

# A Fully Lagrangian Advection Scheme

Mohammad Ali Yassaei

*Department of Mathematical and Statistical Sciences, University of Alberta,  
Edmonton, Alberta T6G 2G1, Canada*

John C. Bowman

*Department of Mathematical and Statistical Sciences, University of Alberta,  
Edmonton, Alberta T6G 2G1, Canada*

Anup Basu

*Department of Computing Science, University of Alberta, Edmonton, Alberta T6G  
2E8, Canada*

---

**Abstract**

A numerical method for passive scalar and self-advection dynamics, *Lagrangian rearrangement*, is proposed. This fully Lagrangian advection algorithm introduces no artificial numerical dissipation or interpolation of parcel values. In the inviscid limit, it preserves the infinity of Casimir invariants associated with parcel rearrangement. In the two-dimensional case presented here, these invariants are arbitrary  $C^1$  functions of the vorticity and concentration fields. The initial parcel centroids are evolved in a Lagrangian frame, using the method of characteristics. At any time this Lagrangian solution may be viewed by projecting it onto an Eulerian grid using a rearrangement map. The resulting rearrangement of initial parcel values is accomplished with a weighted Bresenham algorithm, which identifies quasi-optimal, distributed paths along which chains of parcels are pushed to fill in nearby empty cells. The error introduced by this rearrangement does not propagate to future time steps.

*Key words:*

numerical methods; Lagrangian advection; Casimir invariants; parcel rearrangement; relabelling symmetry

---

**1 Introduction**

The advection-diffusion equation arises in many scientific disciplines such as electro-osmotic flow, geophysical fluid dynamics (including meteorology, climate change, and hurricanes), thermonuclear fusion in plasmas, and mathematical biology. Another example, a model of electro-osmotic advection characterized by extremely small diffusion rates, provided the initial motivation for the development of this work [1]. Consider the advection-diffusion equation

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{U} = \mathbf{D} \nabla^2 \mathbf{U}, \quad (1)$$

where the advecting velocity field  $\mathbf{v} = \mathbf{v}(\mathbf{x}, t)$  is either a specified field (*passive advection*) or a functional of  $\mathbf{U}$  (*self-advection*), and  $\mathbf{D}$  is a constant diagonal diffusion matrix. In this case, we are primarily interested in the transport of a self-advected quantity  $\mathbf{U} = (\omega, C)$  by a two-dimensional fluid that flows in a domain with velocity  $\mathbf{v}$ , where the quantities  $C = C(\mathbf{x}, t)$  and  $\omega \hat{\mathbf{z}} = \boldsymbol{\omega}(\mathbf{x}, t) = \nabla \times \mathbf{v}$  represent the *concentration* and *vorticity* fields, respectively, and  $\hat{\mathbf{z}}$  is a unit vector normal to the plane of the flow. In the case where the velocity  $\mathbf{v}$  is *incompressible* ( $\nabla \cdot \mathbf{v} = 0$ ), the advection equation is an example of a flux-conservative system  $\partial \mathbf{U} / \partial t = -\nabla \cdot \mathbf{F}$ , where  $\mathbf{F} = \mathbf{v} \mathbf{U} - \mathbf{D} \cdot \nabla \mathbf{U}$ . In self-consistent advection, the velocity is typically determined by the incompressible

*Navier–Stokes* equation:

$$\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} = -\frac{1}{\rho} \nabla P + \nu \nabla^2 \mathbf{v}, \quad (2)$$

where  $\mathbf{v} = \mathbf{v}(x, t)$  is the velocity,  $P$  is the pressure,  $\rho$  is the density, and  $\nu$  is the viscosity. It is convenient to take the curl of (2) to eliminate the pressure field  $P$ , which leads to an equation for the vorticity:

$$\frac{\partial \omega}{\partial t} + \mathbf{v} \cdot \nabla \omega = \nu \nabla^2 \omega. \quad (3)$$

The concentration field evolves according to

$$\frac{\partial C}{\partial t} + \mathbf{v} \cdot \nabla C = D \nabla^2 C, \quad (4)$$

where  $D$  is a scalar diffusion constant, so that  $\mathbf{D} = \begin{pmatrix} \nu & 0 \\ 0 & D \end{pmatrix}$  in (1).

The general solution of the passive advection equation with no diffusion,

$$\frac{\partial C}{\partial t} + \mathbf{v} \cdot \nabla C = 0, \quad (5)$$

is a wave moving in the  $\mathbf{x}$  direction at speed  $\mathbf{v}$ . This solution  $C(\mathbf{x}, t) = C(\boldsymbol{\xi}_0(\mathbf{x}, t), 0)$  can be written in terms of the *initial parcel positions*  $\boldsymbol{\xi}_0 = \boldsymbol{\xi}_0(\mathbf{x}, t)$  defined by the Lagrangian position variable  $\boldsymbol{\xi}(t) = \boldsymbol{\xi}_0 + \int_0^t \mathbf{v}(\boldsymbol{\xi}(\tau), \tau) d\tau$  such that  $\boldsymbol{\xi}(t) = \mathbf{x}$ . In the special case where the velocity field  $\mathbf{v} = \mathbf{v}(\tau)$  is uniform, then  $\boldsymbol{\xi}_0(\mathbf{x}, t) = \mathbf{x} - \int_0^t \mathbf{v}(\tau) d\tau$  and  $C(\mathbf{x}, t) = C(\mathbf{x} - \int_0^t \mathbf{v}(\tau) d\tau, 0)$ .

It is well known that straightforward Eulerian finite differencing schemes for advection (e.g. forward-time centered-space differencing) are unstable ([2]). Conventional methods for avoiding this instability sacrifice accuracy. For example, the *Lax scheme* adds a diffusion term, or *numerical dissipation*, to the partial differential equation. This method is stable if the time step is chosen to satisfy the *Courant condition* (cf. [3]). To help counter the effect of numerical dissipation one can use *flux-corrected transport*, which adds an *anti-diffusion* term to the equation ([4]). *Upwind differencing* is a stable discretization that accounts for the fact that the rate of change of the flow is directionally dependent, but it adds unwanted numerical dissipation and is only first-order accurate in the time step. The *staggered leapfrog* method is a centered-time centered-space discretization achieved by using two staggered temporal partitions. It is second-order accurate in time, as is the *two-step Lax–Wendroff* scheme, in which the flux is calculated and used to determine the concentration field at the next time step ([5]).

*Lagrangian schemes* use a grid that moves with the flow as opposed to an Eulerian fixed grid. That is, the derivatives are now calculated in the Lagrangian frame of reference since the advection equation is most naturally described in a Lagrangian frame. In this frame the variable  $\mathbf{x}$  is also a function of  $t$ , and the *total derivative* of  $C$  is zero:

$$\frac{dC(\mathbf{x}(t), t)}{dt} = \frac{d\mathbf{x}}{dt} \cdot \nabla C + \frac{\partial C}{\partial t} = \mathbf{v} \cdot \nabla C + \frac{\partial C}{\partial t} = 0,$$

where  $\mathbf{v} = d\mathbf{x}/dt$ . This conservation equation expresses the fact that the scalar field  $C$  is neither created nor destroyed, only rearranged, by the advecting field  $\mathbf{v}$ . The two key components of any Lagrangian scheme are: (i) a method for following the characteristics and (ii) a method for viewing the solution on an Eulerian grid. In *fully Lagrangian schemes*, the grid is attached to and moves with the flow. In conventional implementations of Lagrangian schemes, one typically needs to re-mesh after a finite number of time steps. In this work, we propose a fully Lagrangian scheme that does not require re-meshing. The characteristics of the flow are followed using the classical fourth-order Runge–Kutta algorithm. The centroids of a finite number of discrete parcels characterized by distinct values of  $\xi_0$  are evolved on a spatial grid. At each time step, the new position of a given parcel is computed using its previous position and the local flow velocity.

To view a fully Lagrangian solution to an advection problem on an Eulerian grid, one also needs a projection scheme. Normally, area-weighted interpolation is used. However, in this work we construct a scheme for projecting onto the rearrangement manifold that respects an infinite hierarchy of conservation laws essential to a proper mathematical model of advection. Even when diffusion is added to this fully Lagrangian algorithm (using a semi-Lagrangian area-weighted interpolation scheme like the one described next), our proposed algorithm (illustrated in Fig. 4) exhibits much better energy decay characteristics (cf. Fig. 12).

The *particle-in-cell* method represents a piecewise constant approximation of the solution as a mesh of moving nodes (“particles”) advected by the flow. First, the positions of the particles are advected in the Lagrangian frame. Their associated physical attributes (in our case vorticity and concentration values) are then projected using area-weighted interpolation (described later) onto a finite Eulerian grid. One can then solve for the contributions to the evolution from diffusion and any other nonadvective terms on the Eulerian grid and project the result back onto the (continuum) Lagrangian grid, again using area-weighted interpolation. The procedure is then repeated for the next time step (for example, see [6] and [7]). Particle-in-cell methods tend to be noisy unless a very large number of particles are used. While they guarantee mass conservation, other conservation laws, such as energy conservation, are not necessarily guaranteed. Godunov [8], developed a discretization for fluid

dynamics with shocks by modeling the fluid as a large number of uniform cells joined by the *Riemann solution* for the dynamics at an interface between two uniform fluid regions. This is a *discontinuous Galerkin* method. For example, L. Fraccarollo, H. Capart, and Y. Zech [9] used the Godunov method to estimate the flux from the solution to the Riemann problem and obtain a finite-difference scheme. A higher-order extension of the Godunov method called the *piecewise parabolic method* is presented by Woodward and Colella [10]; it uses high-order spatial interpolation to represent steep discontinuities. In that work, the addition of diffusion is essential. A comparison between numerical methods for simulating hydrodynamic flow in two dimensions, concentrating on fluid flow with strong shocks, is discussed in [11].

