Brian Kulwik
Aero 623 – CFD II
Final Project – Silencing a Railway Tunnel with 2D Euler CFD
22 April 2015

Background and Motivation

High-speed railways are becoming ever more popular as a cheap and safe mass transit system in many countries. However, there is an issue with acoustic noise generated by these high-speed trains when traveling through tunnels. Because of their high speed, they essentially act as pistons, compressing the air in front of them and generating a shock wave. This shock wave then expands out of the end of the tunnel, potentially damaging structures and causing annoyance of the local population. Currently, the trains are slowed down in these tunnels to compensate for this effect, losing the advantage of their high-speed capabilities.

A potential solution to this problem is the design of "diffusion tunnels," side tunnels that are dug off of the main tunnel which are used to diffuse the effect of the shock wave. There are infinitely many potential designs for this, but it has shown some promise in reducing the effect of the shock wave.

The purpose of this report is to describe the methods used to compute solutions to this problem, including the development of a full Euler equation (not isothermal) 2D CFD solver to model the given problem.

2D Euler Equation Solver – Overview

The 2D Full Euler Equations solve for an inviscid, compressible solution to a given problem, accounting for changing energy (as opposed to the isothermal, which assume energy is constant). The Here, they are used in differential form, although they could also be used in integral form in other situations. They come in the order of a two-dimensional conservation equation as follows:

$$\frac{\partial}{\partial t}\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ E\end{bmatrix}+\frac{\partial}{\partial x}\begin{bmatrix}\rho u\\ \rho u^2+P\\ \rho uv\\ u(E+P)\end{bmatrix}+\frac{\partial}{\partial y}\begin{bmatrix}\rho v\\ \rho uv\\ \rho v^2+P\\ v(E+P)\end{bmatrix}=\begin{bmatrix}0\\0\\0\\0\end{bmatrix}$$

Where P (pressure) is defined as

$$P=(\gamma-1)\left(E+\frac{\rho(u^2+v^2)}{2}\right)$$

A code was written in C++ to solve the 2D Euler Equations of Gas Dynamics, using a second-order MUSCL scheme. First, a grid was defined on which to solve the problem. This was completed using MATLAB, which was then output to a grid file (file extension .bkcfd). A parameter file with which the user can input the desired solution parameters of number of timesteps and the CFL number is used as well.

First, the grid and parameter file are read into the C++ main file. The code is set up to handle a non-constant mesh spacing, so the grid file is parsed to find the minimum grid spacing, and from this a timestep is calculated using the CFL number and the maximum of all wavespeeds in all cells. This global timestep is allowed to change as the solution wavespeeds change, allowing for the fastest possible solution convergence rate.

After reading in the grid, the code then reconstructs each state, using a given limiter, to be piecewise linear across the solution domain. From this, the edge fluxes are calculated and the solution is updated to the halfway state, $u^{t+\frac{1}{2}}$. Then the code takes the halfway state and computes the exact solution to the Riemann problem across each face of each cell. The solution on this interface is then used to calculate the flux on each face of a given cell, and the solution is updated with that flux to the new state, u^{t+1}. This is then repeated for the user-specified number of timesteps and the solution written to a text file. This solution is then read into MATLAB and plotted. It is worth noting that the solutions states are physical values, not normalized values (i.e. pressure in Pa, density in kg/m^3, etc.)

2D Euler Equation Solver – Grid Definition

The grid for the solution was defined using MATLAB, in a function called MakeGridMatlab.m. The user specifies the physical location (in meters) and dimensions of the tunnel, the side tunnel, and the outlet domain, as well as the number of cells in the x and y directions. The user also specifies a global initial condition as well as the inlet initial condition (in this case, taken to be the state downstream of a shock wave propagating downstream at mach 1.2).

