Hughe's flow Second Attempt

Domain

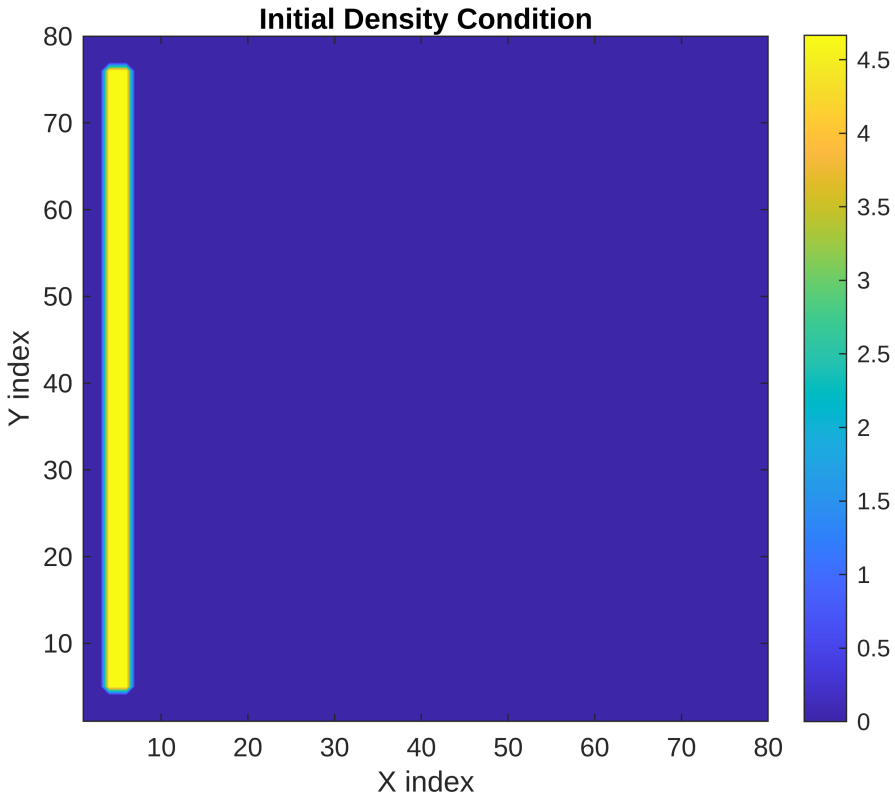
```
Nx = 80; Ny = 80;
Lx = 20; Ly = 20;
dx = Lx/Nx; dy = Ly/Ny;
[x, y] = meshgrid(linspace(dx,Lx,Nx), linspace(dy,Ly,Ny));
```

Parameters

```
Vmax = 2.5;
rho_max = 5.0;
CFL = 0.4;
```

Initial Condition

```
% Assigning zero density everywhere
rho = zeros(Nx,Ny);
% Initial density condition
rho(5:Ny-4,4:6) = 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');
```



