ONE-DIMENSIONAL SHOCK TUBE USING FLUX-VECTOR SPLITTING AND FLUX-DIFFERENCE SPLITTING METHODS

MAE540 Project 7

Adrian Zebrowski
May 8th, 2020

Problem Statement

Four numerical methods are compared, with the objective of evaluating their relative effectiveness at resolving shocks, rarefactions, and contact surfaces. The flux-vector splitting (FVS) Van Leer method, flux-difference splitting (FDS) Roe method, Van Leer method with Monotone Upwind-centered Scheme for Conservation Laws (MUSCL), and Roe method with MUSCL are used to solve a series of five test problems in a one-dimensional shock tube: the Sod problem, the 123 problem, two blast problems, and the shock collision problem. Solutions for density, velocity, pressure, and energy are plotted for each test case. Additional plots are generated for the Sod problem with Δx reduced, which are then compared to the previous plots for the Sod problem so that the effect of Δx on the accuracy of each numerical method can be studied. The convergence rate of each method is plotted by calculating the normalized L2 error norm at a series of Δx values that decrease by a factor of two with each grid refinement, and is compared to reference lines for Δx and $(\Delta x)^2$ order convergence. The ability of each numerical solver to provide numerically stable and accurate solutions for the 123 problem, both blast problems, and shock collision problem is discussed, and the relative effectiveness of both solvers with and without the addition of MUSCL is discussed.

The cases considered in this study are tabulated below for convenience.

Case	ρ_L	u_L	p_L	ρ_R	u_R	p_R	Max time	c_{max}	Δx	Test
1	1.0	0.0	1.0	0.0125	0.0	0.1	0.25	1.0	0.025	Sod
2	1.0	0.0	1.0	0.0125	0.0	0.1	0.25	0.8	0.025	Sod
3	1.0	0.0	1.0	0.0125	0.0	0.1	0.25	1.0	0.0125	Sod
4	1.0	0.0	1.0	0.0125	0.0	0.1	0.25	1.0	0.1	Sod
5	1.0	0.0	1.0	0.0125	0.0	0.1	0.25	1.0	0.05	Sod
6	1.0	0.0	1.0	0.0125	0.0	0.1	0.25	1.0	0.00625	Sod
7	1.0	0.0	1.0	0.0125	0.0	0.1	0.25	1.0	0.003125	Sod
8	1.0	0.0	1.0	0.0125	0.0	0.1	0.25	1.0	0.0015625	Sod
9	1.0	-2.0	0.4	1.0	2.0	0.4	0.15	1.0	0.0125	123
10	1.0	0.0	1000.0	1.0	0.0	0.01	0.012	1.0	0.0125	Blast 1
11	1.0	0.0	0.01	1.0	0.0	100.0	0.035	1.0	0.0125	Blast 2
12	5.999	19.598	460.894	5.994	-6.196	46.095	0.035	1.0	0.0125	Shock collision

Table 1: The initial conditions and key parameters for all of the cases used in this study are detailed.

The handwritten derivations that comprise the Homework 2 portion of this assignment are included in Appendix A. Python code developed for the numerical methods is included in Appendix B.

Method of Solution

The Van Leer method, Roe method, Van Leer method with MUSCL, and Roe method with MUSCL are written as separate functions, and are then imported and called as needed to generate plots in another Python script. The analytical solution is calculated using the *Riemann.py* script, which has been slightly modified so that it returns density, velocity, pressure, and energy rather than a plot of the solution (the plots will be generated separately). All of the functions are structured similarly outside of the time loop that calculates the solution. The function is defined with input parameters of *test*, *c_max*, and *Δx*. The *test* parameter, which is an integer ranging from 1 to 5, corresponds to the various test cases used for this study. Several *if* statements evaluate which value of this parameter has been passed to the function, and then initialize density, velocity, pressure, and the final time accordingly. Length *L* is defined as 1.0, specific heat ratio *gamma* is defined as 1.4 (for air), and number of nodes in the x-direction *ix* is calculated using *L* and *Δx*.

The halfway point of the domain is determined, and an array containing *x* values is created. Storage arrays for density, velocity, pressure, energy, *Q*, *F_plus, F_minus, and F_half* are initialized. These arrays are then populated with values corresponding to the left and right state of the appropriate test case. Energy *e* is then calculated using the equation of state (ideal gas law), and speed of sound is calculated using *gamma*, pressure, and density. The initial timestep *dt* is calculated from *c_max*, *Δx,* and the largest value of the speed of sound *a* plus the absolute value of velocity *u* in the computational domain. The solution for all variables is calculated inside of a *while loop* that runs until the variable *t* (which is initially zero) reaches the final time *t_final*, with *dt* dynamically calculated at the end of the loop.

MUSCL is added to both schemes by way of a separate function that extrapolates variables at the left and right state of each cell face by using Taylor series expansions about the adjacent nodes. This function is defined prior to the main time loop and is called as needed. Flow variables that will be required in the algorithm for the Van Leer and Roe methods are replaced by variables representing their left and right state values, which are calculated in the first *for* loop inside of the time loop by sending the appropriate nodal values of these variables to the MUSCL function. Once the left and right state values are returned by the MUSCL function for each cell face in the domain, the rest of the algorithm can be executed in the same way as before.

The Roe method (both with and without MUSCL) utilizes an additional function, which (like the MUSCL implementation) is defined prior to the time loop. This function is called with flow variables at left and right nodes and density at left and right nodes, and returns the density-weighted Roe average of these variables. This is then used in the Roe method algorithm.

The exact contents of the time loop depend on the particular method, but all loops return the *x* array, density array, velocity array, pressure array, and energy array, which can then be used for plotting. The equations that define the algorithm used for each method are included in the following pages.

Van Leer method algorithm

$$Q^{n+1} = Q^n - \frac{\Delta t}{\Delta x}\left[F^n_{i+\frac{1}{2}} - F^n_{i-\frac{1}{2}}\right] \tag{1}$$

$$F^n = F^+ + F^- \tag{2}$$

$$F^{\pm} = \pm\frac{1}{4}\rho a(1 \pm M)^2 \begin{bmatrix} 1 \\ \frac{2a}{\gamma}\left(\frac{\gamma-1}{2}M \pm 1\right) \\ \frac{2a^2}{\gamma^2-1}\left(\frac{\gamma-1}{2}M \pm 1\right)^2 \end{bmatrix} \tag{3}$$

Roe method algorithm

$$Q^{n+1} = Q^n - \frac{\Delta t}{\Delta x}\left[F^n_{i+\frac{1}{2}} - F^n_{i-\frac{1}{2}}\right] \tag{4}$$

$$F_{i+\frac{1}{2}} = \frac{1}{2}\begin{bmatrix} F_{1,i} + F_{1,i+1} \\ F_{2,i} + F_{2,i+1} \\ F_{3,i} + F_{3,i+1} \end{bmatrix} - \frac{1}{2}\begin{bmatrix} \bar{\alpha}_1|\tilde{u} - \tilde{a}| + \bar{\alpha}_2|\tilde{u} + \tilde{a}| + \bar{\alpha}_3|\tilde{u}| \\ \bar{\alpha}_1|\tilde{u} - \tilde{a}|(\tilde{u} - \tilde{a}) + \bar{\alpha}_2|\tilde{u} + \tilde{a}|(\tilde{u} + \tilde{a}) + \bar{\alpha}_3|\tilde{u}|\tilde{u} \\ \bar{\alpha}_1|\tilde{u} - \tilde{a}|(\tilde{H} - \tilde{u}\tilde{a}) + \bar{\alpha}_2|\tilde{u} + \tilde{a}|(\tilde{H} + \tilde{u}\tilde{a}) + \bar{\alpha}_3|\tilde{u}|\tilde{u}^2/2 \end{bmatrix} \tag{5}$$

Roe average

$$\tilde{\rho} = \sqrt{\rho_L \rho_R} \tag{6}$$

$$\tilde{\beta} = \frac{\left(\beta\sqrt{\rho}\right)_L + \left(\beta\sqrt{\rho}\right)_R}{\left(\sqrt{\rho}\right)_L + \left(\sqrt{\rho}\right)_R} \tag{7}$$

Smooth flux limiter

$$\psi(r) = \frac{|r| + r}{1 + |r|} \tag{8}$$

Monotone Upwind-centered Scheme for Conservation Laws (MUSCL) algorithm

Ratios of consecutive gradients

$$r^{-}_{i+\frac{1}{2}} = \frac{v_{i+1} - v_i}{v_i - v_{i-1}} \tag{9a}$$

$$r^{-}_{i+\frac{3}{2}} = \frac{v_{i+1} - v_i}{v_{i+2} - v_{i+1}} \tag{9b}$$

Flux limiter reciprocal relationships

$$\psi\left(r^{+}_{i-\frac{1}{2}}\right) = \frac{1}{r^{-}_{i+\frac{1}{2}}}\psi\left(r^{-}_{i+\frac{1}{2}}\right) \tag{10a}$$

$$\psi\left(r^{+}_{i+\frac{1}{2}}\right) = \frac{1}{r^{-}_{i+\frac{3}{2}}}\psi\left(r^{-}_{i+\frac{3}{2}}\right) \tag{10b}$$

Definition of + and - operators

$$\Delta^{+}_{i-\frac{1}{2}} = \frac{(v_i - v_{i-1})\psi\left(r^{+}_{i-\frac{1}{2}}\right)}{2} \tag{11a}$$

$$\Delta^{-}_{i+\frac{1}{2}} = \frac{(v_{i+1} - v_i)\psi\left(r^{-}_{i+\frac{1}{2}}\right)}{2} \tag{11b}$$

$$\Delta^{-}_{i+\frac{3}{2}} = \frac{(v_{i+2} - v_{i+1})\psi\left(r^{-}_{i+\frac{3}{2}}\right)}{2} \tag{11c}$$

$$\Delta^{+}_{i+\frac{1}{2}} = \frac{(v_{i+1} - v_i)\psi\left(r^{+}_{i+\frac{1}{2}}\right)}{2} \tag{11d}$$

Definition of L and R operators

$$\Delta^L(v) = (1-\kappa)\Delta^+_{i-\frac{1}{2}} + (1+\kappa)\Delta^-_{i+\frac{1}{2}} \tag{12a}$$

$$\Delta^R(v) = (1-\kappa)\Delta^-_{i+\frac{3}{2}} + (1+\kappa)\Delta^+_{i+\frac{1}{2}} \tag{12b}$$

$$\kappa = \begin{cases} -1 \\ 1 \\ 0 \\ 1/3 \end{cases} \tag{12c}$$

Definition of L and R state values for flow variable

$$v_L = v_i + \frac{1}{2}\Delta^L(v) \tag{13a}$$

$$v_R = v_{i+1} - \frac{1}{2}\Delta^R(v) \tag{13b}$$

Results and Discussion

Part 1

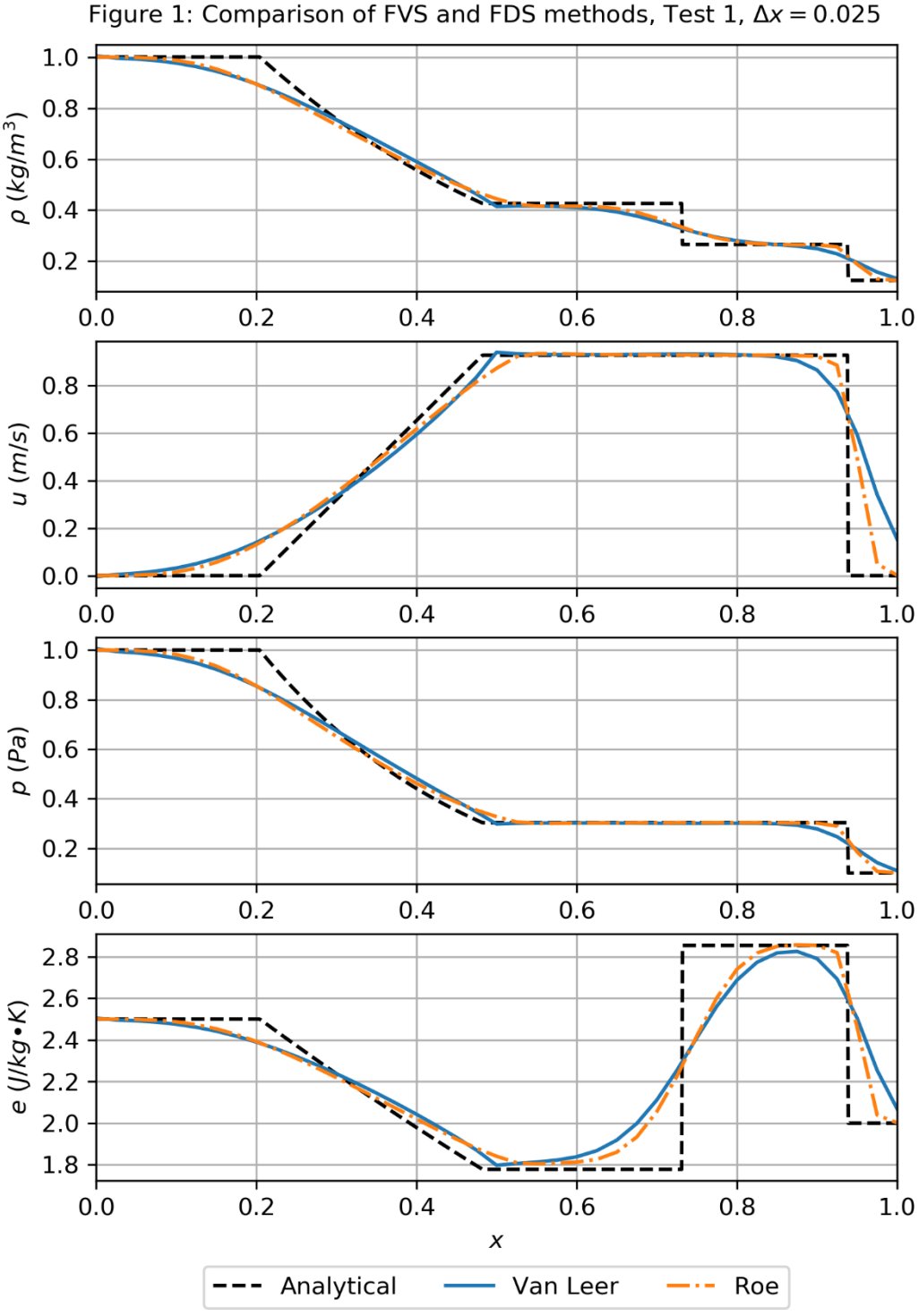

Figure 1: The analytical (Riemann solver) solution, Van Leer method, and Roe method are compared for the Sod problem at a grid resolution of Δx = 0.025.

Figure 1 shows a comparison between the analytical solution (Riemann solver), Van Leer method solution, and Roe method solution to the Sod problem at a grid resolution of $\Delta x = 0.025$. It is apparent that both numerical methods closely approximate the analytical solution in regions away from discontinuities, such as in the regions of constant velocity and pressure across the contact surface or the regions where all flow variables vary linearly across the left-moving rarefaction wave. Dissipative error is evident in both schemes, with smearing of the contact surface apparent in the density and energy plots and peak suppression visible in all plots. The energy plot is the most telling in this regard – both methods fail to capture the discontinuity in energy at the contact surface near $x = 0.7$, and produce a suppressed peak with a spread base rather than the square wave structure of the analytical solution. The Roe method performs slightly better overall, with less severe undershoots in the velocity and energy plots, especially near discontinuities where flow variables are decreasing such as near the shock at $x = 0.9$. Dispersive error does not seem to be a factor for either method, with no oscillations visible near discontinuities. This is expected, since these methods are upwinded descriptions of the flow and should be monotonicity preserving.

Part 2

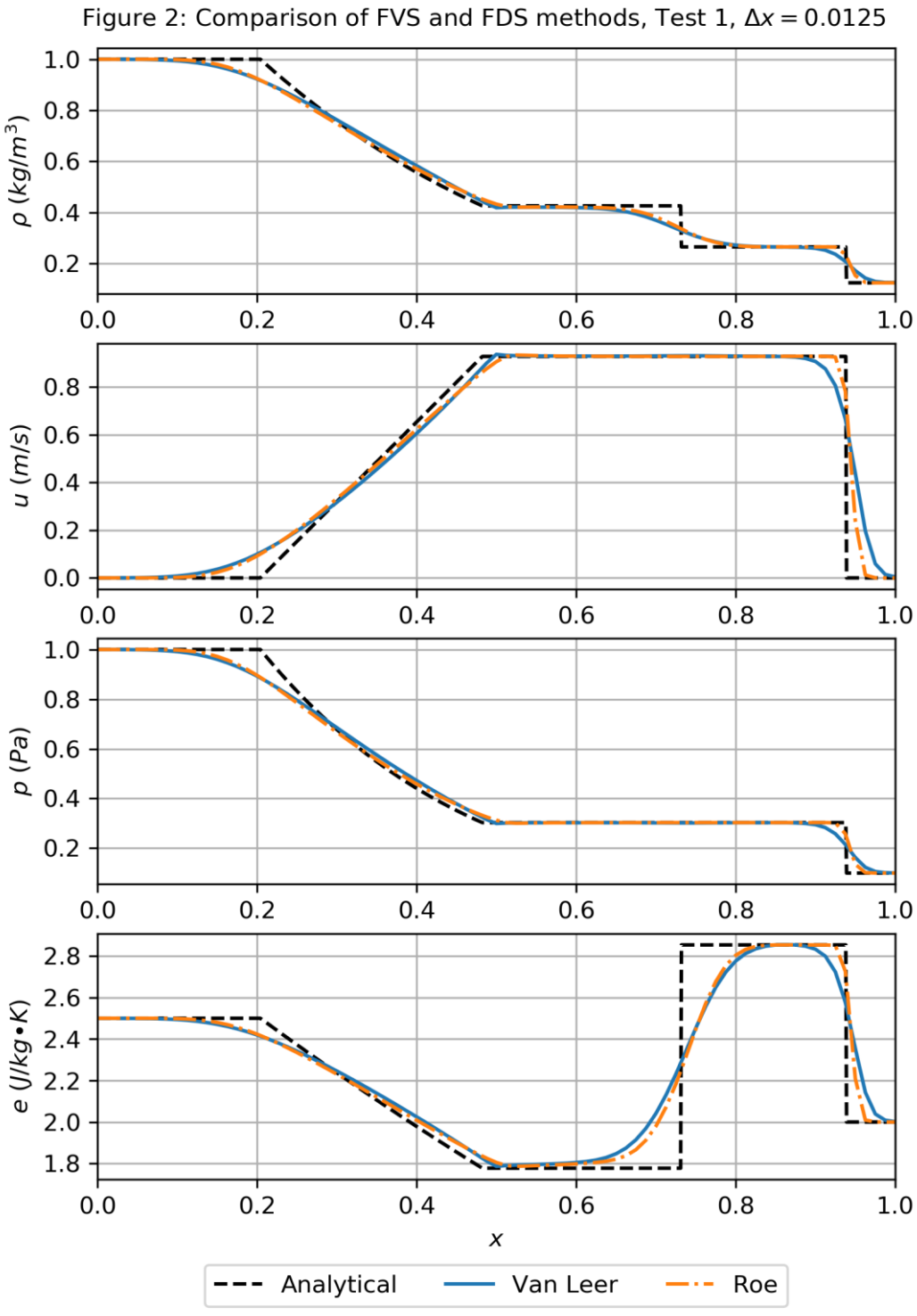

Figure 2: The analytical (Riemann solver) solution, Van Leer method, and Roe method are compared for the Sod problem at a grid resolution of Δx = 0.0125.

Figure 2 shows a comparison between the analytical solution (Riemann solver), Van Leer method solution, and Roe method solution to the Sod problem at a grid resolution of $\Delta x = 0.0125$. Solution accuracy has increased slightly relative to the coarser grid solutions in Fig. 1, most significantly near discontinuities. This can be attributed to a decrease in dissipative error due to grid refinement. The Roe method solution is again more accurate than the Van Leer method solution, especially near discontinuities where variables decrease (such as near $x = 0.9$). Dispersive error is not a factor for either method.

