```
clc;
clear all;
```

Two Dimensional Eikonal Equation using 5th order WENO Fast Sweep algorithm

So after the failure of my first attempt to replicate the result, here I am trying to understand the new algorithm to solve the eikonal equation for a hypothetical two dimensional problem.

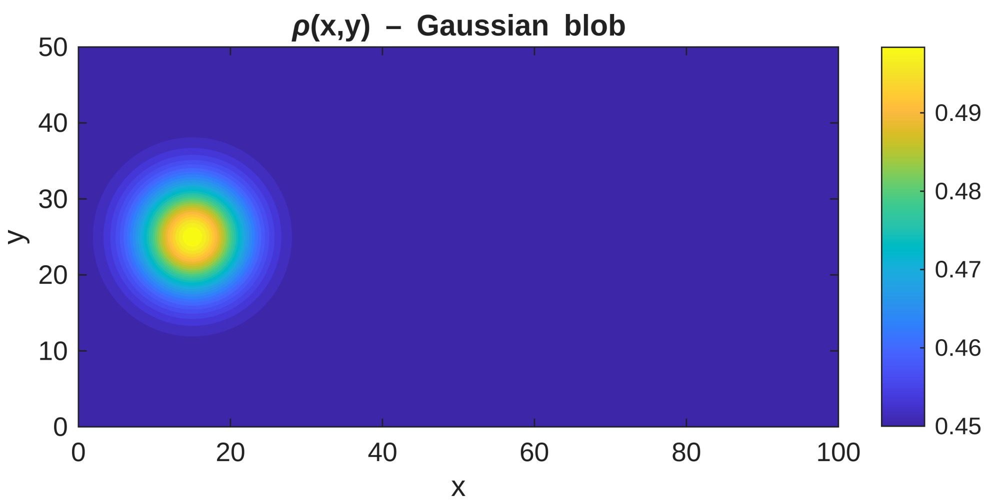
```
% Grid
global Nx Ny;
Nx = 400;
Ny = 200;
Lx = 100;
Ly = 50;

dx = Lx / Nx;
dy = Ly / Ny;

if dx == dy
    h = dx;
end

x = linspace(0, Lx, Nx);
y = linspace(0, Ly, Ny);
[X, Y] = meshgrid(x, y);

INFINITE = 10^2;
eta = 10^-6;
error_eta = 10^-9;

% To track the error for each iterations
iteration_error = [];

% Center of the blob
x0 = 10;
y0 = 25;

CFL = 0.5;
t_total = 20;
```

Initial Condtion

```
% Initial Condition (Gaussian Blob)

xc = 15;      % center x
yc = 25;      % center y
sigma = 5;    % width
```

```
rho_gauss = 0.05*exp( -((X-xc).^2 + (Y-yc).^2) / (2*sigma^2) ) + 0.45;
```

```
% Lets say there is no obstacle on the way
obstacle = false(Ny, Nx);
% obstacle(80:120,200:240) = true;
rho_gauss(obstacle) = 0;

figure(1); clf
contourf(X, Y, rho_gauss, 30, 'LineColor', 'none');
colorbar
axis equal tight
xlabel('x'); ylabel('y');
title('\rho(x,y) – Gaussian blob');
```



And with this density i would like to solve the eikonal equation using 5th order WENO algorithm. And the equation that was defined in the paper will require us to calculate the cost function first and here is how it needs to be defined

