "bw_surface_p_coeff.dat" using 1:2 title "CFD at t=399us" with lines, \
     "bw_surface_p_coeff.dat" using 1:3 title "Modified Newtonian" with lines
EOF

# Plot the axial force coefficient.
awk -f xforce.awk bw.mbcns_shared.log > bw_xforce.dat

gnuplot <<EOF
set term postscript eps 20
set output "bw_xforce.eps"
set title "Blunted wedge: x-force history"
set xlabel "t, microseconds"
set ylabel "x-force, N"
set yrange [0:35000]
set key top left
plot "bw_xforce.dat" using 1:2 title "total" with lines, \
     "bw_xforce.dat" using 1:3 title "cylinder" with lines, \
     "bw_xforce.dat" using 1:4 title "wedge" with lines
EOF
```

4.3 Notes

- This simulation reaches a final time of 399 μs in 3722 steps and, on an Intel Pentium-M 1.73 Ghz system, this takes 6 min, 39 s of CPU time.

- Selection of the `mbcns_shared.log` file showing some x-force data as written during the simulation. Pressure and viscous forces are written separately. Note that the lines are written with several items separated by spaces and the format is mostly self-documenting. The only extra bit of information is that BNDY values are 0, 1, 2 and 3 for boundaries NORTH, EAST, SOUTH and WEST, respectively. See the function `print_forces()` in `cns_xforce.cxx` for details of the format.

```
Step=    420 t= 3.095e-05 dt= 9.413e-08 WC=45.0 WCtFT=419.1 WCtMS=490.7
CFL_min = 1.864646e-03, CFL_max = 4.989532e-01, dt_allow = 9.412608e-08
Smallest CFL_max so far = 3.381486e-02 at t = 1.000000e-07
 dt[0]=9.412608e-08 dt[1]=1.484487e-07
There are 2 active blocks.
Global Residual (for density) = 1.642131e-01
XFORCE: TIME 3.198580e-05 BLOCK 0 BNDY 1 FX_P 2.341363e+04 FX_V 2.125736e+00
XFORCE: TIME 3.198580e-05 BLOCK 1 BNDY 2 FX_P 1.276145e+03 FX_V 1.035778e+01
```

- Awk filter for extracting the x-force data from the simulation log file. Note that there are two pattern-action rules, one for each block.

```
# xforce.awk
# Extract the simulation times and axial force values from the log file.
#

BEGIN {
    print "# time (microseconds)  x-force-total  only-cylinder only-wedge";
}

/XFORCE/ && $5 == 0 {
    # Select just the simulation time and the pressure forces for block 0.
    t = $3;  # in seconds
    fx_p_0 = $9;  # force on cylinder in Newtons
    # Don't do anything until we pick up the wedge data (block 1).
}

/XFORCE/ && $5 == 1 {
    # Select just the simulation time and the pressure forces for block 1.
    t = $3;  # in seconds
    fx_p_1 = $9; # wedge surface in Newtons
    print t*1.0e6, fx_p_0 + fx_p_1, fx_p_0, fx_p_1;
}
```

- Awk filter for normalising the surface pressure data.

```
# surface_pressure.awk
# Normalise the surface pressure with free-stream dynamic pressure and
# compute the distance around from the stagnation point.

BEGIN {
    q_inf = 1.750e6;   # free-stream dynamic pressure
    p_inf = 100.0e3;   # free-stream static pressure
    Rn = 10.0e-3;      # nose radius
    xold = -Rn;        # location of the stagnation point
```

```
    yold = 0.0;
    s = 0.0;            # distance around from stagnation point
    count = 0;
    pi = 3.1415927;
    cone_angle = 10.0/180.0 * pi;
    print "# s/Rn  Cp(CFD)  Cp(Newton)  x(m)  y(m)";
}

