University of Alberta

A FULLY LAGRANGIAN ADVECTION SCHEME

by

Mohammad Ali Yassaei

©

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Science

in

Applied Mathematics

Department of Mathematical and Statistical Sciences
Edmonton, Alberta
Spring, 2006

Your file Votre référence
ISBN: 0-494-13914-5
Our file Notre référence
ISBN: 0-494-13914-5

Abstract

A numerical method for passive scalar and self-advection dynamics, *Lagrangian rearrangement*, is proposed. This fully Lagrangian advection scheme 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.

Acknowledgements

I wish to thank my supervisor, Dr. John Bowman, sincerely for his guidance, constructive instruction, and comments. I would like to express my gratitude for his assistance and commitment. I am also extremely grateful to Dr. Anup Basu for his support and very helpful discussions, particularly for his contributions to the complexity analysis of the Lagrangian rearrangement algorithm. Special thanks are due to my colleagues, particularly, Malcolm Roberts, for helpful discussions and ideas. Above all, I am forever grateful to all my family, especially my wife, Maryam, and my parents for their unlimited support and encouragement. I deeply appreciate their help, inspiration, and dedication.

Table of Contents

List of Figures

Chapter 1

Introduction

Numerical methods are useful for solving nonlinear partial differential equations, such as the advection-diffusion equation

$$\frac{\partial \boldsymbol{U}}{\partial t} + \boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{U} = \boldsymbol{D}\nabla^2\boldsymbol{U}, \tag{1.1}$$

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

$$\frac{\partial \boldsymbol{U}}{\partial t} = -\boldsymbol{\nabla}\cdot\boldsymbol{F}, \tag{1.2}$$

where $\boldsymbol{F} = \boldsymbol{vU} - \boldsymbol{D\nabla U}$.

In self-consistent advection, the velocity is typically determined by the incompressible *Navier–Stokes* equation:

$$\frac{\partial \boldsymbol{v}}{\partial t} + \boldsymbol{v}\cdot\boldsymbol{\nabla v} = -\frac{1}{\rho}\boldsymbol{\nabla}P + \nu\nabla^2\boldsymbol{v}, \tag{1.3}$$

where $\boldsymbol{v} = \boldsymbol{v}(x,t)$ is the velocity, P is the pressure, ρ is the density, and ν is the viscosity. It is convenient to take the curl of (1.3) to eliminate the pressure field P, which leads to an equation for the vorticity (1.4) (see Appendix A):

$$\frac{\partial \omega}{\partial t} + \boldsymbol{v}\cdot\boldsymbol{\nabla}\omega = \nu\nabla^2\omega. \tag{1.4}$$

Moreover, the equation for the concentration field is:

$$\frac{\partial C}{\partial t} + \boldsymbol{v}\cdot\boldsymbol{\nabla}C = D\nabla^2 C. \tag{1.5}$$

Thus, the matrix $\boldsymbol{D}$ in (1.1) can be written

$$\begin{pmatrix} \nu & 0 \\ 0 & D \end{pmatrix},$$

where D is the diffusion of the concentration field.

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

$$\frac{\partial C}{\partial t} + \boldsymbol{v}\cdot\boldsymbol{\nabla}C = 0, \tag{1.6}$$

is a wave moving in $\boldsymbol{x}$ at speed $\boldsymbol{v}$. This solution $C(\boldsymbol{x},t) = C\left(\boldsymbol{\xi}_0(\boldsymbol{x},t),0\right)$ can be written in terms of the *initial parcel positions* $\boldsymbol{\xi}_0 = \boldsymbol{\xi}_0(\boldsymbol{x},t)$ defined by the Lagrangian position variable $\boldsymbol{\xi}(t) = \boldsymbol{\xi}_0 + \int_0^t \boldsymbol{v}(\boldsymbol{\xi}(\tau),\tau)\,d\tau$ such that $\boldsymbol{\xi}(t) = \boldsymbol{x}$. In the special case where the velocity field $\boldsymbol{v} = \boldsymbol{v}(\tau)$ is uniform, then $\boldsymbol{\xi}_0(\boldsymbol{x},t) = \boldsymbol{x} - \int_0^t \boldsymbol{v}(\tau)\,d\tau$ and $C(\boldsymbol{x},t) = C\left(\boldsymbol{x} - \int_0^t \boldsymbol{v}(\tau)\,d\tau, 0\right)$. In order to obtain a numerical solution for a nonuniform velocity field, one needs to represent the equation in a discrete form.

There are problems with the existing methods for discretizing (1.6): certain methods produce an unstable, unphysical, or inaccurate solution, while others involve costly and complex computations. Moreover, some of these schemes do not respect fundamental properties of the flow. For instance, the conservation of mass and other exact invariants is not obeyed by many of these schemes. In this work, a new method that avoids these problems 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.

Before describing our proposed scheme, we begin by reviewing some of the traditional approaches to the discretization of (1.6).

1.A Eulerian schemes

In order to solve partial differential equations numerically, we can use the *finite difference* method to represent them in a discrete form. In this method only the quantities at finite number of points are considered: a numerical grid is superimposed on the domain. For example, in *centered finite differencing*, a partial derivative of a function at a grid point is calculated using the difference between the computed function values at its immediate neighbours.

1.A.1 Forward-time centered-space scheme

In this simplest version of the *Eulerian* scheme, the spatial (x) axis is partitioned into N uniform subintervals of width h and the time (t) axis is partitioned into T uniform subintervals of width τ. For $j = 0, 1, \ldots, N$ and $n = 0, 1, \ldots, T$,

$$x_j = x_0 + jh,$$

$$t_n = t_0 + n\tau,$$

Letting $C_j^n \doteq C(t_n, x_j)$ ($\doteq$ denotes a definition), the one-dimensional advection equation can then be discretized as

$$\frac{C_j^{n+1} - C_j^n}{\tau} = -v\frac{C_{j+1}^n - C_{j-1}^n}{2h}. \tag{1.7}$$

This *forward-time centered-space* scheme (FTCS) is an example of an *explicit* time-advance method. The advantage of the explicit *Euler* method used here for the time advance is that we can express C_j^{n+1} (the quantity at the jth grid position at a time represented by the index $n + 1$) explicitly in terms of quantities evaluated at time n. Only the values at time step n must be known and stored to calculate the information at time step $n+1$; this makes the computations easy and minimizes the storage requirements. The disadvantage of this method is the instability of the solution, even when the *Courant condition* described next is satisfied (e.g., see the Von Neumann stability analysis of the FTCS method by Press *et al.* [1992]).

1.A.2 Lax scheme

Due to the instability problems of the forward-time centered-space Euler differencing algorithm, a better solution has been developed, namely, the *Lax method.* In this scheme, the discrete form of the differential equation is altered by replacing the quantity C_j^n on the left-hand side of (1.7) by $\frac{1}{2}(C_{j+1}^n + C_{j-1}^n)$. This replacement solves the issue of instability but adds a diffusion term, or *numerical dissipation*, to the partial differential equation. This method is stable and satisfies the *Courant condition* (cf. [Courant *et al.* 1967]), which states that the numerical time step τ must be shorter than the time required for a wave moving at speed $|\boldsymbol{v}|$ to travel across a cell of width h:

$$\frac{|\boldsymbol{v}|\,\tau}{h} \leqslant 1.$$

However the Lax scheme is not very accurate. To partially alleviate this problem, which results from unwanted numerical diffusion, we can try to use a smaller grid; however, this will increase the computation time. Another way to get around the problem of unwanted numerical dissipation is to add an *anti-diffusion* term in order to decrease the numerical dissipation of the scheme. The latter solution is known as *flux-corrected transport* (FCT). For example, see the discussion by Wang & Hutter [2001].

1.A.3 Other Eulerian methods

Other similar methods have been developed over the years to avoid the weaknesses of the previously described schemes. For example, *upwind differencing* takes into account the fact that the rate of change of the flow is directionally dependent. The time-advance formula is then modified: if $v_j^n > 0$ we use (in one dimension)

$$\frac{C_j^{n+1} - C_j^n}{\tau} = -v_j^n \frac{C_j^n - C_{j-1}^n}{h},$$

and if $v_j^n < 0$ we use

$$\frac{C_j^{n+1} - C_j^n}{\tau} = -v_j^n \frac{C_{j+1}^n - C_j^n}{h}.$$

This method is stable, but it is only first-order accurate in the time step τ.

The *staggered leapfrog* method, defined for the flux-conservative form (1.2), 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 time step $n+1$ [Lax & Wendroff 1960].

1.B Lagrangian schemes

Instead of using a stationary grid, as in the Eulerian methods discussed previously, *Lagrangian* schemes use a grid that moves with the flow. That is, the derivatives are now calculated in the Lagrangian, as opposed to Eulerian, frame of reference since the advection equation is most naturally described in a Lagrangian frame. In this frame the variable $\boldsymbol{x}$ is also a function of t, and the chain rule tells us that the *total derivative* of C in this frame is zero:

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

where $\boldsymbol{v} = d\boldsymbol{x}/dt$. This conservation equation expresses the fact that the scalar field C is neither created nor destroyed, only rearranged, by the advecting field $\boldsymbol{v}$.

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.

1.B.1 Fully Lagrangian schemes

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 $\boldsymbol{\xi}_0$ are evolved on a spatial grid. At each time step, the new position of each particular 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. 3.3) exhibits much better energy decay characteristics (cf. Fig. 4.22).

1.B.2 Semi-Lagrangian schemes

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 Lagrangian terms of the advection equation, without having to deal with the instability of the FTCS scheme or the inherent complications of fully Lagrangian re-meshing. For example, Behrens [1995] discusses an advection scheme for shallow water waves:

$$\frac{dC(\boldsymbol{x}(t),t)}{dt} = \boldsymbol{a}(\boldsymbol{x},t)\cdot\boldsymbol{\nabla}C + \frac{\partial C}{\partial t} = 0,$$

where the wind $d\boldsymbol{x}/dt = \boldsymbol{a}(\boldsymbol{x},t)$ is given. The new value of the scalar C at a grid point is calculated according to the discretization

$$\frac{C(\boldsymbol{x}_m, t_n+\tau) - C(\boldsymbol{x}_m - \boldsymbol{\alpha}_m, t_n)}{2\tau} = 0,$$

where

$$\boldsymbol{\alpha}_m^{(k+1)} = \tau\boldsymbol{a}\left(\boldsymbol{x}_m - \frac{\boldsymbol{\alpha}_m^{(k)}}{2}, t_n + \frac{\tau}{2}\right)$$

is the displacement relative to the grid point $\boldsymbol{x}_m$. However, 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, effectively transferring information from the Eulerian to the Lagrangian grid. This interpolation can produce large numerical dissipation. Moreover, it does not respect the invariance of certain analytical invariants (specifically, the Casimir invariants discussed in the next Chapter). 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 Behrens & Mentrup [2005] and Leslie & Purser [1995]).

1.B.3 Particle-in-cell method

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 in Sect. 3.A) 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 Leboeuf *et al.* [1979] and Grigoryev *et al.* [2002]). 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.

1.B.4 Godunov schemes

Godunov [1959] developed a discretization for fluid dynamics problems with shocks by modelling the fluid as a large number of uniform cells joined by the *Riemann solution* for the dynamics of an interface between two uniform fluid regions. This is a *discontinuous Galerkin* method. Fraccarollo *et al.* [2003] use 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* (PPM) is presented by Woodward & Colella [1984a]; it uses high-order spatial interpolation to represent steep discontinuities. In their work, the addition of diffusion is essential; however, they claim that this does not have a significant impact on the results. A comparison between numerical methods for simulating hydrodynamic flow in two dimensions, concentrating on fluid flow with strong shocks, is discussed by Woodward & Colella [1984b].

Chapter 2

Lagrangian Rearrangement

In this chapter, we introduce a new method, which we call *Lagrangian rearrangement* (LR), for projecting a fully Lagrangian solution of the passive advection equation, (1.6), onto an Eulerian grid, 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:

$$\begin{aligned}\frac{d}{dt}\int f(C)\,d\boldsymbol{x} &= \int f'(C)\frac{\partial C}{\partial t}\,d\boldsymbol{x} = -\int f'(C)\boldsymbol{v}\cdot\boldsymbol{\nabla} C\,d\boldsymbol{x}\\ &= -\int \boldsymbol{v}\cdot\boldsymbol{\nabla} f(C)\,d\boldsymbol{x} = \int f(C)\boldsymbol{\nabla}\cdot\boldsymbol{v}\,d\boldsymbol{x} = 0, \qquad (2.1)\end{aligned}$$

due to the incompressibility of the velocity field $\boldsymbol{v}$. Note that the above equation also holds in the self-advected case when C is replaced by ω, which depends on the advecting velocity $\boldsymbol{v}$, so that

$$\frac{d}{dt}\int f(\omega)\,d\boldsymbol{x} = 0.$$

This uncountable infinity of invariants are known as *Casimir invariants* (e.g., see Morrison [1998]).

The above argument also holds when f is piecewise constant, where we interpret f' as a distribution. If we take f to be unity for a narrow 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, we see that the resulting infinitesimal patches are 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 as a finite union of piecewise-constant functions. Under this assumption, the continuum property 2.1 reduces to 2.2. The discretized version of the above analytic property that we thus propose in this work should be enforced is

$$\frac{d}{dt}\sum_{i,j} f(C_{i,j}) = 0. \tag{2.2}$$

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 so the Darboux integrability theorem guarantees that $\sum_{i,j} f(C_{i,j})$ on a sequence of uniform grids converges to $\int f(C)\, d\boldsymbol{x}$. Hence the value of the latter integral, like the sum, must be constant. We thus see that (2.2), if it holds for all discrete grids, is a sufficient condition for the exact property 2.1 to hold.

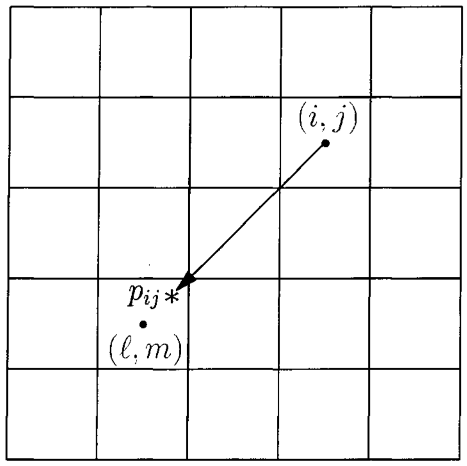


Figure 2.1: The trajectory for parcel p_{ij} from cell (i,j) to cell (ℓ,m).

