Compressible Flow Simulation Using the Rusanov Numerical Method

Jimmy Pare
The University of Vermont

January 3, 2026

Contents

1 Abstract 3

2 Introduction 4

2.1 Shock Tube Background 4

2.2 Compressible Flow Background 4

2.3 Reference Paper 4

3 Methods 5

3.1 Initial and Boundary Conditions 7

4 Results and Discussion 8

5 Conclusions 10

6 References 11

A Python Code 12

1 Abstract

This paper uses the Rusanov numerical method to study the compressible flow property distributions of a shock tube. Shock tubes are widely used to quickly and easily accelerate a flow in many aerospace applications by quickly allowing a high pressure gas to flow into a low pressure region, initially separated by a diaphragm. After the the diaphragm is broken, the compressible flow splits into five regions divided by shocks, expansion waves, and contact discontinuities. The Rusanov numerical scheme was used in Python to predict this behavior. This was done by discretizing the domain of interest and stepping through a time period. Numerical results aligned closely with graphs from a paper written by Gary A. Sod from 1978.

2 Introduction

2.1 Shock Tube Background

Shock tubes are widely used as an alternative method to wind tunnels for accelerating compressible flows. While high-end wind tunnels are able to provide a continuous supersonic flow, they are extremely expensive and require a great amount of power. Shock tubes can accelerate a flow to the same speeds or faster while requiring a fraction of the resources. This is accomplished by pressurizing a region in a tube (driver) and then quickly opening a diaphragm or valve to induce a flow into the driven region. The drawback is that shock tubes are unable to do this continuously, since they are restricted by the duration that they can maintain these conditions.

2.2 Compressible Flow Background

Compressible flows differ from standard fluid mechanics because density changes are no longer negligible. While applying the Navier-Stokes equations, a common assumption to reduce complexity is constant density. This assumption is no longer valid when velocities are faster than .3 times the speed of sound, shown below.

$$M = \frac{v}{a} \geq .3 \tag{1}$$

Where M is the Mach number, v is the relative velocity, and a is the speed of sound. In the case of a shock tube, flow is often accelerated faster than the speed of sound, causing shocks to form. These shocks over-compress the flow, causing expansion waves to form. The interaction between these features in the flow creates extremely complex conditions. When entering these compressible flow regimes, numerical methods are not just useful, but required in order to obtain accurate results.

2.3 Reference Paper

This study seeks to recreate the results of "A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws" by Gary A Sod. This paper compares various different numerical methods using a shock tube problem. Graphs showing the desired density, pressure, and velocity distributions of the shock tube calculated using the Rusanov method are shown on pages 8-9.

3 Methods

This study utilizes the Rusanov scheme to predict shock tube flow behavior. 200 finite elements were used to divide the domain between $x = 0$ and $x = 1$, which was then incremented in time.

This numerical method works by using a flux (or derivative) term between cells to calculate the next time step Δt. The difference between state properties at the current time step is used to calculate the properties at the next time step. Shown in Equation 2, the state property vector U consists of the density ρ, momentum ρu, and Energy E.

$$U = \begin{bmatrix} \rho \\ \rho u \\ E \end{bmatrix} \tag{2}$$

$$U_i^{n+1} = U_i^n - (\frac{\Delta t}{2\Delta x})(F_{i+1}^n - F_{i-1}^n) \tag{3}$$

Equation 3 is the governing equation of the Rusanov method. This equation draws similarities to the explicit Euler method because the derivative is calculated at the current time step, rather than implicitly at the next. Calculating this way cuts back on computational cost and is superior at capturing large changes in variables.