$1 != "#" {
    count += 1;
    x = $1;             # cell-centre position
    y = $2;
    p = $7;             # cell-centre pressure
    if ( count == 1 ) p_pitot = p;  # Close enough to the stagnation point.
    dx = x - xold;
    dy = y - yold;
    s += sqrt(dx * dx + dy * dy);
    # Estimate Cp using Modified Newtonian Model.
    theta = 0.5 * pi - (s/Rn);  # local angle of surface
    if (theta < cone_angle) theta = cone_angle;
    Cp_MN = (p_pitot - p_inf) / q_inf * sin(theta) * sin(theta);
    print s/Rn, (p - p_inf)/q_inf, Cp_MN, x, y;
    xold = x;
    yold = y;
}
```

5 Mach 3 flow over a sharp-nosed two-dimensional body

The specifications for this example come from section 5.2 in JD Anderson's Hypersonics book [4]. It shows the use of a `spline` curve as well as being a source of test data for the Method-of-Characteristics for rotational flow. Data for the spline points was computed from

$$\frac{y}{y_e} = -0.008333 + 0.609425\left(\frac{x}{y_e}\right) - 0.092593\left(\frac{x}{y_e}\right)^2$$

where $y_e = 1.0$.

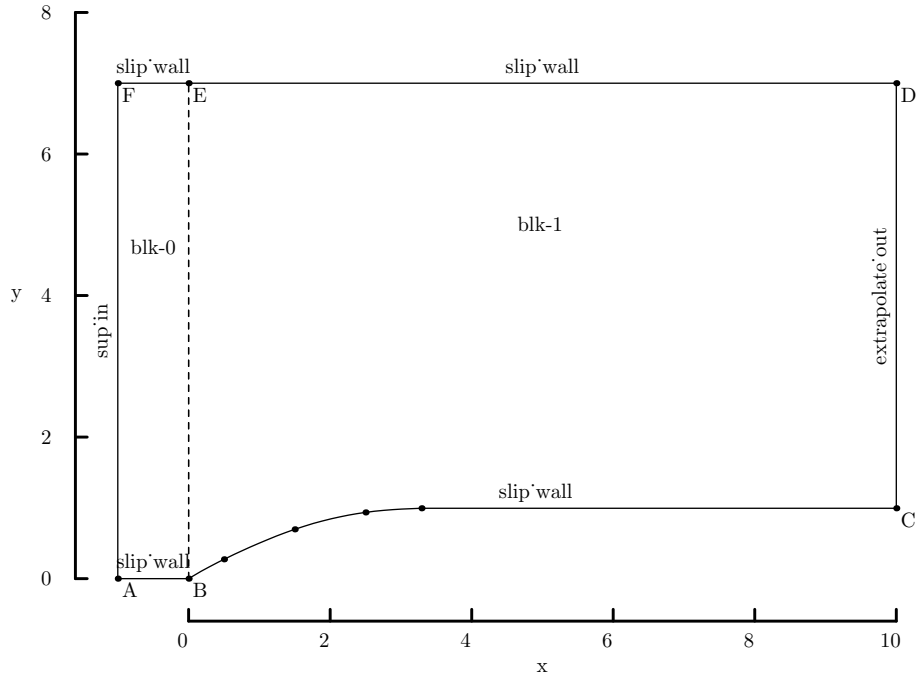


Figure 13: Schematic diagram of the geometry for the sharp body.

The surface pressure (shown in Fig. 15) has been extracted from the solution file by `mbcns_profile.exe` by selecting the south-most line of cells of both blocks. The pressure field (Fig. 16) shows the curved shock clearly.

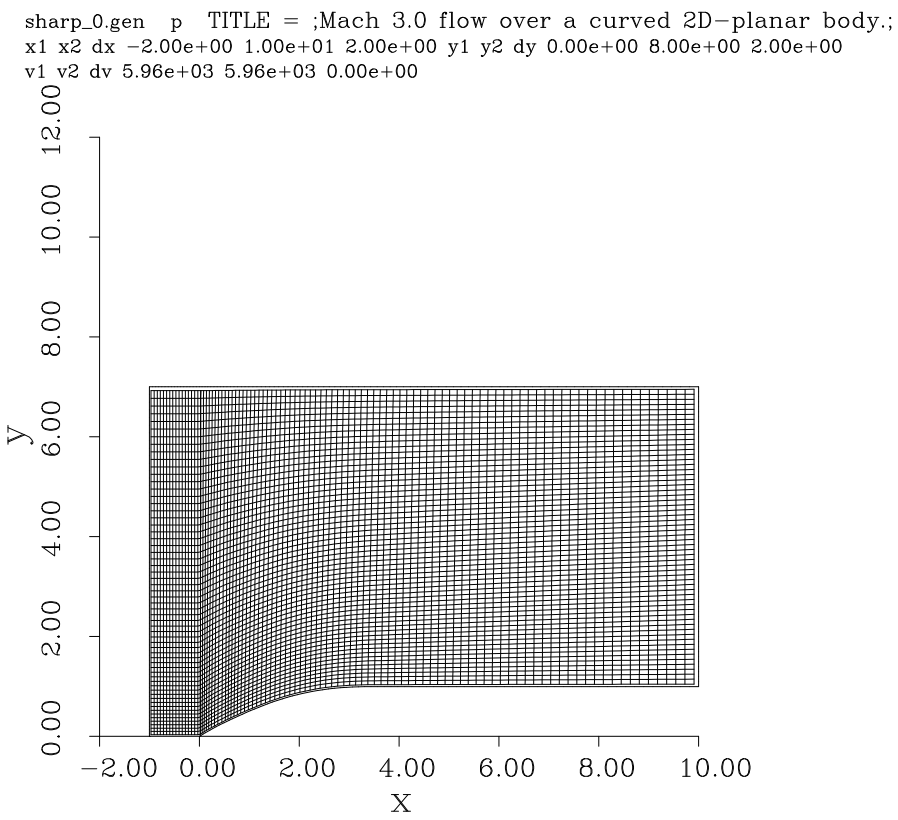


Figure 14: Mesh for the sharp body exercise.

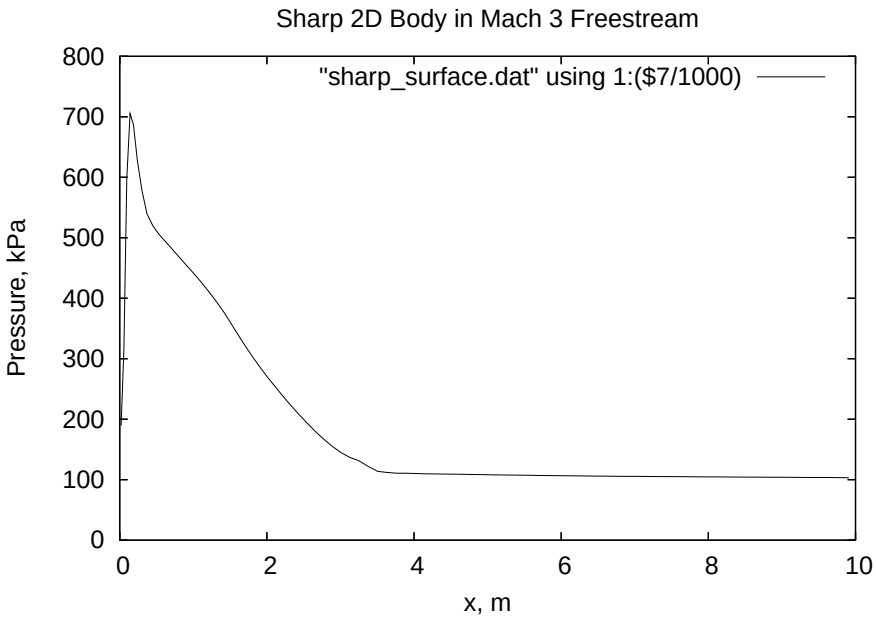


Figure 15: Pressure data along the body surface.

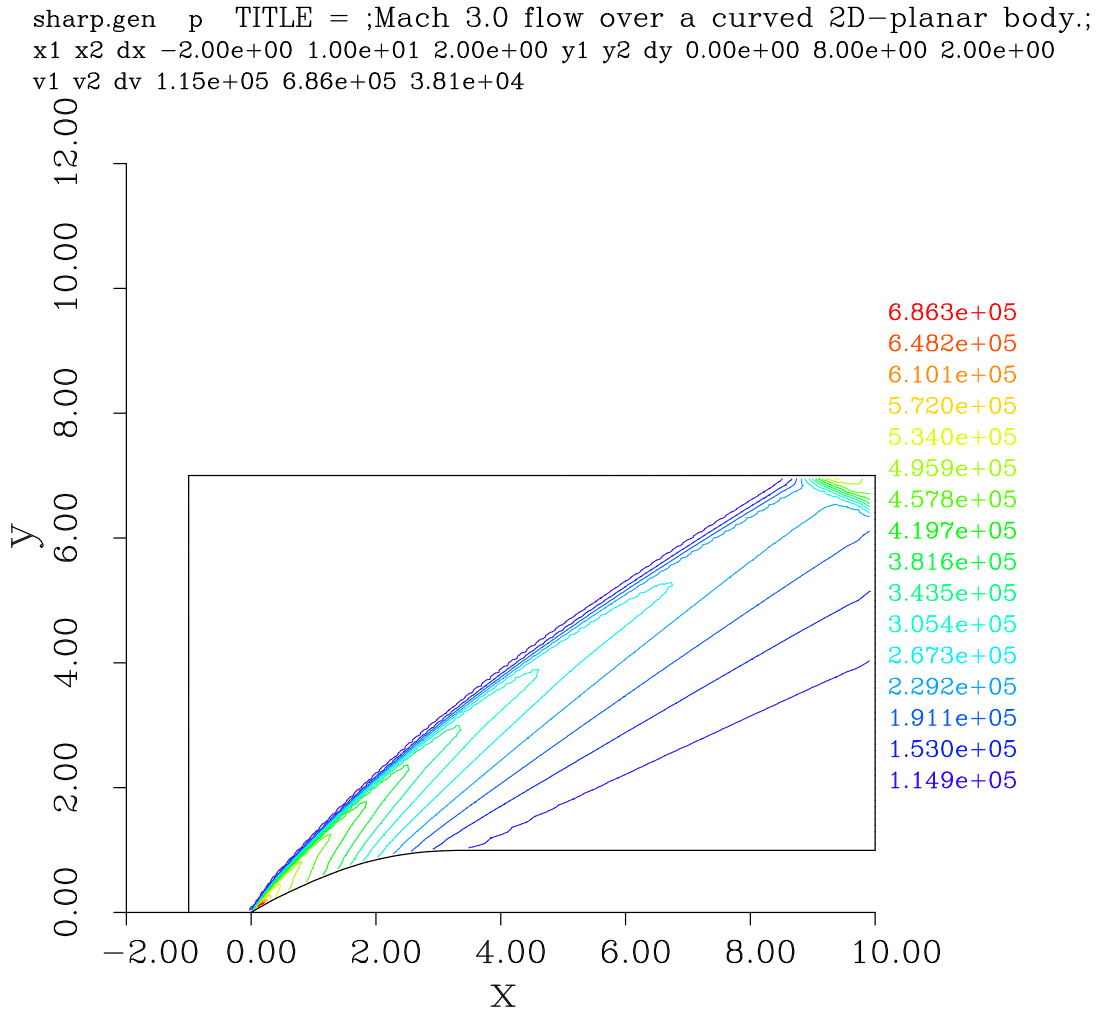


Figure 16: The pressure field for flow over a sharp body. Note that the shock reflects from the upper boundary, which has a SLIP_WALL boundary condition by default.

5.1 .py file

```
# sharp.py
# PJ, 14-Dec-2006
job_title = "Mach 3.0 flow over a curved 2D-planar body."
print job_title
gdata.title = job_title

# Accept defaults giving perfect air (gamma=1.4)
create_perf_gas()
gdata.set_gas_model()
# Define flow conditions
initial = FlowCondition(p=5955.0,  u=0.0,    v=0.0, T=304.0,  mf=[1.0,])
inflow  = FlowCondition(p=95.84e3, u=2000.0, v=0.0, T=1103.0, mf=[1.0,])

# Geometry
def shape(x):
    return -0.008333 + 0.609425*x - 0.092593*x*x

a = Node(-1.0, 0.0, label="A")
b = Node( 0.0, 0.0, label="B")
x_list = [0.5, 1.5, 2.5, 3.291]
b_list = [b,] # to accumulate points in the spline
for x in x_list:
    b_list.append(Node(x, shape(x)))
c = Node(10.0, b_list[-1].y, label="C") # extend at same y-value
d = Node(10.0, 7.0, label="D")
e = Node( 0.0, 7.0, label="E")
f = Node(-1.0, 7.0, label="F")

north0 = Line(f, e)
e0w1 = Line(b, e)
south0 = Line(a, b)
west0 = Line(a, f)
south1 = Polyline([Spline(b_list), Line(b_list[-1], c)])
north1 = Line(e, d)
east1 = Line(c, d)

# Define the blocks, boundary conditions and set the discretisation.
ny = 60
clustery = RobertsClusterFunction(1, 0, 1.3)
clusterx = RobertsClusterFunction(1, 0, 1.2)
blk_0 = Block2D(make_patch(north0, e0w1, south0, west0),
                nni=16, nnj=ny,
                cf_list=[None,clustery,None,clustery],
                fill_conditions=initial)
blk_1 = Block2D(make_patch(north1, east1, south1, e0w1),
                nni=80, nnj=ny,
                cf_list=[clusterx,None,clusterx,clustery],
                fill_conditions=initial)
identify_block_connections()
blk_0.bc_list[WEST]=SupInBC(inflow)
blk_1.bc_list[EAST]=ExtrapolateOutBC()


# Do a little more setting of global data.
gdata.flux_calc = ADAPTIVE
gdata.max_time = 15.0e-3  # seconds
gdata.max_step = 2500
gdata.dt = 1.0e-6

sketch.xaxis(0.0,10.0, 2.0, -0.6)
sketch.yaxis(0.0, 8.0, 2.0, -1.6)
sketch.window(0.0, 0.0, 10.0, 10.0, 0.05, 0.05, 0.17, 0.17)
```

5.2 Shell scripts

```
#! /bin/sh
# sharp_prep.sh
# A sharp axisymmetric body as described in Andersons Hypersonics text.

mbcns_prep.py --job=sharp --do-mpost --do-svg
mpost sharp.mpost

# Extract the initial solution data and reformat.
mbcns_post.exe -fp sharp.p -fg sharp.g -fs sharp.s0 -fo sharp_0 -generic

# Pick up the reformatted data and make a mesh plot.
mb_cont.exe -fi sharp_0.gen -fo sharp_0_mesh.ps -var 6 -ps -mesh \
    -mirror -xrange -2.0 10.0 2.0 -yrange 0.0 8.0 2.0

echo At this point, we should be ready to start the simulation.
```

```
#! /bin/sh
# sharp_run.sh
# Exercise the Navier-Stokes solver for a sharp 2D body.

