Some results using a new compressible flow solver for OpenFoam

L. Gasparini
Fondmetal Technologies, Italy

A new density-based solver for non-viscous compressible flow has been written for OpenFoam, implementing a so-called Godunov-like central (or central-upwind) scheme. A few test cases have been run relative to 1D and 2D compressible flows with shocks and the results are presented and compared against some of the compressible flow solvers distributed with OpenFoam 1.3. The performance of the new solver is interesting.

I. Introduction

A new density-based solver for non viscous compressible flow is presented, which implements a so-called Godunov-like central (or central-upwind) scheme formulation. This family of schemes, extensively documented in a large number of papers which have appeared in the last few years,[1] shares some of the high-resolution properties of classical (approximate) Riemann-solver based upwind schemes, while being much simpler to implement. Specifically, I have applied the semidiscrete second-order formulation of Kurganov, Noelle and Petrova.[2]

At its present development stage the solver, called ***centralFoam***, is an unsteady explicit solver for the compressible flow of a perfect gas. Space discretisation is second-order accurate and based on the reconstruction of the primitive variables (p, U, T); one among the usual variety of limited schemes available in OpenFoam 1.3 is selected at run time to obtain a non-oscillatory reconstruction. Time integration currently employs the first-order (forward) Euler scheme, similarly to the other solvers considered in the present work.

The implementation of the solver has been done at the application level, and no new library has been developed. Thus, two dummy surfaceScalarFields representing positive and negative fluxes (i.e. respectively, flux from the owner to the neighbor cell and from the neighbor to the owner cell) are used to “fool” the interpolation function and reconstruct the primitive variables on both sides of each cell face, as required by any Godunov-like scheme. A cleaner and probably faster implementation is possible writing an appropriate reconstruction class, but this is currently beyond my level of understanding of C++ and of the OpenFoam software package.

In the following sections some results are shown obtained using *centralFoam* on 1D and 2D test cases; they are compared against the results obtained using three of the compressible flow solvers available within the OpenFoam 1.3 distribution, namely *sonicFoam*, *rhoSonicFoam* and *rhopSonicFoam*.

II. Solution parameters and spatial schemes

Since the explicit scheme adopted in *centralFoam* has a Courant number limitation of *Co* < 0.5, all computations have been done selecting for each case a time step which gives a *Co* of the order of 0.4; the same time step has been used for all the other solvers as well, although they could in principle allow for a larger time step (just talking about stability, not about the accuracy of the solution).

By the way, note that here the Courant number is computed, as typical for compressible flows, using the maximum wave propagation velocity, *max*(*mag*(u_N + c), *mag*(u_N - c)), u_N being the flow velocity component normal to the cell face and c the local sound speed. On the contrary, *sonicFoam* includes `compressibleCourantNo.H` from `src\finiteVolume\cfdTools\compressible`; strangely, this code only considers the convective velocity u_N. Similarly to *sonicFoam* do *rhoSonicFoam* and *rhopSonicFoam* as well, although in these the computation of the Courant number is coded directly into the main solver.

Solver	Solution scheme parameters
centralFoam	`None`

SonicFoam

```
PISO
{
    nCorrectors                 2;
    nNonOrthogonalCorrectors 0;
}
```

Solver	Solution scheme parameters
rhoSonicFoam	`None`

rhopSonicFoam

```
PISO
{
    nOuterCorrectors 3;
    nCorrectors      1;
    HbyAblend        0.0;
}
```

Table 1. Solution parameters.

For *sonicFoam*, *rhoSonicFoam* and *rhopSonicFoam* the solution parameters specified in the tutorial's shockTube case (used also in the tutorial's forwardStep case, for the first two solvers) have been also applied to all the other test cases run; note however, that *rhoSonicFoam* actually does not make use of any PISO parameter, although they are still specified in the fvSolution dictionary. The relevant parameters are summarized in Table 1, omitting the specification of the linear solvers used for the various equations. Note that the new solver also does not have any user-adjustable solution parameter (except for the indication of the linear solvers).

Furthermore, for all cases and solvers the Gamma 1.0 limited interpolation scheme has been used (actually, for the original OpenFoam solvers GammaV 1.0 has been used whenever available). No attempt has been made to optimize either the limiter (i.e. the space discretisation scheme) or the time step for each single case or solver.

III. 1D shock tube

This is the 1D shockTube test case which is part of the set of tutorial cases provided with OpenFoam. The only modification is that, in order to emphasize the differences in the shock-capturing capability of the solvers, a coarser grid of 500 cells in x-direction is used, instead of the original 1000 cells. Note that similar cases are typically shown in the literature on meshes with 100 or 200 cells, 400 cells already being considered a rather fine mesh for such a simple problem.

Figure 1 shows the results for the pressure-based *sonicFoam* solver. It is clear that the predicted shock position (i.e. the shock speed) is wrong (too slow, this will also appear in the unsteady 2D cases to be presented later) and that the shock profile is distributed over a large number of cells, due to excessive dissipation. Also, post-shock temperature is inexact. Thus, at least with the set of solution parameters applied here, *sonicFoam* cannot be considered an accurate solver for unsteady flows with shocks.

Figure 2 shows the results for the density-based *rhoSonicFoam* solver. It gives reasonable results, although the contact discontinuity and the shock are not very sharp, but oscillations develop near the shock and at the end of the expansion.

Figure 3 shows the results for the pressure-and-density-based *rhopSonicFoam* solver. It gives the cleanest results among the three original OpenFoam solvers but the shock and the contact discontinuity are still not very sharp.

From Fig. 4 it appears that the new solver gives much sharper profiles for both the contact discontinuity and the shock than any of the original solver, with only very little oscillations near the contact. It also has better accuracy at the two ends of the expansion area.

Table 2 summarizes the CPU time required by the four solvers on my 3GHz PC under Cygwin. Remind that the time required by the original solvers to complete the simulation could be reduced running at higher *Co* number, although the accuracy might suffer.

Solver	**CPU time (s)**
centralFoam	10.6
sonicFoam	17.2
rhoSonicFoam	6.5
rhopSonicFoam	26.3

Table 2. CPU time for the 1D shockTube case.

IV. 2D Riemann problems

A few 2D unsteady test cases have been run among the set presented by Liska and Wendroff.[3] All cases used a 400x400 cells uniform grid as was done in the cited paper. Again, the time step is selected to give a maximum *Co* not far from 0.4.

A. Case 3

Figure 5 presents the results for this case obtained using all four solvers. As for the other 2D Riemann problem cases the pictures show color levels of pressure (the same scale is used for all solvers) and iso-density contours (using the min, max and step values specified in the cited paper). It is evident that while the solution obtained with *centralFoam* compares well with those shown in the literature all other solvers have severe problems (at least with the current set of solution parameters): *sonicFoam* clearly underpredicts shock speeds, giving a completely wrong answer, while the solution of both *rhoSonicFoam* and *rhopSonicFoam* is corrupted by oscillations.

B. Case 4

The situation is similar for this case too, shown in Fig. 6: *centralFoam* gives good, sharp but non-oscillatory results; *sonicFoam* still underpredicts shock speeds and does not show the proper flow structures; *rhoSonicFoam* and *rhopSonicFoam* are again corrupted by oscillations.

C. Case 6

While the previous cases involved shocks, case 6, shown in Fig. 7, only produces contact discontinuities. Thus, all solvers deliver not too bad (*sonicFoam* and *rhopSonicFoam*, both showing some oscillations) or good (*rhoSonicFoam* and *centralFoam*) results.

D. CPU time

Table 3 summarizes the CPU time required by the four solvers in the three cases. It appears that the new solver is not only accurate but also significantly faster than any other (per iteration). As already mentioned, the original solvers could be run at higher *Co* number (at least ~1, instead of ~0.5) thus requiring less time steps to complete the simulation but the effect on solution accuracy has not been checked.

