Hydrodynamics Project Report

Andrew Dubovskiy, Robert Xi, Jiajun Xu

December 2023

1 Introduction

In this project, our team investigated into hydrodynamics simulation in both 1D and 2D systems, employing various computational methods such as the Harten-Lax-van Leer (HLL) approximation and the third-order Runge-Kutta algorithm. The project is structured in three main parts. For the first part, we investigated the hydrodynamics of lower order in a one-dimensional system. Subsequently, we applied higher order approximations in the second part. Finally, in the last part, we combined what we learnt from the previous parts to construct a simulation for a two-dimensional hydrodynamics system.

The idea is simplifying the complex real-world scenario into a finite element system. This allows us to compute the system's status for the next time step using hydrodynamics and the current system information. While theoretically, we could fully solve the problem with complete information about the current system. However, the computational cost would be prohibitively high. Therefore, we omit higher-order terms that have minimal impact on the system, retaining sufficient information to accurately represent the actual case. Of course, the accuracy of the results is directly proportional to the amount of information utilized at the expense of increased computation time.

We conducted a comparison of simulation results in a one-dimensional system using both lower and higher order methods using python. Additionally, results of a two-dimensional simulation under varied initial conditions are also presented.

2 Sod Shock Tube Problem

2.1 Low-order 1D Method

The first part of our project focuses on the application of Low-order 1D method to solve Sod Shock Tube Problem. The Sod shock tube problem is a classical test case in computational fluid dynamics (CFD) and numerical analysis, specifically for solving hyperbolic partial differential equations, such as those governing fluid dynamics. We will first introduce some relevant theories and how to implement them into code, then discuss the results obtained.

The equations of one-dimensional hydrodynamics in the conservation form

$$\frac{\partial \vec{U}}{\partial t} + \frac{\partial \vec{F}}{\partial x} = 0 \tag{1}$$

where $\vec{U} = (\rho, \rho v, E)$ are the conserved variables for mass, momentum, and energy, and $\vec{F} = (\rho v, \rho v^2 + P, (E+P)v)$ are the fluxes of those quantities. ρ is the density, v is the velocity, P is the pressure, and $E = \rho e + \frac{\rho v^2}{2}$ is the total energy density. The quantity e is the specific internal energy. For an ideal gas:

$$P = (\gamma - 1)\rho e \tag{2}$$

where γ is the adiabatic index. Hence, we can derive the following equations from equation 2 and some additional conditions.

$$\vec{U} = \begin{pmatrix} \rho \\ \rho v \\ E \end{pmatrix} = \begin{pmatrix} \rho \\ \rho v \\ \rho e + \frac{\rho v^2}{2} \end{pmatrix} \tag{3}$$

$$\vec{F} = \begin{pmatrix} \rho v \\ \rho v^2 + P \\ (E+P)v \end{pmatrix} = \begin{pmatrix} \rho v \\ \rho(v^2 + e(\gamma - 1)) \\ \rho v(\gamma e + \frac{v^2}{2}) \end{pmatrix} \tag{4}$$

By observation, we can get $\vec{U}$ and $\vec{F}$ just using the values of ρ, v, and e, where γ is just a predetermined constant. Notice that by knowing any one of the three vectors, we are able to derive the other two.

First, we calculate the Harten-Lax-van Leer approximation flux $\vec{F}^{HLL}$ based on the formula and $\vec{U}^{t_n}$ and $\vec{F}^{t_n}$ for time at t_n

$$\vec{F}^{\mathrm{HLL}}_{n+\frac{1}{2}} = \frac{\alpha^+ \vec{F}^{t_n}_L + \alpha^- \vec{F}^{t_n}_R - \alpha^+\alpha^- \left(\vec{U}^{t_n}_R - \vec{U}^{t_n}_L\right)}{\alpha^+ + \alpha^-}. \tag{5}$$

$$\vec{G}^{\mathrm{HLL}}_{n+\frac{1}{2}} = \frac{\alpha^+ \vec{G}^{t_n}_L + \alpha^- \vec{G}^{t_n}_R - \alpha^+\alpha^- \left(\vec{U}^{t_n}_R - \vec{U}^{t_n}_L\right)}{\alpha^+ + \alpha^-}. \tag{6}$$

