Hughe's flow Third Attempt

Defining the Domain

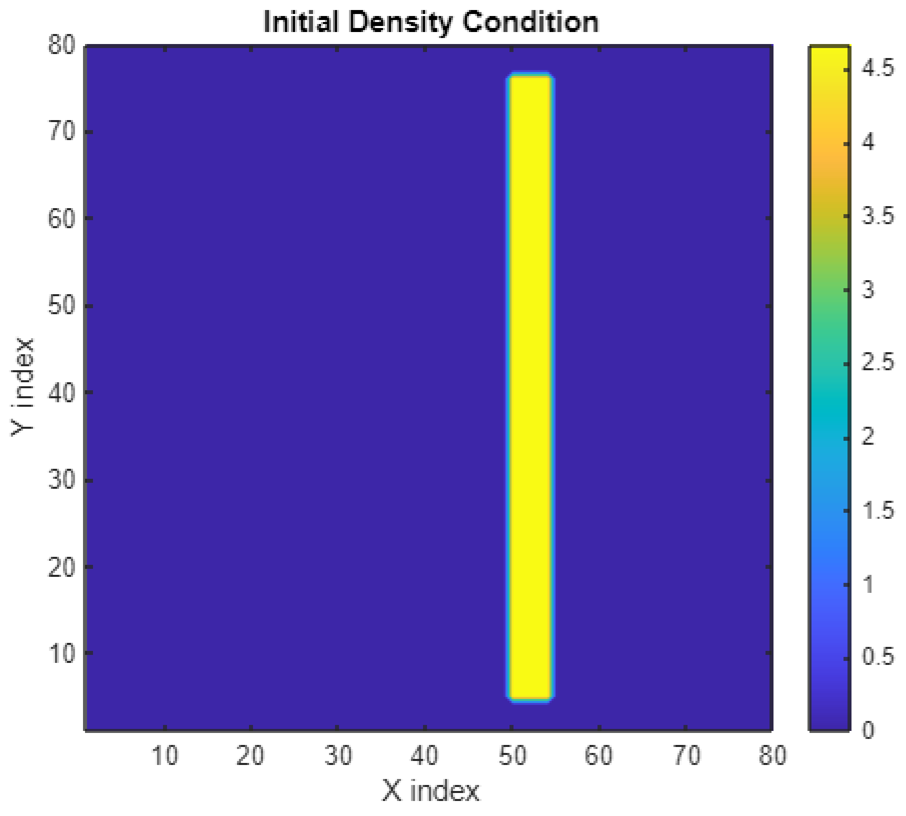
```
Nx = 80; Ny = 80;
Lx = 20; Ly = 20;
dx = Lx/Nx; dy = Ly/Ny;
[x, y] = meshgrid(linspace(dx,Lx,Nx), linspace(dy,Ly,Ny));
```

Setting the Parameters

```
Vmax = 1.4;
rho_max = 5.6;
CFL = 0.1;
```

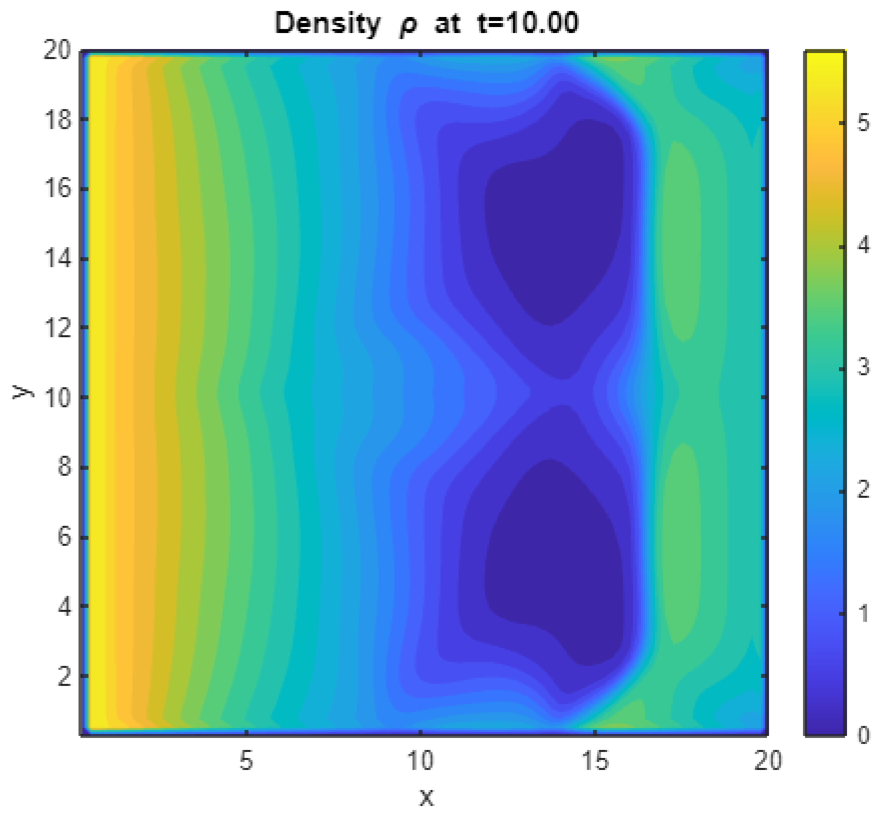
Initialising the Initial Condition

```

% Assigning zero density everywhere
rho = zeros(Nx,Ny);
% Initial density condition
rho(5:Ny-4,50:54) = 4.9;
rho = min(rho,rho_max);
% Plotting the result.
figure;
contourf(rho, 20, 'LineColor', 'none');
colorbar; axis equal tight;
xlabel('X index');
ylabel('Y index');
title('Initial Density Condition');
```