# Integrate the solution in time.
time mbcns_shared.exe --job=sharp --run

echo At this point, we should have a final solution in sharp.s
```

```
#! /bin/sh
# sharp_post.sh
# Sharp 2D body, extract data and plot it.

# Extract the solution data over whole flow domain and reformat.
mbcns_post.exe -fp sharp.p -fg sharp.g -fs sharp.s -fo sharp -generic \
    -t 15.0e-3

# Pick up the reformatted data and make a contour plot.
mb_cont.exe -fi sharp.gen -fo sharp.gif -var 6 -gif -colour -mirror \
    -xrange -2.0 10.0 2.0 -yrange 0.0 8.0 2.0

mb_cont.exe -fi sharp.gen -fo sharp_p.ps -var 6 -ps -colour -mirror \
    -xrange -2.0 10.0 2.0 -yrange 0.0 8.0 2.0

# Extract surface pressure and plot.
mbcns_profile.exe -fp sharp.p -fg sharp.g -fs sharp.s -fo sharp_surface.dat \
    -yline 1 1 1 -t 15.0e-3

gnuplot <<EOF
set term postscript eps 20
set output "sharp_surface_p.eps"
set title "Sharp 2D Body in Mach 3 Freestream"
set xlabel "x, m"
set ylabel "Pressure, kPa"
set xrange [0.0:10.0]
set yrange [0.0:800]
plot "sharp_surface.dat" using 1:(\$7/1000) with lines
EOF

echo At this point, we should have a plotted data.
```

5.3 Notes

- This simulation reaches a final time of 15 ms in 1801 steps and, on a Pentium-M 1.73 Ghz system, this takes 2 min, 48 s of CPU time.

6 Flow through a conical nozzle

Good quality experimental data for wall pressure distribution in a conical nozzle with a circular-arc throat profile and a 15^o divergent section is available in Ref. [5]. In the original experiment the flow of air through the facility was allowed to reach steady state and static pressures were measured at a large number of points along the nozzle wall. In contrast, the present simulation is transient with just the transonic plus supersonic parts of the flow field reaching steady state.

Figure 17 shows the outline of the simulated flow domain which is set up to approximate the largest subsonic area ratio used in the experiment. Note that the geometric calculation of the tangent arcs is done within the input script. This makes use of Python, beyond just being an input format, and allows the specification to be fully parametric. This makes the initial setup of the script a bit more complex than absolutely necessary but does make the running of the simulation for other radii of curvature very simple.

The relatively long upstream part of the simulated tube provides the gas through an unsteady expansion from zero speed and pressure of 500 kPa (state 4) up to a small Mach number (state 3). These state labels refer to the those for the hypothetical shock tube problem in which state 1 is the initial low-pressure condition, state 4 is the initial high-pressure condition, state 2 is the post-shock condition of the low-pressure gas, and state 3 is the expanded high-pressure gas condition. Assuming that flow in the subsonic and transonic regions of the nozzle is steady, the expected Mach number is $M_3 = 0.13812$ for an area ratio of $A_3/A_* = 4.2381$.

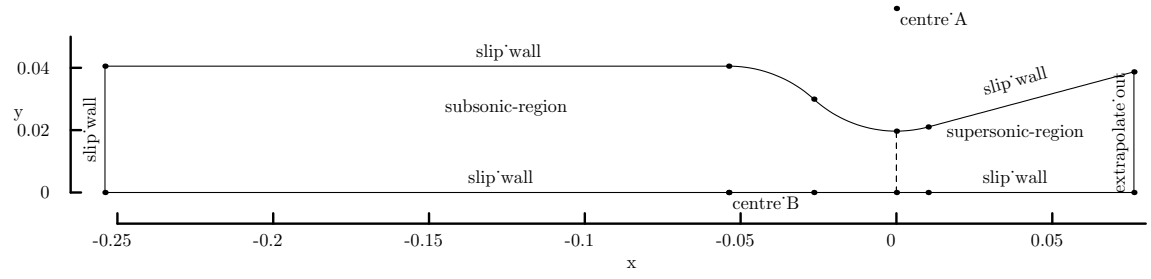


Figure 17: Schematic diagram of the full flow domain for the duct and conical nozzle.

Along the C_+ characteristic connecting states 4 and 3, the Riemann invariant can be written as

$$J_+ = u + \frac{2a}{\gamma - 1} \quad ,$$

and so the relation between states 4 and 3 can be written as

$$\frac{a_4}{a_3} = 1 + \frac{\gamma - 1}{2} M_3 \quad .$$

Because the unsteady expansion is isentropic and $a = \sqrt{\gamma RT}$, the pressure ratio can be

written as

$$\frac{p_4}{p_3} = \left[1 + \frac{\gamma - 1}{2} M_3\right]^{2\gamma/(\gamma-1)}$$

which gives a specific pressure ratio of $p_4/p_3 = 1.2102$. Since the experiment used a steady expansion from a large reservoir at stagnation conditions to the equivalent of our state 3, the corresponding stagnation pressure for state 3 is computed from

$$\frac{p_{03}}{p_3} = \left[1 + \frac{\gamma - 1}{2} M_3^2\right]^{\gamma/(\gamma-1)}$$

which gives $p_{03} = 418.7\,\text{kPa}$ in the current simulation.

Figure 18 shown part of the mesh around the nozzle overlaid with pressure contours. Figure 19 shows the pressure distribution throughout the flow domain at $t = 1.0\,\text{ms}$. A shock, starting at the transition from circular arc to straight wall in the early supersonic part of the nozzle, can be seen propagating toward the centreline as the flow proceeds to the exit plane.

The flow in the nozzle is reasonably steady, as indicated by the histories shown in Fig. 20 but the unsteady expansion can be seen reflecting from the inflow boundary at $x = -0.245\,\text{m}$ in Fig.19.

Figure 21 shows that the simulation matches the experimental data closely. The reflected expansion is shown clearly in the left figure and indicates that the subsonic boundary condition is not working as well as it should.

6.1 .py file

```
# back.py
# Conical nozzle from Back, Massier and Gier (1965)

job_title = "Flow through a conical nozzle."
print job_title
gdata.title = job_title

# Define gas model and flow conditions.
# Accept defaults giving perfect air (gamma=1.4)
create_perf_gas()
gdata.set_gas_model()
stagnation_gas = FlowCondition(p=500.0e3, T=300.0, mf=[1.0,])
low_pressure_gas = FlowCondition(p=30.0, T=300.0, mf=[1.0,])

# Define geometry.
# The original paper specifies sizes in inches, mbcns2 works in metres.
inch = 0.0254 # metres
L_subsonic = 10.0 * inch
L_nozzle = 3.0 * inch
R_tube = 1.5955 * inch
R_throat = 0.775 * inch
R_curve = 1.55 * inch # radius of curvature of throat profile
theta = 15.0 * math.pi / 180.0 # radians