Part 3

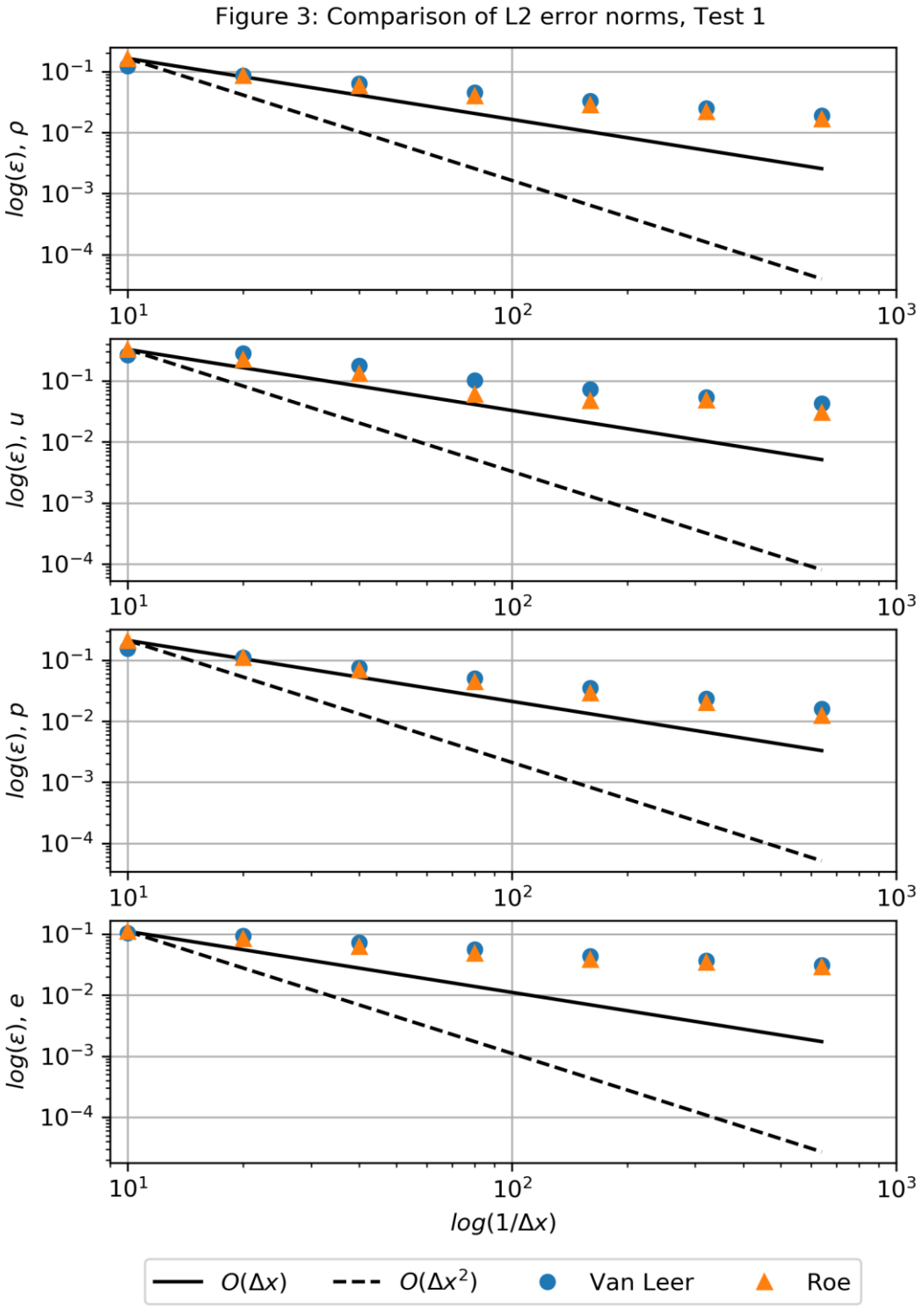

Figure 3: Convergence rates for the Van Leer method and Roe method are plotted for the Sod problem and compared to each other and to first-order and second-order reference lines.

Figure 3 compares normalized L2 error norms for the Van Leer method and Roe method at values of Δx that decrease by a factor of two with each grid refinement. This is a measure of the convergence rate of each method, and is plotted against reference lines corresponding to first-order and second-order convergence. The convergence rate of the two numerical methods appears to be very similar for density, pressure, and energy, with slight differences for velocity. Density, velocity, and pressure appear to converge at similar rates for both methods, falling just short of first-order convergence. Energy is slower to converge for both methods, which may be due to the larger discontinuity present in this flow variable. The largest inaccuracy in the solutions observed in Fig. 1 and Fig. 2 occurred in the energy variable near x = 0.7, where dissipative error of both methods led to significant smearing of the contact surface.

Part 4

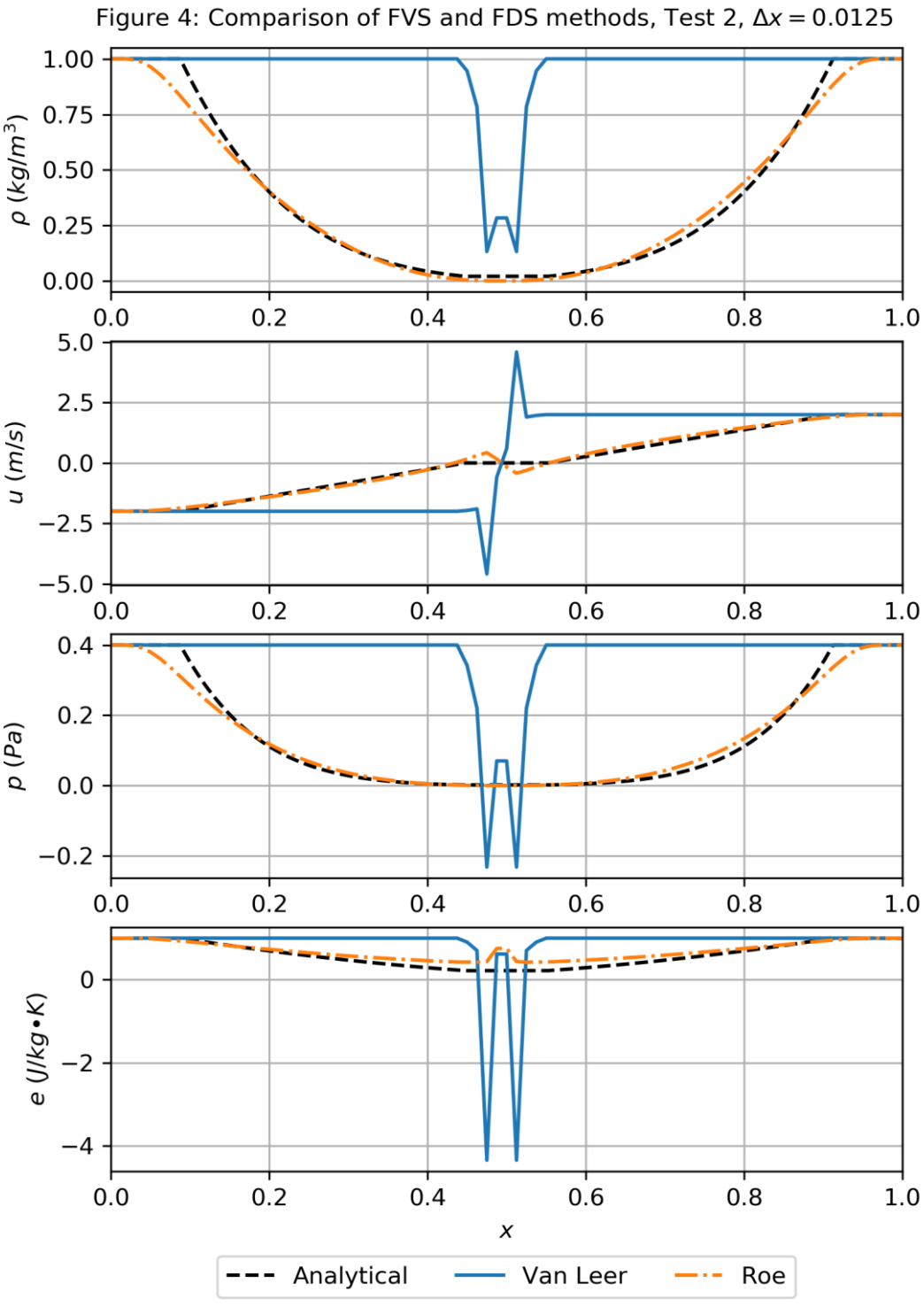

Figure 4: The analytical (Riemann solver) solution, Van Leer method, and Roe method are compared for the 123 problem at a grid resolution of Δx = 0.0125.

Figure 4 shows the analytical solution (Riemann solver), Van Leer method solution, and Roe method solution to the 123 problem at a grid resolution of $\Delta x = 0.0125$. The Van Leer method is not numerically stable for this test case, failing after several iterations due to negative values of energy and subsequently negative values of pressure. This results in an imaginary speed of sound, which cannot be used to calculate the subsequent timestep, breaking the solver. The Roe method, which has an entropy fix implemented, provides a reasonable solution to this problem despite some discontinuities in velocity and energy visible near $x = L/2$ and the presence of dissipative error. It should be noted that without this entropy fix, the 123 problem (which contains two rarefaction waves) results in numerical instability in the Roe method as well. The Roe method assumes that these rarefactions (which are isentropic) behave as shocks (which generate entropy) to eliminate the shock/rarefaction identification logic present in the Godunov method. This fails to satisfy entropy unless a fix is applied.

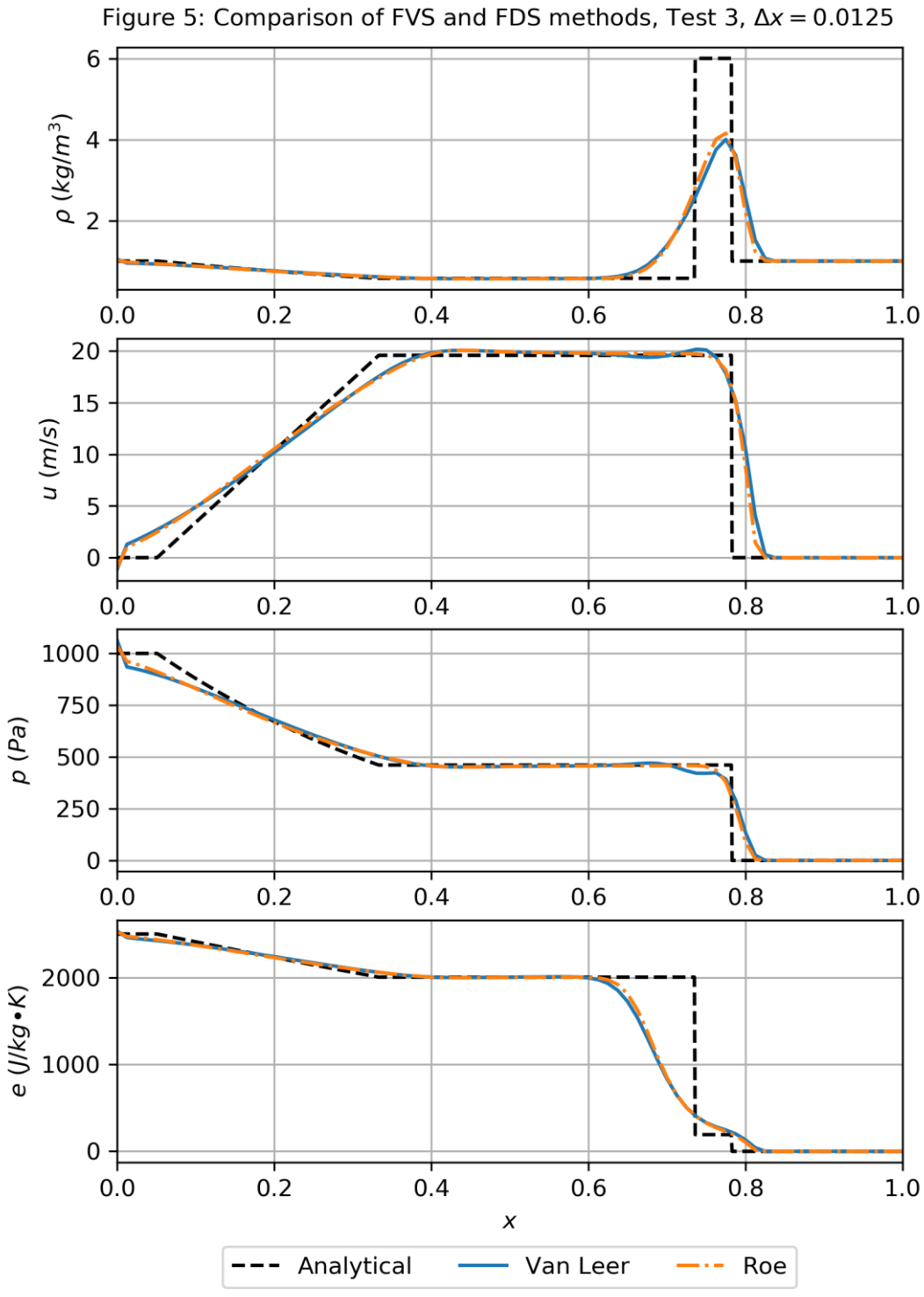

Figure 5: The analytical (Riemann solver) solution, Van Leer method, and Roe method are compared for the blast problem (right running shock, left running rarefaction) at a grid resolution of Δx = 0.0125.

Figure 5 shows the analytical solution (Riemann solver), Van Leer method solution, and Roe method solution to the blast problem with a right running shock and left running rarefaction at a grid resolution of $\Delta x = 0.0125$. Both numerical methods appear to be numerically stable and provide similar solutions to this problem, with only minor differences near discontinuities where the Roe method is slightly more accurate. The solution is accurate in regions where flow variables are constant or linearly varying, but dissipative error results in severe peak suppression and undershoots in regions where multiple discontinuities occur a short distance apart (such as near $x = 0.8$).

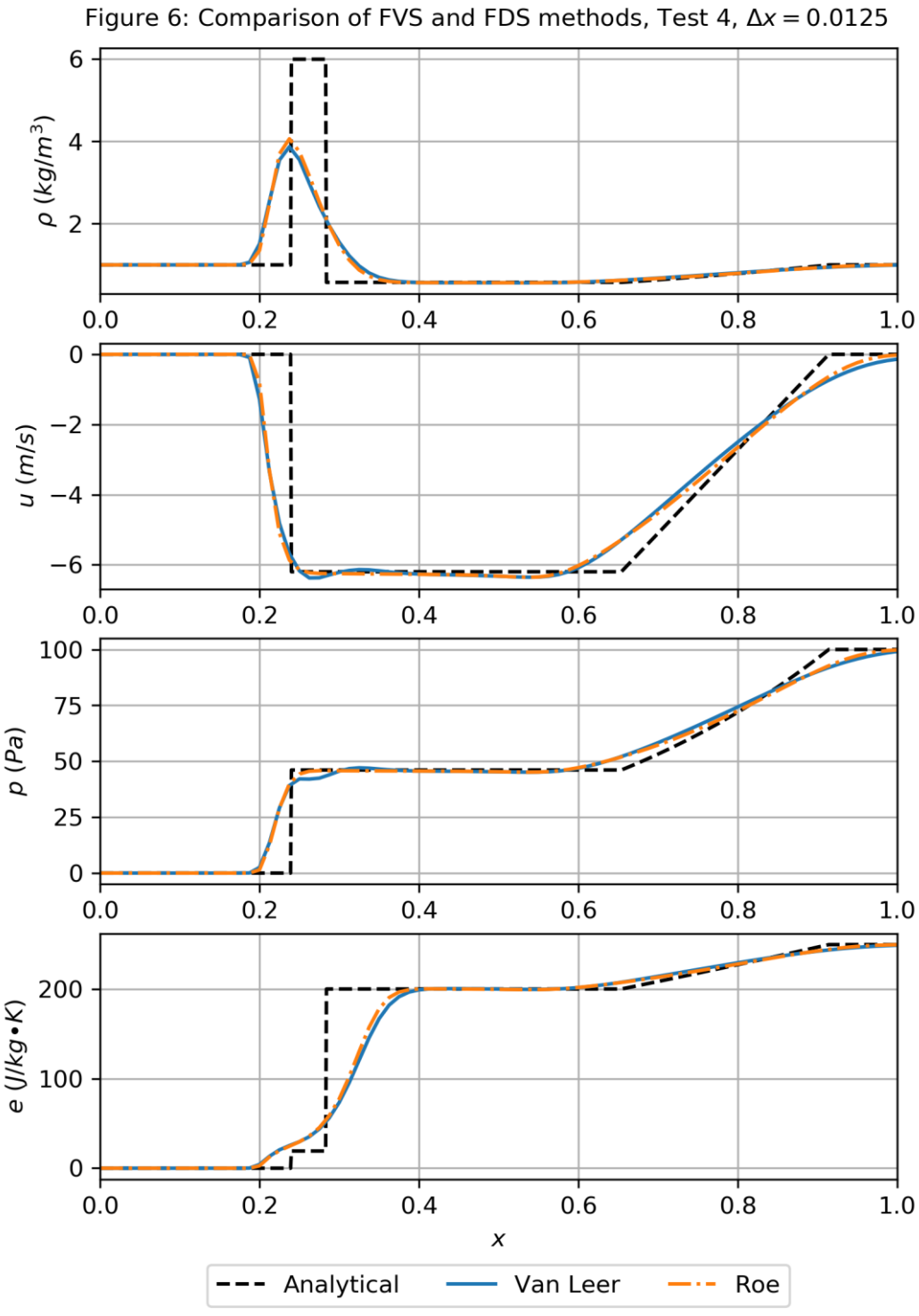

Figure 6: The analytical (Riemann solver) solution, Van Leer method, and Roe method are compared for the blast problem (left running shock, right running rarefaction) at a grid resolution of Δx = 0.0125.

Figure 6 shows the analytical solution (Riemann solver), Van Leer method solution, and Roe method solution to the blast problem with a left running shock and right running rarefaction at a grid resolution of $\Delta x = 0.0125$. The results here are very similar to those of the previous blast problem in Fig. 5 – once again, both methods are numerically stable and provide almost identical solutions. Peak suppression, undershoots, and smearing of contact surfaces are evident due to dissipative error.

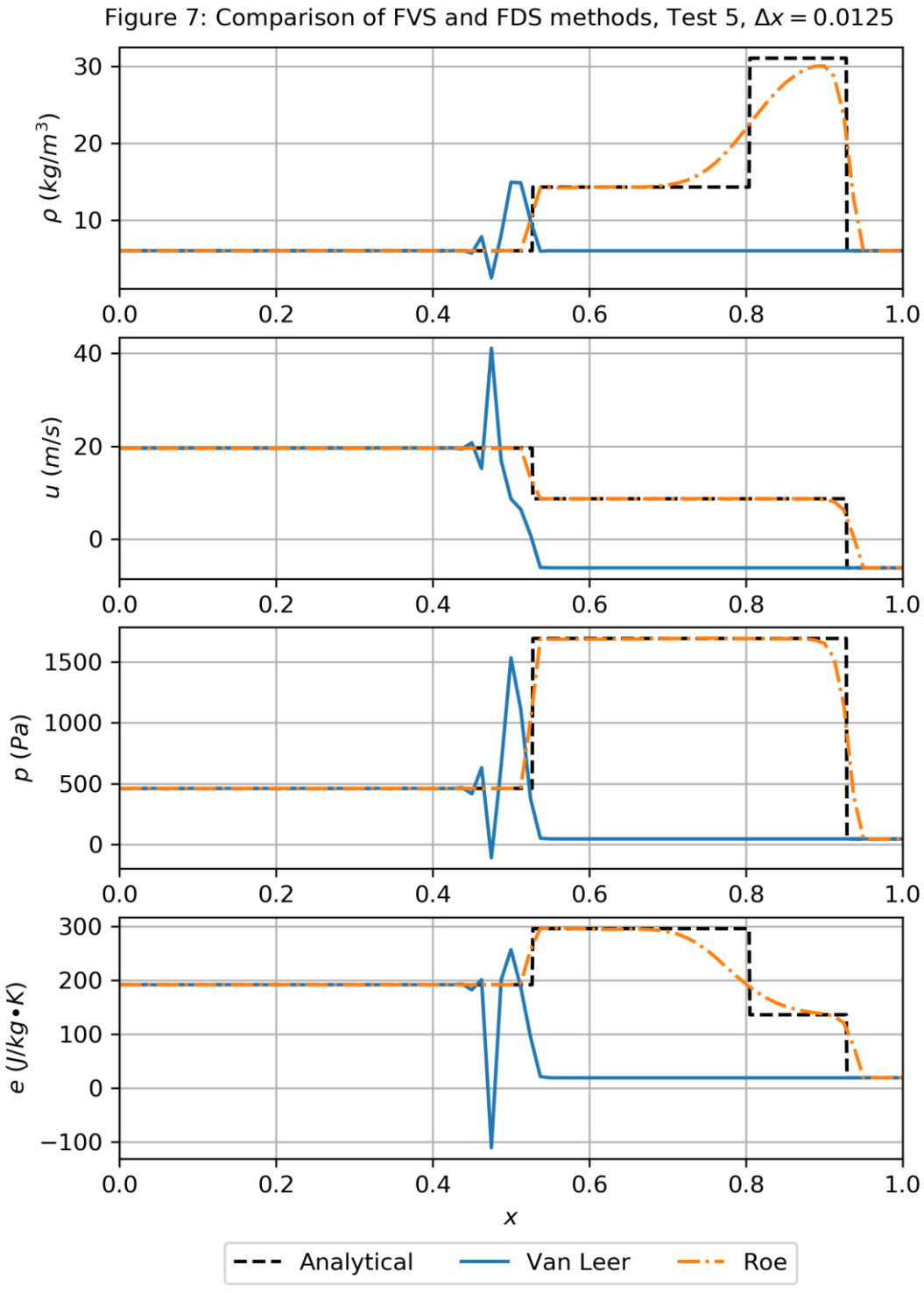

Figure 7: The analytical (Riemann solver) solution, Van Leer method, and Roe method are compared for the shock collision problem at a grid resolution of Δx = 0.0125.

