Project Progress

Domain & grid

The domain I have chosen is a **square domain** with dimension **20*20-meters square**. And the domain is discretized into **80*80 gird points**.

```
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

Assuming the GreenSheiild's, maximum velocity of a single person is taken to be 2.5 meters per seconds, and the maximum density is taken to be 5 person per meter square.

The data about density is obtained from the following webpage → Link.

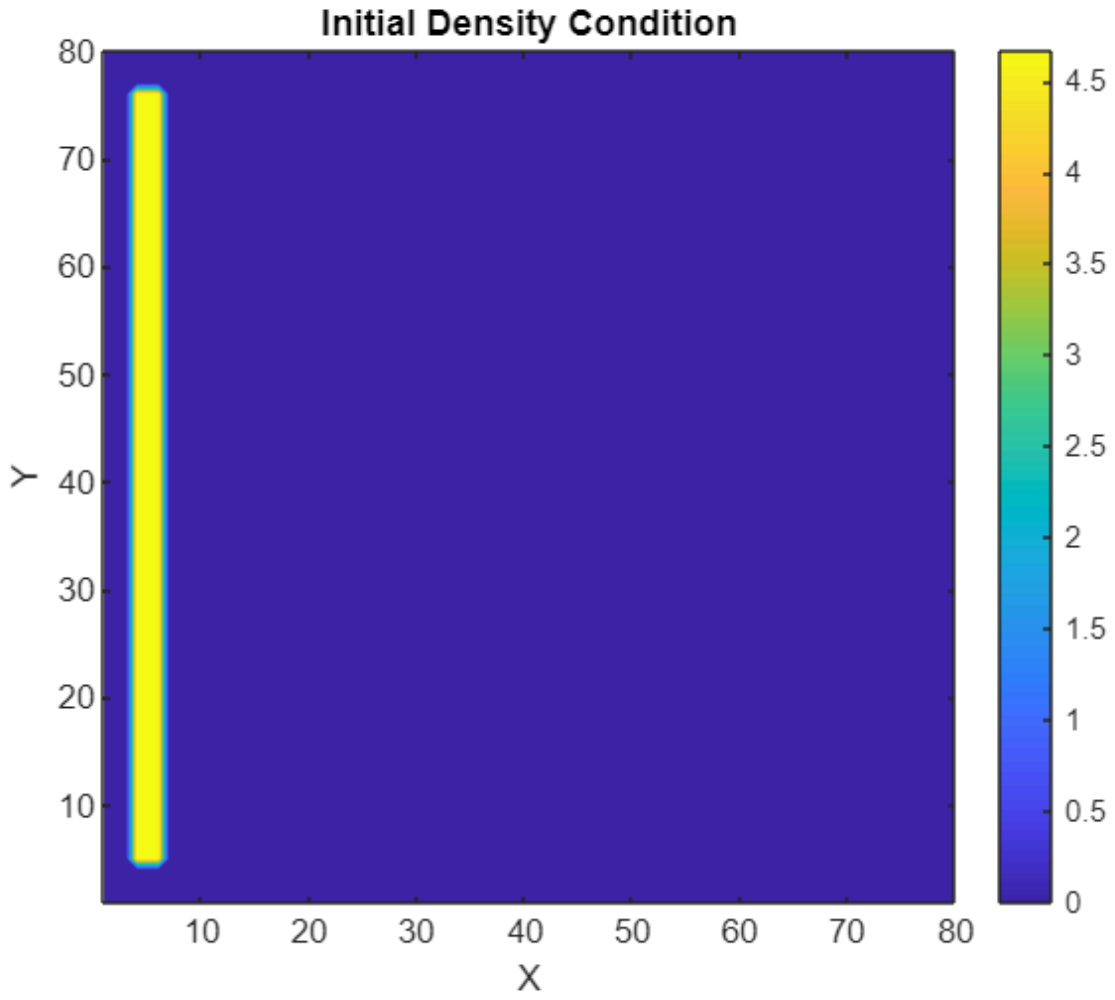
And the value of velocity is obtained from web search → Link.

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

Initial Density Condition

Initial density is assumed to be concentrated, at the left boundary. And the value of density is set to be 4.9 (5.0 is the maximum density). The dimension of region of high density is 2 meter in *X* direction and 71 meters is *Y*.

The left most boundary of the domain is not assigned any value of density as 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');
```

Exit Conditions

The exit I am defining is the complete right boundary on the square domain.

`phi` could be assumed as a scalar field of travel time. At every point on space depending on the density distribution, and direction of exit, there is some time associated to that point, and this time physically means the time, required for the pedestrian at that point to reach the exit. And this time is what is called `phi`. The unit of `phi` is seconds.