To create the actual grid, the code creates a rectangular grid of user-specified dimensions. Then, taking into account the user-specified locations of the tunnel and side tunnel, extracts the solution grid from the global grid. The initial conditions are then computed based on the cell location. This is one restriction of the grid generator, is that it requires that the x-location start at 0. The initial conditions are set such that all cells at x = 0 have the shock downstream condition and all cells that are not at x = 0 have the global initial condition.

The boundary conditions are also set here. This code makes use of “ghost cells” for the enforcement of boundary conditions. These are also initialized here, along with the interior cells. Cells are all assigned an edge state, either equal to zero for an interior cell, one for a wall (reflection boundary condition), or two for a farfield boundary condition (zero gradient) such that the solution is “absorbed” by the wall.

MATLAB then outputs the grid to a file, with name specified by the user and file extension .bkcfd, the code’s standard grid input file extension. The grid is read into the code after being output to the .bkcfd file. It is stored in the C++ code as a vector of cells, where each cell has containers cell number, corner locations_x, corner locations_y, adjacent cell numbers, edge type, and TDstate. TDstate is a two-dimensional state struct, with the quantities density, x-momentum, y-momentum and energy.

A visual depiction of a 160x160 – cell mesh is shown below in Figure 1. Note the mesh size of 160x160 represents the initial rectangular mesh size, before the actual solution domain was cut out of it.

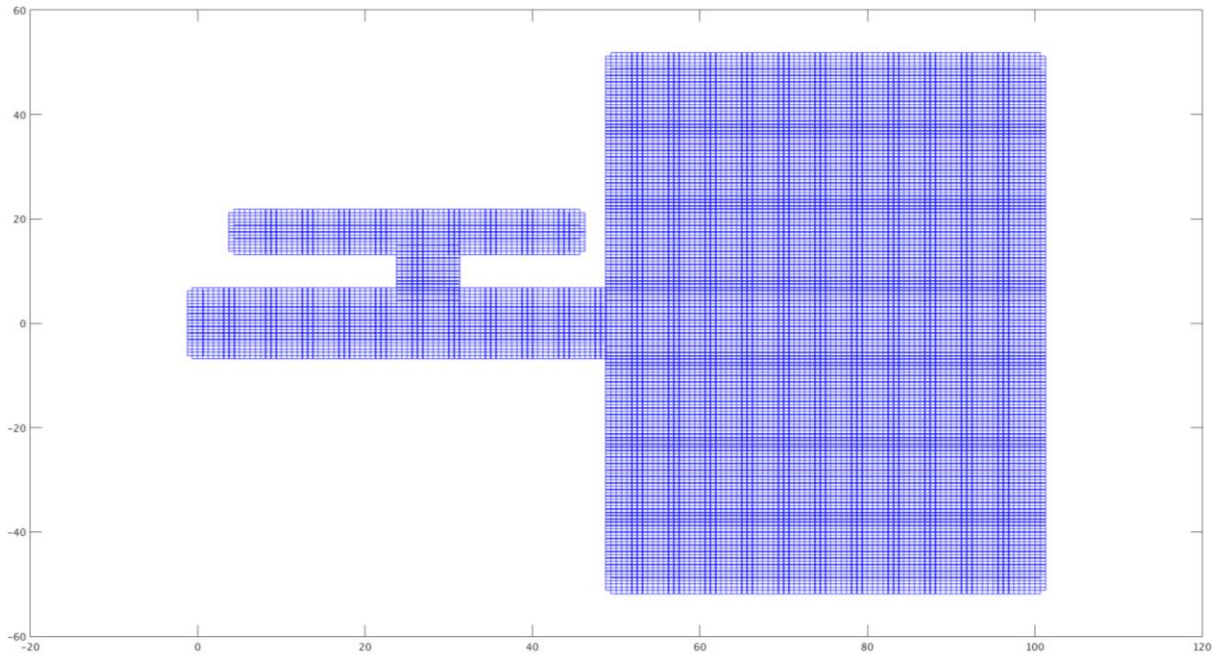

Figure 1 – 160 x 160 Mesh Example

2D Euler Equation Solver – Updating to $\boldsymbol{u}^{t+\frac{1}{2}}$

In order to achieve second-order accuracy, each of the cells were first updated to a "halfway" state, computed first by finding the solution gradient between the left-right and bottom-top cells of this cell. Each of these solution gradients is calculated independently (i.e. so we have a set of left-right solution gradients, then a set of bottom-top solution gradients), and the left-right (F) and bottom-top (G) fluxes are calculated independently, then used together to update the state.