COST FUNCTION

$$C(\rho) = \frac{1}{u(\rho)} + g(\rho)$$

with $u(\rho) = 2(1 - \rho/10)$

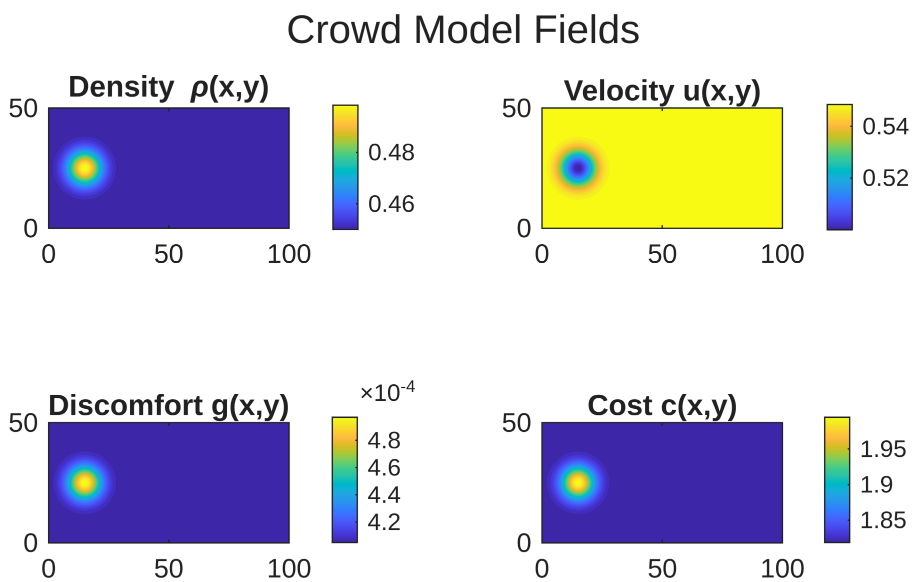
$g(\rho) = (0.002\rho^2)$

and hence

$$C(\rho) = \left[\frac{1}{2(1 - \rho/10)} + (0.002\rho^2)\right]$$

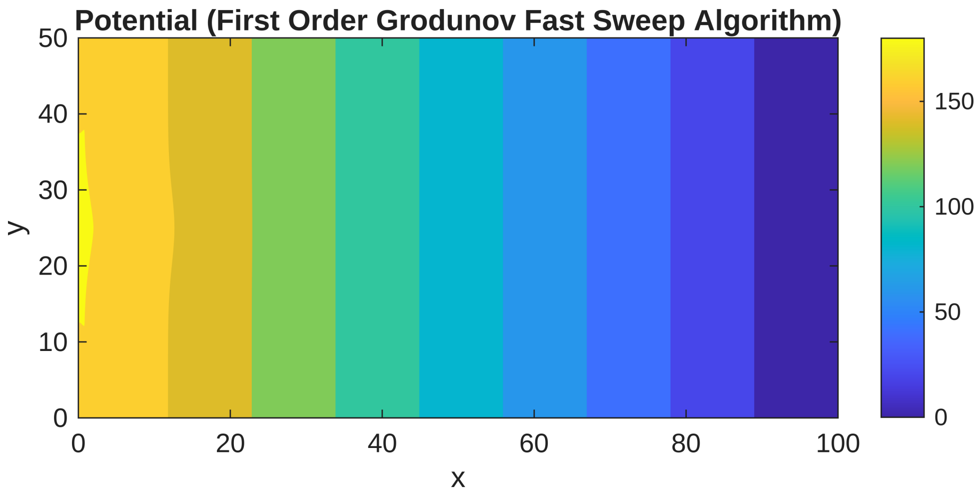
And hence to calculate this first we will calcuate the values of velocity and discomfort and then calculate the cost function

```
% Velocity
u = (1 - rho_gauss);
u = max(u, 1e-6);

% Discomfort
g = 0.002 * rho_gauss.^2;

% Cost
c = 1./u + g;

figure(1); clf

%---------------- Density ----------------%
subplot(2,2,1)
contourf(X, Y, rho_gauss, 30, 'LineColor','none')
colorbar
axis equal tight
title('Density \rho(x,y)')

%---------------- Velocity ----------------%
subplot(2,2,2)
contourf(X, Y, u, 30, 'LineColor','none')
colorbar
axis equal tight
title('Velocity u(x,y)')

%---------------- Discomfort ----------------%
subplot(2,2,3)
contourf(X, Y, g, 30, 'LineColor','none')
colorbar
axis equal tight
title('Discomfort g(x,y)')

%---------------- Cost ----------------%
subplot(2,2,4)
contourf(X, Y, c, 30, 'LineColor','none')
colorbar
axis equal tight
title('Cost c(x,y)')

sgtitle('Crowd Model Fields')
```