In *semi-Lagrangian schemes*, the grid is fixed in time: although the advective derivatives are calculated in a Lagrangian frame, the other spatial derivatives are calculated on an Eulerian grid. The idea is to discretize the advective terms in a Lagrangian frame and then project the Lagrangian solution back onto an Eulerian grid, avoiding the instability of forward-time-centered-space Eulerian schemes and the inherent complications of fully Lagrangian re-meshing. For example, [12] discusses an advection scheme for shallow water waves. In this method, backtracked trajectories seldom land on a grid point. Therefore, interpolation, the most important part of a semi-Lagrangian scheme, is used in order to find the values of  $C$  between grid points. This interpolation can produce large numerical dissipation. Moreover, it does not respect fundamental properties of the flow, specifically the conservation of the Casimir invariants discussed in the next section. In problems involving a mass flow, it is sometimes possible to modify a semi-Lagrangian scheme so that it at least conserves mass (for example, see [13] and [14]).

In this work, a new numerical method for advection problems that avoids introducing artificial dissipation or the need to remesh, is proposed. In particular, we construct a numerical algorithm that conserves the global (integrated) value of an arbitrary smooth function of  $C$  in the limit of zero dissipation. For such systems, future values of the flow quantities are simply rearrangements of the current values. We constrain the numerical discretization to enforce this property by tracking the centroids of discrete parcels from their initial positions forward in time. At any time, the solution may be viewed by projecting it onto a *rearrangement manifold*, the set consisting of all rearrangements of the initial conditions. In Sect. 2 we introduce a method to conserve the Casimirs invariants, and the algorithm for this method is discussed in Sect. 3. The complexity of the algorithm is approximated in Sect. 5. We extend the algorithm to handle both diffusion and self-consistent advection in Sect. 4. Finally, in Sect. 6 the method is illustrated numerically and compared with a conventional semi-implicit scheme.

## 2 Lagrangian rearrangement

In this section we propose the method of *Lagrangian rearrangement* (LR) for projecting a fully Lagrangian solution of the passive advection equation, (5), onto an Eulerian grid. We first illustrate the method in the absence of diffusion. We constrain the numerical discretization to mimic an important analytic property of advection, namely, the conservation of the global integral of any smooth  $C^1$  function of the scalar concentration field:

$$\begin{aligned} \frac{d}{dt} \int f(C) d\mathbf{x} &= \int f'(C) \frac{\partial C}{\partial t} d\mathbf{x} = - \int f'(C) \mathbf{v} \cdot \nabla C d\mathbf{x} \\ &= - \int \mathbf{v} \cdot \nabla f(C) d\mathbf{x} = \int f(C) \nabla \cdot \mathbf{v} d\mathbf{x} = 0, \end{aligned} \quad (6)$$

using the incompressibility of the advecting velocity field  $\mathbf{v}$ . Note that the above equation also holds in the self-advected case when  $C$  is replaced by  $\omega$ , which depends on the advecting velocity  $\mathbf{v}$ , so that  $\frac{d}{dt} \int f(\omega) d\mathbf{x} = 0$ . Equation (6) expresses the conservation of uncountably many *Casimir invariants* (e.g. see [15]). It also holds when  $f$  is piecewise constant, where we interpret  $f'$  as a distribution.

If we take  $f$  to be unity for a specified range of  $C$  values and zero elsewhere, we see that the area of the flow associated with that range of  $C$  values must be invariant. Since connectedness is preserved by the continuous (and area-preserving) advection map, we deduce that a connected parcel having a particular  $C$  range gets mapped to a connected parcel of the same area. Moreover, if the  $C$  values are partitioned into  $n$  uniform ranges, the evolved state will consist of  $n$  distinct nonoverlapping patches associated with these ranges, possibly highly distorted. Therefore, assuming that  $C$  is initially bounded, as  $n$  goes to infinity, the resulting infinitesimal patches become rearranged into a highly complicated but nonoverlapping union of distorted parcels. Values of  $C$  that were not present in the initial configuration cannot be created, nor can existing  $C$  values be destroyed. Motivated by this exact property of infinitesimal parcel rearrangement, in the discrete case, we represent the solution  $C(\mathbf{x})$  as a piecewise-constant function. Under this assumption, the continuum property (6) reduces to

$$\frac{d}{dt} \sum_{i,j} f(C_{i,j}) = 0. \quad (7)$$

We propose that a mathematically faithful numerical model of advection should enforce this discrete version of the above exact infinitesimal property.

On taking  $f(C)$  to be 1 if  $C = C_0$  for some fixed value  $C_0$ , and zero otherwise, we see that the number of cells with value  $C_0$  would then be invariant, just as in the infinitesimal case. That is, the new values of  $C$  at the current time step

are prescribed to be simply rearrangements of the old values (and hence of the initial conditions) at the previous time step. This rearrangement property is known in the literature as a *relabelling symmetry*. If  $C$  is assumed to be piecewise continuous, then  $f(C)$  is certainly integrable and hence the Darboux integrability theorem guarantees that  $\sum_{i,j} f(C_{i,j})$  on a sequence of uniform  $N \times N$  grids converges to  $\int f(C) d\mathbf{x}$  as  $N \rightarrow \infty$ . Hence, the value of the latter integral, like the sum, must be constant. We thus see that (7), if it holds for all discrete grids, is a sufficient condition for the exact property (6) to hold.

In this work we consider a two-dimensional doubly periodic grid of cells. This allows us to focus on the implementation of the parcel rearrangement constraint without being distracted by additional complications associated with fixed boundaries. Each cell has an initial value, which is assigned to a parcel of fluid that will be advected in the Lagrangian frame. The *displacement*  $\boldsymbol{\xi} = \boldsymbol{\xi}_0 + \int_0^\tau \mathbf{v} dt$  of each parcel is calculated at time  $\tau$ , where  $\mathbf{v} = \mathbf{v}(\boldsymbol{\xi}, t)$  is the local velocity of the flow and  $\boldsymbol{\xi}_0$  is the initial displacement. In this fully Lagrangian formulation, the displacement is effectively calculated directly from the initial position, so that errors occurring in a time step do not propagate to future time steps: the Lagrangian to Eulerian projection onto the rearrangement manifold is used only for viewing the current state of the fluid, not for actually evolving the fluid. To evaluate the time integral, it is convenient to express the evolution of  $\boldsymbol{\xi}$  as the initial value problem  $d\boldsymbol{\xi}/dt = \mathbf{v}(\boldsymbol{\xi}, t)$ , where  $\boldsymbol{\xi}(0) = \boldsymbol{\xi}_0$ , for a specified function  $\mathbf{v}(\boldsymbol{\xi}, t)$ . In the *advection step*, the classical *fourth-order Runge–Kutta scheme*, (e.g. [2]) is used to calculate the current Lagrangian displacement of each parcel. No projection to the Eulerian frame is done here; the continuum Lagrangian position of the parcel is retained to initialize future advection steps.

In classical Lagrangian codes for advection by an incompressible flow, the vertices of the initially square parcels are advected by an area-preserving map to form an irregular Lagrangian mesh consisting of quadrilateral cells. To view the Lagrangian solution, one normally uses area-weighted interpolation to project the contributions from the quadrilaterals onto Eulerian cells. However, in this work, we propose that the centroids of these quadrilateral parcels should be mapped onto the rearrangement manifold. Given a fixed velocity field, the above integration at each stage amounts to a linear transformation of the quadrilateral region. Under this transformation, the centroid of a parcel thus maps to the centroid of the new quadrilateral formed by the evolved vertices. For the case of passive advection without diffusion, one does not need to know the actual quadrilateral vertices and instead advects only their centroids.

### 3 The rearrangement algorithm

Whenever we wish to view the current Lagrangian solution, we project a copy of it onto the rearrangement manifold (i.e. onto the Eulerian grid). For each parcel, we first determine the cell in which its current Lagrangian position lies. The problem that immediately arises is that a given cell may contain more than one such parcel; some sort of competition must then be held to determine which parcel should be projected to that cell. Recall that the discrete rearrangement condition underlying (7) requires that each parcel be mapped to a unique cell. Each cell would then adopt, at that instant, the fluid quantities of its associated parcel.

If the grid has  $n$  cells, we will have exactly  $n$  parcels. If each parcel lies within a distinct cell, there will be exactly one parcel per cell and we are done. However, in general, there may be some cells, denoted as “holes,” that do not have any associated parcels, and some cells denoted as “piles,” that contain more than one parcel. To enforce the preservation of the Casimir invariants during parcel rearrangement, only one parcel from a pile can be transferred to a hole; the others must be moved elsewhere. This step is denoted as the *rearrangement step*. By simply taking the extra parcels in a pile and transferring them to the nearest holes, we would create a discontinuity in the flow, which would constitute an enormous numerical defect. To resolve this issue, we propose the following *pushing algorithm*. This algorithm must not be confused with the so-called “particle-pushing” schemes used to follow the characteristics of the advecting flow. At this point, the advection step has been completed and we are now dealing with the problem of rearranging the  $n$  parcels into  $n$  cells for viewing the internal fully Lagrangian solution.

First, we must deal with the issue of treating all of the cells on an equal footing, without giving some the advantage of being processed first. At each stage of the Runge–Kutta advection step, we advect all parcels simultaneously, using the current local velocity, without reference to the locations of any other parcel. However, in the rearrangement step we cannot deal with all piles simultaneously—we must start from one particular cell. In the algorithm below, we start from the piles containing the greatest number of parcels since these are the most difficult cases to resolve. While building the list of such cells, we alternate between putting cells in the front or the back of the list. Refinements that effectively introduce further randomization, to avoid undue bias in our processing decisions, will be discussed in Sect. 3.2. Our rearrangement algorithm proceeds as follows:

- (1) Sort the piles by the number of parcels they contain.
- (2) Start with the piles containing the greatest number  $n$  of parcels. Process these cells first.