The solution gradients were computed using a simple one-sided difference, i.e.

$$s_{i^+,j} = (u_{i+1,j}^n - u_{i,j}^n)/\Delta x$$

$$s_{i^-,j} = (u_{i,j}^n - u_{i-1,j}^n)/\Delta x$$

After computing the solution slopes, a user-specified limiter was used to limit the slopes for the updating step. Users currently can choose between first-order upwind (no limiter), Lax-Wendroff or harmonic.

After computing gradients and limiters, the state on the edge cells was then reconstructed and updated in one step using

$$u_{edge} = u_{center} \pm \frac{1}{2} B(1 - CFL)$$

Where

$$B = (limiter) * slope_{\pm}$$

After each u_{edge} is calculated, the flux on each edge is calculated using the equations for flux.

$$F = \begin{bmatrix} \rho u \\ \rho u^2 + P \\ \rho uv \\ u(E+P) \end{bmatrix} \quad G = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + P \\ v(E+P) \end{bmatrix}$$

Where each state in the flux calculation is the state on the edge.

The updated state is the calculated as follows:

$$u_{i,j}^{t+\frac{1}{2}} = u_{i,j}^{t} - \frac{\Delta t}{2\Delta x}\left(F_{i+\frac{1}{2},j} - F_{i-\frac{1}{2},j}\right) - \frac{\Delta t}{2\Delta x}\left(G_{i,j+\frac{1}{2}} - G_{i,j-\frac{1}{2}}\right)$$

For each state u and flux F in the state.

For all of the solutions presented here, the First Order Upwind limiter was used. Although providing less accuracy than other flux-limited schemes, the accuracy is high enough that we are able to see differences and improvements in tunnel designs relative to each other.

2D Euler Equation Solver – Updating from $u^{t+\frac{1}{2}}$ to u^{t+1} with Exact Riemann Solver

The procedure to calculate the update from $u^{t+\frac{1}{2}}$ to u^{t+1} involves solving the Riemann problem for each cell interface on each cell. The Riemann problem is the thermodynamic solution to the shock tube problem, if you will. For a discontinuous jump between states, the Riemann problem provides a solution to the states that will arise between them. The Riemann solver that was written for this code uses the exact states to the full Euler equations, not linearized and not isothermal. This is consistent with the rest of the solution definition.

The solution to the Riemann problem was computed using a combination of Riemann invariants and isentropic equations. In the center state, the 'star' state, there is a contact discontinuity (the eigenvalue 'u'), across which pressure and velocity need to remain constant, but for which density can be discontinuous. This is all computed based on pressure. An initial 'guess' pressure state is supplied by the code, and the velocity in the left and right 'star' states are calculated. The pressure guess is then iterated to convergence (using Newton's method) to a user-specified convergence criteria, usually about 1E-5. This often takes between two and four iterations of Newton's method, and therefore is very efficient.

It is worth noting that the Riemann problem computes a one-dimensional state, whereas the solution is stored in two-dimensional states. This is remedied by the reading in of the velocity term that is not to be included in the 2D-1D transition and incorporating this into the computation of pressure. This ensures an accurate, 2D solution even though the Riemann problem is by design a 1D solution.

Once the Riemann solution is computed, the solution on the 1D interface is returned (i.e. on the 'x = 0' line). From this, the 2D state is re-computed with the newly updated 1D state. After all four of these cell interface states are computed for a given cell, the fluxes are computed and the states updated in the same fashion as described above, in the previous section.

