Solving the Advection Equation in 2D

Note: I confirm that I did not use codes from the web or from past years' assignments and that the work I submit is my own and my own only.

1 Introduction

What is advection? Advection is the transfer of heat or matter by the flow of a fluid, especially horizontally in the atmosphere or the sea. An example of advection can be leaves in the wind or cars on the freeway. We can denote the variable u to represent density and the variable $\vec{v} = [c_1, c_2, c_3]$. In general, when looking at 1D advection, we get the following equation:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \quad where\ u = u(x,t) \tag{1}$$

Rather than solving the 1D advection case, the goal of this homework assignment is to solve the 2D advection equation:

$$\frac{\partial u}{\partial t} + c_1\frac{\partial u}{\partial x} + c_2\frac{\partial u}{\partial y} = 0 \tag{2}$$

in the domain $[-\frac{\pi}{2}, \frac{\pi}{2}]x[-\frac{\pi}{2}, \frac{\pi}{2}]$ where the velocity vectors c_1 and c_2 are given by:

$$\begin{aligned} c_1(x,y,t) &= -cos(x)sin(y)cos(t) \\ c_2(x,y,t) &= sin(x)cos(y)cos(t) \end{aligned}$$

The imposed boundary conditions are as follows:

$$\begin{aligned} u(x = -\frac{\pi}{2}) &= 0, \\ u(x = +\frac{\pi}{2}) &= 0, \\ u(y = -\frac{\pi}{2}) &= 0, \\ u(y = +\frac{\pi}{2}) &= 0, \end{aligned}$$

and with the initial data:

$$\begin{cases} u(x,y,t=0) = 1, & \text{(x,y)} \in \text{C} \\ u(x,y,t=0) = 0, & \text{otherwise} \end{cases}$$

where C is a circle with center(1,0) and radius 0.25.

Also, this assignment is meant to introduce students to implementing the upwind scheme. In general, upwind scheme characterizes a class of numerical discretization methods for solving hyperbolic PDEs by using differencing biased in the direction determined by the sign of the characteristic speeds. Also, you need to ensure the Courant Friedrichs Lewy condition (CFL) is satisfied for stability:

$$|\frac{c\Delta t}{\Delta x}| \leq 1$$

This assignment asks the following:

a) Implement the upwind scheme for this problem and solve the equation from t = 0 to t = π using 100 grid points in each spatial direction and a time step of $\Delta t = 0.2\Delta x$.
b) Explain approach and provide a few snapshots of the solution using the MATLAB command **mesh** and a few snapshots of the velocity field (c_1, c_2) using the MATLAB command **quiver**.

2 Derivation of Equations

Use conservation laws in 1D to find the advection equation. The rate at which "stuff" is changing is equal to the rate at which the "stuff" is created or destroyed plus "whatever comes in or out (aka the flux)." The 1D advection derivation looks like the following:

$$\begin{aligned} \frac{d}{dt}\int_a^b u dx &= \int_a^b S dx - F_a - F_b \\ \int_a^b \frac{\partial u}{\partial t} dx &= \int_a^b S dx - \int_a^b \frac{\partial F}{\partial x} dx \\ &\rightarrow \frac{\partial u}{\partial t} + \frac{\partial F}{\partial x} = S \end{aligned}$$

Where u is your density, S is the source or sink ($S > 0$ if created, $S < 0$ if destroyed), F is the flux term in the x-direction. This derivation holds for 2D and 3D cases, you would just need to add another flux term for the y- and z-directions.

$$
\begin{aligned}
\frac{\partial u}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial x} &= S \\
\frac{\partial u}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial x} + \frac{\partial K}{\partial x} &= S \\
\frac{\partial u}{\partial t} + \bigtriangledown \cdot J &= S \\
J = \begin{bmatrix} F \\ G \\ K \end{bmatrix}
\end{aligned}
$$

The mathematical model for advection utilizes the nabla operator to denote gradient. This operator is used on the density variable, which has the dot product with the velocity field $\vec{c}$.

$$\frac{\partial u}{\partial t} + \vec{c} \cdot \bigtriangledown u = S \tag{3}$$

3 Algorithms

In order to solve this problem, we utilize tools from the Euler Step method. The Euler method is a numerical method to solve first order first degree differential equation with a given initial value. It is the most basic explicit method for numerical integration of ordinary differential equations and is the simplest Runge Kutta method. The 1D case looks like the following for all grid points i:

$$u_i^{n+1} = u_i^n - c\frac{\Delta t}{\Delta x}(u_{i+1}^n - u_i^n)$$

Lets consider the 1D case to discretize, we can rewrite the 1D PDE advection equation in the following way, assuming the source/sink term is zero:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + c\frac{u_{i+1}^n - u_i^n}{\Delta x} = 0 \tag{4}$$