Solver	CPU time (s)			Average time ratio
	Case 3	Case 4	Case 6	
centralFoam	1980	1659	1947	1.0
sonicFoam	4622	4003	4640	2.4
rhoSonicFoam	2850	2401	2702	1.4
rhopSonicFoam	10182	8409	9943	5.1

Table 3. CPU time for the 2D Riemann problem cases.

It is worth mentioning that because of the very large time required by *rhopSonicFoam* I also tried to reduce the number of `nOuterCorrectors` from 3 to 1, thus making it three times faster. For the 1D shockTube problem this introduces some oscillations at the shock whereas for all the 2D Riemann problem cases the oscillations, already present, become just worse.

V. Oblique shock case

In this well-known steady test case an oblique shock with a 29° incident angle is obtained imposing on the upper boundary of the rectangular domain the exact post shock state (specified with an *inlet* condition on the top side) in a freestream flow at Mach 2.9 (specified with a *supersonicInlet* on the left side). The oblique shock then reflects on the lower symmetry plane. At the outlet an *extrapolatedOutlet* boundary condition is applied. The simulation is run up to Time = 10, when the solution has converged to the steady state.

From Fig. 8 the usual behavior appears, with *sonicFoam* giving a poor, exact but too dissipative, solution, *rhopSonicFoam* suffering from oscillations and *centralFoam* giving sharp shocks without oscillations. Note that *rhoSonicFoam* completely failed to deliver the correct answer in this case, probably because of a problem related to the inlet boundary condition along the top side, as evidenced by the non uniform velocity vectors in the central post shock area.

The time required by the various solvers is reported in Table 4. Again, larger time steps, hence a faster convergence, might be possible with the original solvers; however this has not been investigated since they already showed unacceptable results.

Solver	CPU time (s)	Time ratio
centralFoam	97.7	1.0
sonicFoam	182.3	1.9
rhoSonicFoam	100.5	1.0
rhopSonicFoam	420.8	4.3

Table 4. CPU time for the oblique shock case.

VI. Woodward-Colella supersonic forward step

This is also a well-known test case used among others by Woodward and Colella.[4] A step suddenly appears on the bottom wall of a channel in a Mach 3 flow. A bow shock develops which is then reflected a few times by the top and bottom wall. It is essentially an unsteady test case and the solution at Time = 4 is usually considered in the literature. Eventually (Time > 20), the flow reaches a steady state condition with a single shock across the channel height in front of the step.

The evolution of the flow at a few time instant, according to the computation done with *centralFoam*, is shown by the sequence of pictures in Fig. 9. This computation is done on the same mesh and applying the same boundary conditions used in the tutorial's forwardStep case (available for both *sonicFoam* and *rhoSonicFoam*). However, the tutorial's case starts from zero internal velocity, which results in a different transient from what specified in the literature (although it obviously converge to the same steady state).

It is possible to run the case from the proper initial condition of Mach 3 flow using *sonicFoam* and *rhopSonicFoam* as well, but with *rhoSonicFoam* the solution blows up after a few time steps. The results from the first two solvers at Time = 4 are shown in Fig. 10 along with the results computed by *centralFoam* on the standard grid and on a fine grid with twice the cells in both directions.

Once more the result from *sonicFoam* is completely wrong: the transient is not right and actually even the steady state solution to which the flow converges is wrong; in fact picture 10-a (at Time = 4) is very similar to the one (at Time = 10) presented in section 3.3 of OpenFoam *Programmer's Guide*, where the forwardStep tutorial is illustrated (the influence of viscosity is almost negligible); this is because according to *sonicFoam* the flow will reach its (wrong) steady state shortly after Time = 4. Considering the bad result it would probably be better to remove this case from the Guide, as it demonstrates an incorrect computation!

On the contrary, *rhopSonicFoam* gives as usual the right answer but corrupted by oscillations, while *centralFoam* provides an accurate solution comparable to those given in the literature, although the one on the fine grid is a little noisy. For both solvers the solution along the step wall is actually distorted by a significant numerical boundary layer, so that the reflecting oblique shock is joined to the lower wall by a short normal shock. This behavior is induced by the poor boundary condition applied to the pressure at the wall; in fact the current treatment of slip and non-slip wall in OpenFoam is to apply a zero-order extrapolation of the pressure (zero gradient): this is good enough for the highly clustered grids used for high-Re viscous flow but is rather poor for grids typical of inviscid flow; it is expected that a first-order extrapolation of the pressure, if available, would improve the results.

Finally, Table 5 shows the comparison of the time required by the various solver to run the simulation up to Time = 4.

Solver	CPU time (s)	Time ratio
centralFoam	179	1.0
sonicFoam	459	2.6
rhoSonicFoam	-	-
rhopSonicFoam	596	3.3

Table 5. CPU time for the forward step case.

VII. Conclusion

A new unsteady, non-viscous, compressible, density-based flow solver implementing one version of the so-called central schemes have been written for OpenFoam 1.3. A few test cases, 1D and 2D, unsteady and steady, show that the solver, named *centralFoam*, is typically more accurate (sharper shocks, less oscillations) and faster than any of the three basic compressible flow solvers distributed with OpenFoam 1.3; these, at least in the presented cases, all suffer from either excessive dissipation or significant oscillations in presence of shocks. Actually, no one of the original solvers performed acceptably on the full set of test cases, whereas *centralFoam* consistently delivered faster, correct and high-resolution results.

It is also shown that *sonicFoam* is not an accurate solver in presence of shocks, at least when the set of solution parameters specified in the tutorial's shockTube or forwardStep cases are applied, and that it can lead to completely wrong answers.

References

[1] http://www.cscamm.umd.edu/centpack/publications/

[2] Kurganov, A., Noelle, S. and Petrova, G., "Semidiscrete Central-Upwind Schemes for Hyperbolic Conservation Laws and Hamilton-Jacobi Equations", *SIAM J. Sci. Comput.*, Vol. 23, No. 3, 2001, pp.707-740.

[3] Liska, R. and Wendroff, B., "Comparison of Several Difference Schemes on 1D and 2D Test Problems for the Euler Equations", *SIAM J. Sci. Comput.*, Vol. 25, No. 3, 2002, pp.995-1017.

[4] Woodward, P. and Colella, P., "The Numerical Simulation of Two-Dimensional Fluid Flow with Strong Shocks", *J. Comput. Phys.*, Vol. 54, 1984, pp.115-173.