where L, R can be $n, n+1$ represents the spacial index. This $\vec{F}^{\mathrm{HLL}}_{n+\frac{1}{2}}$ represents the flux of respective quantity at the boundary or interface between two cells, i.e. the cell between $n, n+1$. The constant $\alpha^{\pm}$ can be calculated by

$$\alpha^{\pm}_{i+\frac{1}{2}} = \max\left\{0, \pm\lambda^{\pm}\left(\vec{U}^{t_n}_i\right), \pm\lambda^{\pm}\left(\vec{U}^{t_n}_{i+1}\right)\right\} \tag{7}$$

where $\lambda^{\pm} = v \pm \sqrt{\frac{\gamma P}{\rho}}$, v is the velocity. Next, We can calculate the derivative of $\vec{U}$ using following equation

$$\frac{\partial \vec{U}_i}{\partial t} = -\frac{\vec{F}_{i+\frac{1}{2}} - \vec{F}_{i-\frac{1}{2}}}{\Delta x} \tag{8}$$

Then, we are able to obtain $\vec{U}^{t_{n+1}}$, which is $\vec{U}$ at next time step $n+1$, given $\vec{U}^{t_n}$ and $\frac{\partial \vec{U}_i}{\partial t}$,

$$\begin{aligned}\vec{U}^{t_{n+1}} &= \vec{U}^{t_n} + \Delta t \cdot \frac{\partial \vec{U}_i^{t_n}}{\partial t} \\ &= \vec{U}^{t_n} + \Delta t \cdot - \frac{\vec{F}_{i+\frac{1}{2}} - \vec{F}_{i-\frac{1}{2}}}{\Delta x}\end{aligned} \tag{9}$$

To summarize, the simulation process begins by utilizing the initial condition $\vec{U}^{t_0}$ to calculate $\vec{F}^{t_0}$. Subsequently, the flux at the interface, $\vec{F}^{\text{HLL}}_{n+\frac{1}{2}}$, is computed using equation 5. This information is then employed in equation 7 to derive the time derivative of $\vec{U}^{t0}$, ultimately leading to the final step of determining $\vec{U}^{t_{n+1}}$ through equation 8.

2.1.1 Derivation And Code Implementation

The idea is to create several numpy arrays with nx columns and nt rows, each column represents physical quantities for different time steps for a single cell, and each row represents system status at the different time step. We defined several arrays to account for different physical parameters, which includes $\vec{U}$, $\vec{F}$, $F^{\vec{HLL}}$, and $U^{deri\vec{v}ative}$. Just as the table shows below for the $\vec{U}$ arrangement in matrix:

...	$U_{i-2}^{t_n}$	$U_{i-1}^{t_n}$	$U_i^{t_n}$	$U_{i+1}^{t_n}$	$U_{i+2}^{t_n}$	...
...	$U_{i-2}^{t_{n+1}}$	$U_{i-1}^{t_{n+1}}$	$U_i^{t_{n+1}}$	$U_{i+1}^{t_{n+1}}$	$U_{i+2}^{t_{n+1}}$	...

For this particular question, we assigned $\rho_L, P_L, e_L = U_i$ for all position on the left half, and $\rho_R, P_R, e_R = U_j$ on the right half such that they satisfy the following equations:

$$\begin{aligned}\frac{P_L}{P_R} &= 8 \\ \frac{\rho_L}{\rho_R} &= 10 \\ \gamma &= 1.4\end{aligned} \tag{10}$$

Using equation 18 and 19, we are able to get the following expressions for velocity v and pressure P:

$$\begin{cases} v = \frac{\rho v}{\rho} = \frac{U(1)}{U(0)} \\ P = (\gamma-1)\rho \cdot \left(\frac{E}{\rho} - \frac{1}{2}v^2\right) = (\gamma-1)U(0)(\frac{U(2)}{U(0)} - \frac{1}{2}(\frac{U(1)}{U(0)})^2) \\ e = \frac{P}{(\gamma-1)U(0)} \end{cases} \tag{11}$$

where $U(i)$ denotes the value in the ith row of matrix $\vec{U}$. Then we can calculate $\vec{F}$ using 11. In order to determine the time derivative of $\vec{U}$, we use the HLL approximation provided in equation 5. From that, we can calculate $\vec{U}$ for the next time step using equation 8.