Figure 7 shows the analytical solution (Riemann solver), Van Leer method solution, and Roe method solution to the shock collision problem at a grid resolution of $\Delta x = 0.0125$. The Van Leer method fails after several iteration due to a negative value of energy and subsequently negative value of pressure, which results in an imaginary speed of sound a similarly to the 123 problem shown in Fig. 4. The Roe method with an entropy fix is numerically stable for this problem, demonstrating that this method can be applied to all five test problems. Suppression of peaks and smearing of contact surfaces due to dissipative error is apparent in the solution, but solution accuracy is acceptable away from these discontinuities.

The Van Leer method is numerically stable for cases that consider both shocks and rarefactions such as the Sod problem and blast problems, but fails for cases that consider shock-shock or rarefaction-rarefaction interactions such as the 123 problem and shock collision problem. The Roe method with an entropy fix is numerically stable for all test cases. Solutions provided by both methods for shock-rarefaction cases (Test 1, 3, and 4) are almost identical. The Van Leer method and Roe method are equally suited to problems concerning shocks and rarefactions, but only the Roe method with an entropy fix is capable of producing a stable solution in shock-shock or rarefaction-rarefaction interaction problems.

Part 5

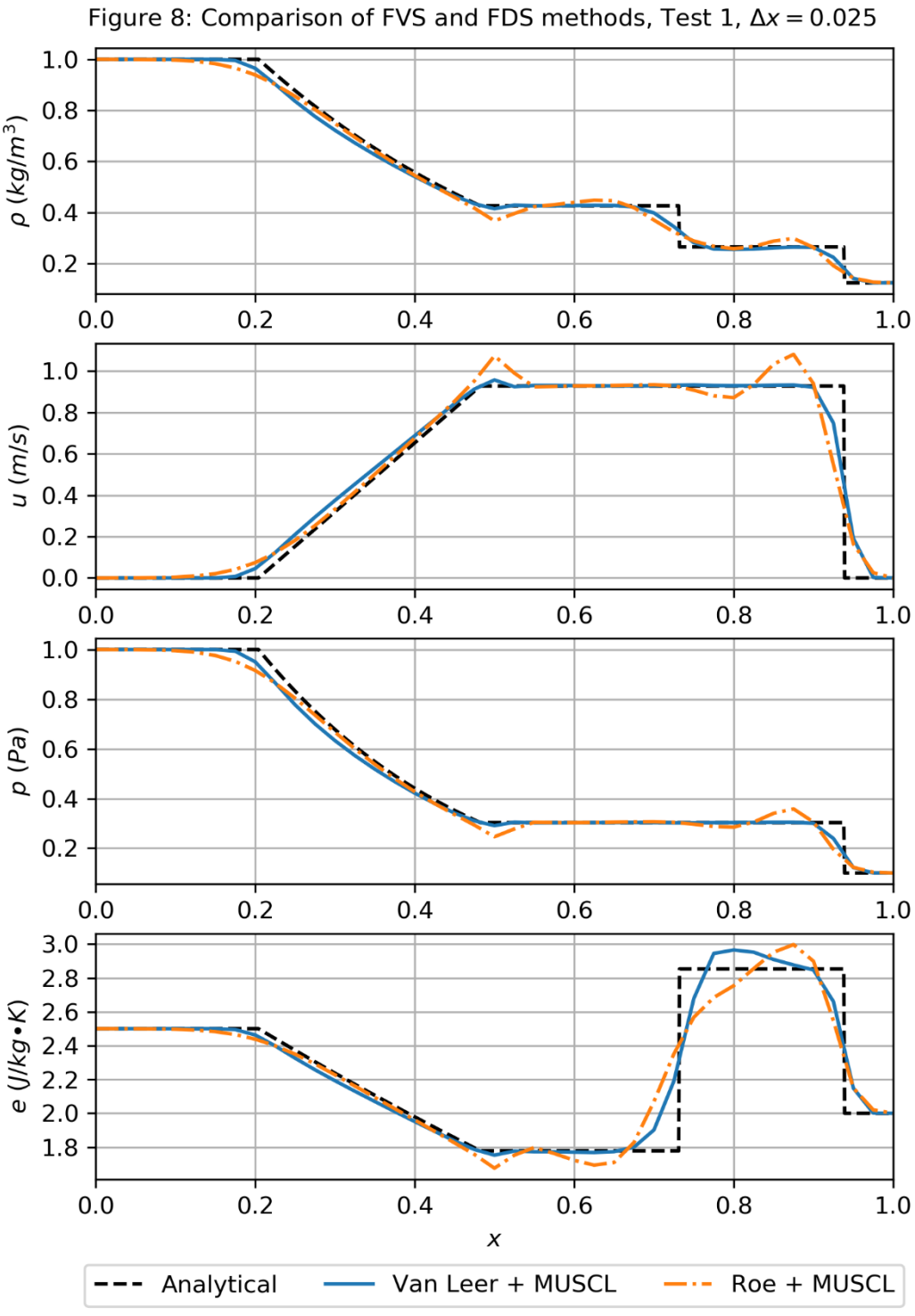

Figure 8: The analytical (Riemann solver) solution, Van Leer method with MUSCL, and Roe method with MUSCL are compared for the Sod problem at a grid resolution of Δx = 0.025.

Figure 8 shows the analytical solution (Riemann solver), Van Leer method with MUSCL solution, and Roe method with MUSCL solution to the Sod problem at a grid resolution of $\Delta x = 0.025$. Implementation of MUSCL into the Van Leer method has slightly increased the accuracy overall, especially near the contact surface located at approximately $x = 0.7$. Dissipative error looks to have been reduced, which results in less contact surface smearing and peak suppression, but there is now an overshoot in the energy variable that was not observed in the corresponding case without MUSCL (see Fig. 1). Implementation of MUSCL into the Roe scheme results in oscillations in the solution near discontinuities, an unexpected result given that the lower-order scheme is monotonicity-preserving. This occurs due to a fundamental mismatch in the assumptions of the Roe method, which is an approximate Riemann solver, and the MUSCL extension. MUSCL extrapolates the variables at the left and right state of a cell face by using Taylor series expansions about the cell nodes to the left and right, constructing some polynomial that describes how these variables vary inside of the left and right cells. This is incompatible with the assumption made by Riemann solvers, which is that these variables have uniform values within cells.

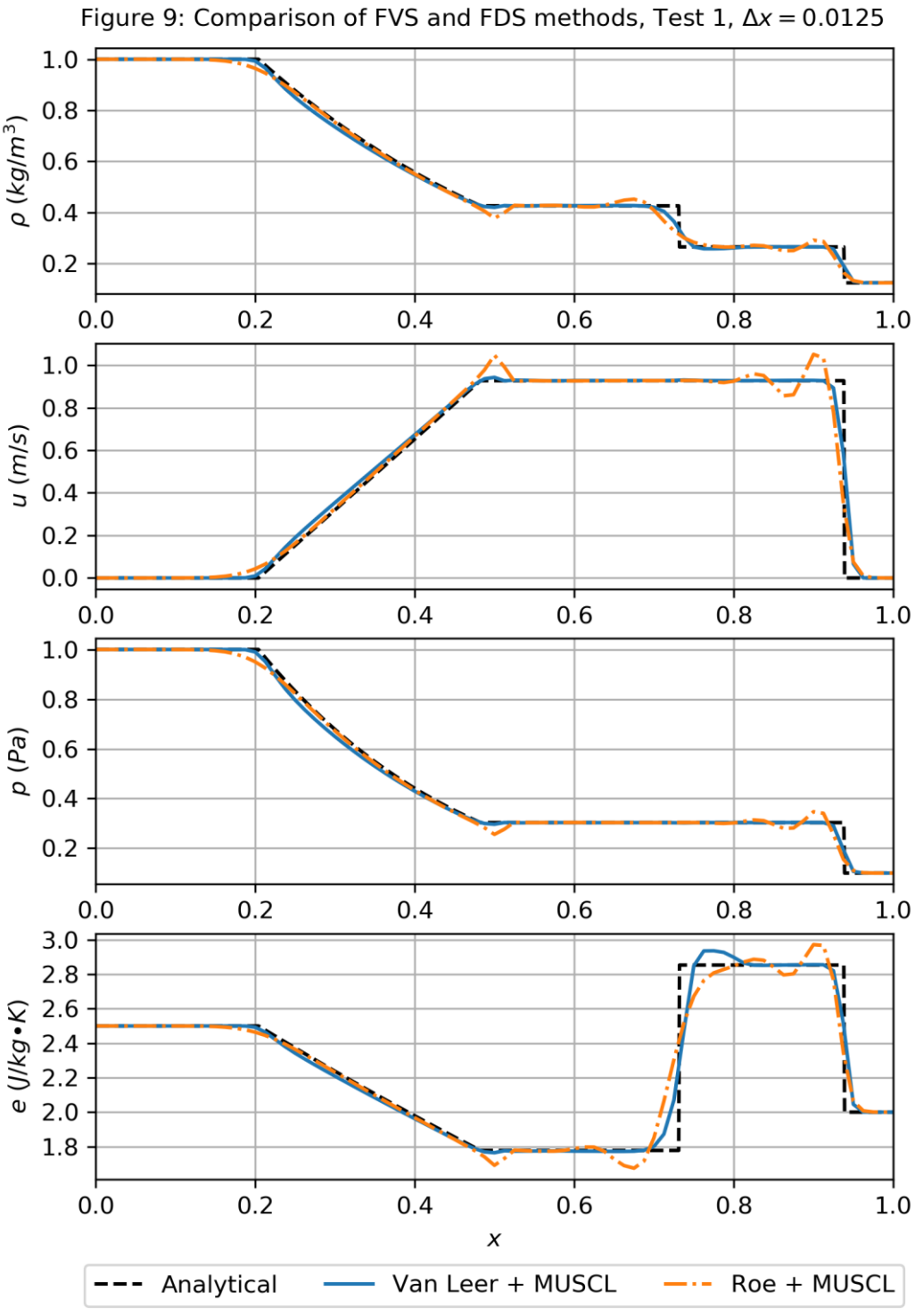

Figure 9: The analytical (Riemann solver) solution, Van Leer method with MUSCL, and Roe method with MUSCL are compared for the Sod problem at a grid resolution of Δx = 0.0125.

Figure 9 shows the analytical solution (Riemann solver), Van Leer method with MUSCL solution, and Roe method with MUSCL solution to the Sod problem at a grid resolution of $\Delta x = 0.0125$. Solution accuracy has increased for both methods due to decreased dissipative error compared to the $\Delta x = 0.025$ case. The overshoot introduced into the Van Leer energy solution near the contact surface with the addition of MUSCL seems to be significantly smaller for this higher resolution, and the solution is highly accurate away from the contact surface. The Roe method again suffers from spurious oscillations, and though contact surfaces are more clearly resolved this method should not be used where negative values of a flow variable caused by such oscillations can result in non-physical results (such as in a problem where species concentrations may become negative, for example).

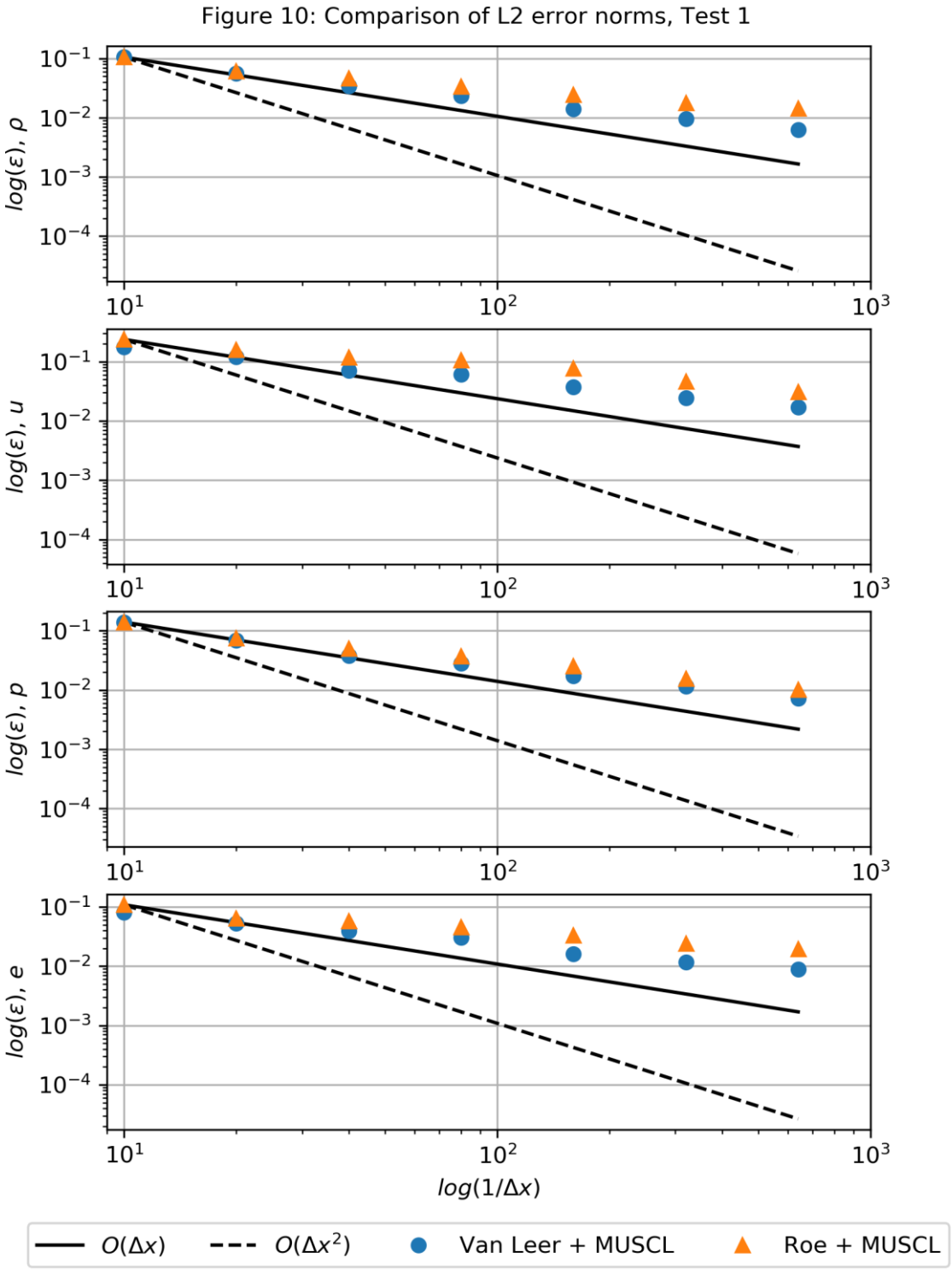

Figure 10: Convergence rates for the Van Leer method with MUSCL and Roe method with MUSCL are plotted for the Sod problem and compared to each other and to first-order and second-order reference lines.

Figure 10 compares normalized L2 error norms for the Van Leer method with MUSCL and Roe method with MUSCL at values of Δx that decrease by a factor of two with each grid refinement. This is a measure of the convergence rate of each method, and is plotted against reference lines corresponding to first-order and second-order convergence. The convergence rate for the two methods is very similar for all flow variables, and appears to be slightly slower than first-order. The energy variable, which had a slower convergence rate in the non-MUSCL versions of the Van Leer and Roe methods (Fig. 4), now appears to converge at the same rate as density, velocity, and pressure. This is evidence that he convergence rate for energy was being slowed by the dissipative smearing of the contact surface near x = 0.7, and that the addition of MUSCL has allowed the contact surface to be resolved more clearly.

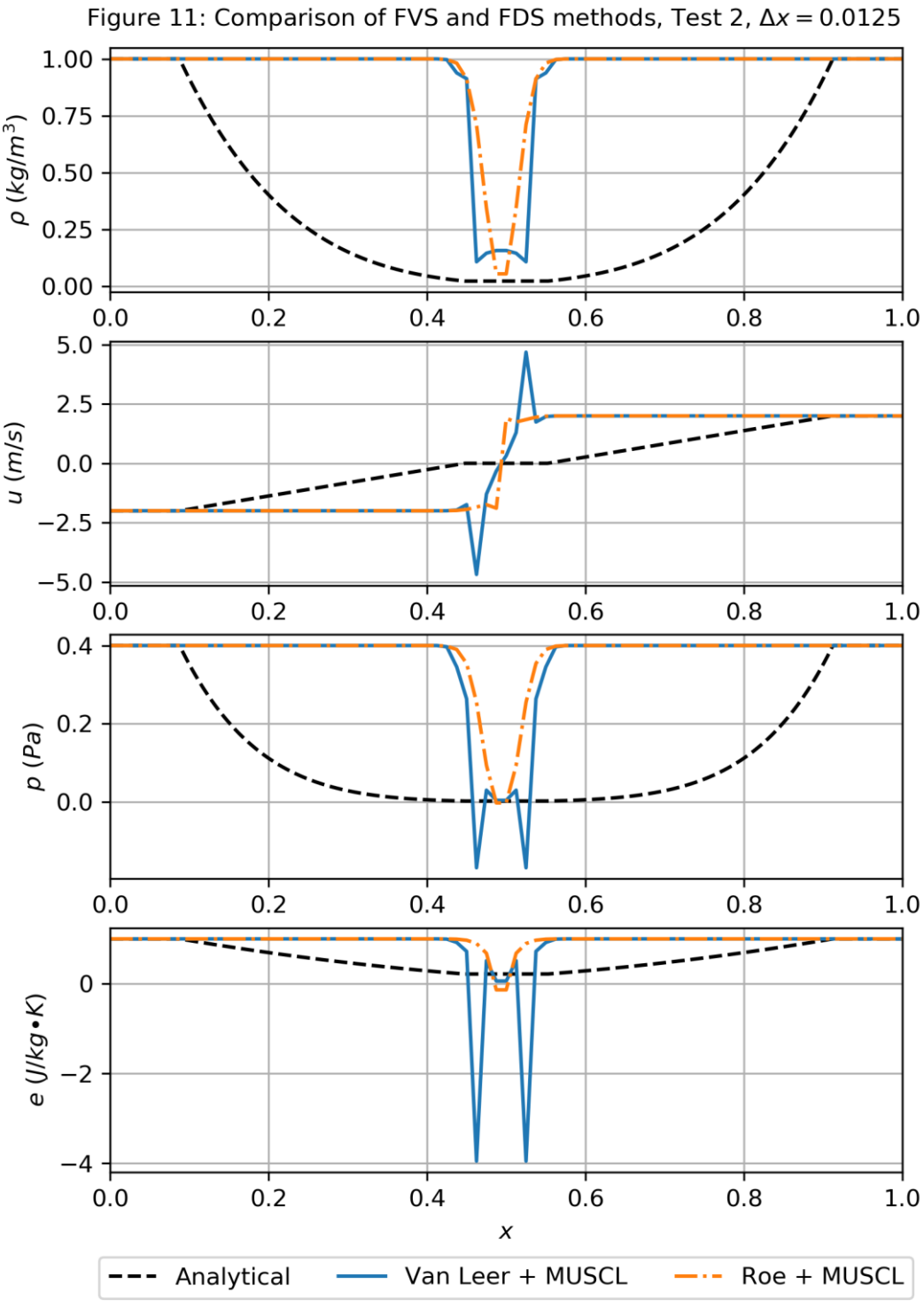

Figure 11: The analytical (Riemann solver) solution, Van Leer method with MUSCL, and Roe method with MUSCL are compared for the 123 problem at a grid resolution of Δx = 0.0125.

