Diffusion Simulation

1 Introduction

This is a simulation of advection. Advection is the movement of a substance, in which the properties of the substance are carried with it even though it is moving. Generally the advected substance is a fluid. The properties that are carried along are things like energy. The advection occurs due to a velocity vector field. A good way to imagine advection is to think of a drop of ink being carried off by a river. For this simulation the substance of the material being advected is unknown, and what is causing the velocity vector field is unkown as well. What we do know is this:

1. The domain of the simulation is in two spacial dimensions:

$$\Omega = [\frac{-\pi}{2}, \frac{\pi}{2}]^2$$

2. The velocity vector field (U) is given by:

$$\mathbf{U} = (u, v)^T$$

where

$$u(x, y, t) = -cos(x)sin(y)cos(t)$$
$$v(x, y, t) = sin(x)cos(y)cos(t)$$

3. We want to find the solution to $\rho = \rho(x, y, t)$ where ρ satisfies the advection equation:

$$\frac{\delta \rho}{\delta t} + \mathbf{U} \cdot \bigtriangledown \rho = 0$$

with boundary conditions:

$$\rho(x = -\pi/2) = 0$$
$$\rho(x = +\pi/2) = 0$$
$$\rho(y = -\pi/2) = 0$$
$$\rho(y = +\pi/2) = 0$$

In order to simulate the system given, I set up the initial condition given by these conditions:

$$\rho(x, y, t = 0) = 1 \texttt{ iff } (x, y) \in C,$$
$$\rho(x, y, t = 0) = 0 \texttt{ otherwise}$$

where C is a circle with center (1,0) and radius .25.

Then I implement the upwind scheme for the advection equation in order to solve for ρ at $t_{final} = \pi$. The simulation discretises the domain into 100 grid nodes in each spacial direction. To iterate through the simulation until t_{final} a time step of $\Delta t = .2\Delta x$ is used. Where $\Delta x = x_{max} - x_{min}/N$ where $N = 100$ and $\Delta x = \Delta y$.

2 Algorithms

I implement an upwind scheme to solve the equation numerically. The upwind scheme I used for the advection equation

$$\frac{\delta\rho}{\delta t} + \mathbf{U} \cdot \bigtriangledown\rho = 0 \tag{1}$$

is given by taking a first-order forward difference to approximate ρ_t and a first-order backward difference for ρ_x and ρ_y, which results in this equation:

1. Start with the advection equation:

   $$\frac{\delta\rho}{\delta t} + \mathbf{U} \cdot \bigtriangledown\rho = 0$$

   and perform the del operation:

   $$\frac{\delta\rho}{\delta t} + \mathbf{U}(\frac{\delta\rho}{\delta x}\vec{i} + \frac{\delta\rho}{\delta y}\vec{j}) = 0$$

2. Then distribute $\mathbf{U}$ and multiply it times the unit directions to obtain the equation:

   $$\frac{\delta\rho}{\delta t} + u\frac{\delta\rho}{\delta x} + v\frac{\delta\rho}{\delta y} = 0$$

3. Take the first-order forward difference to approximate ρ_t

   $$\frac{\rho_{i,j}^{n+1} - \rho_{i,j}^{n}}{\Delta t} + u\frac{\delta\rho}{\delta x} + v\frac{\delta\rho}{\delta y} = 0$$

4. Take the first-order backwards difference to approximate ρ_x and ρ_y

   $$\frac{\rho_{i,j}^{n+1} - \rho_{i,j}^{n}}{\Delta t} + u\frac{\rho_{i,j}^{n} - \rho_{i-1,j}^{n}}{\Delta x} + v\frac{\rho_{i,j}^{n} - \rho_{i,j-1}^{n}}{\Delta y} = 0$$

5. Rearrange to obtain a first-order upwind scheme:

   $$\frac{\rho_{i,j}^{n+1} - \rho_{i,j}^{n}}{\Delta t} = -u\frac{\rho_{i,j}^{n} - \rho_{i-1,j}^{n}}{\Delta x} - v\frac{\rho_{i,j}^{n} - \rho_{i,j-1}^{n}}{\Delta y} \tag{2}$$

   where Δt is the time step and Δx and Δy are the spacial resolutions. The superscripts, n, references the time frame and the subscripts, i and j reference the spacial frame. So, n+1, denotes the subsequent time interval and n denotes the current one. i,j represents the grid node we are looking at, i+1,j is the node to the right, i-1,j is the node to the left, i,j+1 is the grid node above, and i,j-1 is the grid node below. ρ represents the equation we are trying to solve for.