In the code, we defined the following functions: `initialize()` creates the four numpy arrays as mentioned above, as well as the initial condition for $\vec{U}$;

`find_f(U)`, this function calculates $\vec{F}$ for corresponding $\vec{U}$; `find_f_half(U,F))` calculates $\vec{F}^{\text{HLL}}_{n+\frac{1}{2}}$ at the interface; `find_u_der(F_half,delta_x)`, this function calculates the time derivative of $\vec{U}$ based on the input value $\vec{F}^{\text{HLL}}_{n+\frac{1}{2}}$ and Δx; `find_u(U, U_der,delta_t,ii)` computes $\vec{U}$ for the next time step; finally, `evolve(i)` loops all the functions together an allow us to choose how many time steps to simulates.

All these functions can be found in the file called "hydro.py". The main code is in "new Sod Shock Tube.py" file.

2.1.2 Result

For this part, we choose $nx = 300$, $nt = 1000$. While determining the appropriate time step Δt and space step Δx, we consulted the following criterion:

$$\Delta t < \Delta x / max(\alpha^{\pm})$$

By doing some preliminary tests, we found that normally $max(\alpha^{\pm})$ is around 1 to 1.5. Thus, we choose $\Delta x = 1$ and $\Delta t = 0.1$. To make the plot looks clearer, we have depicted values at intervals of every 10 time steps. Figures 1, 2, and 3 showcase the velocity, density, and pressure distribution of the 1D Sod shock tube, respectively, using the lower-order simulation. The horizontal axis denotes the spatial distance along the tube, while the vertical axis represents the corresponding values. To maintain consistency, we have opted to retain the boundary condition identical to the initial condition. Consequently, it's important to note that simulating an extended duration may lead to a substantial impact from this boundary condition on the results, so we always keep the simulation stop before it hit the boundaries.

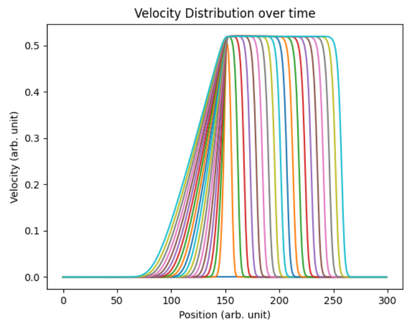


Figure 1: velocity distribution, horizontal axis position, vertical axis velocity. different line represents velocity distribution at different time.nt = 1000, nx = 300, simulated using lower order method

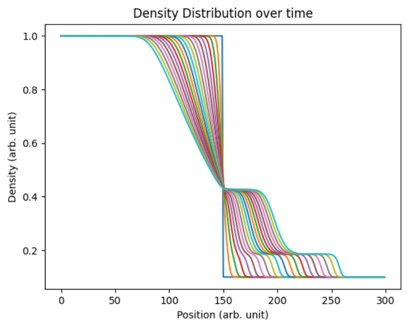


Figure 2: density distribution, horizontal axis position, vertical axis density. different line represents density distribution at different time. nt = 1000, nx = 300, simulated using lower order method

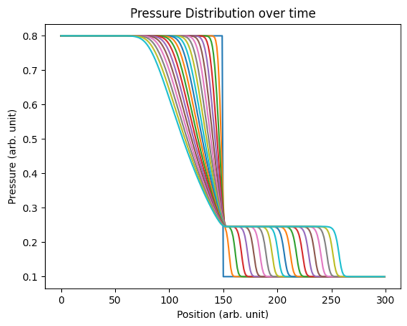


Figure 3: pressure distribution, horizontal axis position, vertical axis pressure. different line represents pressure distribution at different time. nt = 1000, nx = 300, simulated using lower order method

2.2 Higher Order 1D Method

The second part of our project employing a higher-order 1D method to address the Sod Shock Tube Problem. We use the third-order Runge-Kutta to reduce the approximation error. This method fundamentally involves computations with a richer dataset, transitioning from the previous utilization of only three data points around the cell to incorporating five data points. Additionally, for each time step, the method iterates three more times to yield a more precise outcome at the cost of computational speed. The method starts with $U(\vec{t_n})$ to calculate $U(\vec{t_{n+1}})$:

$$\begin{aligned}\vec{U}^{(1)} &= \vec{U}(t_n) + \Delta t L(\vec{U}(t_n)) \\ \vec{U}^{(2)} &= \frac{3}{4}\vec{U}(t_n) + \frac{1}{4}\vec{U}^{(1)} + \frac{1}{4}\Delta t L(\vec{U}^{(1)}) \\ \vec{U}(t_{n+1}) &= \frac{1}{3}\vec{U}(t_n) + \frac{2}{3}\vec{U}^{(2)} + \frac{2}{3}\Delta t L(\vec{U}^{(2)})\end{aligned} \tag{12}$$

In space we go to higher order by using more temporal points to determine the left and right conditions on each cell interface from which to calculate fluxes, instead of just the piece-wise constant model. For example, there exists $\frac{1}{4}\Delta t$ and $\frac{2}{3}\Delta t$, which reflect more temporal points.