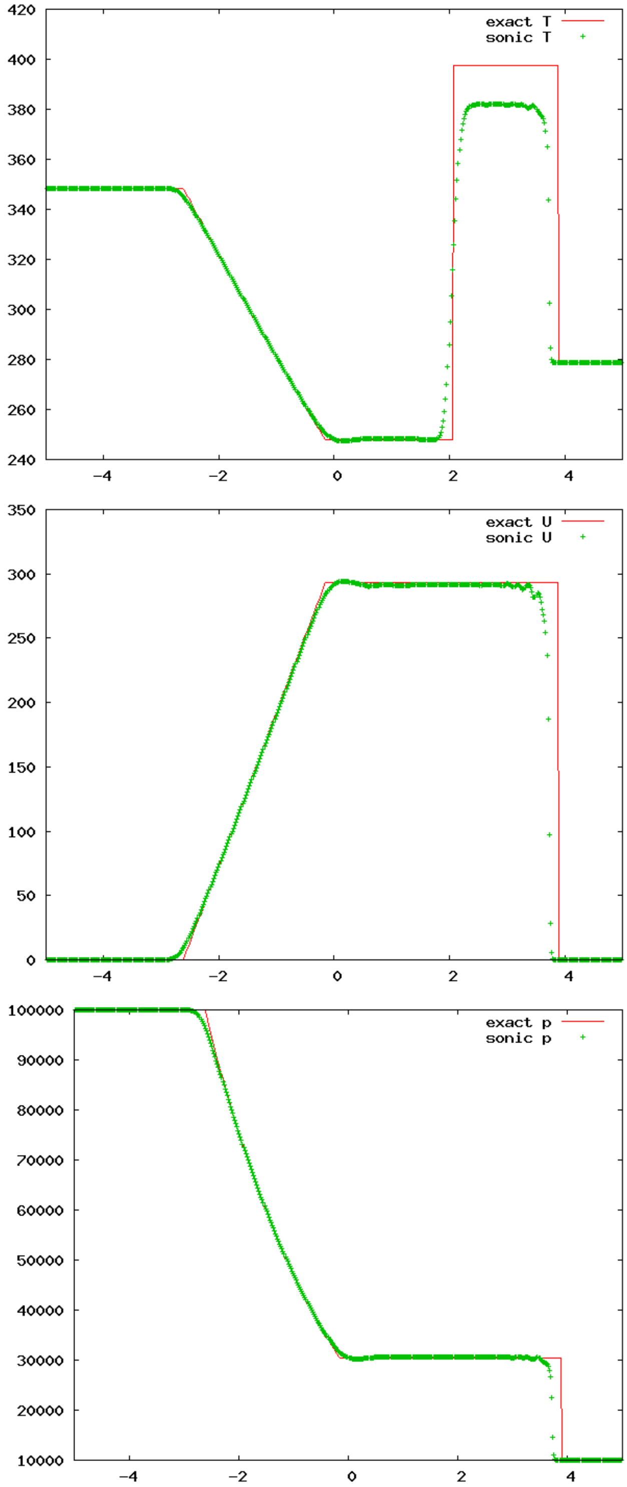


Figure 1. Shock tube, *sonicFoam* results.

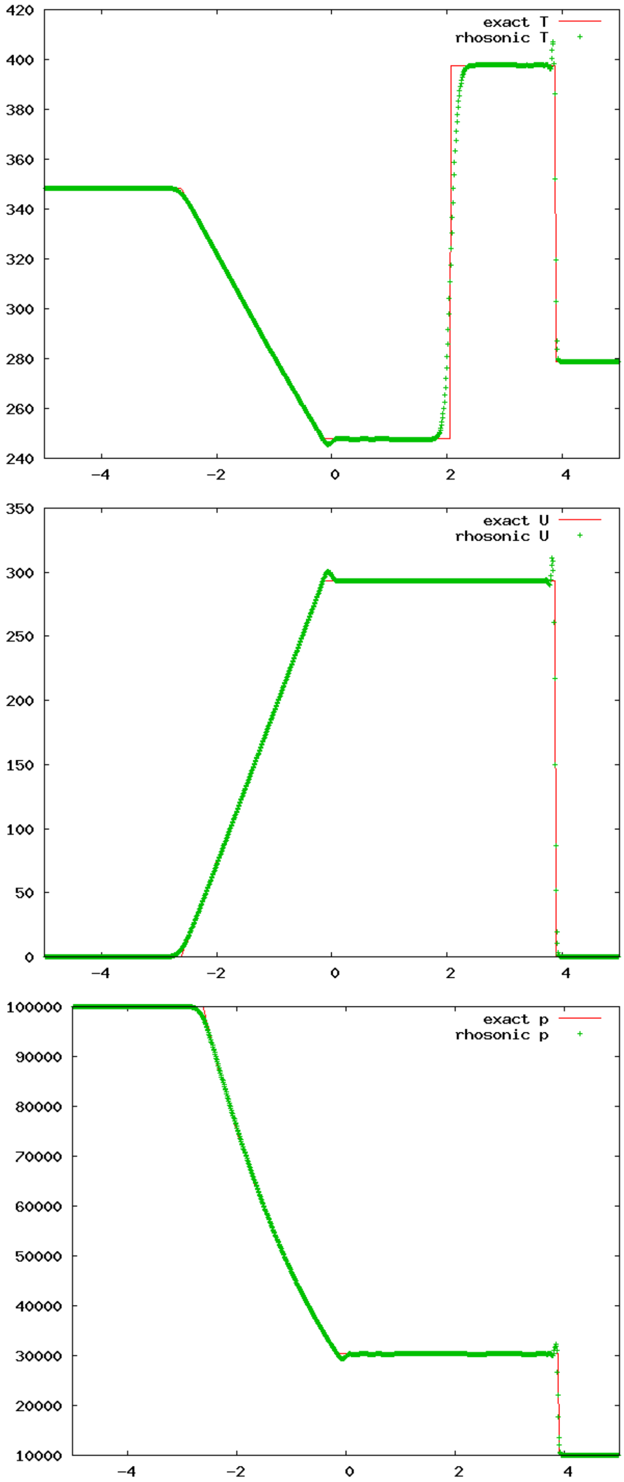


Figure 2. Shock tube, *rhoSonicFoam* results.

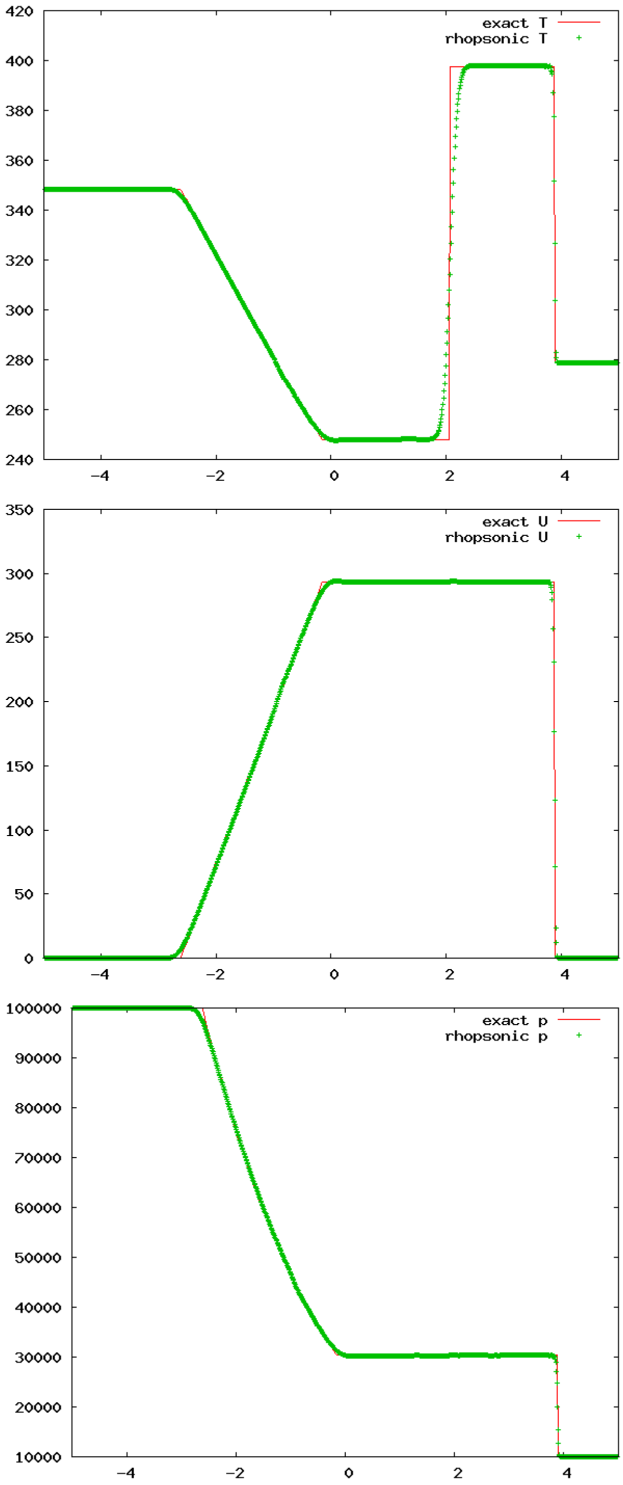


Figure 3. Shock tube, *rhopSonicFoam* results.