# Compute the centres of curvature for the contraction profile.
height = R_throat + R_curve
hypot = R_tube + R_curve
base = math.sqrt(hypot*hypot - height*height)
centre_A = Node(0.0, height, label="centre_A")
```

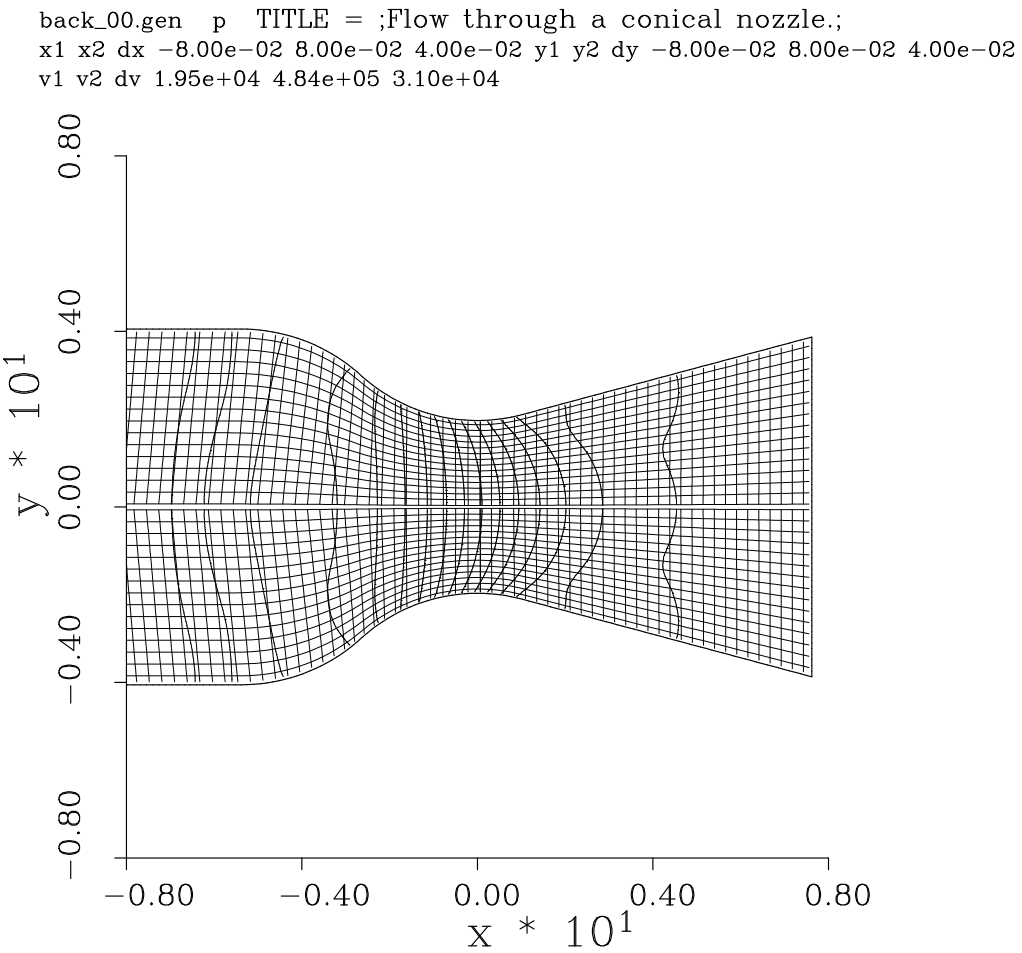


Figure 18: Mesh of lines joining the centres of every-second finite-volume cell with pressure contours superimposed. Only part of the flow domain is shown.

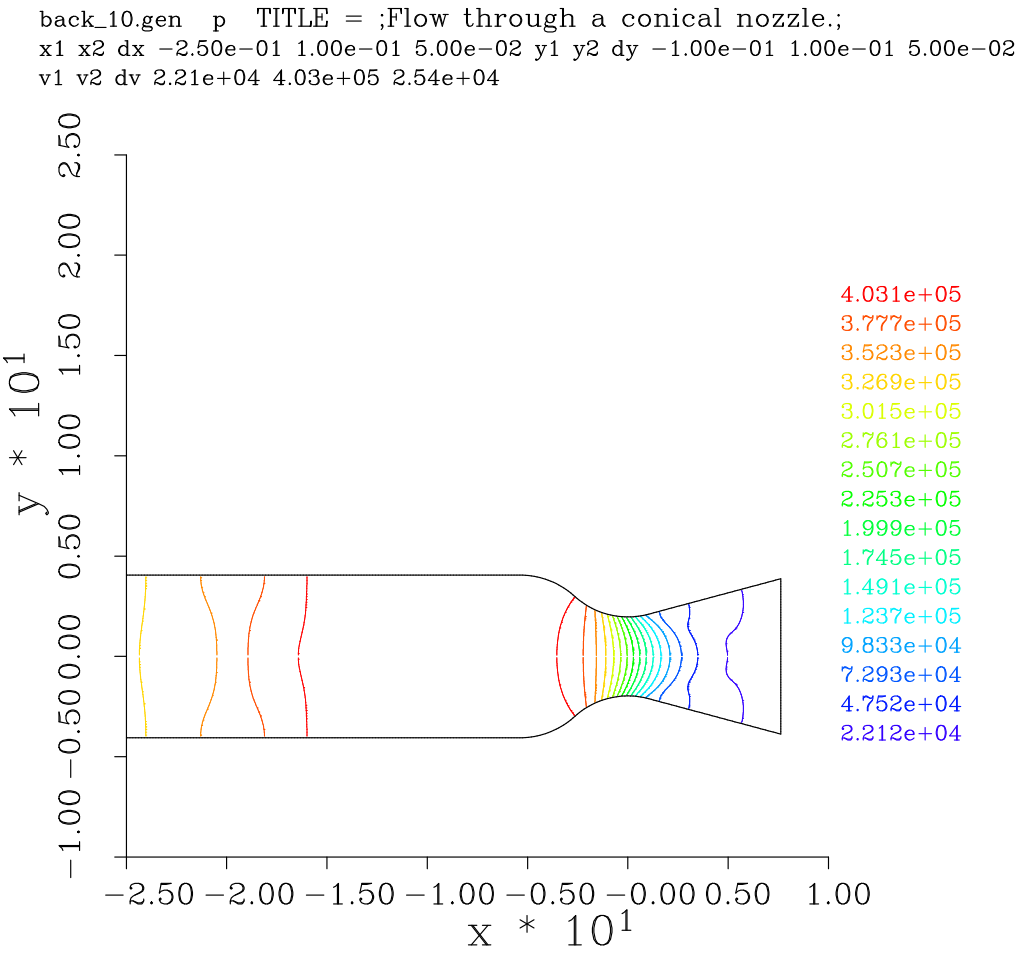


Figure 19: Pressure contours within the flow domain at 1.0 ms.

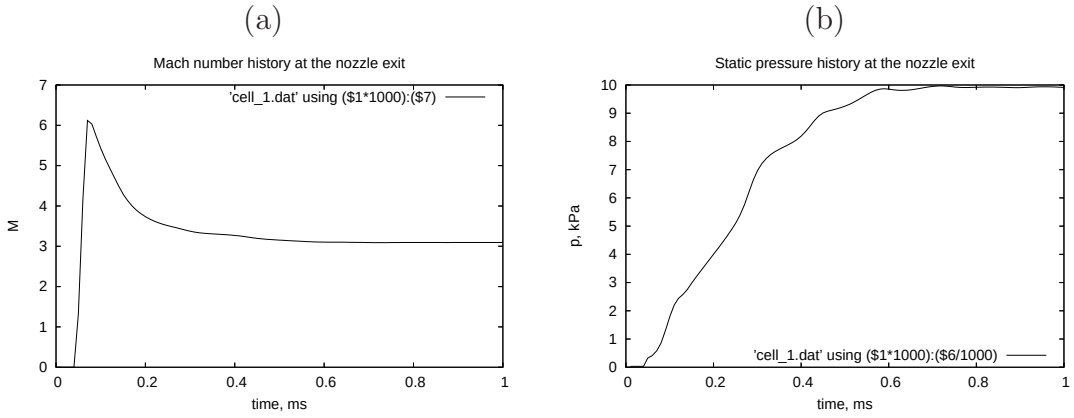


Figure 20: Development of the flow at a "history point" near the centre of the exit plane: (a) Mach number; (b) static pressure.

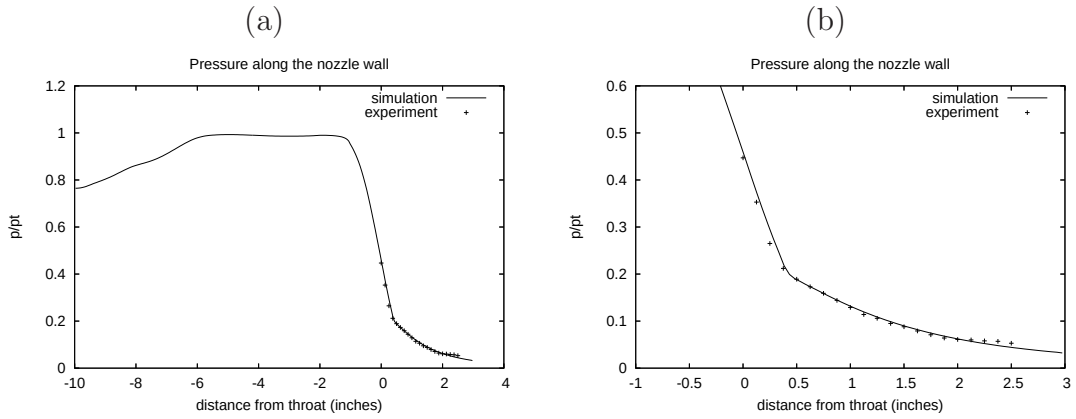


Figure 21: Normalised pressure distribution along the nozzle wall: (a) full length of flow domain; (b) just the supersonic part of the nozzle.