For $c_i = (\rho, P, v_x, v_y)$, we have the following equation to obtain the corresponding values at the right and left interfaces:

$$\begin{aligned} c^L_{i+1/2} &= c_i + \frac{1}{2} minmod[\theta(c_i - c_{i-1}), \frac{1}{2}(c_{i+1} - c_{i-1}), \theta(c_{i+1} - c_i)] \\ c^R_{i+1/2} &= c_{i+1} + \frac{1}{2} minmod[\theta(c_{i+1} - c_i), \frac{1}{2}(c_{i+2} - c_i), \theta(c_{i+2} - c_{i+1})] \end{aligned} \tag{13}$$

where θ is the parameter whose value is between 1 and 2, both of the two methods calculate the same point $C_{i+1/2}$, and

$$minmod(x, y, z) = \frac{1}{4}|sgn(x) + sgn(y)|(sgn(x) + sgn(z))min(|x|, |y|, |z|) \tag{14}$$

The general idea is to find c_i first, and from that, we can compute $c^L_{i+1/2}, c^R_{i+1/2}$ using equation(12). Since $c(0) = \rho$, $c(1) = P$, and $c(3) = v$, we can get the three values from C-half and the three values can be used to represent U (U^L and U^R) and F (F^L and F^R) by the equation (3) and (4). Then, we can use those values to get F-half (F^{HLL}) which can be used to get the time derivative of U. The rest will be the same as the lower order case except for each time step, we need to iterates three times using equation (11). Here is a breakdown of the higher-order simulation process:

$$U^{t=n} \rightarrow (\rho, P, v_x, v_y)_i \rightarrow (\rho, P, v_x, v_y)^{L/R}_{i+1/2} \rightarrow (U, F/R)^{L/R} \rightarrow F_{i+1/2}/G_{i+1/2} \rightarrow \frac{\partial U}{\partial t} \rightarrow U^{t=n+1}$$

2.2.1 Code Implementation

The code for this part was similarly structured as the last part. However, the majority of the functions need to change. Here's a summary of all the function we used and their utilities: `initialize()`, same as before, except more empty numpy arrays are created as it's higher order; `find_c_half(U)` returns $c^{L/R}_{i+1/2}$ for given U; `c_to_ULR_FLR(cL,cR)` returns $(U^{L/R}, F^{L/R})$ for given $c^{L/R}_{i+1/2}$; `find_f_half(UL,UR,FL,FR)` computes $F_{i+1/2}$; `find_u_der(F_half, delta_x)` is similar as before; `find_u(U, U_der,delta_t, ii)` also similar as before; `evolve(i)` loops all functions together as before.

One important point to note is that when calculating the flux between different cells, the nature of this computation inherently results in having one less data point. So when writing the code, it's really easy to cause errors when doing array operations between two quantities. To address this issue, we added an additional zero columns at the boundary to make the two arrays have the same size. This also fix the boundary condition at the same time.

All these functions can be found in the file called "hydro2.py". The main code is in "NEW Higher Order 1D Method.py" file.

2.2.2 Result

For this part, we choose $nx = 300$, which is same as the lower order one, and $nt = 1500$. Using the same nx allows us to compare the difference between higher order and lower order simulation results. Using the same criteria as before, we choose $\Delta x = 1$ and $\Delta t = 0.1$. The boundary condition is imposed by fixing the boundary to be constant. Similar figures on the distribution of pressure, velocity, and density is plotted in figure 4,5,6.