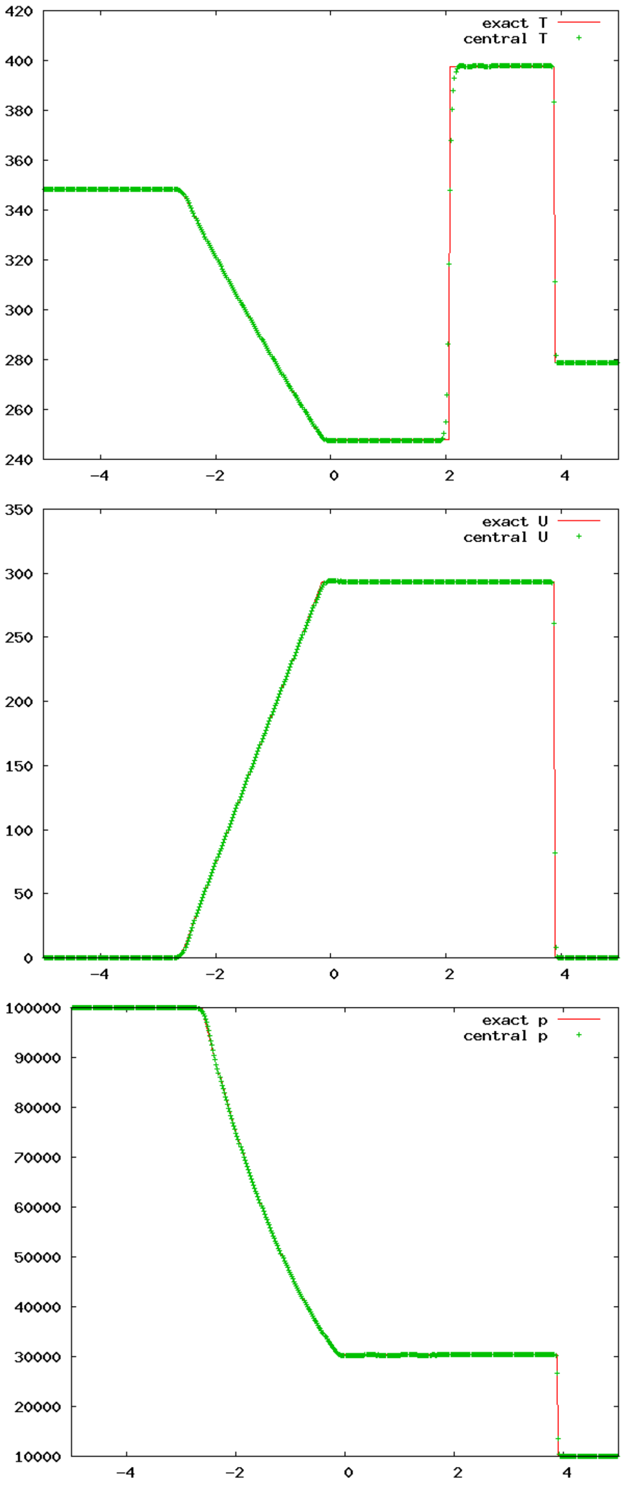
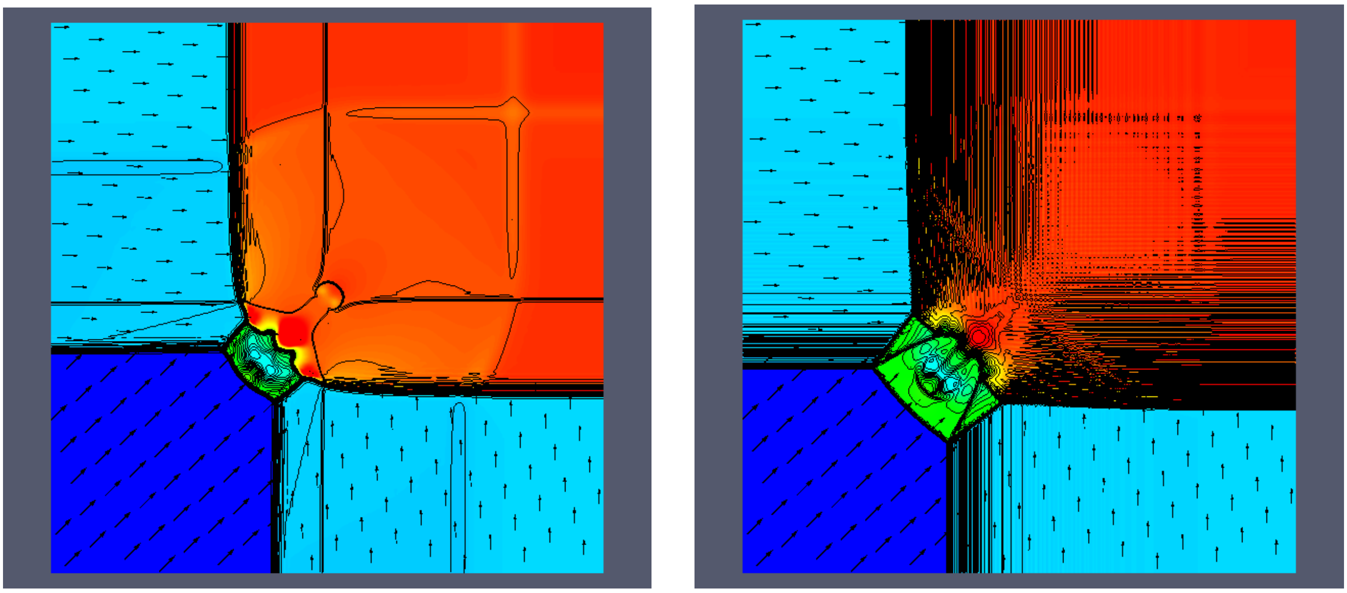
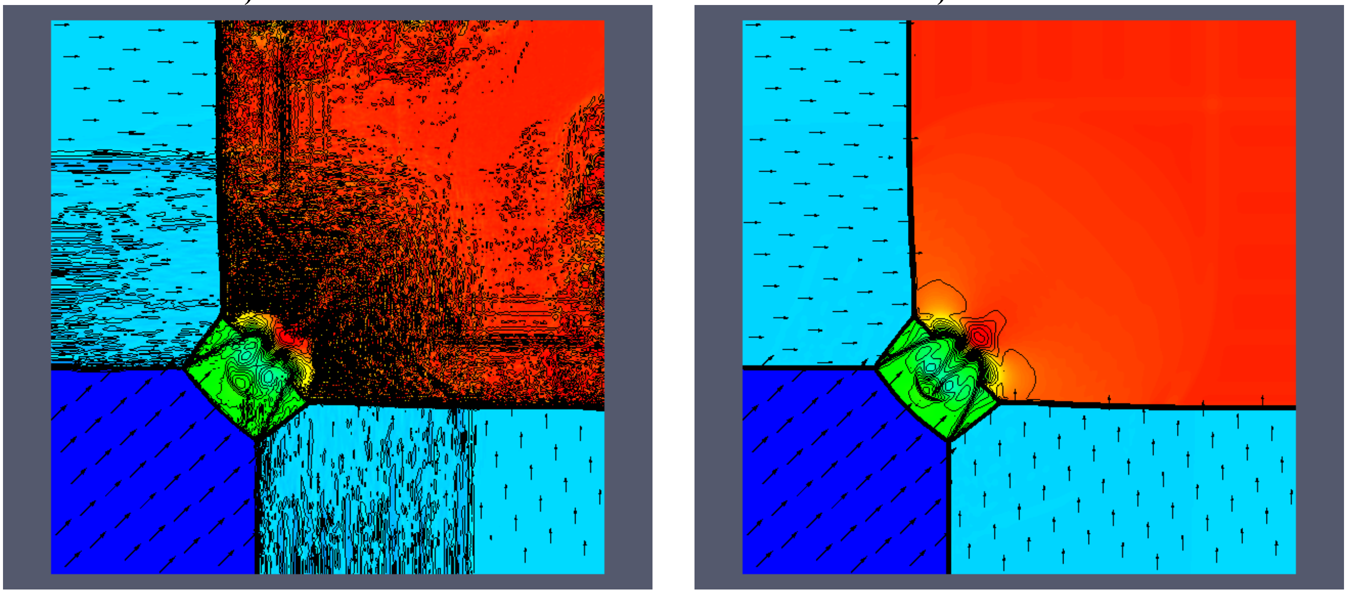


Figure 4. Shock tube, *centralFoam* results.

