University of Massachusetts, Lowell
Department of Mechanical Engineering

Finite Volume Solver for the Convection-Diffusion Equations

Patrick Heng

MECH 5200

Prof. D.J. Willis

03/30/25

Abstract

In the following, we explore a MATLAB based finite volume solver for the convection-diffusion equations over a rectangular domain. We develop the classic upwind and Lax Wendroff flux schemes and compare their ability to simulate the pure convection of a Gaussian initial condition. To this end, we also discuss the stability, numerical diffusion, and numerical dissipation introduced by both schemes. As a case study, we model the transport of a pollutant by convection and observe the effect of pumps and diffusion on the pollutant distribution over time.

Academic Integrity

I hereby affirm that the following work submitted is my own and that I have not received help from others outside of this class. Furthermore, I shall reference any resources used outside of class lectures and notes.

– P.V. Heng

Contents

1 Finite Volume Formulation of the Governing Equations

Consider the general transport equation for a conserved quantity, ϕ,

$$\frac{\partial \phi}{\partial t} + \nabla \cdot (\mathbf{u}\phi) = \nabla \cdot (D \nabla \phi) + R,$$

where $\mathbf{u}$ is a velocity of the fluid transporting ϕ, D is the diffusion coefficient, and R is a source term ($R = 0$ in our case). To derive the finite volume form of the equation, consider the 'volume' integral over an arbitrary cell, $V_{i,j}$ [1],

$$\int_{V_{i,j}} \frac{\partial \phi}{\partial t}\, dV + \int_{V_{i,j}} \nabla \cdot (\mathbf{u}\phi)\, dV = \int_{V_{i,j}} \nabla \cdot (D \nabla \phi)\, dV + \int_{V_{i,j}} R\, dV.$$

To manipulate the above equation, we assume (desire) our solution is regular enough that we may switch the time derivative and volume integral. Additionally, we may apply the divergence theorem to the flux terms to convert them into boundary integrals. Combining these manipulations yields,

$$\frac{d}{dt} \int_{V_{i,j}} \phi\, dV + \int_{\partial V_{i,j}} \phi \mathbf{u} \cdot \mathbf{n}\, dA = \int_{\partial V_{i,j}} D \nabla \phi \cdot \mathbf{n}\, dA + \int_{V_{i,j}} R\, dV.$$

where $\partial V_{i,j}$ denotes the boundary of $V_{i,j}$. Recall that an average quantity is defined by,

$$\phi^{avg} \coloneqq \frac{1}{V} \int_V \phi\, dV.$$

thus, dividing the previous equation by the cell volume, $V_{i,j}$,

$$\frac{d}{dt} \underbrace{\left(\frac{1}{V_{i,j}} \int_{V_{i,j}} \phi\, dV \right)}_{=\phi_{i,j}^{avg}} + \frac{1}{V_{i,j}} \int_{\partial V_{i,j}} \phi \mathbf{u} \cdot \mathbf{n}\, dA = \frac{1}{V_{i,j}} \int_{\partial V_{i,j}} D \nabla \phi \cdot \mathbf{n}\, dA + \underbrace{\frac{1}{V_{i,j}} \int_{V_{i,j}} R\, dV}_{=R_{i,j}^{avg}}.$$

We will hereby drop the average superscripts for convenience, leaving us with,

$$\frac{d\phi_{i,j}}{dt} + \frac{1}{V_{i,j}} \int_{\partial V_{i,j}} \phi \mathbf{u} \cdot \mathbf{n}\, dA = \frac{1}{V_{i,j}} \int_{\partial V_{i,j}} D \nabla \phi \cdot \mathbf{n}\, dA + R_{i,j}.$$

Consider the (very) special case of a Cartesian mesh with uniform grid spacing of Δx and Δy. The volume of each cell is then $V_{i,j} = \Delta x \Delta y$ and the boundary integrals break into 4 line integrals along the edges of the cell. For example, for the convective flux,

$$\begin{aligned} \frac{1}{V_{i,j}} \int_{\partial V_{i,j}} \phi \mathbf{u} \cdot \mathbf{n}\, dA = & \frac{1}{\Delta x \Delta y} \left[\left(\int_{y_{j-1/2}}^{y_{j+1/2}} \phi u dy \right)_{i+1/2,j} - \left(\int_{y_{j-1/2}}^{y_{j+1/2}} \phi u dy \right)_{i-1/2,j} \right] \\ & + \frac{1}{\Delta x \Delta y} \left[\left(\int_{x_{i-1/2}}^{x_{i+1/2}} \phi v dx \right)_{i,j+1/2} - \left(\int_{x_{i-1/2}}^{x_{i+1/2}} \phi v dx \right)_{i,j-1/2} \right]. \end{aligned}$$

where u and v are the x and y components of the velocity respectively. Observe that all manipulations we have performed thus far have been exact and no approximations have been made. It is here we introduce our first approximation where we will assume that the fluxes, ϕu and ϕv, are constant along the faces. This greatly simplifies the integrals as they now become the flux at the face times the size of the face,

$$\frac{1}{V_{i,j}} \int_{\partial V_{i,j}} \phi \mathbf{u} \cdot \mathbf{n}\, dA = \frac{1}{\Delta x \Delta y} \left[(\phi u)_{i+1/2,\, j} \Delta y - (\phi u)_{i-1/2,\, j} \Delta y \right] + \frac{1}{\Delta x \Delta y} \left[(\phi v)_{i,\, j+1/2} \Delta x - (\phi v)_{i,\, j-1/2} \Delta x \right].$$

[1] Since our problem is in 2D, the 'volume' integral is actually a double integral and the area integrals are actually line integrals along the boundaries of the 'volume'.

Let $F^x = \phi u$ and $F^y = \phi v$ denote the numerical convective fluxes, then we finally have,

$$\frac{1}{V_{i,j}} \int_{\partial V_{i,j}} \phi \mathbf{u} \cdot \mathbf{n} \, dA = \frac{1}{\Delta x} \left[F^x_{i+1/2,j} - F^x_{i-1/2,j} \right] + \frac{1}{\Delta y} \left[F^y_{i,j+1/2} - F^y_{i,j-1/2} \right].$$

A nearly identical analysis may be performed for the diffusive flux, but instead of using ϕu and ϕv, the components of the gradient are used. Explicitly, if $G^x = D\partial_x \phi$ and $G^y = D\partial_y \phi$ are the numerical diffusive fluxes, then,

$$\frac{1}{V_{i,j}} \int_{\partial V_{i,j}} D \nabla \phi \cdot \mathbf{n} \, dA = \frac{1}{\Delta x} \left[G^x_{i+1/2,j} - G^x_{i-1/2,j} \right] + \frac{1}{\Delta y} \left[G^y_{i,j+1/2} - G^y_{i,j-1/2} \right].$$

Combining all of the developed expressions yields the following,

$$\begin{aligned} \frac{d\phi_{i,j}}{dt} = &- \frac{1}{\Delta x} \left[F^x_{i+1/2,j} - F^x_{i-1/2,j} \right] - \frac{1}{\Delta y} \left[F^y_{i,j+1/2} - F^y_{i,j-1/2} \right] \\ &+ \frac{1}{\Delta x} \left[G^x_{i+1/2,j} - G^x_{i-1/2,j} \right] + \frac{1}{\Delta y} \left[G^y_{i,j+1/2} - G^y_{i,j-1/2} \right] + R_{i,j}, \end{aligned}$$

as desired.

2 Computational Mesh and Discretization

Since ϕ is stored/computed at the cell centers, it is logical to store the locations of the cell centers. If there are n cell centers in the x direction and m cell centers in the y direction, then the step sizes may be coded by,

$$\text{dx} = \text{Lx/n}; \;\; \text{dy} = \text{Ly/m},$$

where 'Lx' and 'Ly' are the domain lengths. The cell center locations are then generated by,

$$\text{x} = \text{linspace}(\text{dx}/2, \text{Lx} - \text{dx}/2, \text{n}); \;\; \text{y} = \text{linspace}(\text{dy}/2, \text{Ly} - \text{dy}/2, \text{m}),$$

which may be turned into the computational mesh by,

$$[\text{X}, \text{Y}] = \text{meshgrid}(\text{x}, \text{y}).$$

This stores the locations of all of the cell centers on the (x, y) grid and makes for convenient evaluation of $u(x, y)$ and $v(x, y)$ when pumps are introduced. Conceptually, it is also helpful to include 'ghost nodes' which lie outside of the physical domain and are not actually part of the solution. These extra nodes are never computed, but they make thinking about inflow and outflow conditions simpler. For global cell numbering, we take a similar approach to the finite difference code and use the node numbering function, 'NN',

$$NN = n(j-1) + i,$$

where i denotes the x cell center indices and j denotes the y cell center indices. The following figure summarizes the structure of the mesh used in this study.

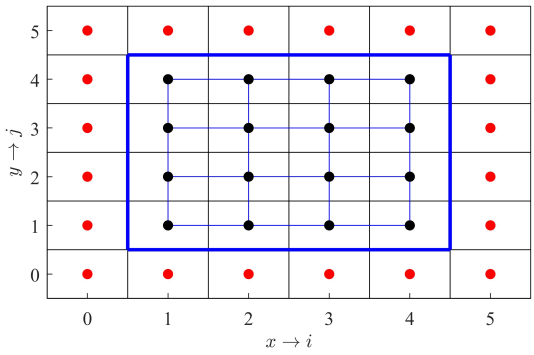


Figure 1: 4×4 Computational mesh with cell center indexing. The bold blue line denotes the physical boundary of the domain, the light blue lines denote the [X,Y] meshgrid, the black dots denote the physical cell centers, and the red dots denote the ghost cells. By the node numbering convention, the $(i, j) = (1, 1)$ node is the first node with $NN = 1$ and the $(i, j) = (4, 4)$ node is the last node with $NN = 16$.

To denote the placement of fluxes on the cell faces, we adopt the half index notation. For example, a flux on the east face of a cell would be denoted with indices $(i + 1/2, j)$ and similarly, a flux on the north face of a cell would be denoted with indices $(i, j + 1/2)$. This is shown visually in the following figure where F^x and F^y denote the fluxes in the x and y directions respectively.

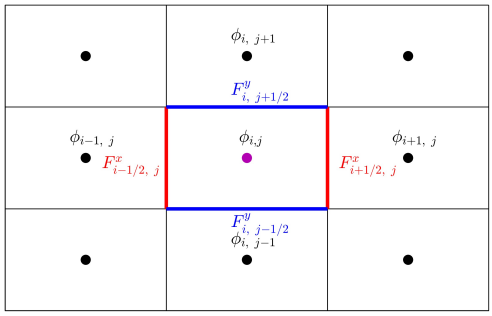


Figure 2: Placement of fluxes relative to cell centers

3 Upwind Flux Scheme

The upwind flux scheme assumes that the quantity being transported takes the value 'upwind' of the flow. For example, for the convective flux through the east face,

$$F^x_{i+1/2,j} = \begin{cases} u_{i+1/2,j}\phi_{i,j} & u_{i+1/2,j} > 0 \\ u_{i+1/2,j}\phi_{i+1,j} & u_{i+1/2,j} \leq 0 \end{cases}.$$

In other words, whichever cell the flow is leaving from, the transported quantity is the ϕ value from that cell. For computational purposes, it is preferable to compute the upwind flux as,

$$F^x_{i+1/2,j} = \frac{1}{2}u_{i+1/2,j}\left(\phi_{i+1,j} + \phi_{i,j}\right) - \frac{1}{2}\left|u_{i+1/2,j}\right|\left(\phi_{i+1,j} - \phi_{i,j}\right).$$

To show the equivalence the the above expressions, consider the case when $u_{i+1/2,j} > 0$, then $\left|u_{i+1/2,j}\right| = u_{i+1/2,j}$ and thus,

$$\begin{aligned} F^x_{i+1/2,j} &= \frac{1}{2}u_{i+1/2,j}\left(\cancel{\phi_{i+1,j}} + \phi_{i,j}\right) - \frac{1}{2}u_{i+1/2,j}\left(\cancel{\phi_{i+1,j}} - \phi_{i,j}\right) \\ &= u_{i+1/2,j}\phi_{i,j}. \end{aligned}$$

Similarly, when $u_{i+1/2,j} \leq 0$, we have $\left|u_{i+1/2,j}\right| = -u_{i+1/2,j}$, giving,

$$\begin{aligned} F^x_{i+1/2,j} &= \frac{1}{2}u_{i+1/2,j}\left(\phi_{i+1,j} + \cancel{\phi_{i,j}}\right) + \frac{1}{2}u_{i+1/2,j}\left(\phi_{i+1,j} - \cancel{\phi_{i,j}}\right) \\ &= u_{i+1/2,j}\phi_{i+1,j}. \end{aligned}$$

The above arguments may be generalized to the fluxes on all faces by appropriately replacing the indices. By summing over the east, west, north, and south faces, we obtain the total convective flux, $F_{i,j}$, from cell (i,j) ,

$$F_{i,j} = \frac{1}{\Delta x}\left[F^x_{i+1/2,j} - F^x_{i-1/2,j}\right] + \frac{1}{\Delta y}\left[F^y_{i,j+1/2} - F^y_{i,j-1/2}\right],$$

which is computed as,

$$\begin{aligned} F_{i,j} = &\frac{1}{2\Delta x}\left[u_{i+1/2,j}\left(\phi_{i+1,j} + \phi_{i,j}\right) - \left|u_{i+1/2,j}\right|\left(\phi_{i+1,j} - \phi_{i,j}\right)\right] \\ &- \frac{1}{2\Delta x}\left[u_{i-1/2,j}\left(\phi_{i,j} + \phi_{i-1,j}\right) - \left|u_{i-1/2,j}\right|\left(\phi_{i,j} - \phi_{i-1,j}\right)\right] \\ &+ \frac{1}{2\Delta y}\left[v_{i,j+1/2}\left(\phi_{i,j+1} + \phi_{i,j}\right) - \left|v_{i,j+1/2}\right|\left(\phi_{i,j+1} - \phi_{i,j}\right)\right] \\ &- \frac{1}{2\Delta y}\left[v_{i,j-1/2}\left(\phi_{i,j} + \phi_{i,j-1}\right) - \left|v_{i,j-1/2}\right|\left(\phi_{i,j} - \phi_{i,j-1}\right)\right]. \end{aligned}$$

In order to generalize the flux function to implicit and explicit time schemes [2], we wish to form a matrix, $\mathcal{F}$, such that $F = \mathcal{F}\phi$, where F and ϕ are vectors of the total flux and concentration values respectively. To this end, we group central coefficients,

$$\begin{aligned} F_{i,j} = &\underbrace{\frac{1}{2\Delta x}\left[u_{i+1/2,j} - \left|u_{i+1/2,j}\right|\right]}_{=E_{i,j}}\phi_{i+1,j} - \underbrace{\frac{1}{2\Delta x}\left[u_{i-1/2,j} + \left|u_{i-1/2,j}\right|\right]}_{=W_{i,j}}\phi_{i-1,j} \\ &+ \underbrace{\frac{1}{2\Delta y}\left[v_{i,j+1/2} - \left|v_{i,j+1/2}\right|\right]}_{=N_{i,j}}\phi_{i,j+1} - \underbrace{\frac{1}{2\Delta y}\left[v_{i,j-1/2} + \left|v_{i,j-1/2}\right|\right]}_{=S_{i,j}}\phi_{i,j-1} \\ &+ \underbrace{\left[\frac{1}{2\Delta x}\left(u_{i+1/2,j} + \left|u_{i+1/2,j}\right| - u_{i-1/2,j} + \left|u_{i-1/2,j}\right|\right)\right.}_{=P_{i,j}} \\ &\underbrace{\left. + \frac{1}{2\Delta y}\left(v_{i,j+1/2} + \left|v_{i,j+1/2}\right| - v_{i,j-1/2} + \left|v_{i,j-1/2}\right|\right)\right]}_{=P_{i,j}}\phi_{i,j}. \end{aligned}$$

The cell face velocities, u and v, can be calculated once over the [X,Y] meshgrid variables shifted by $\Delta x/2$ or $\Delta y/2$ to account for their offset from the cell centers. Once these velocities are calculated, they can be vectorized with the 'reshape' command. We may then construct a 'diag' variable from the vectorized stencil coefficients as,

$$\text{diag} = [\text{S}, \text{W}, \text{P}, \text{E}, \text{N}],$$

and place them on the [-n,-1,0,1,n] diagonals of the $\mathcal{F}$ matrix. This is easily done with spdiags in MATLAB,

$$\text{F} = \text{spdiags}(\text{diag}, [-\text{n}, -1, 0, 1, \text{n}], \text{nodes}, \text{nodes}+1);\ \text{F} = \text{F}(:, 1:\text{nodes});.$$

[2] Implicit schemes are less relevant for the purely convective case, but it is useful to have the matrix formulation when the diffusive flux is added.