Equation (2) can be rearranged to produce an iteratable equation for solving for $\rho_{i,j}^{n+1}$:

$$\rho_{i,j}^{n+1} = \rho_{i,j}^{n} - \frac{u\Delta t}{\Delta x}(\rho_{i,j}^{n} - \rho_{i-1,j}^{n}) - \frac{v\Delta t}{\Delta y}(\rho_{i,j}^{n} - \rho_{i,j-1}^{n}) \tag{3}$$

Equation (3) assumes however that the velocity vector field, **U**, is positive in both the x and y directions. Because our field **U** is not always positive in both directions, we have to modify our first-order backwards differences with respect to u and v. So, depending on the signs of u and v we are left with four different schemes:

1. If u is positive and v is positive at grid node (i,j):

$$\rho_{i,j}^{n+1} = \rho_{i,j}^{n} - \frac{u\Delta t}{\Delta x}(\rho_{i,j}^{n} - \rho_{i-1,j}^{n}) - \frac{v\Delta t}{\Delta y}(\rho_{i,j}^{n} - \rho_{i,j-1}^{n}) \tag{3.1}$$

2. If u is positive and v is negative at grid node (i,j):

$$\rho_{i,j}^{n+1} = \rho_{i,j}^{n} - \frac{u\Delta t}{\Delta x}(\rho_{i,j}^{n} - \rho_{i-1,j}^{n}) - \frac{v\Delta t}{\Delta y}(\rho_{i,j+1}^{n} - \rho_{i,j}^{n}) \tag{3.2}$$

3. If u is negative and v is positive at grid node (i,j):

$$\rho_{i,j}^{n+1} = \rho_{i,j}^{n} - \frac{u\Delta t}{\Delta x}(\rho_{i+1,j}^{n} - \rho_{i,j}^{n}) - \frac{v\Delta t}{\Delta y}(\rho_{i,j}^{n} - \rho_{i,j-1}^{n}) \tag{3.3}$$

4. If u is negative and v is negative at grid node (i,j):

$$\rho_{i,j}^{n+1} = \rho_{i,j}^{n} - \frac{u\Delta t}{\Delta x}(\rho_{i+1,j}^{n} - \rho_{i,j}^{n}) - \frac{v\Delta t}{\Delta y}(\rho_{i,j+1}^{n} - \rho_{i,j}^{n}) \tag{3.4}$$

2.1 My Code

My code uses the math presented above. Specifically, my code follows these steps:

1. Set number of nodes, N, equal to 100
2. Establish the domain through the variables xmin, xmax, ymin, and ymax
3. Discretise the x and y dimensions and establish variables dx and dy which correspond to Δx and Δy
4. Establish a time step and simulation length.
5. Set up initial conditions and draw the first frame
6. Perform this loop until t_{final} is reached.

- Calculate u and v for the current time t
- Calculate for the boundary conditions
- Update ρ for t + time step using Equations (3.1)-(3.4) depending on the value of u and v
- Draw the consecutive frame
- Note: Inside the loop the steps presented above are done for each grid node individually since the velocity vector $\mathbf{U}$ has a different value at each grid node and there are conditions for updating ρ that are based on $\mathbf{U}$ for a specific grid point.

3 Results

In `MatLab`, my simulation code is invoked at the prompt by calling the script

`AdvectionSimulation`

`AdvectionSimulation` will run a simulation with 100 grid nodes along the x-dimension and y-dimension and 500 time steps. The value of ρ is represented on the plot by the z-dimension.

See Figures 1-6 for a look at the velocity vector field $\mathbf{U}$ over time:

See Figures 7-16 for a look at the value of ρ over time:

4 Conclusion

I was not able to implement an exact solution, because I was not able to solve for ρ but I believe my simulation is accurate. It is most likely first order accurate, since I implemented a first order upwind scheme as described in the *Algorithms* section. Since there is no source, the value of ρ decreases as it advects over time. The fluid (I take ρ to symbolize a fluid) moves with the vector field and although it dissipates slowly, it never simply dissolves. Because the fluid moves as a whole in accordance with the velocity vector field over time and decreases in value slowly since there is no source, and this is the expected behavior, I believe I have implemented the simulation correctly.