- (3) For each such pile (the *starting cell*), search outward in rectangular “shells” for a hole. If more than one hole is found on a shell, choose the one closest to the starting cell in the sense of having minimal *path weight*, as described in Sect. 3.2).
- (4) Form the discretized path from the starting cell towards the selected hole.
- (5) Attach the extra parcel in the starting cell to the first cell along this path, pushing parcels successively along this path until the selected hole is filled with a parcel belonging to the last cell along the path. The second cell in the path will thus take the extra parcel in the starting cell, and the next cell will take the parcel previously located in the second cell, and so on. Continue this pushing until the selected hole has been assigned a parcel, or in other words, an initial value. Now the starting cell has  $n - 1$  parcels in it.
- (6) Proceed with the next pile containing  $n$  parcels, and repeat steps 3–5 until no more cells containing  $n$  parcels remain.
- (7) Repeat Steps 2–6 until all cells contain exactly one parcel; that is, until  $n = 1$ .

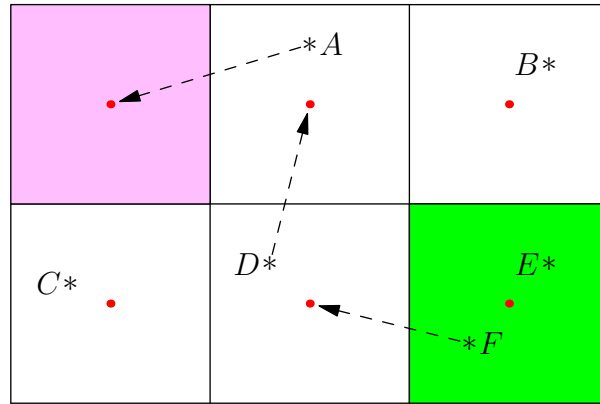


Fig. 1. Rearrangement step: stars denote parcel positions, and dots denote center of the cell.

Notice that after Step 5, all cells along the path will have a new parcel in them relative to their status at the end of the advection step. The corresponding hole is filled with one parcel, and the starting cell will have one less parcel in it than it had before. At the end of Step 7, all cells will contain exactly one parcel, as desired. The rearrangement step is then complete, and each cell can have a unique value assigned to it. For example, in Fig. 1, since the parcels  $E$  and  $F$  were in a pile, the nearest hole, and a discretized path to it, are found for parcel  $F$  and it is pushed to the first cell in the path, pushing the next parcel,  $D$ , to the next cell in the path, and ultimately, putting parcel  $A$  into the hole.

### 3.1 Parcel weights

A straightforward application of Bresenham’s algorithm [16] to select a (discretized) path between a pile and a hole can increase the possibility of a parcel being pushed multiple times. Suppose that in one time step, there are clusters of piles concentrated in one area and a group of holes in another nearby area. In pushing a parcel from a pile to a hole, each parcel in between is pushed once. If we process another parcel from the same crowded pile area and push toward the same hole area, the parcels in between may be pushed a second time. Processing all the parcels in this area can cause multiple pushes for parcels in between the pile-rich and hole-rich areas that can result in spurious streaks in the flow visualization. We can avoid multiple pushes by choosing a random path from a pile to a hole, introducing stochasticity into the algorithm. Moreover, a parcel may be pushed far away from its original Lagrangian position, resulting in large errors. In order to choose the best path, we introduce a path weight for discretizing the line. Let the *parcel weight*  $d$  represent the distance between the Lagrangian position of a parcel and the center of the cell into which it has been pushed. Each time the parcel is pushed, the cell containing it will change, resulting in a change in  $d$  (with no change to the Lagrangian position of the parcel). To alter the Bresenham algorithm to a weighted algorithm, we take  $d$  into account when calculating the path between a pile and a hole: parcels are inserted in such a way that the one with minimum  $d$  will be first in the list, so that it will be processed or pushed first. This ensures that parcels are not pushed too far from their Lagrangian positions.

### 3.2 A weighted Bresenham algorithm

In the weighted version of the code (Appendix A), the next cell to be included in the path will be the cell containing the parcel with the minimum parcel weight of all parcels in certain eligible neighbouring cells. In each case, to minimize excursions from the Lagrangian (advected) positions, we choose the cell that contains the parcel with minimal distance  $d$  from its Lagrangian position. In the rare case when parcels in the selected cells share the same minimal value of  $d$ , we will pick the cell that the original Bresenham algorithm would have picked. Depending on the choice of the next cell, we increment the  $x$  and/or  $y$  coordinate by one, and include the new cell in the path. The problem is thus reduced to a new problem, using the cell just selected as the new starting point. On reaching the hole, the algorithm terminates (see Theorem 1 in Appendix B). Other quadrants are dealt with in the same manner, but the signs of some of the parameters are changed to decrement (rather than increment) the  $x$  and/or  $y$  coordinates by one.

A question arises in the case where more than one hole is found in the same rectangular shell around the pile: which hole should be selected as the destination? We resolve such cases by invoking the weighted Bresenham algorithm on all possible choices of holes within the shell, without actually pushing any parcels. We define the *path weight* of a particular path to be the sum of all  $d$  values of the parcels to be pushed along that particular path. The Bresenham path that returns the minimum path weight and its corresponding hole will be chosen as the desired path and final destination, respectively. The theorem in Appendix B establishes an upper bound,  $\lceil 1.82x \rceil$ , on the number of steps required by our weighted Bresenham algorithm to draw a line between two grid points on a unit square lattice that are  $x$  units apart. This is not much greater than the  $\lceil \sqrt{2}x \rceil$  steps required to draw a diagonal Bresenham line on a unit lattice.

#### 4 Extension to self-consistent advection and diffusion

To solve the general advection-diffusion equation,

$$\frac{\partial U}{\partial t} + \mathbf{v} \cdot \nabla U = D \nabla^2 U, \quad (8)$$

we need to add the diffusion term, as well as a method for handling self-advection, to our algorithm. A significant source of error in the rearrangement algorithm used to project the solution to the Eulerian grid for viewing comes from the somewhat arbitrary algorithm used to pushing parcels from piles to holes. If the rearrangement algorithm were used to solve the diffusion or self-advection terms, errors would accumulate since the pushed values would be reused in calculating diffusion and self-advection. Consider the equation  $\partial \omega / \partial t + \mathbf{v} \cdot \nabla \omega = \nu \nabla^2 \omega$ . The advection term,  $\mathbf{v} \cdot \nabla \omega$ , would use the pushed values of  $\omega$  to calculate  $\mathbf{v}$ , so that errors associated with pushing parcels would propagate to future time steps. More importantly, the use of the rearranged vorticity field in calculating the diffusion term,  $\nu \nabla^2 \omega$ , would introduce large gradients in the flow, resulting in excessive diffusion. To prevent this propagation of error, we calculate the diffusion term as  $D \nabla^2 \omega_I$ , where  $\omega_I$  is the vorticity field on an Eulerian grid obtained by an area-weighted interpolation of the Lagrangian vorticity field  $\omega$  (it is difficult to calculate a Laplacian directly on a nonuniform Lagrangian grid). This decision does not degrade the desired conservation properties (Casimir invariants) of the advective term. Likewise, we calculate the advecting velocity  $\mathbf{v}_I$  from  $\omega$  by inverting a Laplacian. The advecting velocity itself does not need to be a rearrangement of the initial conditions in order to conserve the Casimir invariants. The concentration field  $C$  is treated in the same manner.

We now discuss the scheme for transferring information (interpolating) be-

tween the Lagrangian and Eulerian grids. The transfer is done by an *area-weighted bilinear interpolation*. Ideally, one should account for parcel distortion by the flow and project the area bounded by the evolved vertices of the parcel (which form a quadrilateral) to the Eulerian grid. However, as the evolved parcel shape is not essential to the demonstration of how Lagrangian rearrangement can be integrated with diffusion and self-advection, for simplicity we treat each parcel as a square centered on its current Lagrangian position (the parcel centroid), in analogy with the fixed particle shape used in particle-in-cell methods. To project information from a Lagrangian frame to an Eulerian lattice, consider the  $(\omega, C)$  values  $\mathbf{p}_i$  of the  $i$ th parcel (see Fig. 2). This square will overlap some cells in the grid. For cell  $j$ , let  $A_{ij}$  denote the area contributed by the square around parcel  $i$ . On accounting for the contributions from all parcels whose bounding squares overlap the cell  $j$ , the interpolated value  $\mathbf{U}_j$  is calculated as  $\mathbf{U}_j = \sum_i A_{ij}\mathbf{p}_i / \sum_i A_{ij}$ . If no parcels contribute to a cell, we search outward in successive rectangular shells around the empty cell for cells that have a contribution from some parcel. The first shell that is found to contain such cells is used to assign a value, namely the average interpolated value for these active cells, to the empty cell.

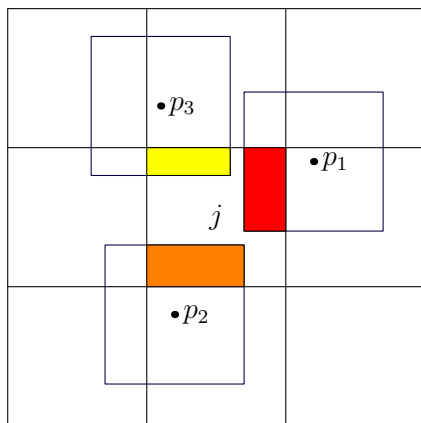


Fig. 2. Projection from a Lagrangian to Eulerian grid.

To transform information from the Eulerian to the Lagrangian frame, again consider a parcel and its bounding square (see Fig. 3). This square will overlap at most four cells in the grid. For each of the overlapping cells, compute the Lagrangian value for the parcel as  $\sum_j A_j \mathbf{U}_j$ , where  $\mathbf{U}_j$  is the Eulerian value for the  $j$ th cell and  $A_j$  is the overlapping area with the parcel's bounding square.

These above two procedures are typically used in tandem. For example, to calculate  $\nabla^2 \omega$  we need to interpolate the Lagrangian values onto the Eulerian grid, where it is convenient to calculate the Laplacian, and then transfer this contribution to the evolution back to the Lagrangian frame.