In the current formulation, the boundary conditions will be incorrect, but these will be addressed later.

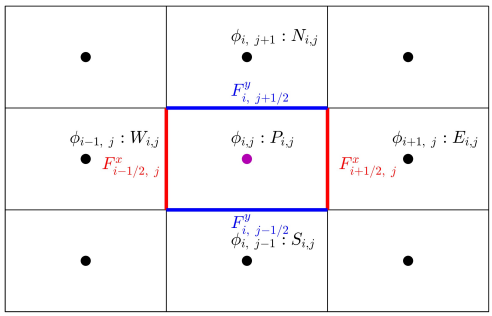


Figure 3: Placement of stencil coefficients with respect to cell (i, j)

A pseudo code overview of the upwind flux scheme is presented in the following code block.

Algorithm 1 Upwind flux scheme

1: Form node numbering function: NN = @(i,j) n*(j-1) + i;
2: Evaluate face velocities for fluxes using meshgrid variables. For example if 'U' is the u velocity function, then the east face velocity is calculated by: ue = U(X+dx/2,Y);
3: Vectorize face velocities, for example: ue = reshape(ue,nodes,1);
4: Ensure the left boundary velocity is the same as the right boundary velocity: uw(NN(1,1:m)) = ue(NN(n,1:m));
5: Evaluate stencil coefficients: S, W, P, E, N. For example: E = 0.5*(ue-abs(ue)/dx;
6: Form the bulk of $\mathcal{F}$ matrix using spdiags:
diag = [S,W,P,E,N];
F = spdiags(diag,[-n,-1,0,1,n],nodes,nodes+1);
F = F(:,1:nodes) + Fx_BC;
7: Apply boundary condition corrections, see code blocks in 'Time Stepping and Boundary Conditions' section

4 Lax Wendroff Flux Scheme

The Lax Wendroff flux scheme provides higher order spatial and temporal resolution than the classic upwind scheme which is only first order accurate [5]. As an example of its implementation, the flux on the east face is calculated as,

$$F^x_{i+1/2,j} = u_{i+1/2,j}\left(\frac{\phi_{i+1,j} + \phi_{i,j}}{2}\right) - \left(u_{i+1/2,j}\right)^2 \frac{\Delta t}{2\Delta x}\left(\phi_{i+1,j} - \phi_{i,j}\right).$$

The above is only a 1D flux function and cannot fully account for the 2D nature of our problem. Thus, to better capture any transport that is not aligned with the coordinate axes, we wish to implement 'dimensional splitting', where we separately update the x and y fluxes. Symbolically, the solution at time step $k+1$ is calculated from the solution at step k by,

$$\phi^*_{i,j} = \phi^k_{i,j} - \frac{\Delta t}{\Delta x}\left[F^x_{i+1/2,j} - F^x_{i-1/2,j}\right]^k,$$

followed by,

$$\phi^{k+1}_{i,j} = \phi^*_{i,j} - \frac{\Delta t}{\Delta y}\left[F^y_{i,j+1/2} - F^y_{i,j-1/2}\right]^*.$$

The superscript '$*$' denotes the intermediate solution after performing the x update and the F^y fluxes are evaluated using the $\phi^*_{i,j}$ values. Again, for practical computation of these fluxes, we wish to form $\mathcal{F}_x$ and $\mathcal{F}_y$ matrices such that the total x flux, F^x, and total y flux, F^y, are given by $F^x = \mathcal{F}_x\phi$ and $F^y = \mathcal{F}_y\phi$ respectively. This is done by summing over the east and west fluxes and north and south fluxes,

$$\begin{aligned} F^x_{i,j} = &\frac{1}{\Delta x}\left[u_{i+1/2,j}\left(\frac{\phi_{i+1,j}+\phi_{i,j}}{2}\right) - \left(u_{i+1/2,j}\right)^2\frac{\Delta t}{2\Delta x}\left(\phi_{i+1,j}-\phi_{i,j}\right)\right] \\ &- \frac{1}{\Delta x}\left[u_{i-1/2,j}\left(\frac{\phi_{i,j}+\phi_{i-1,j}}{2}\right) - \left(u_{i-1/2,j}\right)^2\frac{\Delta t}{2\Delta x}\left(\phi_{i,j}-\phi_{i-1,j}\right)\right], \end{aligned}$$

$$\begin{aligned} F^y_{i,j} = &\frac{1}{\Delta y}\left[v_{i,j+1/2}\left(\frac{\phi_{i,j+1}+\phi_{i,j}}{2}\right) - \left(v_{i,j+1/2}\right)^2\frac{\Delta t}{2\Delta y}\left(\phi_{i,j+1}-\phi_{i,j}\right)\right] \\ &- \frac{1}{\Delta y}\left[v_{i,j-1/2}\left(\frac{\phi_{i,j}+\phi_{i,j-1}}{2}\right) - \left(v_{i,j-1/2}\right)^2\frac{\Delta t}{2\Delta y}\left(\phi_{i,j}-\phi_{i,j-1}\right)\right]. \end{aligned}$$

We may then group the central coefficients,

$$\begin{aligned} F^x_{i,j} = &\underbrace{\frac{1}{2\Delta x}\left[u_{i+1/2,j} - \left(u_{i+1/2,j}\right)^2\frac{\Delta t}{\Delta x}\right]}_{=E_{i,j}}\phi_{i+1,j} - \underbrace{\frac{1}{2\Delta x}\left[u_{i-1/2,j} + \left(u_{i-1/2,j}\right)^2\frac{\Delta t}{\Delta x}\right]}_{=W_{i,j}}\phi_{i-1,j} \\ &\underbrace{+\frac{1}{2\Delta x}\left[u_{i+1/2,j} + \left(u_{i+1/2,j}\right)^2\frac{\Delta t}{\Delta x} - u_{i-1/2,j} + \left(u_{i-1/2,j}\right)^2\frac{\Delta t}{\Delta x}\right]}_{=P^x_{i,j}}\phi_{i,j}. \end{aligned}$$

$$\begin{aligned} F^y_{i,j} = &\underbrace{\frac{1}{2\Delta y}\left[v_{i,j+1/2} - \left(v_{i,j+1/2}\right)^2\frac{\Delta t}{\Delta y}\right]}_{=N_{i,j}}\phi_{i,j+1} - \underbrace{\frac{1}{2\Delta y}\left[v_{i,j-1/2} + \left(v_{i,j-1/2}\right)^2\frac{\Delta t}{\Delta y}\right]}_{=S_{i,j}}\phi_{i,j-1} \\ &\underbrace{+\frac{1}{2\Delta y}\left[v_{i,j+1/2} + \left(v_{i,j+1/2}\right)^2\frac{\Delta t}{\Delta y} - v_{i,j-1/2} + \left(v_{i,j-1/2}\right)^2\frac{\Delta t}{\Delta y}\right]}_{=P^y_{i,j}}\phi_{i,j}. \end{aligned}$$

Similar to the upwind flux, 'diag' variables are formed from the vectorized stencil coefficients as,

$$\text{diag\_x} = [\text{W}, \text{Px}, \text{E}];\ \ \text{diag\_y} = [\text{S}, \text{Py}, \text{N}],$$

along with the appropriate diagonal indices,

$$\text{idx\_x} = [-1, 0, 1];\ \ \text{idx\_y} = [-\text{n}, 0, \text{n}].$$

Forming the flux matrices then uses spdiags; for example, the 'Fx' matrix is formed by,

$$\text{Fx} = \text{spdiags}(\text{diag\_x}, \text{idx\_x}, \text{nodes}, \text{nodes}+1);\ \ \text{Fx} = \text{Fx}(:, 1:\text{nodes});,$$

and which is nearly the same as the construction of the 'Fy' matrix. The pseudo code for Lax Wendroff follows the same structure as the upwind algorithm, with the only difference being the evaluation of the stencil coefficients.

5 Time Stepping and Boundary Conditions

Time Stepping

Explicit marching in time without dimensional splitting may be done with the update equation,

$$\phi^{k+1}_{i,j} = \phi^k_{i,j} - \frac{\Delta t}{\Delta x}\left[F^x_{i+1/2,j} - F^x_{i-1/2,j}\right]^k - \frac{\Delta t}{\Delta y}\left[F^y_{i,j+1/2} - F^y_{i,j-1/2}\right]^k$$

or it terms of the total convective flux, $F^k_{i,j}$,

$$\phi^{k+1}_{i,j} = \phi^k_{i,j} - \Delta t F^k_{i,j}.$$

Recall that we formed the $\mathcal{F}$ matrix such that $F^k = \mathcal{F}\phi^k$. Therefore, instead of considering a single update for $\phi_{i,j}$, consider an update to the entire vector of ϕ values,

$$\begin{aligned}\phi^{k+1} &= \mathbf{I}\phi^k - \Delta t \mathcal{F}\phi^k \\ &= (\mathbf{I} - \Delta t \mathcal{F})\,\phi^k,\end{aligned}$$

where $\mathbf{I}$ is the identity matrix. We define an $\mathbf{A}$ matrix such that,

$$\mathbf{A} = \mathbf{I} - \Delta t \mathcal{F}.$$

Thus, updating the solution simply becomes,

$$\phi^{k+1} = \mathbf{A}\phi^k.$$

All that remains is to modify $\mathbf{A}$ such that the boundary conditions are satisfied at every time step. With dimensional splitting, separate $\mathbf{A}_x$ and $\mathbf{A}_y$ matrices are formed,

$$\mathbf{A}_x = \mathbf{I} - \Delta t \mathcal{F}_x, \quad \mathbf{A}_y = \mathbf{I} - \Delta t \mathcal{F}_y.$$

The splitted update may then be performed,

$$\phi^* = \mathbf{A}_x \phi^k,$$

$$\phi^{k+1} = \mathbf{A}_y \phi^* = \mathbf{A}_y \left(\mathbf{A}_x \phi^k\right) = \left(\mathbf{A}_y \mathbf{A}_x\right) \phi^k.$$

Thus, we may form a total update matrix, $\mathbf{A}$,

$$\mathbf{A} = \mathbf{A}_y \mathbf{A}_x,$$

which gives the familiar equation,

$$\phi^{k+1} = \mathbf{A}\phi^k.$$

Again, we must be mindful of the implementation of boundary conditions. With time stepping accounted for, we now require an initial concentration distribution, for which we will assume a Gaussian centered at $(x, y) = (0.5, 0.5)$,

$$\phi(\mathbf{x}, 0) = \Phi * \exp\left(\frac{-(x-0.5)^2 - (y-0.5)^2}{2\sigma^2}\right),$$

where Φ is the peak amplitude and σ is the standard deviation. For all of the simulations in this report, $\Phi = 0.25$ and $\sigma = 0.1$.

Periodic Boundary Conditions

For periodic boundary conditions, we must impose that the beginning and end (left and right, bottom and top) of the domain obtain the same values. Mathematically, for the top and bottom boundaries,

$$\phi|_{top} = \phi|_{btm}.$$

Numerically, this means that ϕ boundary cells are updated with ϕ values from the opposite boundary cells. This is best explained visually with a matrix sparsity plot.

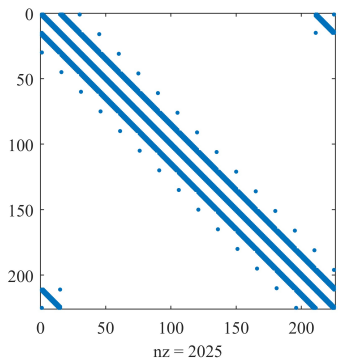


Figure 4: Lax Wendroff flux **A** matrix with dimensional splitting over a 30×30 grid

We see that on the top right and bottom left of the matrix, there are extra diagonal terms that would normally be zeros. For example, on the bottom boundary corresponding to the top rows of the **A** matrix, the $\phi_{i,j-1}$ nodes do not exist. However, the periodicity imposes that $\phi_{i,j-1} = \phi_{i,m}$, where m is the top y index. This generates the extra top right diagonal which accounts for the inclusion of the top nodes in the update calculation of the bottom nodes. This may be generalized to the top, left, and right boundaries using similar arguments. An alternative way to think of this is in terms of matrix-vector products, any non-zero entries in a row means that when **A** is applied to the ϕ vector, those non-zero entries are the nodes that will be used to update the current node. Therefore, the extra top right diagonal denotes the coupling of fluxes between the top and bottom boundaries.

In terms of practical code, we will start with the [S,W,P,E,N] stencil coefficients and modify them to account for the boundaries. To account for left-right periodicity, we first form a $(n * m) \times 2$ stencil matrix, 'diag', of zeros and fill it with the stencil coefficients along the left and right boundaries,

$$\text{diag}(\text{NN}(\text{n}, 1:\text{m})) = \text{E}(\text{NN}(\text{n}, 1:\text{m}));\ \text{diag}(\text{NN}(1, 1:\text{m}), 2) = \text{W}(\text{NN}(1, 1:\text{m}));.$$

We then place these new stencil coefficients on the appropriate diagonals using spdiags,

$$\text{Fx\_BC} = \text{spdiags}(\text{diag}, [-\text{n}+1, \text{n}-1], \text{nodes}, \text{nodes}+1);\ \text{Fx\_BC} = \text{Fx\_BC}(:, 1:\text{nodes});.$$

The $-\text{n}+1$ and $\text{n}-1$ offsets of the diagonals denote the coupling between the left and right nodes. The $\mathcal{F}_x$ matrix is then updated with the new diagonals,

$$\text{Fx} = \text{Fx} + \text{Fx\_BC};.$$

A similar procedure may be performed for the top and bottom boundaries. A summary of the implementation of periodic boundary conditions is presented as follows.