Figure 11 shows the analytical solution (Riemann solver), Van Leer method with MUSCL solution, and Roe method with MUSCL solution to the 123 problem at a grid resolution of $\Delta x = 0.0125$. The Van Leer method with MUSCL fails to produce a solution for this rarefaction-rarefaction interaction problem, failing after several iterations due to an imaginary speed of sound (due to negative energy and subsequently negative pressure). This result is expected, since the Van Leer method was not numerically stable without MUSCL for this test case. The Roe method, which produced a stable solution without MUSCL for this test case, also fails after several iterations with the addition of MUSCL. This can be attributed to the incompatibility in assumptions made by the Roe method and the MUSCL scheme, as discussed previously.

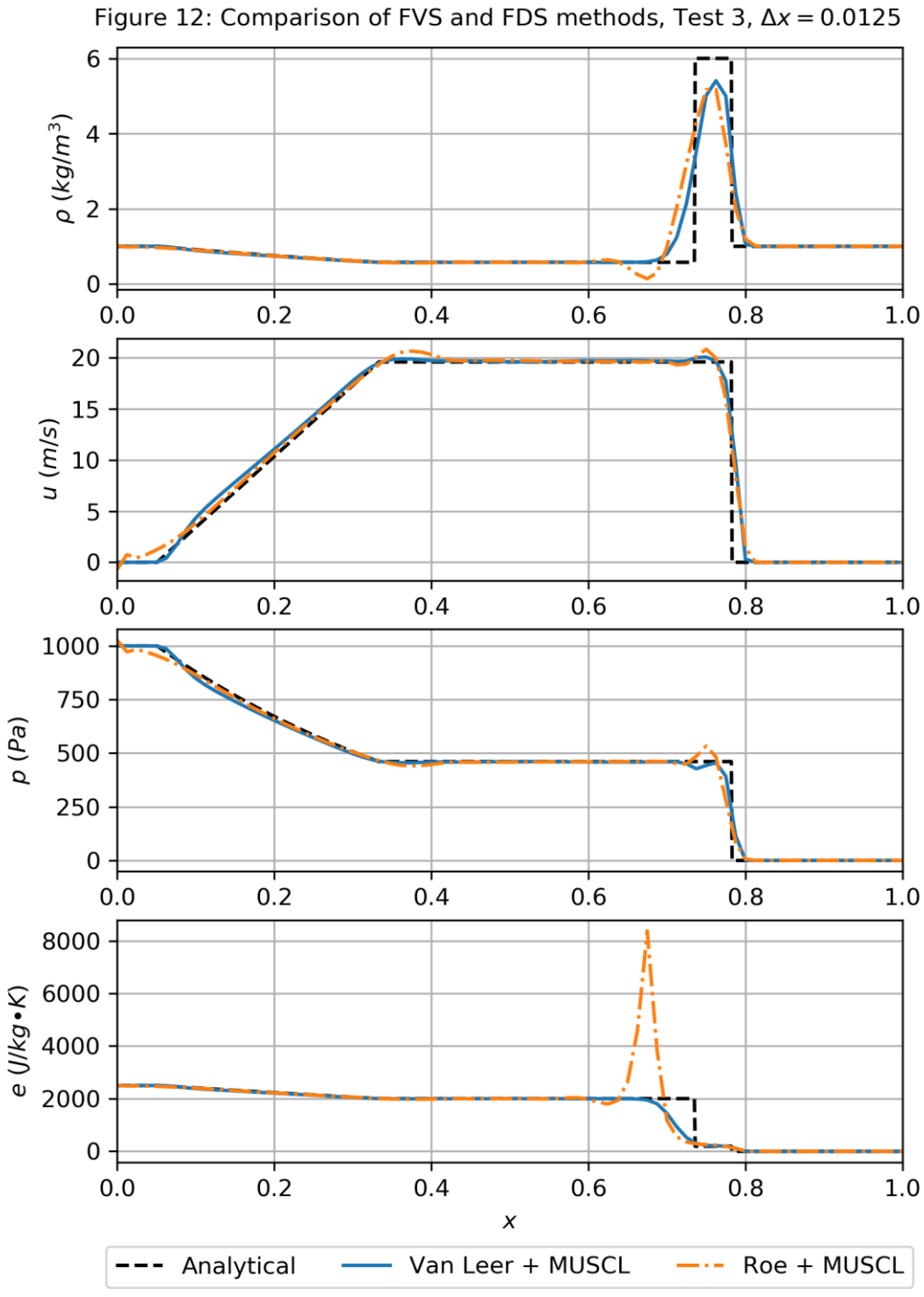

Figure 12: The analytical (Riemann solver) solution, Van Leer method with MUSCL, and Roe method with MUSCL are compared for the blast problem (right running shock, left running rarefaction) at a grid resolution of Δx = 0.0125.

Figure 12 shows the analytical solution (Riemann solver), Van Leer method with MUSCL solution, and Roe method with MUSCL solution to the blast problem with a right running shock and left running rarefaction at a grid resolution of $\Delta x = 0.0125$. The addition of MUSCL to the Van Leer scheme significantly improves the results for this test case relative to the same test case without MUSCL (Fig. 5). Dissipative error has been significantly reduced, leading to less peak suppression and contact surface smearing – this is especially evident in the density variable solution near the contact surface. The Roe method similarly suffers from less dissipation, but oscillations are present in the solution near discontinuities. These oscillations are fairly minor in the density, velocity, and pressure variables, but result in a large peak in the energy variable solution that occurs at approximately $x = 0.65$. This peak is more than quadruple the analytical solution, a clearly non-physical result.

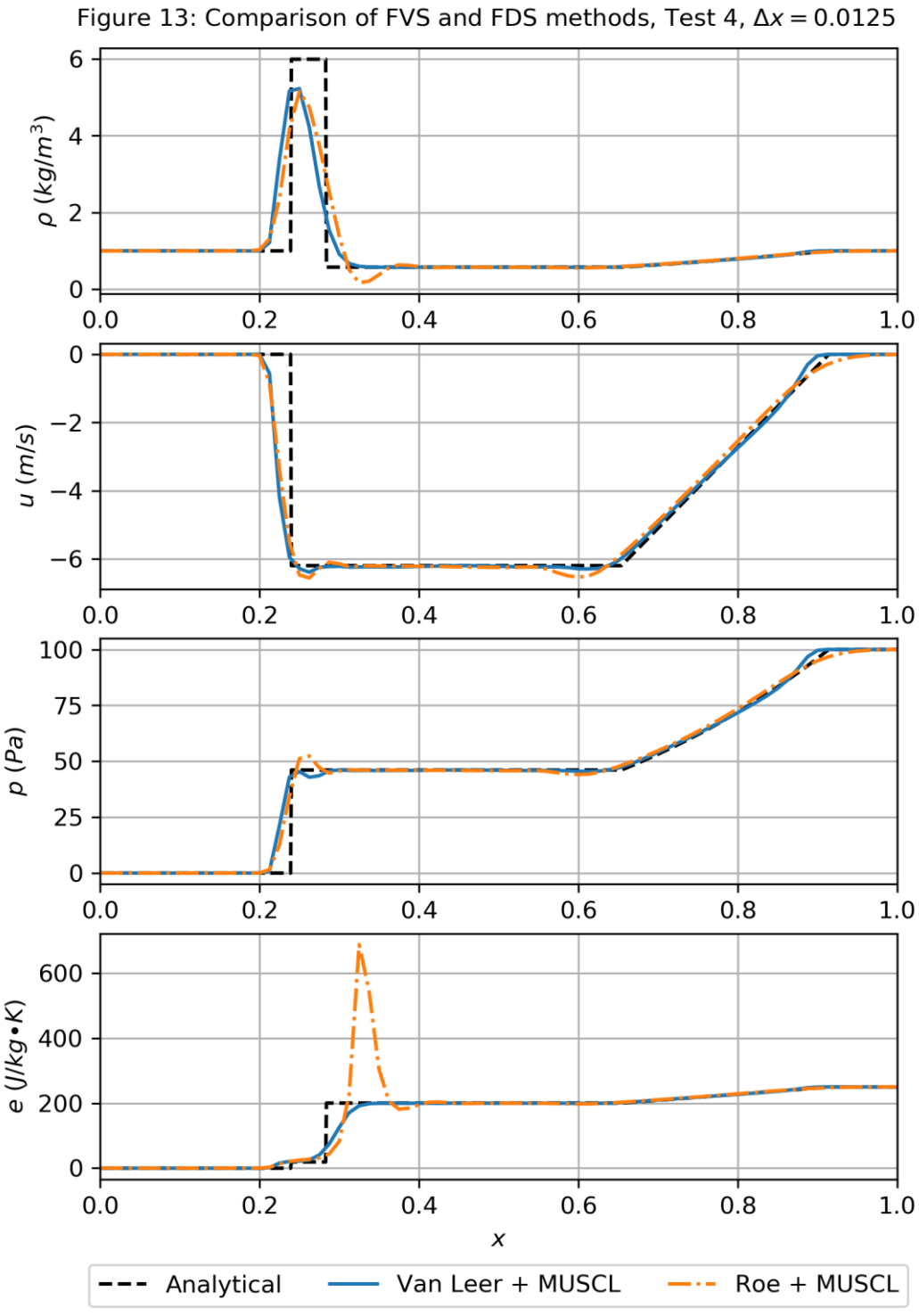

Figure 13: The analytical (Riemann solver) solution, Van Leer method with MUSCL, and Roe method with MUSCL are compared for the blast problem (left running shock, right running rarefaction) at a grid resolution of Δx = 0.0125.

Figure 13 shows the analytical solution (Riemann solver), Van Leer method with MUSCL solution, and Roe method with MUSCL solution to the blast problem with a left running shock and right running rarefaction at a grid resolution of $\Delta x = 0.0125$. These results mirror those of Fig. 12 – the Van Leer method with MUSCL is more accurate than the same scheme without MUSCL, especially for the density variable near the contact surface, while the Roe method with MUSCL contains oscillations that are most pronounced in the energy variable. There is again a large peak in the energy variable near the contact surface for the Roe method with MUSCL, but this overshoot is of smaller magnitude relative to the analytical solution than it is in Fig. 12. This suggests that the severity of these oscillatory overshoots relative to the analytical solution is dependent upon the magnitude of the discontinuity that causes the oscillation.

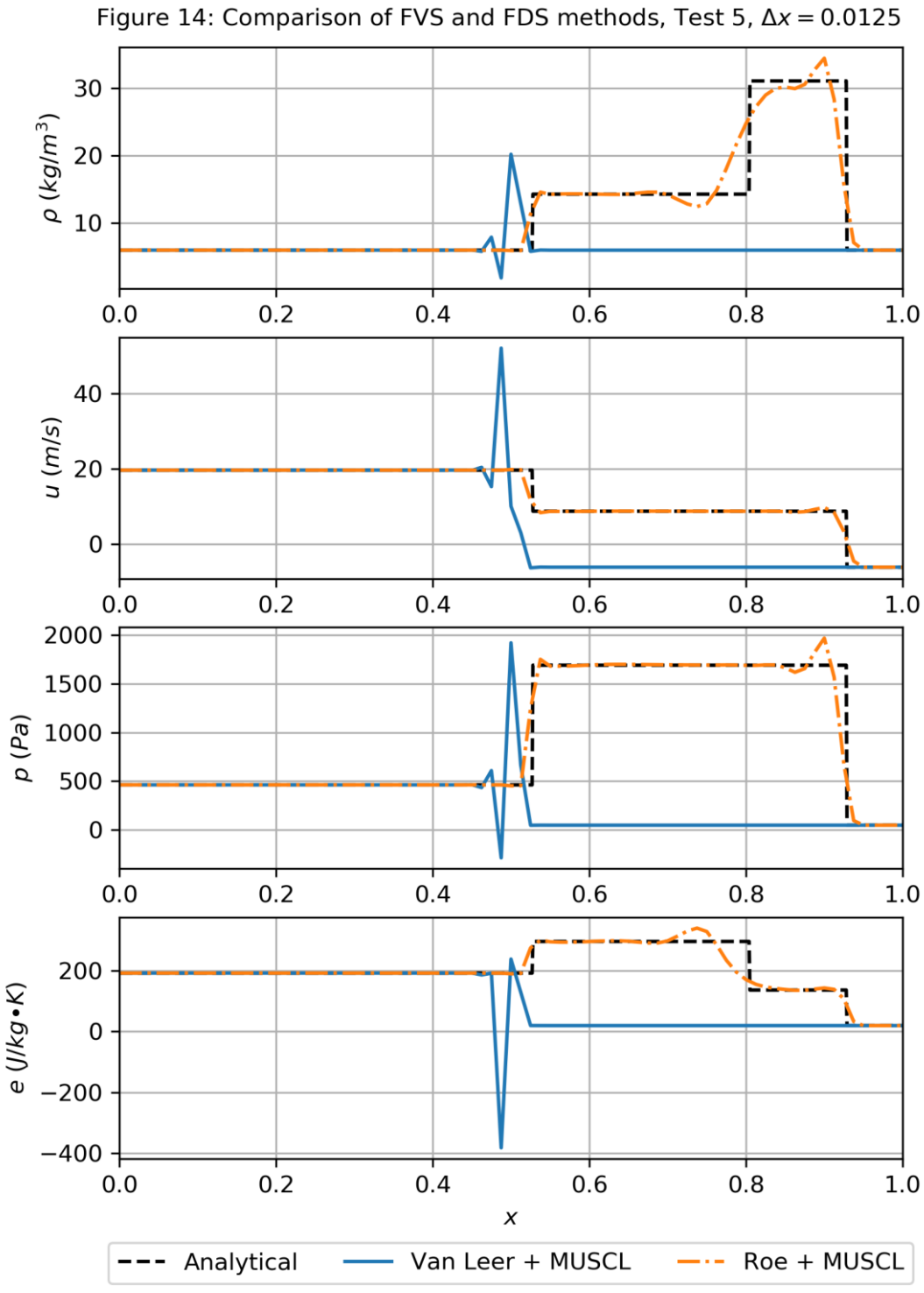

Figure 14: The analytical (Riemann solver) solution, Van Leer method with MUSCL, and Roe method with MUSCL are compared for the shock collision problem at a grid resolution of Δx = 0.0125.

Figure 14 shows the analytical solution (Riemann solver), Van Leer method with MUSCL solution, and Roe method with MUSCL solution to the shock collision problem at a grid resolution of $\Delta x = 0.0125$. The Van Leer method with MUSCL fails after several iterations due to a negative value of energy (and subsequently pressure) in a similar fashion as the same method without MUSCL. The Roe method with MUSCL produces a numerically stable solution, but oscillations are visible throughout the domain (except for the velocity solution, which appears monotonic).

The addition of MUSCL to the Van Leer scheme has resulted in some improvements in solution accuracy, especially in the vicinity of contact surfaces, but has not changed the numerical stability of the method (or lack thereof, for the 123 problem and shock collision problem) for the cases tested. The addition of MUSCL to the Roe scheme has not only introduced oscillations into the solution, but has also caused the solver to fail after several iterations for the 123 problem (for which the non-MUSCL Roe solver was able to produce a stable solution).

Summary and Conclusion

Comparison of the Van Leer method and Roe method for a one-dimensional shock tube suggests that the Roe method is not only slightly more accurate for the relatively benign shock-rarefaction interaction cases (Test 1, 3, and 4), but is also capable of producing numerically stable solutions for rarefaction-rarefaction and shock-shock cases (Test 2 and 5 respectively) if an entropy fix is implemented. Fundamentally, both the Van Leer method and Roe method are upwind schemes, which tend to have high dissipative error – this is observed in the solutions produced by these methods, which display significant peak suppression and contact surface smearing. Grid refinement lowers dissipative error and improves solution accuracy, but contact surfaces still appear smeared even with finer grids and convergence rate for both methods appears to be slightly slower than first-order.

The Van Leer and Roe methods tested here are linear monotonicity-preserving schemes, or total variation diminishing (TVD) schemes, which can at most be first-order accurate according to Godunov's theorem. To extend these schemes such that they are higher-order accurate, variables at the left and right states at cell faces are extrapolated based on nodal values. This extrapolation, termed MUSCL, appears to make the Van Leer scheme more accurate near discontinuities (especially contact surfaces), and increases the convergence rate of the energy variable in particular (previously slower than density, velocity, or pressure, and significantly slower than first-order). The addition of MUSCL to the Roe method introduces oscillations into the solutions and causes the solver to fail for the 123 problem test case, which was previously (for the non-MUSCL Roe solver) numerically stable. The MUSCL scheme uses adjacent nodes to create (using a Taylor series expansion) a polynomial extrapolation for the left and right state flow variables at cell faces. This assumed variation of flow variables within each cell is incompatible with the assumptions made by the Roe method (an approximate Riemann solver), which assumes constant flow variables within a cell.

The preceding project concerning the interaction of a two-dimensional isentropic vortex with a shock was modeled using the MacCormack method and Rusanov method, which suffered from oscillations due to dispersion and suppression of vorticity due to dissipation respectively. The Van Leer scheme with MUSCL might be an appropriate numerical method to apply to such a problem: oscillations would not be a concern given the upwinded nature of the scheme, higher-order spatial accuracy could be achieved, and dissipative error would remain relatively low.

Appendix A – Homework 2 (analytical component)

See following pages for scanned handwritten derivations.