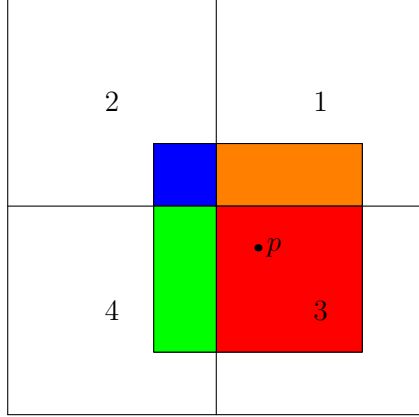


Fig. 3. Projection from a Eulerian to Lagrangian grid.

#### 4.1 Diffusion

In this section we consider the numerical treatment of the diffusion term in (8). The advection term is dealt with using the Lagrangian algorithm discussed in Sect. 2, and diffusion is handled by a *Crank–Nicholson* scheme (e.g. see [17]) in an Eulerian frame:

$$\frac{\mathbf{U}(t + \tau) - \mathbf{U}(t)}{\tau} = \mathbf{D} \frac{\nabla^2 \mathbf{U}(t + \tau) + \nabla^2 \mathbf{U}(t)}{2}.$$

To incorporate diffusion into our algorithm, it is helpful to split (8) into two pieces, one accounting for advection and the other for diffusion, using *operator splitting* (e.g. see [17] and [2]). Let  $\tilde{\mathbf{U}}$  be the Eulerian projection of the Lagrangian solution to the advection equation  $\partial \tilde{\mathbf{U}} / \partial t = -\mathbf{v} \cdot \nabla \tilde{\mathbf{U}}$  using area-weighted interpolation. Then, to solve for  $\mathbf{U}$ , including the effects of diffusion, we use the temporal finite-difference formula

$$\frac{\mathbf{U} - \tilde{\mathbf{U}}}{\tau} = \mathbf{D} \nabla^2 \left( \frac{\mathbf{U} + \tilde{\mathbf{U}}}{2} \right). \quad (9)$$

In this formulation, the most up-to-date (i.e., already advected) value,  $\tilde{\mathbf{U}}$ , is used as the starting value to calculate the diffused value  $\mathbf{U}$ . This implicit equation can be rewritten as  $\mathcal{L}(-\tau)\mathbf{U} = \mathcal{L}(\tau)\tilde{\mathbf{U}}$ , where  $\mathcal{L} = \mathbf{1} + \frac{1}{2}\mathbf{D}\tau\nabla^2$ . In order to calculate  $\mathbf{U}$  numerically, one needs to invert the Helmholtz operator  $\mathcal{L}$ . We accomplish this inversion with an efficient *multigrid* solver, using a single *V*-cycle iteration and the area-weighted interpolated Eulerian velocity  $\mathbf{U}_I$  as the initial guess. To determine the contribution of diffusion on the Lagrangian solution, we compute the difference  $\mathbf{U} - \tilde{\mathbf{U}}$ , project it back onto the Lagrangian frame using area-weighted bilinear interpolation, and add it onto the parcel values. Note that we do not simply project the diffused solution  $\mathbf{U}$  itself onto the Lagrangian frame, as this would contaminate the Lagrangian solution, violating the preservation of the Casimir invariants in the limit of

zero diffusion.

## 4.2 Self-advection

Until now, we have considered only the case of passive advection, where the velocity of the advecting flow is prescribed. In this section, we discuss self-consistent advection (self-advection), where the velocity of the underlying flow is a functional of  $\mathbf{U}$  itself. To calculate the velocity  $\mathbf{v}$ , it is convenient to adopt the vorticity formulation: using the Euler-projected value of the vorticity  $\omega_I = (\mathbf{U}_I)_\omega$  on the Eulerian grid determined by area-weighted interpolation, one can compute the stream function  $\psi$  from  $\omega_I = \nabla^2\psi$ . The inversion of the Laplacian here is done with 5 iterations (except for the very first step, when we used 40 iterations, due to the lack of a good initial guess) of a *V-cycle multigrid* solver, using the value of the stream function from the previous time step as the initial guess. Once a good approximation to  $\psi$  is determined, it is straightforward to calculate the advecting velocity:  $\mathbf{v} = \hat{\mathbf{z}} \times \nabla\psi$ . This velocity is used to evolve both the vorticity and the concentration fields, self-consistently, in the Lagrangian frame.

The entire self-consistent Eulerian–Lagrangian advection-diffusion algorithm is displayed in Fig. 4, in comparison with a conventional semi-Lagrangian scheme. In the red loop, we calculate the new Lagrangian parcel positions  $\boldsymbol{\xi}(t) = \boldsymbol{\xi}_0 + \int_0^t \mathbf{v}(\boldsymbol{\xi}(\tau), \tau) d\tau$ , increment the time step, and then repeat the procedure. In the blue boxes, to account for the effects of the diffusion term, we interpolate  $\mathbf{U}$  from the Lagrangian to the Eulerian frame, using area-weighted interpolation, and then use operator splitting and the Crank–Nicholson method in the Eulerian frame to solve for  $\mathbf{U}$  from (9). The diffused solution  $\mathbf{U} = \mathcal{L}^{-1}(-\tau)\mathcal{L}(\tau)\tilde{\mathbf{U}}$  is the conventional semi-Lagrangian solution (orange output). We then project the difference  $\mathbf{U} - \tilde{\mathbf{U}}$  in each Eulerian cell back onto the Lagrangian frame, using area-weighted interpolation to accrue the contributions from diffusion onto the parcels overlapping each cell. Finally, the time step is incremented and the procedure repeats. In the green boxes, we use the area-weighted interpolated value  $\tilde{\mathbf{U}}$  to calculate the stream function  $\psi = \nabla^{-2}\omega$  and thereby the self-advected velocity  $\mathbf{v} = \hat{\mathbf{z}} \times \nabla\psi$ . Our Lagrangian rearrangement algorithm is only used to project the values onto an Eulerian frame when we want to view the solution (yellow branch). Notice that the error in rearrangement does not propagate to future time steps since we do not feed it back to the solution in the advection loop.



Therefore, the probability of having a hole is  $P(0) = (1 - 1/n)^n$ , which approaches  $1/e$  for  $n$  sufficiently large; the probability of not having a hole thus approaches  $1 - 1/e$ . For most numerical simulations the domain chosen is very large (in our test simulations  $n$  is typically set to  $2^{18}$ ).

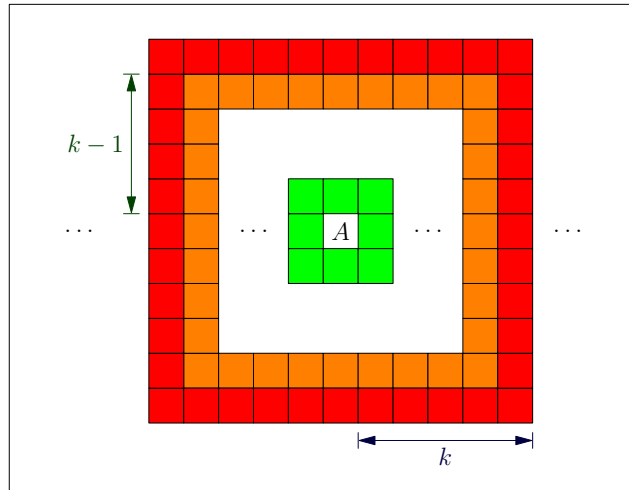


Fig. 5. Search order and search level.

For computing the cost of the search, one must calculate the probability of not finding a hole. In general, the goal is to start with a pile and search outward for the nearest hole in a shell-like domain. In Fig. 5, cell  $A$  is the pile, which is the center of the search, and the green ring is the first shell searched. Let the *level of the search*  $k = 1, 2, \dots$ , denote the shell number being searched, as shown in Fig. 5. Then the number of cells searched so far, up to but not including shell  $k$ , will be  $4k(k-1)$ , and the number of cells in the  $k$ th shell is  $8k$ . The algorithm is going to reach level  $k$  if it did not find any holes up to level  $k-1$ . Therefore, the probability of not having a hole in the first  $k-1$  shells is  $(1 - 1/e)^{4k(k-1)}$ , which is equivalent to the probability of searching  $k-1$  shells. Here, it is assumed that  $4k(k-1)$  is much smaller than  $n$ , so that the probability of not having a hole is independent of the number of parcels. Also, we assume that the probabilities of cells being or not being holes are independent of one another.

The final approximated *average searching cost*,  $A_s(n)$ , will be the sum, over the entire domain, of the above probability multiplied by the number of cells in shell  $k$ , so the searching cost is given by

$$A_s(n) = \sum_{k=1}^{\sqrt{n}/2} 8k \left(1 - \frac{1}{e}\right)^{4k(k-1)},$$

which tends to a small constant, (approximately 8.4) as  $n$  becomes large, meaning that the average cost of a single search grows insignificantly, as the domain size gets bigger. This sum was evaluated numerically using the sym-

bolic algebra program **Maple**.<sup>1</sup> To calculate the cost of identifying the path, recall that if more than one hole is detected in a shell, the Bresenham algorithm is carried out for all such holes in that shell. Therefore, we must also consider the probability of finding more than one hole in the  $k$ th shell. We show in Theorem 1 (Appendix B) that at most  $\lceil 1.82x \rceil$  steps are needed to find a path between a pile and a hole separated by a distance  $x$ . As in the above calculation of the searching cost, the probability of having to perform the Bresenham algorithm to find a path from a parcel to a cell in the  $k$ th shell is the same as the probability of not having a hole in the first  $k - 1$  shells, which is  $(1 - 1/e)^{4k(k-1)}$ . Moreover, the diagonal distance from the center of a search to a cell in the  $k$ th shell is  $k\sqrt{2}$ . Thus, the *average weighted Bresenham cost*  $A_b(n)$  is 1.82 multiplied by the approximate probable distance from a pile to a hole times the expected number of holes in the first shell  $k$  that contains a hole (given by the conditional probability in Appendix C):

$$A_b(n) \approx 1.82 \sum_{k=1}^{\sqrt{n}/2} k\sqrt{2} \left(1 - \frac{1}{e}\right)^{4k(k-1)} 8k \left(\frac{1}{e}\right) \frac{1}{1 - \left(1 - \frac{1}{e}\right)^{8k}}.$$

Using **Maple**, we found  $A_b(n)$  tends to a small constant (approximately 8.6) as the domain gets larger. A path is now identified, and we must push the parcels in this path. The actual pushing cost  $A_p(n)$  is

$$A_p(n) = 1.82 \sum_{k=1}^{\sqrt{n}/2} k\sqrt{2} \left(1 - \frac{1}{e}\right)^{4k(k-1)}.$$

Again, using **Maple** we found  $A_b(n)$  converges to a small constant (approximately 2.7) as the domain gets larger. In conclusion, the total complexity of the entire Lagrangian algorithm remains  $O(n)$  (that is, it is bounded by a constant times  $n$ ).