We adopt a two-dimensional N_x by N_y grid, with $N = N_x \times N_y$ grid points or *cells.* Each cell has an initial value, which is assigned to a parcel of fluid that will be advected in the Lagrangian frame. The *displacement* $\boldsymbol{\xi} = \boldsymbol{\xi}_0 + \int_0^\tau \boldsymbol{v}\, dt$ of each parcel is calculated at time τ, where $\boldsymbol{v} = \boldsymbol{v}(\boldsymbol{\xi}, t)$ 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 integral, it is natural to express the evolution of $\boldsymbol{\xi}$ as the initial value problem

$$\frac{d\boldsymbol{\xi}}{dt} = \boldsymbol{v}(\boldsymbol{\xi}, t), \qquad \boldsymbol{\xi}(0) = \boldsymbol{\xi}_0,$$

for a specified function $\boldsymbol{v}(\boldsymbol{\xi}, t)$. The classical *fourth-order Runge–Kutta scheme*, (e.g., Press *et al.* [1992]) can be used to calculate the current Lagrangian displacement $\boldsymbol{\xi}$ of each parcel, denoting the time step by τ:

$$\boldsymbol{\xi}(\tau) = \boldsymbol{\xi}_0 + \frac{1}{6}\boldsymbol{\xi}_1 + \frac{1}{3}\boldsymbol{\xi}_2 + \frac{1}{3}\boldsymbol{\xi}_3 + \frac{1}{6}\boldsymbol{\xi}_4 + \mathcal{O}(\tau^5),$$

where

$$\begin{aligned}
\boldsymbol{\xi}_1 &= \tau \boldsymbol{v}\left(\boldsymbol{\xi}_0, 0\right), \\
\boldsymbol{\xi}_2 &= \tau \boldsymbol{v}\left(\boldsymbol{\xi}_0 + \frac{\boldsymbol{\xi}_1}{2}, \frac{\tau}{2}\right), \\
\boldsymbol{\xi}_3 &= \tau \boldsymbol{v}\left(\boldsymbol{\xi}_0 + \frac{\boldsymbol{\xi}_2}{2}, \frac{\tau}{2}\right), \\
\boldsymbol{\xi}_4 &= \tau \boldsymbol{v}\left(\boldsymbol{\xi}_0 + \boldsymbol{\xi}_3, \tau\right).
\end{aligned}$$

This process is called the *advection step.* For example, in Fig. 2.1, the parcel p_{ij} is initially in the cell (i, j). After the advection step at time t, the parcel p_{ij} now lies in the cell (ℓ, m). Note that 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 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, we do not need to know the actual quadrilateral vertices and instead only advect their centroids.

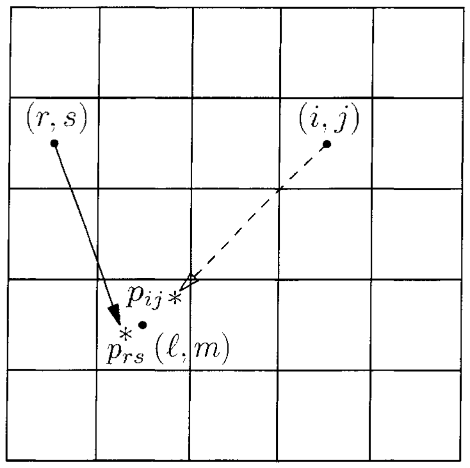


Figure 2.2: Competition between the parcels p_{ij} and p_{rs} for the cell (ℓ, m).

2.A The pushing algorithm

Whenever we wish to view the current Lagrangian solution, we project a copy of it to the rearrangement manifold (i.e. the Eulerian grid). We first determine for each parcel the cell containing its Lagrangian position. The problem that immediately arises is that a given cell may contain more than one parcel. In general, there may be a competition between two or more parcels to be projected to the same cell. Such a case is shown in Fig. 2.2. The parcels p_{rs} and p_{ij} respectively from cells (r, s) and (i, j) are both competing for the cell (ℓ, m).

To respect the discrete rearrangement condition underlying (2.2), each parcel must be mapped to a unique cell. 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. The cells would then adopt those parcels as their projected values. However, in general, there may be some cells, denoted by "*holes*," that do not have any associated parcels, and some cells denoted by "*piles*," that contain more than one parcel. In Fig. 2.3, the cell that initially contained the parcel A will be a hole after the advection step, and the cell that now contains a and e is a pile. To enforce the preservation of the previously discussed Casimir invariants, resulting from parcel rearrangement, only one of parcels a and e can be transferred to cell F; the other must be transferred elsewhere. This step is denoted as a *rearrangement step.* By simply taking the extra parcels in a pile and transferring them to the nearest holes, we would cause a "jump" or 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

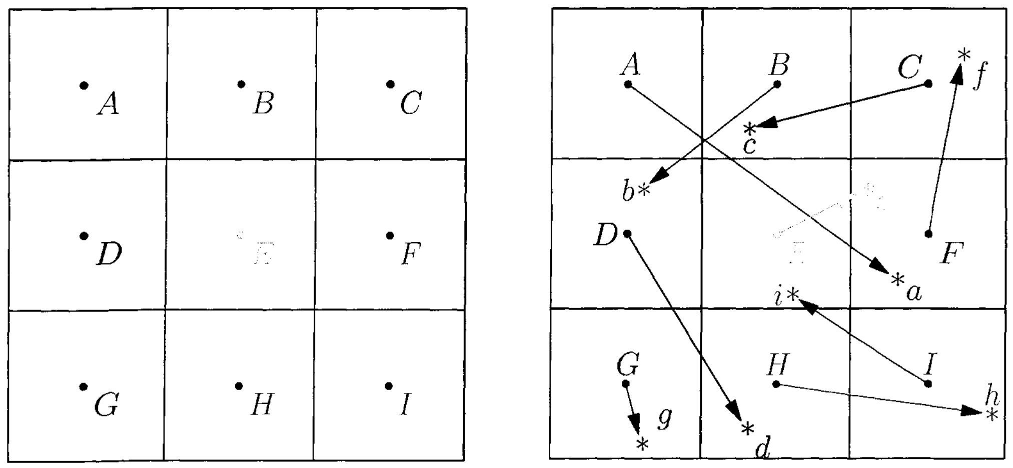


Figure 2.3: Initial position and the advection step.

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. 2.B.1. Here now is our rearrangement algorithm:

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. 2.B.1).

4. Form the discretized path from the starting cell toward the selected hole. (We will discuss the algorithms for discretization of the path later).

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 $n = 1$.

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 C value assigned to it. For example, in Fig. 2.4, since the

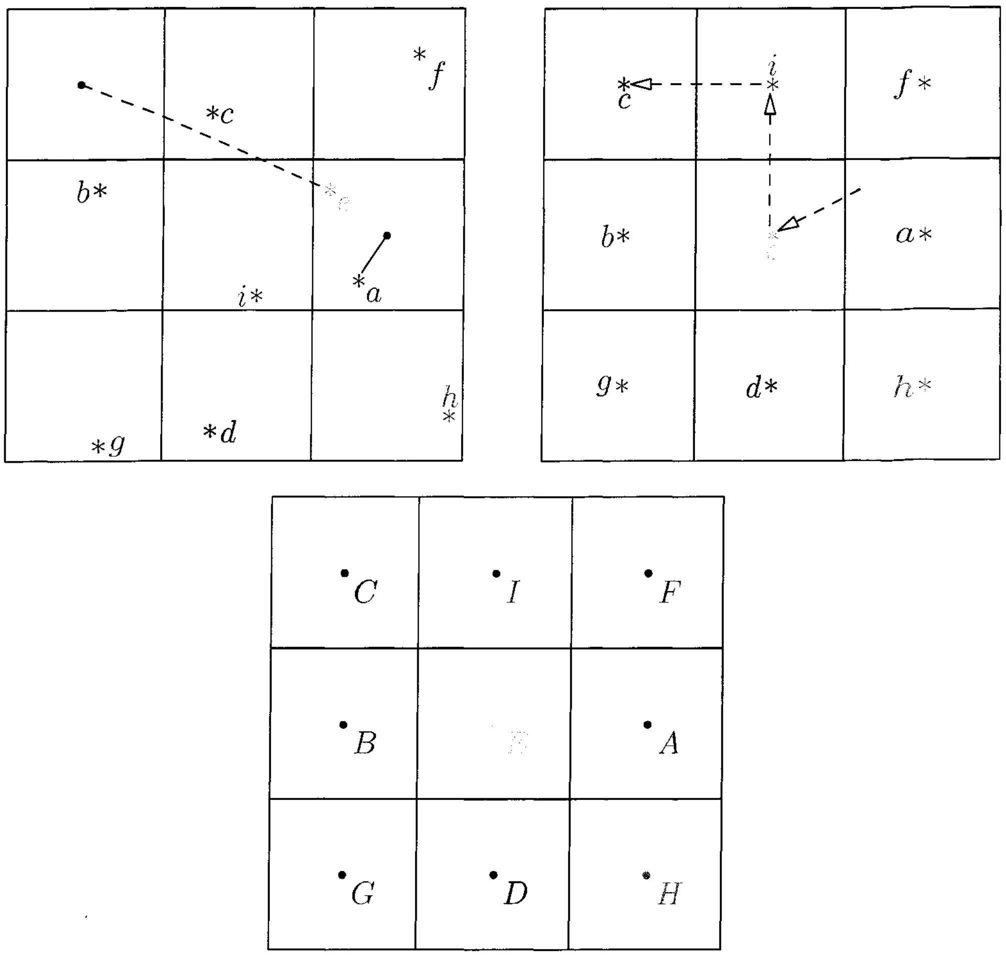


Figure 2.4: Rearrangement step and final position.

parcels e and a were in a pile, the nearest hole, and a discretized path to it, are found for parcel e. Now parcel e is pushed to the first cell in the path, pushing the next parcel, i, to the next cell in the path, and ultimately, putting parcel, c into the hole.

2.A.1 The Bresenham line-drawing algorithm

To find a discretized path from the extra parcel in the pile to the hole, we considered the *Bresenham* algorithm for drawing digitized lines [Bresenham 1965] on an integer lattice. One would like to discretize a line from point A to point B. For simplicity, suppose the point B lies within the *first octant* of a two-dimensional Cartesian coordinate system, with A at the origin, so that B lies above the x-axis and below the line $y = x$. The slope m of the line from A to B then satisfies $0 < m < 1$. As illustrated in Fig. 2.5, point B has greater x and y coordinates than A. After including the discrete point (x, y) in the path, one increments the x coordinate. To find the next point to include, one decides whether or not the y coordinate should be changed as well. This decision is made by comparing the vertical distance of the line to the grid points $(x+1, y)$ and $(x+1, y+1)$. Let ϵ be the vertical distance from the current discrete (x, y) coordinate to the line, which will be the difference of the vertical coordinate on the line and y, always satisfying the inequality $-0.5 \leq \epsilon < 0.5$. In the next step this distance will be incremented by m. One includes $(x + 1, y)$ if $\epsilon + m < 0.5$ and $(x + 1, y + 1)$ otherwise.

Now add m to ϵ and continue on toward B in the same manner, until B is reached. A discretized line from A to B, indicated by the blue dots in Fig. 2.5, is thus obtained. A few simply modifications of this algorithm allow it to be applied to the general case where the final point B lies in any octant (Appendix B).

2.B Problems and modifications

The discretization of lines between holes and piles using Bresenham's approach increases the possibility of a parcel being pushed more than once since we are always pushing parcels in a straight line. Suppose that in one time step, there are clusters of piles concentrated in one area and a group of holes in another

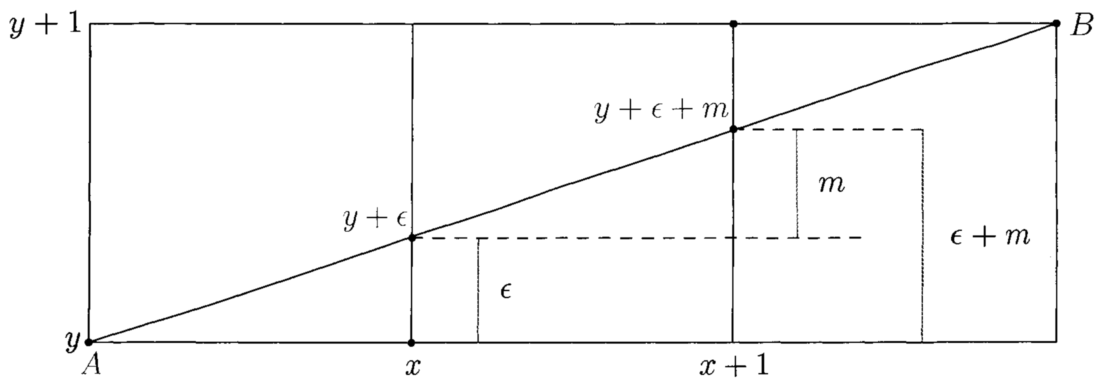


Figure 2.5: The Bresenham algorithm.

nearby area. In pushing a parcel from a pile to 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. 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 (denoted by the real coordinate pair (x, y)) and the center of the cell in which it currently lies. For example, when the parcel is pushed to cell (ℓ, m), then $d^2 = (x - \ell)^2 + (y - m)^2$. 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 the holes and piles. Initially, the parcel with the largest d in the starting cell (pile) is chosen to be pushed since it is furthest from the center of that cell. However, in the path between the pile and the hole, the 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.

2.B.1 A weighted Bresenham algorithm

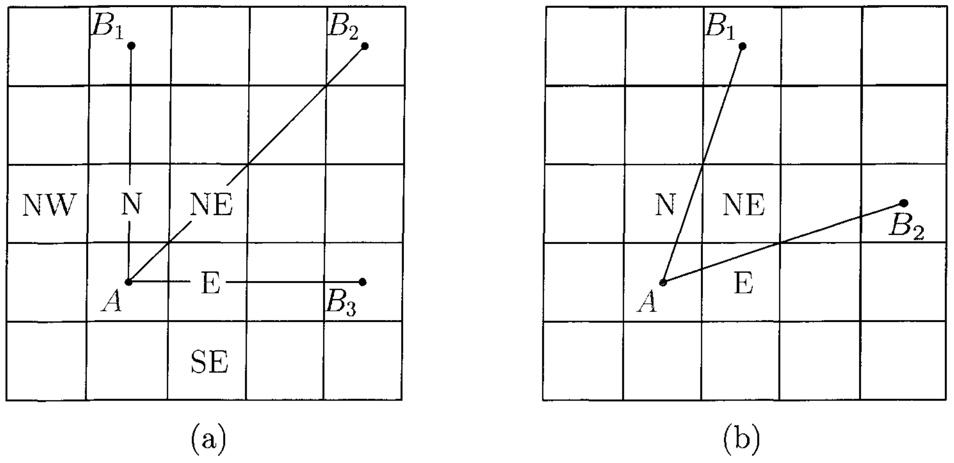


Figure 2.6: Two or three choices in the weighted algorithm.

In the weighted version of the code (Appendix C), 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. For simplicity, we consider the case where the destination cell, B, lies in the first quadrant with respect to the source cell, A. In each step, if the line connecting A to B passes through the center of a neighbouring cell, we have three choices for the next cell. In Fig. 2.6(a), for B_1 we will choose one of the cells NW (north-west), N (north), or NE (north-east), for B_2 we can choose one of the cells N, NE, or E (east), and for B_3 we have the choice of one of the cells NE, E, or

SE (south-east). Otherwise we have a selection of two cells to choose from. In Fig. 2.6(b), for B_1 we can choose one of the cells N or NE, and for B_2 we will choose one of the cells NE or E. 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 that 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 2.1). 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, accordingly.

One question arises in the case where more than one hole is found in the same rectangular shell around the pile. We resolve such cases by performing the 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. In Fig. 2.7, the number in each cell represent the minimum d for parcels in that cell, and the path, shown by the red connected lines, is the selected final path from pile to hole.

The following theorem establishes that at most $\lceil 1.82\overline{AB} \rceil$ steps will be required by our weighted Bresenham algorithm to draw a line from A to B. This is not excessively more than the $\lceil \sqrt{2}\,\overline{AB} \rceil$ steps required to draw a diagonal Bresenham line on a unit lattice.