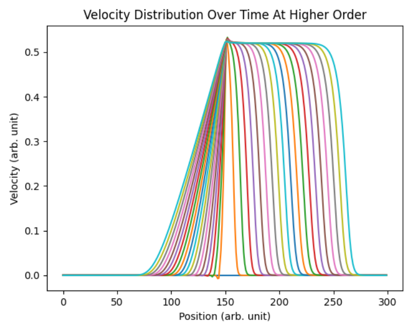


Figure 4: velocity distribution, horizontal axis position, vertical axis velocity. different line represents velocity distribution at different time. nt = 1500, nx = 300, simulated using higher order method

2.3 Comparison Between Lower and Higher Order Method Results

The depicted figures closely resemble the ones from the lower-order method, indicating the correct implementation of our higher-order method. Nevertheless, there are some subtle differences. Firstly, in the higher-order case, the distribution for velocity, density, and pressure all seems to be more gradual. Secondly, for the first few time steps, there's a peak around the middle appears for higher order method, which we are not sure if they are the artifacts or what actually happens.

We also did a computational time comparison. We first set the number of cells nx to be 300, and change the number of time steps nt to see its dependence

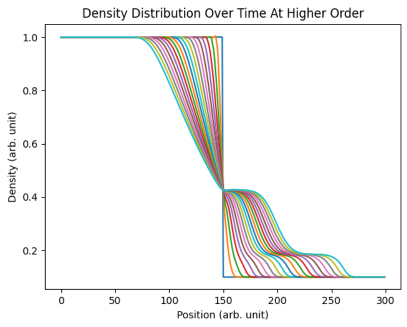


Figure 5: density distribution, horizontal axis position, vertical axis density. different line represents density distribution at different time. nt = 1500, nx = 300, simulated using higher order method

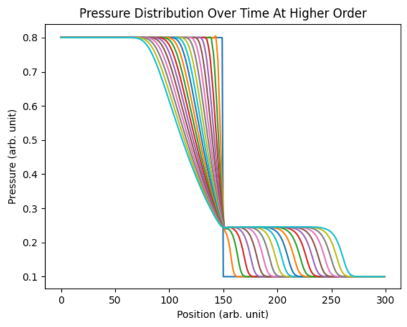
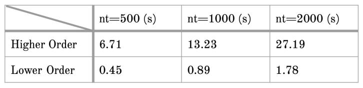


Figure 6: Pressure distribution, horizontal axis position, vertical axis pressure. different line represents density distribution at different time. nt = 1500, nx = 300, simulated using higher order method

on the speed. Figure 7 below shows the result. We observed a linear dependence of nt on computational time. This is expected because nt is basically the number of iteration we choose to compute. We then fixed nt to see the effect of nx on computational time. Figure 8 plotted the result. We see that the computational time is proportional to the square of nx.

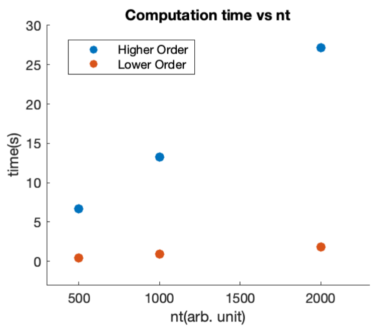
3 2D Higher-Order Method

The previous two parts explores the hydrodynamics in 1D. In this part, we are going to show our result for the simulation in 2D. To utilize 2D Higher-order method, we extend our variables to a higher dimensions. In two dimensions, nx=300:



	nt=500 (s)	nt=1000 (s)	nt=2000 (s)
Higher Order	6.71	13.23	27.19
Lower Order	0.45	0.89	1.78

(a) raw data



(b) nt vs computational time

Figure 7: The dependence of nt on computational time, fixing nx=300

the equation become:

$$\frac{\partial \vec{U}}{\partial t} + \frac{\partial \vec{F}}{\partial x} + \frac{\partial \vec{G}}{\partial y} = 0 \tag{15}$$

where

$$\begin{aligned} \vec{U} &= (\rho, \rho v_x, \rho v_y, E) \\ \vec{F} &= (\rho v_x, \rho v_x^2 + P, \rho v_x v_y, (E+P)v_x) \\ \vec{G} &= (\rho v_y, \rho v_x v_y, \rho v_y^2 + P, (E+P)v_y) \end{aligned} \tag{16}$$

and the total energy is $E = \rho e + \frac{1}{2}\rho(v_x^2 + v_y^2)$. In this case, the time derivative of the conserved quantities is:

$$\frac{\partial \vec{U}}{\partial t} = L(\vec{U}) = -\frac{\vec{F}_{i+\frac{1}{2}} - \vec{F}_{i-\frac{1}{2}}}{\Delta x} - \frac{\vec{G}_{i+\frac{1}{2}} - \vec{G}_{i-\frac{1}{2}}}{\Delta x} \tag{17}$$