## 6 Results

An important feature of Lagrangian rearrangement is the conservation of Casimirs. In the absence of diffusion, any  $C^2$  function of vorticity is conserved. For example, the concentration field must attain the same set of values at all time steps. To test this attribute thoroughly we have initially set the concentration at the  $n$  grid points to  $n$  distinct values. At each time step the code verifies, in the absence of diffusion, that exactly one cell contains each assigned value at all times; that is, the predicted configuration is simply a rearrangement of the initial condition. To examine this visually, consider the concentration field

---

<sup>1</sup> Available from <http://www.maplesoft.com>

with an initial condition that consists only of the values zero and one, which we display as black and white pixels, respectively. We self-consistently evolve the field, which consists of two black strips above and below a white band, with semi-Lagrangian interpolation and Lagrangian rearrangement, using identical initial conditions and zero diffusion. The vorticity field initially is prescribed to be  $\omega = -4\pi \sin(2\pi x) \cos(2\pi y)$ , which corresponds to the initial velocity field  $v_x = \sin(2\pi x) \cos(2\pi y)$ , and  $v_y = -\cos(2\pi x) \sin(2\pi y)$ . We used a  $512 \times 512$  grid, so that the grid scale is  $h = 1.95 \times 10^{-3}$ . The time-step  $\tau$  was chosen to be 10 times the Courant condition, or  $1.95 \times 10^{-2}$  units (we checked that the Lagrangian displacements computed by our fourth-order Runge–Kutta integration were still sufficiently accurate at this large time step). The fact that we can run the algorithm at 10 times the Courant condition is an important feature of Lagrangian schemes.

A snapshot of the same advected stage predicted by the two methods is shown in Fig. 6. The left frame shows the concentration field for the semi-Lagrangian method, and the frame on the right illustrates the predictions of the LR method. With the selected palette, it is observed that the interpolation in the semi-Lagrangian method leads to coloured pixels, despite the absence of physical diffusion. This indicates that the method introduces spurious numerical diffusion, whereas the LR method produces only black and white pixels, because there is no numerical diffusion.

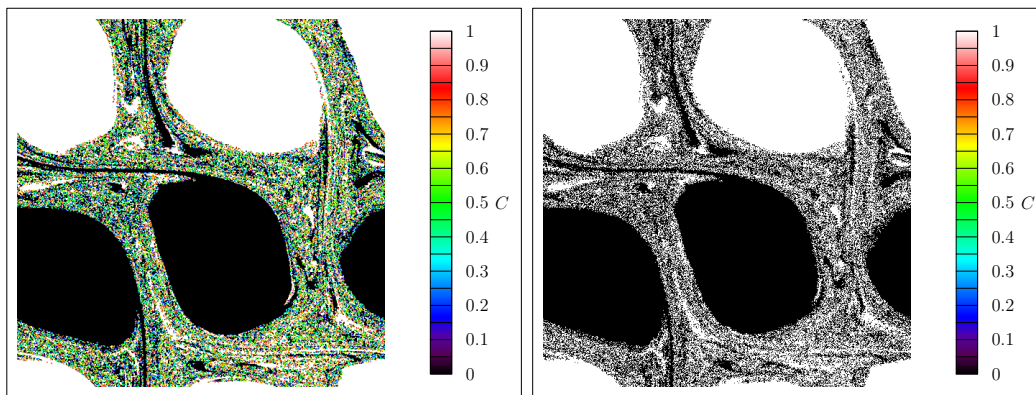


Fig. 6. Semi-Lagrangian interpolation vs. LR solution after 750 time steps.

Lagrangian rearrangement can also be applied to the full advection-diffusion equation. Using the same initial vorticity field as in the previous example, we consider a self-advected flow with diffusion constant  $D = 2 \times 10^{-6}$ . We compare the pushing method for viewing the Lagrangian data with area-weighted interpolated (semi-Lagrangian) projection. We initialized the runs with the same doubly periodic initial concentration field (0 at the top and bottom varying gradually to 1 in the middle). Figs. 7–9 demonstrate the advection and diffusion of this field, with the semi-Lagrangian result on the left frame of each figure, and the LR prediction on the right frame. Although the

two methods produce similar results, the solution in the left frame is seen to be smoother, due to the interpolation of concentration values by the semi-Lagrangian method. In contrast, the slight roughness at the pixel level exhibited in the right frame is a consequence both of lack of anomalous numerical diffusion and the inherent arbitrariness of our parcel-pushing algorithm.

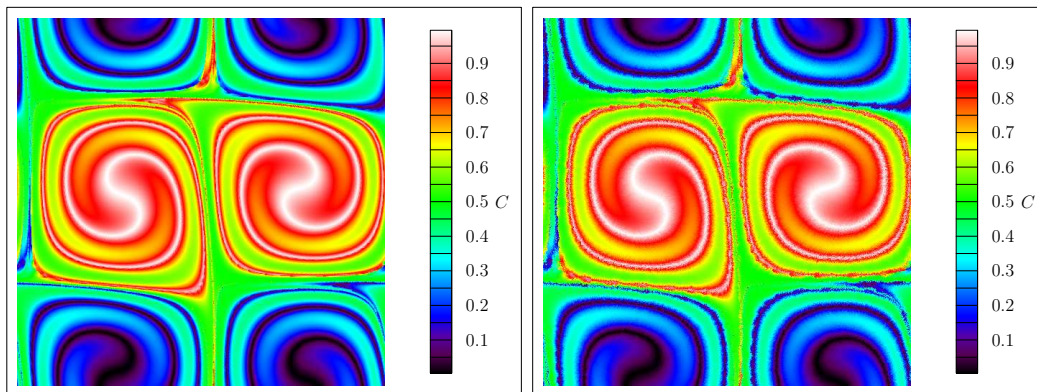


Fig. 7. Semi-Lagrangian vs. LR after 100 time steps, with diffusion.

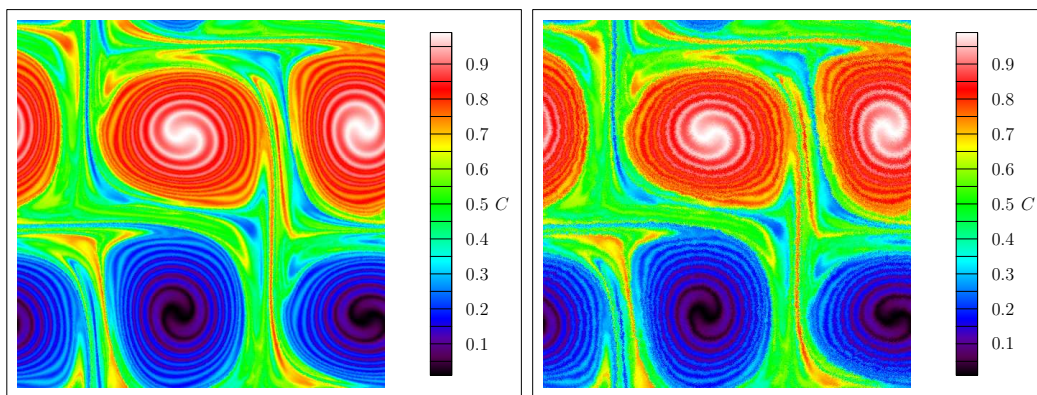


Fig. 8. Semi-Lagrangian vs. LR after 500 time steps, with diffusion.

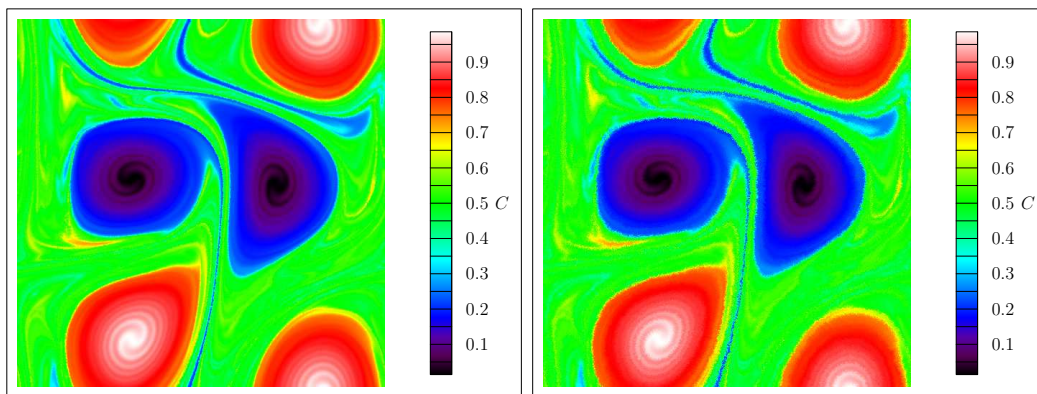


Fig. 9. Semi-Lagrangian vs. LR after 1000 time steps, with diffusion.

