# MPCS 51087
# Project 1, Milestone 1

January 3, 2024

## 1 Serial Advection

**Due Date: Sunday, Jan 7, 2024 by 6pm**

The advection equation is a hyperbolic partial-differential equation (PDE) governing the conserved movement of a specified distribution of material with a given velocity. In 2D, the governing equation is:

$$\frac{\partial C}{\partial t} + u\frac{\partial C}{\partial x} + v\frac{\partial C}{\partial y} = 0 \tag{1}$$

where $C(x, y, t)$ is some scalar concentration and $\vec{v} = (u, v)$ is the 2D velocity. In general $\vec{v} = \vec{v}(x, y, t)$ but in this milestone we'll assume that the velocity is constant in space and time and can thus be represented by two scalar values.

Those who have taken a numerical analysis course are familiar with the many techniques used for solving PDEs on computers. Those who haven't can find everything they need to complete the exercise in this document. In particular, finite difference approximations allow us to convert this PDE to a discrete system of equations. Given an intial condition $C(x, y, 0) = C_0(x, y)$ and specified behavior on the boundaries, we can then use the discrete equation to estimate the solution on a digital computer.

Your task is to numerically estimate the solution to Equation 1 using the Lax method. The goal is to take the initial 2D system state at $t = 0$ and to simulate how the system will develop over time. We can begin this by using the finite difference method, wherein the continuous spatial and temporal dependencies of Equation 1 are discretized. We can then iterate over discrete timesteps $n$, computing $C^{n+1}$ based off of $C^n$ using an explicit expression. To determine the expression used to move the system forward in time, we will first consider a forward time, center space (FTCS) finite difference approximation:

$$\frac{C^{n+1}_{i,j} - C^n_{i,j}}{\Delta t} + u\frac{C^n_{i+1,j} - C^n_{i-1,j}}{2\Delta x} + v\frac{C^n_{i,j+1} - C^n_{i,j-1}}{2\Delta y} = 0 \tag{2}$$

where $C^n_{i,j}$ is the scalar field (e.g. temperature) at each physical location $i, j$ on a discretized two-dimensional mesh, n is the discrete time unit, $\Delta x$ and $\Delta y$ are the mesh spacings in the $x$ and $y$ directions respectively, and $(u, v)$ are the two components of the velocity field. With the discrete initial condition $C^0_{i,j}$ specified, this equation can be iterated to get the new values $C^{n+1}_{i,j}$ at the next time step, by re-writing Equation 2 into the following form:

$$C^{n+1}_{i,j} = C^n_{i,j} - \frac{\Delta t}{2}\left[u\frac{C^n_{i+1,j} - C^n_{i-1,j}}{\Delta x} + v\frac{C^n_{i,j+1} - C^n_{i,j-1}}{\Delta y}\right] \tag{3}$$

We can further simplify things for this assignment by assuming that we have evenly discretized our spatial mesh in both dimensions, such that $\Delta x = \Delta y$, so that our FTCS finite difference expression becomes:

$$C^{n+1}_{i,j} = C^n_{i,j} - \frac{\Delta t}{2\Delta x}\left[u\left(C^n_{i+1,j} - C^n_{i-1,j}\right) + v\left(C^n_{i,j+1} - C^n_{i,j-1}\right)\right] \tag{4}$$

Finally, we will alter our FTCS equation further by way of the Lax method, wherein the term $C^n_{i,j}$ has been averaged by way of its four neighbors, as:

$$C^n_{i,j} = \frac{C^n_{i-1,j} + C^n_{i+1,j} + C^n_{i,j-1} + C^n_{i,j+1}}{4} \tag{5}$$

to reach our final Lax method explicit timestep expression of:

$$\boxed{C^{n+1}_{i,j} = \frac{1}{4}\left(C^n_{i-1,j} + C^n_{i+1,j} + C^n_{i,j-1} + C^n_{i,j+1}\right) - \frac{\Delta t}{2\Delta x}\left[u\left(C^n_{i+1,j} - C^n_{i-1,j}\right) + v\left(C^n_{i,j+1} - C^n_{i,j-1}\right)\right]} \tag{6}$$

One thing to note regarding the Lax method, is that the approximation made in Equation 5 adds in a dispersive effect into the advection problem. This is not great from a fidelity stand point, but for the purposes of the homework assignment the dispersive effect makes for more interesting plots.