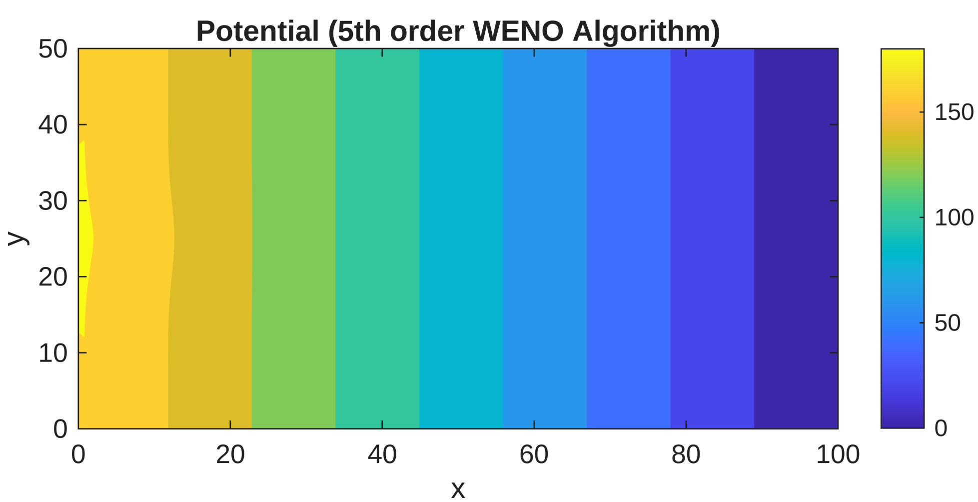
And now is the implementation of 5th order Fast sweep algorithm

```
% Returns the phi values at the current time step
phi_first_order =
First_order_Godunov_Fast_Sweep_Algorithm(c,dx,Nx,Ny,obstacle,INFINITE);

figure;
contourf(X, Y, phi_first_order, 'LineColor', 'none');
colorbar;
axis equal tight;
title('Potential (First Order Grodunov Fast Sweep Algorithm)');
xlabel('x'); ylabel('y');
```



And now I will be using this as my initial gues in the WENO algorithm

```
phi_initial =
weno_fastsweep(c,dx,Nx,Ny,obstacle,eta,error_eta,INFINITE,phi_first_order,ite
ration_error);
figure;
contourf(X, Y, phi_initial, 'LineColor', 'none');
colorbar;
axis equal tight;
title('Potential (5th order WENO Algorithm)');
xlabel('x'); ylabel('y');
```



First order Godunov Fast Sweep Algorithm (Older Version)

This is the algorithjm that is first order and was deisgned in last semeter. The solution that is obtained form this will be used as the initial condition for the WEO algorithm. This is done as said in the paper. The part of paper that says this is given below:

> The fast sweeping WENO method starts with the following initialization. Based on the b
> $(x,y) \in \Gamma_d$, we assign the exact boundary values on Γ_d. The solution from the first-order (
> (Zhao, 2005) is used as the initial guess at all other grid points. The first-order Godunov fa
> but is less accurate than the high-order WENO scheme. By using its result as the initial
> scheme, we can obtain the more accurate high-order WENO result with fewer iterations. I
> tance to Γ_d is less than or equal to $2h$, we fix their solution values as the initial guess duri
> The following Gauss–Seidel iterations with four alternating direction sweepings are the