In comparing the LR and semi-Lagrangian methods, we now emphasize the conservation of Casimir invariants. In the absence of diffusion, the numerical approximation of the *concentration energy*  $\frac{1}{2} \int C^2 d\mathbf{x}$  and the *enstrophy*

$Z = \frac{1}{2} \int \omega^2 d\mathbf{x}$  should be conserved. In Figs. 10 and 11 it is observed that the LR method indeed respects the invariance of these two important quantities. On the other hand, in the semi-Lagrangian method both of these quantities decay as a result of unwanted numerical diffusion. Moreover, in the case of a viscous fluid we will consider the energy equation obtained by multiplying both sides of (8) by  $\mathbf{U}$  and integrating over the domain. On integrating by parts and invoking the incompressibility condition and doubly periodic boundary conditions, one finds that the concentration energy fields should decay as

$$\frac{1}{2} \frac{\partial}{\partial t} \int C^2 d\mathbf{x} = -D \int |\nabla C|^2 d\mathbf{x}. \quad (10)$$

It is convenient to introduce the normalized energy decay rates

$$\frac{\frac{\partial}{\partial t} \int C^2 d\mathbf{x}}{\int C^2 d\mathbf{x}} \quad \text{and} \quad \frac{-2D \int |\nabla C|^2 d\mathbf{x}}{\int C^2 d\mathbf{x}}. \quad (11)$$

Analogous quantities for the enstrophy decay rate, obtained by replacing  $C$  by the scalar vorticity  $\omega$  are also of interest.

According to (10), the energy decay rates for each field, as calculated by the two corresponding expressions, should agree. The values in (11) for both the semi-Lagrangian and LR methods are plotted in Fig. 12 (and in Fig. 13 for the  $\omega$  field). As seen in the graphs, the decay rates predicted by the semi-Lagrangian method (denoted by the subscript  $I$ ) do not agree, whereas the rates for the rearranged Lagrangian solution (denoted by the subscript  $R$ ) agree much better, to within the expected spatial discretization error. The anomalous and erratic numerical diffusion exhibited by the semi-Lagrangian solution is evident both in the departure of the blue and green curves and in the suppression of the energy content of  $\nabla C_I$  relative to the other predictions. This shows that the term  $\mathbf{v} \cdot \nabla \mathbf{U}$  is not modelled by the semi-Lagrangian method to respect the correct energy decay rate for a real fluid.

## 7 Conclusion

We propose a new fully Lagrangian method for solving advection equations. This method preserves Casimir invariants such as energy and momentum, just as inviscid fluids do. We argue in the nondiffusive case that the discretized values of the concentration field, when viewed on an Eulerian grid, should only be rearranged, not changed, to enforce a discretized version of the Casimir invariance property of the nonlinear advection term. That is, the pixels that are ultimately used to visualize the flow should be treated as infinitesimal

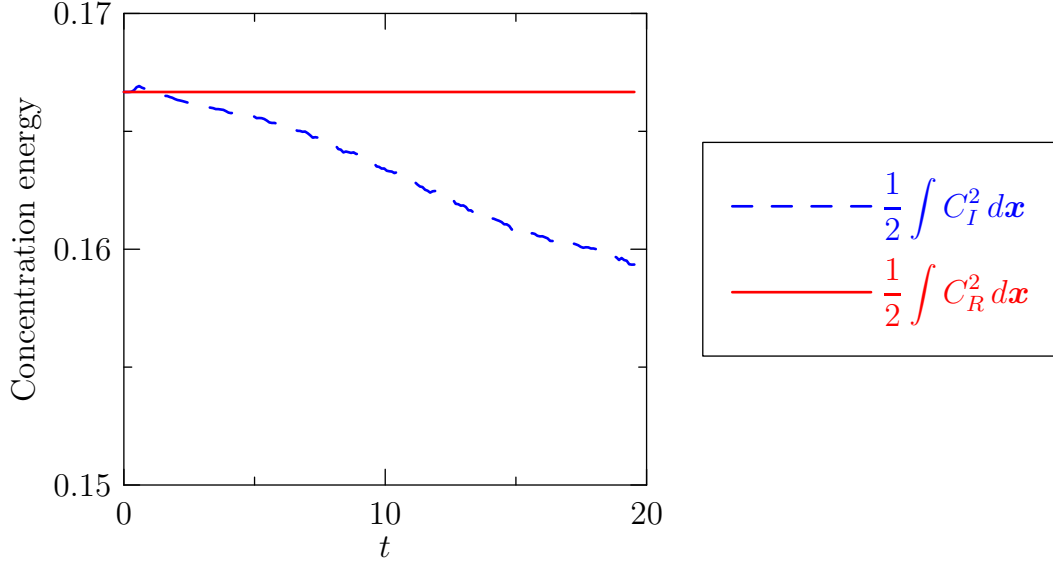


Fig. 10. Evolution of the concentration field energy predicted by the semi-Lagrangian ( $I$ ) and rearrangement ( $R$ ) methods when  $\nu = D = 0$ .

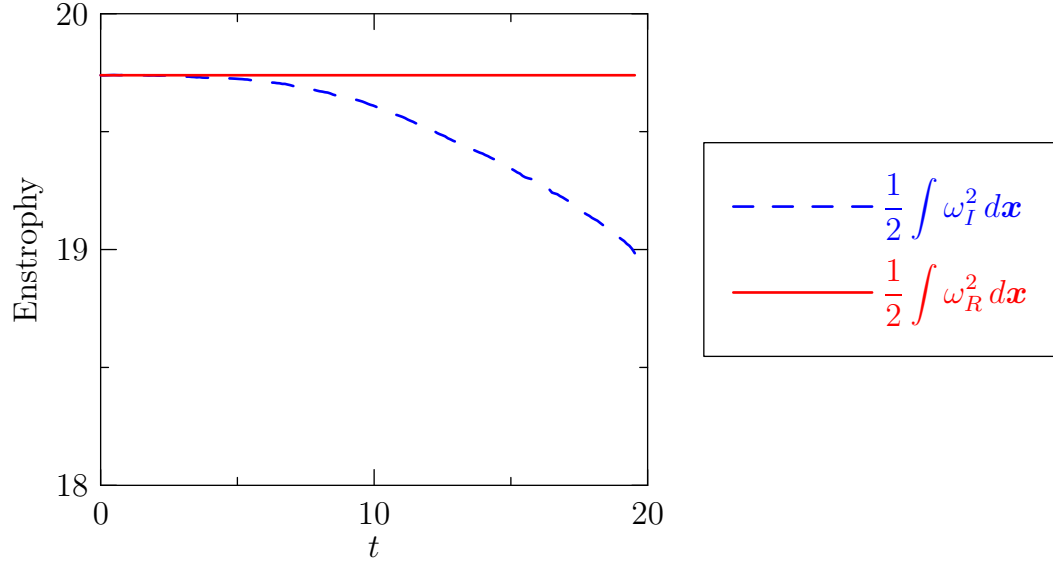


Fig. 11. Evolution of the enstrophy predicted by the semi-Lagrangian ( $I$ ) and rearrangement ( $R$ ) methods when  $\nu = D = 0$ .

parcels: at all times, the outputted concentration field should be some rearrangement of its initial state. The velocity field is used to advect the values of the concentration and vorticity field in the Lagrangian frame. In projecting the Lagrangian solution to an Eulerian frame, some of the cells (holes) will have no corresponding Lagrangian value, and some of the cells (piles) will have more than one value. In order to find the best projection from Lagrangian to Eulerian coordinates, we determine a path from a pile to the nearest hole and push the chain of parcels (values) towards the hole. This path is calculated using a weighted version of the Bresenham algorithm for drawing digitized

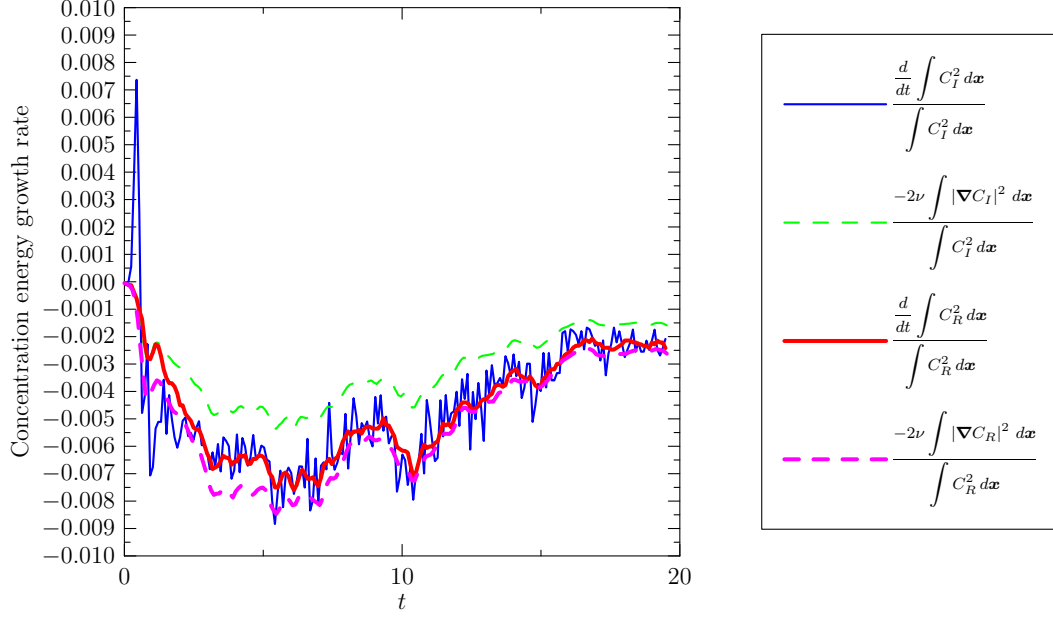


Fig. 12. Energy decay rates for the concentration field predicted by the semi-Lagrangian ( $I$ ) and rearrangement ( $R$ ) methods.

