```
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.

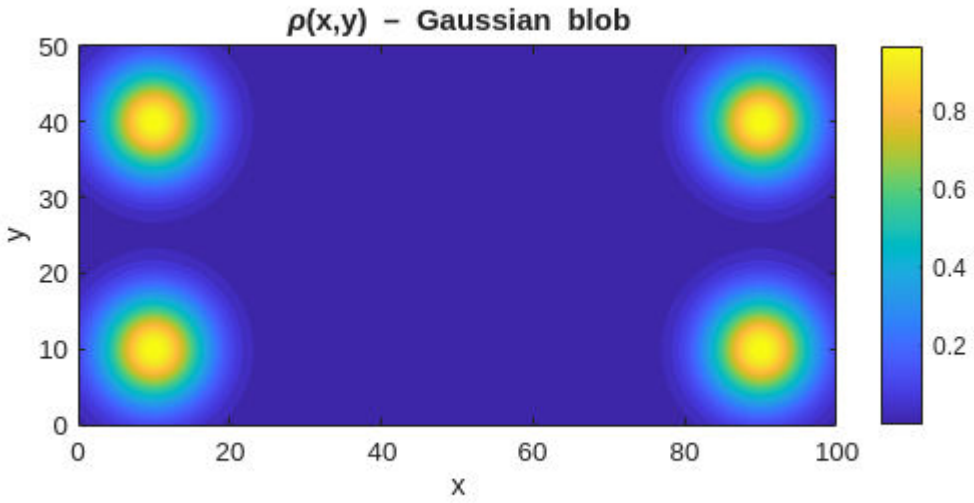
```
% Grid
Nx = 400;
Ny = 200;
Lx = 100;
Ly = 50;

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

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;

% Standard deviation
sigma = 5;
```

Initial Condtion

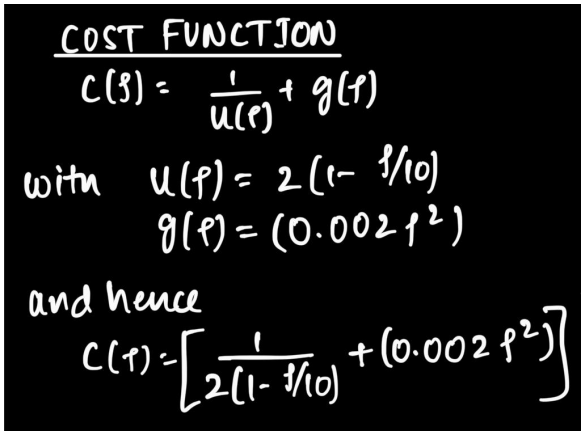
```
% Gaussian blob
%rho_gauss = exp(-((X - x0).^2 + (Y - y0).^2) / (2 * sigma^2));
% Two Gaussian Blob
%rho_gauss = exp(-((X-10).^2 + (Y-25).^2)/(2*4^2)) + exp(-((X-30).^2 +
(Y-15).^2)/(2*6^2));
% Radial Peak
%rho_gauss = exp(-((X-5).^2 + (Y-5).^2)/(2*3^2));
rho_gauss = ...
    exp(-((X-10).^2  + (Y-10).^2)/(2*5^2)) + ...
```

```
    exp(-((X-90).^2  + (Y-10).^2)/(2*5^2)) + ...
    exp(-((X-10).^2  + (Y-40).^2)/(2*5^2)) + ...
    exp(-((X-90).^2  + (Y-40).^2)/(2*5^2));
```

```
% 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');
```



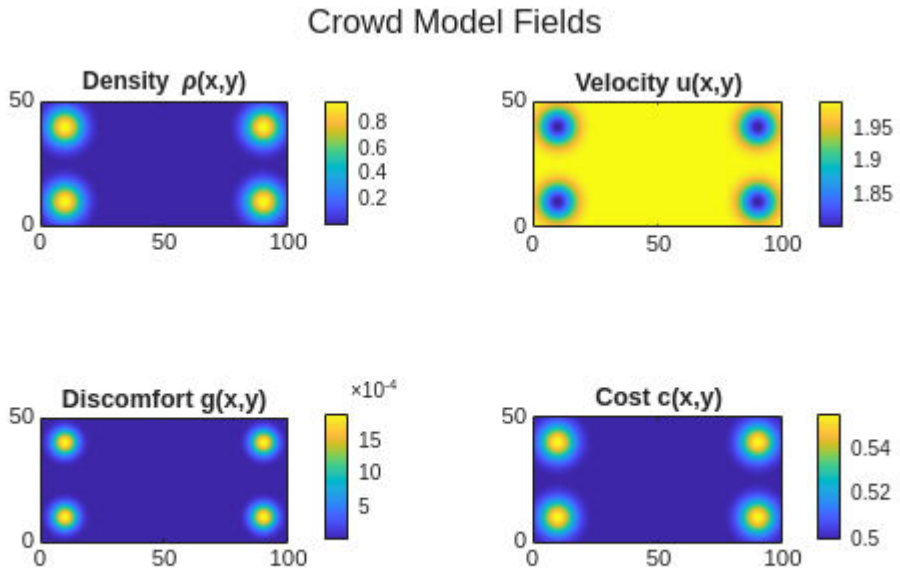
And with this density i would like to solve the eikonal equation using 5th order WENO algorithm. And the equation that was defined in the paper will require us to calculate the cost function first and here is how it needs to be defined



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