But now we need to worry about when c is positive ($c \geq 0$) or negative ($c < 0$). The Euler Step becomes a bit different when looking at positive/negative c values, this is called "upwind":

$$c \geq 0 : \quad u_i^{n+1} = u_i^n - c\frac{\Delta t}{\Delta x}(u_i^n - u_{i-1}^n)$$

$$c < 0 : \quad u_i^{n+1} = u_i^n - c\frac{\Delta t}{\Delta x}(u_{i+1}^n - u_i^n)$$

Now lets consider the 2D advection case, which will have two different velocity fields c_1 and c_2:

$$\frac{\partial u}{\partial t} + c_1 \frac{\partial u}{\partial x} + c_2 \frac{\partial u}{\partial y} = 0$$

As we did with the 1D case, we need to worry about the sign of c_1 and c_2. There will be four main cases of the velocity field that we will be interested in:

$$\begin{aligned} &1. c_1 \geq 0, c_2 \geq 0 \\ &2. c_1 \geq 0, c_2 < 0 \\ &3. c_1 < 0, c_2 \geq 0 \\ &4. c_1 < 0, c_2 < 0 \end{aligned} \tag{5}$$

Therfore to approximate the 2D advection equation with the upwind scheme, we need to use the following equations:

$$c_1 \begin{cases} \frac{u^n_{i+1,j} - u^n_{i,j}}{\Delta x} & \text{if } c_1 < 0 \\ \frac{u^n_{i,j} - u^n_{i-1,j}}{\Delta x} & \text{if } c_1 \geq 0 \end{cases} \qquad c_2 \begin{cases} \frac{u^n_{i,j+1} - u^n_{i,j}}{\Delta y} & \text{if } c_2 < 0 \\ \frac{u^n_{i,j} - u^n_{i,j-1}}{\Delta y} & \text{if } c_2 \geq 0 \end{cases}$$

Now that we are able to know the four different cases for the 2D advection case, we can solve for the first order upwind scheme in terms of $u^{n+1}_{i,j}$. Referring to **(5)** above for the following correlations:

$$\begin{aligned} 1.\quad u^{n+1}_{i,j} &= u^n_{i,j} - c_1 \Delta t\left(\frac{u^n_{i,j} - u^n_{i-1,j}}{\Delta x}\right) - c_2 \Delta t \frac{u^n_{i,j} - u^n_{i,j-1}}{\Delta y} \\ 2.\quad u^{n+1}_{i,j} &= u^n_{i,j} - c_1 \Delta t\left(\frac{u^n_{i,j} - u^n_{i-1,j}}{\Delta x}\right) - c_2 \Delta t \frac{u^n_{i,j+1} - u^n_{i,j}}{\Delta y} \\ 3.\quad u^{n+1}_{i,j} &= u^n_{i,j} - c_1 \Delta t\left(\frac{u^n_{i+1,j} - u^n_{i,j}}{\Delta x}\right) - c_2 \Delta t \frac{u^n_{i,j} - u^n_{i,j-1}}{\Delta y} \\ 4.\quad u^{n+1}_{i,j} &= u^n_{i,j} - c_1 \Delta t\left(\frac{u^n_{i+1,j} - u^n_{i,j}}{\Delta x}\right) - c_2 \Delta t \frac{u^n_{i,j+1} - u^n_{i,j}}{\Delta y} \end{aligned}$$

Now the final step is to code these equations into MATLAB and use the two commands **mesh** and **quiver** to obtain snapshots of the solution and velocity field (c_1, c_2).