And I have found two important information form the underlines of text. They are that initial condition has to be taken to be taken to be taken from First order Godunov fast sweep, and the next condition is that the grid points present two point near the boundary has to kept as it is. And this makes it very clear to talk about the fifth order scheme, becayuse to calculate the value of phi at i would mean I will have to use the value of phi at (i-2), (i-1), i, (i+1), (i+2). And here in this implmenetation I have not taken any ghost points.

```
function phi =
First_order_Godunov_Fast_Sweep_Algorithm(c,h,Nx_dash,Ny_dash,obstacle_dash,IN
FINITE)
    %--------------------------------------------
    % Initialization
    %--------------------------------------------
    phi = INFINITE*ones(Ny_dash, Nx_dash);    % Initial potential field
    phi(:,Nx_dash) = 0;              % Exit Boundary

    % Set phi inside obstacle to Inf
    phi(obstacle_dash) = INFINITE;

    %--------------------------------------------
    % Fast Sweeping Iterations
    %--------------------------------------------
    for sweep = 1:200
        % === Sweep 1
        for i = 2:Nx_dash-1
            for j = 2:Ny_dash-1
                if ~obstacle_dash(j, i)
                    phi = update_phi(phi, c, i, j, h);
                end
            end
        end
        % === Sweep 2
        for i = Nx_dash-1:-1:2
            for j = 2:Ny_dash-1
```

```
                 if ~obstacle_dash(j, i)
                     phi = update_phi(phi, c, i, j, h);
                 end
             end
         end
         % === Sweep 3
         for i = Nx_dash-1:-1:2
             for j = Ny_dash-1:-1:2
                 if ~obstacle_dash(j, i)
                     phi = update_phi(phi, c, i, j, h);
                 end
             end
         end
        % === Sweep 4
        for i = 2:Nx_dash-1
             for j = Ny_dash-1:-1:2
                 if ~obstacle_dash(j, i)
                     phi = update_phi(phi, c, i, j, h);
                 end
             end
        end

        phi(:,1) = phi(:,2);                % Right
        phi(1, :) = phi(2, :);              % Top
        phi(Ny_dash, :) = phi(Ny_dash-1,:);% Bottom

    end

end


% Subfunction: Update one grid point using Godunov scheme

function phi = update_phi(phi, invf, i, j, dx)
    a = min(phi(j, i-1), phi(j, i+1)); % x-direction neighbors
    b = min(phi(j-1, i), phi(j+1, i)); % y-direction neighbors
    f = invf(j, i); % Local inverse speed
    if abs(a - b) >= f * dx
        phi(j, i) = min(a, b) + f * dx;
        else
            inside = 2 * (f * dx)^2 - (a - b)^2;
        if inside >= 0
            phi(j, i) = (a + b + sqrt(inside)) / 2;
        end
    end
end
```

WENO FAST SWEEP

Now that the code has successfully converged,it deserves a proper explanation. And here it is. So the program starts by initializing the value that are obtained form the first order Godunov solution that was programme din last semester. Due to the algorithm being non linear it was important to be careful when assigning the initial condition. I first made the

Important

There is an error in the papaer. And it is that the sign of the equlity has to be the other way around. **This was the reason I was gettiing the value of phi to be equal to be imaginary**. But now that it is clear, I am seeing the algorithm working and it is doing pretty good. Solutioin could be seen above

Error

$$\phi_{i,j}^{\text{new}} = \begin{cases} \min(\phi_{i,j}^{x\min}, \phi_{i,j}^{y\min}) + c_{i,j}h, & \text{if } |\phi_{i,j}^{x\min} - \phi_{i,j}^{y\min}| \leqslant c \\ \dfrac{\phi_{i,j}^{x\min} + \phi_{i,j}^{y\min} + (2c_{i,j}^2 h^2 - (\phi_{i,j}^{x\min} - \phi_{i,j}^{y\min})^2)^{\frac{1}{2}}}{2}, & \text{otherwise,} \end{cases}$$