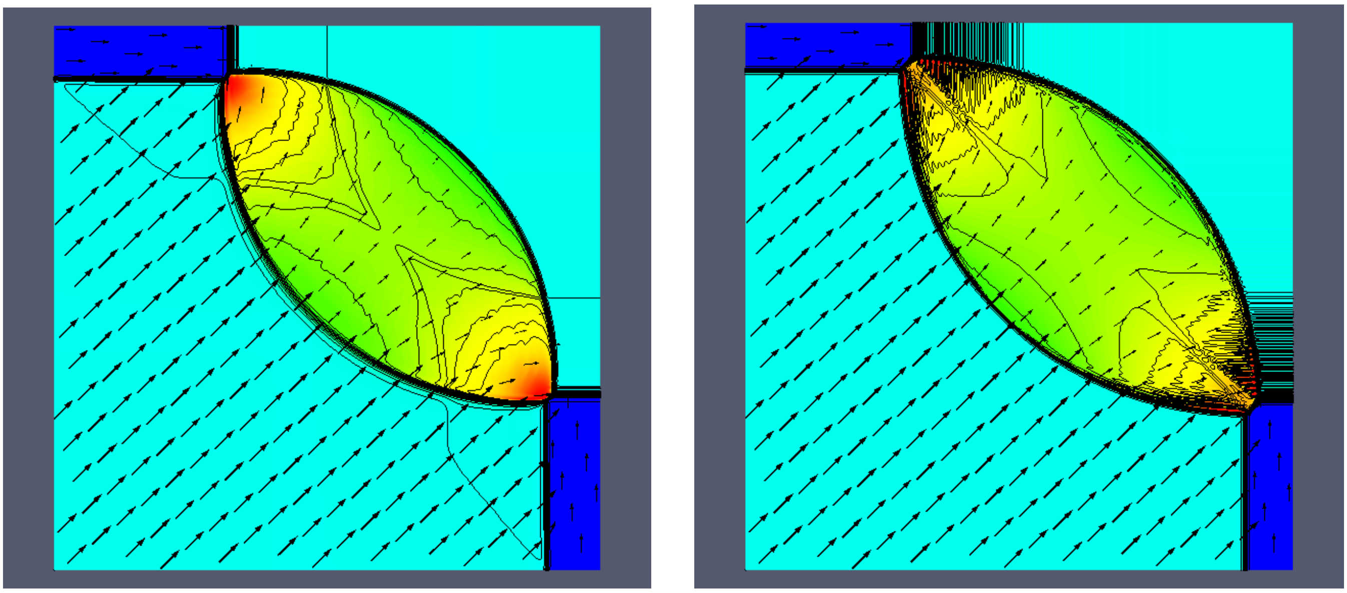


a) ***sonicFoam*** **b)** ***rhoSonicFoam***

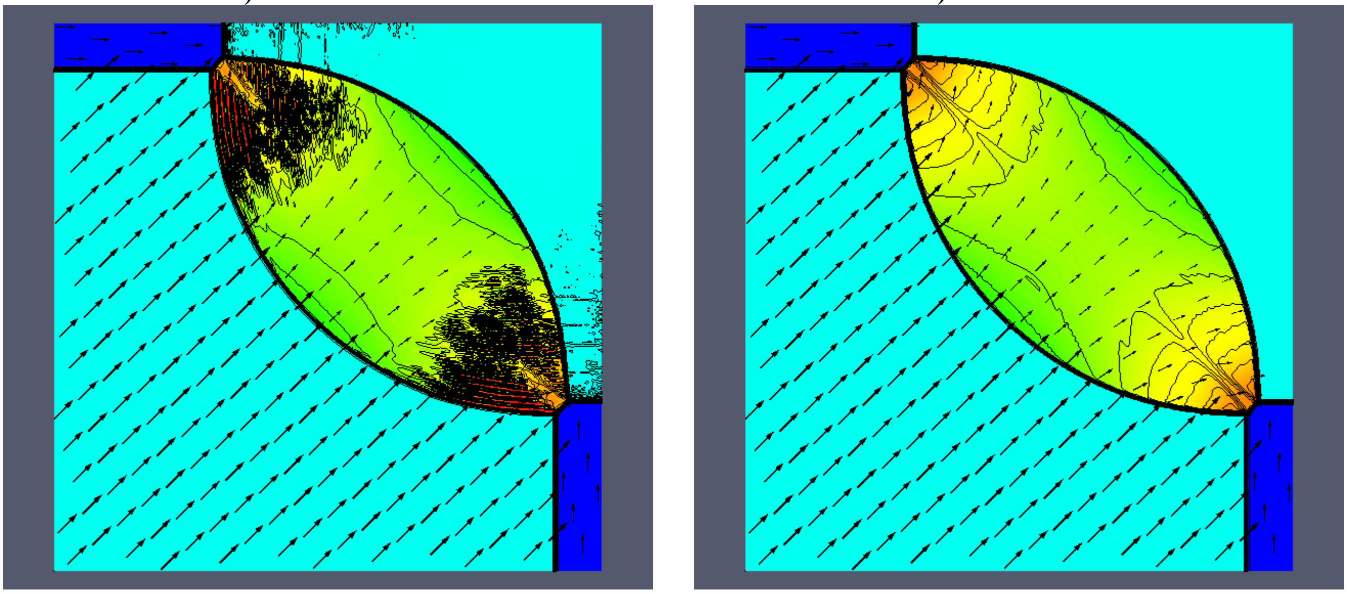


c) ***rhopSonicFoam*** **d)** ***centralFoam***

Figure 5. 2D Riemann problem, case 3. Pressure is displayed by color (from 0.0 to 1.6), density by contours and velocity by arrows.