Algorithm 2 Periodic boundary conditions

1: Form stencil for left-right periodic condition:
diag = zeros(nodes,2);
diag(NN(n,1:m),1) = E(NN(n,1:m));
diag(NN(1,1:m),2) = W(NN(1,1:m));
2: Form sparse corrector matrix for left-right periodicity:
Fx_BC = spdiags(diag,[-n+1,n-1],nodes,nodes+1);
Fx_BC = Fx_BC(:,1:nodes);
3: Avoid double counting of boundary nodes:
E(NN(n,1:m)) = 0; W(NN(1,1:m)) = 0;
4: Form sparse corrector matrix for top-bottom periodicity:
diag = [N,S];
Fy_BC = spdiags(diag,[-nodes+n,nodes-n],nodes,nodes+1);
Fy_BC = Fy_BC(:,1:nodes);
5: Form $\mathcal{F}$ matrix for upwind or $\mathcal{F}_x$ and $\mathcal{F}_y$ matrices for Lax Wendroff
6: Apply corrections:
Upwind: F = F + Fx_BC + Fy_BC;
Lax Wendroff: Fx = Fx + Fx_BC; Fy = Fy + Fy_BC;

Inflow and Outflow Boundary Conditions

The inflow boundary condition can be implemented in two main ways. Either the bottom ghost nodes are set to zero, $\phi_{i,0} = 0$, or the value at the face between $\phi_{i,0}$ and $\phi_{i,1}$ is set to zero,

$$\frac{\phi_{i,0} + \phi_{i,1}}{2} = 0.$$

The first scheme is more upwind biased while the second scheme is more accurate since it is essentially doing linear interpolation between the cell centers. In this assignment, the second scheme was chosen since it produced accurate results without introducing oscillations [3]. To implement this, notice the boundary condition can be rewritten as,

$$\phi_{i,0} = -\phi_{i,1}.$$

If one recollects the stencil coefficients for the bottom boundary with the above condition, we find that only the $P^y_{i,j}$ coefficients change and that they can be simply updated by,

$$\mathrm{Py(NN(1:n,1))} = \mathrm{Py(NN(1:n,1))} - \mathrm{S(NN(1:n,1))};\ \mathrm{S(NN(1:n,1))} = 0;,$$

which is applied before formation of the $\mathcal{F}_x$ matrix. For the outflow boundary condition, we essentially wish to impose a Neumann condition,

$$\frac{\partial \phi}{\partial y}\Big|_{top} = 0.$$

If we use a second order finite difference centered about $j = m + 1/2$ (the top boundary faces), we find [4],

$$\frac{\phi_{i,m+1} - \phi_{i,m}}{\Delta y} = 0.$$

We equivalently write the above as,

$$\phi_{i,m+1} = \phi_{i,m},$$

where we can recollect stencil coefficients. Performing this analysis simply reduces to the update equation,

$$\mathrm{Py(NN(1:n,m))} = \mathrm{Py(NN(1:n,m))} + \mathrm{N(NN(1:n,m))};\ \mathrm{N(NN(1:n,m))} = 0;,$$

[3] If given more time, it would be interesting study the effects of these two boundary schemes on the solution.

[4] There is a more rigorous explanation of this condition discussed in Leveque 7.2.1 [5], but it essentially amounts to using finite difference approximations at the boundary.

which is again applied before formation of the $\mathcal{F}_y$ matrix. A summary of these boundary conditions is presented as follows.

Algorithm 3 Inflow/outflow boundary conditions

1: Form stencil for left-right periodic condition:
diag = zeros(nodes,2);
diag(NN(n,1:m),1) = E(NN(n,1:m));
diag(NN(1,1:m),2) = W(NN(1,1:m));
2: Form sparse corrector matrix for left-right periodicity:
Fx_BC = spdiags(diag,[-n+1,n-1],nodes,nodes+1);
Fx_BC = Fx_BC(:,1:nodes);
3: Avoid double counting of boundary nodes:
E(NN(n,1:m)) = 0; W(NN(1,1:m)) = 0;
4: Apply inflow/outflow corrections to P coefficients:
Bottom correction:
P(NN(1:n,1)) = P(NN(1:n,1)) - S(NN(1:n,1));
S(NN(1:n,1)) = 0;
Top correction:
P(NN(1:n,m)) = P(NN(1:n,m)) + N(NN(1:n,m));
N(NN(1:n,m)) = 0;
5: Form $\mathcal{F}$ matrix for upwind or $\mathcal{F}_x$ and $\mathcal{F}_y$ matrices for Lax Wendroff
6: Apply periodicity corrections:
Upwind: F = F + Fx_BC + Fy_BC;
Lax Wendroff: Fx = Fx + Fx_BC; Fy = Fy + Fy_BC;

6 Pseudo Code

To get an overview of the implementation of the finite volume solver, consider the following pseudo code. For a full implementation of the functions developed for the solver, refer to Appendix A. For a full implementation of the time marching and test scripts, refer to Appendix B.

Algorithm 4 Pre-processing for finite volume solver for the convection-diffusion equation

1: Set number of nodes (cell centers) in the computational domain, simulation length, Courant number
n = val; m = val; t_end = val; Co = val;
2: Declare initial condition function:
IC = @(X,Y) 0.25*exp(-50*((X-0.5).^2+(Y-0.5).^2));
3: Declare velocity component functions:
Flow velocity components in absence of sources/sinks, sink strength:
u0 = val; v0 = val; q = val;
Velocity components as functions of the grid points:
U = @(X,Y) u0 - q*(X-2)./((X-2).^2+(Y-2).^2);
V = @(X,Y) v0 - q*(Y-2)./((X-2).^2+(Y-2).^2);
4: Spatial discretization:
nodes = n*m;
dx = Lx/n; dy = Ly/m;
x = linspace(dx/2,Lx-dx/2,n);
y = linspace(dy/2,Ly-dy/2,m);
[X,Y] = meshgrid(x,y);
5: Temporal discretization:
dt = Co/(u0/dx+v0/dy);

Algorithm 5 Finite volume solver for the convection-diffusion equation

Calculate $\mathcal{F}$ matrix for upwind or $\mathcal{F}_x$ and $\mathcal{F}_y$ matrices for Lax Wendroff. See upwind code block.
Form **A** matrix:
  Upwind: A = speye(nodes) - dt*F;
  Lax Wendroff:
  Ax = speye(nodes) - dt*Fx;
  Ay = speye(nodes) - dt*Fy;
  A = Ay*Ax;
Evaluate initial condition at cell centers:
  phi = IC(X,Y);
Calculate total number of time steps to get to end time:
  time_steps = ceil(t_end/dt);
for k=1:time_steps **do**
    phi_tot(k) = sum(phi,'all')*dx*dy; ▷ Get total pollutant mass
    ∼ Perform plotting for animation
    phi = reshape(phi',nodes,1); ▷ Vectorize phi
    phi = A*phi; ▷ Update solution
    phi = reshape(phi,n,m)'; ▷ Convert phi back to meshgrid for plotting
end for
Perform any post-processing, for example, residual plots:
  figure
  r = abs(phi_tot-phi_tot(1));
  semilogy(1:time_steps,r)

7 Showcase of Solutions

As a demonstration of the solution and the qualitative behavior of the two flux schemes, we showcase a gallery of surface and contour plots of the pollution distribution at various time steps. These tests were performed on a $(n, m) = (120, 60)$ grid with $(Lx, Ly) = (2, 1)$, $(u, v) = (2, 2)$ units/s, and $Co = 0.7$ ($\Delta t = 0.0029$ s).

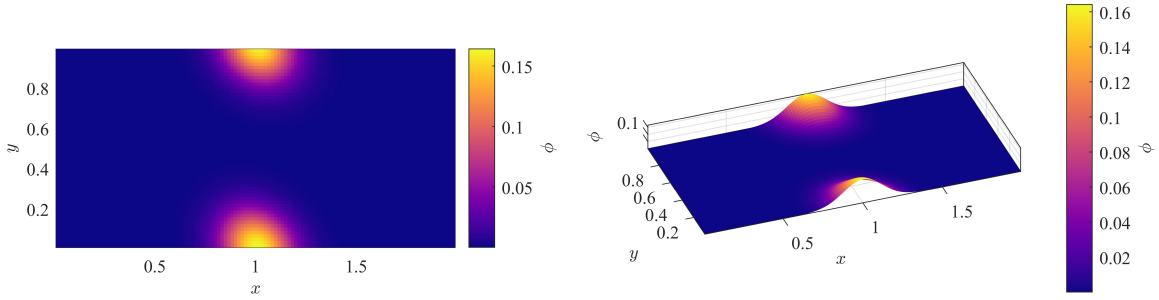


Figure 5: Upwind pollution distribution at $t = 0.5$ s

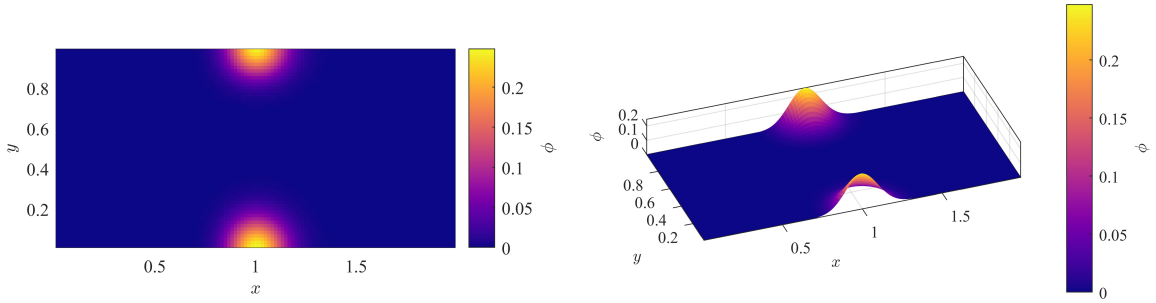


Figure 6: Lax Wendroff pollution distribution at $t = 0.5$ s

From this, we can clearly see that the upwind scheme is diffusive as the peak value of the Gaussian has already decreased to about 0.16. In comparison, the Lax Wendroff scheme maintains the peak at about 0.25, which is close to the original value, indicating a more accurate scheme. From these plots, we can also visually verify that the periodic boundary conditions on the top and bottom are being satisfied.

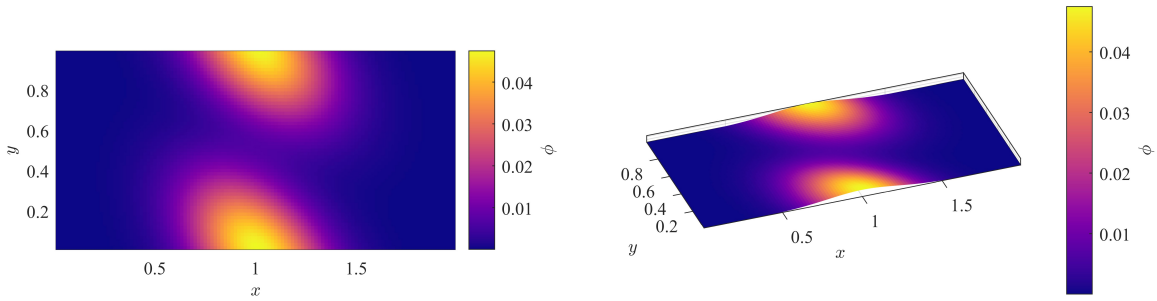


Figure 7: Upwind pollution distribution at $t = 2.25$ s

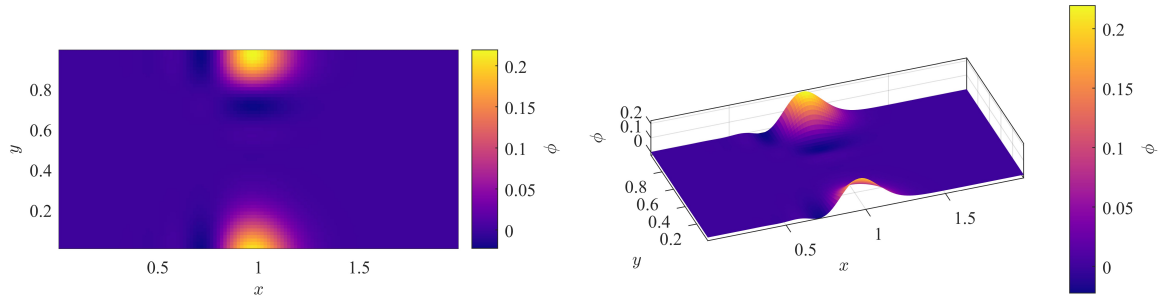


Figure 8: Lax Wendroff pollution distribution at $t = 2.25$ s

At $t = 2.25$ s, the upwind scheme has clearly diffused the pollution even more such that the peak value is only about 0.05, which is clearly non-physical for a purely convective flow. The Lax Wendroff scheme still maintains good accuracy as the peak value is still at about 0.22 and the standard deviation of the Gaussian is still tight. However, it is here we see the oscillations introduced by the scheme as 'dimples' or wave patterns are formed at the base of the Gaussian. This is again clearly non-physical since negative values of concentration are impossible.