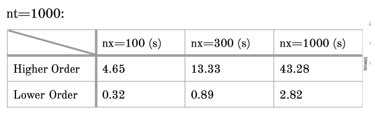
We basically followed the procedure of 1D lower order method, changing the dimension from 1 to 2. $\rho_{L/R}, P_{L/R}, e_{L/R}$ still satisfy the following equations (9). Firstly, based on equation (15) we can find F and G using the equations below:

$$\vec{F} = \begin{pmatrix} U(1) \\ U(1)v_x + P \\ U(1)v_y \\ [U(0)(e + \frac{1}{2}(v_x^2 + v_y^2)) + P]v_x \end{pmatrix} \tag{18}$$

$$\vec{G} = \begin{pmatrix} U(2) \\ U(2)v_x \\ U(2)v_y + P \\ [U(0)(e + \frac{1}{2}(v_x^2 + v_y^2)) + P]v_y \end{pmatrix} \tag{19}$$

where $P = (\gamma - 1)U(0)(\frac{U(3)}{U(0)} - \frac{1}{2}(v_x^2 + v_y^2))$ and $e = \frac{P}{U(0)(\gamma-1)}$.

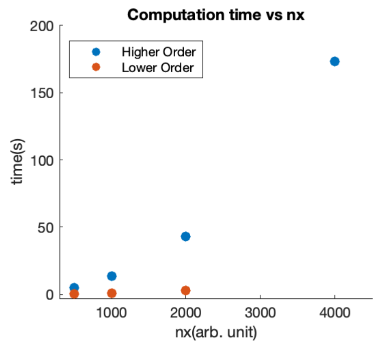
Then, based on equation (5), we can find F-half and G-half, which can be plugged into equation (16) to get $\frac{\partial U}{\partial t}$



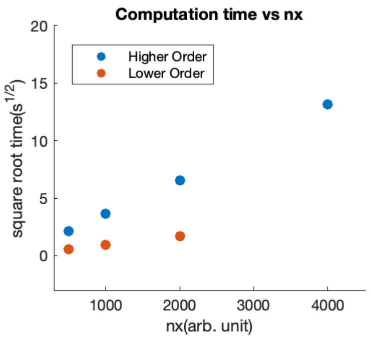
nt=1000:

	nx=100 (s)	nx=300 (s)	nx=1000 (s)
Higher Order	4.65	13.33	43.28
Lower Order	0.32	0.89	2.82

(a) raw data



(b) nx vs computational time



(c) $nx^{0.5}$ vs computational time

Figure 8: The dependence of nx on computational time, fixing nt=1000

3.1 Code Implementation

The logic for code implementation remains consistent with the previous approach. However, given the shift to two dimensions, adjustments have been made to the dimensions of our variables, and additional calculations are now required. There's `initialize()`, `find_FG(U)`, `find_f_half(U,F)`, `find_u_der(F_half, G_half, delta_x)`, `find_u(U, U_der,delta_t,ii)`, `evolve(i)`. Each function has similar uses as before with slight adjustment for 2D. For the boundary condition, we set all variables to be constant, same as before. The initial condition is defined using the following:

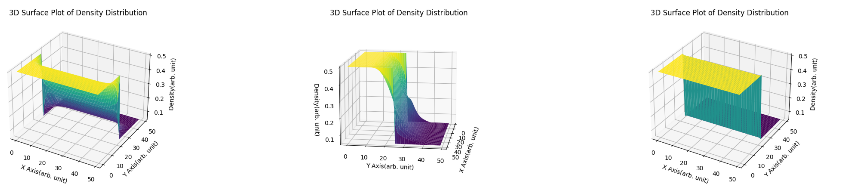
```
for i in range(nx):
   if i < nx / 2 :
       rho_L = 0.5
       P_L = 0.8
       U[i,:, 0:4] = np.array([rho_L, rho_L*v_x0,
                   rho_L*v_y0, P_L])
   else:
```

```
rho_R = 0.05
P_R = 0.1
U[i,:, 0:4] = np.array([rho_R, rho_R*v_x0,
            rho_R*v_y0, P_R])
```

All these functions can be found in the file called "hydro3.py". The main code is in "new-part3.py" file.