2D Euler Equation Solver – Handling of Boundary Conditions

The boundary conditions for a given face are specified in the input grid file, from MATLAB, to either be "interior" – no boundary, "wall" – reflection, or "farfield" – zero gradient. For the interior cells, nothing is done. The only cells that will not be specified as "interior" are the ghost cells.

For the "wall" boundary, which includes all cells in the tunnel as well as the exit volume along the mountain, the ghost cell state is first set to the state of its' adjacent interior cell. Then, the velocity is made negative in the normal direction, such that whatever state is in the edge interior cell is reflected back from the boundary.

For the "farfield" boundary, which includes the top, right and bottom edges of the exit volume, the ghost cell state is set to equal its' adjacent interior cell's state. This way, whatever solution is in the interior cell will be allowed to pass uneventfully into the "rest of the universe" outside of the solution domain.

Code Verification

The code was verified on a simple initial condition, a 200x200 cell, 3x3 meter evenly-spaced grid with wall boundary conditions. One cell was set to a high pressure, and the solution allowed to propagate for 300 timesteps at a CFL of 0.9. The accuracy of this solution gave confidence to press forward with solving the train problem at hand. A plot of solution pressure contours can be seen in Figure 2. Note that the boundary conditions of "wall" are being enforced, as evidenced by the shock reflection off of the left and bottom cell. Also note the symmetry of the solution, as expected. The cell set with a higher pressure is located at (0.65, 0.65).

Solutions – Initial Conditions

The downstream state of the train-induced shock was set to be the downstream conditions from a shock propagating at a relative mach number of 1.2, and was set to the state on the inlet. This was calculated to be

$$\boldsymbol{u_{inlet}} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ E \end{bmatrix} = \begin{bmatrix} 1.3416 \\ 216.09 \\ 0 \\ 285263 \end{bmatrix}$$

From a global initial condition of

$$\boldsymbol{u_{inf}} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ E \end{bmatrix} = \begin{bmatrix} 1.000 \\ 0.000 \\ 0.000 \\ 200000 \end{bmatrix}$$

Using this and four separate geometries, the solution was calculated and an optimization was attempted.