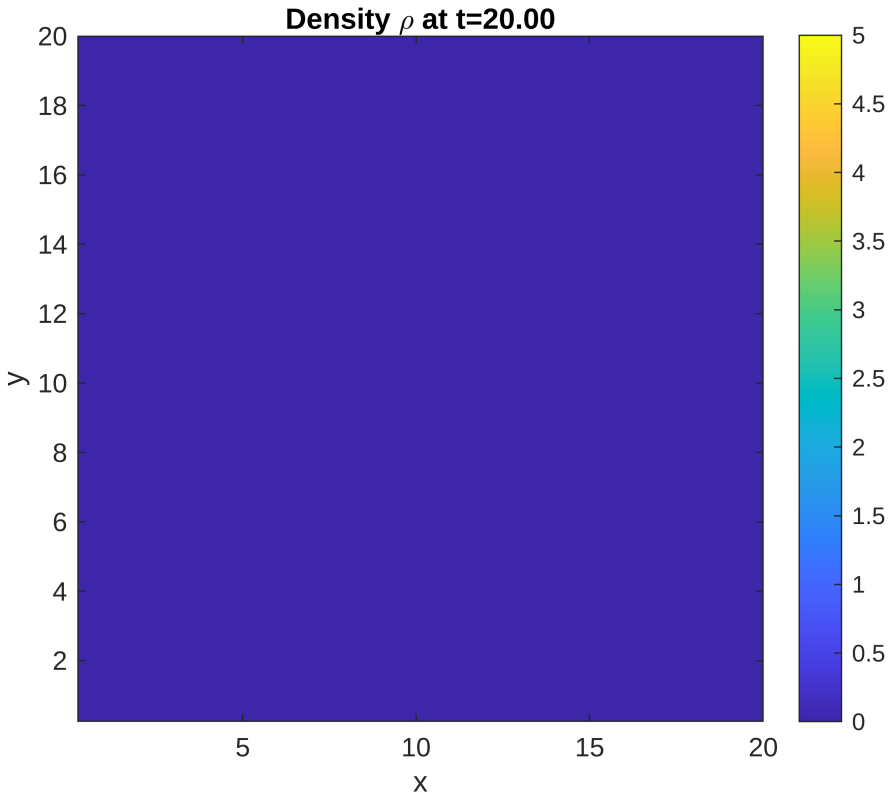
The Simulation Loop

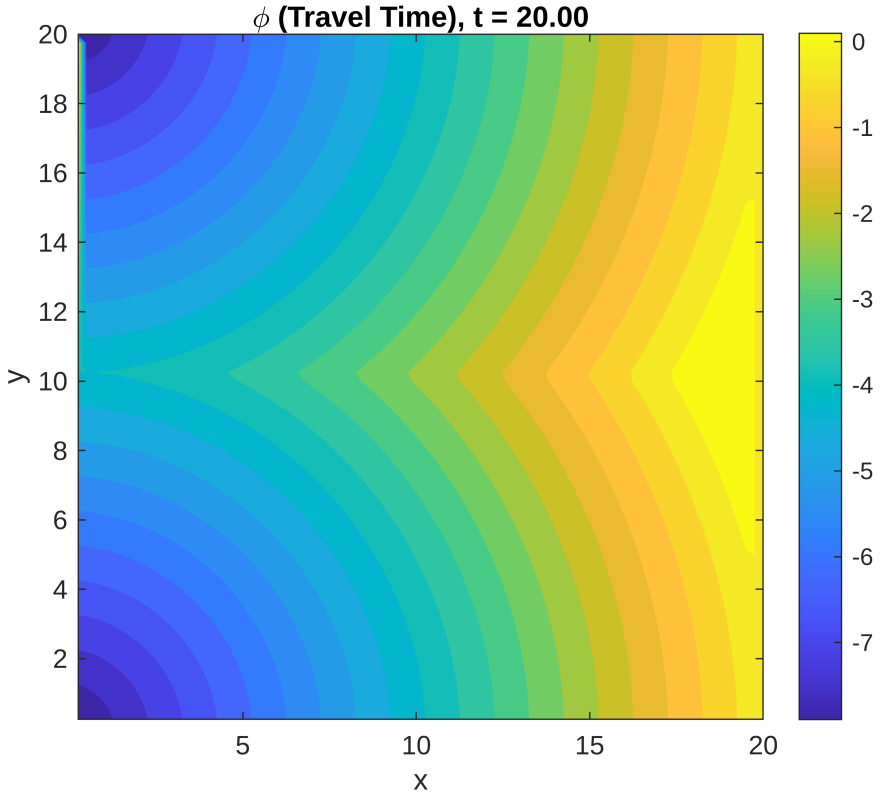
```
Tfinal = 20;
t = 0;
while t < Tfinal
    % Speed field
    f = Vmax*(1 - rho/rho_max);
    % To speed doesn't become zero completely
    f(f<1e-6) = 1e-6;
    % Solve eikonal equation
    phi = fast_sweeping(1./f, dx,Nx,Ny);
    % Direction field
    [phix, phiy] = gradient(phi, dx, dy);
    grad_mag = 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;
```