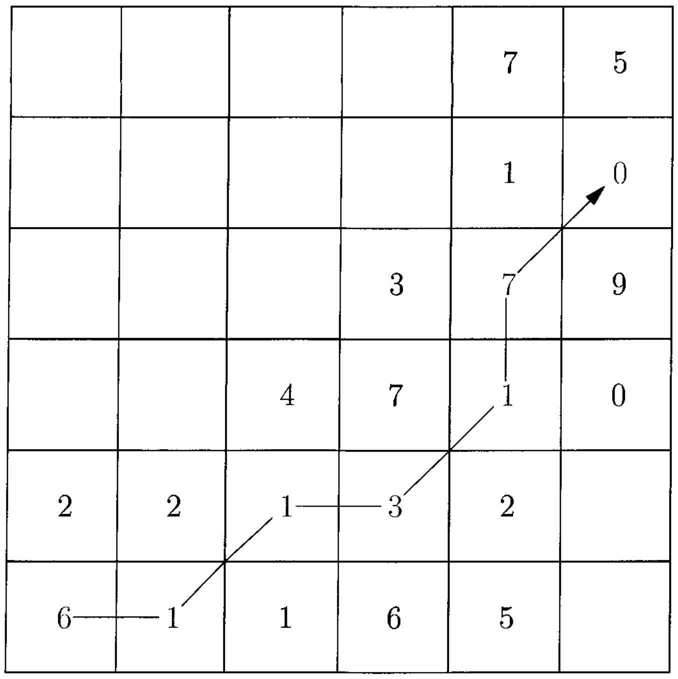


Figure 2.7: A weighted Bresenham path.

Theorem 2.1 (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* $\left\lceil \left(\sqrt{5} - \sqrt{6-\sqrt{10}}\right) x \right\rceil$ *steps are needed to connect two points a distance* x *apart.*

Proof. 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 points to choose the next cell for inclusion 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 δ. The algorithm will then terminate in a finite number of steps.

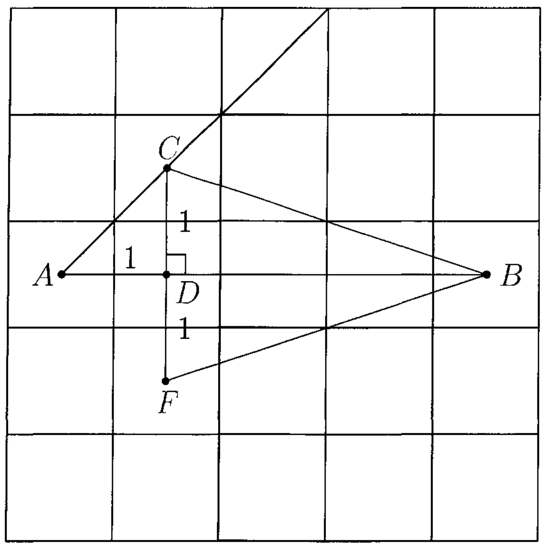


Figure 2.8: Case(i): $m = 0$.

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. 2.8), 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

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

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

$$\delta^2 - 4\delta + 2 \leq 0 \Rightarrow \delta \in \left[2 - \sqrt{2}, 1\right).$$

The same argument of course also holds for the choice F. The distance reduction in this case is thus at least $2 - \sqrt{2}$.

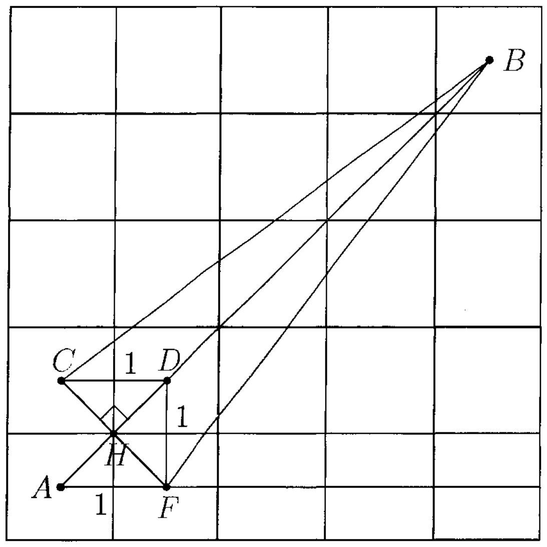


Figure 2.9: Case(ii): $m = 1$.

Case (ii): Assume that the slope of the line from A to B is 1. In this case (Fig. 2.9), 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 find

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

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 $2\sqrt{2} - \sqrt{5}$.

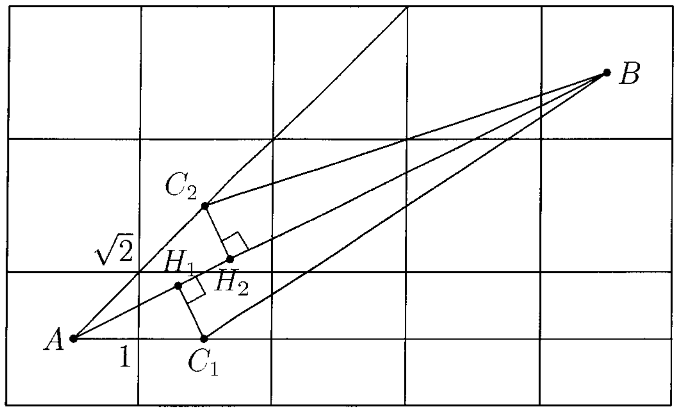


Figure 2.10: Case(iii): $0 < m < 1$.

Case (iii): Assume B lies inside the first octant ($0 < m < 1$). In this case, Fig. 2.10, 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

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

Thus

$$-2x\delta + \delta^2 < -\sqrt{2}xz + z^2,$$

so that

$$z^2 - \delta^2 > x(\sqrt{2}z - 2\delta) \geq \sqrt{5}(\sqrt{2}z - 2\delta).$$

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

So, in any case, the distance between the point B and the new included point is 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.

$\square$

Chapter 3

The General Advection-Diffusion Problem

To solve the general advection-diffusion equation,

$$\frac{\partial \boldsymbol{U}}{\partial t} + \boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{U} = \boldsymbol{D}\nabla^2\boldsymbol{U}, \tag{3.1}$$

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 $\partial\omega/\partial t + \boldsymbol{v}\cdot\boldsymbol{\nabla}\omega = \nu\nabla^2\omega$. The advection term, $\boldsymbol{v}\cdot\boldsymbol{\nabla}\omega$, would use the pushed values of ω to calculate $\boldsymbol{v}$, 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 ω_I is the vorticity field on an Eulerian grid obtained by an area-weighted interpolation of the Lagrangian vorticity field ω (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 $\boldsymbol{v}_I$ from ω 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.

3.A Area-weighted interpolation

We now discuss the scheme for transferring information (interpolating) between the Lagrangian and Eulerian grids. The transfer is done *via* 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), as is often done in particle-in-cell methods.

To project information from a Lagrangian frame to an Eulerian lattice, consider the (ω, C) values $\boldsymbol{p}_i$ of the ith parcel centered about the parcel (see Fig. 3.1). This square will overlap some cells in the grid. For cell j, one calculates the area A_{ij} 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 $\boldsymbol{U}_j$ is calculated as

$$\boldsymbol{U}_j = \frac{\sum_i A_{ij}\boldsymbol{p}_i}{\sum_i A_{ij}}.$$

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.

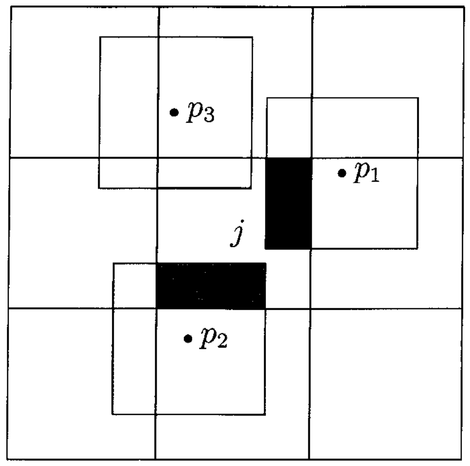


Figure 3.1: Translation from Lagrangian grid to Eulerian grid.

To transform information from the Eulerian to the Lagrangian frame, again consider a parcel and its bounding square (see Fig. 3.2). 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 $\sum_j A_j \boldsymbol{U}_j$, where $\boldsymbol{U}_j$ is the Eulerian value for the jth 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.

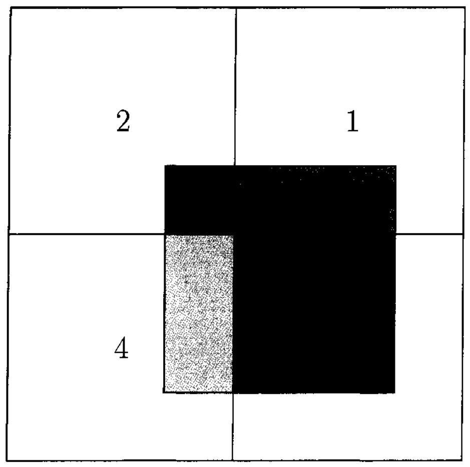


Figure 3.2: Translation from Eulerian grid to Lagrangian grid.

3.B Diffusion

In this section we consider the numerical treatment of the diffusion term in (3.1). The advective term is handled using the Lagrangian algorithm discussed in Chapter 2, and the diffusion term is treated with a *Crank–Nicholson* scheme (e.g., see Ames [1977]) in the Eulerian frame. In general, to use a Crank–Nicholson scheme to solve the equation

$$\frac{\partial \boldsymbol{U}}{\partial t} = \mathscr{S}(\boldsymbol{U}),$$

where $\mathscr{S}$ is an operator, we integrate both side with respect to t and use the trapezoidal rule:

$$\begin{aligned}\int_t^{t+\tau} \frac{\partial \boldsymbol{U}}{\partial t}\, dt &= \int_t^{t+\tau} \mathscr{S}(\boldsymbol{U}) \\ \Rightarrow \quad \boldsymbol{U}(t+\tau) - \boldsymbol{U}(t) &= \frac{\mathscr{S}\boldsymbol{U}(t+\tau) + \mathscr{S}\boldsymbol{U}(t)}{2}\tau \\ \Rightarrow \quad \frac{\boldsymbol{U}(t+\tau) - \boldsymbol{U}(t)}{\tau} &= \frac{\mathscr{S}\boldsymbol{U}(t+\tau) + \mathscr{S}\boldsymbol{U}(t)}{2}.\end{aligned}$$

To incorporate diffusion into our algorithm, it is helpful to split (3.1) into two pieces, one due to advection and one due to diffusion, using a procedure known as *operator splitting.* One then regards the solution as function of two different times t_1 and t_2; that is, $\boldsymbol{U}(t) = \boldsymbol{U}(t_1, t_2)$, where

$$\begin{cases} \dfrac{\partial \boldsymbol{U}}{\partial t_1} = -\boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{U}, \\ \dfrac{\partial \boldsymbol{U}}{\partial t_2} = \boldsymbol{D}\nabla^2\boldsymbol{U}. \end{cases}$$

The chain rule tells us over a time interval $\Delta t_1 = \Delta t_2 = \Delta t$ that the total change in $\boldsymbol{U}$ is given by

$$\begin{aligned}\Delta\boldsymbol{U} &= \frac{\partial \boldsymbol{U}}{\partial t_1}\Delta t_1 + \frac{\partial \boldsymbol{U}}{\partial t_2}\Delta t_2 \\ &= \left(\frac{\partial \boldsymbol{U}}{\partial t_1} + \frac{\partial \boldsymbol{U}}{\partial t_2}\right)\Delta t \\ &= (-\boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{U} + \boldsymbol{D}\nabla^2\boldsymbol{U})\Delta t.\end{aligned}$$

This suggests that one could deal with each term separately and combine the result. However, the above formulation does not use the most up-to-date value for $\boldsymbol{U}$ in the finite differencing of the diffusion equation.

Motivated by the above considerations, let $\widetilde{\boldsymbol{U}}$ be the Eulerian projection of the Lagrangian solution to the advection equation $\partial\widetilde{\boldsymbol{U}}/\partial t = -\boldsymbol{v}\cdot\boldsymbol{\nabla}\widetilde{\boldsymbol{U}}$ using area-weighted interpolation. Then, to solve for $\boldsymbol{U}$, including the effects of diffusion, we use the temporal finite-difference formula

$$\frac{\boldsymbol{U}-\widetilde{\boldsymbol{U}}}{\tau} = \boldsymbol{D}\nabla^2\left(\frac{\boldsymbol{U}+\widetilde{\boldsymbol{U}}}{2}\right).$$

In this formulation, one is now using the most up-to-date (i.e. already advected) value, $\widetilde{\boldsymbol{U}}$, as the starting value to calculate the diffused value $\boldsymbol{U}$. This implicit equation can be rewritten

$$\mathcal{L}(-\tau)\boldsymbol{U} = \mathcal{L}(\tau)\widetilde{\boldsymbol{U}},$$

where $\mathcal{L} = \mathbf{1} + \frac{1}{2}\boldsymbol{D}\tau\nabla^2$. In order to calculate $\boldsymbol{U}$ numerically, one needs to invert the Helmholtz operator $\mathcal{L}$. We accomplish this inversion with an efficient *multigrid* solver (see Appendix D), using a single V-cycle iteration and $\boldsymbol{U}_I$ as the initial guess.

To project the contribution of diffusion on the Lagrangian solution, we compute the difference $\boldsymbol{U} - \widetilde{\boldsymbol{U}}$, 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 $\boldsymbol{U}$ 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.

3.C 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 $\boldsymbol{U}$ itself. To calculate the velocity $\boldsymbol{v}$, it is convenient to adopt the vorticity formulation (cf. Appendix A): using the Euler-projected value of the vorticity $\omega_I = (\boldsymbol{U}_I)_\omega$ on the Eulerian grid determined by area-weighted interpolation, one can compute the stream function ψ 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 (see Appendix D), using the value of the stream function from the previous time step as the initial guess. Once a good approximation to ψ is determined, it is straightforward to calculate the advecting velocity:

$$\boldsymbol{v} = \hat{\boldsymbol{z}} \times \boldsymbol{\nabla} \psi.$$

This velocity is used to evolve both the vorticity and the concentration fields, self-consistently, in the Lagrangian frame.

3.D Summary

The entire self-consistent Eulerian–Lagrangian advection-diffusion algorithm is displayed in Fig. 3.3, in comparison with a conventional semi-Lagrangian scheme. In the red loop, we calculate the new Lagrangian position $\boldsymbol{\xi}(t) = \boldsymbol{\xi}_0 + \int_0^t \boldsymbol{v}(\boldsymbol{\xi}(\tau), \tau)\, d\tau$, increment the time step, and then repeat the procedure. In the blue boxes, to account for the effects of the diffusion term, we interpolate $\boldsymbol{U}$ from the Lagrangian to the Eulerian frame, using the area-weighted interpolation discussed in Sect. 3.A, and then use operator splitting and the Crank–Nicholson method in the Eulerian frame to solve for $\boldsymbol{U}$ from

$$\frac{\boldsymbol{U}-\widetilde{\boldsymbol{U}}}{\tau} = \boldsymbol{D}\nabla^2\left(\frac{\boldsymbol{U}+\widetilde{\boldsymbol{U}}}{2}\right),$$

where $\widetilde{\boldsymbol{U}}$ is the advected solution interpolated onto the Eulerian grid. The diffused solution $\boldsymbol{U} = \mathcal{L}^{-1}(-\tau)\mathcal{L}(\tau)\widetilde{\boldsymbol{U}}$ is the conventional semi-Lagrangian solution (orange output). We then project the difference $\boldsymbol{U}-\widetilde{\boldsymbol{U}}$ 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 $\widetilde{\boldsymbol{U}}$ to calculate the stream function $\psi = \nabla^{-2}\omega$ and thereby the self-advected velocity $\boldsymbol{v} = \hat{\boldsymbol{z}}\times\boldsymbol{\nabla}\psi$. 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.