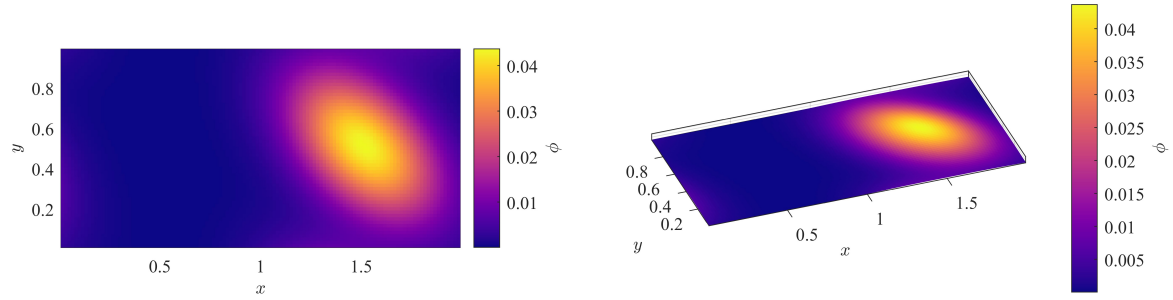


Figure 9: Upwind pollution distribution at $t = 2.5$ s

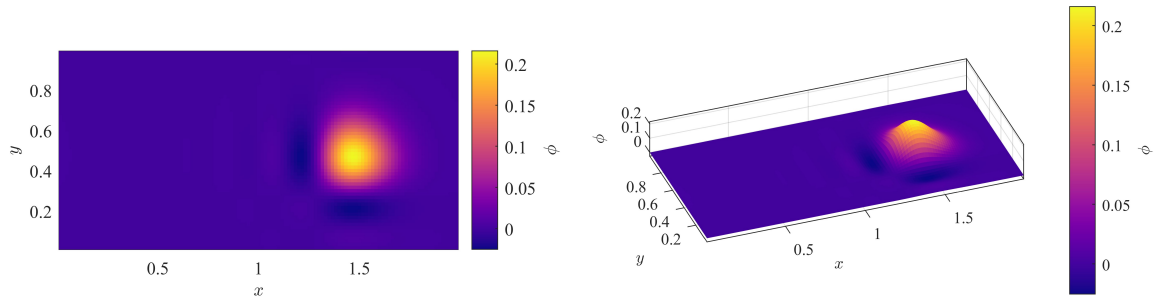


Figure 10: Lax Wendroff pollution distribution at $t = 2.5$ s

At $t = 2.5$ s, we can more clearly see the extent of the diffusion introduced by the upwind scheme as the spread of the Gaussian now spans the entire domain. We can also see that the largest amount of diffusion occurs perpendicular to the flow direction while the diffusion in the flow direction is much less. For the Lax Wendroff scheme, we see than the wave patterns are only formed in the upwind direction and propagate mainly along the coordinate directions. This is likely a consequence of using dimensional splitting rather than using a full 2D flux function.

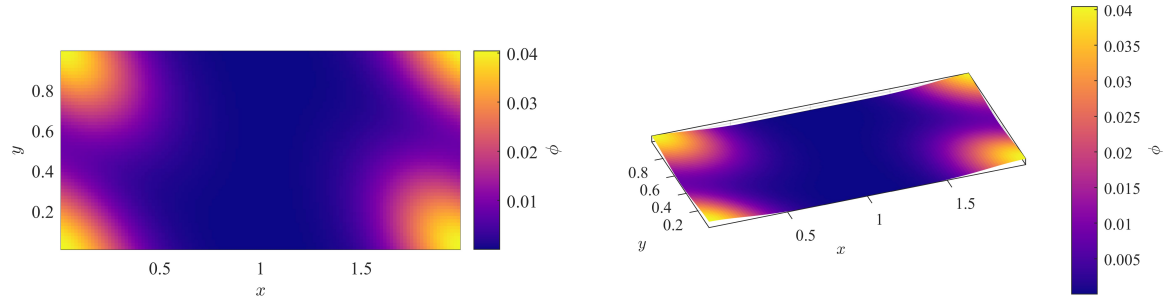


Figure 11: Upwind pollution distribution at $t = 2.75$ s

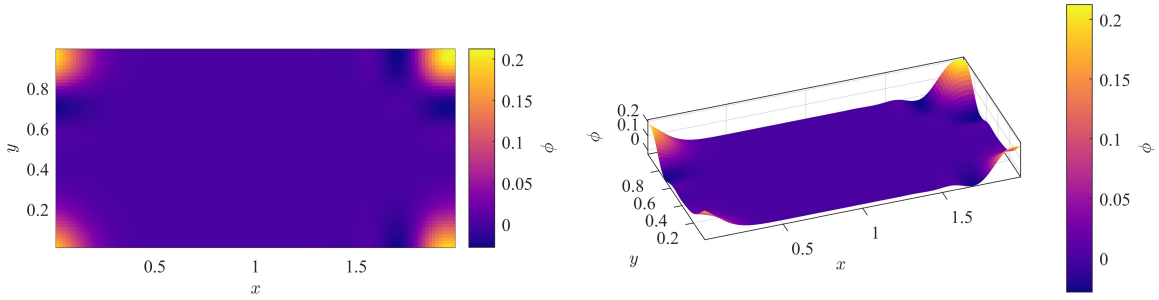


Figure 12: Lax Wendroff pollution distribution at $t = 2.75$ s

The $t = 2.75$ s solutions mainly verify that the periodic boundary conditions are satisfied at the top-bottom and left-right boundary pairs. In conclusion, we notice that neither of the schemes are able to to perfectly convect the Gaussian and introduce either excessive diffusion or dispersion in the computed solution.

8 Comparison of Flux Schemes

Time Step Stability

One method to study the maximum timestep is to simply run a number of simulations at different time steps and see what happens. However, a much more elegant and systematic approach is presented as follows. By the way we defined the $\mathbf{A}$ matrix, the solution vector at time step $k+1$ is given by multiplying the solution at time k by $\mathbf{A}$, i.e.,

$$\phi^{k+1} = \mathbf{A}\phi^k.$$

By recursively applying the above relation, the solution at $k+1$ is connected to the initial condition, ϕ^1, by,

$$\phi^{k+1} = \mathbf{A}^k\phi^1.$$

Assuming $\mathbf{A}$ has an eigenvalue decomposition, $\mathbf{A} = \mathbf{T}\mathbf{\Lambda}\mathbf{T}^{-1}$, we have,

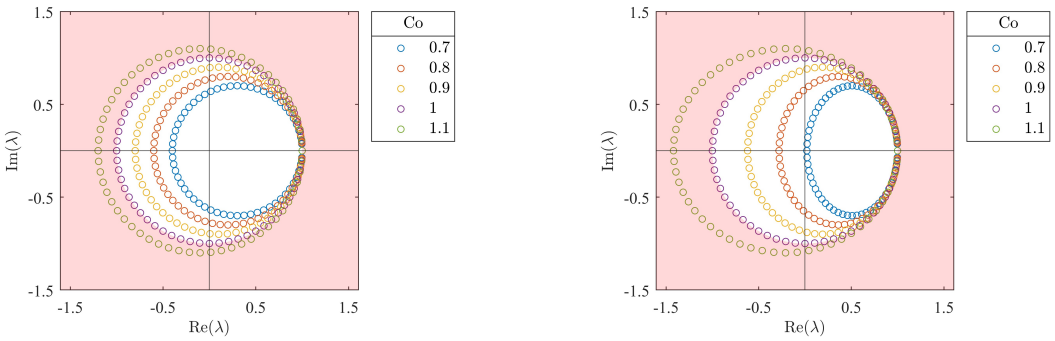
$$\phi^{k+1} = \mathbf{T}\mathbf{\Lambda}^k\mathbf{T}^{-1}\phi^1.$$

Therefore, if a time step is stable, it *must* generate an $\mathbf{A}$ that has eigenvalues with a magnitude of 1 or less, or else successive powers of the eigenvalues (and matrix) will diverge. Thus, to test various time steps, we simply vary the Courant number, Co [5],

$$\Delta t = Co\frac{\Delta x}{u},$$

and plot the eigenvalues of the resulting A matrix. If the eigenvalues lie outside the unit circle, then we know the time step is unstable.

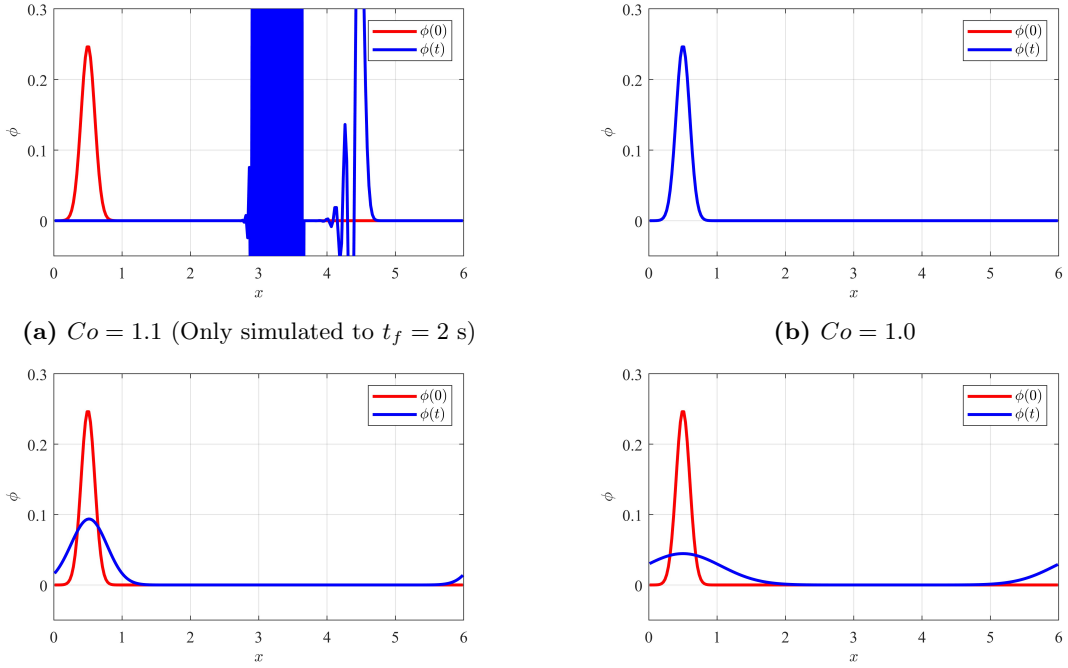
[5] In 2D, we take $\Delta t = Co/(u/\Delta x + v/\Delta y)$.



(a) Upwind **(b)** Lax Wendroff

Figure 13: Distribution of the eigenvalues of the **A** matrix in the complex plane for different flux schemes and Courant numbers. The red region is where $|\lambda| > 1$. For both schemes, the maximum Courant number is 1 before any of the eigenvalues diverge. The Lax Wendroff eigenvalues also lie on ellipses rather than circles, this probably one of the reasons for oscillations.

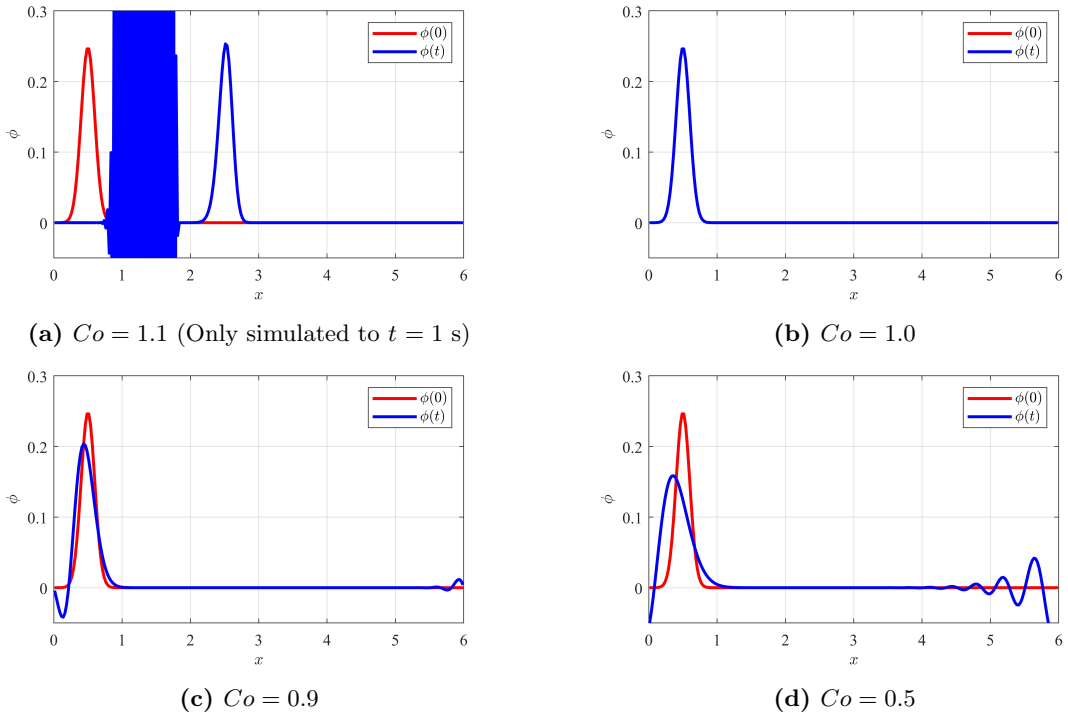
To verify the above analysis, consider the following plots of ϕ along the centerline of the domain, $y = 0.5$. The simulations were run on a $(n, m) = (240, 40)$ grid with $(Lx, Ly) = (6, 1)$ and $(u, v) = (2, 0)$ units/s. They were run until $t_f = 12$ s which are the final solutions shown.



(a) $Co = 1.1$ (Only simulated to $t_f = 2$ s) **(b)** $Co = 1.0$

(c) $Co = 0.9$ **(d)** $Co = 0.5$

Figure 14: Centerline plots of upwind flux scheme for pure convection of a Gaussian



(a) $Co = 1.1$ (Only simulated to $t = 1$ s)

(b) $Co = 1.0$

(c) $Co = 0.9$

(d) $Co = 0.5$

Figure 15: Centerline plots of Lax Wendroff flux scheme for pure convection of a Gaussian

From these plots, we verify that any Courant number greater than 1 is unstable and any Courant number less than 1 introduces undesirable numerical artifacts.