1) $Q = [\rho, \rho u, \rho e_t]^T \qquad F = [\rho u, \rho u^2 + P, \rho u H]^T$

$e_t = e + u^2/2 \qquad H = h + u^2/2 = \underbrace{e + u^2/2}_{e_t} + \frac{P}{\rho}$

Start by manipulating F_2 assuming a calorically perfect gas eq. of state:

$$e = P/[(\gamma-1)\rho]$$

$$F = \begin{bmatrix} \rho u \\ \rho u^2 + P \\ \rho u\left(e_t + \frac{P}{\rho}\right) \end{bmatrix} \qquad \left.\begin{aligned} \rho u &= m \\ \rho e_t &= \epsilon \end{aligned}\right\} \text{ by definition}$$

$$\boxed{F_1 = m}$$

Try to manipulate F_2 so it contains only γ, ϵ, m, and ρ:

$$\rho u^2 + P = \underbrace{\rho u^2 + e(\gamma-1)\rho}_{\text{from EOS}} = \rho u^2 + \left[e_t - \frac{u^2}{2}\right](\gamma-1)\rho$$

$$= \rho u^2 + \underbrace{\rho e_t(\gamma-1)}_{(\gamma-1)\epsilon} - \frac{\rho u^2}{2}(\gamma-1) = (\gamma-1)\epsilon + \rho u^2\left[1 - \frac{(\gamma-1)}{2}\right]$$

$$= (\gamma-1)\epsilon + \rho u^2\left[1 - \frac{\gamma}{2} + \frac{1}{2}\right] = (\gamma-1)\epsilon + \rho u^2\left[\frac{3}{2} - \frac{\gamma}{2}\right]$$

recognize that $m^2 = \rho^2 u^2$, so $\rho u^2 = \frac{m^2}{\rho}$

$$\boxed{F_2 = (\gamma-1)\epsilon + (3-\gamma)m^2/2\rho}$$

Now, manipulate F_3 so that it is only in terms of γ, ϵ, m, and ρ:

→

$$\rho u\left(e_t + \frac{p}{\rho}\right) = \rho u\left(e_t + \underbrace{\frac{(\gamma-1)\rho e}{\rho}}_{\text{from EOS}}\right) = \rho u\left(e_t + (\gamma-1)e\right)$$

$$= \rho u\left[e_t + (\gamma-1)\left(e_t - \frac{u^2}{2}\right)\right] = \rho u\left[\underbrace{e_t + (\gamma-1)e_t}_{e_t + \gamma e_t - e_t} - (\gamma-1)u^2/2\right]$$

$$= \rho u\left[\gamma e_t - (\gamma-1)u^2/2\right] = \underbrace{\gamma \rho e_t u}_{\epsilon} - (\gamma-1)\rho u^3/2$$

recognize that $u = \frac{m}{\rho}$ and $u^3 = \frac{m^3}{\rho^3}$

$$F_2 = \frac{\gamma m \epsilon}{\rho} - (\gamma-1)\frac{\rho m^3}{2\rho^3}$$

$$\boxed{F_2 = \frac{\gamma m \epsilon}{\rho} - (\gamma-1)m^3/2\rho^2}$$

So, our Q and F vectors are now:

$$F = \begin{bmatrix} m \\ (\gamma-1)\epsilon + (3-\gamma)m^2/2\rho \\ \gamma m\epsilon/\rho - (\gamma-1)m^3/2\rho^2 \end{bmatrix}$$

$$Q = \begin{bmatrix} \rho \\ m \\ \epsilon \end{bmatrix}$$

now, calculate $\underset{\sim}{A} = \frac{dF}{dQ}$

$$\underset{\sim}{A} = \frac{dF}{dQ} = \begin{bmatrix} \frac{\partial F_1}{\partial Q_1} & \frac{\partial F_1}{\partial Q_2} & \frac{\partial F_1}{\partial Q_3} \\ \frac{\partial F_2}{\partial Q_1} & \frac{\partial F_2}{\partial Q_2} & \frac{\partial F_2}{\partial Q_3} \\ \frac{\partial F_3}{\partial Q_1} & \frac{\partial F_3}{\partial Q_2} & \frac{\partial F_3}{\partial Q_3} \end{bmatrix}$$

→

$$\frac{\partial F_1}{\partial Q_1} = \frac{\partial m}{\partial \rho} = 0 \qquad \frac{\partial F_1}{\partial Q_2} = \frac{\partial m}{\partial m} = 1$$

$$\frac{\partial F_1}{\partial Q_3} = \frac{\partial m}{\partial \epsilon} = 0 \qquad \frac{\partial F_2}{\partial Q_1} = \frac{\partial}{\partial \rho}\left((\gamma-1)\epsilon + (3-\gamma)m^2/2\rho\right)$$

$$\frac{\partial}{\partial \rho}\left(\frac{1}{\rho}\right) = -\frac{1}{\rho^2} \rightarrow \frac{\partial F_2}{\partial Q_1} = (\gamma-3)m^2/2\rho^2$$

$$\frac{\partial F_2}{\partial Q_2} = \frac{\partial}{\partial m}\left[(\gamma-1)\epsilon + (3-\gamma)m^2/2\rho\right] \rightarrow \frac{\partial F_2}{\partial Q_2} = (3-\gamma)m/\rho$$

$2m$

$$\frac{\partial F_2}{\partial Q_3} = \frac{\partial}{\partial \epsilon}\left[(\gamma-1)\epsilon + (3-\gamma)m^2/2\rho\right] \rightarrow \frac{\partial F_2}{\partial Q_3} = \gamma - 1$$

$$\frac{\partial F_3}{\partial Q_1} = \frac{\partial}{\partial \rho}\left[\frac{\gamma m\epsilon}{\rho} - \frac{(\gamma-1)m^3}{2\rho^2}\right] = -\frac{\gamma m\epsilon}{\rho^2} + \frac{(\gamma-1)m^3}{\rho^3}$$

$$\downarrow$$

$$\frac{\partial}{\partial \rho}(-\rho^{-2}) = 2\rho^{-3}$$

$$\frac{\partial F_3}{\partial Q_1} = \frac{(\gamma-1)m^3}{\rho^3} - \frac{\gamma m\epsilon}{\rho^2} \qquad \frac{\partial F_3}{\partial Q_2} = \frac{\partial}{\partial m}\left[\frac{\gamma m\epsilon}{\rho} - \frac{(\gamma-1)m^3}{2\rho^2}\right]$$

$$\frac{\partial F_3}{\partial Q_2} = \frac{\gamma\epsilon}{\rho} - \frac{3(\gamma-1)m^2}{2\rho^2} \qquad \frac{\partial F_3}{\partial Q_3} = \frac{\partial}{\partial \epsilon}\left[\frac{\gamma m\epsilon}{\rho} - \frac{(\gamma-1)m^3}{2\rho^2}\right]$$

$$\frac{\partial F_3}{\partial Q_3} = \frac{\gamma m}{\rho}$$

Now, we can assemble the $\underset{\sim}{A}$ matrix

$\rightarrow$

$$\underset{\sim}{A} = \begin{bmatrix} 0 & 1 & 0 \\ \frac{(\gamma-3)m^2}{2\rho^2} & (3-\gamma)\frac{m}{\rho} & \gamma-1 \\ \frac{(\gamma-1)m^3}{\rho^3} - \frac{\gamma m \epsilon}{\rho^2} & \frac{\gamma\epsilon}{\rho} - \frac{3(\gamma-1)m^2}{2\rho^2} & \frac{\gamma m}{\rho} \end{bmatrix}$$

Finally, express each entry in $\underset{\sim}{A}$ in terms of u, γ, and $a \left(= \sqrt{\frac{\gamma p}{\rho}}\right)$

$$\underset{\sim}{A}_{21} = \frac{(\gamma-3)m^2}{2\rho^2} = (\gamma-3)u^2/2 \qquad \underset{\sim}{A}_{22} = (3-\gamma)\frac{m}{\rho} = (3-\gamma)u$$

$$A_{31} = \frac{(\gamma-1)m^3}{\rho^3} - \frac{\gamma m \epsilon}{\rho^2} = (\gamma-1)u^3 - \frac{\gamma u \epsilon}{\rho} = (\gamma-1)u^3 - \frac{\gamma u \rho e_t}{\rho}$$

$$= (\gamma-1)u^3 - \gamma u e_t = (\gamma-1)u^3 - \gamma u\left(e + u^2/2\right)$$

$e = \frac{\rho}{(\gamma-1)\rho}$ from EOS

$$= (\gamma-1)u^3 - \frac{\gamma \rho u}{\rho(\gamma-1)} - \frac{\gamma u^3}{2}$$

$\frac{\gamma p}{\rho} = a^2$

$$A_{31} = (\gamma-1)u^3 - \frac{u a^2}{\gamma-1} - \frac{1}{2}\gamma u^3$$

$$A_{32} = \frac{\gamma\epsilon}{\rho} - \frac{3(\gamma-1)m^2}{2\rho^2} = \frac{\gamma\rho(e + u^2/2)}{\rho} - \frac{3(\gamma-1)u^2}{2}$$

$m^2 = \rho^2 u^2$

$$= \frac{\gamma u^2}{2} + \frac{\gamma p}{\rho(\gamma-1)} - \frac{3(\gamma-1)u^2}{2} = \left[\gamma - 3(\gamma-1)\right]\frac{u^2}{2} + \frac{a^2}{\gamma-1}$$

$\frac{\gamma p}{\rho} = a^2$

→

$$A_{32} = \frac{(\gamma - 3\gamma + 3)}{2} u^2 + \frac{a^2}{\gamma - 1}$$

$$\boxed{A_{32} = \left(\frac{3}{2} - \gamma\right) u^2 + \frac{a^2}{\gamma - 1}} \qquad \boxed{A_{33} = \frac{\gamma m}{\rho} = \gamma u}$$

Finally, assemble $\underset{\sim}{A}$ matrix in terms of u, γ, a:

$$\underset{\sim}{A} = \begin{bmatrix} 0 & 1 & 0 \\ \frac{(\gamma-3)u^2}{2} & (3-\gamma)u & \gamma - 1 \\ (\gamma-1)u^3 - \frac{ua^2}{\gamma-1} - \frac{1}{2}\gamma u^3 & \left(\frac{3}{2} - \gamma\right)u^2 + \frac{a^2}{\gamma-1} & \gamma u \end{bmatrix}$$

2) $F \equiv \underset{\sim}{A} Q = \begin{bmatrix} 0 & 1 & 0 \\ (\gamma-3)u^2/2 & (3-\gamma)u & \gamma-1 \\ (\gamma-1)u^3 - \frac{ua^2}{\gamma-1} - \frac{\gamma u^3}{2} & (\frac{3}{2}-\gamma)u^2 + \frac{a^2}{\gamma-1} & \gamma u \end{bmatrix} \begin{bmatrix} \rho \\ m \\ \epsilon \end{bmatrix}$

$$F = \begin{bmatrix} 0\rho + 1m + 0\epsilon \\ (\gamma-3)\rho u^2/2 + (3-\gamma)mu + (\gamma-1)\epsilon \\ \rho(\gamma-1)u^3 - \frac{\rho u a^2}{\gamma-1} - \frac{\gamma\rho u^3}{2} + (\frac{3}{2}-\gamma)mu^2 + \frac{ma^2}{\gamma-1} + \gamma u\epsilon \end{bmatrix}$$

$u = \frac{m}{\rho}$ $\qquad$ $u = \frac{m}{\rho}$

$\boxed{F_1 = m}$ $\qquad F_2 = (\gamma-3)\rho\left(\frac{m}{\rho}\right)^2/2 + (3-\gamma)m\left(\frac{m}{\rho}\right) + (\gamma-1)\epsilon$

$F_2 = (\gamma-3)m^2/2\rho + (3-\gamma)m^2/\rho + (\gamma-1)\epsilon = (\gamma-1)\epsilon + \left[\frac{1}{2}(\gamma-3) + (3-\gamma)\right]m^2/\rho$

$F_2 = (\gamma-1)\epsilon + \left[\frac{\gamma}{2} - \frac{3}{2} + 3 - \gamma\right]m^2/\rho = (\gamma-1)\epsilon + \left[\frac{3}{2} - \frac{\gamma}{2}\right]m^2/\rho$

$\boxed{F_2 = (\gamma-1)\epsilon + (3-\gamma)m^2/2\rho}$

$F_3 = \rho(\gamma-1)\frac{m^3}{\rho^3} - \frac{ma^2}{\gamma-1} - \gamma\rho m^3/2\rho^3 + \left(\frac{3}{2}-\gamma\right)m\frac{m^2}{\rho^2} + \frac{ma^2}{\gamma-1} + \frac{\gamma m\epsilon}{\rho}$

$u^3 = \left(\frac{m}{\rho}\right)^3$ $\qquad$ these cancel

$F_3 = (\gamma-1)m^3/\rho^2 - \gamma m^3/2\rho^2 + (\frac{3}{2}-\gamma)m^3/\rho^2 + \frac{\gamma m\epsilon}{\rho}$

$F_3 = \left[(\gamma-1) - \frac{\gamma}{2} + \left(\frac{3}{2}-\gamma\right)\right]m^3/\rho^2 + \frac{\gamma m\epsilon}{\rho}$

$F_3 = \left[\frac{1}{2} - \frac{\gamma}{2}\right]m^3/\rho^2 + \frac{\gamma m\epsilon}{\rho} = (1-\gamma)m^3/\rho^2 + \frac{\gamma m\epsilon}{\rho}$

→

$$F_3 = \frac{\gamma m \epsilon}{\rho} - \frac{(\gamma-1)m^3}{\rho^2}$$

Now, reassemble F vector:

$$F = \underset{\sim}{A}Q = \begin{bmatrix} m \\ (\gamma-1)\epsilon + (3-\gamma)m^2/2\rho \\ \gamma m \epsilon/\rho - (\gamma-1)m^3/\rho^2 \end{bmatrix}$$

which is exactly eq. (3.16) from the typeset class notes

3) $F_1^{\pm} = \frac{\rho}{2\gamma}\left\{\lambda_1^{\pm} + \lambda_2^{\pm} + 2(\gamma-1)\lambda_3^{\pm}\right\}$

manipulate each entry in $F^{\pm}$ so that it is in terms of ρ, a, M, and γ

$\lambda_1^{\pm} = \frac{(u-a) \pm |u-a|}{2}$

$\lambda_2^{\pm} = \frac{(u+a) \pm |u+a|}{2}$ $\qquad \lambda_3^{\pm} = \frac{u \pm |u|}{2}$

$\lambda_1^{\pm} = \frac{a[(M-1) \pm |M-1|]}{2}$ $\qquad \lambda_2^{\pm} = \frac{a[(M+1) \pm |M+1|]}{2}$

$\lambda_3^{\pm} = \frac{a[M \pm |M|]}{2}$

These substitutions will be used for $F_2^{\pm}$ and $F_3^{\pm}$ as well.

$F_1^{\pm} = \frac{\rho a}{2\gamma}\left\{\frac{(M-1) \pm |M-1|}{2} + \frac{(M+1) \pm |M+1|}{2} + 2(\gamma-1)\left(\frac{M \pm |M|}{2}\right)\right\}$

$= \frac{\rho a}{2\gamma}\left\{\underbrace{\frac{M-1}{2} + \frac{M+1}{2}}_{=M} \pm \left(\frac{|M+1|}{2} + \frac{|M-1|}{2}\right) + \gamma M \pm \gamma|M| - M \mp |M|\right\}$

↳ if $-1 \le M \le 1$, $\frac{|M+1|}{2} + \frac{|M-1|}{2} = 1$

This substitution will appear again later!

$F_1^{\pm} = \frac{\rho a}{2\gamma}\left\{M \pm 1 + \gamma M \pm \gamma|M| - M \mp |M|\right\}$

→ eliminate

→

$$F_1^{\pm} = \frac{\rho a}{2}\left\{ \pm\frac{1}{\gamma} + \underbrace{M \pm |M|}_{\text{good}} \mp \frac{1}{\gamma}|M| \right\}$$

$$\boxed{F_1^{\pm} = \frac{\rho a}{2}\left\{ M \pm |M| \pm \left(1 - |M|\right)\frac{1}{\gamma} \right\}}$$

$$F_2^{\pm} = \frac{\rho}{2\gamma}\left\{ (u-a)\lambda_1^{\pm} + (u+a)\lambda_2^{\pm} + 2(\gamma-1)u\lambda_3^{\pm} \right\}$$

$$= \frac{\rho a}{2\gamma}\left\{ (M-1)\lambda_1^{\pm} + (M+1)\lambda_2^{\pm} + 2(\gamma-1)M\lambda_3^{\pm} \right\}$$

substitute in $\lambda_1^{\pm}, \lambda_2^{\pm}, \lambda_3^{\pm}$ definitions as before

$$= \frac{\rho a}{2\gamma}\left\{ (M-1)a\frac{[(M-1) \pm |M-1|]}{2} + (M+1)a\frac{[(M+1) \pm |M-1|]}{2} \right.$$

$$\left. + 2(\gamma-1)Ma\frac{[M \pm |M|]}{2} \right\}$$

$$= \frac{\rho a^2}{2\gamma}\left\{ \frac{(M-1)^2 \pm (M-1)|M-1|}{2} + \frac{(M+1)^2 \pm (M+1)|M+1|}{2} \right.$$

$$\left. + (\gamma-1)\left[M^2 \pm M|M|\right] \right\}$$

$$= \frac{\rho a^2}{2\gamma}\left\{ \frac{M^2 - 2M + 1}{2} + \frac{M^2 + 2M + 1}{2} \pm \left[\frac{(M+1)|M+1|}{2} + \frac{(M-1)|M-1|}{2}\right] \right.$$

$$\left. + \gamma M^2 \pm \gamma M|M| - M^2 \mp M|M| \right\}$$

$$= \frac{\rho a^2}{2\gamma}\left\{ M^2 + 1 \pm \left[M\left(\frac{|M+1|}{2} + \frac{|M-1|}{2}\right) + \frac{|M+1|}{2} - \frac{|M-1|}{2} \right] \right.$$

$$\left. + \gamma M^2 \pm \gamma M|M| - M^2 \mp M|M| \right\}$$

→

recall that $\frac{|M+1|}{2} + \frac{|M-1|}{2} = 1$ for $-1 \le M \le 1$

also recognize that $\frac{|M+1|}{2} - \frac{|M-1|}{2} = M$ for $-1 \le M \le 1$

make both of these substitutions:

$$F_2^{\pm} = \frac{\rho a^2}{2\gamma}\left\{1 \pm \left[M(1) + M\right] \pm \gamma M|M| \mp M|M| + \gamma M^2\right\}$$

$$= \frac{\rho a^2}{2}\left\{M^2 + \frac{1}{\gamma} \pm \frac{2M}{\gamma} \pm M|M| \mp \frac{M|M|}{\gamma}\right\}$$

$$= \frac{\rho a^2}{2}\left\{M^2 + \frac{1}{\gamma} \pm \left(\frac{2M}{\gamma} + M|M| - \frac{M|M|}{\gamma}\right)\right\}$$

recognize $M|M| = \frac{\gamma}{\gamma} M|M|$, so $\frac{\gamma}{\gamma}M|M| - \frac{M|M|}{\gamma} = M|M|\left(\frac{\gamma}{\gamma} - \frac{1}{\gamma}\right)$

$= M|M|\frac{\gamma-1}{\gamma}$ (a similar substitution is made later)

$$\boxed{F_2^{\pm} = \frac{\rho a^2}{2}\left\{M^2 + \frac{1}{\gamma} \pm M\left(\frac{2}{\gamma} + |M|\frac{\gamma-1}{\gamma}\right)\right\}}$$

$$F_3^{\pm} = \frac{\rho}{2\gamma}\left\{\underbrace{\frac{1}{2}(u-a)^2\lambda_1^{\pm} + \frac{1}{2}(u+a)^2\lambda_2^{\pm}}_{①} + \underbrace{(\gamma-1)u^2\lambda_3^{\pm}}_{②}\right.$$

$$\left. + \underbrace{\frac{(3-\gamma)a^2(\lambda_1^{\pm} + \lambda_2^{\pm})}{2(\gamma-1)}}_{③}\right\}$$

Due to the size of this, it is convenient to group terms and treat each group independently →

①: $\frac{\rho}{2\gamma}\left\{\frac{1}{2}(u-a)^2\lambda_1^{\pm} + \frac{1}{2}(u+a)^2\lambda_2^{\pm}\right\}$

$= \frac{\rho}{2\gamma}\left\{\frac{1}{2}a^2(M-1)^2 a\frac{[(M-1)\pm|M-1|]}{2} + \frac{1}{2}a^2(M+1)^2 a\frac{[(M+1)\pm|M+1|]}{2}\right\}$

$= \frac{\rho a^3}{2\gamma}\left\{\frac{1}{2}\frac{(M-1)^3 \pm (M-1)^2|M-1|}{2} + \frac{1}{2}\frac{(M+1)^3 \pm (M+1)^2|M+1|}{2}\right\}$

expand $(M-1)^3$, $(M+1)^3$, $(M-1)^2$, $(M+1)^2$

$= \frac{\rho a^3}{2\gamma}\left\{\frac{1}{2}\left[\frac{M^3 - 3M^2 + 3M - 1}{2} + \frac{M^3 + 3M^2 + 3M + 1}{2} \pm (M^2 - 2M + 1)\frac{|M-1|}{2}\right.\right.$

$\left.\left. \pm (M^2 + 2M + 1)\frac{|M+1|}{2}\right]\right\}$ eliminate some terms, separate the $2M$

$= \frac{\rho a^3}{2\gamma}\left\{\frac{1}{2}\left[M^3 + 3M \pm \left((M^2+1)\frac{|M-1|}{2} - 2M\frac{|M-1|}{2} + (M^2+1)\frac{|M+1|}{2}\right.\right.\right.$

$\left.\left.\left. + 2M\frac{|M+1|}{2}\right)\right]\right\}$ collect terms on $|M-1|$ and $|M+1|$

$= \frac{\rho a^3}{2\gamma}\left\{\frac{1}{2}\left[M^3 + 3M \pm \left((M^2+1)\underbrace{\left(\frac{|M+1|}{2} + \frac{|M-1|}{2}\right)}_{=1} + 2M\underbrace{\left(\frac{|M+1|}{2} - \frac{|M-1|}{2}\right)}_{=M}\right)\right]\right\}$

make the same substitutions as previously

$= \frac{\rho a^3}{2\gamma}\left\{\frac{1}{2}\left[M^3 + 3M \pm \left(M^2 + 1 + 2M^2\right)\right]\right\}$