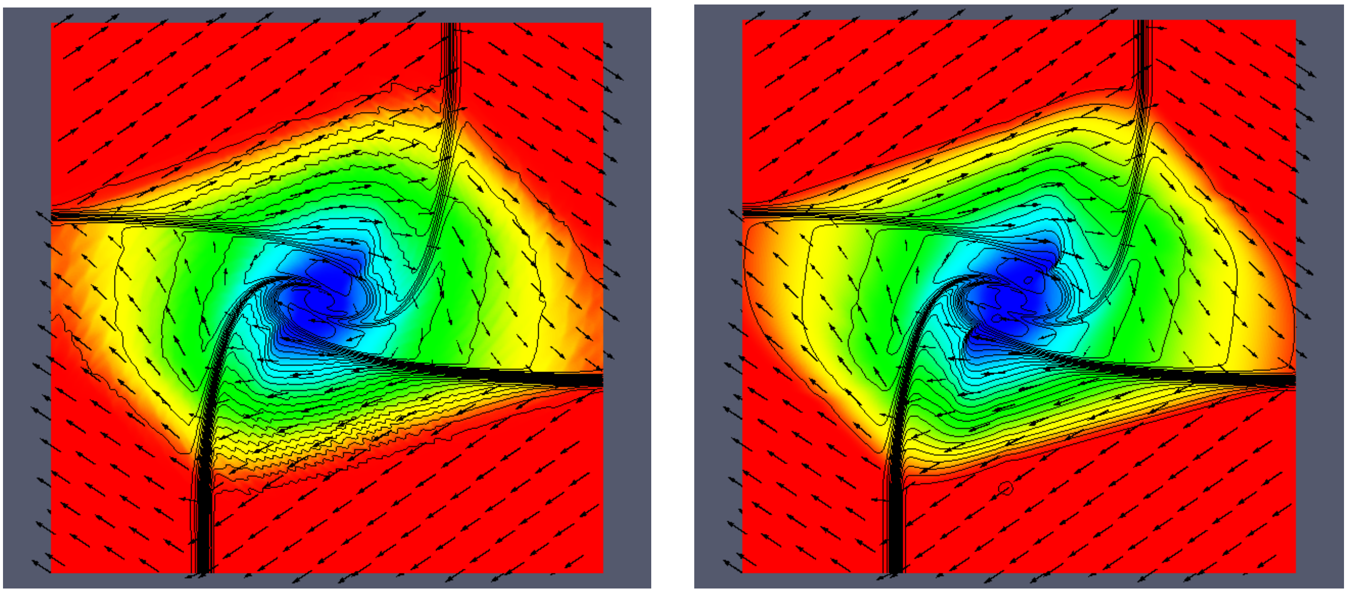


a) *sonicFoam* b) *rhoSonicFoam*

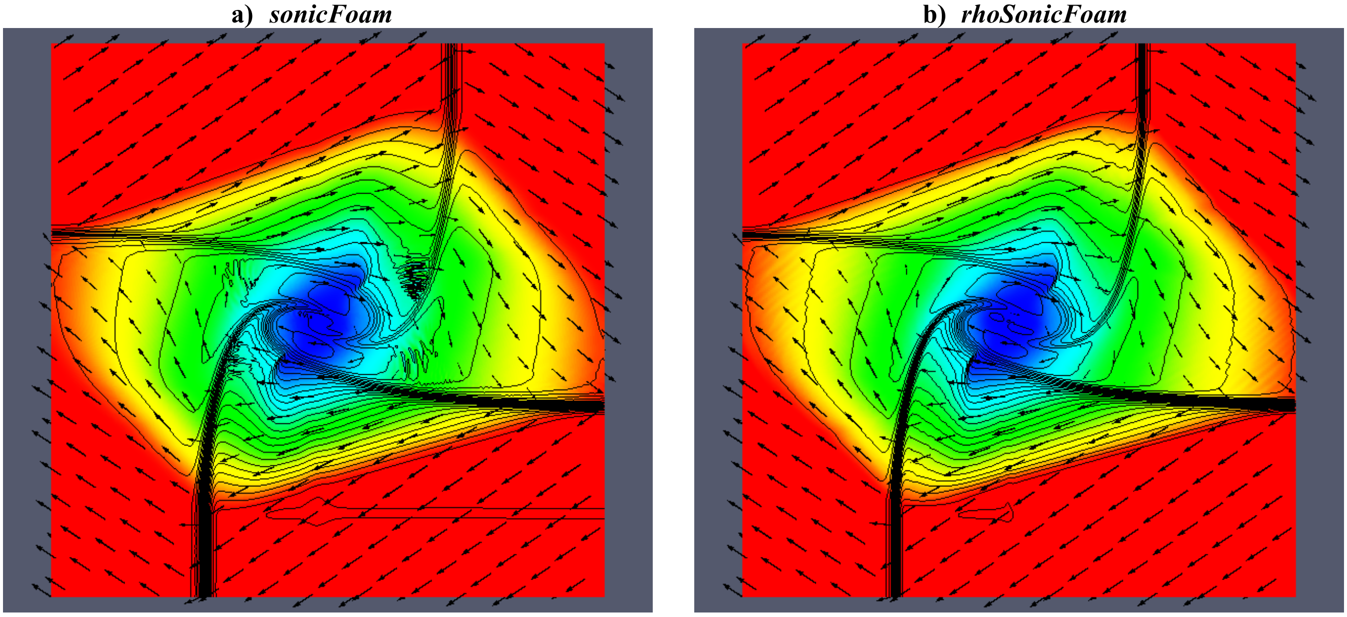


c) *rhopSonicFoam* d) *centralFoam*

Figure 6. 2D Riemann problem, case 4. Pressure is displayed by color (from 0.4 to 2.8), density by contours and velocity by arrows.

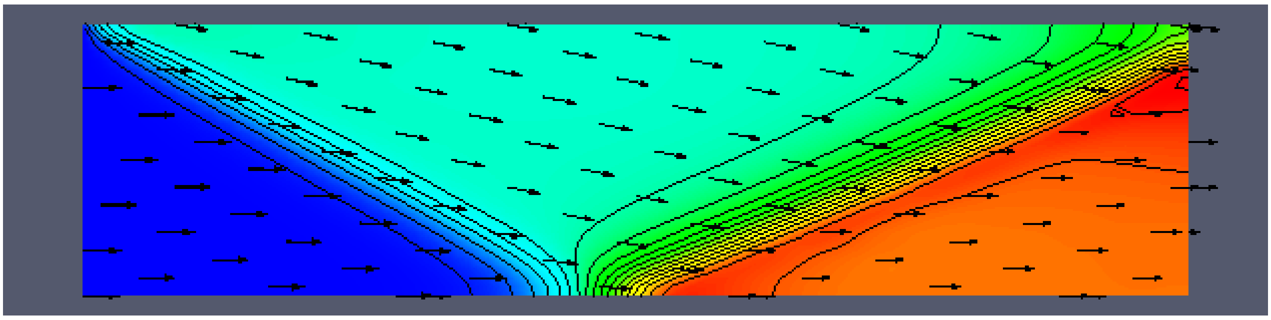


a) *sonicFoam* b) *rhoSonicFoam*

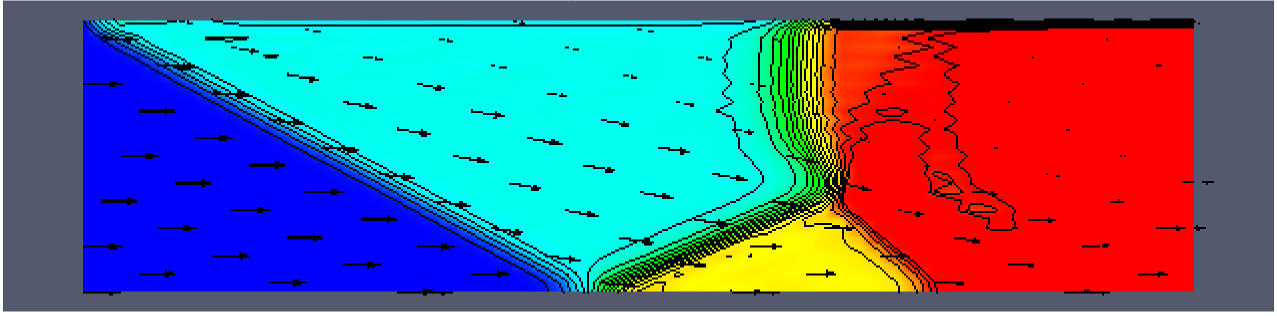


c) *rhopSonicFoam* d) *centralFoam*

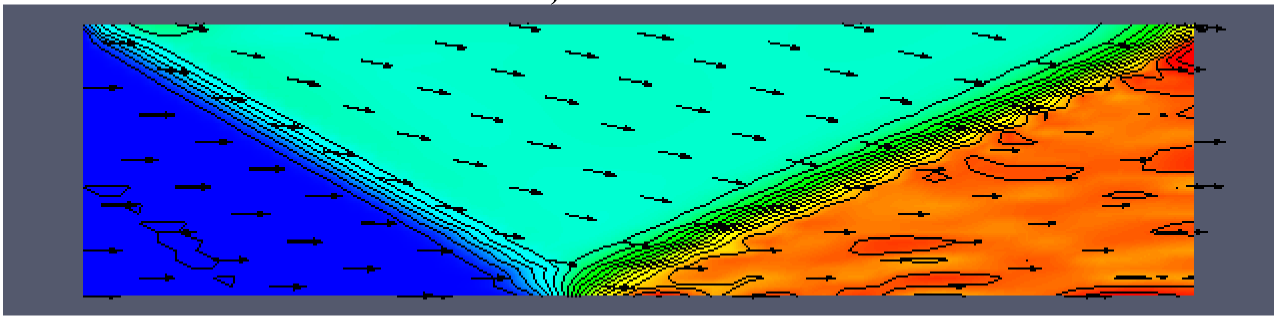
Figure 7. 2D Riemann problem, case 6. Pressure is displayed by color (from 0.15 to 1.0), density by contours and velocity by arrows.