Time loop

The time of computation is defined to be 50 seconds.

1. With the initial values of density (`rho`), values of speed (`f`) are obtained using Greenshields's relation.
2. Values of `phi` are updated using the previously obtained values of speed (`f`). And this is done using the function `fast_sweeping` . It uses fast sweeping scheme which is known to be used for solving Eikonal equation.
3. Gradient and direction cosine of the velocity is obtained using the updated `phi` values. And this is done according to definition of direction cosine mentioned in the paper.

$$\hat{\phi}_x = \frac{-\frac{\partial \phi}{\partial x}}{\sqrt{\left(\frac{\partial \phi}{\partial x}\right)^2 + \left(\frac{\partial \phi}{\partial y}\right)^2}}, \qquad \hat{\phi}_y = \frac{-\frac{\partial \phi}{\partial y}}{\sqrt{\left(\frac{\partial \phi}{\partial x}\right)^2 + \left(\frac{\partial \phi}{\partial y}\right)^2}},$$

4. Then the velocities are defined using the obtained direction cosine, and the speed.

$$u = f(\rho)\hat{\phi}_x \quad v = f(\rho)\hat{\phi}_y,$$

5. Appropriate boundary conditions are set onto the component of velocity. Specifically, no penetration condition through walls, and maximum velocity outwards at the exit.
6. Values of density is updated using the updated values of velocity, and previous values of densities. And this is done using the function `upwind_update` .
7. And then the results are plotted.

```
Tfinal = 50;
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, dy,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; % Left wall
    vx(:,end) = Vmax % Exit velocity at right wall
    vy(1,:)   = 0; % Bottom wall
    vy(end,:) = 0; % Top wall
```

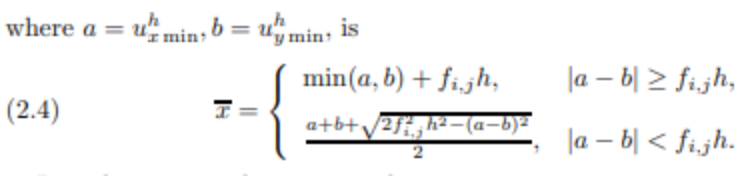
```
% Time step
maxspeed = max(max(sqrt(vx.^2 + vy.^2)));
dt = CFL * min(dx,dy) / maxspeed;
if t+dt > Tfinal, dt = Tfinal-t; end

% Update rho
rho = upwind_update(rho, vx, vy, dx, dy, dt);

% Advance time
t = t + dt;

% Plots
if mod(round(t/dt),2) == 0

        % Figure 2: 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;

     % Figure 2: Potential (phi)
    figure(2);
    contourf(x, y, phi, 20, 'LineColor', 'none');
    colorbar;
    title('\phi (Travel Time)');
    xlabel('x'); ylabel('y'); axis equal tight;

    % Figure 3: 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
```

-- Helper functions : `fast_sweeping` ---

This function uses the fast-sweeping method for solving the Eikonal equation.

For solving the Eikonal equation I have followed some part of this resource → Link to those resources



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

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

The above equation is obtained from the resource provided above.

`phi` is to be updated, and this is done using fast sweeping method. As a part of boundary condition, `phi` is assigned to be 30 seconds at walls, and zero seconds at the exit in the current problem.

During every sweep, points at the boundary are excluded from computation. But are used to compute the interior points.

```
function phi = fast_sweeping(invf, dx, dy,Nx,Ny)
    % Boundary Condition
    phi = 30*ones(Ny,Nx);
    phi(:,Nx) = 0;


    for sweep=1:200
        % The limit starts from column and row 2, and ends at column and row
(N-1).
        % This excludes the boundary points.
        for i=2:Ny-1
            for j=2:Nx-1
                % Calculating the minimum value of phi, in it's neighbour in x dire
```

```
ction
            a = min(phi(i,j-1), phi(i,j+1));

             % Calculating the minimum value of phi, in it's neighbour in y direct
ion
            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
```

--- Helper functions : `upwind_update_2d` --

Up-winding scheme is used in solving the continuity equation.

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x}(\rho u) + \frac{\partial}{\partial y}(\rho v) = 0.$$

The above equation is obtained from Hughe's original paper (2002).

Gradient of flux (first order derivative) in any direction could be calculated using forward or backward difference approximation. And this will depend on the direction of propagation of disturbance, which is direction of velocity at that point. Similar approach is used in the function.

```
function rho_new = upwind_update_2d(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 equat
ion.
    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
```

Results:

The animation of results is present in the email.

I would like to kindly request your time to discuss the results, and please let me know if the method I have used to solve the equation is appropriate.