L_2 and L_∞ Mesh Convergence

Let $h = \max(\Delta x, \Delta y)$ and let $\phi^*_{i,j}$ be the numerical solution on a highly refined mesh. By evaluating the solution at a given timestep, we expect that the error of less refined meshes to converge to the refined mesh on the order of $\mathcal{O}(h)$ for the upwind scheme and $\mathcal{O}(h^2)$ for the Lax Wendroff scheme [5]. To this end, we define the approximate error, $e_{i,j}$, as,

$$e_{i,j} = \phi^*_{i,j} - \phi_{i,j}.$$

The L_2 norm error may then be approximated as,

$$||e||_2 \approx \left(\sum_{j=1}^{m} \sum_{i=1}^{n} |e_{i,j}|^2 \, \Delta x \Delta y \right)^{1/2},$$

where $\phi^*_{i,j}$ is evaluated at every k steps to make it the same size as $\phi_{i,j}$. For example, if the most refined mesh is 16×16, then it would be evaluated at every 4 steps to compare it to the solution on a 4×4 grid. In MATLAB syntax, this is related to the Frobenius ('Fro') norm of e [3],

$$||e||_2 = ||e||_{Fro} \sqrt{\Delta x \Delta y},$$

which is coded by,

$$\text{e\_L2} = \text{norm}(\text{phi\_star}(1:\text{k}:\text{end}, 1:\text{k}:\text{end}) - \text{phi}, \text{"Fro"}) * \text{sqrt}(\text{dx} * \text{dy}).$$

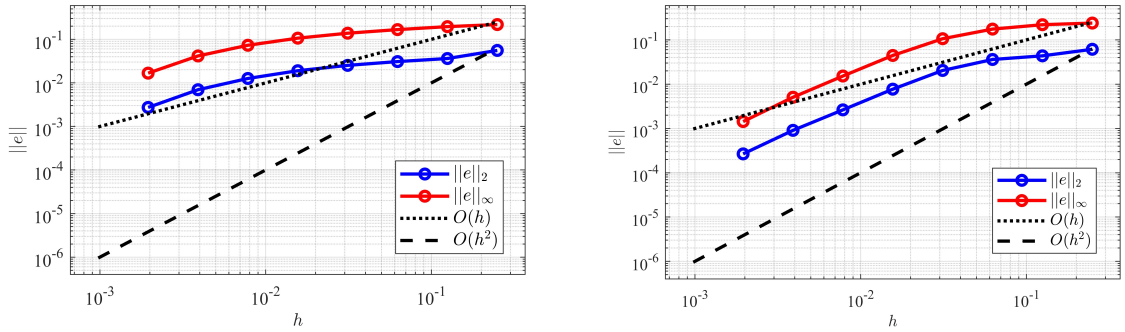
As an absolute error bound, the L_∞ norm may be calculated by,

$$||e||_\infty = \max(|e_{i,j}|)$$

which is coded as,

$$\text{e\_inf} = \max(\text{abs}(\text{phi\_star}(1:\text{k}:\text{end}, 1:\text{k}:\text{end}) - \text{phi}), [\,], \text{"all"}).$$

The convergence test was performed using $Co = 0.9$ and simulated until $t_f = 1$ s over a $(Lx, Ly) = (2, 1)$ grid with $(u, v) = (2, 2)$ units/s. The finest mesh tested was with $(n, m) = (2048, 1024)$ [6] with $h = \Delta x = \Delta y = 9.77\text{e-}4$. Successive meshes had n and m halved each time until the coarsest mesh with $(n, m) = (8, 4)$ was tested. These errors may be plotted on a log-log plot and compared to $\mathcal{O}(h)$ and $\mathcal{O}(h^2)$ which appear as straight lines.



(a) Upwind **(b)** Lax Wendroff

Figure 16: Mesh convergence tests in the L_2 and L_∞ norms

From this, we can verify that the slope of the upwind line matches the slope of the $\mathcal{O}(h)$ line after the first few coarse meshes, indicating first order spatial accuracy, as expected. Similarly, the Lax Wendroff error nearly matches the slope (slightly less) of the $\mathcal{O}(h^2)$ line after the initial coarse meshes. Therefore, we conclude that Lax Wendroff is second order accurate in space.

Numerical Dispersion and Oscillations

We have already seen the diffusion and dissipation introduced by the upwind and Lax Wendroff schemes respectively, however, as a more severe test, we may test the convection of a step function initial condition rather than a smooth Gaussian. To do this, the Gaussian IC was modified such that any values below $0.01 * \Phi$ were set to 0 and any values above $0.01 * \Phi$ were set to Φ. This essentially creates a circular step centered at $(x, y) = (0.5, 0.5)$ with height Φ.

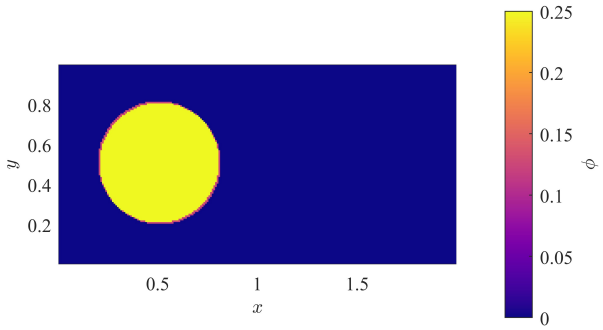


Figure 17: Step initial condition over a $(n, m) = (120, 60)$ mesh with $(Lx, Ly) = (2, 1)$

[6] 2105344×2105344 **A** matrix! Good thing we don't have to solve the system!

The simulations were run on a $(n, m) = (120, 60)$ mesh with $(Lx, Ly) = (2, 1)$ and $(u, v) = (2, 2)$ units/s. Additionally, the simulation was run until $t_f = 2$ s and used a Courant number of 0.7.

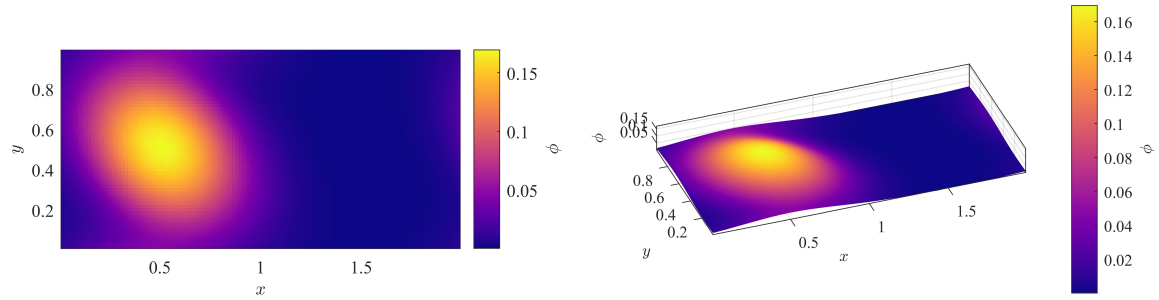


Figure 18: Upwind pollution distribution at $t = 2$ s (step IC)

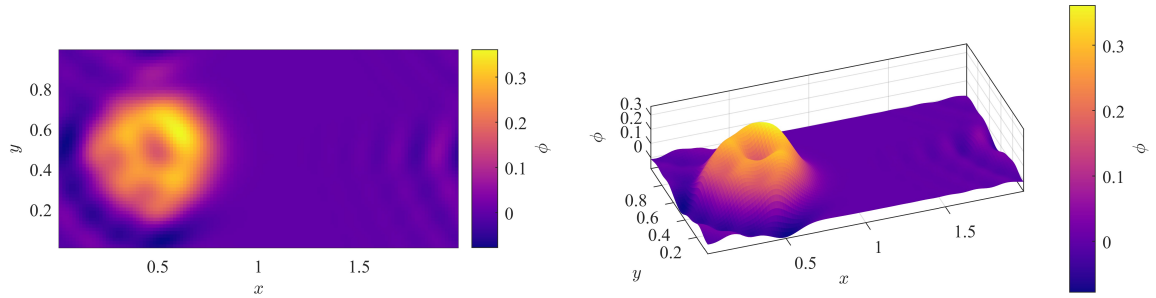


Figure 19: Lax Wendroff pollution distribution at $t = 2$ s (step IC)

The upwind scheme completely smooths out the step function and essentially reforms a Gaussian; from our previous studies, we would expect this smoothing to occur. Perhaps a more interesting result is with the Lax Wendroff scheme where the peak amplitude of the solution *increases* to about 0.35 rather than diffusing out. This clearly demonstrates the influence of the anti-diffusive term in the Lax Wendroff scheme as the pollutant moves from an area of low concentration to high concentration. To maintain conservation, this increase in peak amplitude must be balanced by a decrease in the pollutant concentration elsewhere. We see this is reflected in the deeper wave patterns which have negative concentrations of up to -0.05.

Discrete Conservation

Let Ω denote the entire computational domain, that is,

$$\Omega = \bigcup_{i,j} V_{i,j}.$$

Due to the periodic boundary conditions, we expect the total mass of pollutant to be constant over Ω for all time, that is,

$$\frac{d}{dt} \int_{\Omega} \phi \, dV = 0,$$

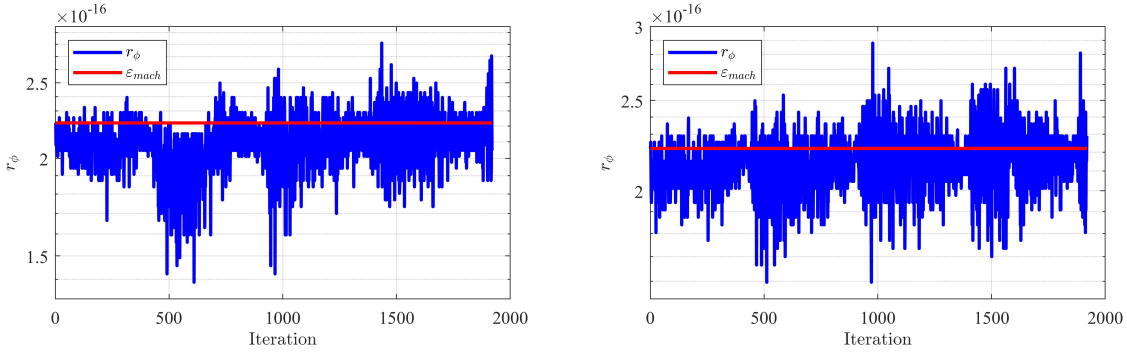
This can be numerically verified by approximating the integral,

$$\phi_{tot} = \int_{\Omega} \phi \, dV \approx \sum_{j=1}^{m} \sum_{i=1}^{n} \phi_{i,j} \Delta x \Delta y,$$

which should remain the same at each time step. For plotting purposes, it is more convenient to define the residual, r_ϕ, defined as,

$$r_\phi \coloneqq \phi_{tot}^k - \phi_{tot}^1,$$

which essentially measures the deviation from the initial pollutant mass. To perform this study, a $(n, m) = (240, 40)$ mesh with $(Lx, Ly) = (6, 1)$ and $(u, v) = (2, 0)$ units/s was used. Additionally, the simulation was run until $t_f = 12$ s and used a Courant number of 0.5.



(a) Upwind **(b)** Lax Wendroff

Figure 20: Residual plots of upwind and Lax Wendroff flux schemes

From this, we observe that the residual is on the order of 10^{-16}, machine epsilon (ε_{mach}), which means the pollutant mass is conserved up to numerical round off. We have thus verified that the finite volume scheme is conservative.

9 Numerical Study of Pollutant Removal

To model pumps, we will superimpose a 'sink' field derived from potential flow. This sink will be placed at $(x, y) = (2, 2)$, outside of the domain, which modifies the velocity components by,

$$u = 2 - q\frac{x-2}{(x-2)^2 + (y-2)^2} \quad , \quad v = -q\frac{y-2}{(x-2)^2 + (y-2)^2},$$

where q is the sink strength. To make sure that the pollutant is removed, we impose the previously discussed inflow and outflow boundary conditions on the bottom and top respectively. To perform this study, a $(n, m) = (240, 40)$ mesh with $(Lx, Ly) = (6, 1)$ was used. Additionally, the simulation was run until $t_f = 20$ with a Courant number of 0.7.

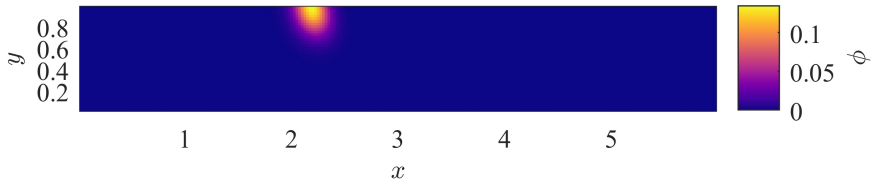
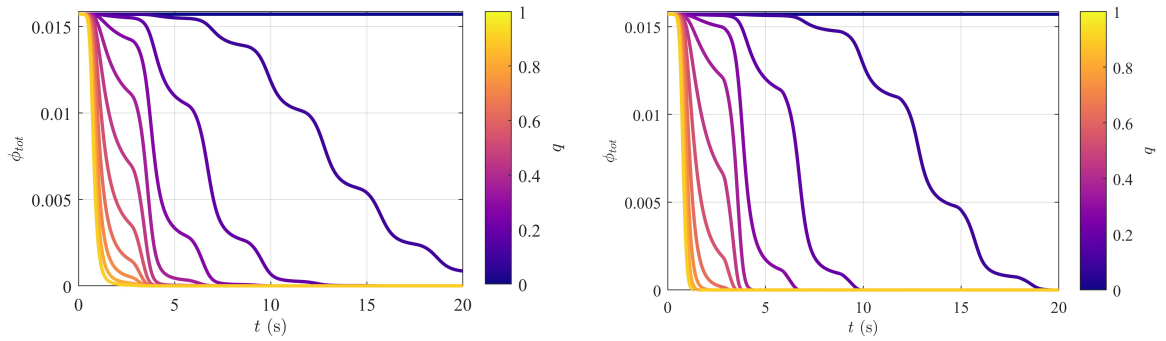


Figure 21: Upwind pollution distribution after $t = 0.5$ s with $q = 1$

From the above, we can verify that the outflow boundary condition is implemented correctly as the flow is undisturbed by the presence of the boundary.



(a) Upwind **(b)** Lax Wendroff

Figure 22: Total pollutant mass plots over time for upwind and Lax Wendroff flux schemes

We see that both flux schemes agree when $q = 0$ and for higher q values close to 1. For $q = 0$, both schemes are essentially conservative and a negligible amount of ϕ exits the domain through the top. For q close to 1, the sink is so strong that it almost immediately sucks all the pollutant in, making the difference between the flux schemes negligible. For intermediate values of q, we observe that the pollution decreases in almost a step-like nature. This is due to the long length of the domain where a lot of pollutant is removed as it passes near the sink at $x = 2$, but much less is removed at $x = 6$. The periodicity in the x direction still ensures that the eventually all of the pollution is removed. We can also see that there is a slight difference in the pollution time distribution between the flux schemes. This difference can be explained by the diffusive nature of the upwind scheme. Since upwind is diffusive, the pollution is much more spread out and thus, the sink has a harder time pulling in all of the pollutant. The Lax Wendroff scheme is more accurate and keeps the spatial pollution distribution tighter. Therefore, the pump is able to pull in the tighter pollutant envelope and remove more pollutant at once, making for a shorter pollutant removal time.