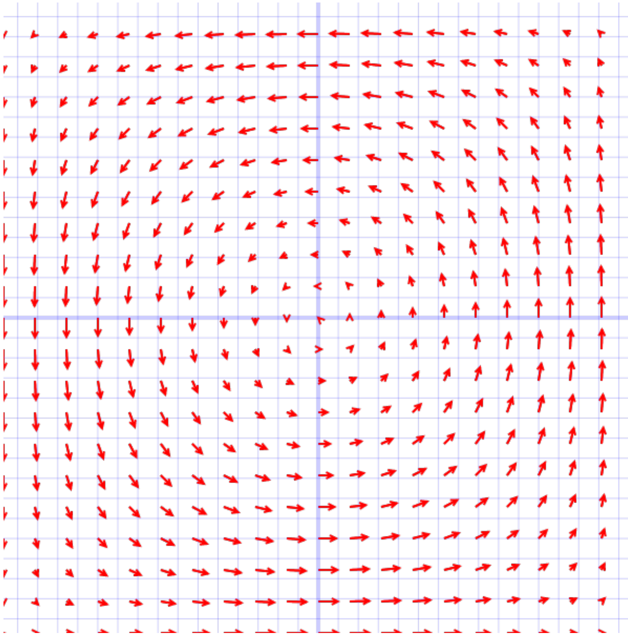

Figure 1: $t = 0$ (initial condition)

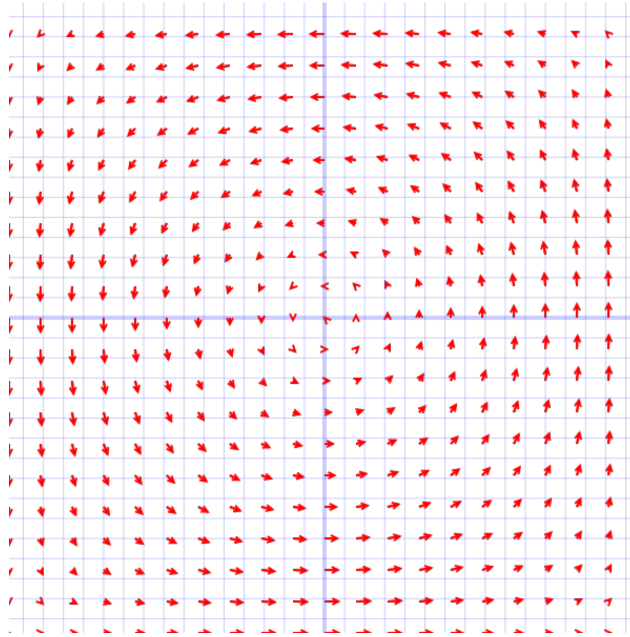

Figure 2: $t = \frac{t_{final}}{4}$

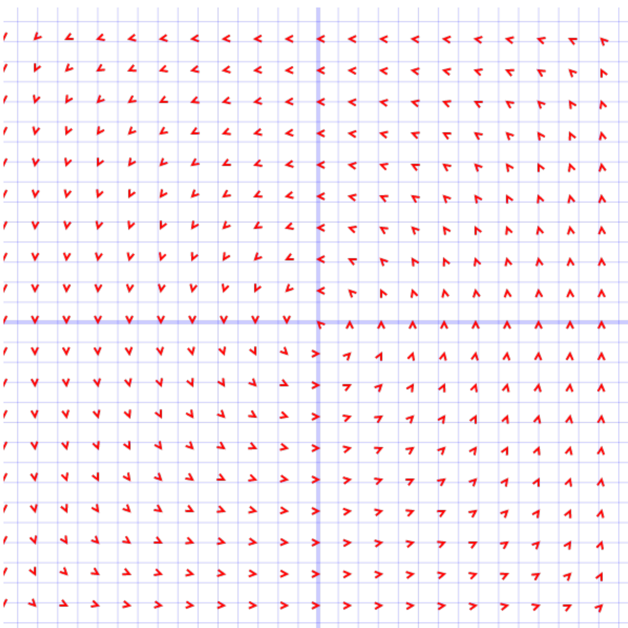

Figure 3: $t \approx \frac{t_{final}}{2}$ (just before halfway)