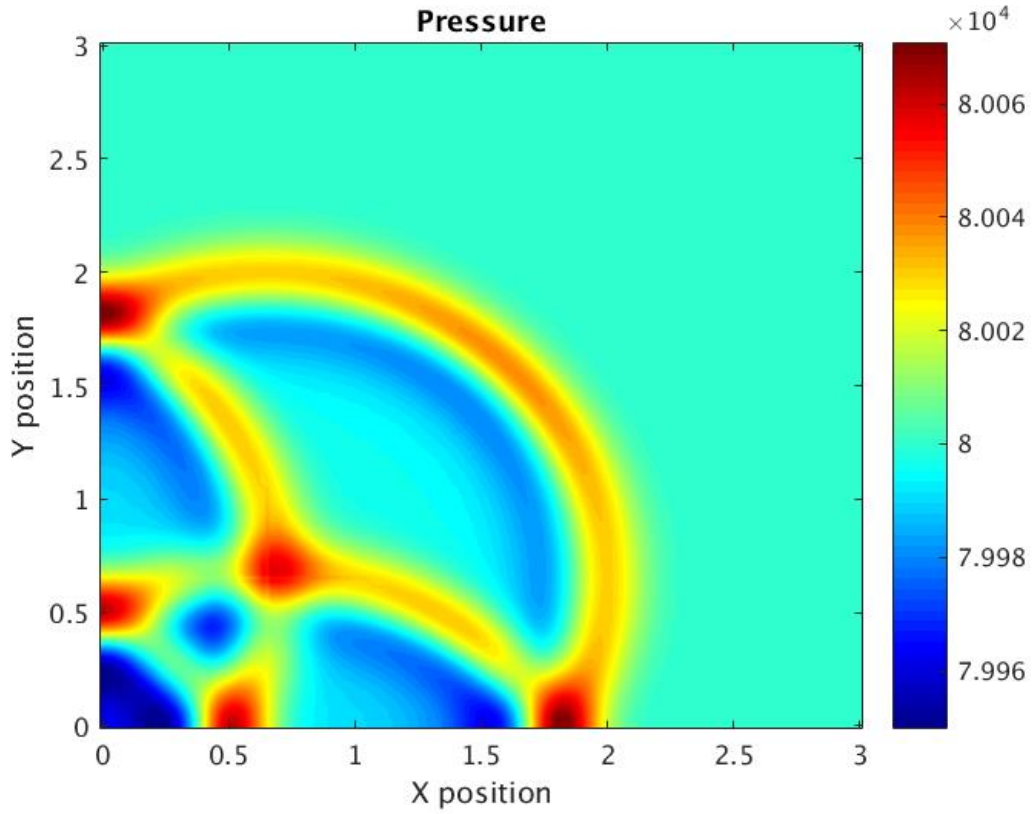

Figure 2 – Code Verification using 200x200 grid

Solution – Presentation

The solution for the four tunnel geometries is shown below in Figures 3-6. All solution are computed on a 160x160 grid at a CFL of 0.95 with 450 timesteps, to a final time of 0.517 seconds. The solution was calculated with a relatively small expansion volume for the purpose of decreasing the computation time while maintaining a high spatial resolution in the main tunnel and side tunnels. The thought is that if a higher pressure is seen in this small domain, that higher pressure will be maintained, relative to other configurations, at large distances away from the end of the tunnel.

Solution – Validation

In order to validate this solution, the same case was run on a much finer grid, in this case, a 240x240 grid for the symmetric side tunnel case. All solution parameters were kept constant, with the number of timesteps increasing to 675 to account for the change in grid spacing. The comparison between the 160x160 and 240x240 grid revealed that there was an increased resolution for the shock, however the change in pressure resolution was insignificant, validating the comparisons made on the 160x160 grid.

Conclusion

Throughout the course of this project, a C++ code was written to compute the solution to the 2D Full Euler Equations of Gas Dynamics using a MUSCL scheme. This code was validated on simple cases, then applied to the full case of a train's shock expanding out of an enclosed tunnel. By comparing with multiple cases, it appears that there is not a significant effect of adding the side tunnel, as the expanding shock's pressure was actually increased in two of the three configurations tested. However, one case, the symmetric side tunnel case, showed some promise, decreasing the shock strength by ~7%. Based on this analysis, it is recommended to pursue the symmetric side tunnel case in future calculations.