```
centre_B = Node(-base, 0.0, label="centre_B")
fraction = R_tube/hypot
intersect_point = centre_B + Vector(fraction*base, fraction*height)

# The following Nodes will be rendered in the SVG and MetaPost files.
z0 = Node(-L_subsonic, 0.0)  # assemble from coordinates
p0 = Node(-L_subsonic, R_tube)
z1 = Node(centre_B)  # initialize from a previously defined Node
p1 = Node(centre_B + Vector(0.0,R_tube))  # vector sum
p2 = Node(intersect_point)
z2 = Node(p2.x, 0.0)  # on the axis, below p2
z3 = Node(0.0, 0.0)
p3 = Node(0.0, R_throat)
# Compute the details of the conical nozzle
p4 = Node(R_curve*math.sin(theta), height - R_curve*math.cos(theta))
z4 = Node(p4.x, 0.0)
L_cone = L_nozzle - p4.x
p5 = Node(p4 + Vector(L_cone, L_cone*math.tan(theta)))
z5 = Node(p5.x, 0.0)

# Print coordinates to check against old TCL input file
print "p0=", p0, "p1=", p1, "p2=", p2
print "p3=", p3, "p4=", p4, "p5=", p5

north0 = Polyline([Line(p0,p1),Arc(p1,p2,centre_B),Arc(p2,p3,centre_A)])
east0west1 = Line(z3, p3)
south0 = Line(z0, z3)
west0 = Line(z0, p0)
north1 = Polyline([Arc(p3,p4,centre_A), Line(p4,p5)])
east1 = Line(z5, p5)
south1 = Line(z3, z5)

# Define the blocks, boundary conditions and set the discretisation.
nx0 = 180; nx1 = 60; ny = 30
subsonic_region = Block2D(make_patch(north0, east0west1, south0, west0),
                          nni=nx0, nnj=ny,
                          fill_conditions=stagnation_gas,
                          label="subsonic-region")
supersonic_region = Block2D(make_patch(north1, east1, south1, east0west1),
                            nni=nx1, nnj=ny,
                            fill_conditions=low_pressure_gas,
                            label="supersonic-region")
identify_block_connections()
supersonic_region.bc_list[EAST] = ExtrapolateOutBC()

# Flow-history to be recorded at the following points.
HistoryLocation(0.001, 0.002, "nozzle-throat")
HistoryLocation(L_nozzle-0.001, 0.002, "nozzle-exit")
```

```
# Do a little more setting of global data.
gdata.axisymmetric_flag = 1
gdata.flux_calc = ADAPTIVE
gdata.max_time = 1.0e-3  # seconds
gdata.max_step = 5000
gdata.dt = 1.0e-7
gdata.dt_plot = 0.2e-3
gdata.dt_history = 10.0e-6

sketch.xaxis(-0.250, 0.080, 0.050, -0.01)
sketch.yaxis( 0.0,   0.050, 0.020, -0.015)
sketch.window(-0.25, 0.0, 0.10, 0.05, 0.03, 0.03, 0.24, 0.06)
```

6.2 Shell scripts

```
#!/bin/sh
#  back.bat
#  Exercise the Navier-Stokes solver for the conical nozzle
#  as used by Back, Massier and Gier (1965) AIAA J. 3(9):1606-1614.

#  It is assumed that the path is set and that
#  the script file "back.py" has been correctly written.

#  Stage 1:
#  Generate the parameter file, grid file and initial solution
#  (back.p, back.g and back.s0 respectively)
#  from the script file.

mbcns_prep.py --job=back --do-mpost --do-svg
mpost back.mpost

#  Stage 2:
#  Pick up the grid and initial solution files
#  and integrate the solution in time.
#  The solution data at later times will be written
#  to the solution output file "back.s"

time mbcns_shared.exe --job=back --run

@echo "Finished simulation."
```

```
#!/bin/sh
#  back_plot.sh

# Stage 4: Extract particular times from the solution set.
# The following line extracts the first solution in the "back.s" file.
#
mbcns_post.exe -fp back.p -fg back.g -fs back.s \
               -t 0.0 -fo back_00 -generic

# Pick up the reformatted data and make a plot of the mesh.
# Note that this plotted mesh is created by joining cell-centres.
# It is not the *true* mesh used by the flow solver.
# The ixskip, iyskip also allow the plotted mesh to be coarser
# than the true mesh.
#
mb_cont.exe -fi back_00.gen -fo back_00.ps -ps -mesh -var 6 \
            -ixskip 2 -iyskip 2 -mirror \
            -xrange -0.080 0.080 0.040 -yrange -0.080 0.080 0.040

# Static pressure contours after nozzle-flow settles.
#
```

```
mbcns_post.exe -fp back.p -fg back.g -fs back.s \
           -t 1.0e-3 -fo back_10 -generic

mb_cont.exe -fi back_10.gen -fo back_10.ps -ps -colour -var 6 \
           -mirror \
           -xrange -0.250 0.100 0.050 -yrange -0.100 0.100 0.050

# Finished:
```

```
# back_history.sh
# Extract the flow history data at the nozzle exit plane.
# This is then plotted using gnuplot and an assessment
# can be made as to whether the flow has reached steady state.

mbcns_history.exe -fi back.h -fo cell_1.dat -ncell 2 -cell 1

gnuplot <<EOF
set term postscript eps 20
set output 'back_history_M.eps'

set title 'Mach number history at the nozzle exit'
set xrange [0.0:1.0]
set xlabel 'time, ms'
set ylabel 'M'

plot 'cell_1.dat' using (\$1*1000):(\$7) with lines
EOF

gnuplot <<EOF
set term postscript eps 20
set output 'back_history_p.eps'

set title 'Static pressure history at the nozzle exit'
set key bottom right
set xrange [0.0:1.0]
set xlabel 'time, ms'
set ylabel 'p, kPa'

plot 'cell_1.dat' using (\$1*1000):(\$6/1000) with lines
EOF
```

6.3 Notes

- The simulation reaches a final time of 1 ms in 1431 steps and, on a Pentium-M 1.73 Ghz system, this takes 2 min, 50 s of CPU time. This is equivalent to 16.5 μs per cell per predictor-corrector time step.

- The pressure is normalised with respect to the stagnation pressure using the following AWK script.

```
# normalize.awk
# Normalize the surface pressure over the length of the nozzle.
BEGIN {
    p0 = 418.7e3
    print "# Normalized surface pressure for the Back nozzle (simulation)"
    print "# x(inches) p/pt"
}