where $c_{i,j} = C(x_i, y_j, t)$, and

$$\begin{cases} \phi_{i,j}^{x\min} = \min(\phi_{i,j}^{\text{old}} - h(\phi_x)_{i,j}^-, \phi_{i,j}^{\text{old}} + h(\phi_x)_{i,j}^+), \\ \phi_{i,j}^{y\min} = \min(\phi_{i,j}^{\text{old}} - h(\phi_y)_{i,j}^-, \phi_{i,j}^{\text{old}} + h(\phi_y)_{i,j}^+), \end{cases}$$

```
function phi =
weno_fastsweep(c,h,Nx_dash,Ny_dash,obstacle_dash,eta,error_eta,INFINITE,initi
al_guess,iteration_error)
    % Phi is intialised with infinity
    phi = initial_guess;
    phi_old = INFINITE * ones(size(phi));

    % Iterations
    iterations = 0;
    loop_safety = 0;

    % Exit
    phi(:,Nx_dash) = 0;

    % Iterations
    while (sum(abs(phi(:)-phi_old(:))) > error_eta && loop_safety<100)

        %fprintf("Iteration = %d ",iterations);
        err = sum(abs(phi(:)-phi_old(:)));
        %fprintf("Error = %0.10f\n",err);

        phi_old = phi;
```

```
        iterations = iterations + 1;
        iteration_error(iterations) = err;
        loop_safety = loop_safety + 1;

        for sweep = 1:4
            switch sweep
                case 1
                    ix = 3:1:Nx_dash-2; jy = 3:1:Ny_dash-2;
                case 2
                    ix = 3:1:Nx_dash-2; jy = Ny_dash-2:-1:3;
                case 3
                    ix = Nx_dash-2:-1:3; jy = 3:1:Ny_dash-2;
                case 4
                    ix = Nx_dash-2:-1:3; jy = Ny_dash-2:-1:3;
            end



            for i = ix
                for j = jy
                    %if the point is outside the obstacle
                    if(~obstacle_dash(j,i))
                        %------------Calcualting the phix_min---------_%

                        r_back = (eta + (phi(j,i) - 2*phi(j,i-1) +
phi(j,i-2))^2)/(eta + (phi(j,i+1) - 2*phi(j,i) + phi(j,i-1))^2);
                        r_front = (eta + (phi(j,i) - 2*phi(j,i+1) +
phi(j,i+2))^2)/(eta + (phi(j,i+1) - 2*phi(j,i) + phi(j,i-1))^2);

                        % Calculate the w
                        w_front = 1/(1+2*(r_front^2));
                        w_back = 1/(1+2*(r_back^2));

                        % Dell phi by Dell x plus and minus
                        phix_minus = (1-w_back)*((phi(j,i+1)-phi(j,i-1))/
(2*h)) + w_back*((3*phi(j,i) - 4*phi(j,i-1) + phi(j,i-2))/(2*h));
                        phix_plus = (1-w_front)*((phi(j,i+1)-phi(j,i-1))/
(2*h)) + w_front*((-3*phi(j,i)+4*phi(j,i+1)-phi(j,i+2))/(2*h));


                        phix_min = min((phi(j,i) - h*phix_minus),(phi(j,i) +
h*phix_plus));

                        %-----------calculating the phiy_min---------%

                        r_back = (eta + (phi(j,i) - 2*phi(j-1,i) +
phi(j-2,i))^2) / (eta + (phi(j+1,i) - 2*phi(j,i) + phi(j-1,i))^2);
                        r_front = (eta + (phi(j,i) - 2*phi(j+1,i) +
phi(j+2,i))^2) / (eta + (phi(j+1,i) - 2*phi(j,i) + phi(j-1,i))^2);
```

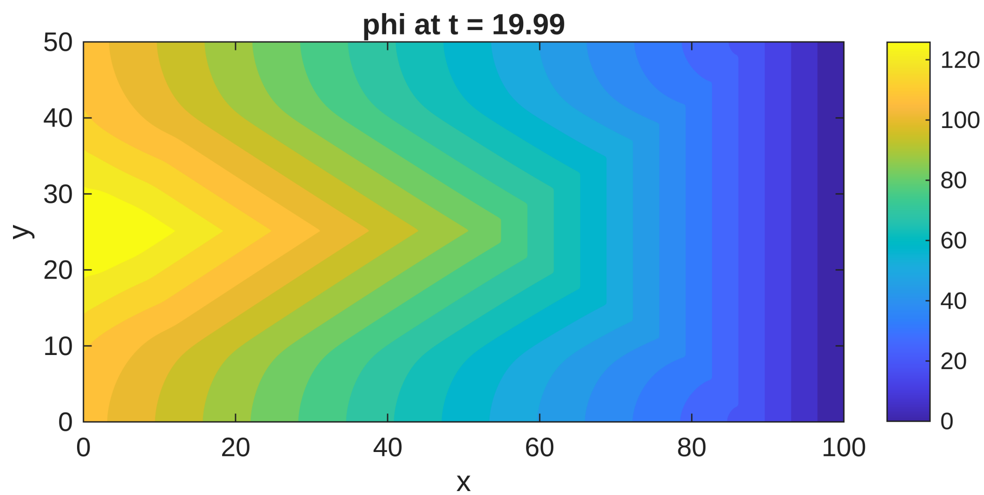
```
                        % Calculate the w
                        w_front = 1 / (1 + 2*(r_front^2));
                        w_back = 1 / (1 + 2*(r_back^2));

                        % Dell phi by Dell x plus and minus
                        phiy_minus = (1-w_back) * ((phi(j+1,i)-phi(j-1,i))/
(2*h)) + w_back * ((3*phi(j,i) - 4*phi(j-1,i) + phi(j-2,i))/(2*h));
                        phiy_plus = (1-w_front) * ((phi(j+1,i) - phi(j-1,i))/
(2*h)) + w_front * ((-3*phi(j,i) + 4*phi(j+1,i) - phi(j+2,i))/(2*h));


                        phiy_min = min((phi(j,i) - h*phiy_minus), (phi(j,i)
+ h*phiy_plus));

                        if abs(phix_min - phiy_min) >= c(j,i)*h
                            phi(j,i) = min(phiy_min,phix_min) + c(j,i)*h;
                        else
                            phi(j,i) = ((phix_min + phiy_min) +
(2*(c(j,i)*h)^2 - (phix_min - phiy_min)^2)^0.5)/2;
                        end
                    % If the point is inside the obstacle
                    else
                        phi(j,i) = INFINITE;
                    end
                end
            end

            %------Extrapolation------%

            % Bottom edge %
            %phi(2,:) = phi(3,:);
            %phi(1,:) = phi(3,:);
            % Top edge %
            %phi(Ny_dash,:) = phi(Ny_dash-2,:);
            %phi(Ny_dash-1,:) = phi(Ny_dash-2,:);
            % Right edge %
            %phi(:,Nx_dash-1) = phi(:,Nx_dash-2);
            % Left edge %
            %phi(:,2) = phi(:,3);
            %phi(:,1) = phi(:,3);
        end
    end
end
```