$① = \frac{\rho a^3}{2\gamma}\left\{\frac{1}{2}\left[M^3 + 3M \pm (3M^2 + 1)\right]\right\}$

$① = \frac{\rho a^3}{2}\left\{\frac{M^3}{2\gamma} + \frac{3M}{2\gamma} \pm \left(\frac{3M^2}{2\gamma} + \frac{1}{2\gamma}\right)\right\}$ now work on term ②

→

② : $\frac{\rho}{2\gamma}\left\{(\gamma-1)u^2\lambda_3^{\pm}\right\}$

$= \frac{\rho}{2\gamma}\left\{(\gamma-1)M^2a^2\frac{a[M\pm|M|]}{2}\right\} = \frac{\rho a^3}{2\gamma}\left\{(\gamma-1)\frac{M^3\pm M^2|M|}{2}\right\}$

$= \frac{\rho a^3}{2\gamma}\left\{\frac{\gamma M^3}{2} - \frac{M^3}{2} \pm \frac{\gamma M^2|M|}{2} \mp \frac{M^2|M|}{2}\right\}$

$= \frac{\rho a^3}{2\gamma}\left\{\frac{\gamma M^3}{2} - \frac{M^3}{2} \pm \left(\frac{\gamma M^2|M|}{2} - \frac{M^2|M|}{2}\right)\right\}$

$= \frac{\rho a^3}{2}\left\{\frac{M^3}{2} - \frac{M^3}{2\gamma} \pm \left(\frac{M^2|M|}{2} - \frac{M^2|M|}{2\gamma}\right)\right\}$

$\frac{M^2|M|}{2} = \frac{\gamma M^2|M|}{2\gamma}$

$= \frac{\rho a^3}{2}\left\{\frac{M^3}{2} - \frac{M^3}{2\gamma} \pm \left(\frac{\gamma M^2|M|}{2\gamma} - \frac{M^2|M|}{2\gamma}\right)\right\}$

$② = \frac{\rho a^3}{2}\left\{\frac{M^3}{2} - \frac{M^3}{2\gamma} \pm \frac{M^2|M|}{2}\left(\frac{\gamma-1}{\gamma}\right)\right\}$

now, work on term ③

③ : $\frac{\rho}{2\gamma}\left\{\frac{(3-\gamma)a^2}{2(\gamma-1)}(\lambda_1^{\pm}+\lambda_2^{\pm})\right\}$

$\frac{a[(M-1)\pm|M-1|]}{2} + \frac{a[(M+1)\pm|M+1|]}{2} = a\left[\underbrace{\frac{(M-1)}{2}+\frac{(M+1)}{2}}_{=M} \pm \left(\underbrace{\frac{|M+1|}{2}+\frac{|M-1|}{2}}_{=1 \text{ from before}}\right)\right] = a[M\pm 1]$

→

$$③ = \frac{\rho}{2\gamma}\left\{\frac{(3-\gamma)a^3}{2(\gamma-1)}\left[M \pm 1\right]\right\} = \frac{\rho a^3}{2}\left\{\frac{3-\gamma}{2\gamma(\gamma-1)}M \pm \frac{3-\gamma}{2\gamma(\gamma-1)}\right\}$$

now, assemble $F_3^{\pm} = ① + ② + ③$

$$F_3^{\pm} = \frac{\rho a^3}{2}\left\{\cancel{\frac{M^3}{2\gamma}} + \frac{3M}{2\gamma} \pm \left(\frac{3M^2}{2\gamma} + \frac{1}{2\gamma}\right) + \frac{M^3}{2} \cancel{- \frac{M^3}{2\gamma}} \pm \frac{M^2|M|}{2}\left(\frac{\gamma-1}{\gamma}\right)\right.$$

$$\left. + \frac{(3-\gamma)M}{2\gamma(\gamma-1)} \pm \frac{3-\gamma}{2\gamma(\gamma-1)}\right\}$$

eliminate circled terms

collect terms on M^3, $M^2|M|$

$$F_3^{\pm} = \frac{\rho a^3}{2}\left\{\frac{M^3}{2} \pm \frac{M^2|M|}{2}\left(\frac{\gamma-1}{\gamma}\right) + \frac{3M}{2\gamma} \pm \left(\frac{3M^2}{2\gamma} + \frac{1}{2\gamma}\right)\right.$$

$$\left. + \frac{(3-\gamma)M}{2\gamma(\gamma-1)} \pm \frac{3-\gamma}{2\gamma(\gamma-1)}\right\}$$

$$F_3^{\pm} = \frac{\rho a^3}{2}\left\{\underbrace{\frac{M^2}{2}\left(M \pm |M|\frac{\gamma-1}{\gamma}\right)}_{\text{good!}} + \underbrace{\frac{3M(\gamma-1)}{2\gamma(\gamma-1)}}_{\cdot\frac{\gamma-1}{\gamma-1}} \pm \left(\frac{3M^2}{2\gamma} + \underbrace{\frac{\gamma-1}{2\gamma(\gamma-1)}}_{\cdot\frac{\gamma-1}{\gamma-1}}\right)\right.$$

$$\left. + \frac{(3-\gamma)M}{2\gamma(\gamma-1)} \pm \frac{3-\gamma}{2\gamma(\gamma-1)}\right\}$$

add terms circled

$$F_3^{\pm} = \frac{\rho a^3}{2}\left\{\frac{M^2}{2}\left(M \pm |M|\frac{\gamma-1}{\gamma}\right) + \underbrace{\frac{3M\gamma - 3M + 3M - M\gamma}{2\gamma(\gamma-1)}}_{= \frac{2\gamma M}{2\gamma(\gamma-1)}} \pm \left(\frac{3M^2}{2\gamma} + \underbrace{\frac{3-\gamma+\gamma-1}{2\gamma(\gamma-1)}}_{= \frac{2}{2\gamma(\gamma-1)}}\right)\right\}$$

→

Finally, we can make some simplifications

$$F_3^{\pm} = \frac{\rho a^3}{2}\left\{\frac{M^2}{2}\left(M \pm |M|\frac{\gamma-1}{\gamma}\right) + \frac{M}{\gamma-1} \pm \left(\frac{3M^2}{2\gamma} + \frac{1}{\gamma(\gamma-1)}\right)\right\}$$

bring this inside, it becomes $\pm \frac{M}{\gamma-1}$

$$\boxed{F_3^{\pm} = \frac{\rho a^3}{2}\left\{\frac{M^2}{2}\left(M \pm |M|\frac{\gamma-1}{\gamma}\right) \pm \left(\frac{3M^2}{2\gamma} \pm \frac{M}{\gamma-1} + \frac{1}{\gamma(\gamma-1)}\right)\right\}}$$

Now we can assemble $F^{\pm}$!

$$\boxed{F^{\pm} = \begin{bmatrix} \frac{\rho a}{2}\left\{M \pm |M| \pm (1-|M|)\frac{1}{\gamma}\right\} \\ \frac{\rho a^2}{2}\left\{M^2 + \frac{1}{\gamma} \pm M\left(\frac{2}{\gamma} + |M|\frac{\gamma-1}{\gamma}\right)\right\} \\ \frac{\rho a^3}{2}\left\{\frac{M^2}{2}\left(M \pm |M|\frac{\gamma-1}{\gamma}\right) \pm \left(\frac{3M^2}{2\gamma} \pm \frac{M}{\gamma-1} + \frac{1}{\gamma(\gamma-1)}\right)\right\} \end{bmatrix}}$$

4) Starting from

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho e_t \end{bmatrix} \qquad F = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho u H \end{bmatrix} \qquad z = \sqrt{\rho} \begin{bmatrix} 1 \\ u \\ H \end{bmatrix}$$

Write out each Roe parameter vector term:

$$z_1 = \sqrt{\rho} \qquad z_2 = \sqrt{\rho}\, u \qquad z_3 = \sqrt{\rho} H = \sqrt{\rho}\left(e + \frac{p}{\rho} + \frac{u^2}{2}\right)$$

Useful to know e, p in terms of z_1, z_2, z_3, γ

$$z_3 = \sqrt{\rho}\left(e + \underbrace{\frac{p}{\rho}}_{= (\gamma-1)e \text{ from EOS}} + u^2/2\right) = \underbrace{\sqrt{\rho}}_{= z_1}\left(e + (\gamma-1)e + u^2/2\right)$$

$$z_3 = z_1\left(\gamma e + u^2/2\right) \qquad \rho z_3 = z_1\left(\rho\gamma e + \underbrace{\rho u^2/2}_{= z_2^2/2}\right)$$

($\rho = z_1^2$)

$$z_1^2 z_3 = z_1\left(z_1^2 \gamma e + z_2^2/2\right) \qquad z_1 z_3 = z_1^2 \gamma e + z_2^2/2$$

$$z_1 z_3 - \frac{z_2^2}{2} = z_1^2 \gamma e \quad \rightarrow \quad \boxed{e = \frac{z_3}{z_1 \gamma} - \frac{z_2^2}{2 z_1^2 \gamma}} \quad (1)$$

from EOS, $p = e(\gamma-1)\rho =$

$$p = (\gamma-1) z_1^2 \left[\frac{z_3}{z_1 \gamma} - \frac{z_2^2}{2 z_1^2 \gamma}\right] = (\gamma-1)\left[\frac{z_1 z_3}{\gamma} - \frac{z_2^2}{2\gamma}\right]$$

$$\boxed{p = z_1 z_3 - \frac{z_2^2}{2} - \frac{z_1 z_3}{\gamma} + \frac{z_2^2}{2\gamma}} \quad (2) \qquad \rightarrow$$

Now, consider each element of Q and F:

$$Q_1 = \rho = \sqrt{\rho}^2 \rightarrow \boxed{Q_1 = z_1^2}$$

$$Q_2 = \rho u = \sqrt{\rho}\sqrt{\rho}\,u \rightarrow \boxed{Q_2 = z_1 z_2}$$

$$Q_3 = \rho e_t = \rho(e + u^2/2) = \underbrace{\rho e}_{z_1^2} + \underbrace{\rho u^2/2}_{(\sqrt{\rho}u)^2/2}$$

→ substitute ①

$$Q_3 = z_1^2\left[\frac{z_3}{z_1\gamma} + \frac{z_2^2}{2z_1^2\gamma}\right] + \frac{z_2^2}{2}\cdot\frac{\gamma}{\gamma}$$

$$Q_3 = \frac{z_1 z_3}{\gamma} + \frac{z_2^2}{2\gamma} - \frac{z_2^2}{2\gamma} \rightarrow \boxed{Q_3 = \frac{z_1 z_3}{\gamma} + \frac{(\gamma-1)}{2\gamma} z_2^2}$$

$$\boxed{F_1 = \rho u = z_1 z_2}$$

$$F_2 = \underbrace{\rho u^2}_{(\sqrt{\rho}u)^2} + \underbrace{p}_{\text{substitute in ②}} = z_2^2 + z_1 z_3 - \frac{z_2^2}{2} - \frac{z_1 z_3}{\gamma} + \frac{z_2^2}{2\gamma}$$

Group z_2^2, $z_1 z_3$ terms:

$$F_2 = z_2^2\left(1 - \frac{1}{2} + \frac{1}{2\gamma}\right) + z_1 z_3\left(1 - \frac{1}{\gamma}\right)$$

$$= z_2^2\left(\frac{1}{2}\cdot\frac{\gamma}{\gamma} + \frac{1}{2\gamma}\right) + z_1 z_3\left(1\cdot\frac{\gamma}{\gamma} - \frac{1}{\gamma}\right)$$

$$\boxed{F_2 = \left(\frac{\gamma-1}{\gamma}\right) z_1 z_3 + \left(\frac{\gamma+1}{2\gamma}\right) z_2^2}$$

→

$$F_3 = \rho u H = \underbrace{\sqrt{\rho}\,u}_{z_2}\,\underbrace{\sqrt{\rho}\,H}_{z_3} \rightarrow \boxed{F_3 = z_2 z_3}$$

Now, assemble Q and F in terms of the z Roe parameter vector:

$$Q = \begin{bmatrix} z_1^2 \\ z_1 z_2 \\ \frac{z_1 z_3}{\gamma} + \frac{(\gamma-1)}{2\gamma} z_2^2 \end{bmatrix}$$

$$F = \begin{bmatrix} z_1 z_2 \\ \left(\frac{\gamma-1}{\gamma}\right) z_1 z_3 + \left(\frac{\gamma+1}{2\gamma}\right) z_2^2 \\ z_2 z_3 \end{bmatrix}$$

5) First, it is important to note that Q and F are quadratic in z, which means that the jump in states at $i+\frac{1}{2}$ are exactly equal to Δz times the Jacobian ($\frac{\partial Q}{\partial z}$ and $\frac{\partial F}{\partial z}$ respectively) evaluated at the arithmetic average of z (denoted as $\bar{z}$) using z_L (left state) and z_R (right state).

$$\underbrace{\Delta Q}_{\text{jump in Q}} = \underbrace{\left.\frac{\partial Q}{\partial z}\right|_{\bar{z}}}_{\text{Jacobian}} \Delta z = \underset{\sim}{B}\Delta z \qquad \Delta z = \begin{bmatrix} \Delta z_1 \\ \Delta z_2 \\ \Delta z_3 \end{bmatrix}$$

$$\underbrace{\Delta F}_{\text{jump in F}} = \left.\frac{\partial F}{\partial z}\right|_{\bar{z}} \Delta z = \underset{\sim}{C}\Delta z$$

where $\bar{z} = \frac{1}{2}(z_L + z_R)$

$$\frac{\partial Q}{\partial z} = \begin{bmatrix} \frac{\partial Q_1}{\partial z_1} & \frac{\partial Q_1}{\partial z_2} & \frac{\partial Q}{\partial z_1} \\ \frac{\partial Q_2}{\partial z_1} & \frac{\partial Q_2}{\partial z_2} & \frac{\partial Q_2}{\partial z_3} \\ \frac{\partial Q_3}{\partial z_1} & \frac{\partial Q_3}{\partial z_2} & \frac{\partial Q_3}{\partial z_3} \end{bmatrix}$$

$$\frac{\partial Q_1}{\partial z_1} = \frac{\partial}{\partial z_1}\left(z_1^2\right) = 2z_1 \qquad \frac{\partial Q_1}{\partial z_2} = \frac{\partial}{\partial z_2}\left(z_1^2\right) = 0$$

→

$$\frac{\partial Q_1}{\partial z_3} = \frac{\partial}{\partial z_3}\left(z_1^2\right) = 0 \qquad \frac{\partial Q_2}{\partial z_1} = \frac{\partial}{\partial z_1}\left(z_1 z_2\right) = z_2$$

$$\frac{\partial Q_2}{\partial z_2} = \frac{\partial}{\partial z_2}\left(z_1 z_2\right) = z_1 \qquad \frac{\partial Q_2}{\partial z_3} = \frac{\partial}{\partial z_3}\left(z_1 z_2\right) = 0$$

$$\frac{\partial Q_3}{\partial z_1} = \frac{\partial}{\partial z_1}\left(\frac{z_1 z_3}{\gamma} + \frac{(\gamma-1)}{2\gamma} z_2^2\right) = \frac{z_3}{\gamma}$$

$$\frac{\partial Q_3}{\partial z_2} = \frac{\partial}{\partial z_2}\left(\frac{z_1 z_3}{\gamma} + \frac{(\gamma-1)}{2\gamma} z_2^2\right) = \frac{(\gamma-1)}{\gamma} z_2$$

$$\frac{\partial Q_3}{\partial z_3} = \frac{\partial}{\partial z_3}\left(\frac{z_1 z_3}{\gamma} + \frac{(\gamma-1)}{2\gamma} z_2^2\right) = \frac{z_1}{\gamma}$$

Now, assemble $\frac{\partial Q}{\partial z}$

$$\frac{\partial Q}{\partial z} = \begin{bmatrix} 2z_1 & 0 & 0 \\ z_2 & z_1 & 0 \\ z_3/\gamma & (\gamma-1)z_2/\gamma & z_1/\gamma \end{bmatrix}$$

evaluate $\frac{\partial Q}{\partial z}$ at $\bar{z}$

$$z_1 = \bar{z}_1 = \frac{1}{2}\left(z_{1L} + z_{1R}\right) \qquad z_3 = \bar{z}_3 = \frac{1}{2}\left(z_{3L} + z_{3R}\right)$$

$$z_2 = \bar{z}_2 = \frac{1}{2}\left(z_{2L} + z_{2R}\right)$$

$$\underset{\sim}{B} = \left.\frac{\partial Q}{\partial z}\right|_{\bar{z}} = \begin{bmatrix} 2\bar{z}_1 & 0 & 0 \\ \bar{z}_2 & \bar{z}_1 & 0 \\ \bar{z}_3/\gamma & (\gamma-1)\bar{z}_2/\gamma & \bar{z}_1/\gamma \end{bmatrix}$$

Now, calculate $\frac{\partial f}{\partial z}$ and $\underset{\sim}{C} = \left.\frac{\partial f}{\partial z}\right|_{\bar{z}}$ →

$$\frac{\partial F}{\partial z} = \begin{bmatrix} \frac{\partial F_1}{\partial z_1} & \frac{\partial F_1}{\partial z_2} & \frac{\partial F_1}{\partial z_3} \\ \frac{\partial F_2}{\partial z_1} & \frac{\partial F_2}{\partial z_2} & \frac{\partial F_2}{\partial z_3} \\ \frac{\partial F_3}{\partial z_1} & \frac{\partial F_3}{\partial z_2} & \frac{\partial F_3}{\partial z_3} \end{bmatrix}$$

$$\frac{\partial F_1}{\partial z_1} = \frac{\partial}{\partial z_1}\left(z_1 z_2\right) = z_2 \qquad \frac{\partial F_1}{\partial z_2} = z_1$$

$$\frac{\partial F_1}{\partial z_3} = 0 \qquad \frac{\partial F_2}{\partial z_1} = \frac{\partial}{\partial z_1}\left(\frac{(\gamma-1)}{\gamma} z_1 z_3 + \frac{(\gamma+1)}{2\gamma} z_2^2\right) = \frac{(\gamma-1)}{\gamma} z_3$$

$$\frac{\partial F_2}{\partial z_2} = \frac{(\gamma+1)}{\gamma} z_2 \qquad \frac{\partial F_2}{\partial z_3} = \frac{(\gamma-1)}{\gamma} z_1 \qquad \frac{\partial F_3}{\partial z_1} = \frac{\partial}{\partial z_1}\left(z_2 z_3\right) = 0$$

$$\frac{\partial F_3}{\partial z_2} = z_3 \qquad \frac{\partial F_3}{\partial z_3} = z_2$$

Assemble $\frac{\partial F}{\partial z}$:

$$\frac{\partial F}{\partial z} = \begin{bmatrix} z_2 & z_1 & 0 \\ \frac{(\gamma-1)}{\gamma} z_3 & \frac{(\gamma+1)}{\gamma} z_2 & \frac{(\gamma-1)}{\gamma} z_1 \\ 0 & z_3 & z_2 \end{bmatrix}$$

evaluate at $\bar{z}$

$$\underset{\sim}{C} = \left.\frac{\partial F}{\partial z}\right|_{\bar{z}} = \begin{bmatrix} \bar{z}_2 & \bar{z}_1 & 0 \\ \frac{(\gamma-1)}{\gamma}\bar{z}_3 & \frac{(\gamma+1)}{\gamma}\bar{z}_2 & \frac{(\gamma-1)}{\gamma}\bar{z}_1 \\ 0 & \bar{z}_3 & \bar{z}_2 \end{bmatrix}$$

→

Finally, assemble full expressions for jumps in Q and F at interface:

$$\Delta Q = \underset{\sim}{B}\Delta z = \begin{bmatrix} 2\bar{z}_1 & 0 & 0 \\ \bar{z}_2 & \bar{z}_1 & 0 \\ \frac{\bar{z}_3}{\gamma} & \frac{(\gamma-1)}{\gamma}\bar{z}_2 & \frac{\bar{z}_1}{\gamma} \end{bmatrix} \begin{bmatrix} \Delta z_1 \\ \Delta z_2 \\ \Delta z_3 \end{bmatrix}$$

$$\Delta F = \underset{\sim}{C}\Delta z = \begin{bmatrix} \bar{z}_2 & \bar{z}_1 & 0 \\ \frac{(\gamma-1)}{\gamma}\bar{z}_3 & \frac{(\gamma+1)}{\gamma}\bar{z}_2 & \frac{(\gamma-1)}{\gamma}\bar{z}_1 \\ 0 & \bar{z}_3 & \bar{z}_2 \end{bmatrix} \begin{bmatrix} \Delta z_1 \\ \Delta z_2 \\ \Delta z_3 \end{bmatrix}$$