3.E Applications

Lagrangian rearrangement could be applied to many scientific problems where advection arises, such as electro-osmotic flow, geophysical fluid dynamics (including meteorology, climate change, and hurricanes), thermonuclear fusion in plasmas, mathematical biology, and other fields where advection–diffusion equations play an important role. For example, a theoretical model of electro-osmotic advection characterized by extremely small diffusion rates, which provided the initial motivation for this numerical project, is discussed by Alam & Bowman [2002]. This flow is based on the distribution of charged substances in a fluid that is affected both by an applied electrical potential and electrical forces within the ionized advecting fluid. In this case, the charged fluid has a layer with a high concentration of counter-ions. This layer will be attracted toward the electrode with the opposite sign, when an electric field is applied to the fluid.

The incompressible Navier–Stokes equation (and its inviscid version, the Euler equation; e.g. see LeVeque [1990]) is also used in aerodynamics, to model the air flow around a moving object. For an example of nonlinear scalar hyperbolic conservation laws and the calculation of the air flow see Yee & Harten [1985] and Yoko Takakura [1989]). The advection-diffusion equation also has a significant role in biomedical applications, say in models of blood solutes in vascular lumen, which use the advection-diffusion equation with the blood flow as the advected field. In this model, the assumption is that blood is an incompressible fluid, so the flow is modelled by the Navier–Stokes equation (see Quarteroni *et al.* [2002]).

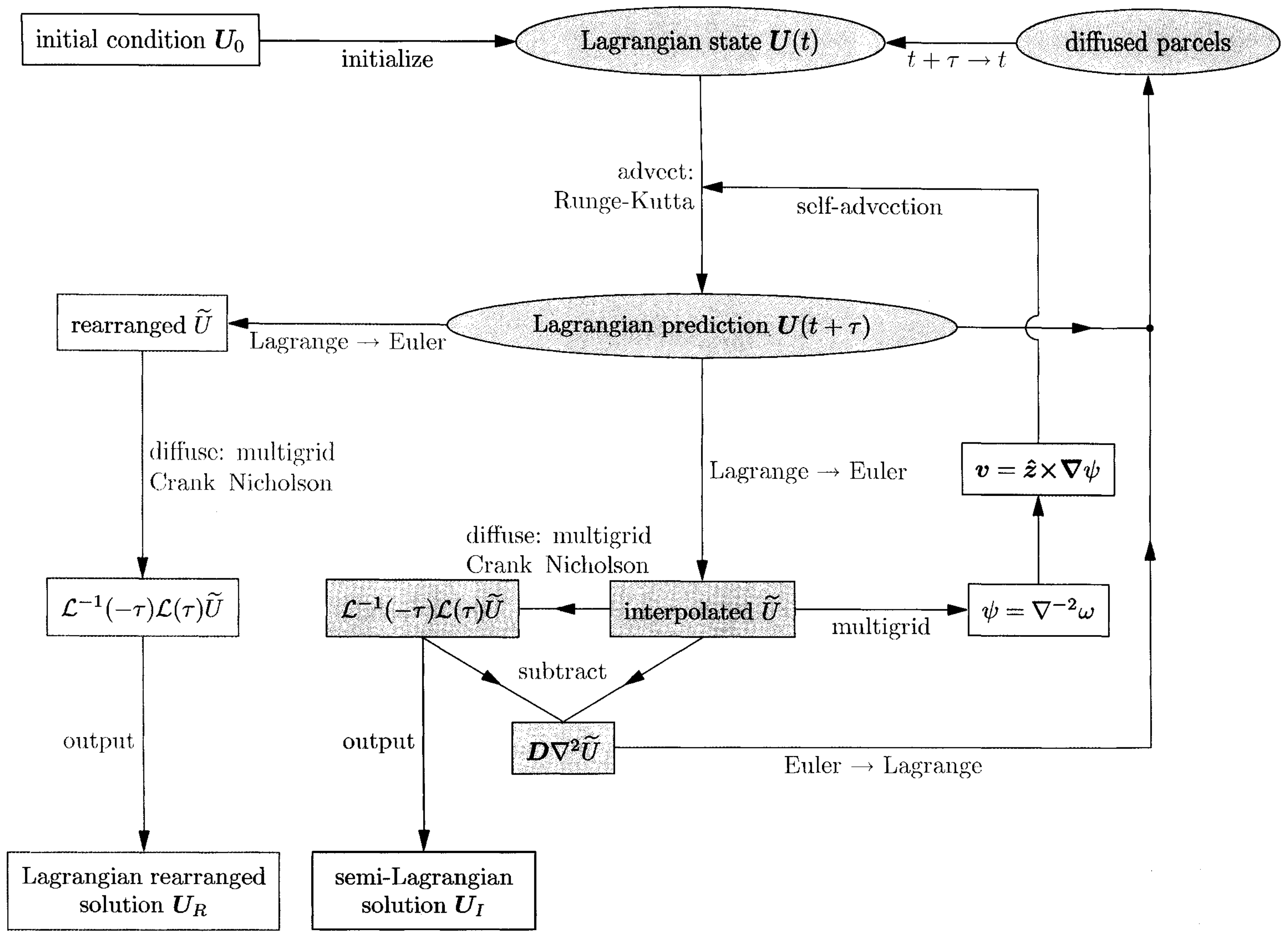

initial condition $\boldsymbol{U}_0$
initialize
Lagrangian state $\boldsymbol{U}(t)$
$t+\tau \to t$
diffused parcels
advect:
Runge-Kutta
self-advection
rearranged $\widetilde{U}$
Lagrange → Euler
Lagrangian prediction $\boldsymbol{U}(t+\tau)$
diffuse: multigrid
Crank Nicholson
Lagrange → Euler
$\boldsymbol{v} = \hat{\boldsymbol{z}} \times \boldsymbol{\nabla}\psi$
diffuse: multigrid
Crank Nicholson
$\mathcal{L}^{-1}(-\tau)\mathcal{L}(\tau)\widetilde{U}$
$\mathcal{L}^{-1}(-\tau)\mathcal{L}(\tau)\widetilde{U}$
interpolated $\widetilde{U}$
multigrid
$\psi = \nabla^{-2}\omega$
subtract
output
output
$\boldsymbol{D}\nabla^2\widetilde{U}$
Euler → Lagrange
Lagrangian rearranged solution $\boldsymbol{U}_R$
semi-Lagrangian solution $\boldsymbol{U}_I$

Figure 3.3: Semi-Lagrangian and Lagrangian rearrangement schemes.

Chapter 4

Analysis

4.A Complexity

A computer code in the C^{++} programing language was used to implement our Lagrangian rearrangement algorithm. We consider a two-dimensional square grid with $n = 2^{2m}$ grid points and doubly periodic boundary conditions, where m is a positive integer. The cost of running the code with respect to time, the *complexity*, will be calculated as a function of n. For the most part of our algorithm, we are performing jobs that run for all parcels, from the first parcel to the last, and then move on to another job (e.g. allocating memory for each cell or initializing the cell to a certain value). In this Chapter, we establish that the computation time for our program scales linearly with respect to n. That is, the complexity of our algorithm is $\mathcal{O}(n)$. The only places that must be focused on is the search for the nearest hole, and the pushing of the parcels, procedures that must be repeated for many of the n cells. As a result, the complexity of the algorithm potentially could be more than of order $\mathcal{O}(n)$. In computing the complexity of these parts of the algorithm we concentrate on the *average complexity*, which computes the cost of an event while taking into account the probability of that event occurring (see Basse & Gelder [2000]). First, assume that the area of the entire grid is of unit size. Then, the area of each cell is $1/n$. If a parcel is randomly assigned to a cell, the probability p of a cell containing that parcel is $1/n$, and the probability q that it does not is $1 - 1/n$. As discussed before, there are exactly n parcels. Using the binomial distribution, the probability of a cell containing k parcels is

$$P(k) \doteq \binom{n}{k} p^k q^{n-k} = \binom{n}{k} \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k}.$$

Therefore, the probability of having a hole is $P(0) = (1 - 1/n)^n$. Notice that,

$$\lim_{n\to\infty} \left(1 - \frac{1}{n}\right)^n = \frac{1}{e}.$$

Thus, for n sufficiently large, the probability of having a hole is approximately $1/e$, and the probability of not having a hole is $1 - 1/e$. For most numerical simulations the domain chosen is very large (in our simulations it is typically 2^{18}); thus, the assumption of having very large n is a safe claim.

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. 4.1, cell A is the pile, which is the center of the search. First, the green ring is searched for a hole, and if the hole is not found, then the search domain extends to the orange region, and so on. Let the *level of the search* k denote the shell number being searched, starting with 1, as shown in Fig. 4.2. Then the number of cells searched so far, up to but not including shell k, will be $(2(k-1)+1)^2 - 1 = 4k(k-1)$,

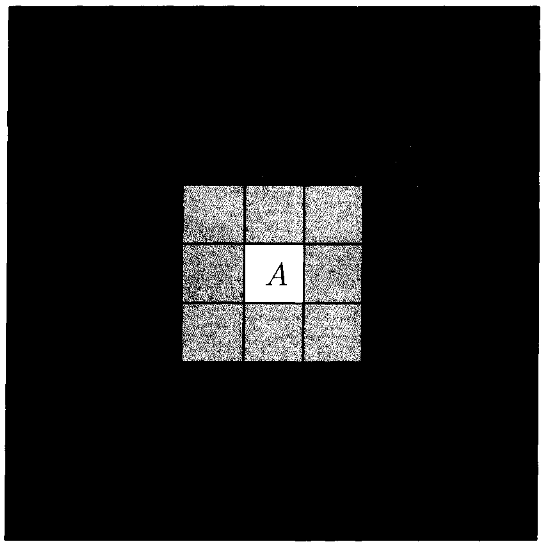


Figure 4.1: Search order.

and the number of cells in the kth shell is $(2k+1)^2 - (2k-1)^2 = 8k$.

The algorithm is going to reach level k if it did not find any holes up to level $k-1$. The probability of not having a hole was calculated to be $1 - 1/e$. Therefore, the probability of not having a hole in the first $k-1$ shells is

$$\left(1 - \frac{1}{e}\right)^{4k(k-1)},$$

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. Hence, the obtained result will be an approximation. Also, we assume that the probabilities of cells being or not being holes are independent of one another. As mentioned before, at shell k, where the algorithm has found a hole, there are $8k$ cells, so 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

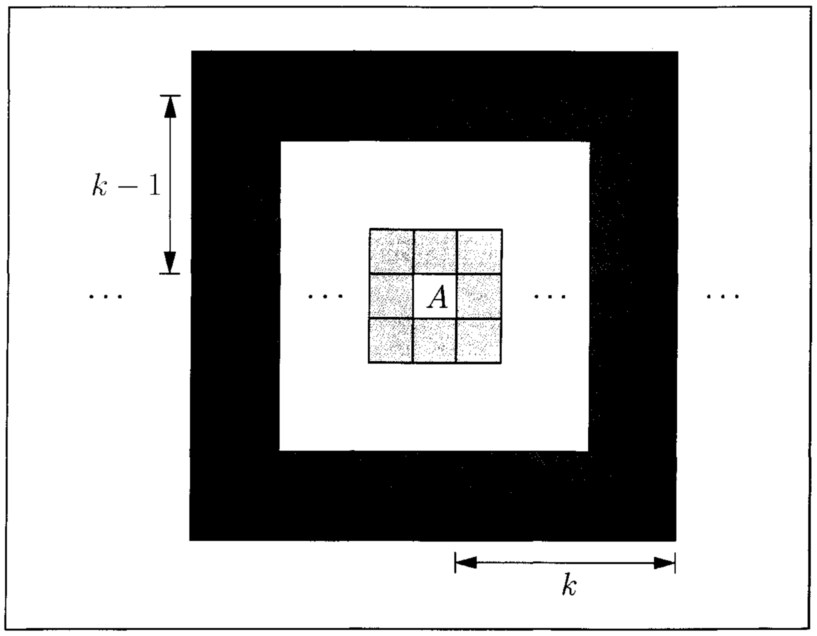


Figure 4.2: Search level.

shell k. Notice that each side of the grid has $\sqrt{n} = 2^m$ cells, so the searching cost $A_s(n)$ is given by

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

As n becomes large, the above cost tends to a small constant, meaning that the average cost of a single search grows insignificantly, as the domain size gets bigger:

$$\lim_{n\to\infty} A_s(n) = 8 \sum_{k=1}^{\infty} k \left(1 - \frac{1}{e}\right)^{4k(k-1)} \approx 8.4,$$

where the sum was evaluated numerically using the symbolic algebra program `Maple` (available from `http://www.maplesoft.com`).

To calculate the cost of our weighted Bresenham algorithm, we will make use of Theorem 2.1. Moreover, if more that 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 kth shell. We determined in Theorem 2.1 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 kth 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 kth shell is $k\sqrt{2}$. Then, the approximate probable distance from a pile to a hole is

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

In appendix E we show, as one would anticipate, that the expected number of holes in the first shell k that contains a hole is given by the conditional probability

$$\frac{8k\left(\frac{1}{e}\right)}{1 - \left(1 - \frac{1}{e}\right)^{8k}}.$$

Thus the *average weighted Bresenham cost* $A_b(n)$ is

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

Therefore,

$$A_b(n) \approx \frac{20.59}{e} \sum_{k=1}^{\sqrt{n}/2} \frac{k^2}{1 - \left(1 - \frac{1}{e}\right)^{8k}} \left(1 - \frac{1}{e}\right)^{4k(k-1)}.$$

Using `Maple`, we found $A_b(n)$ tends to a small constant as the domain gets larger:

$$\lim_{n\to\infty} A_b(n) \approx 8.6.$$

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 as domain gets larger:

$$\lim_{n\to\infty} A_p(n) \approx 2.7.$$

In conclusion, the final cost to search for a hole, identify the weighted Bresenham path, and push parcels along the selected path is $8.4 + 8.6 + 2.7 = 19.7$ iterations. That is, the combined cost of searching and pushing (the yellow branch in Fig. 3.3) is of the order $20n$ (assuming each of the three loops has roughly the same number of machine instructions in it) and will not contribute prohibitively to the overall computation time: the total complexity of the entire Lagrangian algorithm remains $\mathcal{O}(n)$ (that is, it is bounded by a constant times n).

4.B Stability analysis

As mentioned above, Lagrangian rearrangement uses the fact that the Casimir invariants are conserved and the solution at all times will be a rearrangement of the original initial condition. At each time step, the parcels are advected by the local interpolated velocity, using Runge–Kutta integration. If diffusion is added to the equation, the solution will in general no longer be a rearrangement of the initial state since diffusion tends to smooth out gradients in the solution. In this case we use the Crank–Nicholson method, which is unconditionally stable. In the case of self-advection we evolve the vorticity equation, in addition to C, in order to update the advecting velocity field.