The difference in the state properties between cells is used to calculate the flux F shown below.

$$F(U) = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ u(E + p) \end{bmatrix} \tag{4}$$

$$F_{i+\frac{1}{2}} = \frac{1}{2}(F_L + F_R) - \frac{1}{2}a_{i+1/2}(U_R - U_L) \tag{5}$$

Where a is a diffusive term shown in Equation 7. After calculating the flux between cells, the state properties at the next time step are calculated and updated. For an ideal polytropic gas, the pressure p can be extracted from the energy using Equation 6.

$$p = (\gamma - 1)(E - \frac{1}{2}\rho u^2) \tag{6}$$

Where $\gamma = 1.4$ is the ratio of heat capacities for air.

The following relation for the diffusive term a is used to stabilize the simulation. Without this term to help model viscous effects and smooth gradients, the Rusanov method usually does not converge. The use of a is necessary both for computational stability and to help model viscous effects.

$$a_{i+\frac{1}{2}} = \max(|u_L| + c_L, |u_R + c_R|) \tag{7}$$

The Rusanov numerical method is able to accurately and reliably model compressible flow behavior such as shocks, expansion waves, and contact discontinuities. Although the Rusanov method can reliably model compressible flows, it behaves as an overdamped system which leads to errors around sharp property changes.

3.1 Initial and Boundary Conditions

Before $t = 0$ in the simulation, the domain is split into driver and driven regions, on the left and right hand sides, respectfully. The driver region is the initially pressurized domain and the driven region is the low pressure domain. Within the driver region:

- $p = 1$
- $\rho = 1$

Within the driven region:

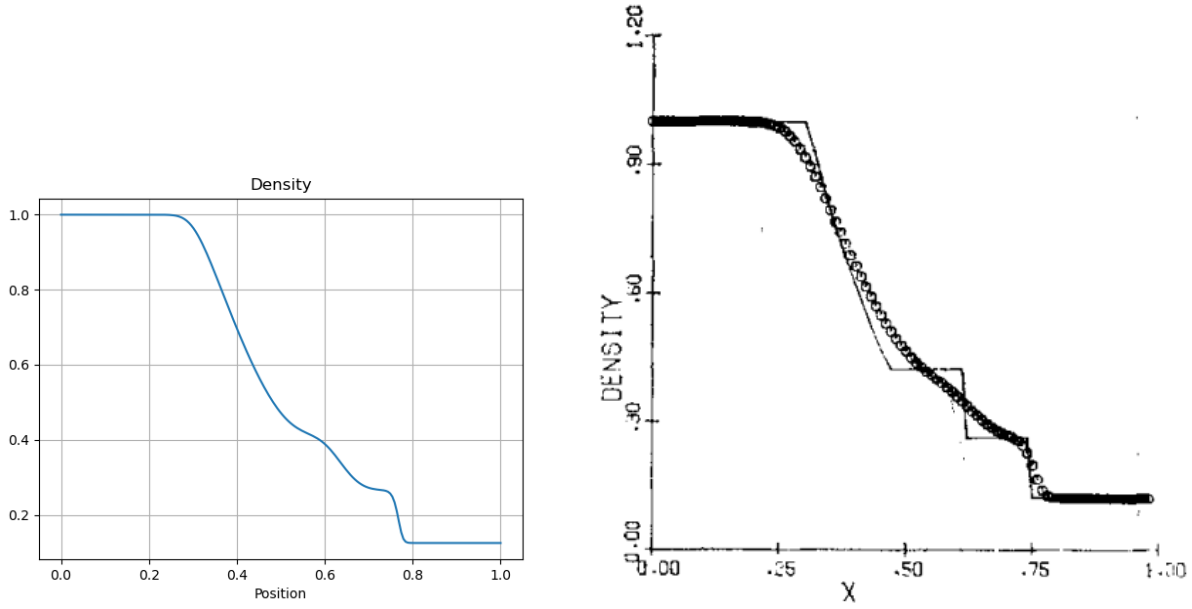
- $p = .1$
- $\rho = .125$

Velocity is set to 0 across the whole domain. It should be noted that all variables are unitless, including time. In order for units to be applied, the same convention must be applied to all variables. Once $t = 0$, the Rusanov method is allowed to calculate fluxes across the property gradients and predict flow regimes.