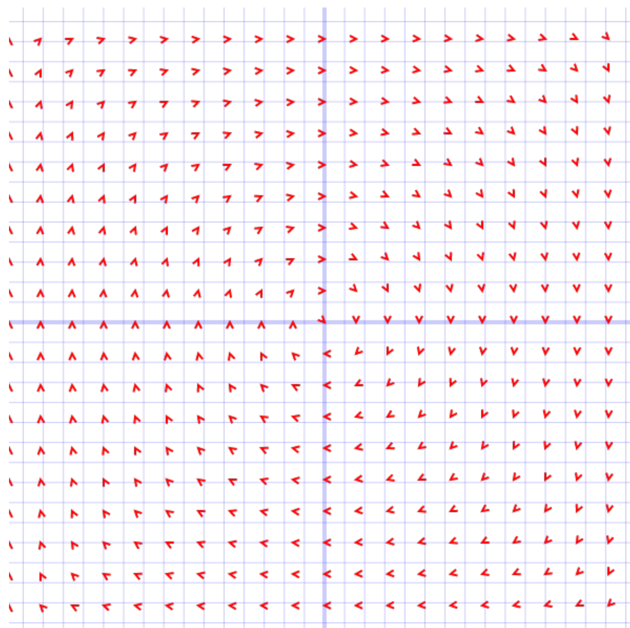

Figure 4: $t \approx \frac{t_{final}}{2}$ (just after halfway)

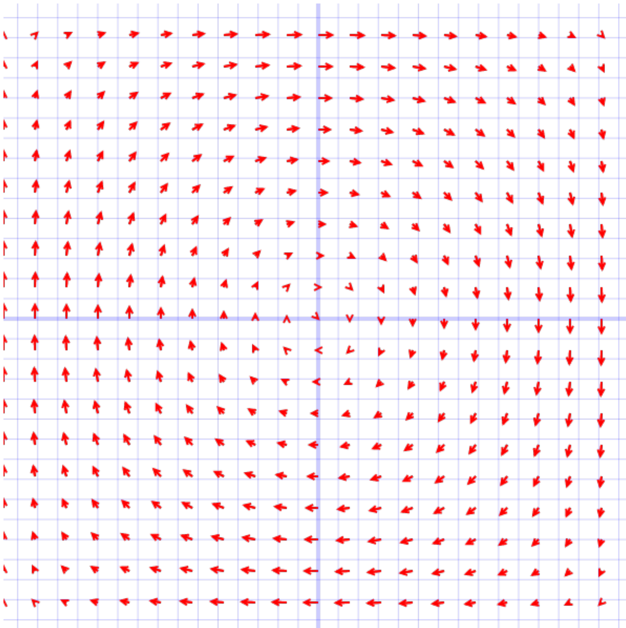

Figure 5: $t = 3\frac{t_{final}}{4}$

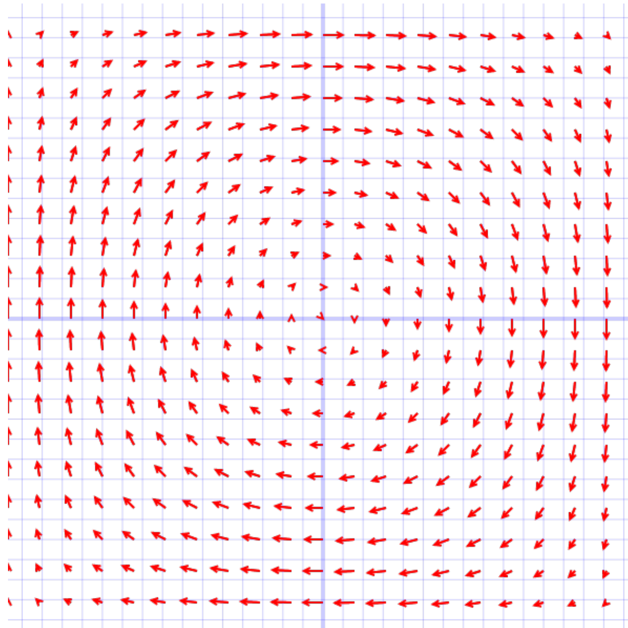

Figure 6: $t = t_{final}$

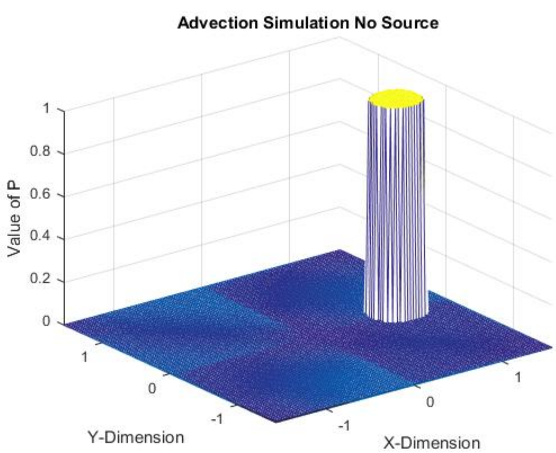

Figure 7: $t = 0$ (initial condition)