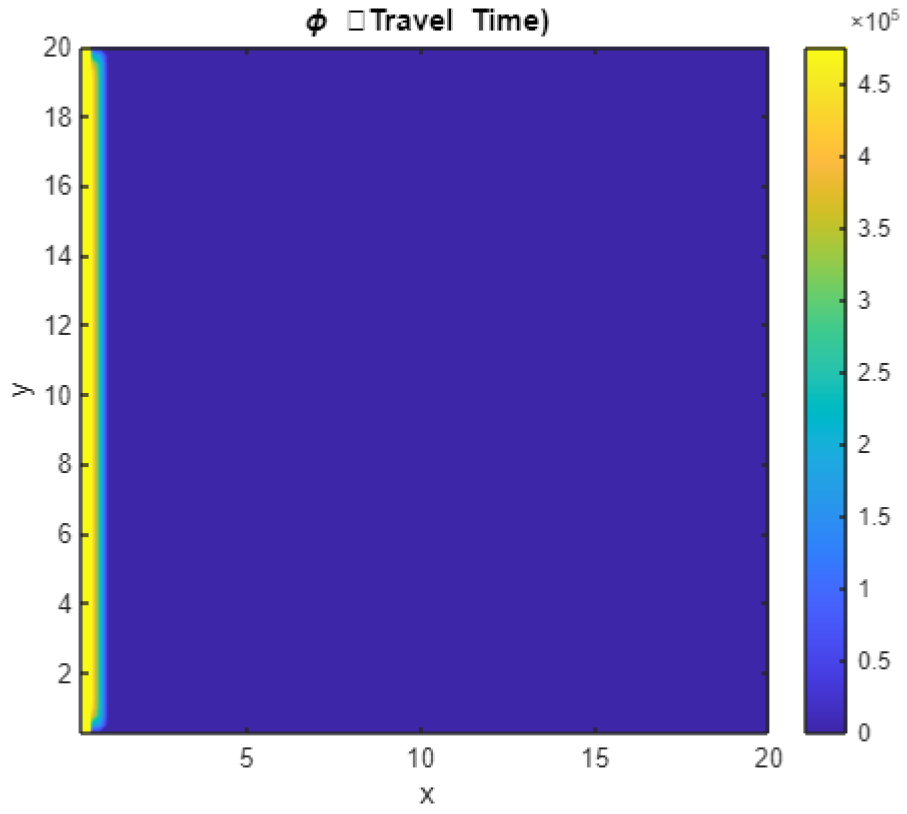


The main function for time marching

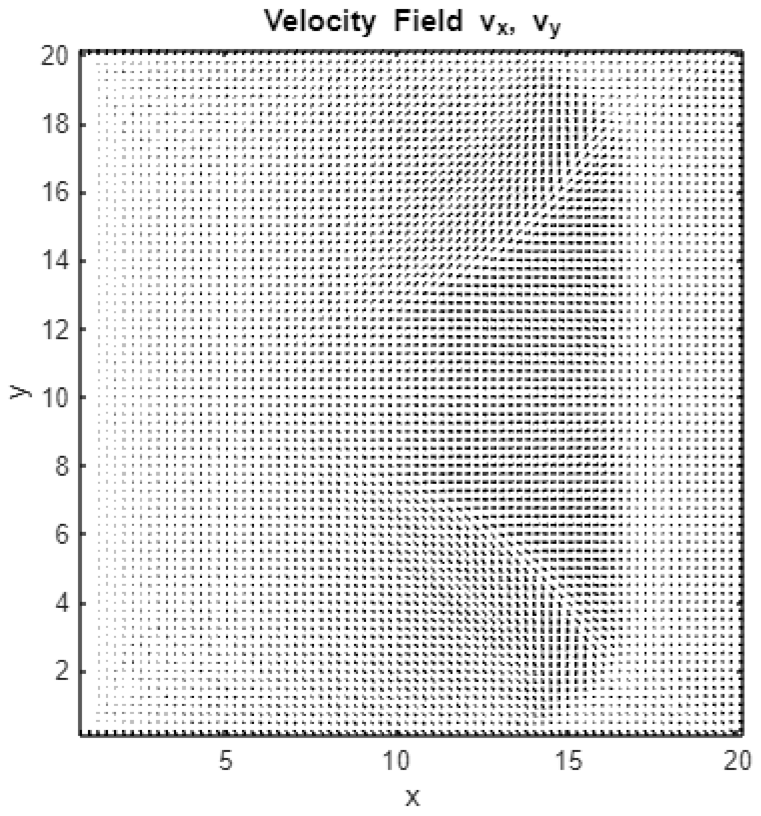
```
Tfinal = 10;
t = 0;
z = 0.1;
while t < Tfinal
    % Speed field
    f = Vmax*(1 - rho/rho_max);
    % To speed doesn't become zero completely
    f(f<1e-6) = 1e-6;
    % Solve eikonal equation
    phi = fast_sweeping(1./f,dy,dx,Nx,Ny);
    % Direction field
    [phix, phiy] = gradient(phi, dx, dy);
    grad_mag = sqrt(phix.^2 + phiy.^2) + 1e-12;
    dirx = -phix ./ grad_mag;
    diry = -phiy ./ grad_mag;
    % Velocity field
    vx = f .* dirx;
    vy = f .* diry;
    % Boundary conditions (No Penetration Condition)
    %vx(:,1) = 0;
    %vx(:,end) = Vmax;
    %vy(1,:) = 0;
    %vy(end,:) = 0;
    % Time step
    maxspeed = max(max(sqrt(vx.^2 + vy.^2)));
    dt = CFL * min(dx,dy) / maxspeed;
    if t+dt > Tfinal, dt = Tfinal-t; end
    % Update rho
    rho = (z)*Lax_Wendroff_update(rho, vx, vy, dx, dy, dt,rho_max) + (1-
z)*upwind_update(rho, vx, vy, dx, dy, dt,rho_max);
    % Advance time
    t = t + dt;
    % Plots
    if mod(round(t/dt),2) == 0
        % Density (phi)
        figure(1);
        contourf(x, y, rho, 20, 'LineColor', 'none');
        colorbar;
        caxis([0 rho_max]);
        title(sprintf('Density \\rho at t=%.2f', t));
        xlabel('x'); ylabel('y');
        axis equal tight;
        % Potential (phi)
        figure(2);
        contourf(x, y, phi, 20, 'LineColor', 'none');
        colorbar; title('\phi ⍰Travel Time)');
        xlabel('x'); ylabel('y');
        axis equal tight;
        % Velocity field
        figure(3);
        quiver(x, y, vx, vy, 0.5, 'k');
        title('Velocity Field \bf{v_x, v_y}');
        xlabel('x'); ylabel('y');
        axis equal tight;
        drawnow;
    end
end
```



Density ρ at t=10.00
y
x



φ Travel Time)
×10^5
y
x

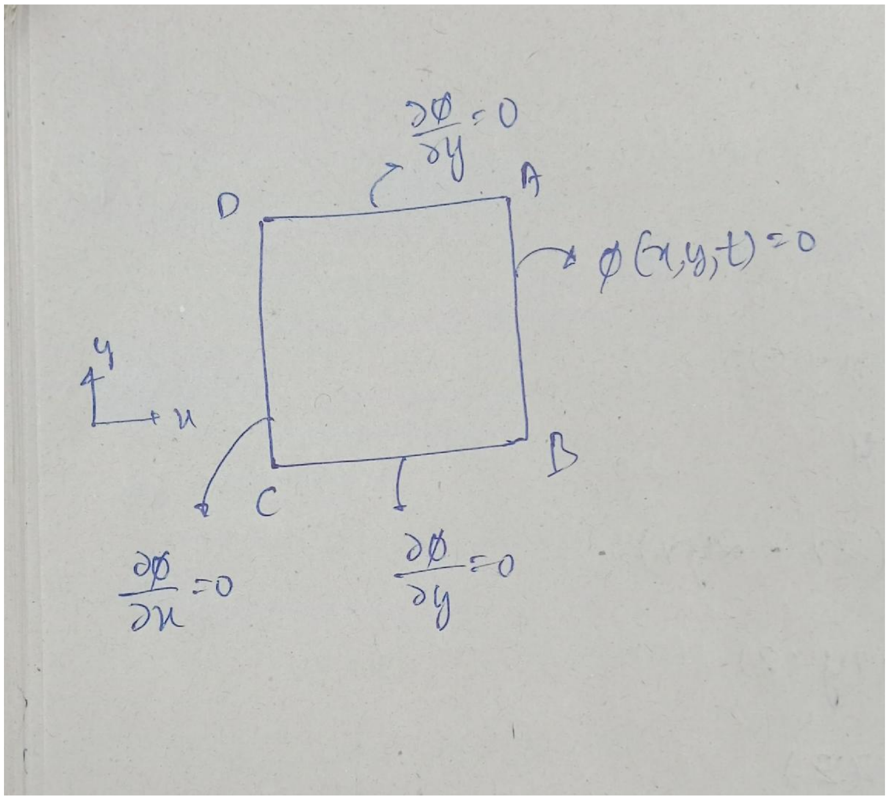


Fast sweep Algorithm

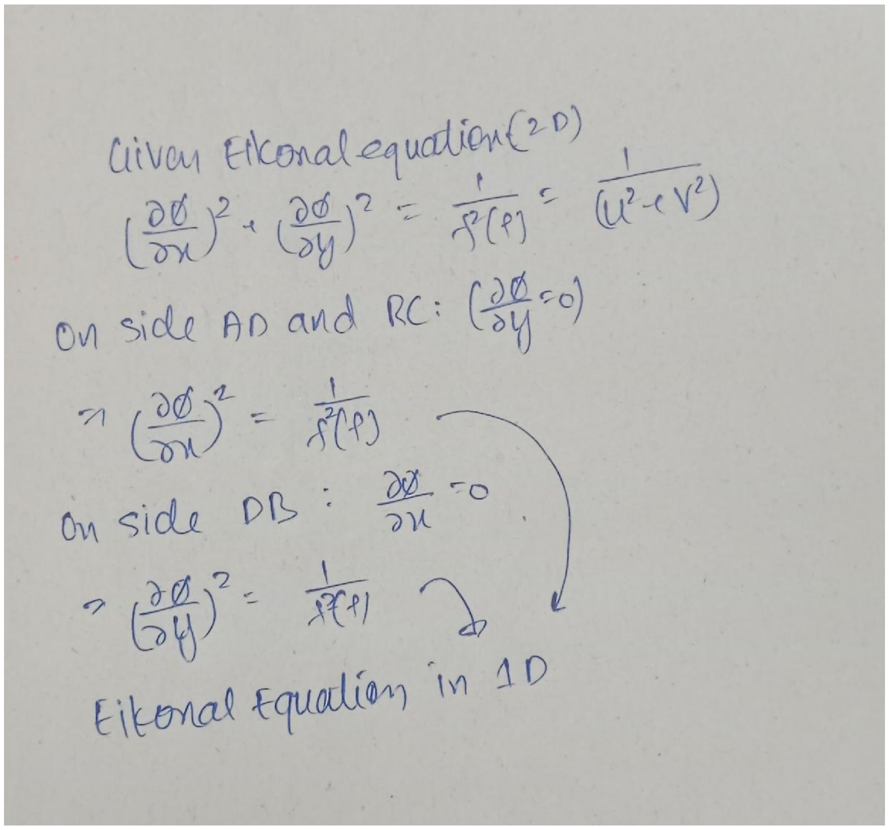
So eher is what the algorithm does. It first assigns a very large value of the $\phi(x, y, t)$, surely more than the maximum value that could be obtained. And then The neumann boundary condition is imposed. And for that the following approach is choosen.