Since Lagrangian rearrangement is nothing more than a filter applied to view the semi-Lagrangian solution by projecting it onto the rearrangement manifold, it inherits all of the stability properties of the semi-Lagrangian method.

4.C Comparisons and performance

An important feature of our algorithm 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 times 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 with an initial condition that consists only of the values zero and one, which we display as black and white pixels, respectively. We start evolving this frame self-consistently, (Fig. 4.3), in two different ways, using the same 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

$$\begin{cases} v_x = \sin(2\pi x) \cos(2\pi y), \\ v_y = -\cos(2\pi x) \sin(2\pi y). \end{cases}$$

We used a 512×512 grid, so that the grid scale is $h = 1.95 \times 10^{-3}$. The time-step τ was chosen to be 10 times the Courant condition, or 1.95×10^{-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.

Figures 4.4–4.8 depict snapshots of the same advected stage, where, in each figure, the top frame shows the advected stage for the semi-Lagrangian method, and the bottom frame illustrates the result for the LR method. With our 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.

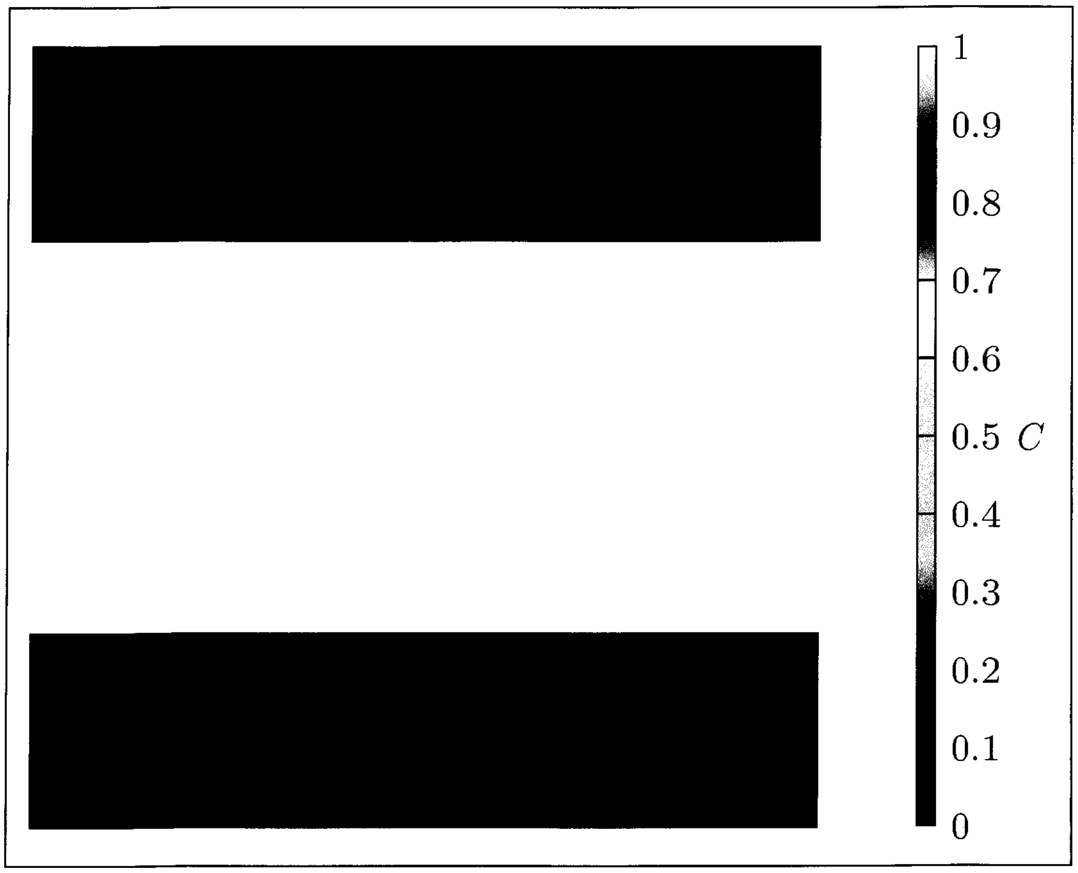
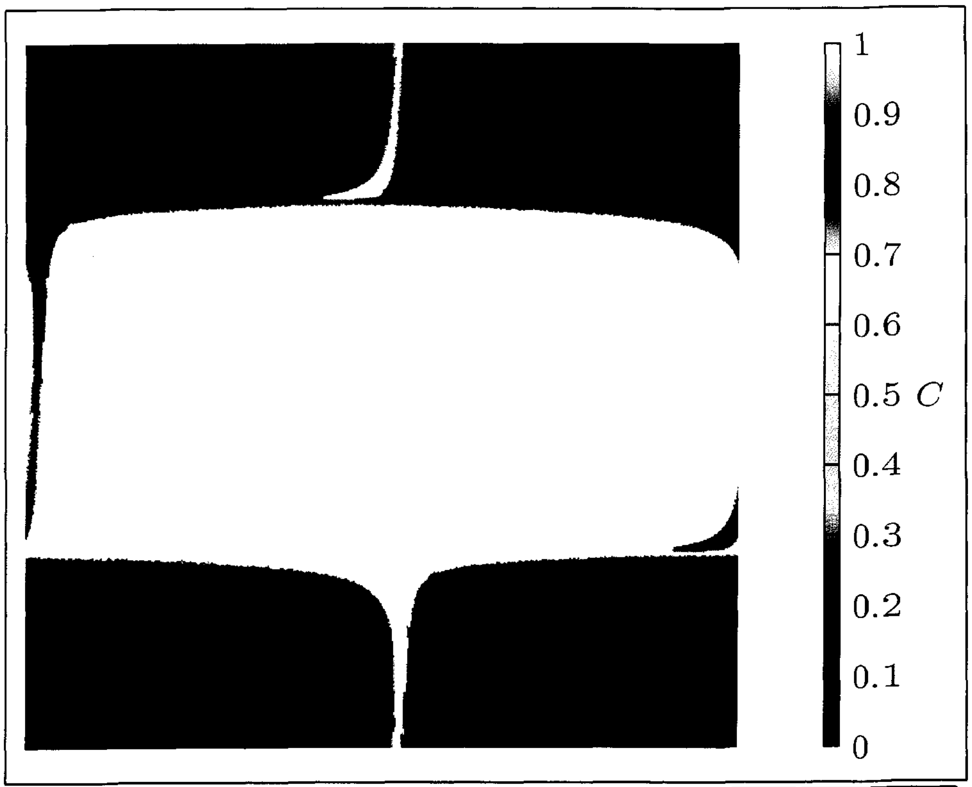


Figure 4.3: Black and white initial condition.



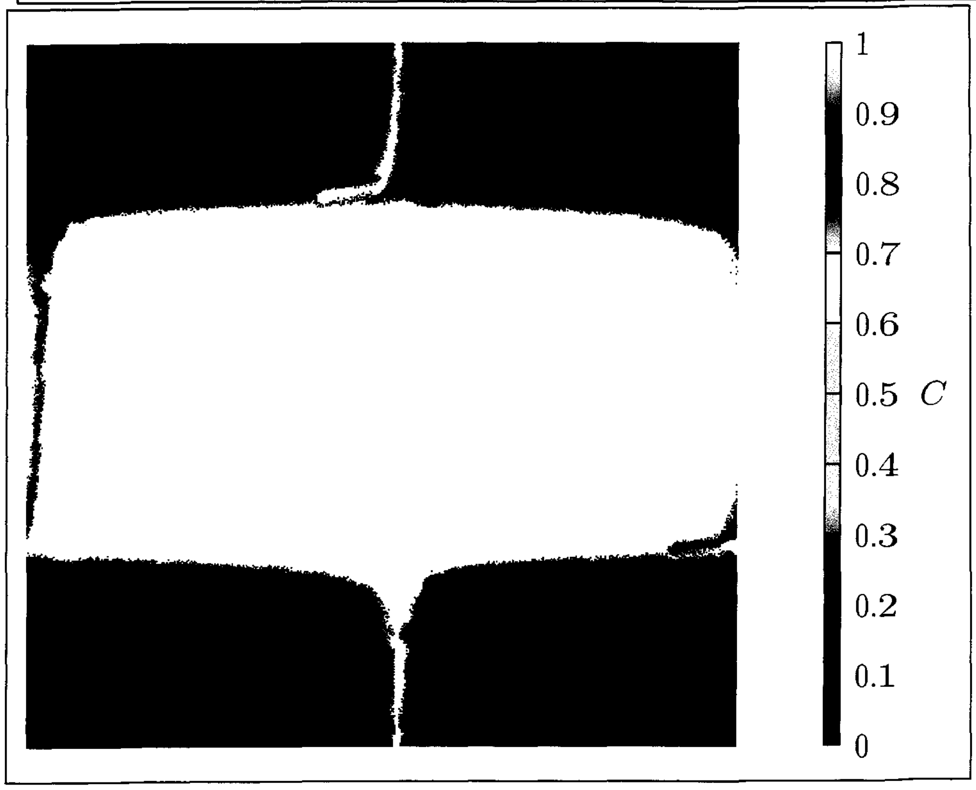
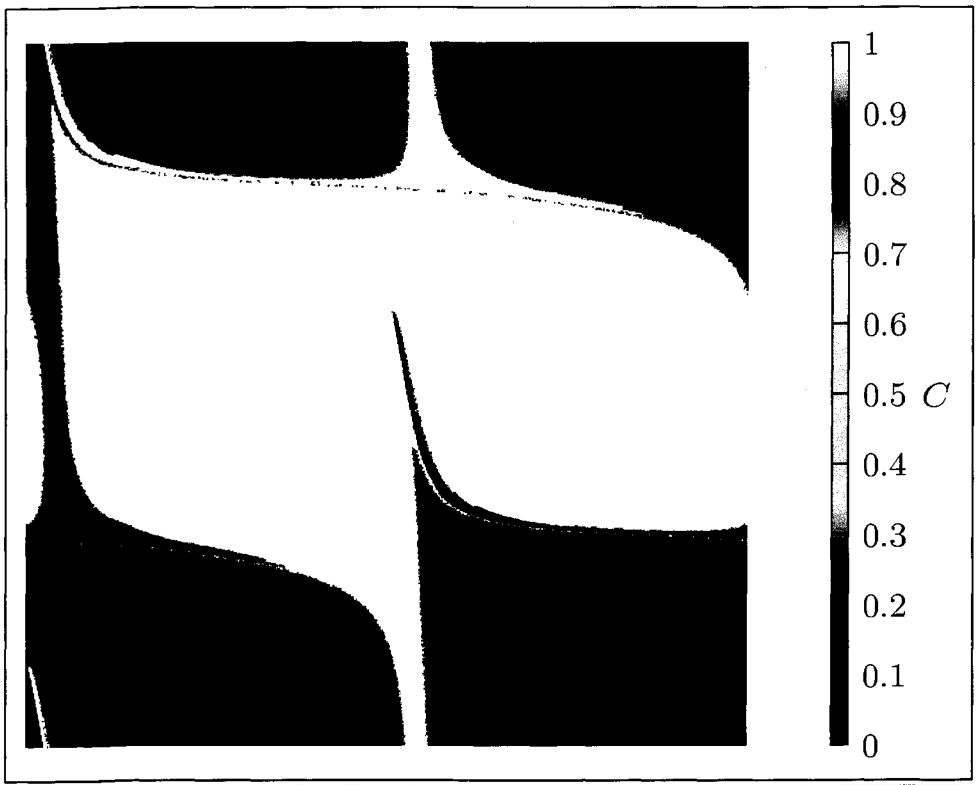


Figure 4.4: Semi-Lagrangian interpolation vs. LR solution after 75 time steps.



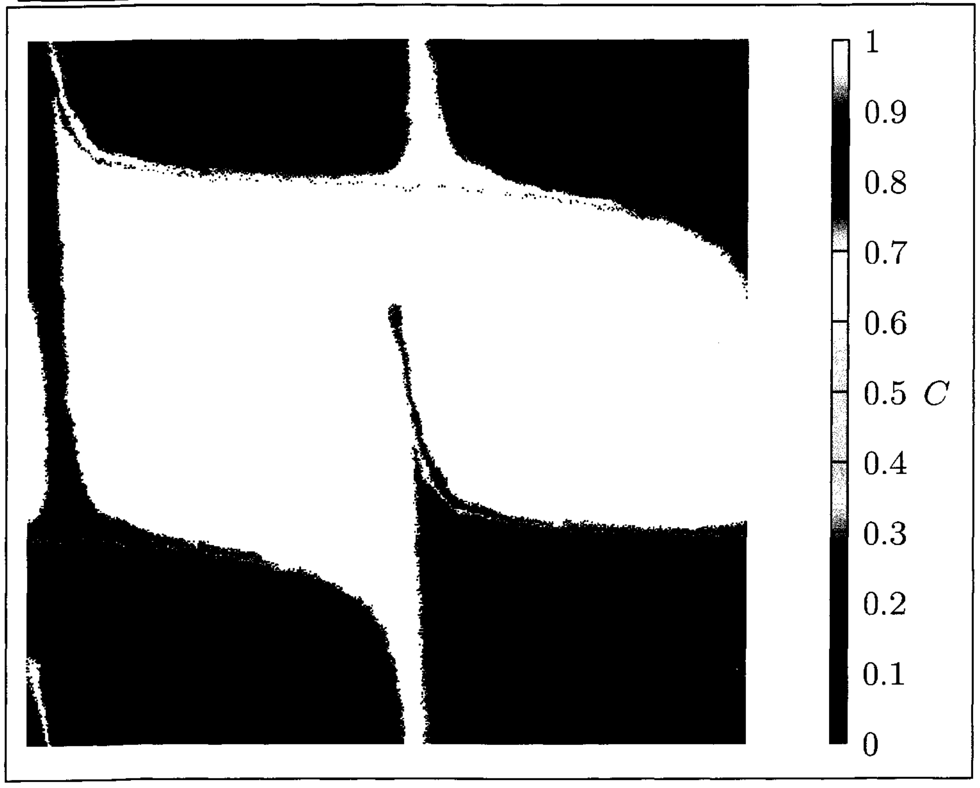
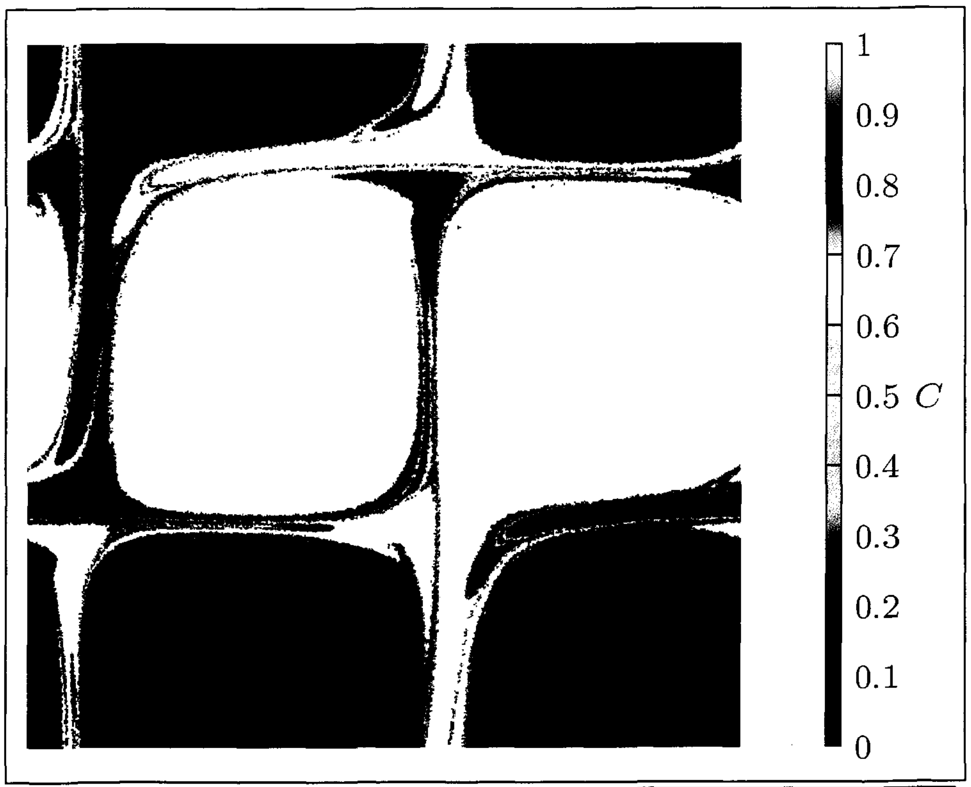


Figure 4.5: Semi-Lagrangian interpolation vs. LR solution after 125 time steps.