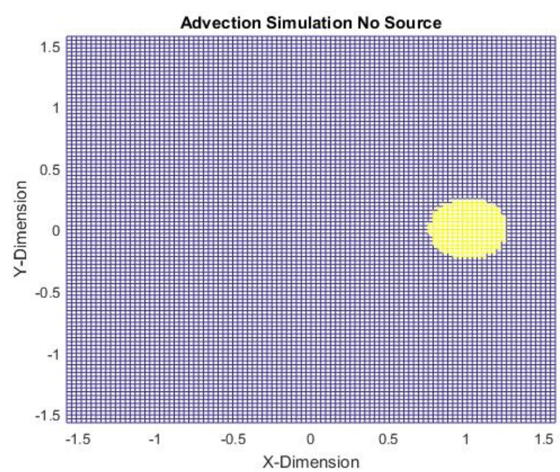

Figure 8: $t = 0$ (initial condition) aerial

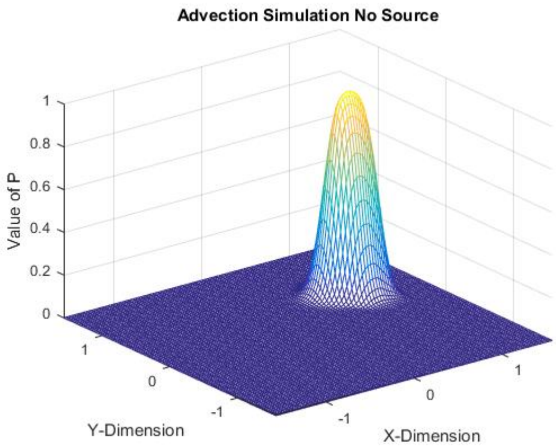

Figure 9: $t = \frac{t_{final}}{4}$

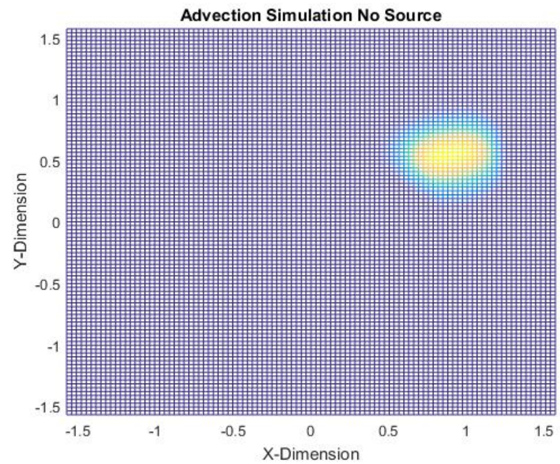

Figure 10: $t = \frac{t_{final}}{4}$) aerial

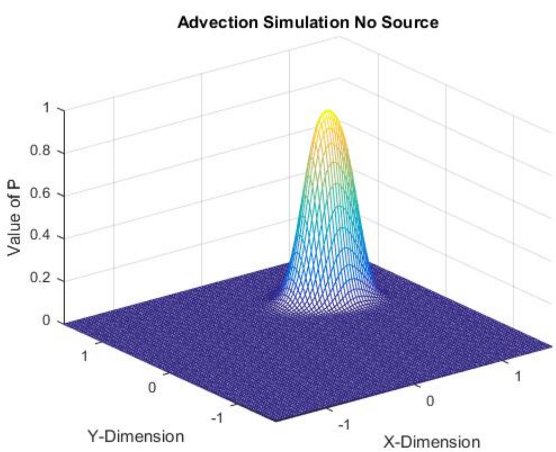

Figure 11: $t = \frac{t_{final}}{2}$

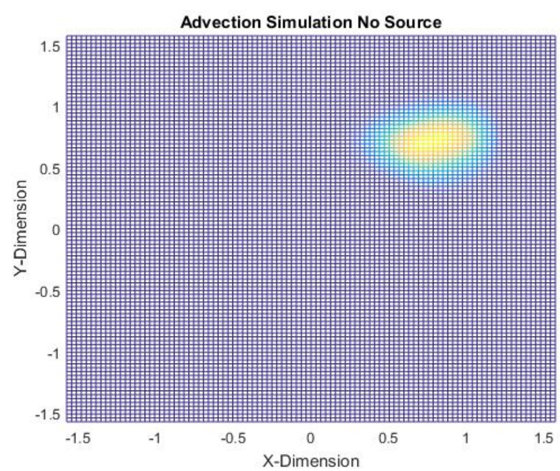