Neumann Boundary Condition

So in order to apply the neumann boundary condition for the normal gradient of $\phi(x, y, t)$ to be equa to zero, I will be solving some different kind of governign equation at the boundary that will incorporate the normal gradient equal to zero. Schematic of the problem is given below:


$\frac{\partial \phi}{\partial y} = 0$
A
D
$\phi(x,y,t) = 0$
y
x
B
C
$\frac{\partial \phi}{\partial x} = 0$
$\frac{\partial \phi}{\partial y} = 0$

Here the side AB will have dirichlet condition and hence doesnt have to worry about. But in the side AD and CB, the derivative of phi is equal to zero w.r.t to normal which in this case, is in y direction, and similarly the same is true but in x direction at the CD side. So the **Eikonal equation** that i will be using to update the values at the boundary will be as shown below:



Given Eikonal equation (2D)

$$\left(\frac{\partial \phi}{\partial x}\right)^2 + \left(\frac{\partial \phi}{\partial y}\right)^2 = \frac{1}{f^2(\rho)} = \frac{1}{(u^2+v^2)}$$

On side AD and BC: $\left(\frac{\partial \phi}{\partial y} = 0\right)$

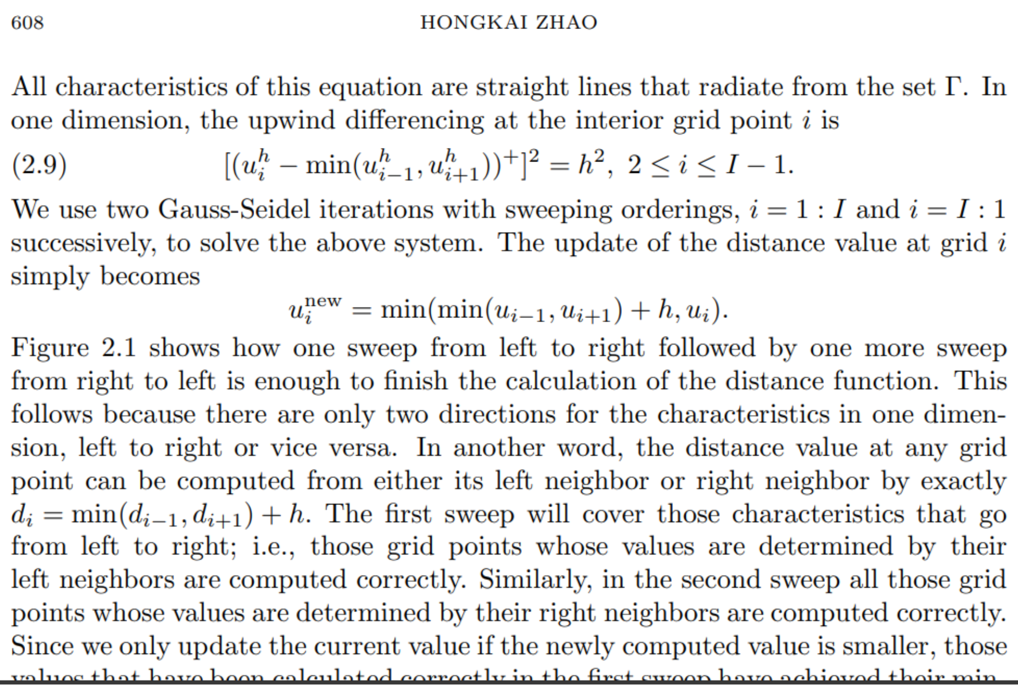
$$\Rightarrow \left(\frac{\partial \phi}{\partial x}\right)^2 = \frac{1}{f^2(\rho)}$$

On side DB : $\frac{\partial \phi}{\partial x} = 0$

$$\Rightarrow \left(\frac{\partial \phi}{\partial y}\right)^2 = \frac{1}{f^2(\rho)}$$

Eikonal Equation in 1D

And interestingly this is **Eikonal Equation in 1D**. And in the paper link to which was provided above, The author has tried solving the eikonal equation in 1D but the he solved it for **Distance Function**. The boundary condition for solving these new set of equation is that at x=1, or the right most tip of the domain (bot at y=0, and y=1) will have value of $\phi(x, y, t)$ equal to zero, which is a dirichlet boundary condition.



608 HONGKAI ZHAO

All characteristics of this equation are straight lines that radiate from the set Γ. In one dimension, the upwind differencing at the interior grid point i is

$$[(u_i^h - \min(u_{i-1}^h, u_{i+1}^h))^+]^2 = h^2, \; 2 \le i \le I-1. \tag{2.9}$$

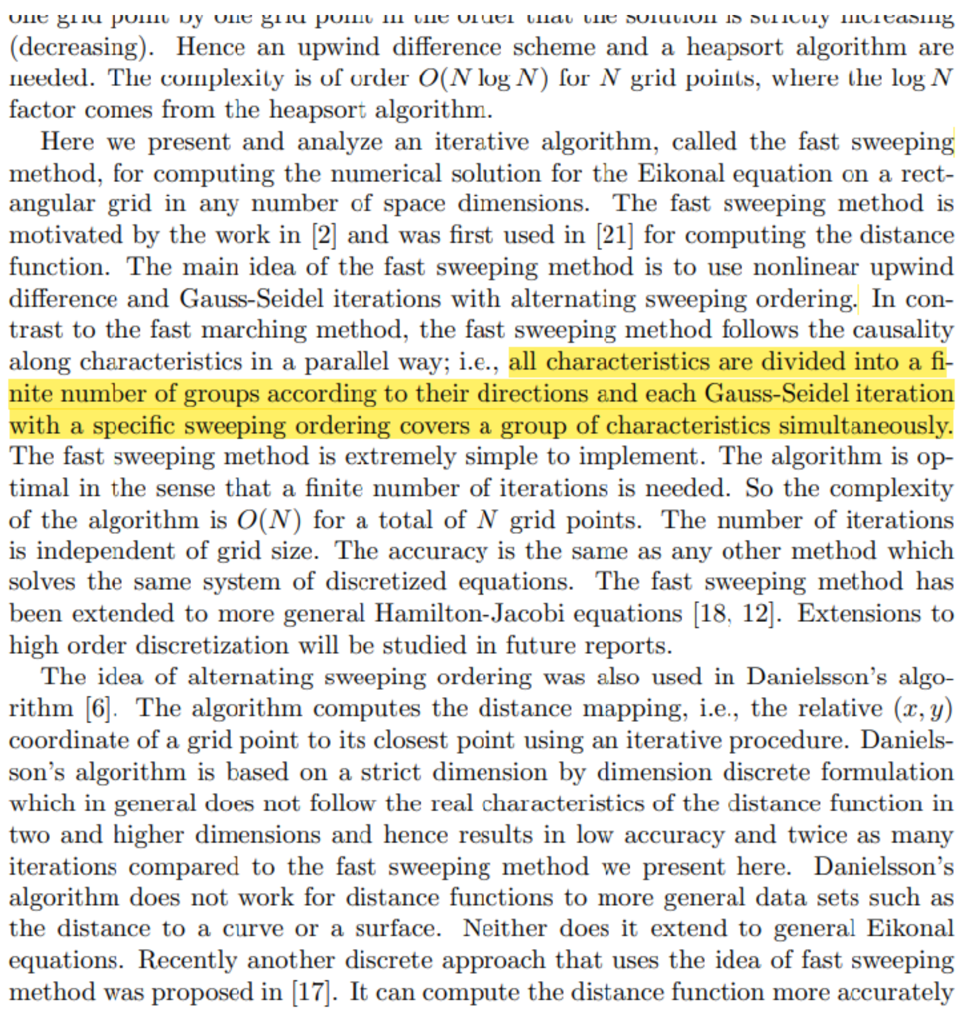
We use two Gauss-Seidel iterations with sweeping orderings, $i = 1 : I$ and $i = I : 1$ successively, to solve the above system. The update of the distance value at grid i simply becomes

$$u_i^{\text{new}} = \min(\min(u_{i-1}, u_{i+1}) + h, u_i).$$

Figure 2.1 shows how one sweep from left to right followed by one more sweep from right to left is enough to finish the calculation of the distance function. This follows because there are only two directions for the characteristics in one dimension, left to right or vice versa. In another word, the distance value at any grid point can be computed from either its left neighbor or right neighbor by exactly $d_i = \min(d_{i-1}, d_{i+1}) + h$. The first sweep will cover those characteristics that go from left to right; i.e., those grid points whose values are determined by their left neighbors are computed correctly. Similarly, in the second sweep all those grid points whose values are determined by their right neighbors are computed correctly. Since we only update the current value if the newly computed value is smaller, those values that have been calculated correctly in the first sweep have achieved their min