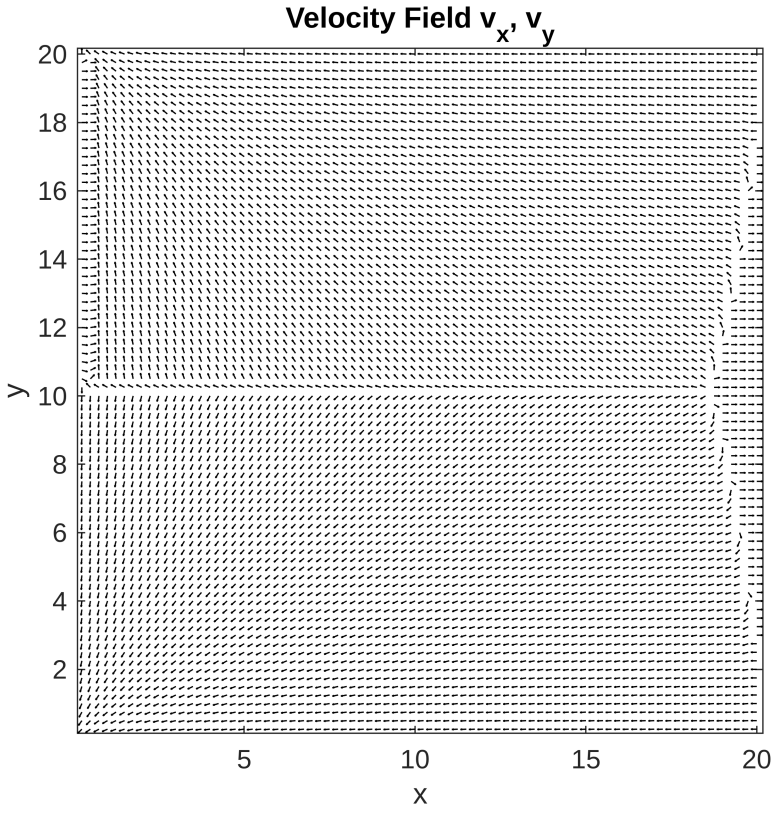
```
    % Time step
    maxspeed = max(max(sqrt(vx.^2 + vy.^2)));
    dt = CFL * min(dx,dy) / maxspeed;
    if t+dt > Tfinal, dt = Tfinal-t; end
    % Update rho
    rho = upwind_update(rho, vx, vy, dx, dy, dt,rho_max);
    % Advance time
    t = t + dt;
    % Plots
    if mod(round(t/dt),2) == 0
        % 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(sprintf('\\phi (Travel Time), t = %.2f', t));
        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=20.00
20
18
16
14
12
10
8
6
4
2
y
5
10
15
20
x
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0


ϕ (Travel Time), t = 20.00
20
18
16
14
12
10
8
6
4
2
y
5
10
15
20
x
0
-1
-2
-3
-4
-5
-6
-7



Modified Fast Sweep function for Potential

```
function phi = fast_sweeping(invf, dx,Nx,Ny)
    % Intialization of phi
    phi = inf(Ny,Nx);
    phi(:,end) = 0; % Dirichlet condition on the left edge of the domain.

    % Boundary Condition

    % Since the boundary points are excluded from the computation, our task
    % to use only use the boundary points to get the vales of boundary
    % points. And once the boundary points are prepared we can finally go
    % into the sweep function and let the interior poins use those boundary
    % points.

    for sweep_boundary=1:100
        % For the top boundary (column = 2:Nx-1 | row = Ny)
        % This will update phi at boundary from top right to top left.
        for i=Nx:-1:2
            phi(Ny,i-1) = phi(Ny,i) - invf(Ny,i)*dx;
        end
        % For the bottom boundary (column = 2:Nx-1 | row = 1)
        % This will update phi at boundary from bottom right to bottom left
        for i=Nx:-1:2
            phi(1,i-1) = phi(1,i) - invf(1,i)*dx;
```