10 Diffusive Flux Scheme

The diffusive fluxes require approximations of derivatives which can be implemented by using finite difference approximations centered at cell faces. For example, for the diffusive flux on the east face [1],

$$G^x_{i+1/2,j} = D\frac{\phi_{i+1,j} - \phi_{i,j}}{\Delta x},$$

and for the north face,

$$G^x_{i,j+1/2} = D\frac{\phi_{i,j+1} - \phi_{i,j}}{\Delta y}.$$

By summing over all of the faces, we may arrive at the total diffusive flux from cell (i, j), $G_{i,j}$,

$$\begin{aligned}
G_{i,j} &= \frac{1}{\Delta x}\left[G^x_{i+1/2,j} + G^x_{i-1/2,j}\right] + \frac{1}{\Delta y}\left[G^x_{i,j+1/2} - G^x_{i,j-1/2}\right] \\
&= \frac{D}{\Delta x^2}\left[\phi_{i+1,j} - \phi_{i,j} - \phi_{i,j} + \phi_{i-1,j}\right] + \frac{D}{\Delta y^2}\left[\phi_{i,j+1} - \phi_{i,j} - \phi_{i,j} + \phi_{i,j-1}\right] \\
&= \underbrace{\left[\frac{D}{\Delta x^2}\right]}_{E_{i,j}}\phi_{i+1,j} + \underbrace{\left[\frac{D}{\Delta x^2}\right]}_{W_{i,j}}\phi_{i-1,j} + \underbrace{\left[\frac{D}{\Delta y^2}\right]}_{N_{i,j}}\phi_{i,j+1} + \underbrace{\left[\frac{D}{\Delta y^2}\right]}_{S_{i,j}}\phi_{i,j-1} \\
&\quad \underbrace{-\left[\frac{2D}{\Delta x^2} + \frac{2D}{\Delta y^2}\right]}_{P_{i,j}}\phi_{i,j}.
\end{aligned}$$

Which is the familiar '[1,-2,1]' pattern from finite differences. Similar to the convective fluxes, we wish to form a matrix, $\mathcal{G}$ such that $G = \mathcal{G}\phi$. The bulk of the $\mathcal{G}$ matrix is formed using spdiags,

$$\text{diag} = [\text{S}, \text{W}, \text{P}, \text{E}, \text{N}];,$$

$$\text{G} = \text{spdiags}(\text{diag}, [-\text{n}, -1, 0, 1, \text{n}], \text{nodes}, \text{nodes}+1);\ \text{G} = \text{G}(:, 1:\text{nodes});.$$

The boundary conditions are then corrected in a similar manner to the convective fluxes. To implement this diffusive flux into the update matrix, $\mathbf{A}$, we use the definition of $\mathcal{G}$,

$$\mathbf{A} = \mathbf{I} - \Delta t\mathcal{F} + \Delta t\mathcal{G}.$$

or with dimensional splitting [7],

$$\mathbf{A}_x = \mathbf{I} - \Delta t\mathcal{F}_x, \quad \mathbf{A}_y = \mathbf{I} - \Delta t\mathcal{F}_y + \Delta t\mathcal{G}, \quad \mathbf{A} = \mathbf{A}_y\mathbf{A}_x.$$

Time Step Stability

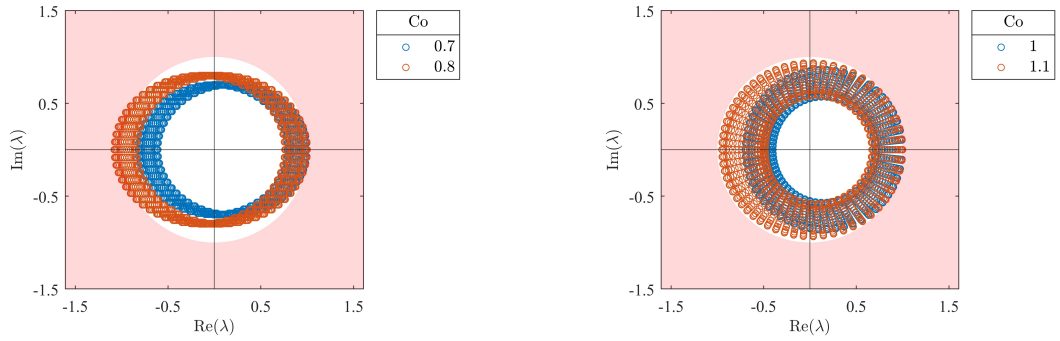
Due to the inclusion of convection and diffusion, it is more difficult to determine a general criteria for time step stability and we will not discuss it further here [8]. However, from the FTCS stability analysis for the pure 1D diffusion equation, we know the time step in that case must satisfy,

$$\Delta t \leq \frac{\Delta x^2}{2D}.$$

The inclusion of the convection term can relax this condition, but we might expect that for a given grid spacing and flow velocity, the time step should be smaller than the pure convection case due to the Δx^2 term. For convenience of implementation in the existing code, the Courant number will still be used to calculate the time step,

$$\Delta t = \frac{Co}{u/\Delta x + v/\Delta y},$$

even though it is not the characteristic dimensionless number of the flow. To get a sense of reasonable time steps, we may again plot the eigenvalues of the $\mathbf{A}$ matrix. However, since diffusion is included, the distribution of eigenvalues is now dependent on $\Delta x^2/(D\Delta t)$ and not Co alone. For the eigenvalue plots, a $(n, m) = (60, 30)$ grid with $(Lx, Ly) = (2, 1)$ and $(u, v) = (2, 0)$ units/s was used. The diffusion constant was also set to $D = 0.005$.



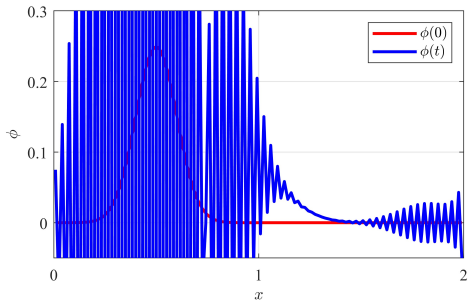
(a) Upwind **(b)** Lax Wendroff

Figure 23: Distribution of the eigenvalues of the $\mathbf{A}$ matrix in the complex plane for different flux schemes with diffusion. The red region is where $|\lambda| > 1$. The max time step for upwind is about $Co = 0.7$ and for Lax Wendroff it is about $Co = 1.1$.

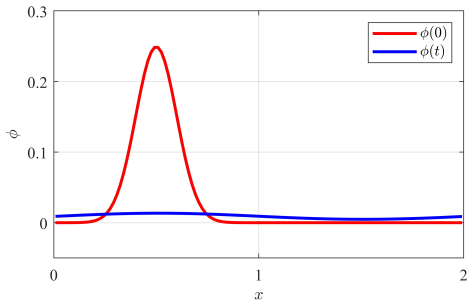
[7] $\mathcal{G}$ can be splitted as well, but diffusion has no preferred direction, so it makes little difference.
[8] For more information, see: Numerical solution of the convection–diffusion equation

From this, we see that the max time step of the upwind scheme *decreases* while the time step *increases* for the Lax Wendroff scheme. One physical motivation for this phenomena is that for the upwind scheme, the diffusive time scale becomes extremely fast with both numerical and physical diffusion. Therefore, the time integrator cannot keep up with the rapid change, making the scheme unstable. In contrast, for the Lax Wendroff scheme, the physical diffusion balances the numerical anti-diffusion and allows for an increase in the time step.

To further study the implementation of the diffusion term, simulations were run on a $(n, m) = (120, 60)$ grid with $(Lx, Ly) = (2, 1)$, $(u, v) = (2, 0)$ units/s and $D = 0.005$.

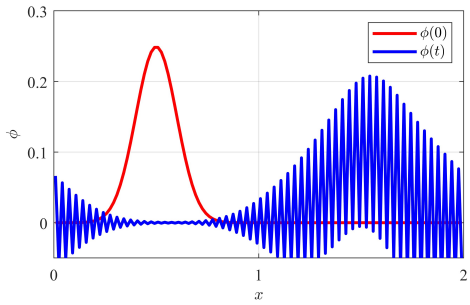


(a) $Co = 0.7$, simulated to $t_f = 1.2$ s

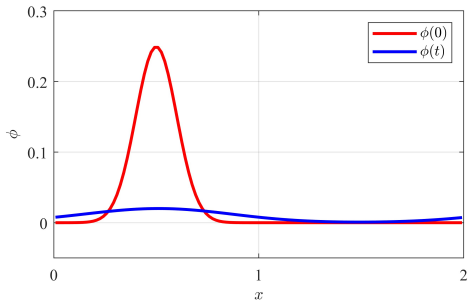


(b) $Co = 0.6$, simulated to $t_f = 12$ s

Figure 24: Centerline plots of upwind flux scheme for convective-diffusive transport of a Gaussian



(a) $Co = 1.3$, simulated to $t_f = 1$ s (diverges)



(b) $Co = 1.2$, simulated to $t_f = 12$ s

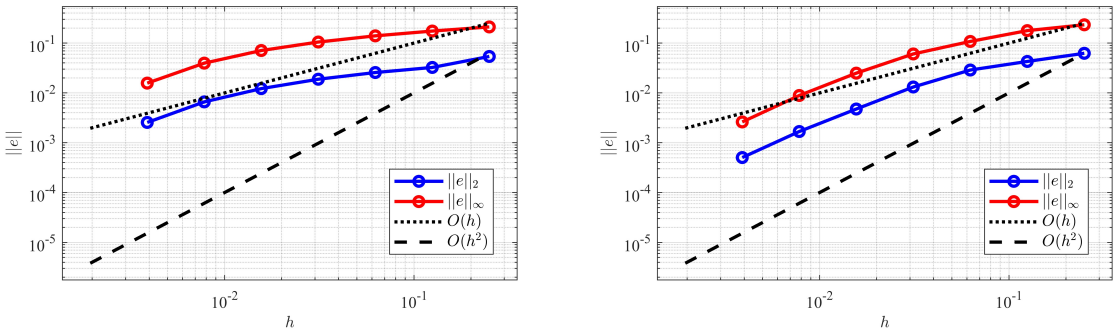
Figure 25: Centerline plots of Lax Wendroff flux scheme for convective-diffusive transport of a Gaussian

From these plots, we verify that the max time step of the upwind scheme decreases to about $Co = 0.6$ while the max step of the Lax Wendroff scheme increases to about $Co = 1.2$. These results are slightly different from the eigenvalue plots since the grid spacing is twice as refined. We can also see from these plots that the pollution becomes very spread out after $t = 12$ s and is nearly flat. This is expected as diffusion tends to spread things out and $D = 0.005$ is moderately strong compared to the convective term.

L_2 and L_∞ Mesh Convergence

The convergence test was performed using $Co = 0.1$ [9] and simulated until $t_f = 0.5$ s over a $(Lx, Ly) = (2, 1)$ grid with $(u, v) = (2, 0)$ units/s and $D = 0.005$. The finest mesh tested was with $(n, m) = (1024, 512)$ with $h = \Delta x = \Delta y = 0.002$. Successive meshes had n and m halved each time until the coarsest mesh with $(n, m) = (8, 4)$ was tested. These errors may be plotted on a log-log plot and compared to $\mathcal{O}(h)$ and $\mathcal{O}(h^2)$ which appear as straight lines.

[9] The time step has to be very small since diffusion forces the upwind scheme to have a small max time step.



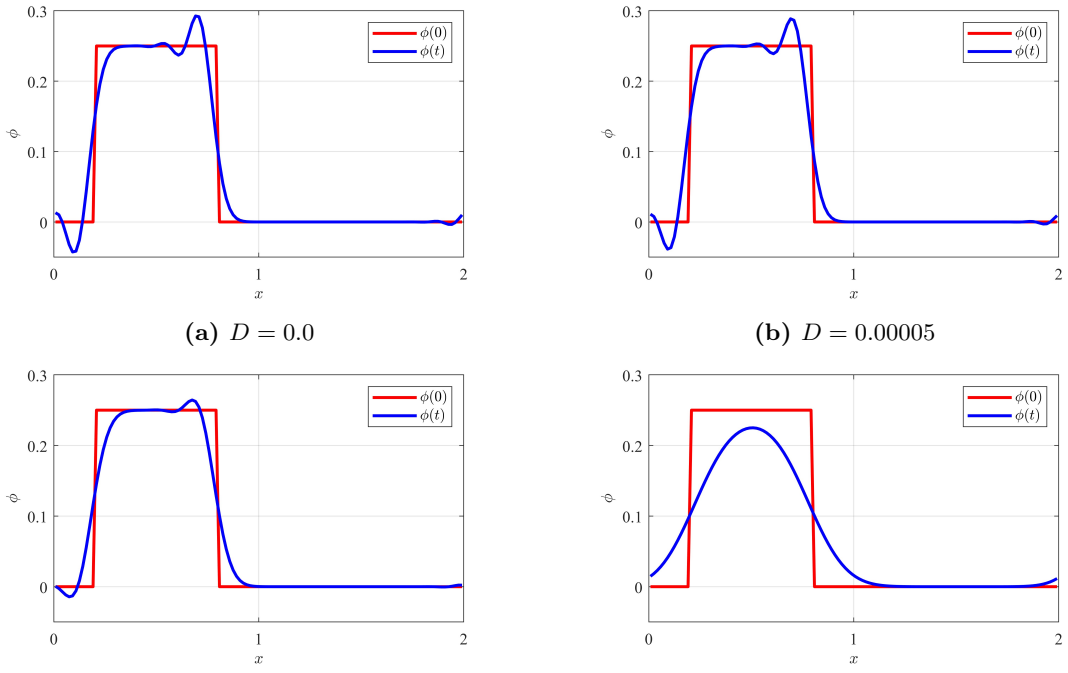
(a) Upwind **(b)** Lax Wendroff

Figure 26: Mesh convergence tests in the L_2 and L_∞ norms

From this, we can verify that the slope of the upwind line matches the slope of the $\mathcal{O}(h)$ line after the first few coarse meshes, indicating first order spatial accuracy. Similarly, the Lax Wendroff error nearly matches the slope (slightly less) of the $\mathcal{O}(h^2)$ line after the initial coarse meshes. Therefore, we verify that Lax Wendroff is second order accurate in space.

Numerical Dispersion and Oscillations

The upwind scheme only gets more diffusive with large diffusion constants, so it will not explored further here. However, we might expect that the numerical dispersion of the Lax Wendroff scheme may be damped out by the physical diffusion and give an overall scheme that is reasonable. This hypothesis may be verified by running a simulation of a step function on a $(n, m) = (120, 60)$ grid with $(Lx, Ly) = (2, 1)$ and $(u, v) = (2, 0)$ units/s. The diffusion constant was varied for each experiment and simulated until $t = 2$ s with a Courant number of 0.9.



(a) $D = 0.0$ **(b)** $D = 0.00005$

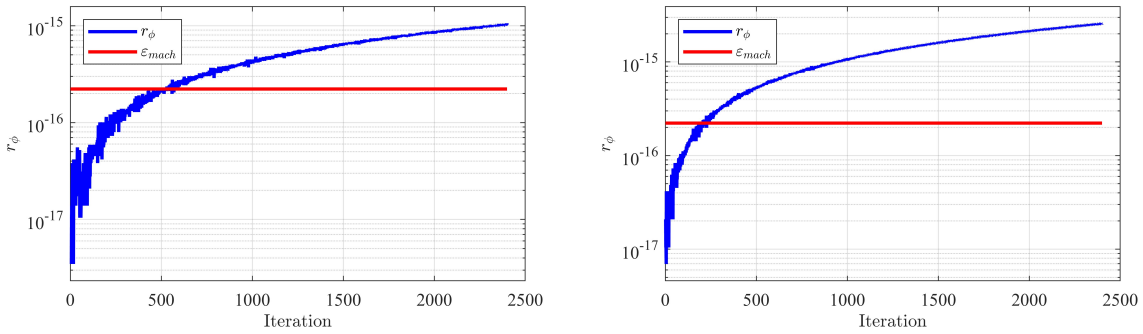
(c) $D = 0.0005$ **(d)** $D = 0.005$

Figure 27: Centerline plots of Lax Wendroff flux scheme for convective-diffusive transport of a Gaussian

We can see that the inclusion of a small diffusion term in the Lax Wendroff flux helps damp oscillations and makes for more realistic looking solution to the pure convection problem when $D = 0.0005$. This technique of introducing diffusion to a hyperbolic problem is inspired by the 'vanishing viscosity' approach to shock solutions for PDE's [6].