Now what i aim to do is just use the same scheme as proposed by the **Hongkai Zhao** paper, and use it to somehow simulate the neumann boundary condiuiton. It is important to mention that:

1. For each sweep, I have first update the boundary points using the above notion. And it is important to note that to update the boundary points using the above scheme, I am required to have only neighbouring boundary points in the previous time step. And not the interior point.
2. And after the boundary points are updated, I am updating the interior points using the boundary points, and during this sweep, the value of boundary points are not updated. And this completes each sweep. It is also important to mention that the sweep has to be done from all the directions. This was also stated in the paper, and it says it is important to preserve the causloity of solution.


one grid point by one grid point in the order that the solution is strictly increasing (decreasing). Hence an upwind difference scheme and a heapsort algorithm are needed. The complexity is of order $O(N \log N)$ for N grid points, where the $\log N$ factor comes from the heapsort algorithm.

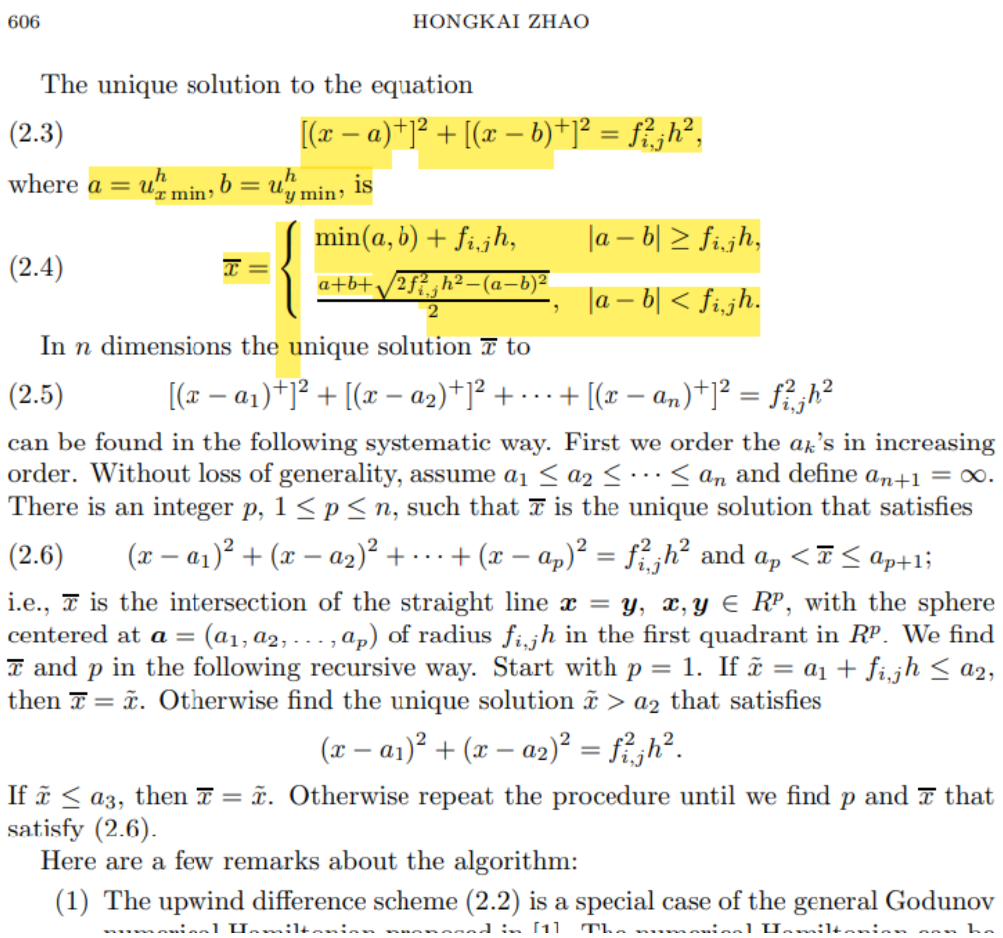
Here we present and analyze an iterative algorithm, called the fast sweeping method, for computing the numerical solution for the Eikonal equation on a rectangular grid in any number of space dimensions. The fast sweeping method is motivated by the work in [2] and was first used in [21] for computing the distance function. The main idea of the fast sweeping method is to use nonlinear upwind difference and Gauss-Seidel iterations with alternating sweeping ordering. In contrast to the fast marching method, the fast sweeping method follows the causality along characteristics in a parallel way; i.e., all characteristics are divided into a finite number of groups according to their directions and each Gauss-Seidel iteration with a specific sweeping ordering covers a group of characteristics simultaneously. The fast sweeping method is extremely simple to implement. The algorithm is optimal in the sense that a finite number of iterations is needed. So the complexity of the algorithm is $O(N)$ for a total of N grid points. The number of iterations is independent of grid size. The accuracy is the same as any other method which solves the same system of discretized equations. The fast sweeping method has been extended to more general Hamilton-Jacobi equations [18, 12]. Extensions to high order discretization will be studied in future reports.

The idea of alternating sweeping ordering was also used in Danielsson's algorithm [6]. The algorithm computes the distance mapping, i.e., the relative (x, y) coordinate of a grid point to its closest point using an iterative procedure. Danielsson's algorithm is based on a strict dimension by dimension discrete formulation which in general does not follow the real characteristics of the distance function in two and higher dimensions and hence results in low accuracy and twice as many iterations compared to the fast sweeping method we present here. Danielsson's algorithm does not work for distance functions to more general data sets such as the distance to a curve or a surface. Neither does it extend to general Eikonal equations. Recently another discrete approach that uses the idea of fast sweeping method was proposed in [17]. It can compute the distance function more accurately

So this is how the neumann condition is handled. And also just to clarify when calculating $\phi(x, y, t)$, we jsut need to speed at all the location which is already known from density.

Calculating the value of ϕ in the interior points

After calculating the value of ϕ at th boundary, we move on to calculating the values of ϕ at the boundary. And this is done as follows:



606 HONGKAI ZHAO

The unique solution to the equation

$$[(x-a)^+]^2 + [(x-b)^+]^2 = f_{i,j}^2 h^2, \tag{2.3}$$

where $a = u^h_{x\,\min}, b = u^h_{y\,\min}$, is

$$\overline{x} = \begin{cases} \min(a,b) + f_{i,j}h, & |a-b| \geq f_{i,j}h, \\ \frac{a+b+\sqrt{2f_{i,j}^2h^2-(a-b)^2}}{2}, & |a-b| < f_{i,j}h. \end{cases} \tag{2.4}$$

In n dimensions the unique solution $\overline{x}$ to

$$[(x-a_1)^+]^2 + [(x-a_2)^+]^2 + \cdots + [(x-a_n)^+]^2 = f_{i,j}^2 h^2 \tag{2.5}$$

can be found in the following systematic way. First we order the a_k's in increasing order. Without loss of generality, assume $a_1 \leq a_2 \leq \cdots \leq a_n$ and define $a_{n+1} = \infty$. There is an integer p, $1 \leq p \leq n$, such that $\overline{x}$ is the unique solution that satisfies

$$(x-a_1)^2 + (x-a_2)^2 + \cdots + (x-a_p)^2 = f_{i,j}^2 h^2 \text{ and } a_p < \overline{x} \leq a_{p+1}; \tag{2.6}$$

i.e., $\overline{x}$ is the intersection of the straight line $\boldsymbol{x} = \boldsymbol{y}$, $\boldsymbol{x}, \boldsymbol{y} \in R^p$, with the sphere centered at $\boldsymbol{a} = (a_1, a_2, \ldots, a_p)$ of radius $f_{i,j}h$ in the first quadrant in R^p. We find $\overline{x}$ and p in the following recursive way. Start with $p = 1$. If $\tilde{x} = a_1 + f_{i,j}h \leq a_2$, then $\overline{x} = \tilde{x}$. Otherwise find the unique solution $\tilde{x} > a_2$ that satisfies

$$(x-a_1)^2 + (x-a_2)^2 = f_{i,j}^2 h^2.$$

If $\tilde{x} \leq a_3$, then $\overline{x} = \tilde{x}$. Otherwise repeat the procedure until we find p and $\overline{x}$ that satisfy (2.6).