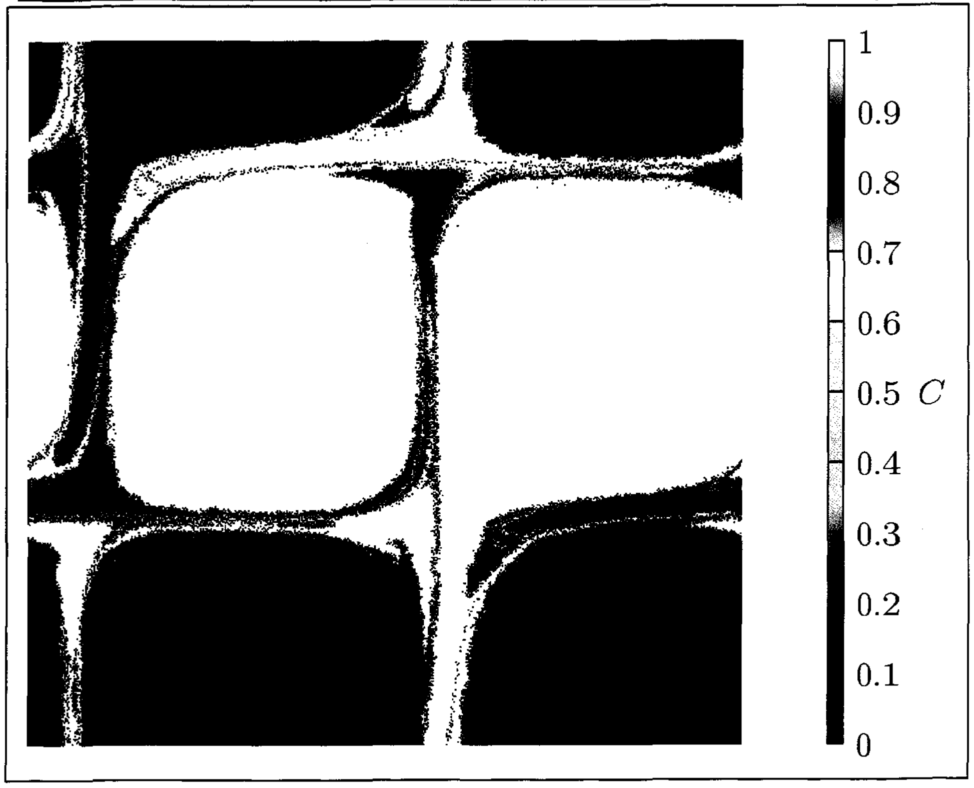


Figure 4.6: Semi-Lagrangian interpolation vs. LR solution after 250 time steps.

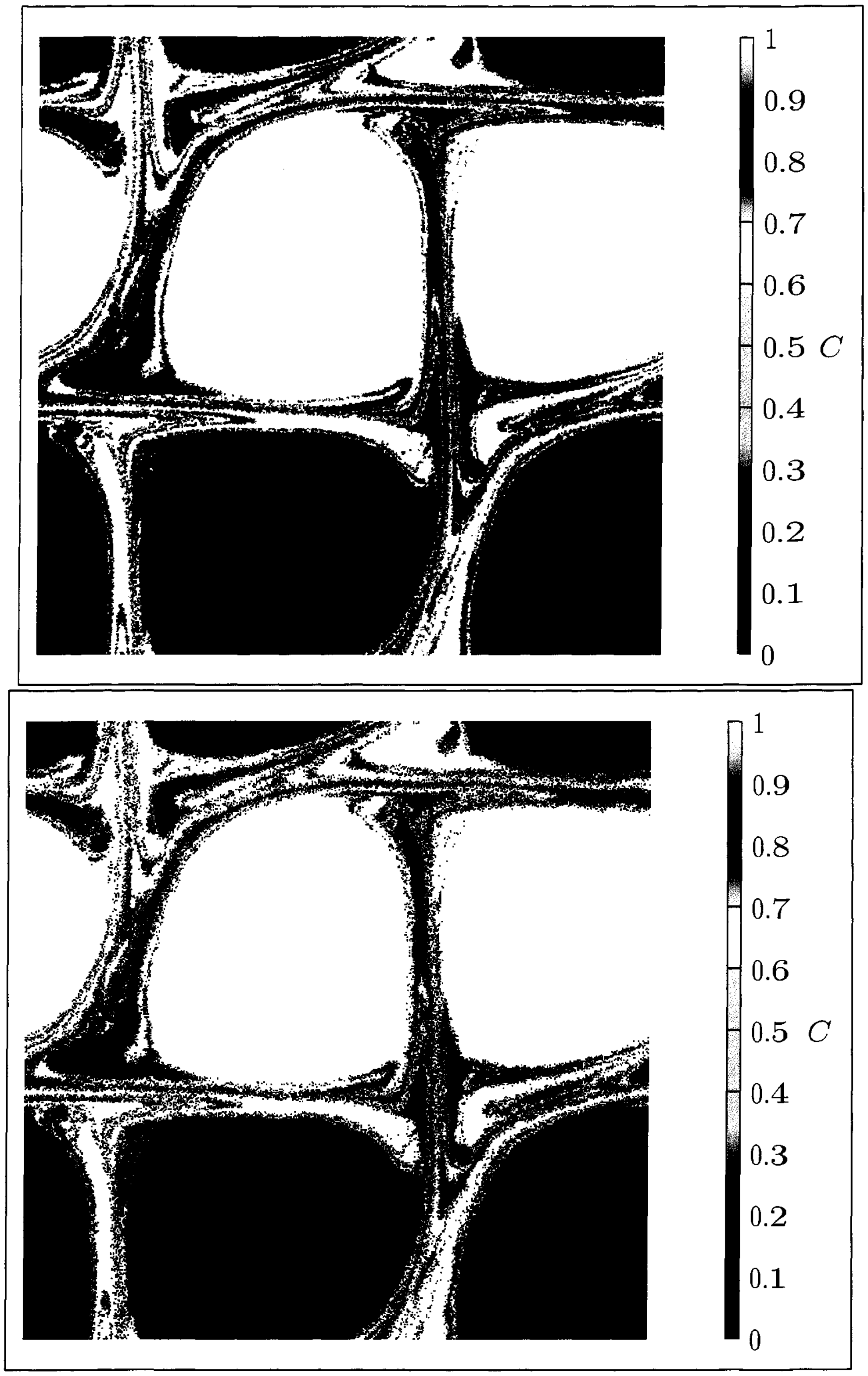
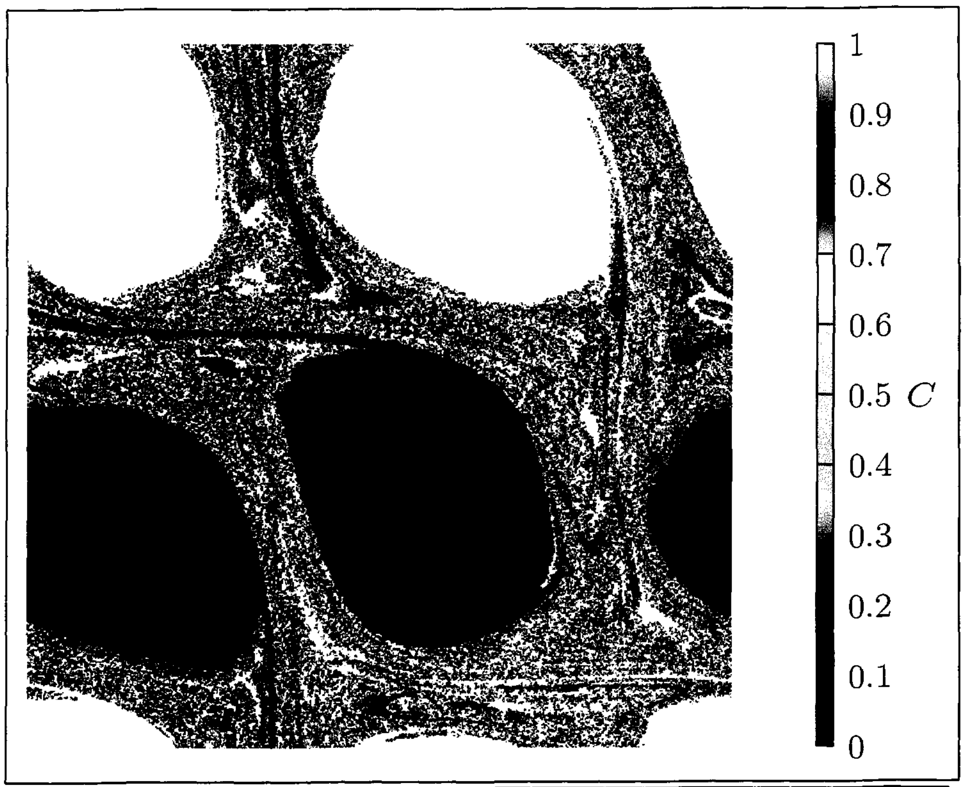


Figure 4.7: Semi-Lagrangian interpolation vs. LR solution after 400 time steps.



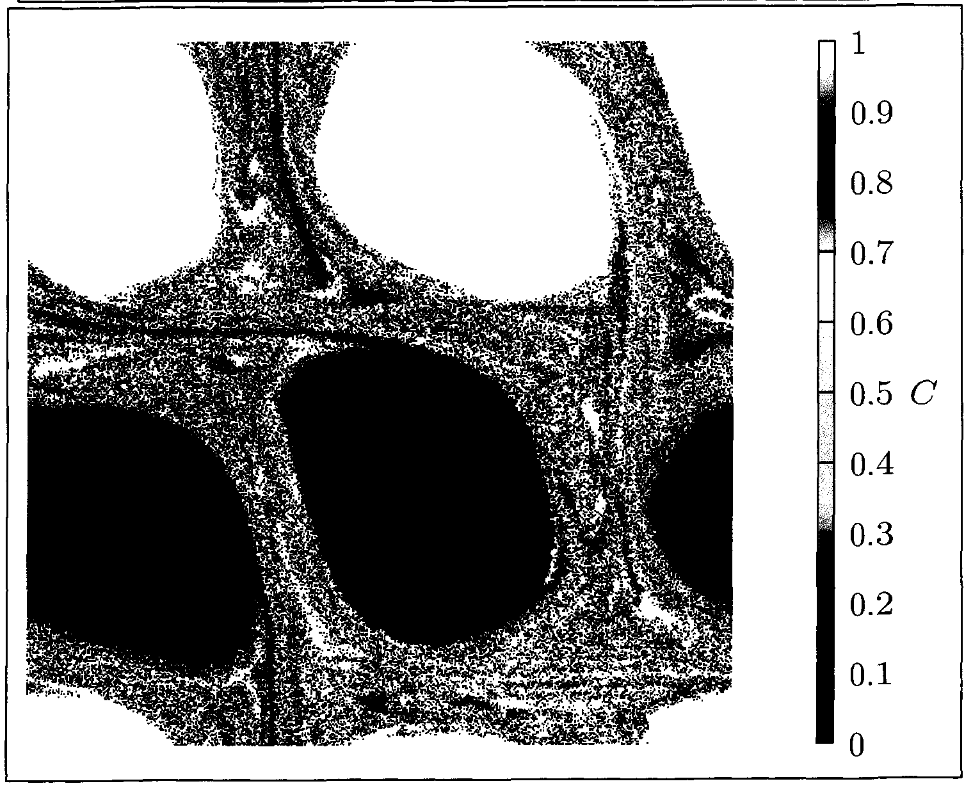


Figure 4.8: Semi-Lagrangian interpolation vs. LR solution after 750 time steps.

Our algorithm can also solve the general 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 of viewing the Lagrangian data against area-weighted interpolated (semi-Lagrangian) projection. Both methods start from the same initial condition. Fig. 4.9 shows the concentration field that will be advected and diffused by the two methods. Figs. 4.10–4.19 demonstrate the advection of this field, where the result of using semi-Lagrangian method is illustrated on the top frame of each figure, and the prediction of the LR method is shown on the bottom frame. Observe that the two methods output nearly identical results. The observed smoothness in the top frame is the result of interpolation by the semi-Lagrangian method, whereas the slight roughness at the pixel level exhibited in the bottom frame is both a consequence of the inherent arbitrariness of our parcel-pushing algorithm and the lack of anomalous numerical diffusion.