$1 != "#" {  # For non-comment lines in the data file do...
```

```
    p = $7
    r = $2
    x = $1
    print x/0.0254, p/p0
}
```

7 A section of an ideal compressible-flow vortex

This flow example was used by Ian Johnston in his thesis and it comes with an analytic solution [6]. With respect to MB_CNS, it illustrates the use of a specified flow profile as an input and it shows the use of profile extraction, again.

The flow domain (Fig. 22) includes only part of the first quadrant of an ideal vortex flow in inviscid air with $R = 287\text{J/kg·K}$, γ=1.4). The NORTH and SOUTH boundaries are specified as reflecting walls at radii r_o and r_i, representing the outer and inner radii of the vortex segment that is centred at node A. The WEST boundary has the specified inflow as a function of radius

$$\begin{aligned} \rho(r) &= \rho_i \left[1 + \frac{\gamma - 1}{2} M_i^2 \left\{ 1 - \left(\frac{r_i}{r} \right)^2 \right\} \right]^{\frac{1}{\gamma - 1}} \quad , \\ p(r) &= p_i \left(\frac{\rho}{\rho_i} \right)^{\gamma} \quad , \\ u(r) &= u_i \frac{r_i}{r} \quad , \end{aligned}$$

with $r_o = 1.384 r_i$ and the properties at the inner radius being $M_i = 2.25$, $\rho_i = 1.0\,\text{kg/m}^3$ and $p_i = 100\,\text{kPa}$.

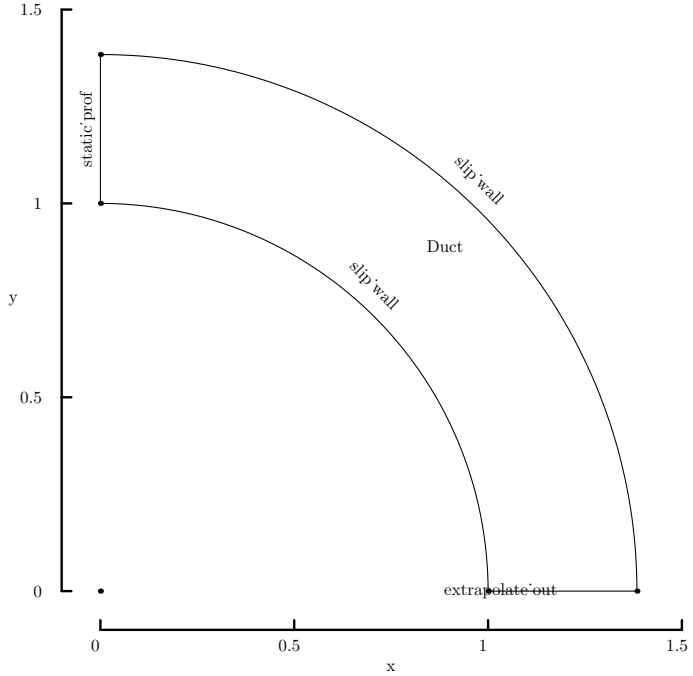


Figure 22: Schematic diagram of the first quadrant domain for the compressible-flow vortex.

Figures 23 through 25 show the radial distributions of flow properties and highlight some of the problems with the crude reflecting-wall boundary condition. Other than at the boundaries, there is close agreement between the analytic and numerical solutions.

The errors at the inner and outer radii stand out clearly because we know that the trends of the flow property variations should continue at these boundaries and not mirror what is just inside the flow domain.

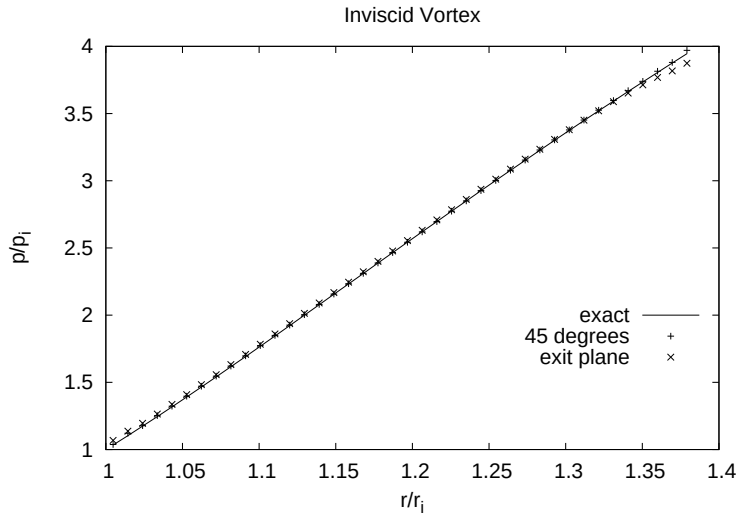


Figure 23: Radial distributions of pressure.

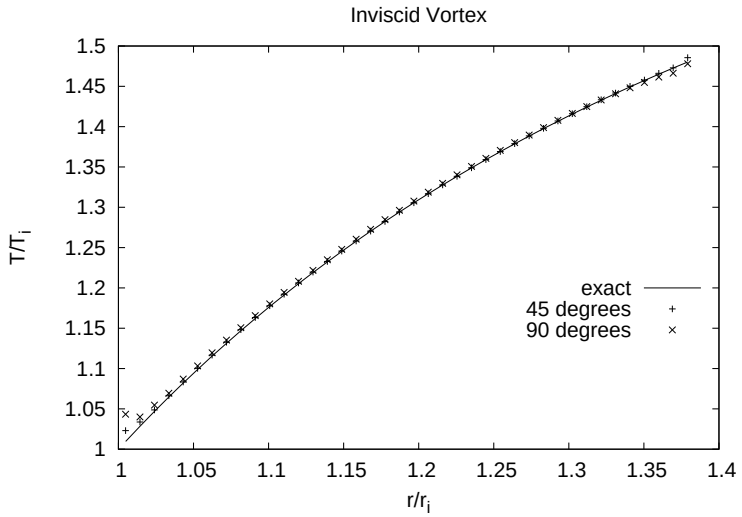


Figure 24: Radial distributions of temperature.

7.1 .py file

```
# file: vtx.py
# PJ, 14-Dec-2006
gdata.title = "Inviscid supersonic vortex -- flow in a bend."

# Geometry
```

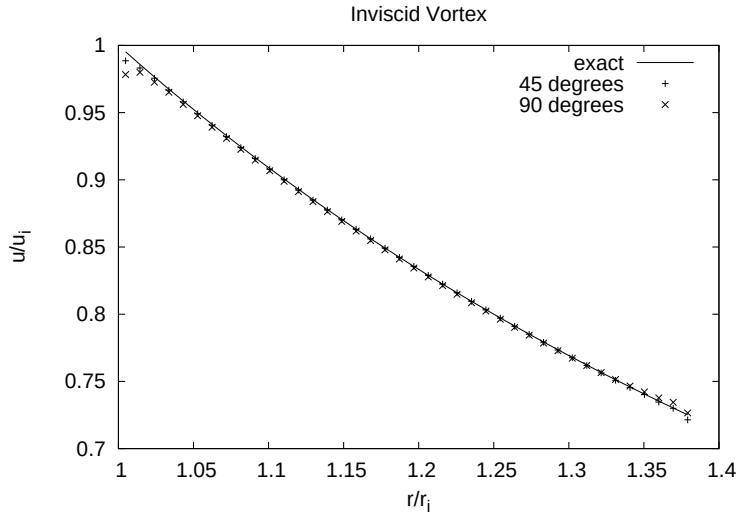


Figure 25: Radial distribution of circumferential velocity.

```
R_inner = 1.0
R_outer = 1.384
a = Node(0.0, 0.0)
b = Node(0.0, R_inner)
c = Node(0.0, R_outer)
d = Node(R_inner, 0.0)
e = Node(R_outer, 0.0)
north0 = Arc(c, e, a)
east0 = Line(d, e)
south0 = Arc(b, d, a)
west0 = Line(b, c)

# The following are not really important because the
# actual data will be taken from the profile.dat file.
create_perf_gas() # Accept defaults giving perfect air (gamma=1.4)
gdata.set_gas_model()
initial = FlowCondition(p=1000.0, u=0.0,     v=0.0, T=348.43, mf=[1.0,])

blk_0 = Block2D(psurf=make_patch(north0, east0, south0, west0),
                fill_conditions=initial,
                nni=80, nnj=40,
                bc_list=[SlipWallBC(), ExtrapolateOutBC(),
                         SlipWallBC(), StaticProfBC()],
                label="Duct")

# Simulation-control information
gdata.flux_calc = ADAPTIVE
gdata.max_time = 20.0e-3
gdata.max_step = 6000
gdata.dt = 1.0e-6
gdata.dt_plot = 5.0e-3

# Some hints to scale and place the Metapost figure.
sketch.xaxis(0.0, 1.5, 0.5, -0.1)
sketch.yaxis(0.0, 1.5, 0.5, -0.1)
sketch.window(0.0, 0.0, 1.5, 1.5, 0.05, 0.05, 0.17, 0.17)
```

7.2 Shell scripts

```
#! /bin/sh
# vtx_run.sh
# Exercise the Navier-Stokes solver for the inviscid vortex test case.
# It is assumed that the path is set correctly.

mbcns_prep.py --job=vtx --do-mpost --do-svg
mpost vtx.mpost

# Generate the inflow profile.
awk -f make_profile.awk

# Integrate the solution in time.
time mbcns_shared.exe --job=vtx --run

echo At this point, we should have a computed solution in vtx.s
```

```
#! /bin/sh
# vtx_plot.sh
# Exercise the Navier-Stokes solver for the inviscid vortex test case.
# It is assumed that the path is set correctly.

T_FINAL=20.0e-3

# Extract the solution data and reformat.
mbcns_post.exe -fp vtx.p -fg vtx.g -fs vtx.s -fo vtx -t $T_FINAL -generic

# Pick up the reformatted data and make a contour plot.
mb_cont.exe -fi vtx.gen -fo vtx.ps -var 6 -ps -colour \
   -xrange 0.0 1.5 0.5 -yrange 0.0 1.5 0.5

# Extract radial profiles at 45 degrees and at 90 degrees from the inlet.
mbcns_profile.exe -fp vtx.p -fg vtx.g -fs vtx.s -fo vtx_profile_45.dat \
   -t $T_FINAL -xline 0 0 40
awk -f extract_radial.awk vtx_profile_45.dat > radial_profile_45.dat

mbcns_profile.exe -fp vtx.p -fg vtx.g -fs vtx.s -fo vtx_profile_90.dat \
   -t $T_FINAL -xline 0 0 80
awk -f extract_radial.awk vtx_profile_90.dat > radial_profile_90.dat

# Generate postscript plots of the radial profiles.
gnuplot radial_profile.gnu

echo At this point, we should have a plotted the solution
```

7.3 Notes

- This simulation reaches a final time of 20 ms in 2717 steps and, on a Pentium-M 1.73 Ghz system, this takes 2 min, 16 s of CPU time.
- The inflow that was applied to the WEST boundary as a "Static Profile" was generated with the following AWK script and written to the file `profile.dat`. MBCNS2 looks for this file when the `StaticProfBC()` boundary condition is used. See comments in the `init_profile_data()` function in the C-module `cns_bc.cxx` for details of the expected file format.