Here are a few remarks about the algorithm:

(1) The upwind difference scheme (2.2) is a special case of the general Godunov numerical Hamiltonian proposed in [1]. The numerical Hamiltonian can be

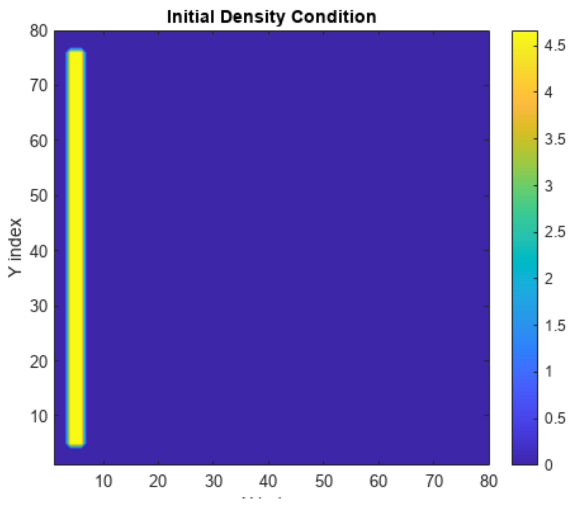
```
function phi = fast_sweeping(invf,dy,dx,Nx,Ny)
    %Boundary Condition
    phi = inf(Ny,Nx);
    for sweep=1:200
        % The limit starts from column and row 2, and ends at column and row
        % This excludes the boundary points.
        % Updating the boundary points. For doing so i am going to sweep in
        % that too
        for boundary_sweep = 1:2
            % Updating the side AB
            phi(:,Nx) = 0;
            % Updating the side BC
            for i = 2 : Nx-1
```

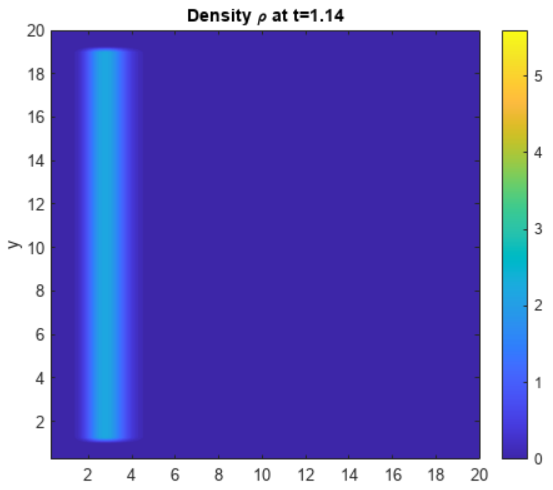
```
                phi(1,i) = min((min(phi(1,i-1),phi(1,i+1)) +
invf(1,i)*dx),phi(1,i));
            end
            for i = Nx-1 : 2 : -1
                phi(1,i) = min((min(phi(1,i-1),phi(1,i+1)) +
invf(1,i)*dx),phi(1,i));
            end

            % Updating the side AD
            for i = 2 : Nx-1
                phi(Ny,i) = min((min(phi(Ny,i-1),phi(Ny,i+1)) +
invf(Ny,i)*dx),phi(Ny,i));
            end
            for i = Nx-1 : 2 : -1
                phi(Ny,i) = min((min(phi(Ny,i-1),phi(Ny,i+1)) +
invf(Ny,i)*dx),phi(Ny,i));
            end

            % Updating the side CD
            for j = 2 : Ny-1
                phi(j,1) = min((min(phi(j-1,1),phi(j+1,1))+invf(j,1)*dy),phi(j,1));
            end
            for j = Ny-1 : 2 : -1
                phi(j,1) = min((min(phi(j-1,1),phi(j+1,1))+invf(j,1)*dy),phi(j,1));
            end
        end

        % Now we are done with updating the boundary phi values with the
        % neumann type boundary condition at the three sides and one
        % dirichlet type boundary condition at the other side AB

        % Sweep direction 1
        for i=2:Nx-1
            for j=2:Ny-1
                % Calculating the minimum value of phi, in it's neighbor in x
direction
                a = min(phi(j-1,i), phi(j+1,i));
                % Calculating the minimum value of phi, in it's neighbor in y
direct
                b = min(phi(j,i-1), phi(j,i+1));
                if abs(a-b) >= invf(j,i)*dx
                    %phi = min(a,b) + (step_time)
                    phi(j,i) = min(a,b) + invf(j,i)*dx;
                else
                    %phi = solution of quadratic equation
                    phi(j,i) = (a+b + sqrt(2*(invf(j,i)*dx)^2 - (a-b)^2))/2;
                end
            end
        end
        % Sweep direction 2
        for i = Nx-1 : 2 : -1
            for j = 2 : Ny-1
                % Calculating the minimum value of phi, in it's neighbour in x
direction
                a = min(phi(j-1,i), phi(j+1,i));
                % Calculating the minimum value of phi, in it's neighbour in y
direct
                b = min(phi(j,i-1), phi(j,i+1));
                if abs(a-b) >= invf(j,i)*dx
                    %phi = min(a,b) + (step_time)
                    phi(j,i) = min(a,b) + invf(j,i)*dx;
                else
                    %phi = solution of quadratic equation
                    phi(j,i) = (a+b + sqrt(2*(invf(j,i)*dx)^2 - (a-b)^2))/2;
                end
            end
        end

        % Sweep direction 3
        for i = 2 : Nx-1
            for j = Ny-1 : 2 : -1
                % Calculating the minimum value of phi, in it's neighbour in x
direction
                a = min(phi(j-1,i), phi(j+1,i));
                % Calculating the minimum value of phi, in it's neighbour in y
direct
                b = min(phi(j,i-1), phi(j,i+1));
                if abs(a-b) >= invf(j,i)*dx
                    %phi = min(a,b) + (step_time)
                    phi(j,i) = min(a,b) + invf(j,i)*dx;
                else
                    %phi = solution of quadratic equation
                    phi(j,i) = (a+b + sqrt(2*(invf(j,i)*dx)^2 - (a-b)^2))/2;
                end
            end
        end

        % Sweep direction 4
        for i = Nx-1 : 2 : -1
            for j = Ny-1 : 2 : -1
                % Calculating the minimum value of phi, in it's neighbour in x
direction
                a = min(phi(j-1,i), phi(j+1,i));
                % Calculating the minimum value of phi, in it's neighbour in y
direct
                b = min(phi(j,i-1), phi(j,i+1));
                if abs(a-b) >= invf(j,i)*dx
                    %phi = min(a,b) + (step_time)
                    phi(j,i) = min(a,b) + invf(j,i)*dx;
                else
                    %phi = solution of quadratic equation
                    phi(j,i) = (a+b + sqrt(2*(invf(j,i)*dx)^2 - (a-b)^2))/2;
                end
            end
        end
    end
end
```

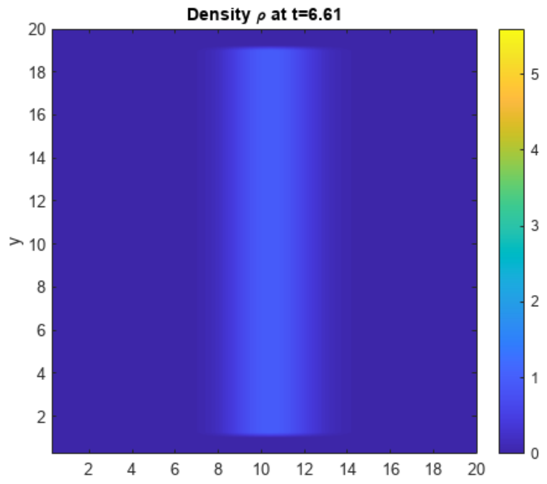
Up-winding scheme

So here is the thing with **First order upwind scheme**. It has huge amounts of diffusion, which is also why my solution keeps on getting diffused. So maybe I will have to switch my schemes. And for that i am going to use a better scheme which is **Lax–Wendroff**, and this is used after getting the results from first order up-winding scheme. The results for first order up-winding scheme is shown below, and this will help us understand how the diffusion is ruining the solution in Hughes,flow:

So the problem was that I have same rectangular density concentration, and the a constant velocity is given to the medium in x direction. What's expected was a clean advection with no change in shape or even concentration, since all the points inside the concentration will be traveling with same speed in same direction. But what was observed was enough to convince that diffusion was creating the problem in numeric simulation.