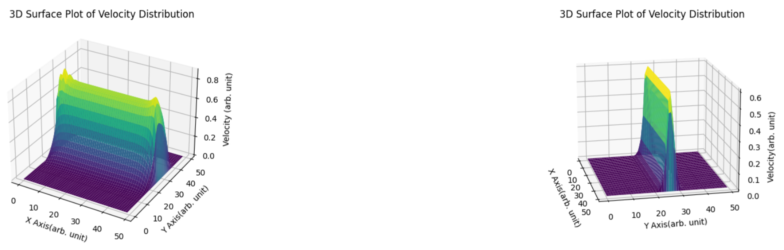
3.2 Result

In this section, due to the transition to two dimensions, the computation time has increased exponentially. To manage this, we have opted to reduce the size of the space and time for the simulation by setting $nx = 50$ and $nt = 300$. Using the same criteria as before, we choose $\Delta x = 1$ and $\Delta t = 0.1$. Here we plotted the pressure, velocity, and density distribution at different times for the given initial conditions above in figure 9, 10, 11.



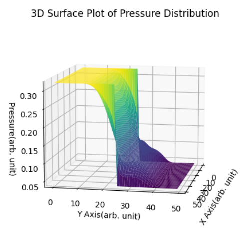
(a) nt=300 (b) nt=300, another view (c) nt=0, initial condition

Figure 9: 3D Plots for Density Distribution, nx=50. Two horizontal axis represents the position, vertical axis represents density

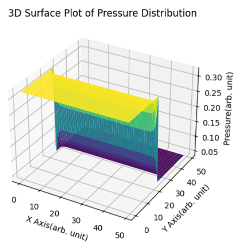


(a) nt=300 (b) nt=50

Figure 10: 3D Plots for Velocity Distribution, nx=50. Two horizontal axis represents the position, vertical axis represents velocity magnitude



(a) nt=300



(b) nt=50

Figure 11: 3D Plots for Pressure Distribution, nx=50. Two horizontal axis represents the position, vertical axis represents pressure

3.3 Discussion

The obtained results align closely with our expectations. When examining the 3D plots from a lateral perspective, they exhibit striking similarities to their 2D counterparts. Notably, the pressure distribution reveals two distinct ”bumps” on the right, similar to the pattern observed in the 2D scenarios. Similarly, the density distribution features a single ”bump.” An observation is that the at the boundary is always the same as the initial condition, which is on purpose when we define the boundary conditions. Overall, these findings validate the coherence between our expectations and the simulated outcomes, contributing valuable insights into the behavior of the system for two dimension.

4 Conclusion

In conclusion, this hydrodynamics project aimed to explore the simulation of fluid dynamics in both 1D and 2D systems using computational methods such as the Harten-Lax-van Leer (HLL) approximation and the third-order Runge-Kutta algorithm. The investigation was structured in three main parts, starting with a focus on lower-order methods in a one-dimensional system, progressing to higher-order approximations, and finally extending the simulations to a two-dimensional system.

In the first part, we successfully implemented the lower-order method to solve the Sod Shock Tube Problem in a one-dimensional setting. The results demonstrated the expected behavior of shock waves and waves propagating through the fluid. The implementation involved solving the hydrodynamic equations in conservative form using the HLL flux approximation for time evolution.

Subsequently, in the second part, we extended the methodology to a higher-order 1D simulation. The third-order Runge-Kutta algorithm was employed to reduce the approximation error, resulting in smoother and more accurate solutions. A comparison between the lower and higher-order methods revealed differences in wave propagation speed and distribution smoothness, affirming the benefits of employing higher-order algorithms.

The final part focused on the simulation of a two-dimensional hydrodynamics system. The extension to two dimensions required adjustments to the conservation equations, fluxes, and numerical methods. The simulations provided insights into the evolution of density, velocity, and pressure in a two-dimensional space over time, and we visualized them as a 3D graph.

In summary, this project presented the application of numerical methods in solving hydrodynamic problems, emphasizing the transition from lower to higher-order approximations and extending the simulations to a two-dimensional system. The obtained results contribute to a better understanding of fluid behavior under different conditions and pave the way for more sophisticated simulations in fluid dynamics.