Appendix B – Python code

Van Leer 1D

```
def vanleer1D(test,c_max,dx):
    import numpy as np

    if test == 1:
        # case 1 - Sod problem
        rhoL=1.0
        uL=0.0
        pL=1.0
        rhoR=0.125
        uR=0.0
        pR=0.1
        t_final = 0.25
    if test == 2:
        # case 2 - 123 problem - expansion left and expansion right
        rhoL=1.0
        uL=-2.0
        pL=0.4
        rhoR=1.0
        uR=2.0
        pR=0.4
        t_final = 0.15
    if test == 3:
        # case 3 - blast problem - shock right, expansion left
        rhoL=1.0
        uL=0.0
        pL=1000
        rhoR=1.0
        uR=0.
        pR=0.01
        t_final = 0.012
    if test == 4:
        # case 4 - blast problem - shock left, expansion right
        rhoL=1.0
        uL=0.0
        pL=0.01
        rhoR=1.0
        uR=0.
        pR=100
        t_final = 0.035
    if test == 5:
        # case 5 - shock collision - shock left and shock right
        rhoL=5.99924
        uL=19.5975
        pL=460.894
        rhoR=5.99242
        uR=-6.19633
        pR=46.0950
        t_final = 0.035

    L = 1.0
    gamma = 1.4
    ix = int(L/dx+1)
    half = int(np.floor(ix/2))
    x = np.linspace(0,L,ix)

    # storage vectors
    rho = np.zeros(ix)
    u = np.zeros(ix)
    p = np.zeros(ix)
    e = np.zeros(ix)
    Q = np.zeros((3,ix))
    F_plus = np.zeros((3,ix))
    F_minus = np.zeros((3,ix))
    F_half = np.zeros((3,ix))

    #establish initial condition vectors
    rho[0:half] = rhoL
    rho[half:ix] = rhoR
    u[0:half] = uL
    u[half:ix] = uR
    p[0:half] = pL
    p[half:ix] = pR
    e = p/((gamma-1)*rho)
    a = np.sqrt((gamma*p/rho))
    M = u/a

    t = 0
    dt = c_max*dx/np.max(np.abs(u)+a)

    #calculate initial Q and F vectors
    Q[0,:] = rho
    Q[1,:] = rho*u
    Q[2,:] = rho*(e+u**2/2)

    while t <= t_final:
        for i in range(0,ix): # Loop for calculating Fn

            im=i-1
            ip=i+1

            # Neumann BCs implemented here using a series of if
statements
            if i == 0:
                im = 1
            if i == ix-1:
                ip = ix-2

            F_plus[0,i] = ((1/4)*rho[i]*a[i]*(1+M[i])**2)
            F_plus[1,i] =
((1/4)*rho[i]*a[i]*(1+M[i])**2)*(2*a[i]/gamma)*(((gamma-
1)/2)*M[i]+1)
            F_plus[2,i] =
((1/4)*rho[i]*a[i]*(1+M[i])**2)*(2*a[i]**2/(gamma**2-
1))*(((gamma-1)/2)*M[i]+1)**2

            F_minus[0,i] = -((1/4)*rho[ip]*a[ip]*(1-M[ip])**2)
            F_minus[1,i] = -((1/4)*rho[ip]*a[ip]*(1-
M[ip])**2)*(2*a[ip]/gamma)*(((gamma-1)/2)*M[ip]-1)
            F_minus[2,i] = -((1/4)*rho[ip]*a[ip]*(1-
M[ip])**2)*(2*a[ip]**2/(gamma**2-1))*(((gamma-1)/2)*M[ip]-
1)**2

            F_half[:,i] = F_plus[:,i] + F_minus[:,i]

        for i in range(0,ix): # Q_update
            im=i-1
            ip=i+1

            # Neumann BCs implemented here using a series of if
statements
            if i == 0:
                im = 1
            if i == ix-1:
                ip = ix-2

            Q[:,i] = Q[:,i]-(dt/dx)*(F_half[:,i]-F_half[:,im])
```

```
    rho = Q[0,:]
    u = Q[1,:]/rho
    e = Q[2,:]/rho-u**2/2
    p = e*(gamma-1)*rho
    a = np.sqrt((gamma*p/rho))
    M = u/a

    dt = c_max*dx/np.max(np.abs(u)+a)
    t = t+dt

    print(t)

  return rho, u, p, e, x
```

Roe 1D

```
def roe1D(test,c_max,dx):
  import numpy as np

  def roeavg(betaL,betaR,rhoL,rhoR):
    beta_tilde =
(betaL*np.sqrt(rhoL)+betaR*np.sqrt(rhoR))/(np.sqrt(rhoL)+np.sqrt
(rhoR))
    return beta_tilde

  if test == 1:
    # case 1 - Sod problem
    rhoL=1.0
    uL=0.0
    pL=1.0
    rhoR=0.125
    uR=0.0
    pR=0.1
    t_final = 0.25
  if test == 2:
    # case 2 - 123 problem - expansion left and expansion right
    rhoL=1.0
    uL=-2.0
    pL=0.4
    rhoR=1.0
    uR=2.0
    pR=0.4
    t_final = 0.15
  if test == 3:
    # case 3 - blast problem - shock right, expansion left
    rhoL=1.0
    uL=0.0
    pL=1000
    rhoR=1.0
    uR=0.
    pR=0.01
    t_final = 0.012
  if test == 4:
    # case 4 - blast problem - shock left, expansion right
    rhoL=1.0
    uL=0.0
    pL=0.01
    rhoR=1.0
    uR=0.
    pR=100
    t_final = 0.035
  if test == 5:
    # case 5 - shock collision - shock left and shock right
    rhoL=5.99924
    uL=19.5975
    pL=460.894
    rhoR=5.99242
    uR=-6.19633
    pR=46.0950
    t_final = 0.035

  L = 1.0
  gamma = 1.4
  ix = int(L/dx+1)
  half = int(np.floor(ix/2))
  x = np.linspace(0,L,ix)

  # storage vectors
  rho = np.zeros(ix)
  u = np.zeros(ix)
  p = np.zeros(ix)
  e = np.zeros(ix)
  Q = np.zeros((3,ix))
  F = np.zeros((3,ix))
  F_half = np.zeros((3,ix))

  #establish initial condition vectors
  rho[0:half] = rhoL
  rho[half:ix] = rhoR
  u[0:half] = uL
  u[half:ix] = uR
  p[0:half] = pL
  p[half:ix] = pR
  e = p/((gamma-1)*rho)
  H = e+p/rho+u**2/2
  a = np.sqrt((gamma*p/rho))

  t = 0
  dt = c_max*dx/np.max(np.abs(u)+a)

  #calculate initial Q and F vectors
  Q[0,:] = rho
  Q[1,:] = rho*u
  Q[2,:] = rho*(e+u**2/2)

  F[0,:] = rho*u
  F[1,:] = rho*u**2+p
  F[2,:] = rho*u*(e+p/rho+u**2/2)

  while t <= t_final:
    for i in range(0,ix): # Loop for calculating Fn

      im=i-1
      ip=i+1

      # Neumann BCs implemented here using a series of if
statements
      if i == 0:
        im = 1
      if i == ix-1:
        ip = ix-2

      rho_tilde = np.sqrt(rho[i]*rho[ip])
      u_tilde = roeavg(u[i],u[ip],rho[i],rho[ip])
      a_tilde = roeavg(a[i],a[ip],rho[i],rho[ip])
      H_tilde = roeavg(H[i],H[ip],rho[i],rho[ip])

      del_p = p[ip]-p[i]
      del_u = u[ip]-u[i]
      del_rho = rho[ip]-rho[i]
```

```
        alpha1_tilde = (del_p-
a_tilde*rho_tilde*del_u)/(2*a_tilde**2)
        alpha2_tilde =
(del_p+a_tilde*rho_tilde*del_u)/(2*a_tilde**2)
        alpha3_tilde = del_rho - del_p/a_tilde**2

        lambda1 = np.abs(u_tilde-a_tilde)
        lambda2 = np.abs(u_tilde+a_tilde)
        lambda3 = np.abs(u_tilde)

        eps1 = max(0,((u_tilde-a_tilde)-(u[i]-a[i])),((u[ip]-a[ip])-
(u_tilde-a_tilde)))
        eps2 = max(0,((u_tilde+a_tilde)-(u[i]+a[i])),((u[ip]+a[ip])-
(u_tilde+a_tilde)))

        if eps1 > lambda1:
            lambda1 = eps1
        if eps2 > lambda2:
            lambda2 = eps2

        F_half[0,i] = 0.5*(F[0,i]+F[0,ip]) -
0.5*(alpha1_tilde*lambda1+alpha2_tilde*lambda2+alpha3_tilde*
lambda3)
        F_half[1,i] = 0.5*(F[1,i]+F[1,ip]) -
0.5*(alpha1_tilde*lambda1*(u_tilde-
a_tilde)+alpha2_tilde*lambda2*(u_tilde+a_tilde)+alpha3_tilde*la
mbda3*u_tilde)
        F_half[2,i] = 0.5*(F[2,i]+F[2,ip]) -
0.5*(alpha1_tilde*lambda1*(H_tilde-
u_tilde*a_tilde)+alpha2_tilde*lambda2*(H_tilde+u_tilde*a_tilde)
+alpha3_tilde*lambda3*(u_tilde**2/2))

    for i in range(0,ix): # Q_update
        im=i-1
        ip=i+1

        # Neumann BCs implemented here using a series of if
statements
        if i == 0:
            im = 1
        if i == ix-1:
            ip = ix-2

        Q[:,i] = Q[:,i]-(dt/dx)*(F_half[:,i]-F_half[:,im])

    rho = Q[0,:]
    u = Q[1,:]/rho
    e = Q[2,:]/rho-u**2/2
    p = e*(gamma-1)*rho
    H = e+p/rho+u**2/2
    a = np.sqrt((gamma*p/rho))

    F[0,:] = rho*u
    F[1,:] = rho*u**2+p
    F[2,:] = rho*u*(e+p/rho+u**2/2)

    dt = c_max*dx/np.max(np.abs(u)+a)
    t = t+dt

    print(t)

  return rho, u, p, e, x
```

Van Leer 1D with MUSCL

```
def vanleer1DMUSCL(test,c_max,dx):
import numpy as np

if test == 1:
  # case 1 - Sod problem
  rhoL=1.0
  uL=0.0
  pL=1.0
  rhoR=0.125
  uR=0.0
  pR=0.1
  t_final = 0.25
if test == 2:
  # case 2 - 123 problem - expansion left and expansion right
  rhoL=1.0
  uL=-2.0
  pL=0.4
  rhoR=1.0
  uR=2.0
  pR=0.4
  t_final = 0.15
if test == 3:
  # case 3 - blast problem - shock right, expansion left
  rhoL=1.0
  uL=0.0
  pL=1000
  rhoR=1.0
  uR=0.
  pR=0.01
  t_final = 0.012
if test == 4:
  # case 4 - blast problem - shock left, expansion right
  rhoL=1.0
  uL=0.0
  pL=0.01
  rhoR=1.0
  uR=0.
  pR=100
  t_final = 0.035
if test == 5:
  # case 5 - shock collision - shock left and shock right
  rhoL=5.99924
  uL=19.5975
  pL=460.894
  rhoR=5.99242
  uR=-6.19633
  pR=46.0950
  t_final = 0.035

L = 1.0
gamma = 1.4
ix = int(L/dx+1)
half = int(np.floor(ix/2))
x = np.linspace(0,L,ix)

# storage vectors
rho = np.zeros(ix)
u = np.zeros(ix)
p = np.zeros(ix)
e = np.zeros(ix)
Q = np.zeros((3,ix))
F_plus = np.zeros((3,ix))
F_minus = np.zeros((3,ix))
F_half = np.zeros((3,ix))

#establish initial condition vectors
rho[0:half] = rhoL
```

```
rho[half:ix] = rhoR
u[0:half] = uL
u[half:ix] = uR
p[0:half] = pL
p[half:ix] = pR
e = p/((gamma-1)*rho)
a = np.sqrt((gamma*p/rho))

t = 0
dt = c_max*dx/np.max(np.abs(u)+a)

#calculate initial Q and F vectors
Q[0,:] = rho
Q[1,:] = rho*u
Q[2,:] = rho*(e+u**2/2)

rhoL = np.zeros(ix)
rhoR = np.zeros(ix)
uL = np.zeros(ix)
uR = np.zeros(ix)
pL = np.zeros(ix)
pR = np.zeros(ix)

rhoL[:] = rho
rhoR[:] = rho
uL[:] = u
uR[:] = u
pL[:] = p
pR[:] = p

def MUSCL(vm1,v,vp1,vp2):
   kappa = -1
   rmp12 = (v-vm1+1.e-30)/(vp1-v+1.e-30)
   rmp32 = (vp1-v+1.e-30)/(vp2-vp1+1.e-30)
   phi_rmp12 =  (np.abs(rmp12)+rmp12)/(1+np.abs(rmp12))
   phi_rmp32 = (np.abs(rmp32)+rmp32)/(1+np.abs(rmp32))
   phi_rpp12 = (1/rmp32)*phi_rmp32
   phi_rpm12 = (1/rmp12)*phi_rmp12
   del_pm12 = (v-vm1)*phi_rpm12/2
   del_mp12 = (vp1-v)*phi_rmp12/2
   del_mp32 = (vp2-vp1)*phi_rmp32/2
   del_pp12 = (vp1-v)*phi_rpp12
   del_L = (1-kappa)*del_pm12+(1+kappa)*del_mp12
   del_R = (1-kappa)*del_mp32+(1+kappa)*del_pp12
   vL12 = v+0.5*del_L
   vR12 = vp1-0.5*del_R
   return (vL12,vR12)

while t <= t_final:
   for i in range(1,ix-2): # Loop for calculating Fn

      im=i-1
      ip=i+1
      ip2=i+2

      rhoL[i],rhoR[i] = MUSCL(rho[im],rho[i],rho[ip],rho[ip2])
      uL[i],uR[i] = MUSCL(u[im],u[i],u[ip],u[ip2])
      pL[i],pR[i] = MUSCL(p[im],p[i],p[ip],p[ip2])

   aL = np.sqrt(gamma*pL/rhoL)
   aR = np.sqrt(gamma*pR/rhoR)
   ML = uL/aL
   MR = uR/aR

   for i in range(0,ix): # Loop for calculating Fn
         im=i-1
         ip=i+1

         # Neumann BCs implemented here using a series of if
statements
         if i == 0:
            im = 1
         if i == ix-1:
            ip = ix-2

         F_plus[0,i] = ((1/4)*rhoL[i]*aL[i]*(1+ML[i])**2)
         F_plus[1,i] =
((1/4)*rhoL[i]*aL[i]*(1+ML[i])**2)*(2*aL[i]/gamma)*(((gamma-
1)/2)*ML[i]+1)
         F_plus[2,i] =
((1/4)*rhoL[i]*aL[i]*(1+ML[i])**2)*(2*aL[i]**2/(gamma**2-
1))*(((gamma-1)/2)*ML[i]+1)**2

         F_minus[0,i] = -((1/4)*rhoR[i]*aR[i]*(1-MR[i])**2)
         F_minus[1,i] = -((1/4)*rhoR[i]*aR[i]*(1-
MR[i])**2)*(2*aR[i]/gamma)*(((gamma-1)/2)*MR[i]-1)
         F_minus[2,i] = -((1/4)*rhoR[i]*aR[i]*(1-
MR[i])**2)*(2*aR[i]**2/(gamma**2-1))*(((gamma-1)/2)*MR[i]-
1)**2

         F_half[:,i] = F_plus[:,i] + F_minus[:,i]

   for i in range(0,ix): # Q_update
      im=i-1
      ip=i+1

      # Neumann BCs implemented here using a series of if
statements
      if i == 0:
         im = 1
      if i == ix-1:
         ip = ix-2

      Q[:,i] = Q[:,i]-(dt/dx)*(F_half[:,i]-F_half[:,im])

   rho = Q[0,:]
   u = Q[1,:]/rho
   e = Q[2,:]/rho-u**2/2
   p = e*(gamma-1)*rho
   a = np.sqrt((gamma*p/rho))
   dt = c_max*dx/np.max(np.abs(u)+a)
   t = t+dt

   print(t)

return rho, u, p, e, x
```