On the left and right borders of the domain, a Neumann boundary condition was used, meaning the derivative of the state properties with respect to space was assumed to be 0. This prohibits waves from reflecting upstream, most closely replicating a tube with open ends.

4 Results and Discussion

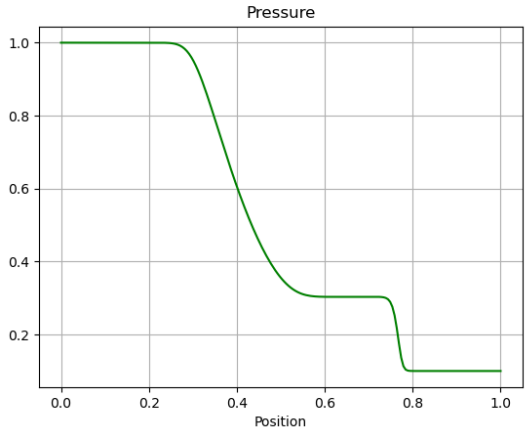
After running the Python simulation, plots were generated for the density, pressure, and velocity at $t = .15$. The time step for these snapshots was not explicitly listed in Sod's paper, but estimated to be between $t = .14$ and $t = .15$ by "Riemann Solvers and Numerical Methods for Fluid Dynamics" by E. F. Toro in 1997. Figure 1 shows the calculated density distribution compared to Sod's findings. The following Figures on the left show this paper's findings compared to Sod's findings on the right. The solid line in Sod's findings is the analytical solution for each variable.



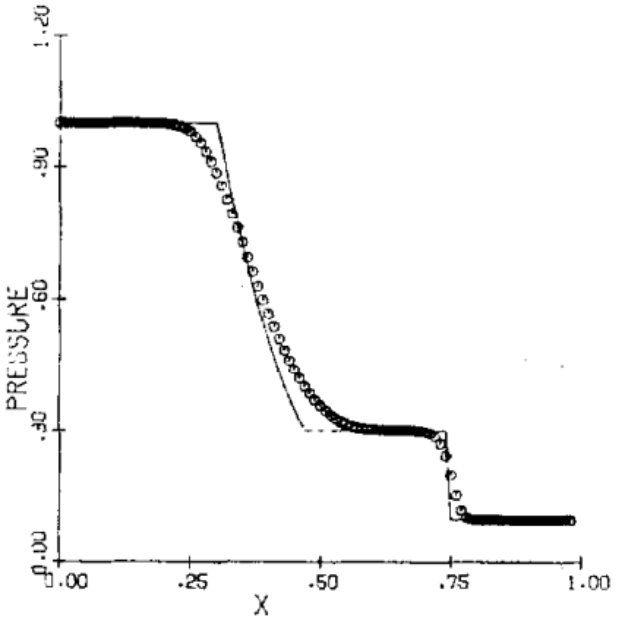
(a) A graph of the normalized density vs. axial distance using the recreated Rusanov scheme.

(b) A graph of the normalized density vs. axial distance from Sod's paper.

Figure 1: A comparison between the recreated density graph and Sod's density figure from the paper.

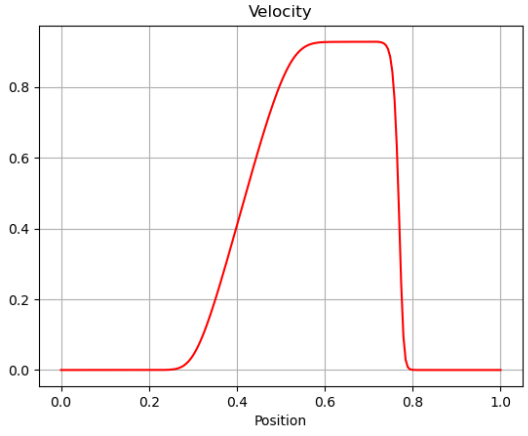


(a) A graph of the normalized pressure vs. axial distance using the recreated Rusanov scheme.