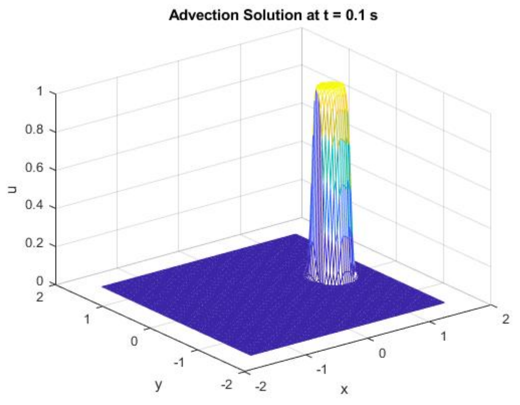
4 Implementation in Matlab and Results

The Matlab implementation, with comments, is given here:

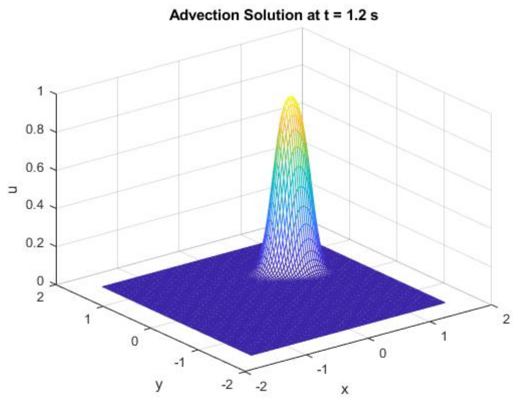
```
%ME 17 HW4 - Alex Nguyen

% I confirm that I did not use codes from the web or from past years'
% assignments and that the work I submit is my own and my own only

clc; clear; close all;

%% Given

% Domain:
xmin = -pi/2; xmax = pi/2; mx = 100;
ymin = -pi/2; ymax = pi/2; my = 100;

% Discretize the x-axis and the y-axis:
x = linspace(xmin, xmax, mx); dx = x(2) - x(1);
y = linspace(ymin, ymax, my); dy = y(2) - y(1);

% Initial and final time:
t = 0; tfinal = pi;
dt = 0.2*dx; % time step

% Initial data:
un = zeros(mx, my);
for i = 1:mx
    for j = 1:my
        if sqrt((x(i)-1)^2 + (y(j)-0)^2) <= 0.25
            un(i,j) = 1;
        end
    end
end

%% Computation:

% Preallocation
unp1 = zeros(mx,my);
c1   = zeros(mx,my);
c2   = zeros(mx,my);

while t < tfinal
```

```
    if t + dt > tfinal
        dt = tfinal - t; % ensures tfinal is reached
    end

    % Define the velocity field:
    for i = 1:mx
        for j = 1:my
            c1(i,j) = -cos(x(i))*sin(y(j))*cos(t);
            c2(i,j) =  sin(x(i))*cos(y(j))*cos(t);
        end
    end

    figure(1);
    quiver(x, y, c1', c2',0);
    xlabel('x'); ylabel('y'); title('Vector Field (c_1, c_2)'); pause(dt);

for i = 2:mx-1
        for j = 2:my-1
            if c1(i,j) < 0 && c2(i,j) < 0

                     unp1(i,j) = un(i,j)-dt*c1(i,j)*(un(i+1,j)-un(i,j))/dx ...
                         - dt*c2(i,j)*(un(i,j+1)-un(i,j))/dy;

         elseif c1(i,j) >= 0 && c2(i,j) >= 0

                     unp1(i,j) = un(i,j)-dt*c1(i,j)*(un(i,j)-un(i-1,j))/dx ...
                         - dt*c2(i,j)*(un(i,j)-un(i,j-1))/dy;

         elseif c1(i,j) >= 0 && c2(i,j) < 0

                     unp1(i,j) = un(i,j)-dt*c1(i,j)*(un(i,j)-un(i-1,j))/dx ...
                         - dt*c2(i,j)*(un(i,j+1)-un(i,j))/dy;

         elseif c1(i,j) < 0 && c2(i,j) >= 0

                     unp1(i,j) = un(i,j)-dt*c1(i,j)*(un(i+1,j)-un(i,j))/dx ...
                         - dt*c2(i,j)*(un(i,j)-un(i,j-1))/dy;

         end
```

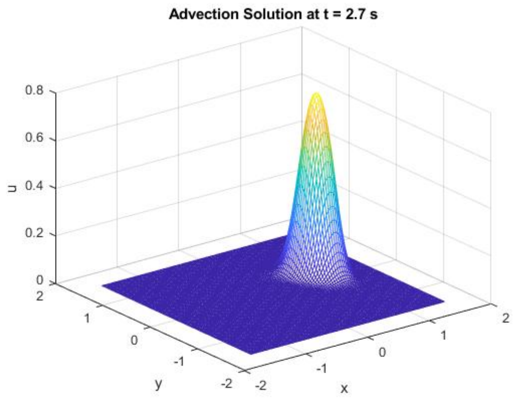
```
     end
 end

    % Update time and prepare for next iterate:
    t = t + dt;
    un = unp1;

    figure(2);
    mesh(x,y,un');
    xlabel('x'); ylabel('y'); zlabel('u');
    title('Advection Solution at t = \pi');
    pause(dt);
end
```