```
# make_profile.awk
# Set up an inflow profile for the inviscid vortex case
```

```
# PJ, 20-Feb-01, 14-Dec-06 write 1.0 for mass-fraction[0]
#
function pow( base, exponent ) {
   # print base, exponent
   return exp( exponent * log(base) )
}

BEGIN {
   Rgas  = 287       # J/kg.K
   g     = 1.4       # ratio of specific heats

   n     = 40
   r_i   = 1.0       # metres
   r_o   = 1.384
   dr    = (r_o - r_i) / n

   # Set flow properties ar the inner radius.
   p_i   = 100.0e3                  # kPa
   M_i   = 2.25
   rho_i = 1.0                      # kg/m**3
   T_i   = p_i / (Rgas * rho_i)     # K
   a_i   = sqrt( g * Rgas * T_i )   # m/s
   u_i   = M_i * a_i                # m/s
   # print p_i, M_i, rho_i, T_i, a_i, u_i

   # Generate the profile along the radial direction.
   print n > "profile.dat"
   for ( i = 1; i <= n; ++i ) {
      r   = r_i + dr * (i - 0.5)
      # print "i= ", i, "r=", r
      u   = u_i * r_i / r
      t1  = r_i / r
      t2  = 1.0 + 0.5 * (g - 1.0) * M_i * M_i * (1.0 - t1 * t1)
      rho = rho_i * pow( t2, 1.0/(g - 1.0) );
      p   = p_i * pow( rho/rho_i, g )
      T   = p / (rho * Rgas)
      print p, u, 0.0, T, 1.0 > "profile.dat"
      print r/r_i, p/p_i, u/u_i, 0.0, T/T_i, 1.0 > "radial_profile_0.dat"
   } # end for
}
```

- The plots were generated via the following scripts

```
# extract_radial.awk
# Extract the radial profile data from mb_prof.exe generated files.
BEGIN{
   r_i = 1.0; p_i = 100.0e3; u_i = 841.87; T_i = 348.43;
}

$1 != "#" {
   x = $1; y = $2; p = $7; u = $4; v = $5; T = $10
   r = sqrt( x * x + y * y )
   speed = sqrt( u * u + v * v )
   print r/r_i, p/p_i, speed/u_i, 0.0, T/T_i
}
```

```
# radial_profile.gnu

set term postscript eps enhanced 20
set output "radial_profile_p.eps"
set title "Inviscid Vortex"
set xlabel "r/r_i"
set ylabel "p/p_i"
```

```
# set yrange [1.0:4.5]
set key 1.35, 2
plot "radial_profile_0.dat" using 1:2 title "exact" with lines, \
     "radial_profile_45.dat" using 1:2 title "45 degrees", \
     "radial_profile_90.dat" using 1:2 title "exit plane"

set term postscript eps enhanced 20
set output "radial_profile_u.eps"
set title "Inviscid Vortex"
set xlabel "r/r_i"
set ylabel "u/u_i"
# set yrange [0.7:1.0]
set key
plot "radial_profile_0.dat" using 1:3 title "exact" with lines, \
     "radial_profile_45.dat" using 1:3 title "45 degrees", \
     "radial_profile_90.dat" using 1:3 title "90 degrees"

set term postscript eps enhanced 20
set output "radial_profile_T.eps"
set title "Inviscid Vortex"
set xlabel "r/r_i"
set ylabel "T/T_i"
# set yrange [1.0:1.7]
set key 1.35, 1.2
plot "radial_profile_0.dat" using 1:5 title "exact" with lines, \
     "radial_profile_45.dat" using 1:5 title "45 degrees", \
     "radial_profile_90.dat" using 1:5 title "90 degrees"
```

8 Pressure on a flat-faced cylinder

This example models the bar gauge type of pressure sensor as used in the expansion-tube facilities. It also shows the application of a multiple-block grid to describe the flow domain (Figure 26) around a flat-faced cylinder whose axis is aligned with the free-stream flow direction. The free-stream Mach number is 4.76 to match one of the higher Mach number conditions reported in Ref.[7].

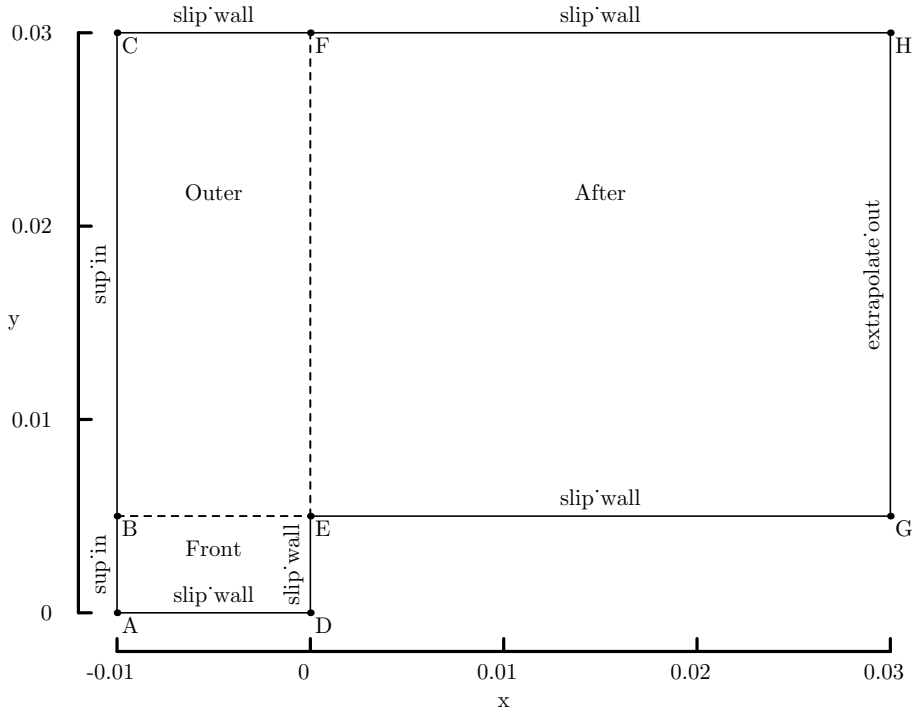


Figure 26: Schematic diagram of the full flow domain around the flat-faced cylinder.

The simulation is started with low pressure stationary gas throughout the domain and the inflow conditions are applied to the west boundaries of blocks “Front” and “Outer”. After allowing 50 μs for the flow to reach steady state, the pressure distribution throughout the domain is shown in Fig. 27. The stand-off distance of 2.814 mm was determined by searching for the pressure jump along the row of cells adjacent to the centreline.

Figure 28 shows the distribution of pressure across the face of the cylinder. The simulation data agrees closely with Kendall’s measurements except in the region the sharp corner where there is inadequate resolution and an absence of viscous effects in the simulation.

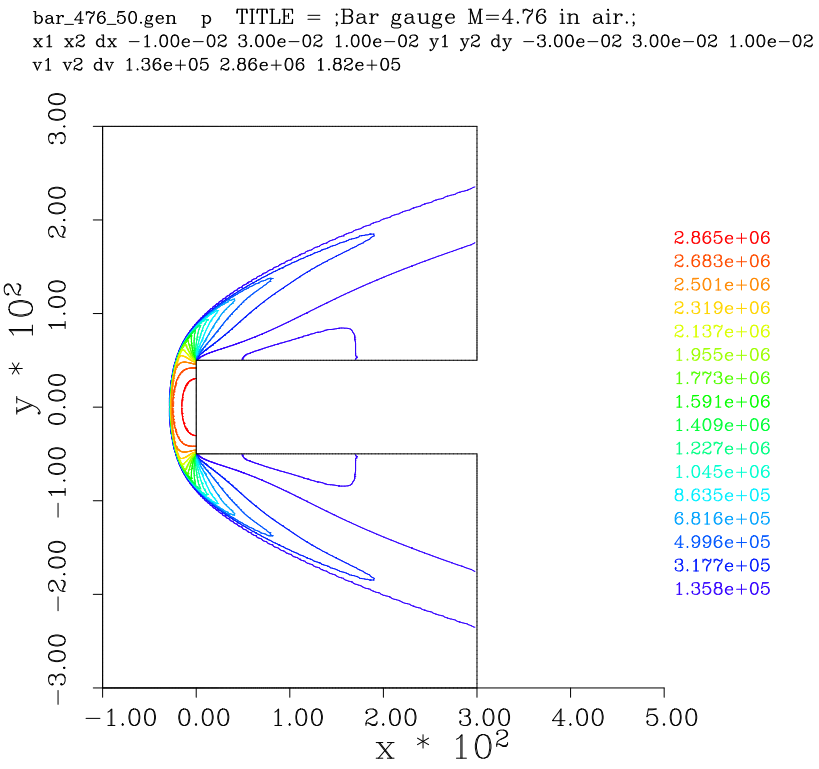


Figure 27: Pressure contours within the flow domain at 50 μs.

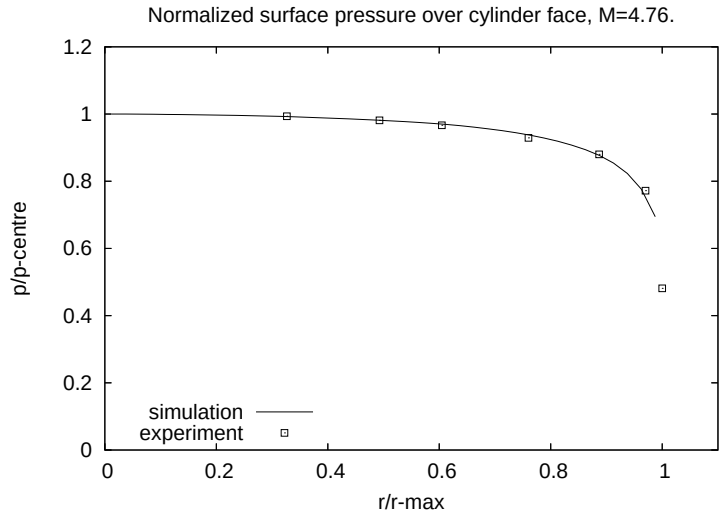


Figure 28: Normalised pressure across the face of the cylinder compared with experimental measurements [7].

8.1 .py file