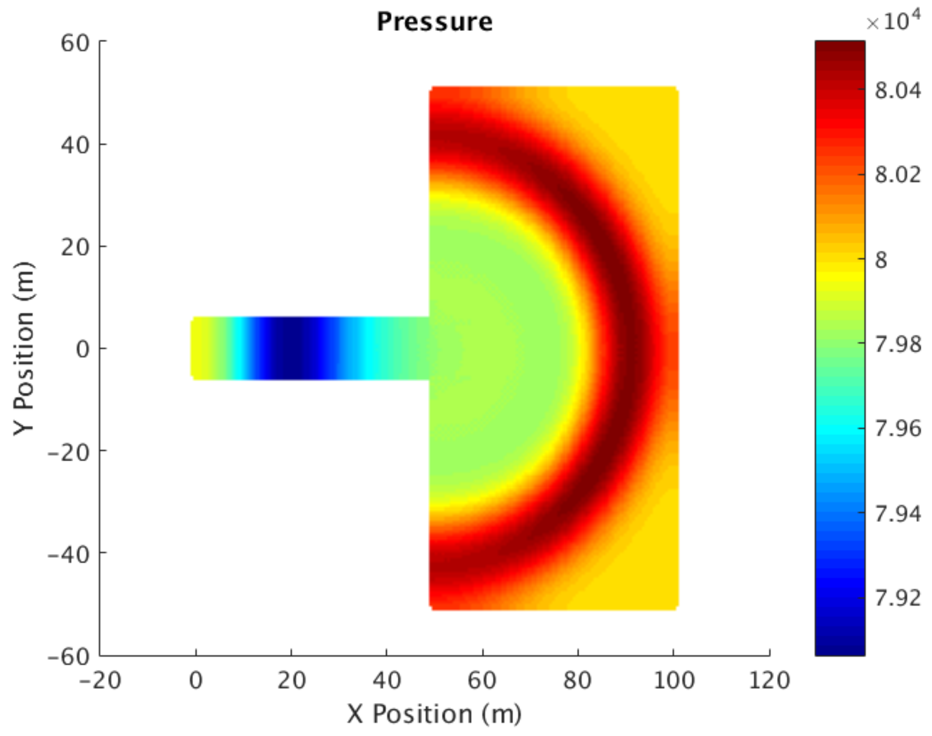

Figure 3 – Solution for no side tunnels. Max pressure in expanding shock: 80492 Pa

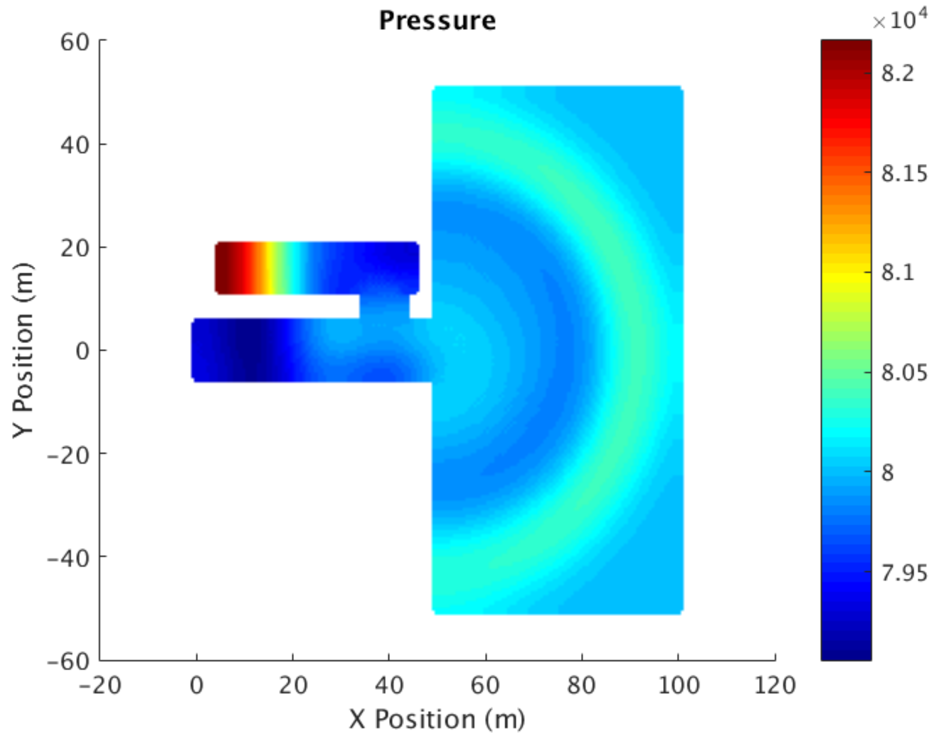

Figure 4 – Solution with left-facing side tunnel. Max pressure in expanding shock: 80503 Pa