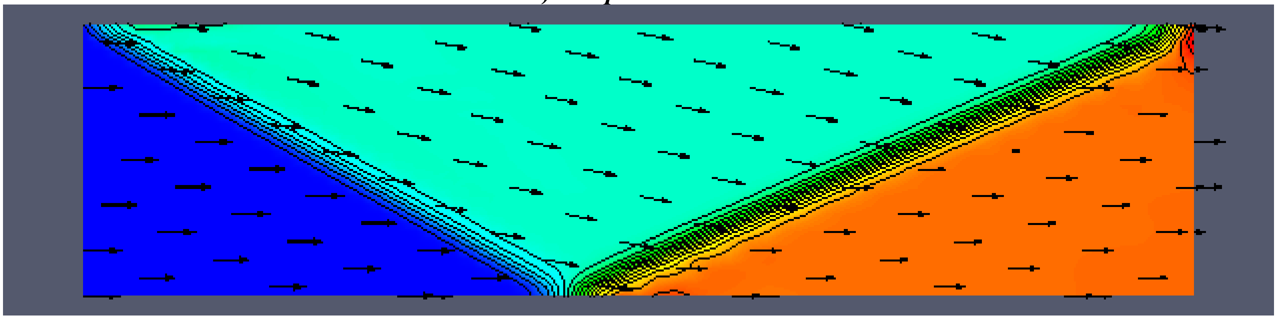
a) ***sonicFoam***



b) ***rhoSonicFoam***

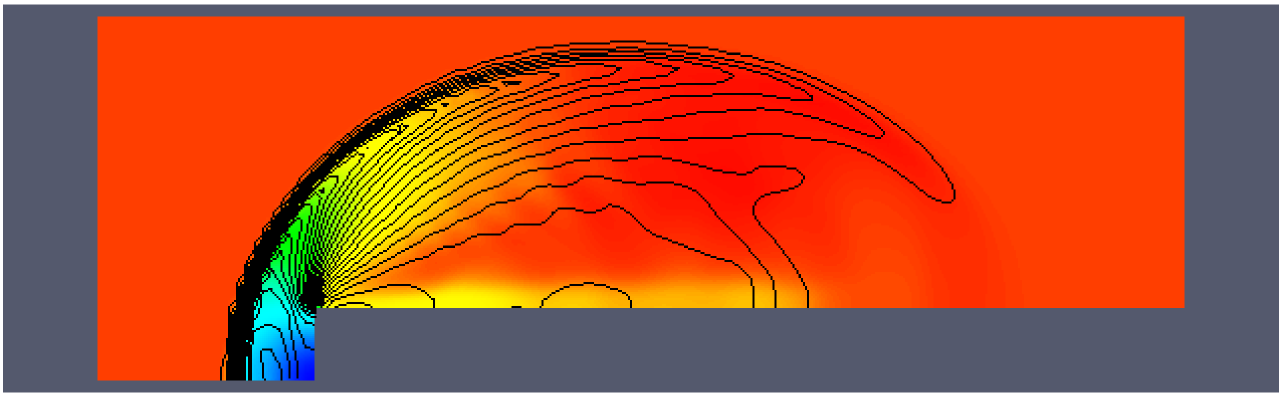


c) ***rhopSonicFoam***

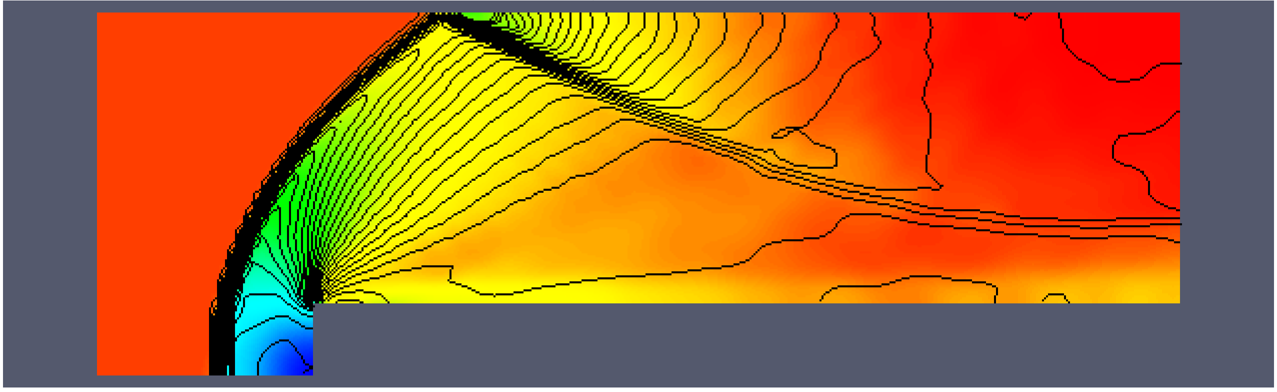


d) ***centralFoam***

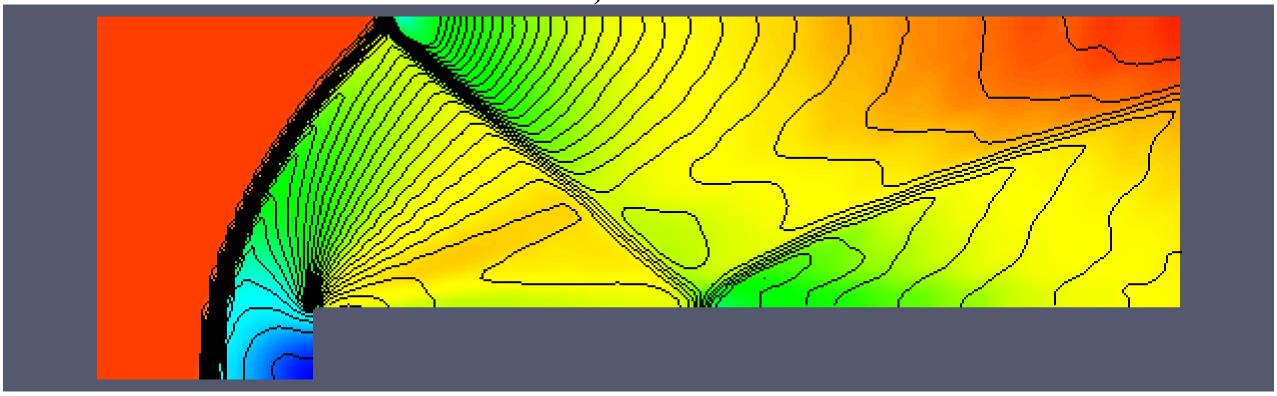
Figure 8. Steady, oblique shock case. Pressure is displayed by color and contours, velocity by vectors.