Roe 1D with MUSCL

```
def roe1DMUSCL(test,c_max,dx):
   import numpy as np

   def roeavg(betaL,betaR,rhoL,rhoR):
      beta_tilde =
(betaL*np.sqrt(rhoL)+betaR*np.sqrt(rhoR))/(np.sqrt(rhoL)+np.sqrt
(rhoR))
      return beta_tilde

   def MUSCL(vm1,v,vp1,vp2):
      kappa = -1
      rmp12 = (v-vm1+1.e-30)/(vp1-v+1.e-30)
      rmp32 = (vp1-v+1.e-30)/(vp2-vp1+1.e-30)
```

```
    phi_rmp12 =  (np.abs(rmp12)+rmp12)/(1+np.abs(rmp12))
    phi_rmp32 = (np.abs(rmp32)+rmp32)/(1+np.abs(rmp32))
    phi_rpp12 = (1/rmp32)*phi_rmp32
    phi_rpm12 = (1/rmp12)*phi_rmp12
    del_pm12 = (v-vm1)*phi_rpm12/2
    del_mp12 = (vp1-v)*phi_rmp12/2
    del_mp32 = (vp2-vp1)*phi_rmp32/2
    del_pp12 = (vp1-v)*phi_rpp12
    del_L = (1-kappa)*del_pm12+(1+kappa)*del_mp12
    del_R = (1-kappa)*del_mp32+(1+kappa)*del_pp12
    vL12 = v+0.5*del_L
    vR12 = vp1-0.5*del_R
    return (vL12,vR12)

if test == 1:
    # case 1 - Sod problem
    rhoL=1.0
    uL=0.0
    pL=1.0
    rhoR=0.125
    uR=0.0
    pR=0.1
    t_final = 0.25
if test == 2:
    # case 2 - 123 problem - expansion left and expansion right
    rhoL=1.0
    uL=-2.0
    pL=0.4
    rhoR=1.0
    uR=2.0
    pR=0.4
    t_final = 0.15
if test == 3:
    # case 3 - blast problem - shock right, expansion left
    rhoL=1.0
    uL=0.0
    pL=1000
    rhoR=1.0
    uR=0.
    pR=0.01
    t_final = 0.012
if test == 4:
    # case 4 - blast problem - shock left, expansion right
    rhoL=1.0
    uL=0.0
    pL=0.01
    rhoR=1.0
    uR=0.
    pR=100
    t_final = 0.035
if test == 5:
    # case 5 - shock collision - shock left and shock right
    rhoL=5.99924
    uL=19.5975
    pL=460.894
    rhoR=5.99242
    uR=-6.19633
    pR=46.0950
    t_final = 0.035

L = 1.0
gamma = 1.4
ix = int(L/dx+1)
half = int(np.floor(ix/2))
x = np.linspace(0,L,ix)

# storage vectors
rho = np.zeros(ix)
u = np.zeros(ix)
p = np.zeros(ix)
e = np.zeros(ix)
Q = np.zeros((3,ix))
F = np.zeros((3,ix))
F_half = np.zeros((3,ix))

#establish initial condition vectors
rho[0:half] = rhoL
rho[half:ix] = rhoR
u[0:half] = uL
u[half:ix] = uR
p[0:half] = pL
p[half:ix] = pR
e = p/((gamma-1)*rho)
H = e+p/rho+u**2/2
a = np.sqrt((gamma*p/rho))

t = 0
dt = c_max*dx/np.max(np.abs(u)+a)

#calculate initial Q and F vectors
Q[0,:] = rho
Q[1,:] = rho*u
Q[2,:] = rho*(e+u**2/2)

F[0,:] = rho*u
F[1,:] = rho*u**2+p
F[2,:] = rho*u*(e+p/rho+u**2/2)

rhoL = np.zeros(ix)
rhoR = np.zeros(ix)
uL = np.zeros(ix)
uR = np.zeros(ix)
pL = np.zeros(ix)
pR = np.zeros(ix)
eL = np.zeros(ix)
eR = np.zeros(ix)

rhoL[:] = rho
rhoR[:] = rho
uL[:] = u
uR[:] = u
pL[:] = p
pR[:] = p
eL[:] = e
eR[:] = e

while t <= t_final:
    for i in range(1,ix-2): # Loop for calculating Fn

        im=i-1
        ip=i+1
        ip2=i+2

        rhoL[i],rhoR[i] = MUSCL(rho[im],rho[i],rho[ip],rho[ip2])
        uL[i],uR[i] = MUSCL(u[im],u[i],u[ip],u[ip2])
        pL[i],pR[i] = MUSCL(p[im],p[i],p[ip],p[ip2])
        eL[i],eR[i] = MUSCL(e[im],e[i],e[ip],e[ip2])

    aL = np.sqrt(gamma*pL/rhoL)
    aR = np.sqrt(gamma*pR/rhoR)
    HL = eL+pL/rhoL+uL**2/2
    HR = eR+pR/rhoR+uR**2/2
```

```
    for i in range(0,ix): # Loop for calculating Fn

        im=i-1
        ip=i+1

        # Neumann BCs implemented here using a series of if
statements
        if i == 0:
            im = 1
        if i == ix-1:
            ip = ix-2

        rho_tilde = np.sqrt(rhoL[i]*rhoR[i])
        u_tilde = roeavg(uL[i],uR[i],rhoL[i],rhoR[i])
        a_tilde = roeavg(aL[i],aR[i],rhoL[i],rhoR[i])
        H_tilde = roeavg(HL[i],HR[i],rhoL[i],rhoR[i])

        del_p = pR[i]-pL[i]
        del_u = uR[i]-uL[i]
        del_rho = rhoR[i]-rhoL[i]

        alpha1_tilde = (del_p-
a_tilde*rho_tilde*del_u)/(2*a_tilde**2)
        alpha2_tilde =
(del_p+a_tilde*rho_tilde*del_u)/(2*a_tilde**2)
        alpha3_tilde = del_rho - del_p/a_tilde**2

        lambda1 = np.abs(u_tilde-a_tilde)
        lambda2 = np.abs(u_tilde+a_tilde)
        lambda3 = np.abs(u_tilde)

        eps1 = max(0,((u_tilde-a_tilde)-(uL[i]-aL[i])),((uR[i]-aR[i])-
(u_tilde-a_tilde)))
        eps2 = max(0,((u_tilde+a_tilde)-(uL[i]+aL[i])),((uR[i]+aR[i])-
(u_tilde+a_tilde)))

        if eps1 > lambda1:
            lambda1 = eps1
        if eps2 > lambda2:
            lambda2 = eps2

        F_half[0,i] = 0.5*(F[0,i]+F[0,ip]) -
0.5*(alpha1_tilde*lambda1+alpha2_tilde*lambda2+alpha3_tilde*
lambda3)
        F_half[1,i] = 0.5*(F[1,i]+F[1,ip]) -
0.5*(alpha1_tilde*lambda1*(u_tilde-
a_tilde)+alpha2_tilde*lambda2*(u_tilde+a_tilde)+alpha3_tilde*la
mbda3*u_tilde)
        F_half[2,i] = 0.5*(F[2,i]+F[2,ip]) -
0.5*(alpha1_tilde*lambda1*(H_tilde-
u_tilde*a_tilde)+alpha2_tilde*lambda2*(H_tilde+u_tilde*a_tilde)
+alpha3_tilde*lambda3*(u_tilde**2/2))

    for i in range(0,ix): # Q_update
        im=i-1
        ip=i+1

        # Neumann BCs implemented here using a series of if
statements
        if i == 0:
            im = 1
        if i == ix-1:
            ip = ix-2

        Q[:,i] = Q[:,i]-(dt/dx)*(F_half[:,i]-F_half[:,im])

        rho = Q[0,:]
        u = Q[1,:]/rho
        e = Q[2,:]/rho-u**2/2
        p = e*(gamma-1)*rho
        H = e+p/rho+u**2/2
        a = np.sqrt((gamma*p/rho))

        F[0,:] = rho*u
        F[1,:] = rho*u**2+p
        F[2,:] = rho*u*(e+p/rho+u**2/2)

        dt = c_max*dx/np.max(np.abs(u)+a)
        t = t+dt

        print(t)

    return rho, u, p, e, xdef roe1DMUSCL(test,c_max,dx):
    import numpy as np

    def roeavg(betaL,betaR,rhoL,rhoR):
        beta_tilde =
(betaL*np.sqrt(rhoL)+betaR*np.sqrt(rhoR))/(np.sqrt(rhoL)+np.sqrt
(rhoR))
        return beta_tilde

    def MUSCL(vm1,v,vp1,vp2):
        kappa = -1
        rmp12 = (v-vm1+1.e-30)/(vp1-v+1.e-30)
        rmp32 = (vp1-v+1.e-30)/(vp2-vp1+1.e-30)
        phi_rmp12 =  (np.abs(rmp12)+rmp12)/(1+np.abs(rmp12))
        phi_rmp32 = (np.abs(rmp32)+rmp32)/(1+np.abs(rmp32))
        phi_rpp12 = (1/rmp32)*phi_rmp32
        phi_rpm12 = (1/rmp12)*phi_rmp12
        del_pm12 = (v-vm1)*phi_rpm12/2
        del_mp12 = (vp1-v)*phi_rmp12/2
        del_mp32 = (vp2-vp1)*phi_rmp32/2
        del_pp12 = (vp1-v)*phi_rpp12
        del_L = (1-kappa)*del_pm12+(1+kappa)*del_mp12
        del_R = (1-kappa)*del_mp32+(1+kappa)*del_pp12
        vL12 = v+0.5*del_L
        vR12 = vp1-0.5*del_R
        return (vL12,vR12)

    if test == 1:
        # case 1 - Sod problem
        rhoL=1.0
        uL=0.0
        pL=1.0
        rhoR=0.125
        uR=0.0
        pR=0.1
        t_final = 0.25
    if test == 2:
        # case 2 - 123 problem - expansion left and expansion right
        rhoL=1.0
        uL=-2.0
        pL=0.4
        rhoR=1.0
        uR=2.0
        pR=0.4
        t_final = 0.15
    if test == 3:
        # case 3 - blast problem - shock right, expansion left
        rhoL=1.0
        uL=0.0
```

```
    pL=1000
    rhoR=1.0
    uR=0.
    pR=0.01
    t_final = 0.012
if test == 4:
    # case 4 - blast problem - shock left, expansion right
    rhoL=1.0
    uL=0.0
    pL=0.01
    rhoR=1.0
    uR=0.
    pR=100
    t_final = 0.035
if test == 5:
    # case 5 - shock collision - shock left and shock right
    rhoL=5.99924
    uL=19.5975
    pL=460.894
    rhoR=5.99242
    uR=-6.19633
    pR=46.0950
    t_final = 0.035

L = 1.0
gamma = 1.4
ix = int(L/dx+1)
half = int(np.floor(ix/2))
x = np.linspace(0,L,ix)

# storage vectors
rho = np.zeros(ix)
u = np.zeros(ix)
p = np.zeros(ix)
e = np.zeros(ix)
Q = np.zeros((3,ix))
F = np.zeros((3,ix))
F_half = np.zeros((3,ix))

#establish initial condition vectors
rho[0:half] = rhoL
rho[half:ix] = rhoR
u[0:half] = uL
u[half:ix] = uR
p[0:half] = pL
p[half:ix] = pR
e = p/((gamma-1)*rho)
H = e+p/rho+u**2/2
a = np.sqrt((gamma*p/rho))

t = 0
dt = c_max*dx/np.max(np.abs(u)+a)

#calculate initial Q and F vectors
Q[0,:] = rho
Q[1,:] = rho*u
Q[2,:] = rho*(e+u**2/2)

F[0,:] = rho*u
F[1,:] = rho*u**2+p
F[2,:] = rho*u*(e+p/rho+u**2/2)

rhoL = np.zeros(ix)
rhoR = np.zeros(ix)
uL = np.zeros(ix)
uR = np.zeros(ix)
pL = np.zeros(ix)
pR = np.zeros(ix)
eL = np.zeros(ix)
eR = np.zeros(ix)

rhoL[:] = rho
rhoR[:] = rho
uL[:] = u
uR[:] = u
pL[:] = p
pR[:] = p
eL[:] = e
eR[:] = e

while t <= t_final:
    for i in range(1,ix-2): # Loop for calculating Fn

        im=i-1
        ip=i+1
        ip2=i+2

        rhoL[i],rhoR[i] = MUSCL(rho[im],rho[i],rho[ip],rho[ip2])
        uL[i],uR[i] = MUSCL(u[im],u[i],u[ip],u[ip2])
        pL[i],pR[i] = MUSCL(p[im],p[i],p[ip],p[ip2])
        eL[i],eR[i] = MUSCL(e[im],e[i],e[ip],e[ip2])

    aL = np.sqrt(gamma*pL/rhoL)
    aR = np.sqrt(gamma*pR/rhoR)
    HL = eL+pL/rhoL+uL**2/2
    HR = eR+pR/rhoR+uR**2/2

    for i in range(0,ix): # Loop for calculating Fn

        im=i-1
        ip=i+1

        # Neumann BCs implemented here using a series of if
statements
        if i == 0:
            im = 1
        if i == ix-1:
            ip = ix-2

        rho_tilde = np.sqrt(rhoL[i]*rhoR[i])
        u_tilde = roeavg(uL[i],uR[i],rhoL[i],rhoR[i])
        a_tilde = roeavg(aL[i],aR[i],rhoL[i],rhoR[i])
        H_tilde = roeavg(HL[i],HR[i],rhoL[i],rhoR[i])

        del_p = pR[i]-pL[i]
        del_u = uR[i]-uL[i]
        del_rho = rhoR[i]-rhoL[i]

        alpha1_tilde = (del_p-
a_tilde*rho_tilde*del_u)/(2*a_tilde**2)
        alpha2_tilde =
(del_p+a_tilde*rho_tilde*del_u)/(2*a_tilde**2)
        alpha3_tilde = del_rho - del_p/a_tilde**2

        lambda1 = np.abs(u_tilde-a_tilde)
        lambda2 = np.abs(u_tilde+a_tilde)
        lambda3 = np.abs(u_tilde)

        eps1 = max(0,((u_tilde-a_tilde)-(uL[i]-aL[i])),((uR[i]-aR[i])-
(u_tilde-a_tilde)))
        eps2 = max(0,((u_tilde+a_tilde)-(uL[i]+aL[i])),((uR[i]+aR[i])-
(u_tilde+a_tilde)))
```

```
        if eps1 > lambda1:
            lambda1 = eps1
        if eps2 > lambda2:
            lambda2 = eps2

        F_half[0,i] = 0.5*(F[0,i]+F[0,ip]) -
0.5*(alpha1_tilde*lambda1+alpha2_tilde*lambda2+alpha3_tilde*
lambda3)
        F_half[1,i] = 0.5*(F[1,i]+F[1,ip]) -
0.5*(alpha1_tilde*lambda1*(u_tilde-
a_tilde)+alpha2_tilde*lambda2*(u_tilde+a_tilde)+alpha3_tilde*la
mbda3*u_tilde)
        F_half[2,i] = 0.5*(F[2,i]+F[2,ip]) -
0.5*(alpha1_tilde*lambda1*(H_tilde-
u_tilde*a_tilde)+alpha2_tilde*lambda2*(H_tilde+u_tilde*a_tilde)
+alpha3_tilde*lambda3*(u_tilde**2/2))

      for i in range(0,ix): # Q_update
        im=i-1
        ip=i+1

        # Neumann BCs implemented here using a series of if
statements
        if i == 0:
            im = 1
        if i == ix-1:
            ip = ix-2

        Q[:,i] = Q[:,i]-(dt/dx)*(F_half[:,i]-F_half[:,im])

      rho = Q[0,:]
      u = Q[1,:]/rho
      e = Q[2,:]/rho-u**2/2
      p = e*(gamma-1)*rho
      H = e+p/rho+u**2/2
      a = np.sqrt((gamma*p/rho))

      F[0,:] = rho*u
      F[1,:] = rho*u**2+p
      F[2,:] = rho*u*(e+p/rho+u**2/2)

      dt = c_max*dx/np.max(np.abs(u)+a)
      t = t+dt

      print(t)

   return rho, u, p, e, xdef roe1DMUSCL(test,c_max,dx):
   import numpy as np

   def roeavg(betaL,betaR,rhoL,rhoR):
      beta_tilde =
(betaL*np.sqrt(rhoL)+betaR*np.sqrt(rhoR))/(np.sqrt(rhoL)+np.sqrt
(rhoR))
      return beta_tilde

   def MUSCL(vm1,v,vp1,vp2):
      kappa = -1
      rmp12 = (v-vm1+1.e-30)/(vp1-v+1.e-30)
      rmp32 = (vp1-v+1.e-30)/(vp2-vp1+1.e-30)
      phi_rmp12 =  (np.abs(rmp12)+rmp12)/(1+np.abs(rmp12))
      phi_rmp32 = (np.abs(rmp32)+rmp32)/(1+np.abs(rmp32))
      phi_rpp12 = (1/rmp32)*phi_rmp32
      phi_rpm12 = (1/rmp12)*phi_rmp12
      del_pm12 = (v-vm1)*phi_rpm12/2
      del_mp12 = (vp1-v)*phi_rmp12/2
      del_mp32 = (vp2-vp1)*phi_rmp32/2
      del_pp12 = (vp1-v)*phi_rpp12
      del_L = (1-kappa)*del_pm12+(1+kappa)*del_mp12
      del_R = (1-kappa)*del_mp32+(1+kappa)*del_pp12
      vL12 = v+0.5*del_L
      vR12 = vp1-0.5*del_R
      return (vL12,vR12)

if test == 1:
   # case 1 - Sod problem
   rhoL=1.0
   uL=0.0
   pL=1.0
   rhoR=0.125
   uR=0.0
   pR=0.1
   t_final = 0.25
if test == 2:
   # case 2 - 123 problem - expansion left and expansion right
   rhoL=1.0
   uL=-2.0
   pL=0.4
   rhoR=1.0
   uR=2.0
   pR=0.4
   t_final = 0.15
if test == 3:
   # case 3 - blast problem - shock right, expansion left
   rhoL=1.0
   uL=0.0
   pL=1000
   rhoR=1.0
   uR=0.
   pR=0.01
   t_final = 0.012
if test == 4:
   # case 4 - blast problem - shock left, expansion right
   rhoL=1.0
   uL=0.0
   pL=0.01
   rhoR=1.0
   uR=0.
   pR=100
   t_final = 0.035
if test == 5:
   # case 5 - shock collision - shock left and shock right
   rhoL=5.99924
   uL=19.5975
   pL=460.894
   rhoR=5.99242
   uR=-6.19633
   pR=46.0950
   t_final = 0.035

L = 1.0
gamma = 1.4
ix = int(L/dx+1)
half = int(np.floor(ix/2))
x = np.linspace(0,L,ix)

# storage vectors
rho = np.zeros(ix)
u = np.zeros(ix)
p = np.zeros(ix)
e = np.zeros(ix)
Q = np.zeros((3,ix))
```

```
F = np.zeros((3,ix))
F_half = np.zeros((3,ix))

#establish initial condition vectors
rho[0:half] = rhoL
rho[half:ix] = rhoR
u[0:half] = uL
u[half:ix] = uR
p[0:half] = pL
p[half:ix] = pR
e = p/((gamma-1)*rho)
H = e+p/rho+u**2/2
a = np.sqrt((gamma*p/rho))

t = 0
dt = c_max*dx/np.max(np.abs(u)+a)

#calculate initial Q and F vectors
Q[0,:] = rho
Q[1,:] = rho*u
Q[2,:] = rho*(e+u**2/2)

F[0,:] = rho*u
F[1,:] = rho*u**2+p
F[2,:] = rho*u*(e+p/rho+u**2/2)

rhoL = np.zeros(ix)
rhoR = np.zeros(ix)
uL = np.zeros(ix)
uR = np.zeros(ix)
pL = np.zeros(ix)
pR = np.zeros(ix)
eL = np.zeros(ix)
eR = np.zeros(ix)

rhoL[:] = rho
rhoR[:] = rho
uL[:] = u
uR[:] = u
pL[:] = p
pR[:] = p
eL[:] = e
eR[:] = e

while t <= t_final:
   for i in range(1,ix-2): # Loop for calculating Fn

      im=i-1
      ip=i+1
      ip2=i+2

      rhoL[i],rhoR[i] = MUSCL(rho[im],rho[i],rho[ip],rho[ip2])
      uL[i],uR[i] = MUSCL(u[im],u[i],u[ip],u[ip2])
      pL[i],pR[i] = MUSCL(p[im],p[i],p[ip],p[ip2])
      eL[i],eR[i] = MUSCL(e[im],e[i],e[ip],e[ip2])

   aL = np.sqrt(gamma*pL/rhoL)
   aR = np.sqrt(gamma*pR/rhoR)
   HL = eL+pL/rhoL+uL**2/2
   HR = eR+pR/rhoR+uR**2/2

   for i in range(0,ix): # Loop for calculating Fn

      im=i-1
      ip=i+1

      # Neumann BCs implemented here using a series of if
statements
      if i == 0:
         im = 1
      if i == ix-1:
         ip = ix-2

      rho_tilde = np.sqrt(rhoL[i]*rhoR[i])
      u_tilde = roeavg(uL[i],uR[i],rhoL[i],rhoR[i])
      a_tilde = roeavg(aL[i],aR[i],rhoL[i],rhoR[i])
      H_tilde = roeavg(HL[i],HR[i],rhoL[i],rhoR[i])

      del_p = pR[i]-pL[i]
      del_u = uR[i]-uL[i]
      del_rho = rhoR[i]-rhoL[i]

      alpha1_tilde = (del_p-
a_tilde*rho_tilde*del_u)/(2*a_tilde**2)
      alpha2_tilde =
(del_p+a_tilde*rho_tilde*del_u)/(2*a_tilde**2)
      alpha3_tilde = del_rho - del_p/a_tilde**2

      lambda1 = np.abs(u_tilde-a_tilde)
      lambda2 = np.abs(u_tilde+a_tilde)
      lambda3 = np.abs(u_tilde)

      eps1 = max(0,((u_tilde-a_tilde)-(uL[i]-aL[i])),((uR[i]-aR[i])-
(u_tilde-a_tilde)))
      eps2 = max(0,((u_tilde+a_tilde)-(uL[i]+aL[i])),((uR[i]+aR[i])-
(u_tilde+a_tilde)))

      if eps1 > lambda1:
         lambda1 = eps1
      if eps2 > lambda2:
         lambda2 = eps2

      F_half[0,i] = 0.5*(F[0,i]+F[0,ip]) -
0.5*(alpha1_tilde*lambda1+alpha2_tilde*lambda2+alpha3_tilde*
lambda3)
      F_half[1,i] = 0.5*(F[1,i]+F[1,ip]) -
0.5*(alpha1_tilde*lambda1*(u_tilde-
a_tilde)+alpha2_tilde*lambda2*(u_tilde+a_tilde)+alpha3_tilde*la
mbda3*u_tilde)
      F_half[2,i] = 0.5*(F[2,i]+F[2,ip]) -
0.5*(alpha1_tilde*lambda1*(H_tilde-
u_tilde*a_tilde)+alpha2_tilde*lambda2*(H_tilde+u_tilde*a_tilde)
+alpha3_tilde*lambda3*(u_tilde**2/2))

   for i in range(0,ix): # Q_update
      im=i-1
      ip=i+1

      # Neumann BCs implemented here using a series of if
statements
      if i == 0:
         im = 1
      if i == ix-1:
         ip = ix-2

      Q[:,i] = Q[:,i]-(dt/dx)*(F_half[:,i]-F_half[:,im])

   rho = Q[0,:]
   u = Q[1,:]/rho
   e = Q[2,:]/rho-u**2/2
   p = e*(gamma-1)*rho
   H = e+p/rho+u**2/2
```

```
    a = np.sqrt((gamma*p/rho))

    F[0,:] = rho*u
    F[1,:] = rho*u**2+p
    F[2,:] = rho*u*(e+p/rho+u**2/2)

    dt = c_max*dx/np.max(np.abs(u)+a)
    t = t+dt

    print(t)

return rho, u, p, e, xvvv
```