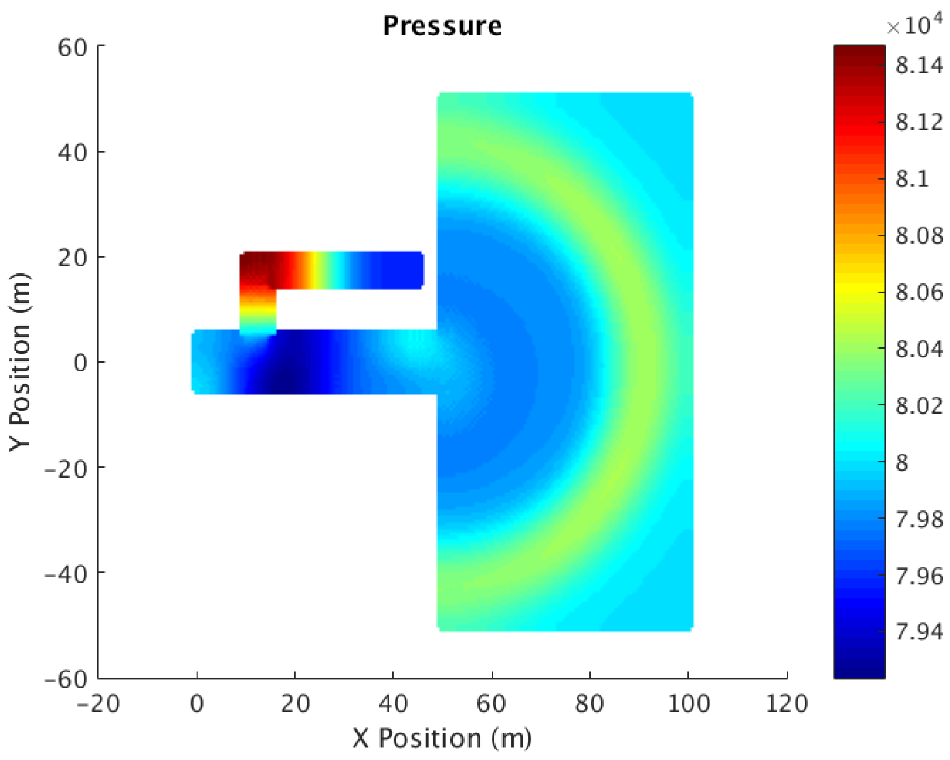

Figure 5 – Solution with right-facing side tunnel. Max pressure in expanding shock: 80520 Pa