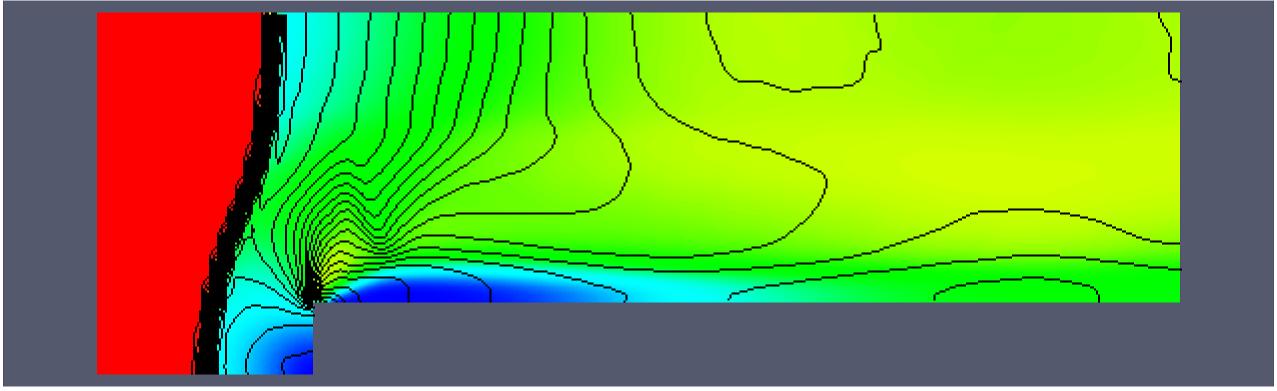
a) T = 0.5



b) T = 1.0

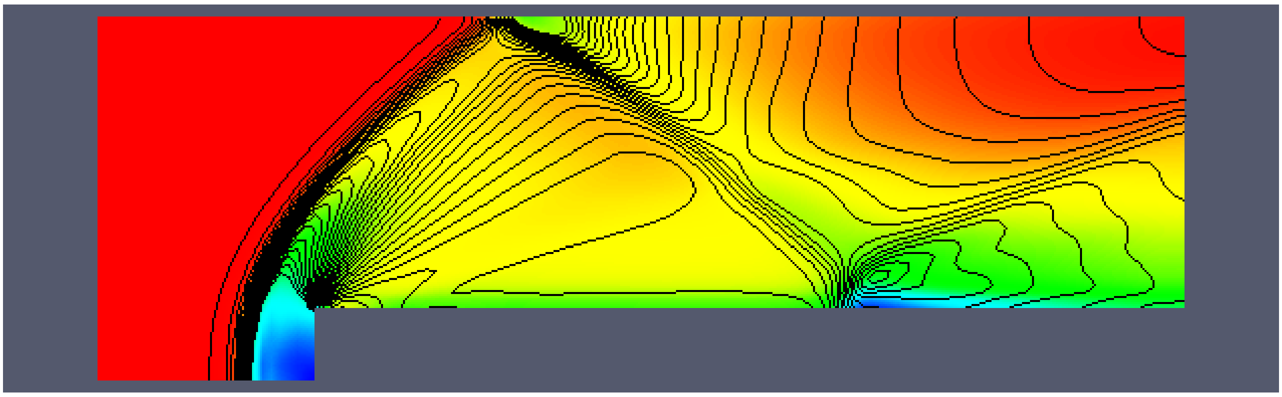


c) T = 2.0

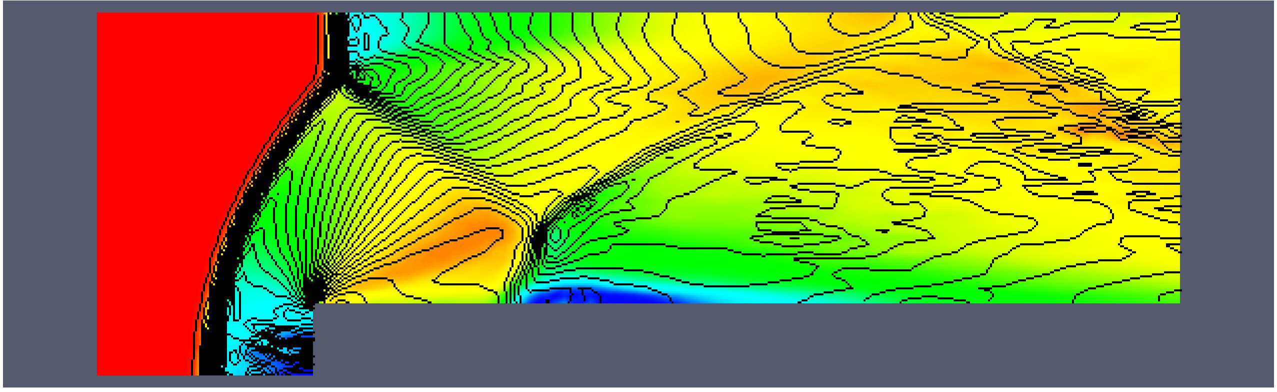


d) T = 25.0

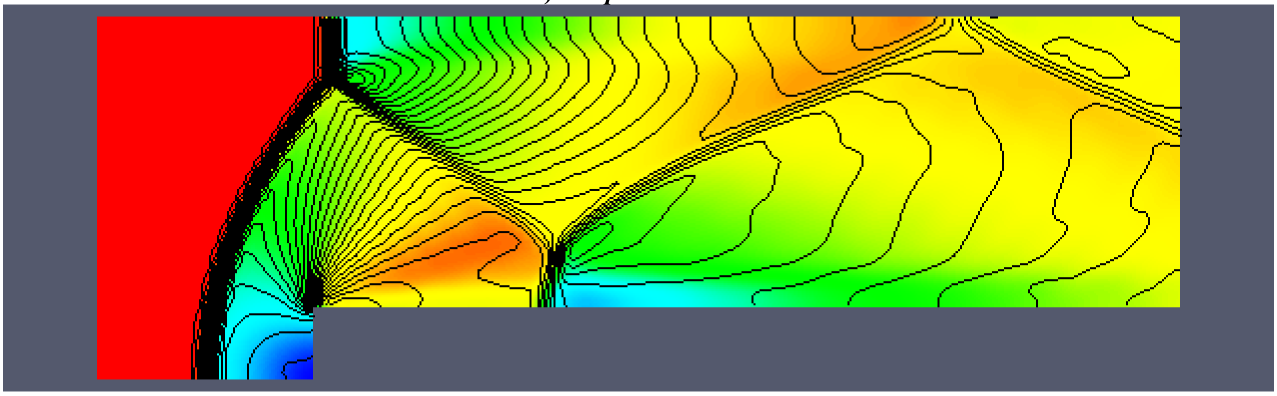
Figure 9. Forward step case. Computation with *centralFoam* on standard grid. Velocity magnitude is displayed by color, density by contours.



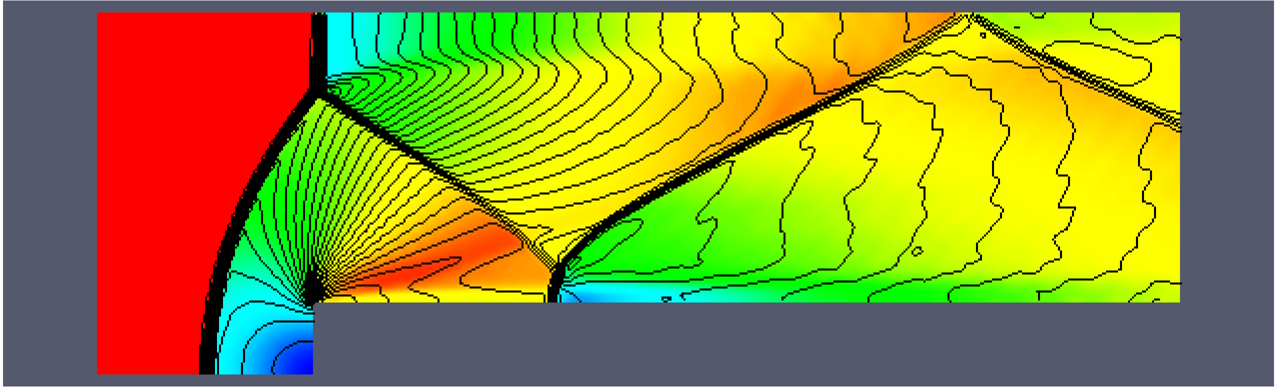
a) *sonicFoam*



b) *rhopSonicFoam*



c) *centralFoam*, standard mesh



d) *centralFoam*, fine mesh

Figure 9. Forward step case. T = 4. Velocity magnitude is displayed by color, density by contours.