5 Just For Fun

Here we tried different initial conditions for the 2D simulation to see how does the system changes under different conditions, and validate the accuracy of our simulation at the same time. We tried out additional three different initial conditions(quarter, slanted, and circle). In the code, we implemented as the following:

```
for i in range(nx):
    for j in range(ny):
        if i < nx / 2 and j < ny / 2: #quarter
```

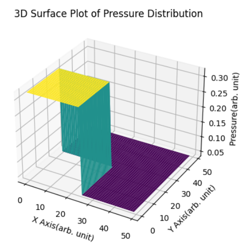
or

```
        if i + j < nx: #slanted
```

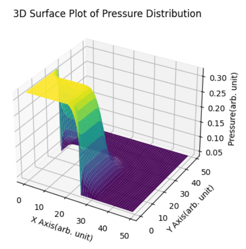
or

```
        if (i-25)**2 + (j-25)**2 < 36: #circle
            rho_L = 0.5
            P_L = 0.8
            U[i,j, 0:4] = np.array([rho_L, rho_L*v_x0,
                rho_L*v_y0, P_L])
        else:
            rho_R = 0.05
            P_R = 0.1
            U[i,j, 0:4] = np.array([rho_R, rho_R*v_x0,
                rho_R*v_y0, P_R])
```

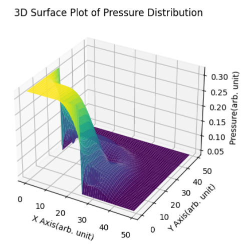
Here's what they looks like:



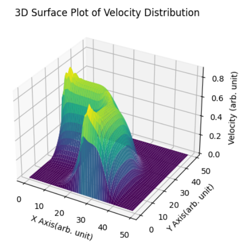
(a) nt=0, pressure distribution



(b) nt=150, pressure distribution

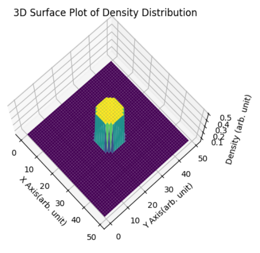


(c) nt=300, pressure distribution

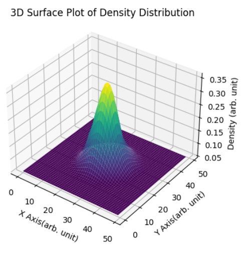


(d) nt=300, velocity distribution

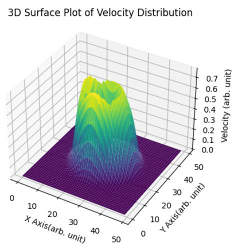
Figure 12: 3D Plots for quarter initial condition, nx=50. Two horizontal axis represents the position, vertical axis represents respective quantity



(a) nt=0, density distribution

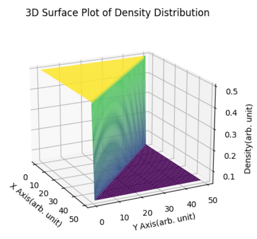


(b) nt=150, density distribution

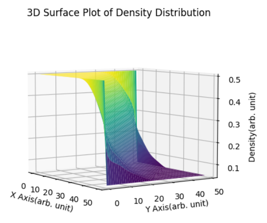


(c) nt=150, velocity distribution

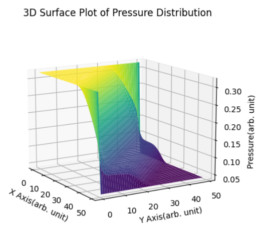
Figure 13: 3D Plots for circle initial condition, nx=50. Two horizontal axis represents the position, vertical axis represents respective quantity



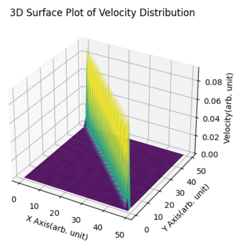
(a) nt=0, density distribution



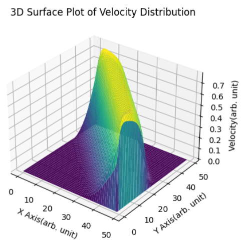
(b) nt=300, density distribution



(c) nt=300, pressure distribution



(d) nt=0, velocity distribution



(e) nt=300, velocity distribution

Figure 14: 3D Plots for slanted initial condition, nx=50. Two horizontal axis represents the position, vertical axis represents respective quantity