Figure 12: $t = \frac{t_{final}}{2}$ aerial

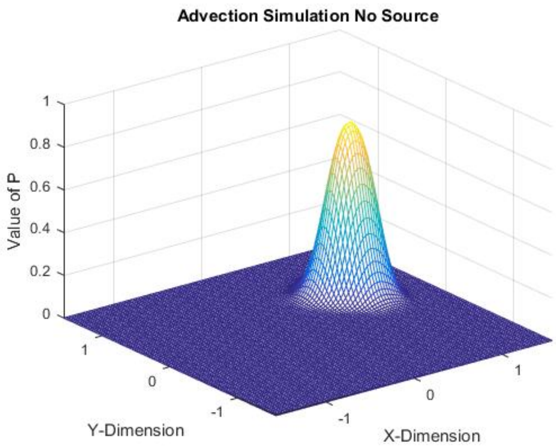

Figure 13: $t = 3\frac{t_{final}}{4}$

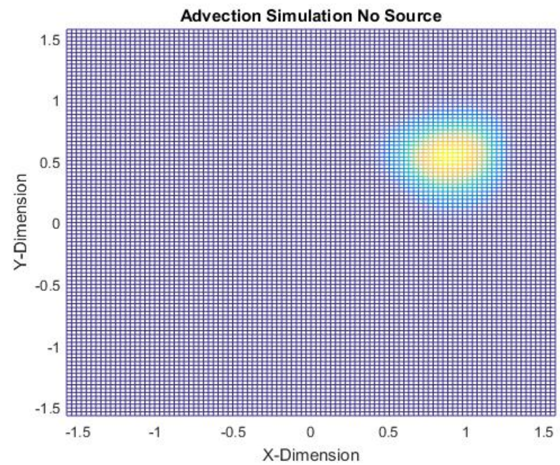

Figure 14: $t = 3\frac{t_{final}}{4}$ aerial

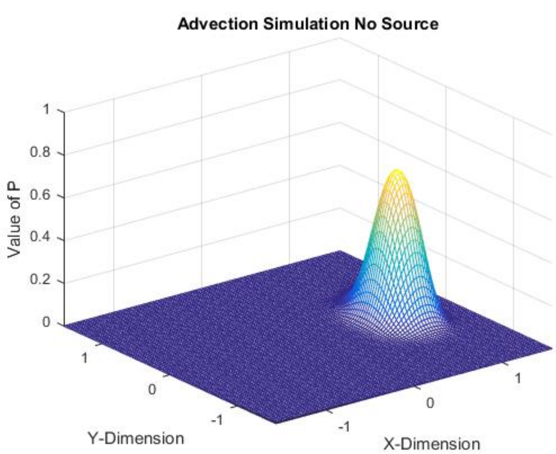

Figure 15: $t = t_{final}$

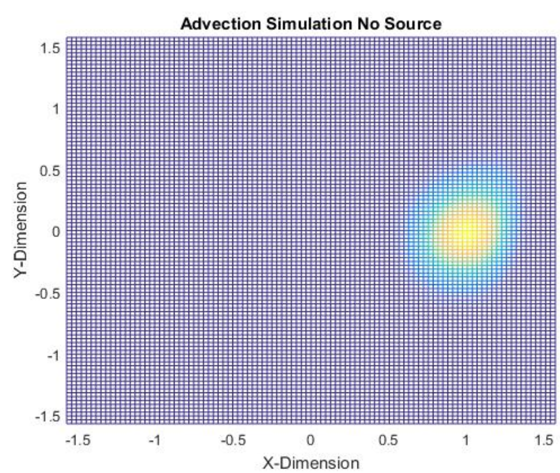

Figure 16: $t = t_{final}$ aerial

A Implementation in MatLab

The `MatLab` implementation:

`AdvectionSimulation.m`

```
close all;
clear all;

N = 100; % in each spacial direction
u(N+1,N+1) = 0;
v(N+1,N+1) = 0;
p(N+1,N+1) = 0;
p1(N+1,N+1) = 0;

%Bounds
xmin = -pi / 2;
xmax = pi / 2;
ymin = -pi / 2;
ymax = pi / 2;

%discretise x and y
dx = (xmax - xmin)/N;
x = xmin : dx : xmax;
dy = (ymax - ymin)/N;
y = xmin : dx : xmax;

%Time
timeStep = .2 * dx;
simLength = pi;
t = 0;

%%%%%%%%%%%%%%%%%% Initial conditions and initial frame %%%%%%%%%%%%%%%%%%%%%%%%
for ix = 1:N+1
    for iy = 1:N+1
        if (sqrt((x(ix)-1)^2+(y(iy)-0)^2) <= .25)
            p(iy,ix) = 1;
        end
    end
end

for ix = 1:N+1
    for iy = 1:N+1
```

```
        u(iy,ix) = -cos(x(ix))*sin(y(iy))*cos(t);
    end
end
for ix = 1:N+1
    for iy = 1:N+1
        v(iy,ix) = sin(x(ix))*cos(y(iy))*cos(t);
    end
end

mesh(x,y,p); hold on;
%quiver(x,y,u,v);
axis([-pi/2 pi/2 -pi/2 pi/2]);
xlabel('X-Dimension');
ylabel('Y-Dimension');
zlabel('Value of P');
title('Advection Simulation No Source');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%% Loop through until simLength %%%%%%%%%%%%%%%%%%%%%%%%%%%%
for t = timeStep: timeStep: simLength

    % Establish u and v for time t %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for ix = 1:N+1
        for iy = 1:N+1
            u(iy,ix) = -cos(x(ix))*sin(y(iy))*cos(t);
        end
    end
    for ix = 1:N+1
        for iy = 1:N+1
            v(iy,ix) = sin(x(ix))*cos(y(iy))*cos(t);
        end
    end
    U = u*timeStep/dx; %Variable to make calculations easier to type
    V = v*timeStep/dy; %Variable to make calculations easier to type
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %BOUNDARY CONDITIONS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for iy = 1:N+1
        p(iy,1) = 0;
        p1(iy,1) = 0;
    end
```

```
    for iy = 1:N+1
        p(iy,N+1) = 0;
        p1(iy,N+1) = 0;
    end
    for ix = 1:N+1
        p(1,ix) = 0;
        p1(1,ix) = 0;
    end
    for ix = 1:N+1
        p(N+1,ix) = 0;
        p1(N+1,ix) = 0;
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    for ix = 2:N
        for iy = 2:N
            if U(iy,ix) >= 0 && V(iy,ix) >= 0
                p1(iy,ix) = p(iy,ix) - U(iy,ix)*(p(iy,ix)-p(iy,ix-1)) ...
                           - V(iy,ix)*(p(iy,ix)-p(iy-1,ix));
            end
            if U(iy,ix) >= 0 && V(iy,ix) < 0
                p1(iy,ix) = p(iy,ix) - U(iy,ix)*(p(iy,ix)-p(iy,ix-1)) ...
                           - V(iy,ix)*(p(iy+1,ix)-p(iy,ix));
            end
            if U(iy,ix) < 0 && V(iy,ix) >= 0
                p1(iy,ix) = p(iy,ix) - U(iy,ix)*(p(iy,ix+1)-p(iy,ix)) ...
                           - V(iy,ix)*(p(iy,ix)-p(iy-1,ix));
            end
            if U(iy,ix) < 0 && V(iy,ix) < 0
                p1(iy,ix) = p(iy,ix) - U(iy,ix)*(p(iy,ix+1)-p(iy,ix)) ...
                           - V(iy,ix)*(p(iy+1,ix)-p(iy,ix));
            end
        end
    end

    p = p1;

    %%%%%%%%%%%% Plot the current state of the simulation
    clf;
    mesh(x,y,p); hold on;
%    quiver(x,y,u,v);
    axis([-pi/2 pi/2 -pi/2 pi/2 0 1]);
```

```
    xlabel('X-Dimension');
    ylabel('Y-Dimension');
    zlabel('Value of P');
    title('Advection Simulation No Source');
%     view(0,90);
    pause(.02);

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```