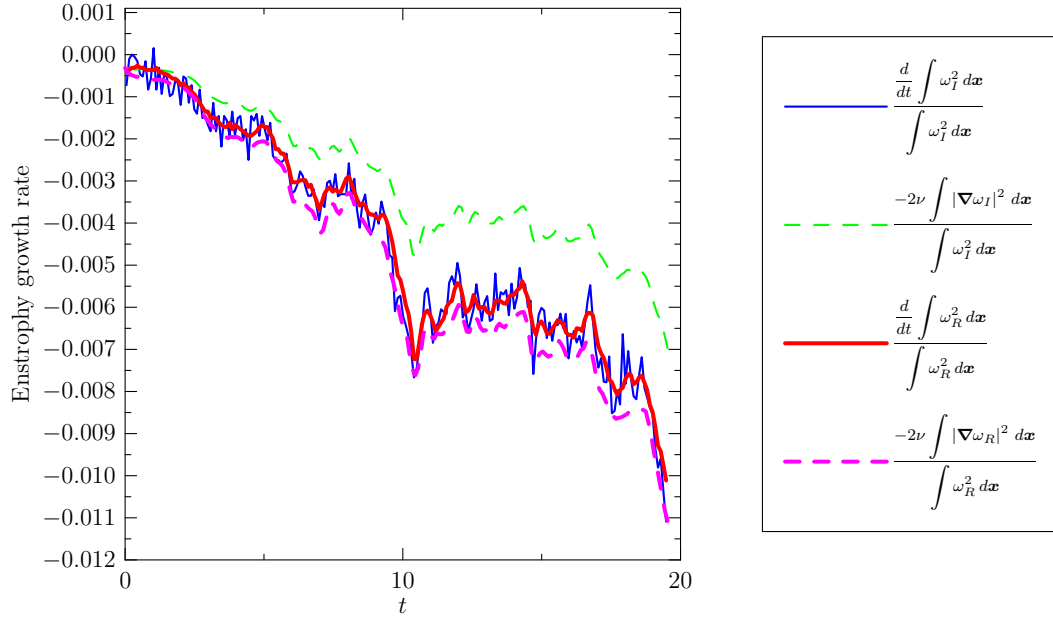


Fig. 13. Enstrophy decay rates predicted by the Lagrangian semi-Lagrangian ( $I$ ) and rearrangement ( $R$ ) methods.

lines. This modified version reduces the error in pushing parcels away from their calculated Lagrangian position: the weight used is the distance between the parcel and its position determined by Lagrangian advection. The weighted version attempts to choose a path containing parcels with minimal weight. To prevent the error in parcel pushing from propagating to the next time step, we do not reuse this information in future time steps. Lagrangian rearrangement

thus merely provides an energy-respecting (and Casimir-respecting) filter for viewing the current Lagrangian solution in an Eulerian frame. To deal with self-advection, we interpolate the advected Lagrangian vorticity field onto the Eulerian grid, avoiding any pushing error from being transmitted to the velocity field. We then calculate the new velocity from the projected vorticity by inverting a Laplacian with a multigrid solver.

In summary, the method of Lagrangian rearrangement can be used to view Lagrangian solutions of passive and self-consistent advection, avoiding the introduction of any artificial numerical diffusion. More importantly, for practical applications, we have shown how operator splitting can be used to account for the effects of physical diffusion in a manner than yields better concentration energy decay characteristics than straightforward Lagrangian interpolation.

## A Weighted Bresenham Algorithm

In our weighted version of the Bresenham algorithm, the choice of the next point in the path depends on the weight of the eligible neighbouring cells. Fig. A.1 shows eight cases depending on the location of the destination point. One seeks a neighbouring cell in the general direction of the path with the lowest weight. If the slope is zero, the cells to the north-east, east, and south-east of the current cell are searched. The north-east and east cells are searched if the slope is between zero and one, the north, north-east, and east cells are searched if the slope is one, and the north and north-east cells are searched if the slope is greater than one. If the slope is infinity, the cells to the north-west, north, and north-east of the current cell are searched. Should the weight of two or more cells be the same, the original Bresenham algorithm will be the tie-breaker. The following is `Asymptote` code (a vector graphics language for technical drawing [19]), along with eight sample output paths, for our weighted Bresenham algorithm (optimized so that all cases are mapped to the first quadrant).

## B Termination of Weighted Bresenham Algorithm

**Theorem 1 (Termination of weighted Bresenham)** *The weighted Bresenham algorithm produces a finite path between any two points on a regular lattice. For a unit square lattice, at most  $\lceil 1.82x \rceil$  steps are needed to connect two points that are a distance  $x$  apart.*

Proof

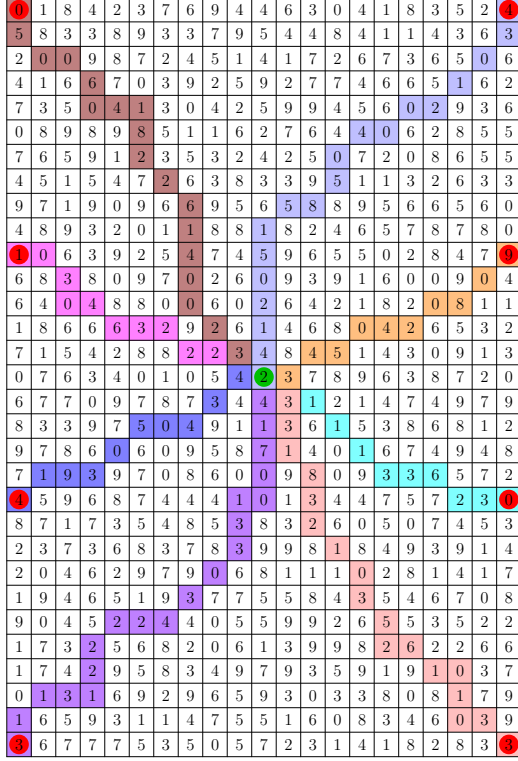


Fig. A.1. Paths of local minimal weight from the center circle to eight edge circles, as determined by the weighted Bresenham algorithm and the indicated numerical parcel weights.

Let  $A$  and  $B$  be given on the grid. We want to find the desired path between them. For simplicity, we assume that  $B$  is inside or on the boundary of the first octant with respect to  $A$ . Without loss of generality, consider a unit square lattice. As described before, depending on the position of  $B$ , we have either two or three choices for the next cell to be included in the path. If one of these choices is the grid point  $B$ , we are done; the algorithm terminates after choosing  $B$ . Otherwise, in choosing one of the immediate neighbours of the current cell, we will take a step of size 1 or  $\sqrt{2}$ . We must show that there exists a fixed number  $\delta > 0$  such that the distance to the point  $B$  in each step is always reduced by at least  $\delta$ . The algorithm will then terminate in a finite number of steps.

Case (i): Assume  $B$  lies on the same horizontal line as  $A$ , so that the slope of the line from  $A$  to  $B$  is zero, ( $m = 0$ ). In this case (Fig. B.1), the next point in the path is one of the points  $C$ ,  $D$ , or  $F$ . If we choose  $D$ , then since  $\overline{DB} = \overline{AB} - 1$ , a step of 1 is taken toward  $B$ . Suppose instead that we choose  $C$ . On letting  $x = \overline{AB} \geq 2$  and  $\delta = x - \overline{CB} = 1 + \overline{DB} - \overline{CB} < 1$  (since  $\overline{DB} < \overline{CB}$ ), and noting that  $\delta = \overline{AD} + \overline{DB} - \overline{CB} = \overline{CD} + \overline{DB} - \overline{CB} > 0$ , we find that

$$(x - \delta)^2 = \overline{CB}^2 = \overline{DB}^2 + 1 = (\overline{AB} - 1)^2 + 1 = \overline{AB}^2 - 2\overline{AB} + 2 = x^2 - 2x + 2,$$

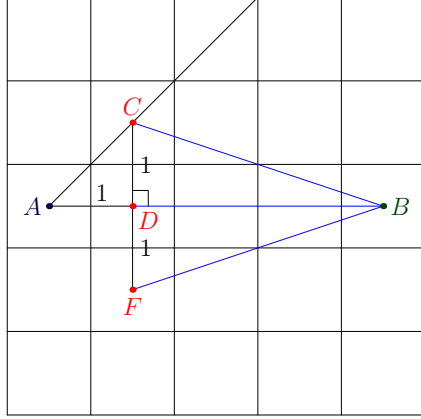


Fig. B.1. Case(i):  $m = 0$ .

so that  $-2x\delta + \delta^2 = -2x + 2$ . We then deduce from  $x \geq 2$  that  $2 - \delta^2 = 2x(1 - \delta) \geq 4(1 - \delta)$ . Thus  $\delta^2 - 4\delta + 2 \leq 0 \Rightarrow \delta \in [2 - \sqrt{2}, 1)$ . The same argument of course also holds for the choice  $F$ . The distance reduction in this case is thus at least  $2 - \sqrt{2}$ .

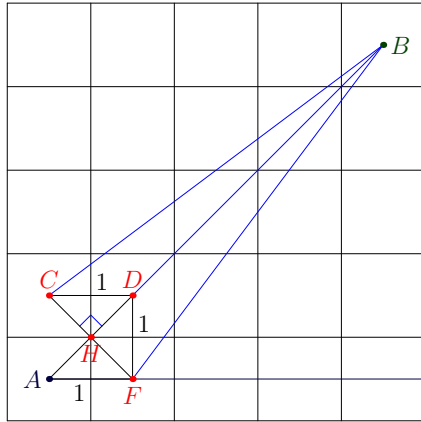


Fig. B.2. Case(ii):  $m = 1$ .