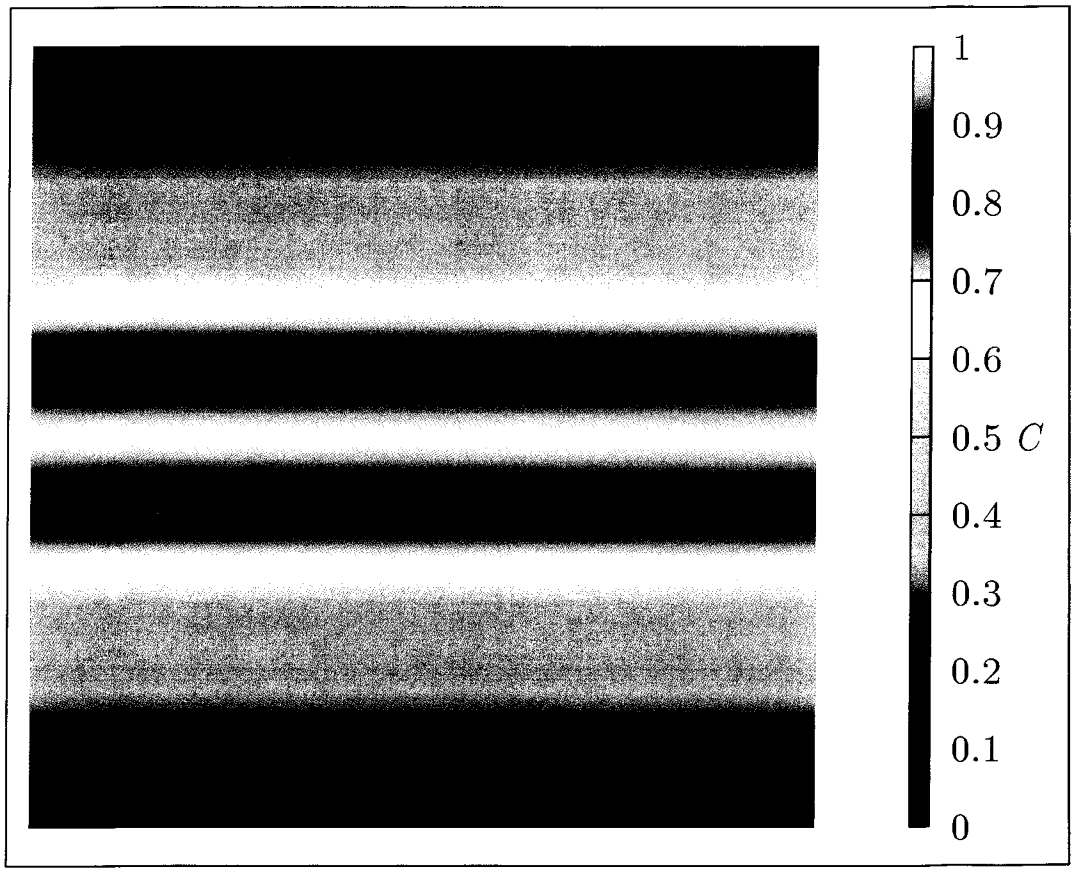
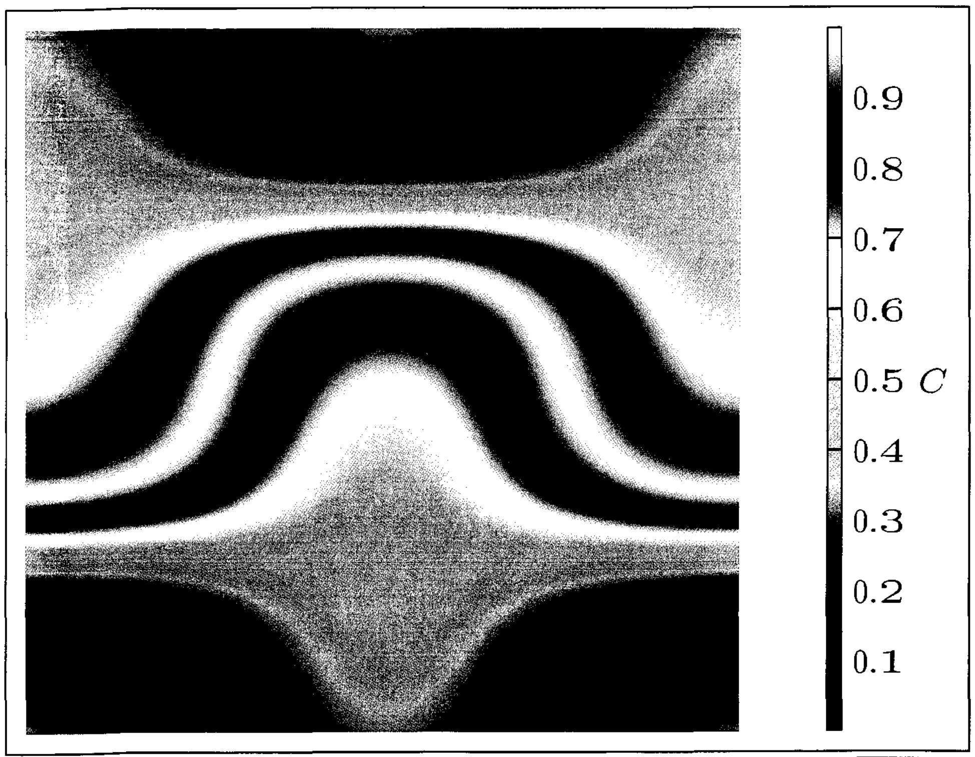


Figure 4.9: Initial condition.



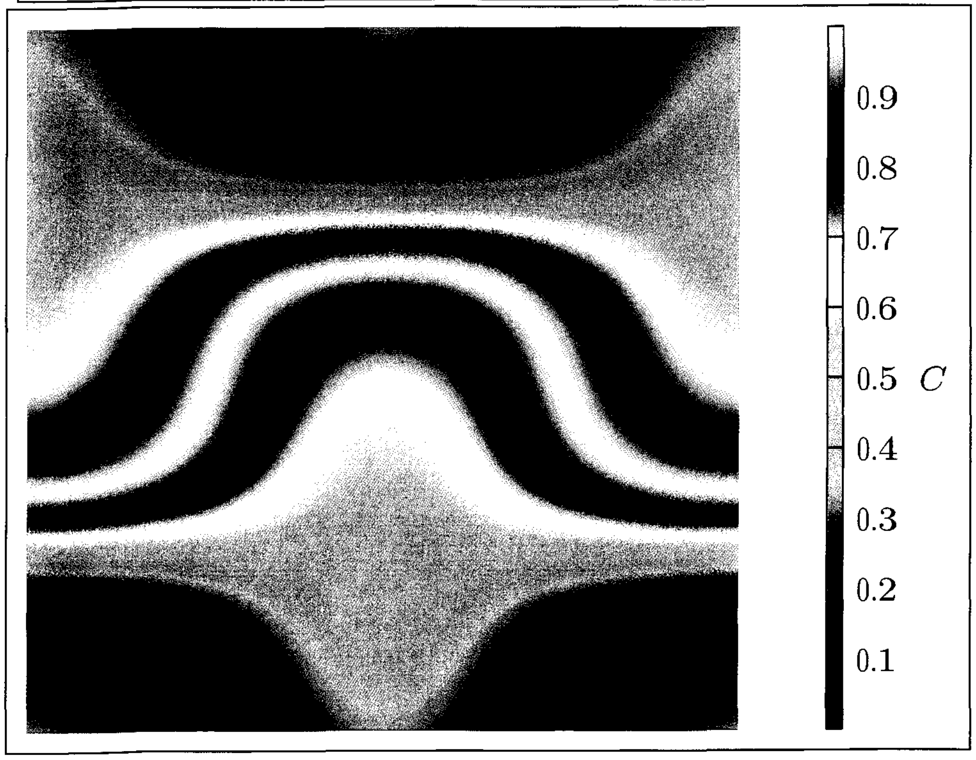
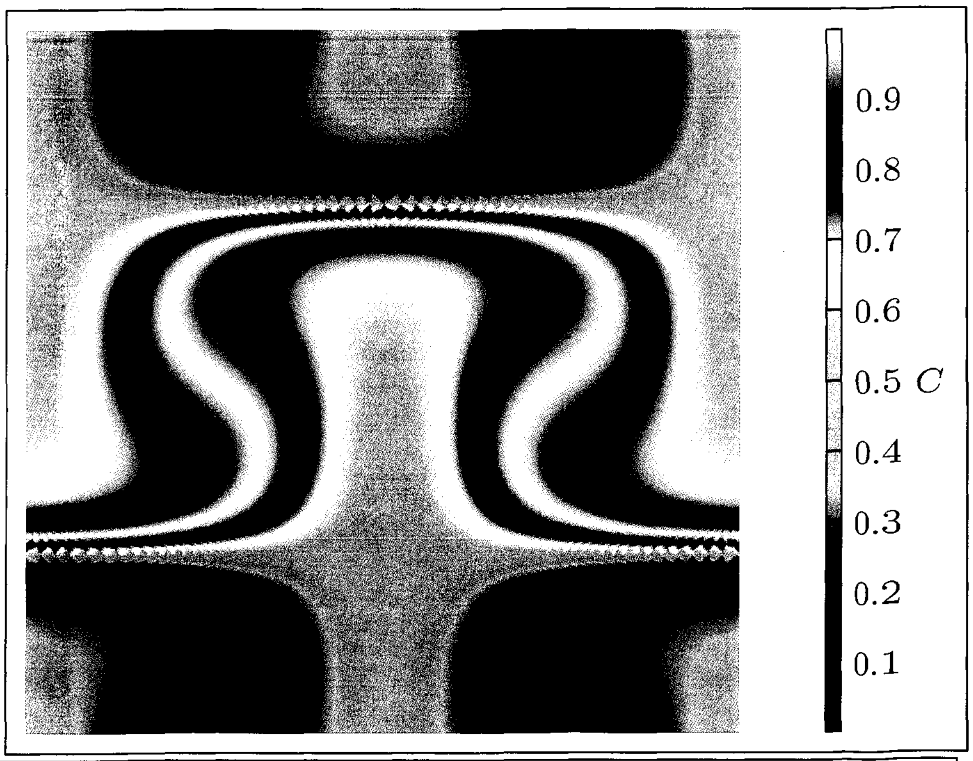


Figure 4.10: Semi-Lagrangian vs. LR after 10 time steps, with diffusion.



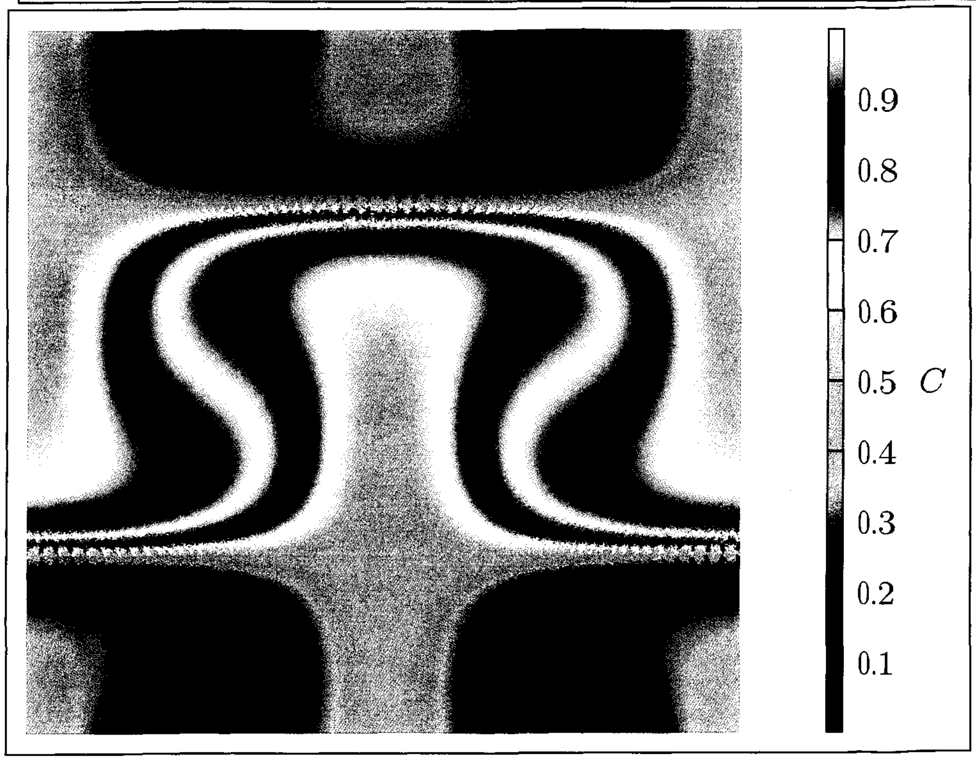
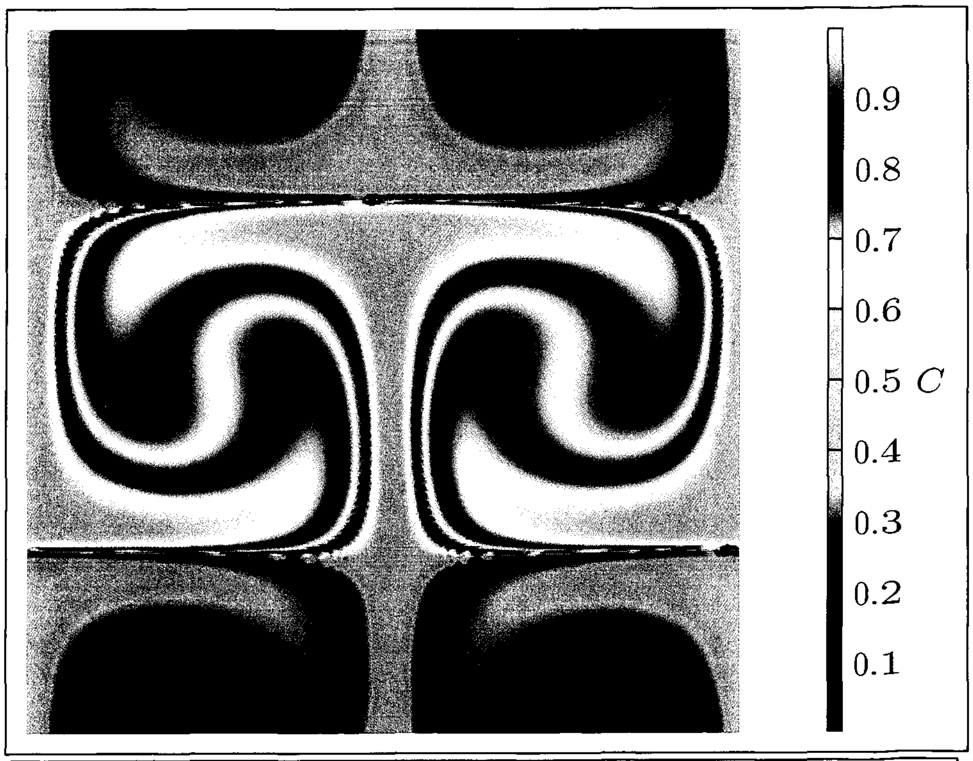


Figure 4.11: Semi-Lagrangian vs. LR after 20 time steps, with diffusion.