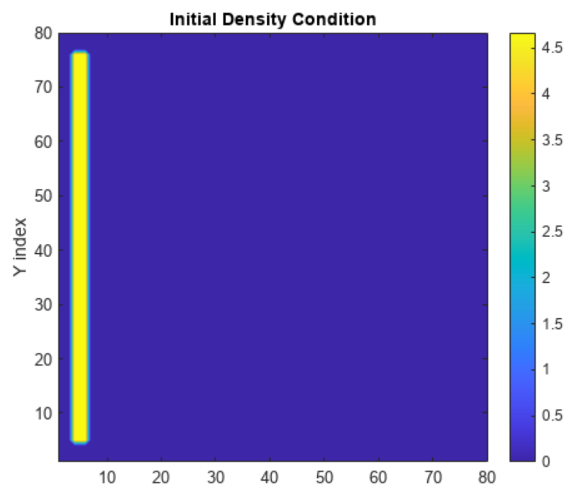


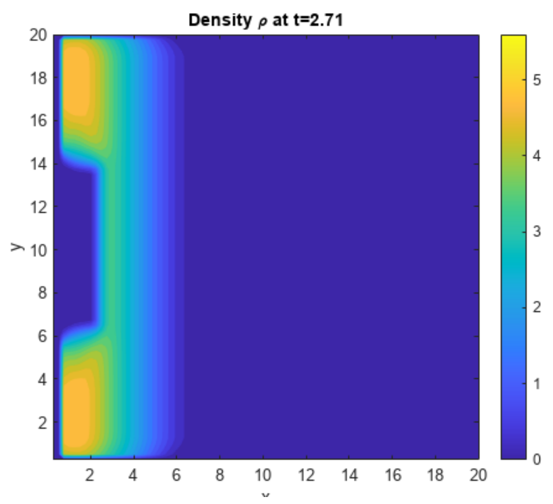


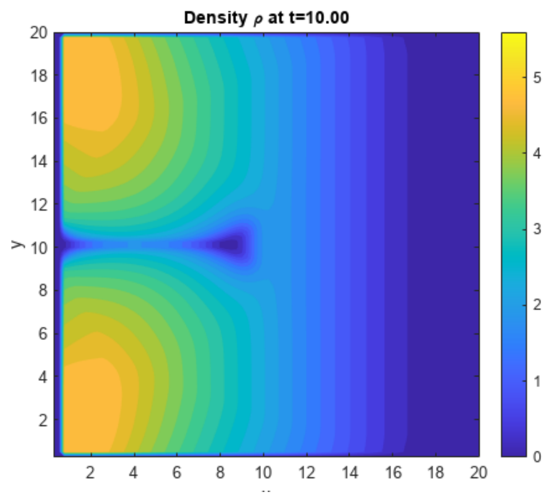


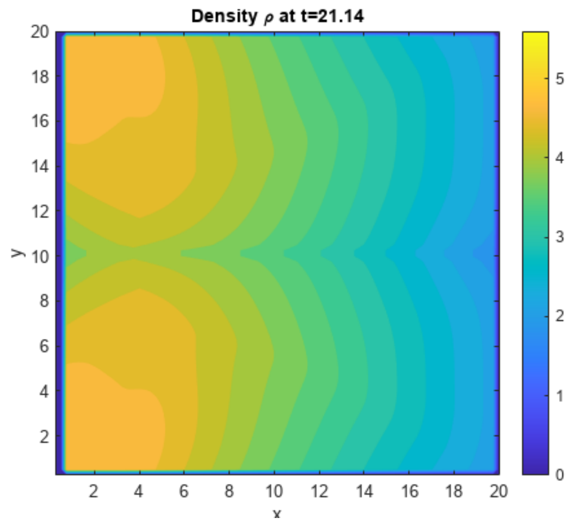
It could be clearly seen that numeric diffusion is the cause of problem. So i have decided to change the scheme, and the below function `Lax_Wendroff_update` is the one which will be used to solve the advection problem with lax wendroff scheme.

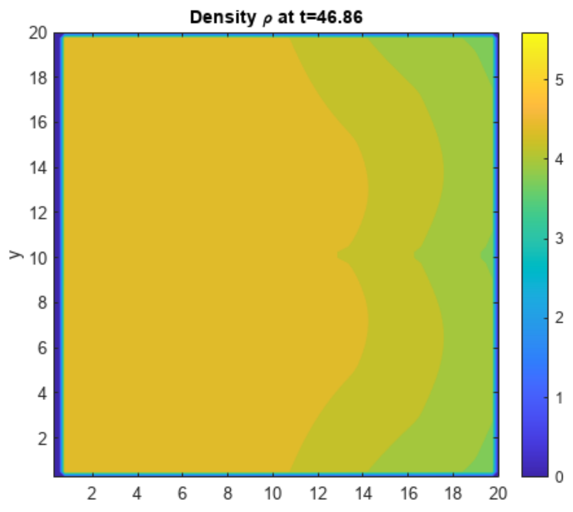
This is also evident with the solution of Hughes' flow where the numeric diffusion has cause the solution to spread through out the domain.











```
function rho_new = upwind_update(rho, vx, vy, dx, dy, dt,rho_max)
    [Ny, Nx] = size(rho);
    % Initalizing the flux values to be zero intially.
    flux_xp = zeros(Ny,Nx);
    flux_xm = zeros(Ny,Nx);
    flux_yp = zeros(Ny,Nx);
    flux_ym = zeros(Ny,Nx);
    % Upwinding scheme implementation
    for j = 2 : Ny-1
        for i = 2 : Nx-1
            % x-direction fluxes
            flux_xp(j,i) = max(vx(j,i),0)*rho(j,i)   + min(vx(j,i),0)*rho(j,i+1);
            flux_xm(j,i) = max(vx(j,i),0)*rho(j,i-1) + min(vx(j,i),0)*rho(j,i);

            % y-direction fluxes
            flux_yp(j,i) = max(vy(j,i),0)*rho(j,i)   + min(vy(j,i),0)*rho(j+1,i);
            flux_ym(j,i) = max(vy(j,i),0)*rho(j-1,i) + min(vy(j,i),0)*rho(j,i);
        end
    end
    % Finite difference equation obtained form discritization of continuity
equation.
    rho_new = rho - (dt/dx) * (flux_xp - flux_xm) - (dt/dy) * (flux_yp - flux_ym);
    % Ensuring density lie between the 0 and rho_max
    rho_new = max(0, min(rho_max, rho_new));
end
```

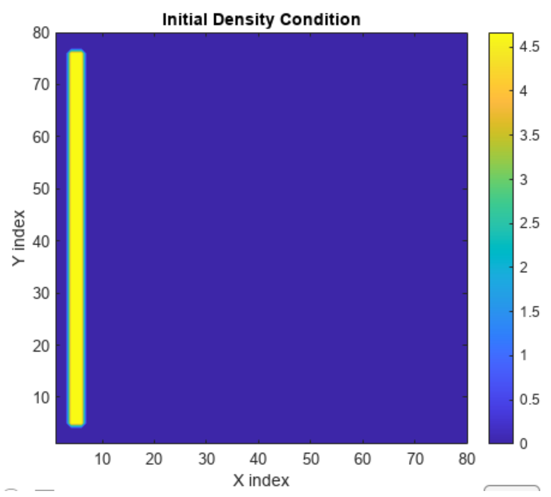
Lax Wendroff Scheme for Advection

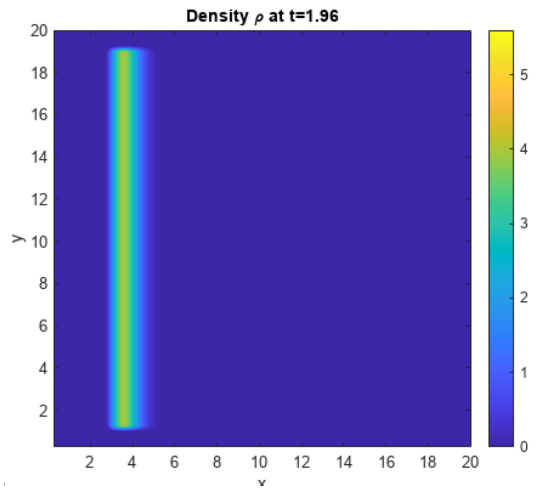
Now with this updated scheme I will be hopefully able to make the a more sharper solution to the Advection problems.