Case (ii): Assume that the slope of the line from  $A$  to  $B$  is 1. In this case (Fig. B.2), the next point in the path will be one of the points  $C$ ,  $D$ , or  $F$ . Here  $\overline{AB} = \overline{DB} - \sqrt{2}$ . If we choose  $D$ , we take a step of size  $\sqrt{2}$  toward  $B$ . Suppose instead that we choose  $C$ . We see that  $\overline{CH} = \overline{AH} = 1/\sqrt{2}$ . On letting  $x = \overline{AB}$ , we obtain

$$(x - \delta)^2 = \overline{CB}^2 = \overline{HB}^2 + \overline{CH}^2 = \left(x - \frac{1}{\sqrt{2}}\right)^2 + \frac{1}{2} = x^2 - \sqrt{2}x + 1.$$

Thus  $-2x\delta + \delta^2 = -\sqrt{2}x + 1$ . We know that  $x \geq 2\sqrt{2}$  since  $B$  is not one of the choices, so  $\delta^2 - 1 = x(2\delta - \sqrt{2}) \geq 2\sqrt{2}(2\delta - \sqrt{2}) = 4\sqrt{2}\delta - 4$ . Now  $\delta^2 - 4\sqrt{2}\delta + 3 \leq 0 \Rightarrow \delta \geq 2\sqrt{2} - \sqrt{5}$ . The same argument of course also holds for the choice  $F$ . The distance reduction in this case is thus at least  $2\sqrt{2} - \sqrt{5}$ .

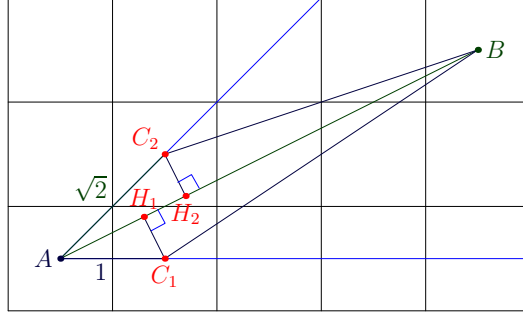


Fig. B.3. Case(iii):  $0 < m < 1$ .

Case (iii): Assume  $B$  lies inside the first octant ( $0 < m < 1$ ). In this case, Fig. B.3, the next point in the path is one of the two points  $C_1$  or  $C_2$ . Let  $x = \overline{AB}$ . Notice that  $x \geq \sqrt{5}$  since  $B$  is not one of the points  $C_1$  or  $C_2$ . Let  $C$  be the point ( $C_1$  or  $C_2$ ) that is selected, drop the perpendicular  $\overline{CH}$  to  $\overline{AB}$ . Let  $y = \overline{CB}$  and  $z = \overline{AC}$  and note that  $z = 1$  if  $C = C_1$  and  $z = \sqrt{2}$  if  $C = C_2$ . Since  $0 < \angle CAH < \pi/4$ , we know that  $\overline{AH}/z > 1/\sqrt{2}$ . On letting  $\delta = x - \overline{CB}$ , we find that

$$\begin{aligned} (x - \delta)^2 &= \overline{CB}^2 = \overline{HB}^2 + \overline{CH}^2 = (\overline{AB} - \overline{AH})^2 + \overline{AC}^2 - \overline{AH}^2 \\ &= x^2 - 2x\overline{AH} + z^2 < x^2 - 2x\frac{z}{\sqrt{2}} + z^2. \end{aligned}$$

Thus  $-2x\delta + \delta^2 < -\sqrt{2}xz + z^2$ , so that  $z^2 - \delta^2 > x(\sqrt{2}z - 2\delta) \geq \sqrt{5}(\sqrt{2}z - 2\delta)$ . Hence,  $\delta^2 - 2\sqrt{5}\delta + \sqrt{10}z - z^2 < 0$ . For  $z = 1$  this implies that  $\delta > \sqrt{5} - \sqrt{6 - \sqrt{10}}$  and for  $z = \sqrt{2}$  this implies that  $\delta > \sqrt{5} - \sqrt{7 - 2\sqrt{5}}$ .

In all cases, the distance between the point  $B$  and the new included point is thus always less than  $\overline{AB}$  by an amount

$$\begin{aligned} \delta &= \min \left\{ 1, \sqrt{2}, 2 - \sqrt{2}, 2\sqrt{2} - \sqrt{5}, \sqrt{5} - \sqrt{6 - \sqrt{10}}, \sqrt{5} - \sqrt{7 - 2\sqrt{5}} \right\} \\ &= \sqrt{5} - \sqrt{6 - \sqrt{10}} > 0.551. \end{aligned}$$

That is, at most  $\lceil 1.82\overline{AB} \rceil$  steps will be required to reach the point  $B$ .

## C Multiple-Hole Expected Value

The expected number of holes in a shell  $k$  (with  $8k$  cells) containing a hole is

$$\langle j \rangle = \frac{\sum_{j=1}^{8k} j \binom{8k}{j} \left(1 - \frac{1}{e}\right)^{8k-j} \left(\frac{1}{e}\right)^j}{\sum_{j=1}^{8k} \binom{8k}{j} \left(1 - \frac{1}{e}\right)^{8k-j} \left(\frac{1}{e}\right)^j} = \frac{\sum_{j=1}^{8k} j \binom{8k}{j} r^j}{\sum_{j=1}^{8k} \binom{8k}{j} r^j},$$

where  $r = 1/(e - 1)$ . Since the probability of not having a hole in  $8k$  cells is  $(1 - 1/e)^{8k}$ , we get

$$\sum_{j=1}^{8k} \binom{8k}{j} \left(1 - \frac{1}{e}\right)^{8k-j} \left(\frac{1}{e}\right)^j = 1 - \left(1 - \frac{1}{e}\right)^{8k},$$

which on multiplying both sides by  $(r + 1)^{8k} = (1 - 1/e)^{-8k}$ , can be written as

$$\sum_{j=1}^{8k} \binom{8k}{j} r^j = (r + 1)^{8k} - 1.$$

On differentiating both sides with respect to  $r$  and then multiplying by  $r$ , it follows that

$$\sum_{j=1}^{8k} \binom{8k}{j} j r^j = 8kr(r + 1)^{8k-1}.$$

The expected number of holes thus evaluates to

$$\langle j \rangle = \frac{8kr(r + 1)^{8k-1}}{(r + 1)^{8k} - 1} = \frac{8k \left(1 - \frac{1}{r + 1}\right)}{1 - (r + 1)^{-8k}} = \frac{8k \left(\frac{1}{e}\right)}{1 - \left(1 - \frac{1}{e}\right)^{8k}}.$$

This of course is just the probability  $1/e$  of finding a hole times the number of holes, conditioned on the probability that the shell contains at least one hole.

## References

- [1] J. Alam and J. Bowman, Energy-conserving simulation of incompressible Electro-Osmotic and pressure-driven flow, *Theoretical and Computational Fluid Dynamics* **16**, 1 (2002).
- [2] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes, The Art of Scientific Computing*, 2nd ed. (Cambridge Univ. Press, Cambridge, 1992).

- [3] R. Courant, K. Friedrichs, and H. Lewy, On the partial differential equations of mathematical physics, *IBM Journal of Research and Development* **11**, 215 (1967).
- [4] Y. Wang and K. Hutter, Comparisons of numerical methods with respect to convectively-dominated problems, *International Journal for Numerical Methods in Fluids* **37**, 721 (2001).
- [5] P. Lax and B. Wendroff, System of conservation laws, *Communications on Pure and Applied Mathematics* **13**, 217 (1960).
- [6] J. Leboeuf, T. Tajima, and J. Dawson, Magnetohydrodynamics Particle Code for Fluid Simulation of Plasmas, *Journal of Computational Physics* **31**, 379 (1979).
- [7] Y. Grigoryev, V. Vshivkov, and M. Fedoruk, *Numerical “Particle-in-Cell” Methods—theory and applications* (Brill Academic Publishers, Utrecht-Boston, 2002).
- [8] S. Godunov, A finite difference method for the numerical computation of discontinuous solutions of the equations of fluid dynamics, *Sbornik. Mathematics* **47**, 271 (1959).
- [9] L. Fraccarollo, H. Capart, and Y. Zech, A Godunov method for the computation of erosional shallow water transients, *International Journal For Numerical Methods In Fluids* **41**, 951 (2003).
- [10] P. Woodward and P. Colella, The piecewise parabolic method (PPM) for gas-dynamical simulations, *Journal of Computational Physics* **54**, 174 (1984).
- [11] P. Woodward and P. Colella, The Numerical Simulation of Two-Dimensional Fluid Flow with Strong Shocks, *Journal of Computational Physics* **54**, 115 (1984).
- [12] J. Behrens, in *Modeling and Computation in Environmental Sciences, Proceedings of the First GAMM-Seminar at ICA Stuttgart*, Stuttgart, Vol. 59 of Notes on Numerical Fluid Mechanics (Vieweg, Braunschweig, 1997), pp. 49–60.
- [13] Behrens and Mentrup, in *Recent Advances in Adaptive Computation*, Providence, edited by Z.-C. Shi, Z. Chen, T. Tang, and D. Yu (American Mathematical Society, Providence, Rhode Island, 2005), Vol. 383, pp. 219–234.
- [14] L. M. Leslie and R. J. Purser, Three-Dimensional Mass-Conserving Semi-Lagrangian Scheme Employing Forward Trajectories, *Monthly Weather Review* **123**, 25 (1995).
- [15] P. J. Morrison, Hamiltonian Description of the Ideal fluid, *Rev. Mod. Phys.* **70**, 467 (1998).
- [16] J. E. Bresenham, Algorithm for computer control of a digital plotter, *IBM Systems Journal* **4**, 25 (1965).

- [17] W. Ames, *Numerical Methods for Partial Differential Equations* (Academic Press, San Diego, CA, 1977).
- [18] S. Basse and A. V. Gelder, *Computer Algorithms, Introduction to Design and Analysis* (Addison-Wesley, Ontario, 2000).
- [19] A. Hammerlindl, J. C. Bowman, and R. T. Prince, available online at <http://asymptote.sourceforge.net>.