Discrete Conservation

To perform this study, a $(n, m) = (120, 60)$ mesh with $(Lx, Ly) = (6, 1)$ and $(u, v) = (2, 0)$ units/s was used. Additionally, the simulation was run until $t_f = 12$ s and used a Courant number of 0.6 and diffusion constant of 0.005.



(a) Upwind **(b)** Lax Wendroff

Figure 28: Residual plots of upwind and Lax Wendroff flux schemes

From this, we observe that the residual is on the order of machine epsilon, ε_{mach}. However, it seems the scheme is slightly non-conservative since the residual seems to grow with the iteration count. This error is still small, but it may become significant for very long time simulations.

References

[1] Saad, T., *The Finite Volume Method.* University of Utah, 2019. https://www.youtube.com/watch?v=4n3DPwcoy4E&ab_channel=ProfessorSaadExplains

[2] https://www.mathworks.com/help/matlab/ref/spdiags.html

[3] https://www.mathworks.com/help/matlab/ref/norm.html

[4] Haberman, R., *Applied Partial Differential Equations with Fourier Series and Boundary Value Problems,* 3rd Ed., Prentice Hall, 1997.

[5] Leveque, Randall, *Finite Volume Methods for Hyperbolic Problems*, Cambridge Texts in Applied Mathematics, 2002.

[6] Evans L.C., *Partial Differential Equations*, 2nd Ed., American Mathematical Society, 2010.

Appendix A - Functions

```
% ---------------------------------------------------------------------
% Patrick Heng
% 08 Mar. 2025
% Upwind flux scheme for the convection term in the convection-
% diffusion equations over a uniformly spaced rectangular grid.
% ---------------------------------------------------------------------

function F = upwind_flux(X,Y,U,V,dx,dy,n,m,nodes,periodic)
    % Node numbering function
    NN = @(i,j) n*(j-1) + i;

    % Evaluate face velocities for fluxes
    uw = U(X-dx/2,Y); ue = U(X+dx/2,Y);
    vs = V(X,Y-dy/2); vn = V(X,Y+dy/2);

    % Vectorize velocities
    uw = reshape(uw',nodes,1); ue = reshape(ue',nodes,1);
    vs = reshape(vs',nodes,1); vn = reshape(vn',nodes,1);

    % Ensure the left boundary velocity is the same as the right
    % boundary velocity
    uw(NN(1,1:m)) = ue(NN(n,1:m));

    % Stencil coefficients
    W = -0.5*(uw+abs(uw))/dx;
    E = 0.5*(ue-abs(ue))/dx;
    S = -0.5*(vs+abs(vs))/dy;
    N = 0.5*(vn-abs(vn))/dy;
    P = 0.5*(ue+abs(ue)-uw+abs(uw))/dx + 0.5*(vn+abs(vn)-vs+abs(vs))/dy;

    % Left, right periodic condition
    diag = zeros(nodes,2);
    diag(NN(n,1:m),1) = E(NN(n,1:m));
    diag(NN(1,1:m),2) = W(NN(1,1:m));

    % Sparse corrector matrix
    Fx_BC = spdiags(diag,[-n+1,n-1],nodes,nodes+1);
    Fx_BC = Fx_BC(:,1:nodes);

    % Avoid double counting
    E(NN(n,1:m)) = 0;
    W(NN(1,1:m)) = 0;

    if periodic == false
        % Bottom correction
        P(NN(1:n,1)) = P(NN(1:n,1)) - S(NN(1:n,1));
        S(NN(1:n,1)) = 0;
        % Top correction
        P(NN(1:n,m)) = P(NN(1:n,m)) + N(NN(1:n,m));
        N(NN(1:n,m)) = 0;
        % Form flux matrix
        diag = [S,W,P,E,N];
```

```
        F = spdiags(diag,[-n,-1,0,1,n],nodes,nodes+1);
        F = F(:,1:nodes) + Fx_BC;
    else
        % Form flux matrix
        diag = [S,W,P,E,N];
        F = spdiags(diag,[-n,-1,0,1,n],nodes,nodes+1);
        F = F(:,1:nodes);
        % Sparse corrector matrix
        diag = [N,S];
        F_BC = spdiags(diag,[-nodes+n,nodes-n],nodes,nodes+1);
        F_BC = F_BC(:,1:nodes);
        F = F + F_BC + Fx_BC;
    end

end

% ---------------------------------------------------------------------
% Patrick Heng
% 08 Mar. 2025
% Lax Wendroff flux scheme for the convection term in the convection-
% diffusion equations over a uniformly spaced rectangular grid.
% ---------------------------------------------------------------------

function [Fx,Fy] = Lax_Wendroff_flux(X,Y,U,V,dx,dy,dt,n,m,nodes,periodic)
    % Node numbering function
    NN = @(i,j) n*(j-1) + i;

    % Evaluate face velocities for fluxes
    uw = U(X-dx/2,Y); ue = U(X+dx/2,Y);
    vs = V(X,Y-dy/2); vn = V(X,Y+dy/2);

    % Vectorize velocities
    uw = reshape(uw',nodes,1); ue = reshape(ue',nodes,1);
    vs = reshape(vs',nodes,1); vn = reshape(vn',nodes,1);

    % Ensure the left boundary velocity is the same as the right
    % boundary velocity
    uw(NN(1,1:m)) = ue(NN(n,1:m));

    % Stencil coefficients
    W = -0.5*(uw+(uw.^2)*dt/dx)/dx;
    E = 0.5*(ue-(ue.^2)*dt/dx)/dx;
    S = -0.5*(vs+(vs.^2)*dt/dy)/dy;
    N = 0.5*(vn-(vn.^2)*dt/dy)/dy;
    P = 0.5*(ue+(ue.^2)*dt/dx-uw+(uw.^2)*dt/dx)/dx;

    % Left, right periodic condition
    diag = zeros(nodes,2);
    diag(NN(n,1:m),1) = E(NN(n,1:m));
    diag(NN(1,1:m),2) = W(NN(1,1:m));

    % Sparse corrector matrix
    Fx_BC = spdiags(diag,[-n+1,n-1],nodes,nodes+1);
    Fx_BC = Fx_BC(:,1:nodes);
```

```
    % Avoid double counting
    E(NN(n,1:m)) = 0;
    W(NN(1,1:m)) = 0;
    diag = [W,P,E];

    % Form Fx matrix with corrections
    Fx = spdiags(diag,[-1,0,1],nodes,nodes+1);
    Fx = Fx(:,1:nodes);
    Fx = Fx + Fx_BC;

    % Py coefficients
    P = 0.5*(vn+(vn.^2)*dt/dy-vs+(vs.^2)*dt/dy)/dy;

    if periodic == false
        % Bottom correction
        P(NN(1:n,1)) = P(NN(1:n,1)) - S(NN(1:n,1));
        S(NN(1:n,1)) = 0;
        % Top correction
        P(NN(1:n,m)) = P(NN(1:n,m)) + N(NN(1:n,m));
        N(NN(1:n,m)) = 0;
    end

    % Sparse correction matrix
    diag = [S,P,N];
    Fy = spdiags(diag,[-n,0,n],nodes,nodes+1);
    Fy = Fy(:,1:nodes);

    if periodic == true
        % Top, bottom periodic condition
        diag = [N,S];
        Fy_BC = spdiags(diag,[-nodes+n,nodes-n],nodes,nodes+1);
        Fy_BC = Fy_BC(:,1:nodes);
        Fy = Fy + Fy_BC;
    end

end

% ---------------------------------------------------------------------
% Patrick Heng
% 22 Mar. 2025
% Diffusive flux scheme for the convection term in the convection-
% diffusion equations over a uniformly spaced rectangular grid.
% ---------------------------------------------------------------------

function G = diffusive_flux(D,dx,dy,n,m,nodes)
    % Node numbering function
    NN = @(i,j) n*(j-1) + i;

    % Fill diffusive flux stencil
    diag = [1/dy^2,1/dx^2,-2/dx^2-2/dy^2,1/dx^2,1/dy^2];
    diag = repmat(diag,nodes,1);
    diag(1:n:nodes,2) = 0;
    diag(n:n:nodes-1,4) = 0;
```

```
    % Place stencil coefficients on diagonals
    G = spdiags(diag,[-n,-1,0,1,n],nodes,nodes+1);
    G = G(:,1:nodes);

    % Boundary Corrections for periodic boundary conditions
    top = [NN(1:n,1)',NN(1:n,m)',ones(n,1)/dy^2];
    btm = [NN(1:n,m)',NN(1:n,1)',ones(n,1)/dy^2];
    left = [NN(1,1:m)',NN(n,1:m)',ones(m,1)/dx^2];
    right = [NN(n,1:m)',NN(1,1:m)',ones(m,1)/dx^2];

    % Form sparse diagonal matrices for corrections
    top = sparse(top(:,1),top(:,2),top(:,3),nodes,nodes);
    btm = sparse(btm(:,1),btm(:,2),btm(:,3),nodes,nodes);
    left = sparse(left(:,1),left(:,2),left(:,3),nodes,nodes);
    right = sparse(right(:,1),right(:,2),right(:,3),nodes,nodes);

    % Apply corrections
    G = D*(G + top + btm + left + right);

end
```