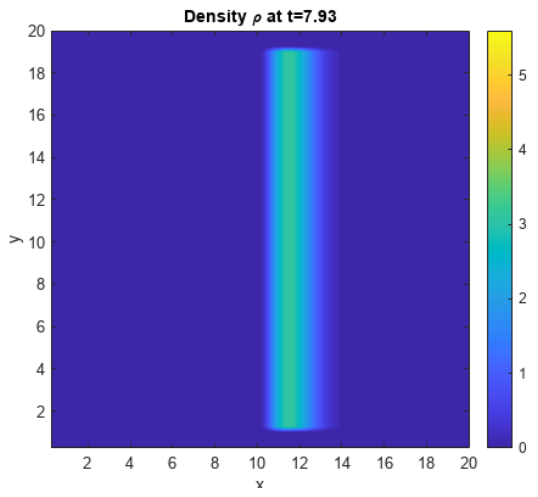
$$
\begin{aligned}
u_{ij}^{n+1} = u_{ij}^{n} &- \frac{C_x}{2}\left(u_{i+1j}^{n} - u_{i-1j}^{n}\right) - \frac{C_y}{2}\left(u_{ij+1}^{n} - u_{ij-1}^{n}\right) \\
&+ \frac{C_x^2}{2}\left(u_{i+1j}^{n} - 2u_{ij}^{n} + u_{i-1j}^{n}\right) + \frac{C_y^2}{2}\left(u_{ij+1}^{n} - 2u_{ij}^{n} + u_{ij-}\right. \\
&+ \frac{C_x C_y}{4}\left(u_{i+1j+1}^{n} - u_{i-1j+1}^{n} - u_{i+1j-1}^{n} + u_{i-1j-1}^{n}\right)
\end{aligned}
$$

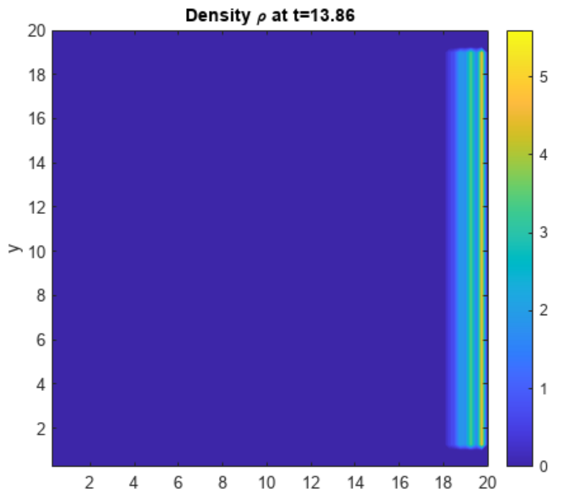
Numeric oscillations is what the scheme offers.A nd this is the draw back of using lax wendroff scheme. In have tried the same problem of constant velocity with this scheme of lax wendroff, and it gave results even better than the up-winding scheme. And therefore find this method to be the better one. But when tried one the hughes flow problem, this method gave oscillations and was very unstable.

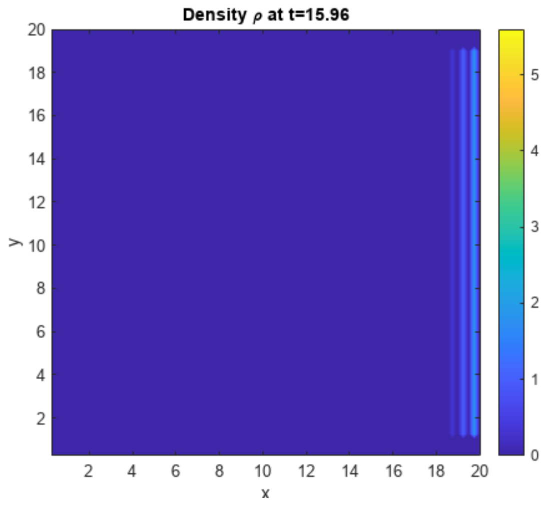
Solution for the synthetic problem using lax wendroff scheme