## 1.1 Serial Implementation

Write serial code in a HPC language of your choice (C, C++, or Fortran) to implement the above scheme. Begin with a 2D Gaussian pulse initial condition:

$$C_{i,j} = \exp\left(-\left(\frac{(x-x_0)^2}{2\sigma_x^2} + \frac{(y-y_0)^2}{2\sigma_y^2}\right)\right) \tag{7}$$

where $(x_0, y_0)$ is the center and $\sigma_x$ and $\sigma_y$ is the "spread" of the Gaussian blob in each dimension (which for this assignment we will fix to be $\sigma_x = \sigma_y = \frac{L}{4}$, where $L$ the Cartesian domain length). Once initialized, use the Lax scheme to integrate the 2D domain forward in time with some constant non-zero velocity. Use periodic, or "wrap-around", boundary conditions. Also, be sure that your code asserts that the Courant stability condition:

$$\Delta t \leq \frac{\Delta x}{\sqrt{2}|\vec{v}|} \tag{8}$$

is satisfied, so as to avoid numerical instability.

The pseudo code for the algorithm you will be implementing is given in Algorithm 1.

**Algorithm 1** Advection Pseudocode

```
Input N, NT, L, T, u, v
Allocate N × N grid for C^n_{i,j}
Allocate N × N grid for C^{n+1}_{i,j}
Δx = L/N
Δt = T/NT
Assert Δt ≤ Δx/√(2(u²+v²))        ▷ Ensure parameters meet the Courant stability condition
Initialize C^n_{i,j} for all i, j as a Gaussian        ▷ Equation 7
for 1 ≤ n ≤ NT do
    for 1 ≤ i ≤ N do
        for 1 ≤ j ≤ N do
            if i, j is on boundary then
                Apply periodic boundary conditions
            end if
            Update C^{n+1}_{i,j}        ▷ Using Equation 6
        end for
    end for
    Set C^n_{i,j} = C^{n+1}_{i,j} for all i, j        ▷ Or, swap references
end for
```

You should read in your inputs for N, NT, L, T, u, and v as command line arguments, in that order. This will allow us to experiment and test your code when grading or if we are helping to debug. You can hard code the Gaussian parameters, however. When you run your code, it should write to stdout the parameters it will simulate before beginning, as well as an estimate for the amount of memory required (when using double precision) to perform the simulation. Some good input parameters for testing and development that meet the Courant stability condition are:

- N = 400 (Matrix Dimension)
- NT = 20000 (Number of timesteps)
- L = 1.0 (Physical Cartesian Domain Length)
- T = 1.0e6 (Total Physical Timespan)
- u = 5.0e-7 (X velocity Scalar)
- v = 2.85e-7 (Y velocity Scalar)

which would be input to your program when running as:

```
./my_advection_program 400 20000 1.0 1.0e6 5.0e-7 2.85e-7
```

## 1.2 Plotting

Your will need to be able to visualize your data in order to ensure that your code is operating as expected. In HPC, it is common to output a file to disk containing data of interest that can be manipulated, analyzed, and plotted later. For this milestone, implement a serial function to write your 2D domain data for a single timestep to file in ASCII format. Use scientific notation and store the data in 2D matrix format. Do not worry about storing any Cartesian x,y location information to file – just store the contents of the matrix.

Then, using the graphing program or library of your choice, write a script that plots your data file. Some suggestions might be python, gnuplot, matlab, or julia.

Generate three plots – one showing your initialized Gaussian distribution, and another two plots at later timesteps that clearly show the development of the advection/dispersive process over time. Include all three of your plots in your write-up for the milestone. Alternatively, if you want, you are allowed to create an animation or video showing all timesteps – just note the file name/path.