Appendix B - Test Scripts

```
% ------------------------------------------------------------------
% 'HW_2_test.m'
% Patrick Heng
% 29 Mar. 2025
% Test script to run the finite volume solver for the convection-
% diffusion equations.
% ------------------------------------------------------------------

close all; clear variables; clc;

% Colormaps from '200 Colormaps' on the MATLAB file exchange
% SWITCH THIS TO A DEFAULT COLORMAP ('parula') IF YOU DO NOT HAVE THE
% FILE INSTALLED
cmap=slanCM(2);

% ------------------------------------------------------------------
% ----- INPUTS -----
upwind = true;
diffusion = false;
step_IC = false;
periodic = true;

% Number of nodes (cell centers) in the computational domain,
% n -> x, m -> y
n = 120;
m = 60;

% Simulation time
```

```
t_end = 12;

% Courant number, used to determine time step size
Co = 1;

% Initial concentration distribution
IC = @(X,Y) 0.25*exp(-50*((X-0.5).^2+(Y-0.5).^2));

% Diffusion constant
D = 0.005;

% Sink strength, 0 = off
q = 0;

% Flow velocity components in absence of sources/sinks
u0 = 2;
v0 = 0;

% Velocity components as functions of the grid points
U = @(X,Y) u0 - q*(X-2)./((X-2).^2+(Y-2).^2);
V = @(X,Y) v0 - q*(Y-2)./((X-2).^2+(Y-2).^2);

% Length of domain
Lx = 2;
Ly = 1;

% ---------------------------------------------------------------------
% ----- SPATIAL DISCRETIZATION -----
% Total number of cell centers
nodes = n*m;

% Spatial discretization steps
dx = Lx/n;
dy = Ly/m;

% Create grid of cell centers
x = linspace(dx/2,Lx-dx/2,n);
y = linspace(dy/2,Ly-dy/2,m);
[X,Y] = meshgrid(x,y);

% Calculate step size based on Courant number
%dt = Co*min(dx,dy)/max(abs(u0),abs(v0));
dt = Co/(u0/dx+v0/dy);

% Determine flux function
if upwind == true
    F = upwind_flux(X,Y,U,V,dx,dy,n,m,nodes,periodic);
else
    [Fx,Fy] = Lax_Wendroff_flux(X,Y,U,V,dx,dy,dt,n,m,nodes,periodic);
end

% Turn on/off diffusion
if diffusion == true
    G = diffusive_flux(D,dx,dy,n,m,nodes);
```

```
    if upwind == true
        A = speye(nodes) - dt*F + dt*G;
    else
        Ax = speye(nodes) - dt*Fx;
        Ay = speye(nodes) - dt*Fy + dt*G;
        A = Ay*Ax;
    end
else
    if upwind == true
        A = speye(nodes) - dt*F;
    else
        Ax = speye(nodes) - dt*Fx;
        Ay = speye(nodes) - dt*Fy;
        A = Ay*Ax;
    end
end

% Evaluate initial condition at cell centers
phi = IC(X,Y);

% If true, any values below 1% of the peak amplitude are set to 0 and
% andy values above are set to the peak amplitude
if step_IC == true
    phi(phi<0.0025) = 0;
    phi(phi>=0.0025) = 0.25;
end

phi0 = phi;

% Determine the amount of time steps to get to tf
time_steps = ceil(t_end/dt);

% Store total pollutant mass, peak value of concentration
phi_tot = zeros(time_steps,1);
phi_max = zeros(time_steps,1);

% ---------------------------------------------------------------------
% ----- TIME INTEGRATION -----
% Loop through time in an explicit marching scheme
for k = 1:time_steps

    % Store current pollutant mass, peak amplitude
    phi_tot(k) = sum(phi,'all')*dx*dy;
    phi_max(k) = max(phi,[],'all');

    % For any of plots, remove '{'  from the '%{' below to use

    %
    colormap(cmap);
    surf(X,Y,phi,linestyle='none')
    axis equal
    view(0,90)
    cb = colorbar;
    ylabel(cb,'$\phi$',interpreter='latex')
```

```
    xlabel('$x$',interpreter='latex')
    ylabel('$y$',interpreter='latex')
    fontsize(16,'points'); fontname('Serif')
    box on
    grid on
    pause(0.01)
    %}

    %{
    colormap(cmap);
    surf(X,Y,phi,linestyle='none')
    xlabel('$x$',interpreter='latex')
    ylabel('$y$',interpreter='latex')
    zlabel('$\phi$',interpreter='latex')
    axis equal
    colorbar
    cb = colorbar;
    ylabel(cb,'$\phi$',interpreter='latex')
    fontsize(16,'points'); fontname('Serif')
    box on
    grid on
    pause(0.01)
    %}

    %}

    % Toggle for centerline plot animation
    %{
    plot(x,phi(round(m/2),:),linewidth=2,marker='none',color='b')
    ylim([-0.05,0.3])
    xticks(0:6); yticks(0:0.1:0.3)
    xlabel('$x$',interpreter='latex')
    ylabel('$\phi$',interpreter='latex')
    fontsize(16,'points'); fontname('Serif')
    pause(0.01)
    %}

    % Vectorize phi
    phi = reshape(phi',nodes,1);
    % Update solution
    phi = A*phi;
    % Convert phi to meshgrid for plotting
    phi = reshape(phi,n,m)';

end

% ---------------------------------------------------------------------
% ----- POST-PROCESSING -----

% Final contour plot of solution
%
figure
colormap(cmap);
surf(X,Y,phi,linestyle='none')
```

```
axis equal
view(0,90)
cb = colorbar;
ylabel(cb,'$\phi$',interpreter='latex')
xlabel('$x$',interpreter='latex')
ylabel('$y$',interpreter='latex')
fontsize(16,'points'); fontname('Serif')
box on
grid on
%}

% Final surface plot of solution
%
figure
colormap(cmap);
surf(X,Y,phi,linestyle='none')
xlabel('$x$',interpreter='latex')
ylabel('$y$',interpreter='latex')
zlabel('$\phi$',interpreter='latex')
axis equal
view(-20,33)
colorbar
cb = colorbar;
ylabel(cb,'$\phi$',interpreter='latex')
fontsize(16,'points'); fontname('Serif')
box on
grid on
%}


% Plot centerline solution at final time against initial condition to
% compare numerical diffusion/dispersion
%
figure
hold on
plot(x,phi0(round(m/2),:),linewidth=2,marker='none',color='r')
plot(x,phi(round(m/2),:),linewidth=2,marker='none',color='b')
ylim([-0.05,0.3])
xticks(0:6); yticks(0:0.1:0.3)
xlabel('$x$',interpreter='latex')
ylabel('$\phi$',interpreter='latex')
fontsize(16,'points'); fontname('Serif')
legend('$\phi(0)$','$\phi(t)$',interpreter='latex',location='northeast')
box on
grid on
%}

% Conservation plot of total mass of pollutant at each iteration compared
% to starting mass of pollutant
%
figure

r = abs(phi_tot-phi_tot(1));
semilogy(1:time_steps,r,linewidth=2,marker='none',color='b')
```

```
hold on
semilogy([1,time_steps],[0,0]+eps,linewidth=2,marker='none',color='r')
xlabel('Iteration',interpreter='latex')
ylabel('$r_{\phi}$',interpreter='latex')
fontsize(16,'points'); fontname('Serif')
legend('$r_{\phi}$','$\varepsilon_{mach}$',interpreter='latex', ...
        location='northwest')
box on
grid on
ylim padded
%}

% Plot peak value of phi as a measure of the diffusion of the flux scheme
%
figure
plot(1:time_steps,phi_max,linewidth=2,marker='none',color='b')
xlabel('Iteration',interpreter='latex')
ylabel('$\phi_{max}$',interpreter='latex')
fontsize(16,'points'); fontname('Serif')
box on
grid on
%}

% ------------------------------------------------------------------------
% 'HW_2_A_Eigenvalues.m'
% Patrick Heng
% 08 Mar. 2025
% Test script to study the stability of explicit time schemes for
% upwind and Lax Wendroff flux schemes.
% ------------------------------------------------------------------------

clear all; close all; clc;

% ------------------------------------------------------------------------
% ----- INPUTS -----
upwind = true;
diffusion = false;
step_IC = false;


% Number of nodes in the computational domain, n -> x, m -> y
n = 60;
m = 30;

% Simulation time
t_end = 1;

% Courant numbers, used to determine time step size
tests = 0.7:0.1:1.1;

% Diffusion constant
D = 0.005;

% Domain lengths
```

```
Lx = 2;
Ly = 1;

% Sink strength, 0 = off
q = 0;

% Flow velocity components in absence of sources/sinks
u0 = 2;
v0 = 0;

% Velocity components as functions of the grid points
U = @(X,Y) u0 - q*(X-2)./((X-2).^2+(Y-2).^2);
V = @(X,Y) v0 - q*(Y-2)./((X-2).^2+(Y-2).^2);

% Total number of cell centers
nodes = n*m;

% Spatial discretization steps
dx = Lx/n;
dy = Ly/m;

% Create grid of cell centers
x = linspace(dx/2,Lx-dx/2,n);
y = linspace(dy/2,Ly-dy/2,m);
[X,Y] = meshgrid(x,y);

% Set up counter
i = 0;
% Store Courant number sfor legend
names = cell(numel(tests),1);

% Loop through Courant numbers
for Co = tests
    % Calculate time step
    dt = Co*dx/2;
    % Increment counter
    i = i + 1;
    % At Co to legend names
    names{i} = [num2str(Co)];

    % Covective flux update matrices
    if upwind == true
        F = upwind_flux(X,Y,U,V,dx,dy,n,m,nodes,'periodic');
    else
        [Fx,Fy] = Lax_Wendroff_flux(X,Y,U,V,dx,dy,dt,n,m,nodes, ...
                                    'periodic');
    end

    % Calculate diffusive flux update matrix
    if diffusion == true
        G = diffusive_flux(D,dx,dy,n,m,nodes);
    else
        G = 0;
    end
```

```
    % Total update matrix
    if upwind == true
        A = speye(nodes) - dt*F + dt*G;
    else
        Ax = speye(nodes) - dt*Fx;
        Ay = speye(nodes) - dt*Fy + dt*G;
        A = Ay*Ax;
    end

    % Get eigenvalues of A matrix
    EIGS = eig(full(A));
    Re = real(EIGS);
    Im = imag(EIGS);
    plot(Re,Im,linestyle='none',marker='o',markersize=4)
    hold on

end

% Plot instable region in red
theta = linspace(0,2*pi,101);
theta2 = flip(theta);
shade_x = [cos(theta),4*cos(theta2)];
shade_y = [sin(theta),4*sin(theta2)];
fill(shade_x,shade_y,'r',FaceAlpha=0.15,linestyle='none');

% Plot principle axes
xline(0); yline(0)

% Axis limits a = size of limits
a = 1.5;
xlim([-a,a]); ylim([-a,a])
xticks(-a:a); yticks(-a:a)

% Pretty plot parameters
xlabel('Re($\lambda$)',interpreter='latex')
ylabel('Im($\lambda$)',interpreter='latex')
leg = legend(names,interpreter='latex',location='northeastoutside');
title(leg,'Co')
axis equal
box on
fontname('Serif'); fontsize(12,'points')

% ------------------------------------------------------------------------
% 'HW_2_pumps_test.m'
% Patrick Heng
% 27 Mar. 2025
% Test script to run the finite volume solver for the convection-
% diffusion equations with potential flow sink.
% ------------------------------------------------------------------------

close all; clear all; clc;

% Colormaps from '200 Colormaps' on the MATLAB file exchange
```

```
% SWITCH THIS TO A DEFAULT COLORMAP ('parula') IF YOU DO NOT HAVE THE
% FILE INSTALLED
cmap=slanCM(2);

% ------------------------------------------------------------------
% ----- INPUTS -----
upwind = true;
pumps = false;
diffusion = false;
step_IC = false;
periodic = false;


% Number of nodes (cell centers) in the computational domain,
% n -> x, m -> y
n = 240;
m = 40;

t_end = 20;


% Courant number, used to determine time step size
Co = 0.7;

% Initial concentration distribution
IC = @(X,Y) 0.25*exp(-50*((X-0.5).^2+(Y-0.5).^2));

% Diffusion constant
D = 0.005;

% Sink strength, 0 = off
Q = 0:0.1:1;

% Flow velocity components in absence of sources/sinks
u0 = 2;
v0 = 0;

% Length of domain
Lx = 6;
Ly = 1;

% ----- SPATIAL DISCRETIZATION -----
% Total number of cell centers
nodes = n*m;

% Spatial discretization steps
dx = Lx/n;
dy = Ly/m;

% Create grid of cell centers
x = linspace(dx/2,Lx-dx/2,n);
y = linspace(dy/2,Ly-dy/2,m);
[X,Y] = meshgrid(x,y);
```

```
% Calculate step size based on Courant number
%dt = Co*min(dx,dy)/max(abs(u0),abs(v0));
dt = Co/(u0/dx+v0/dy);
%


figure
hold on
step = floor(256/length(Q));
col = cmap(1:step:256,:);

i = 1;
for q = Q
% Velocity components as functions of the grid points
U = @(X,Y) u0 - q*(X-2)./((X-2).^2+(Y-2).^2);
V = @(X,Y) v0 - q*(Y-2)./((X-2).^2+(Y-2).^2);


% Determine flux function
if upwind == true
    F = upwind_flux(X,Y,U,V,dx,dy,n,m,nodes,periodic);
else
    [Fx,Fy] = Lax_Wendroff_flux(X,Y,U,V,dx,dy,dt,n,m,nodes,periodic);
end

% Turn on diffusion
if diffusion == true
    G = diffusive_flux(D,dx,dy,n,m,nodes);
    if upwind == true
        A = speye(nodes) - dt*F + dt*G;
    else
        Ax = speye(nodes) - dt*Fx;
        Ay = speye(nodes) - dt*Fy + dt*G;
        A = Ay*Ax;
    end
else
    if upwind == true
        A = speye(nodes) - dt*F;
    else
        Ax = speye(nodes) - dt*Fx;
        Ay = speye(nodes) - dt*Fy;
        A = Ay*Ax;
    end
end


% Determine the amount of time steps to get to tf
time_steps = ceil(t_end/dt);

% Store total pollutant mass, peak value of concentration
phi_tot = zeros(time_steps,1);

% Evaluate initial condition at cell centers
phi = IC(X,Y);
```

```
phi = reshape(phi',nodes,1);

% ----- TIME INTEGRATION -----
% Loop through time in an explicit marching scheme
for k = 1:time_steps

    % Store current pollutant mass, peak amplitude
    phi_tot(k) = sum(phi,'all')*dx*dy;
    % update solution
    phi = A*phi;

end

% Conservation plot of total mass of pollutant at each iteration
%

plot(0:dt:t_end,phi_tot,color=col(i,:),linewidth=2)
i = i + 1;
%plot(0:dt:t_end,phi_tot,linewidth=2,marker='none')
end

xlabel('$t$ (s)',interpreter='latex')
ylabel('$\phi_{tot}$',interpreter='latex')
ylim([0,1.01*max(phi_tot)])

cb = colorbar;
colormap(cmap);
ylabel(cb,'$q$',Interpreter='latex')
clim([0,1])

fontsize(16,'points'); fontname('Serif')
box on
grid on
%}

% ------------------------------------------------------------------------
% 'HW_2_err_conv.m'
% Patrick Heng
% 27 Mar. 2025
% Test script to generate data the finite volume solver error
% convergence plots.
% ------------------------------------------------------------------------

close all; clear all; clc;

% ------------------------------------------------------------------------
% ----- INPUTS -----
upwind = false;
step_IC = false;
periodic = true;

% Simulation length
t_end = 1;
```

```
% Courant number, used to determine time step size
Co = 1;

% Initial concentration distribution
IC = @(X,Y) 0.25*exp(-50*((X-0.5).^2+(Y-0.5).^2));

% Flow velocity components in absence of sources/sinks
u0 = 2;
v0 = 2;

% Velocity conmponents as functions of the grid points
q = 0;
U = @(X,Y) u0 - q*(X-2)./((X-2).^2+(Y-2).^2);
V = @(X,Y) v0 - q*(Y-2)./((X-2).^2+(Y-2).^2);


for kk = 2.^(2:10)

    % Number of nodes (cell centers) in the computational domain,
    % n -> x, m -> y
    n = 2*kk;
    m = kk;

    % Length of domain
    Lx = 2;
    Ly = 1;

    % Total number of cell centers
    nodes = n*m;

    % Spatial discretization steps
    dx = Lx/n;
    dy = Ly/m;

    % Create grid of cell centers
    x = linspace(dx/2,Lx-dx/2,n);
    y = linspace(dy/2,Ly-dy/2,m);
    [X,Y] = meshgrid(x,y);

    % Calculate step size based on Courant number
    %dt = Co*min(dx,dy)/max(abs(u0),abs(v0));
    dt = Co/(u0/dx+v0/dy);

    % Determine flux function
    if upwind == true
        F = upwind_flux(X,Y,U,V,dx,dy,n,m,nodes,periodic);
        A = speye(nodes) - dt*F;
    else
        [Fx,Fy] = Lax_Wendroff_flux(X,Y,U,V,dx,dy,dt,n,m,nodes,periodic);
        Ax = speye(nodes) - dt*Fx;
        Ay = speye(nodes) - dt*Fy;
        A = Ay*Ax;
    end
```

```
    phi = IC(X,Y);
    time_steps = ceil(t_end/dt);

    % Loop through time in an explicit marching scheme
    for k = 1:time_steps
        phi = reshape(phi',nodes,1);
        phi = A*phi;
        phi = reshape(phi,n,m)';
    end
    % Save the physical mesh and solution, CHANGE to 'phi_LW_' or
    % 'phi_UW_' for respective flux
    save(['phi_LW_' num2str(n) '_' num2str(m) '.mat'], ...
            'X','Y','phi','dx','dy')

end

% ---------------------------------------------------------------------
% 'HW_2_err_conv_plots.m'
% Patrick Heng
% 27 Mar. 2025
% Test script plot error convergence from our homework. MUST run
% 'HW_2_err_conv.m' (previous script) first to generate the data.
% ---------------------------------------------------------------------
close all; clc;

% Flux scheme, UW = upwind, LW = Lax Wendroff
flux = 'LW';

% Maximum exponent of the refined mesh (h = 2^k_max)
k_max = 10;

% Load maximum refined mesh and store for comparison
temp = load(['phi_' flux '_2048_1024.mat'],'phi');
phi_star = temp.phi;

% Preallocate error vectors
e_L2 = zeros(k_max-1,1);
e_inf = zeros(k_max-1,1);

% Loop through meshes, starting with the most refined and moving to
% the coarsest mesh
i = 1;
N = 2.^(k_max:-1:2);
h = zeros(k_max-1,1);

for n = N
    % Load the coarser mesh
    temp = load(['phi_' flux '_' num2str(2*n) '_' num2str(n) '.mat']);
    phi = temp.phi;
    dx = temp.dx;
    dy = temp.dy;
    h(i) = dx;

    % Calculate the error by subtracting the refined mesh evaluated at
```

```
    % the coarse mesh grid points. Since the refinement halves each time
    % this means evaluating the u_star at every 'step'.
    step = 2^(i-1);
    e = phi_star(1:step:end,1:step:end)-phi;
    e = e(1:end,:);

    % Calculate normalized L2 error over the domain
    e_L2(i) = norm(e,'fro')*sqrt(dx*dy);
    % Calculate the absolute maximum error
    e_inf(i) = max(e,[],'all');

    % Increment loop counter
    i = i + 1;

end

% Plot error on a log-log plot, compare to O(h) and O(h^2) growth
figure
% L2 error
loglog(h, e_L2,marker='o',color='b',linewidth=2)

hold on
% Maximum error
loglog(h, e_inf,marker='o',color='r',linewidth=2)

% O(h)
loglog(h,h.^1,linestyle=':',color='k',linewidth=2)
% O(h^2)
loglog(h,h.^2,linestyle='--',color='k',linewidth=2)

% Pretty plot parameters
grid on
box on
axis padded
yticks(10.^(-6:1:0))
xlabel('$h$',interpreter='latex')
ylabel('$||e||$',interpreter='latex')

legend('$||e||_{2}$', '$||e||_{\infty}$', '$O(h)$', ...
        '$O(h^2)$', interpreter='latex', location='best')
fontname('Serif'); fontsize(16,'points');
```