```
% Velocity
u = 2 * (1 - rho_gauss/10);
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');
```

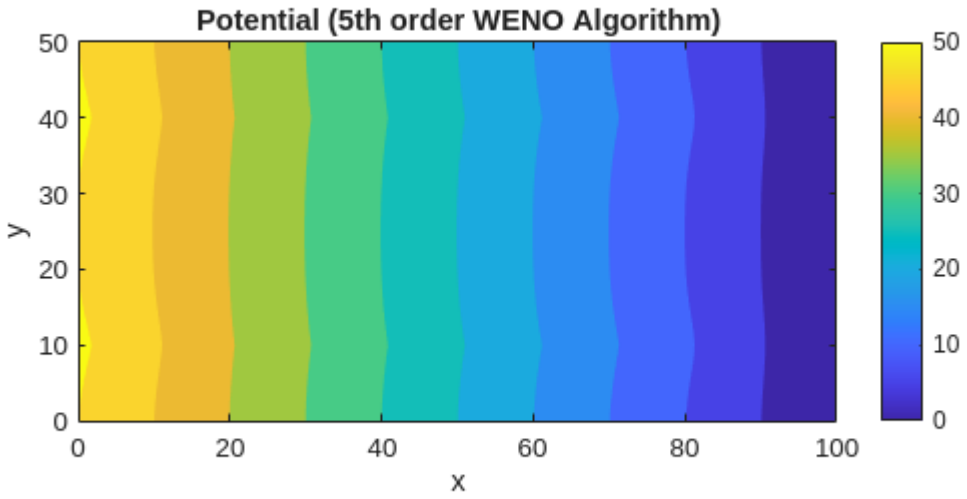
And now I will be using this as my initial gues in the WENO algorithm

```
phi =
weno_fastsweep(c,dx,Nx,Ny,obstacle,eta,error_eta,INFINITE,phi_first_order,ite
ration_error);
```

```
Iteration = 0 Error = 5987056.2130377041
Iteration = 1 Error = 25.6567460456
Iteration = 2 Error = 26.5954454875
Iteration = 3 Error = 26.0277002909
Iteration = 4 Error = 23.8430176989
Iteration = 5 Error = 19.6285974319
Iteration = 6 Error = 16.0305151663
Iteration = 7 Error = 14.6493563700
Iteration = 8 Error = 14.0328250120
Iteration = 9 Error = 12.8637514629
Iteration = 10 Error = 11.8412627306
Iteration = 11 Error = 10.8846517122
Iteration = 12 Error = 9.9854338808
Iteration = 13 Error = 9.2040635764
Iteration = 14 Error = 8.4756330899
Iteration = 15 Error = 7.8508713122
Iteration = 16 Error = 7.2171582904
Iteration = 17 Error = 6.6403551935
Iteration = 18 Error = 6.1459898382
Iteration = 19 Error = 5.7563209244
Iteration = 20 Error = 5.4058533173
Iteration = 21 Error = 5.0330240154
Iteration = 22 Error = 4.6452887822
Iteration = 23 Error = 4.2403714753
Iteration = 24 Error = 3.7818928664
Iteration = 25 Error = 3.3585638153
Iteration = 26 Error = 3.0041748339
Iteration = 27 Error = 2.7355574737
Iteration = 28 Error = 2.1991504667
Iteration = 29 Error = 1.6995448194
Iteration = 30 Error = 1.3343163453
Iteration = 31 Error = 1.0753504327
```

```
Iteration = 32 Error = 0.9116159488
Iteration = 33 Error = 0.7918551866
Iteration = 34 Error = 0.7010890075
Iteration = 35 Error = 0.6227416950
Iteration = 36 Error = 0.4731437946
Iteration = 37 Error = 0.2893754486
Iteration = 38 Error = 0.1622761593
Iteration = 39 Error = 0.0762658414
Iteration = 40 Error = 0.0297264637
Iteration = 41 Error = 0.0089338654
Iteration = 42 Error = 0.0021209819
Iteration = 43 Error = 0.0007476053
Iteration = 44 Error = 0.0002194713
Iteration = 45 Error = 0.0000616217
Iteration = 46 Error = 0.0000224668
Iteration = 47 Error = 0.0000067270
Iteration = 48 Error = 0.0000024241
Iteration = 49 Error = 0.0000007872
Iteration = 50 Error = 0.0000002790
Iteration = 51 Error = 0.0000001006
Iteration = 52 Error = 0.0000000364
Iteration = 53 Error = 0.0000000143
Iteration = 54 Error = 0.0000000059
Iteration = 55 Error = 0.0000000027
Iteration = 56 Error = 0.0000000015
```

```
figure;
contourf(X, Y, phi, '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.
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 = 1:20
        % === 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

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

Error

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
```