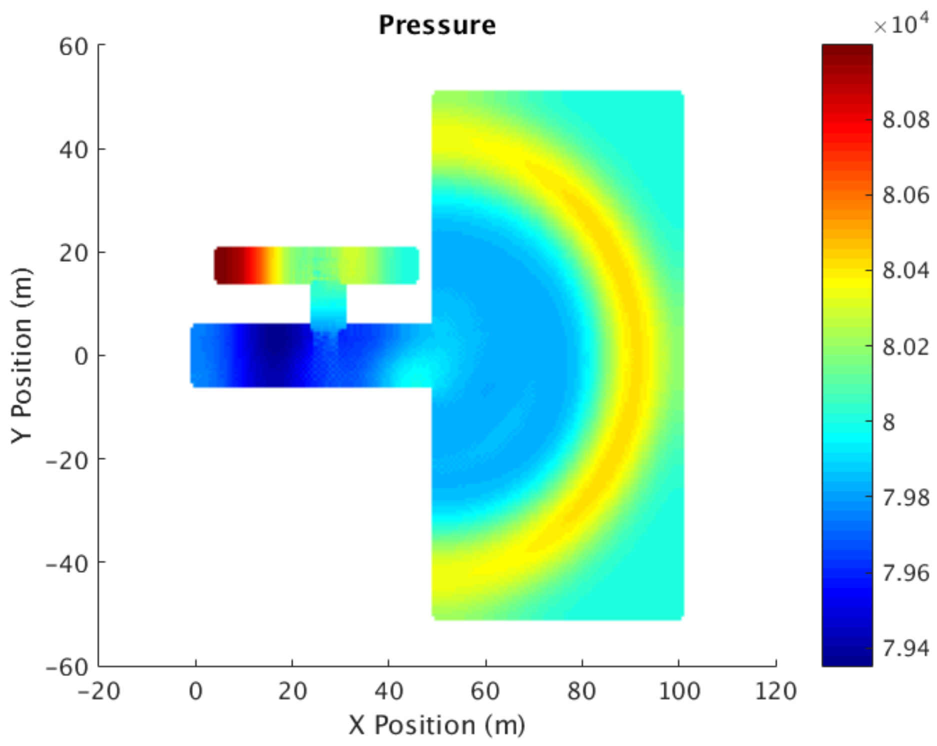

Figure 6 – Solution with centered side tunnel. Max pressure in expanding shock: 80458 Pa

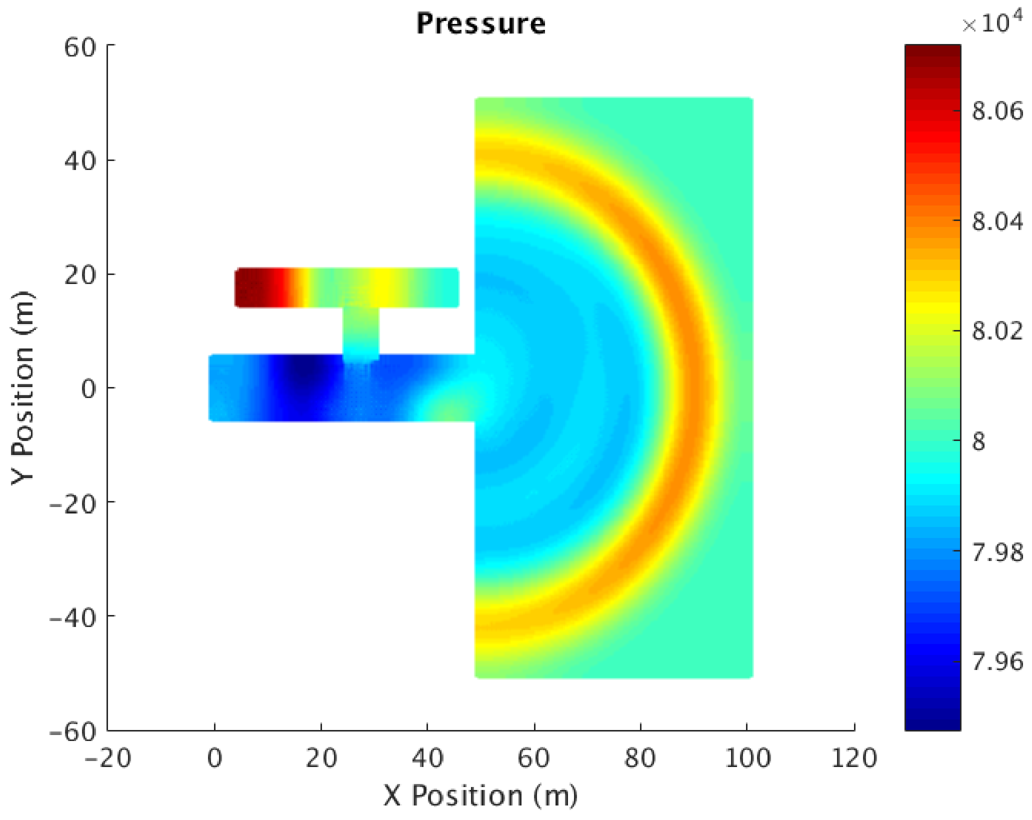

Figure 7 – Solution to centered side tunnel case on a 240x240 grid shows increased spatial resolution but insignificant change in maximum pressure. Max pressure in expanding shock: 80417 Pa