```
## \file bar_476.py
## \brief Bar gauge (or flat-faced cylinder) M=4.76, ideal air
## \author PJ, 14-Dec-2006
##
job_title = "Bar gauge M=4.76 in air."
print job_title

# Accept defaults giving perfect air (gamma=1.4)
create_perf_gas()
gdata.set_gas_model()

# Define flow conditions
initial = FlowCondition(p=30.0,    u=0.0,    v=0.0, T=300.0, mf=[1.0,])
inflow  = FlowCondition(p=100.0e3, u=1653.0, v=0.0, T=300.0, mf=[1.0,])

# Geometry
R = 5.0e-3    # radius of bar in metres
a = Node(-2*R, 0.0, label="A")
b = Node(-2*R, R,   label="B")
c = Node(-2*R, 6*R, label="C")
d = Node( 0.0, 0.0, label="D")
e = Node( 0.0, R,   label="E")
f = Node( 0.0, 6*R, label="F")
g = Node( 6*R, R,   label="G")
h = Node( 6*R, 6*R, label="H")

ad=Line(a,d); be=Line(b,e); cf=Line(c,f); eg=Line(e,g); fh=Line(f,h)
ab=Line(a,b); bc=Line(b,c); de=Line(d,e); ef=Line(e,f); gh=Line(g,h)

# Define the blocks, boundary conditions and set the discretisation.
nx0 = 120; nx2 = 120; ny0 = 40; ny1=80
cfy = RobertsClusterFunction(1, 0, 1.2)
cfx = RobertsClusterFunction(1, 0, 1.1)
blk_0 = Block2D(make_patch(be, de, ad, ab), nni=nx0, nnj=ny0,
                fill_conditions=initial, label="Front",
                hcell_list=[(nx0,1),(nx0,5),(nx0,10)],
                xforce_list=[0,1,0,0])
blk_1 = Block2D(make_patch(cf, ef, be, bc), nni=nx0, nnj=ny1,
                cf_list=[None,cfy,None,cfy],
                fill_conditions=initial, label="Outer")
blk_2 = Block2D(make_patch(fh, gh, eg, ef), nni=nx2, nnj=ny1,
                cf_list=[cfx,cfy,cfx,cfy],
                fill_conditions=initial, label="After")
identify_block_connections()
blk_0.bc_list[WEST] = SupInBC(inflow)
blk_1.bc_list[WEST] = SupInBC(inflow)
blk_2.bc_list[EAST] = ExtrapolateOutBC()

# We can set individual attributes of the global data object.
gdata.title = job_title
gdata.axisymmetric_flag = 1
gdata.flux_calc = ADAPTIVE
gdata.max_time = 50.0e-6  # seconds
gdata.max_step = 15000
gdata.dt = 2.0e-8
gdata.dt_plot = 5.0e-6
gdata.dt_history = 0.5e-6

sketch.xaxis(-0.010, 0.030, 0.010, -0.002)
sketch.yaxis( 0.000, 0.030, 0.010, -0.002)
sketch.window(-0.010, 0.0, 0.030, 0.040, 0.05, 0.05, 0.17, 0.17)
```

8.2 Shell scripts

```
#!/bin/sh
#  bar_476_run.sh
#  Exercise the Navier-Stokes solver for the bar gauge.

mbcns_prep.py --job=bar_476 --do-mpost --do-svg
mpost bar_476.mpost

time mbcns_shared.exe --job=bar_476 --run

echo "Finished:"
```

```
#!/bin/sh
# bar_476_plot.sh

mbcns_post.exe -fp bar_476.p -fg bar_476.g -fs bar_476.s \
    -t 50.0e-6 -fo bar_476_50 -generic

mb_cont.exe -fi bar_476_50.gen -fo bar_476_50.ps -ps -colour -var 6 \
    -mirror -xrange -0.010 0.030 0.010 -yrange -0.030 0.030 0.010

echo "Finished:"
```

```
# bar_476_profile.sh
# Extract the flow data across the face of the bar gauge.

mbcns_profile.exe -fp bar_476.p -fg bar_476.g -fs bar_476.s -fo raw_profile.dat \
   -t 50.0e-6 -xline 0 0 120

awk -f normalize.awk raw_profile.dat > norm_profile.dat

gnuplot <<EOF
set output "bar_476_norm_p.eps"
set term postscript eps 20
set xrange [0:1.1]
set yrange [0:1.2]
set title "Normalized surface pressure over cylinder face, M=4.76."
set xlabel "r/r-max"
set ylabel "p/p-centre"
set key bottom left
plot "norm_profile.dat" using 1:2 title "simulation" with lines, \
     "kendall_profile.dat" using 1:2 title "experiment" with points 4
EOF
```

```
#!/bin/sh
# bar_476_standoff.sh

mbcns_profile.exe -fp bar_476.p -fg bar_476.g -fs bar_476.s \
    -fo bar_476_stag_line.dat \
    -t 50.0e-6 -yline 0 0 1

awk -f locate_shock.awk bar_476_stag_line.dat
```

8.3 Notes

- The simulation reaches a final time of 50 μs in 2932 steps and, on a Pentium-M 1.73 Ghz system, this takes 19 min, 27 s of CPU time. This is equivalent to 17.8 μs per cell per predictor-corrector time step.
- The surface pressure is normalised with respect to the stagnation pressure after the bow shock, using the following AWK script.

```
# normalize.awk
# Normalize the surface pressure over the centreline static pressure.
BEGIN {
  p_centre = -1.0;
}

$1 != "#" {
  p = $7;
  r = $2;
  if (p_centre < 0.0) p_centre = p;
  print r/0.005, p/p_centre;
}
```

- Along a row of cells that have been extracted using `mb_prof`, the shock is detected using the following AWK script.

```
# locate_shock.awk

BEGIN {
    p_old = 0.0;
    x_old = -2.0;
    p_trigger = 200.0e3;
    shock_found = 0;
}

$1 != "#" { # for any non-comment line
    p_new = $7;
    x_new = $1;
    # print "p_new=", p_new, "x_new", x_new
    if ( p_new > p_trigger && shock_found == 0 ) {
        shock_found = 1;
        frac = (p_new - p_trigger) / (p_new - p_old);
        x = x_old + frac * (x_new - x_old);
        print "shock located at x = ", x
    }
    p_old = p_new;
    x_old = x_new;
}

END {
    if ( shock_found == 0 ) {
        print "shock not located";
    }
    print "done."
}
```

References

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

[2] Ames Research Staff. Equations, tables and charts for compressible flow. NACA Report 1135, 1953.

[3] K. Sawada and E. Dendou. Validation of hypersonic chemical equilibrium flow calculations using ballistic-range data. *Shock Waves*, 11:43–51, 2001.

[4] J. D. Anderson. *Hypersonic and High Temperature Gas Dynamics.* McGraw-Hill, New York, 1989.

[5] L. H. Back, P. F. Massier, and H. L. Gier. Comparison of measured and predicted flows through conical supersonic nozzles, with emphasis on the transonic region. *A.I.A.A. Journal*, 3(9):1606–1614, 1965.

[6] A. Aftosmis, D. Gaitonde, and T. S. Tavares. Behaviour of linear reconstruction techniques on unstructured meshes. *A.I.A.A. Journal*, 33(11):2038–2049, 1995.

[7] J. M. Kendall. Experiments on supersonic blunt-body flows. Progress Report 20-372, Jet Propulsopn Laboratory, California Institute of Technology, Pasadena, California., February 1959.