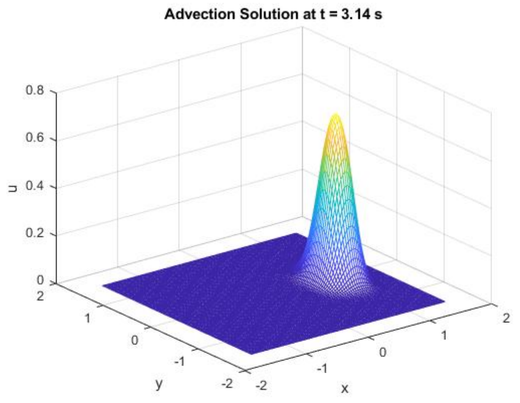
(a) Initial Advection solution at t = 0.1 s



(b) Advection Solution at t = 1.2 s

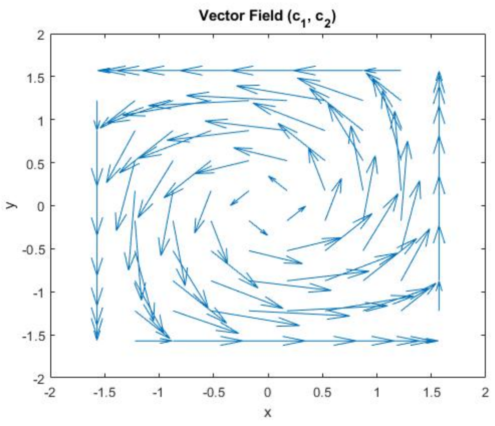


(c) Advection Solution at t = 2.7 s

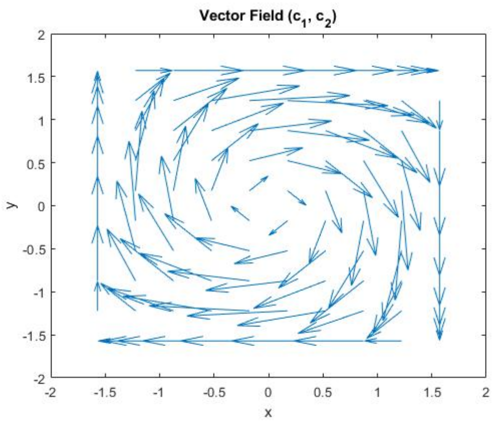


(d) Advection Solution at t = π

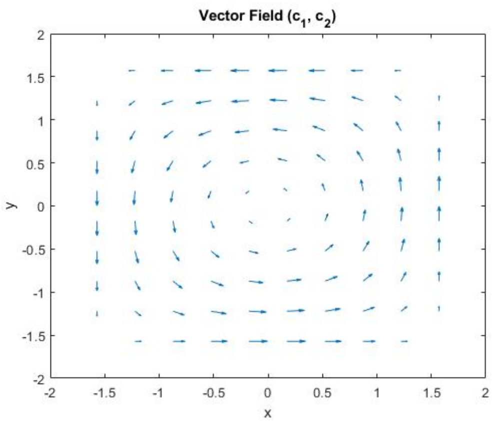
Figure 1: Advection Solution at Different Time Values



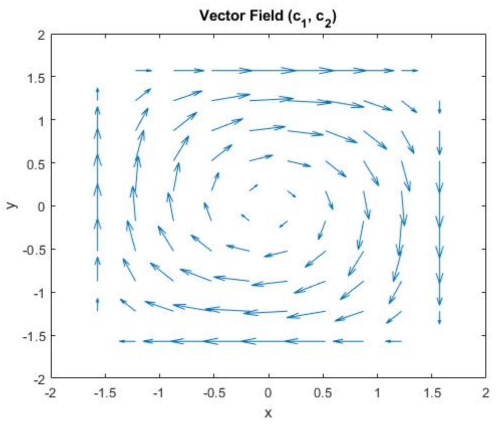
(a) Velocity Field Pointing Counter-Clockwise



(b) Velocity Field Pointing Clockwise



(c) Velocity Field About to Turn



(d) Velocity Field After Turning

Figure 2: Velocity Field (c_1, c_2) at Different Time Values Using Smaller m_x and m_y Values