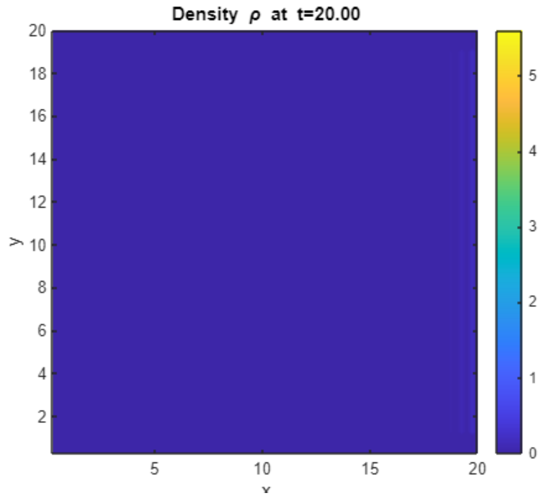






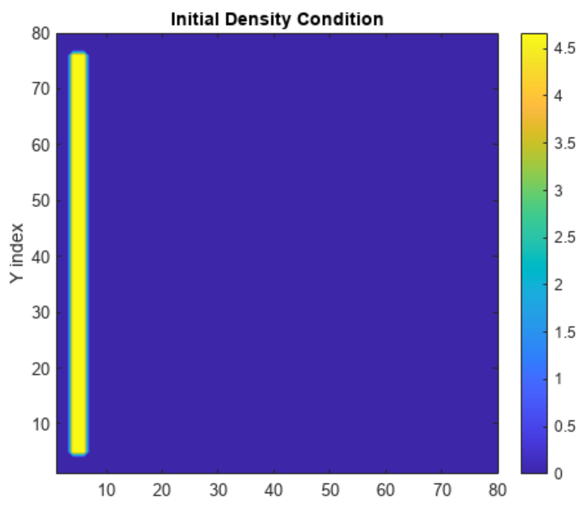


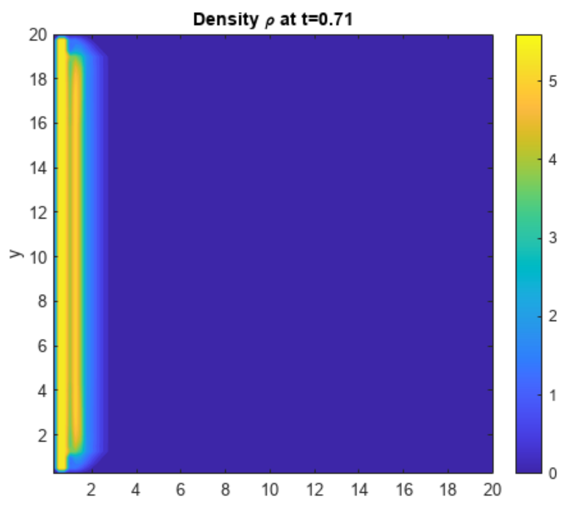


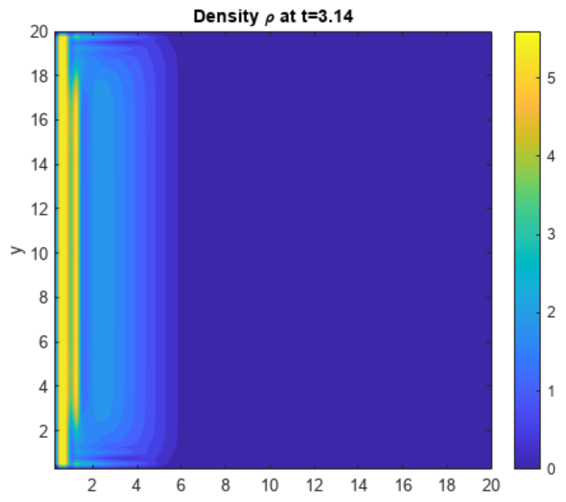


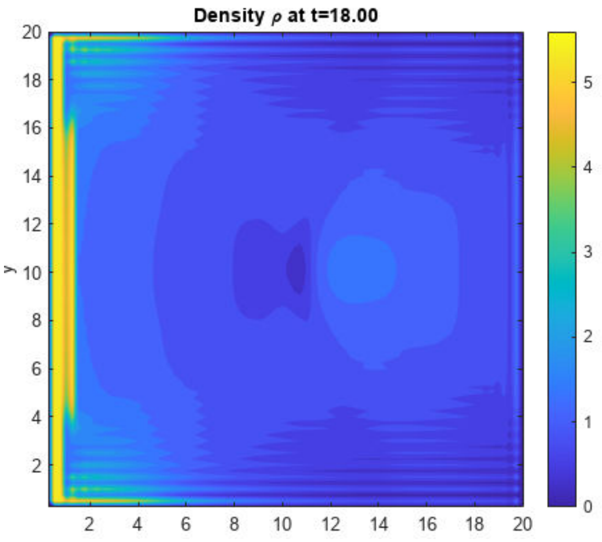
These results are actually better than the previously obtained from first order up-winding scheme. But it could be seen in the fifth frame that the oscillations are taking place.

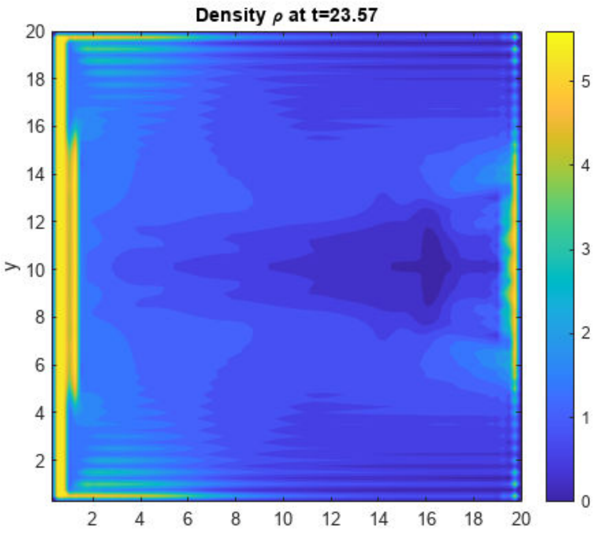
Solution for Hughe's flow with Lax Wendroff scheme

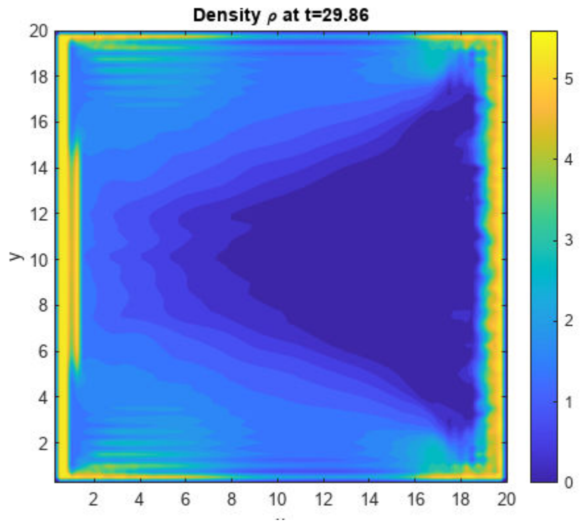

Initial Density Condition
Y index

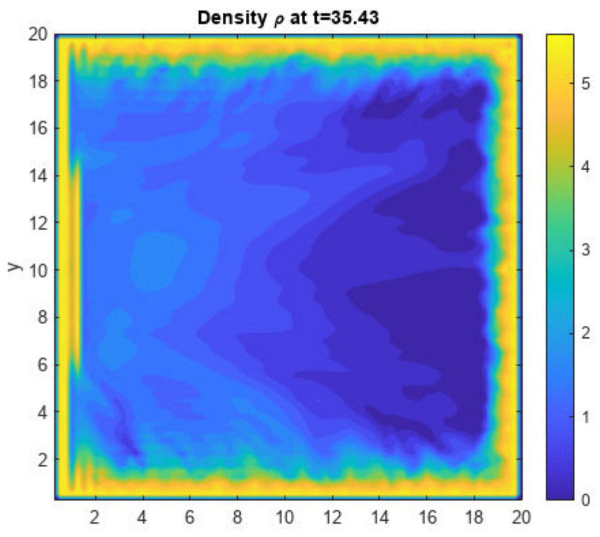

Density ρ at t=0.71
y

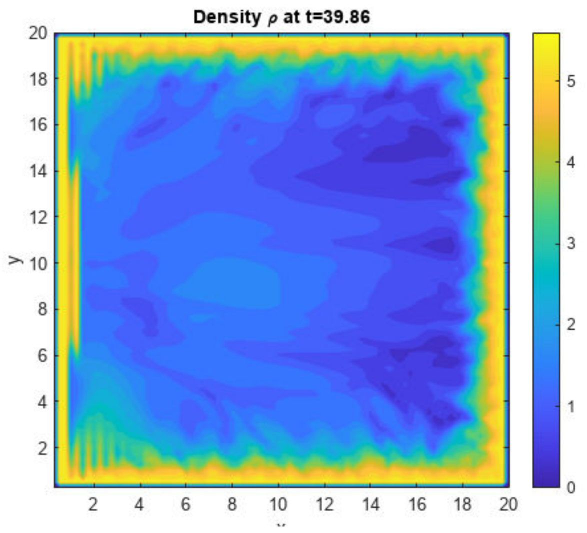

Density ρ at t=3.14
y

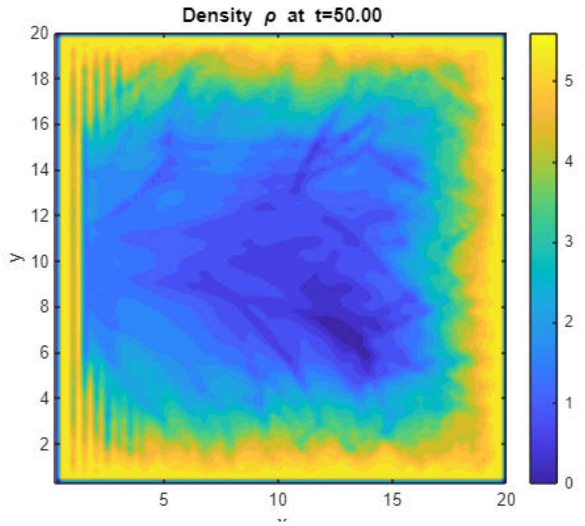

Density ρ at t=18.00
y


Density ρ at t=23.57
y


Density ρ at t=29.86
y







```
function rho_new = Lax_Wendroff_update(rho, vx, vy, dx, dy, dt, rho_max)
    [Ny, Nx] = size(rho);

    alpha_x = dt/dx;
    alpha_y = dt/dy;

    rho_new = rho;  % store new values separately

    for j = 2:Ny-1
        for i = 2:Nx-1
            % First term: central difference for advection
            first_term = (alpha_x/2) * (rho(j,i+1)*vx(j,i+1) -
rho(j,i-1)*vx(j,i-1)) + ...
                         (alpha_y/2) * (rho(j+1,i)*vy(j+1,i) -
rho(j-1,i)*vy(j-1,i));

            % Second term: second derivative corrections
            second_term = (alpha_x^2/2) * (vx(j,i)^2 * (rho(j,i+1) - 2*rho(j,i) +
rho(j,i-1))) + ...
```

```
                          (alpha_y^2/2) * (vy(j,i)^2 * (rho(j+1,i) - 2*rho(j,i) +
rho(j-1,i)));

            % Third term: cross-derivative
            third_term = (dt^2/(4*dx*dy)) * (2 * vx(j,i) * vy(j,i) * ...
                        (rho(j+1,i+1) - rho(j+1,i-1) - rho(j-1,i+1) +
rho(j-1,i-1)));

            rho_new(j,i) = rho(j,i) - first_term + second_term + third_term;
        end
    end

    % Enforce physical bounds
    rho_new = max(0, min(rho_max, rho_new));
end
```