Main Solver

```
rho = rho_gauss;
dt_i = [];
%-----------MAIN SOLVER------------%
t = 0;
```

```
step = 0; % For plotting at regular interval

while t< t_total

    % Updating the Spped, Discomfort and Cost
    speed = 1 - rho;
    discomfort = 0.002 * rho.^2;
    cost = 1./speed + discomfort;

    %-- Calculate Initial Phi guess using Lower order Fast Sweep
    phi_first_order =
First_order_Godunov_Fast_Sweep_Algorithm(cost,h,Nx,Ny,obstacle,INFINITE);
   %-- Calculate The final phi using higher ordfer scheme and above intial
guess
    phi =
weno_fastsweep(cost,h,Nx,Ny,obstacle,eta,error_eta,INFINITE,phi_first_order,i
teration_error);

    %--Calculate the Components of Velocity
    [phi_x, phi_y] = gradient(phi, h, h); % partial derivatives of phi in x
and y
    mag_phi = sqrt(phi_x.^2 + phi_y.^2);
    mag_phi = max(mag_phi, eta);          % To be safe with denominator
    for i = 1:1:Nx
        for j = 1:1:Ny
            x_velocity(j,i) = speed(j,i) * (phi_x(j,i))/mag_phi(j,i);
            y_velocity(j,i) = speed(j,i) * (phi_y(j,i))/mag_phi(j,i);
        end
    end

    % Updating the delta_t with proper CFL condition (Wiki Pedia)
    u_max = max(abs(x_velocity(:)));
    v_max = max(abs(y_velocity(:)));
    dt = CFL / ( u_max/h + v_max/h + 1e-12 );

    % Updating the value of rho
    rho = upwind_update(rho, x_velocity, y_velocity, dx, dy, dt);

    step = step+1;
    dt_i(step) = dt;
    %--Ploting the rho_next;
    if mod(step, 10) == 0
        % ---Density ---
        figure(6);
        clf;
        contourf(x, y, rho, 20, 'LineColor', 'none');
        colorbar;
        title(sprintf('phi at t = %.2f', t));
        xlabel('x'); ylabel('y');
        axis equal tight;
```

```
        drawnow;
    end
    %fprintf('dt = %0.2f\n',dt);
    % Updating the time
    t = t + dt;
end
```



Printing the dt

```
dt_i
```

```
dt_i = 1×320                                                 Columns 304:319
    0.0625    0.0625    0.0625    0.0625    0.0625    0.0625    0.0625    0.0625
```

Upwinding

```
function rho_new = upwind_update(rho, vx, vy, dx, dy, dt)
    global Nx Ny;
    % Boundary Condition
    for j = 2 : Ny-1
        rho(j,1) = 0;
        rho(j,Nx) = rho(j,Nx-1);
    end
    for i = 2 : Nx-1
        rho(1,i) = 0;
        rho(Ny,i) = 0;
    end
    % 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);
end
```