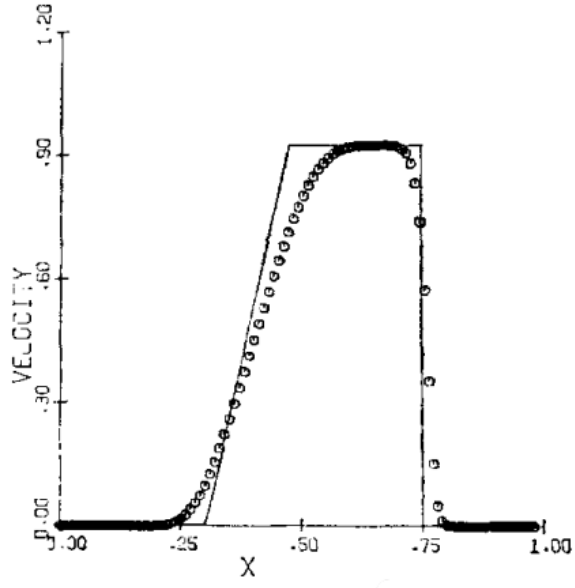


(b) A graph of the normalized pressure vs. axial distance from Sod's paper.

Figure 2: A comparison between the recreated pressure graph and Sod' pressure figure from the paper.



(a) A graph of the normalized velocity vs. axial distance using the recreated Rusanov scheme.



(b) A graph of the normalized velocity vs. axial distance from Sod's paper.

Figure 3: A comparison between the recreated velocity graph and Sod's velocity figure from the paper.

The results closely resemble that of Sod's findings, providing confidence in the model. For each graph, sharp gradients are observed at $x = .75$, indicating a leading shock wave moving left to right. Around $x \approx .4$, an expansion or rarefaction wave exists. This study's calculated results behave as an over damped system, insensitive to rapid changes. As discussed in the Methods section, this is a characteristic of the Rusanov method and is expected.

One area of interest is the secondary step at $x \approx .65$ in Figure 1. This step behind the initial shock is due to the initial density discontinuity. Pressure and temperature gradients smooth out relatively quickly with the Rusanov method, but density discontinuities do not smooth nearly as fast. A slight jump is observed using this paper's model at this location that is not observed with Sod's model. Since this jump in density corresponds to Sod's analytical solution, it suggests that this study's Python simulation is more sensitive than Sod's, showing more confidence in the model.

5 Conclusions

The objective of this study was to use numerical methods for shock tube analysis. The Rusanov scheme provided insight into the pressure, velocity, and density distributions over time of the compressible flow in the shock tube. The flow property distributions align very well with results from Sod's paper, providing confidence in the model.

Simulations display that the flow splits into five regions due to shocks and a rarefaction wave. With higher initial pressure ratios, the induced flow travels more quickly and the strength of the shock increases. The units in this simulation were based on the dimensionless input parameters. To apply results to a real-world scenario, units must be kept consistent for the input parameters.

Future work can be done to better model viscous effects, more accurate boundary conditions, and expand the model from one dimension to three. These improvements would help predictions align better with experiment and provide more confidence in the model.

6 References

[1] Sod, Gary A. "A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws."

[2] Toro, E. F. *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction*. 3rd ed., Springer-Verlag, 2009. (First published 1997).