```
        end
        % For the left boundary (column = 1 | row = 2:Ny-1)
        % This will update phi at boundary from bottom left to top left
        for i=2:Ny-1
            phi(i,1) = phi(i-1,1) + invf(i)*dx;
        end
        % note that there will be some problems in the top left corner grid
        % point. And this could not be solved.
    end
    % Now we have all the boundary point ready to use for calculating the
    % interior points.


    for sweep=1:100
        % The limit starts from column and row 2, and ends at column and row
⊠N⊠1⊠.
        % This excludes the boundary points.
        % This will sweep i from 2 to n-1 and j from 2 to n-1
        for i=2:Nx-1
            for j=2:Ny-1
                % Calculating the minimum value of phi, in it's neighbour in
x direction
                a = min(phi(i,j-1), phi(i,j+1));
                % Calculating the minimum value of phi, in it's neighbour in
y direct
                b = min(phi(i-1,j), phi(i+1,j));
                tmp = sort([a, b]);
                if abs(tmp(1)-tmp(2)) >= invf(i,j)*dx
                    %phi = min(a,b) + (step_time)
                    phi(i,j) = tmp(1) + invf(i,j)*dx;
                else
                    %phi = solution of quadratic equation
                    phi(i,j) = (tmp(1)+tmp(2) + ...
                        sqrt(2*(invf(i,j)*dx)^2 - (tmp(1)-tmp(2))^2))/2;
                end
            end
        end

        % This will sweep i from 2 to n-1 and j from n-1 to 2
        for i=2:Nx-1
            for j=Ny-1:-1:2
                % Calculating the minimum value of phi, in it's neighbour in
x direction
                a = min(phi(i,j-1), phi(i,j+1));
                % Calculating the minimum value of phi, in it's neighbour in
y direct
                b = min(phi(i-1,j), phi(i+1,j));
                tmp = sort([a, b]);
                if abs(tmp(1)-tmp(2)) >= invf(i,j)*dx
                    %phi = min(a,b) + (step_time)
```

```
                    phi(i,j) = tmp(1) + invf(i,j)*dx;
                else
                    %phi = solution of quadratic equation
                    phi(i,j) = (tmp(1)+tmp(2) + ...
                        sqrt(2*(invf(i,j)*dx)^2 - (tmp(1)-tmp(2))^2))/2;
                end
            end
        end

        % This will sweep the i from n-1 to 2 and j from 2 to n-1
        for i=Nx-1:-1:2
            for j=2:Ny-1
                % Calculating the minimum value of phi, in it's neighbour in
x direction
                a = min(phi(i,j-1), phi(i,j+1));
                % Calculating the minimum value of phi, in it's neighbour in
y direct
                b = min(phi(i-1,j), phi(i+1,j));
                tmp = sort([a, b]);
                if abs(tmp(1)-tmp(2)) >= invf(i,j)*dx
                    %phi = min(a,b) + (step_time)
                    phi(i,j) = tmp(1) + invf(i,j)*dx;
                else
                    %phi = solution of quadratic equation
                    phi(i,j) = (tmp(1)+tmp(2) + ...
                        sqrt(2*(invf(i,j)*dx)^2 - (tmp(1)-tmp(2))^2))/2;
                end
            end
        end

        % This will sweep i from n-1 to 2, and j from n-1 to 2
        for i=Nx-1:-1:2
            for j=Ny-1:-1:2
                % Calculating the minimum value of phi, in it's neighbour in
x direction
                a = min(phi(i,j-1), phi(i,j+1));
                % Calculating the minimum value of phi, in it's neighbour in
y direct
                b = min(phi(i-1,j), phi(i+1,j));
                tmp = sort([a, b]);
                if abs(tmp(1)-tmp(2)) >= invf(i,j)*dx
                    %phi = min(a,b) + (step_time)
                    phi(i,j) = tmp(1) + invf(i,j)*dx;
                else
                    %phi = solution of quadratic equation
                    phi(i,j) = (tmp(1)+tmp(2) + ...
                        sqrt(2*(invf(i,j)*dx)^2 - (tmp(1)-tmp(2))^2))/2;
                end
            end
        end
```

```
    end
end
```

Upwind for Advection of Density

```
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
            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);
            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
```