A Python Code

Listing 1: Python code used to generate Figures 1a, 2a, and 3a.

```
import numpy as np
import matplotlib.pyplot as plt

# Simulation paramaters
nx = 200                                # number of cells
dx = 1 / nx                             # cell size
gamma = 1.4                             # ratio of heat capacities is
    1.4 for air (assume const.)
t_end = .15                             # not explicitly given in paper
    . Around .15 or .14 is decent guess
x = np.linspace(0, 1, nx)               # create linspace to set up
    cells
dt = 1e-3

# initial / boundary conditions
rho = np.where(x < 0.5, 1, .125)  # density is 1 on the LHS, and
    .125 on RHS
u = np.zeros(nx)                        # velocity starts off as 0
    everythere
p = np.where(x < 0.5, 1, 0.1 )      # similarly as density, pressure
     is 1 on LHS and .1 on RHS

E = p/(gamma-1) + 0.5*rho*u**2          # total energy equation for
    polytropic process. Essentially dynamic pressure + internal
    energy
U = np.array([rho, rho*u, E])           # state vector U is a 3 x n
    matrix with rows: density, momentum, and energy


def flux(U):
    rho = U[0]                          # unpack state vector U. first
        row is density
    u = U[1]/rho
    p = (gamma - 1)*(U[2] - 0.5*rho*u**2)

```

```
    return np.array([
        rho * u,                        # these flux equations are from
            Sod's paper. the flux function returns the flux on cell
           boundaries.
        rho * u**2 + p,
        u * (U[2] + p)])

t = 0
while t < t_end:                        # use a while loop for main
   loop

    rho = U[0]                          # unpack state vector, same as
       for flux function.
    u = U[1]/rho
    p = (gamma - 1)*(U[2] - 0.5*rho*u**2)

    c = np.sqrt(gamma*p/rho)            # the speed of sound is the
       sqrt(gamma * R * T), but with ideal gas we can sub
       temperature
    t += dt                             # step forward time dt

    F = np.zeros((3, nx+1))             # initialize flux vector. It
       has nx+1 columns because there is one additional cell
       boundary as cells

    for i in range(nx+1):

        if i == 0:
            UL = UR = U[:,0]                # there is a zero gradient
               boundary condition. In other words, the flux on the
               left of the first cell equals the right.
        elif i == nx:
            UL = UR = U[:,-1]               # similarly, zero gradient
               BC on RHS. Net flux equals 0
        else:
            UL, UR = U[:,i-1], U[:,i]       # set state vector on left
               equal to previous cell, state vector on right equal
               to current cell.
```

```
51
52          FL, FR = flux(UL), flux(UR)       # use the flux function
                with state vector on left and state vector on right
53
54          uL = UL[1]/UL[0]                  # divide the momentum by
                the density to get the velocity on the LHS
55          pL = (gamma-1)*(UL[2] - 0.5*UL[0]*uL**2)       # use the
                energy equation to get the pressure on LHS
56          cL = np.sqrt(gamma*pL/UL[0])      # use ideal gas law with
                speed of sound equation
57
58          uR = UR[1]/UR[0]                  # same thing on RHS
59          pR = (gamma-1)*(UR[2] - 0.5*UR[0]*uR**2)
60          cR = np.sqrt(gamma*pR/UR[0])
61
62          a = max(abs(uL)+cL, abs(uR)+cR)     # diffusivity term,
                defined from paper. Helps model converge instead of blow
                up. Accounts for viscous effects.
63
64          F[:,i] = 0.5*(FL + FR) - 0.5*a*(UR - UL)   # this equation
                comes from the paper. The fluxes are added because os
                sign convention.
65
66
67      U[:, :] -= (dt/dx) * (F[:, 1:] - F[:, :-1])    # this is the
            main Rusanov scheme equation. Update the U state vector
68
69  rho = U[0]              # same as before, unpack state vector to plot
        density, velocity, and pressure
70  u = U[1] / rho
71  p = (gamma-1)*(U[2] - 0.5*rho*u**2)
72
73
74  plt.plot(x, rho)
75  plt.title("Density")
76  plt.xlabel("Position")
77  plt.grid()
78  plt.savefig("DensityGraph_t15")
```

```
plt.show()

plt.plot(x, u,color="red")
plt.title("Velocity")
plt.xlabel("Position")
plt.grid()
plt.savefig("VelocityGraph_t15")
plt.show()

plt.plot(x, p, color="green")
plt.title("Pressure")
plt.xlabel("Position")
plt.grid()
plt.savefig("PressureGraph_t15")
plt.show()
```