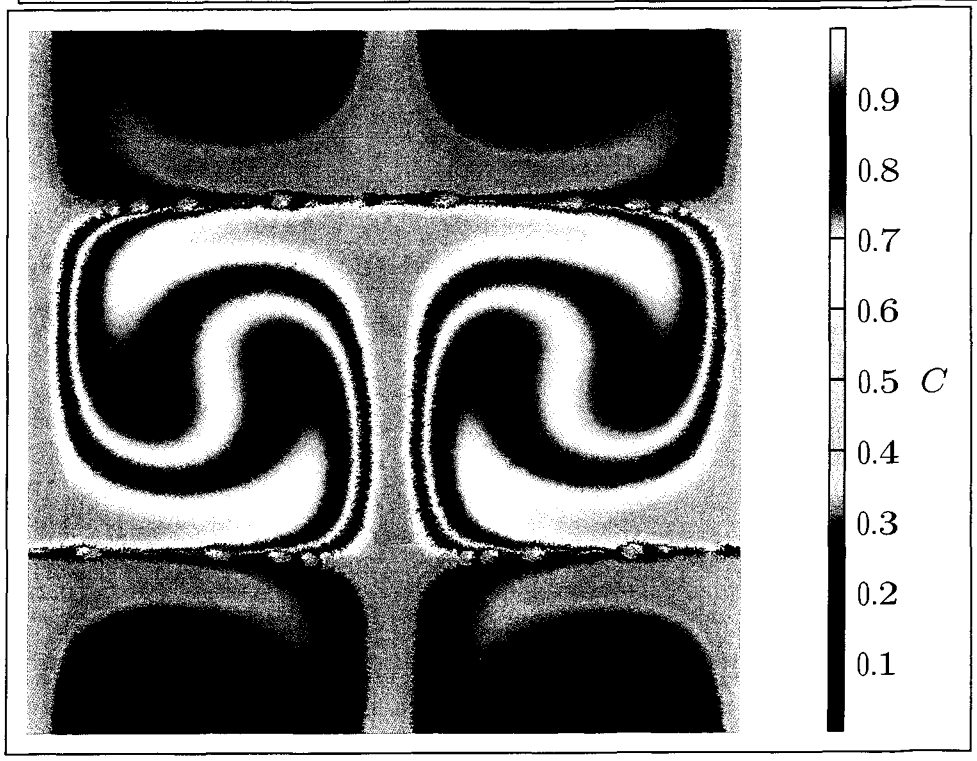
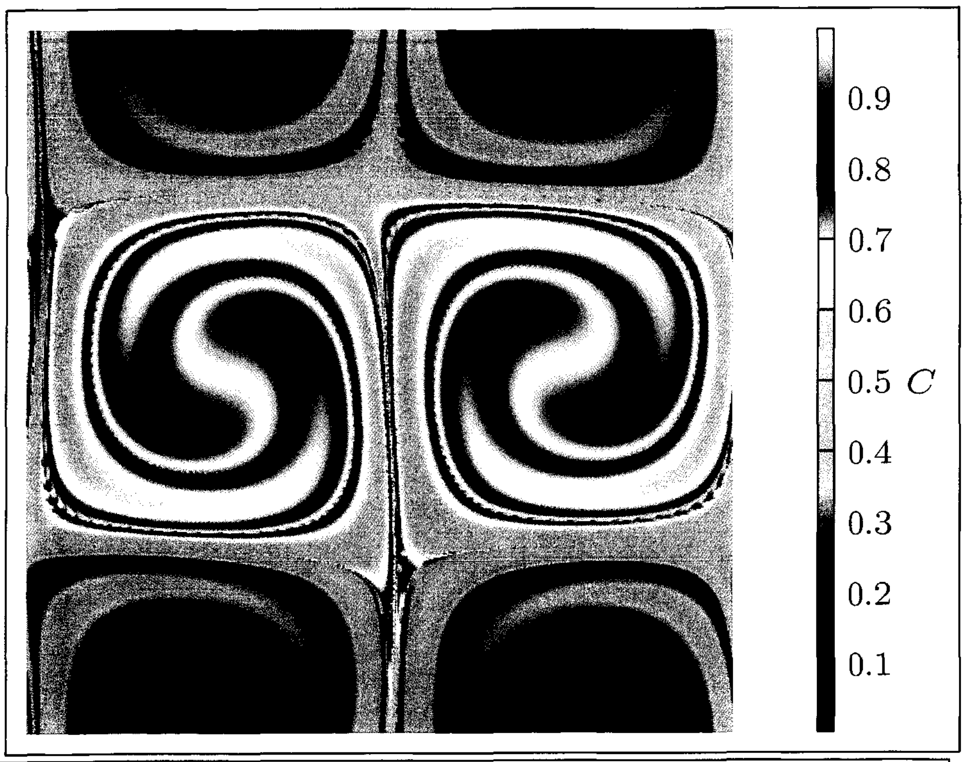


Figure 4.12: Semi-Lagrangian vs. LR after 40 time steps, with diffusion.



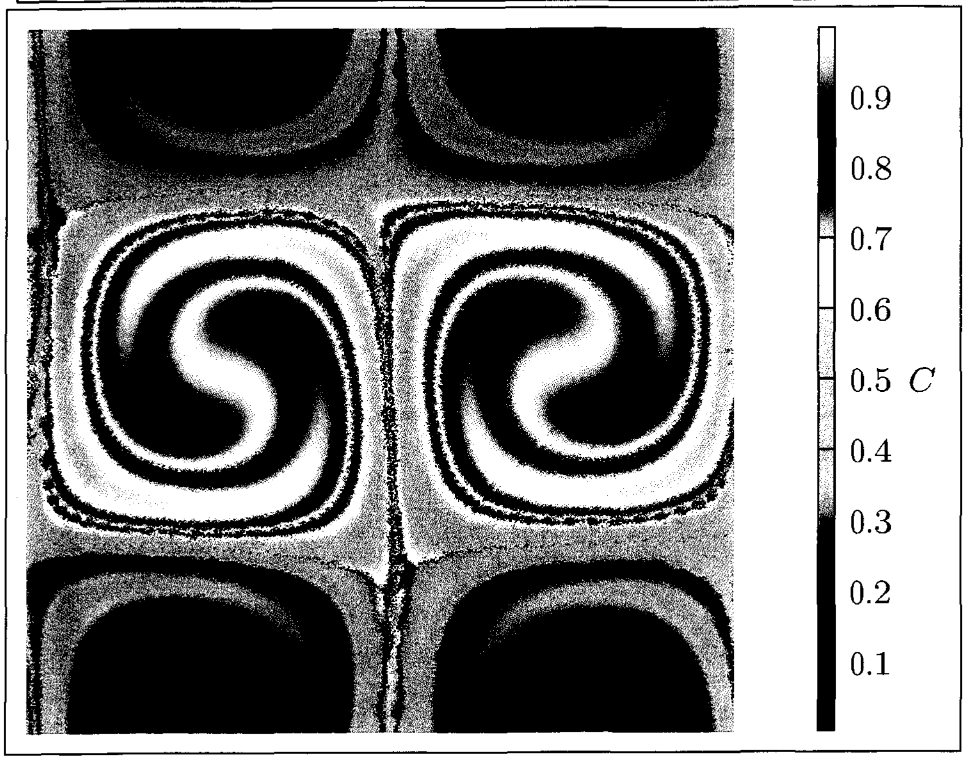
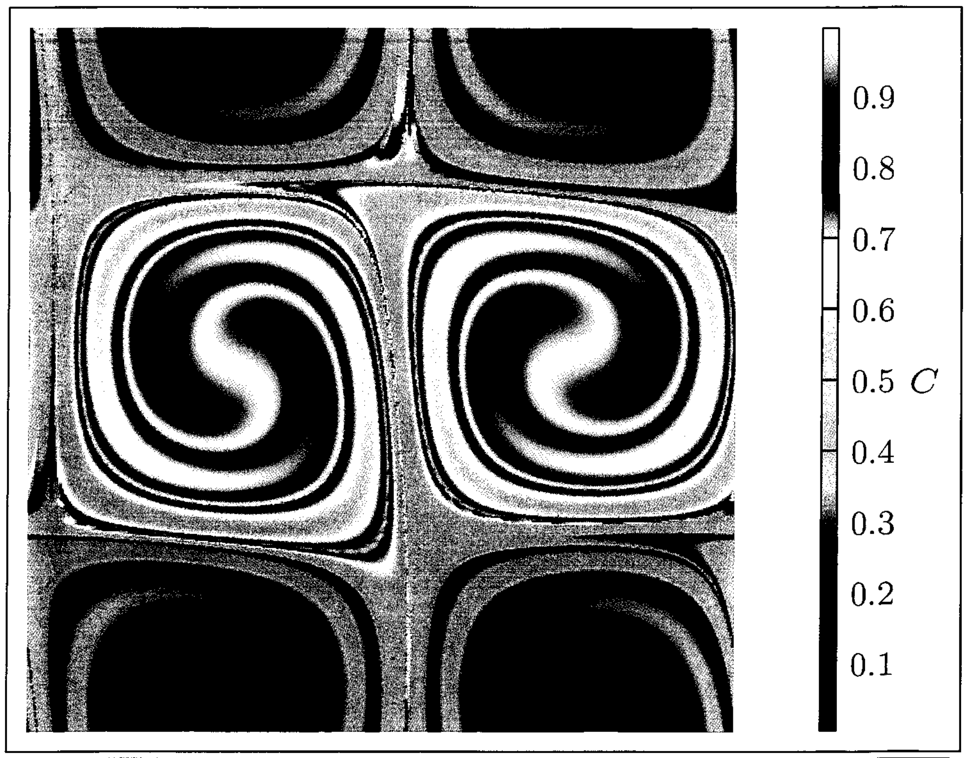


Figure 4.13: Semi-Lagrangian vs. LR after 75 time steps, with diffusion.



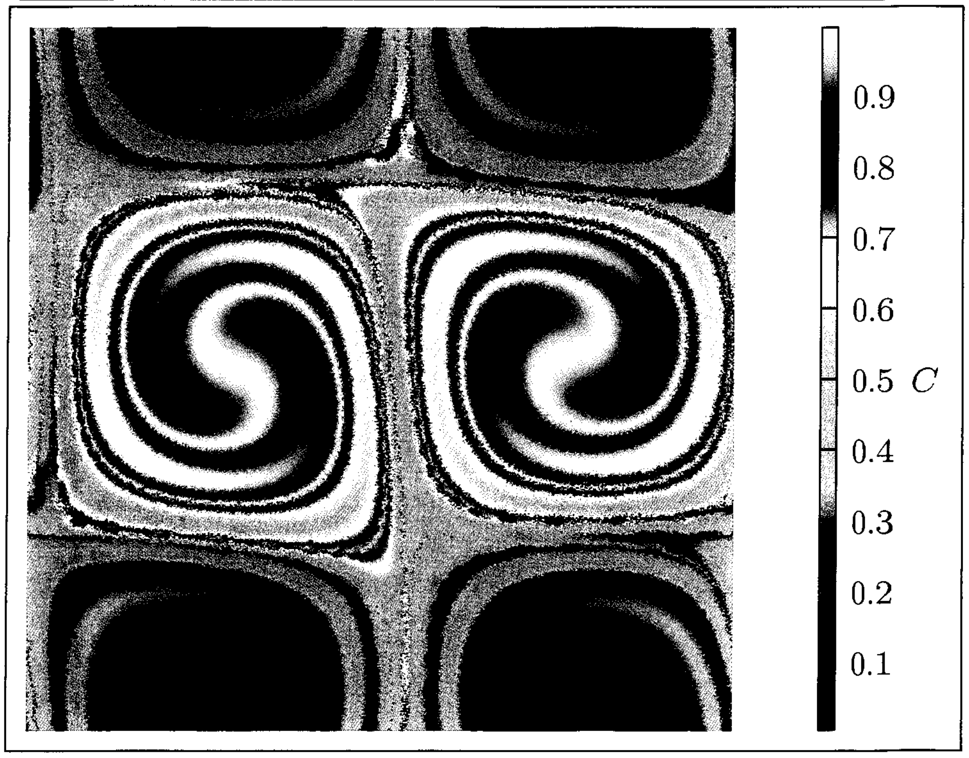
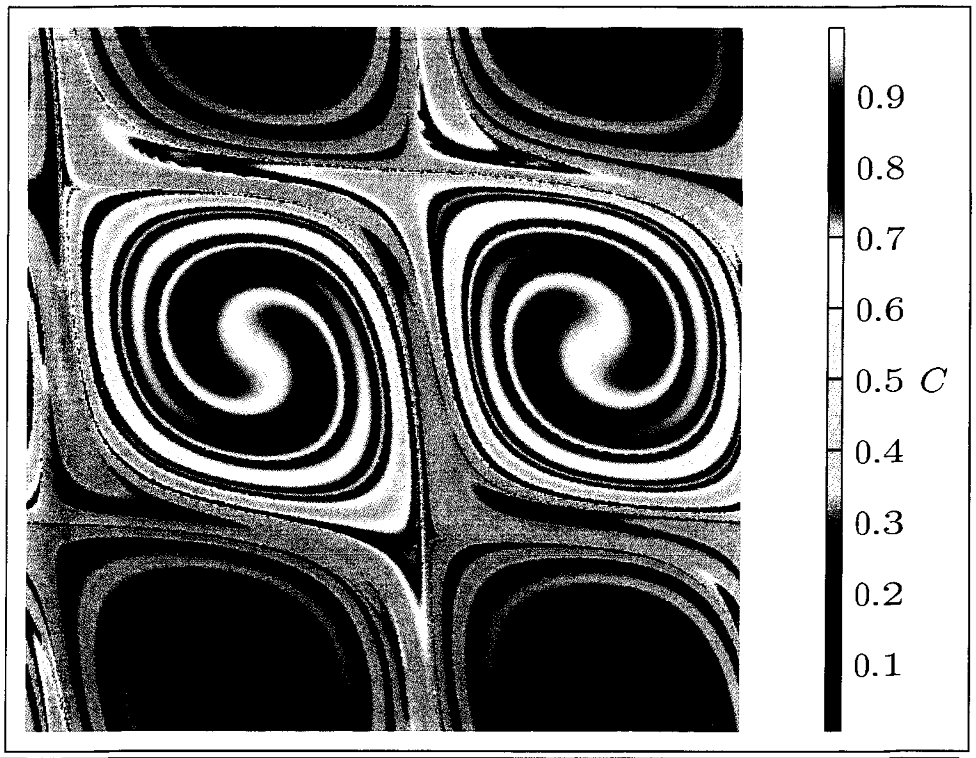


Figure 4.14: Semi-Lagrangian vs. LR after 100 time steps, with diffusion.



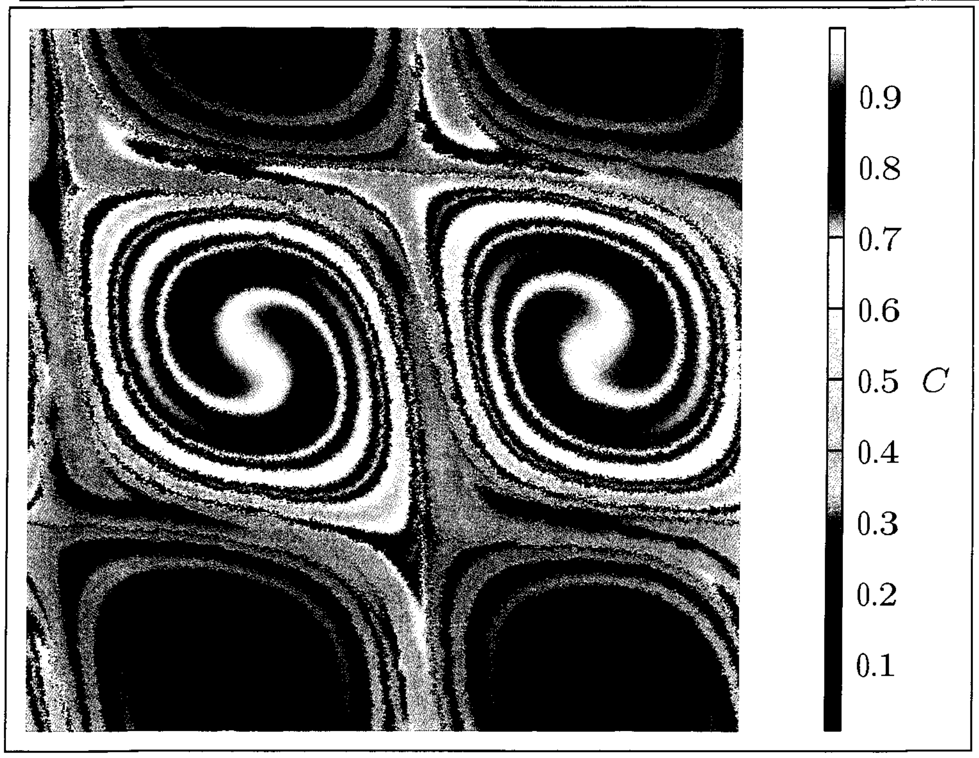
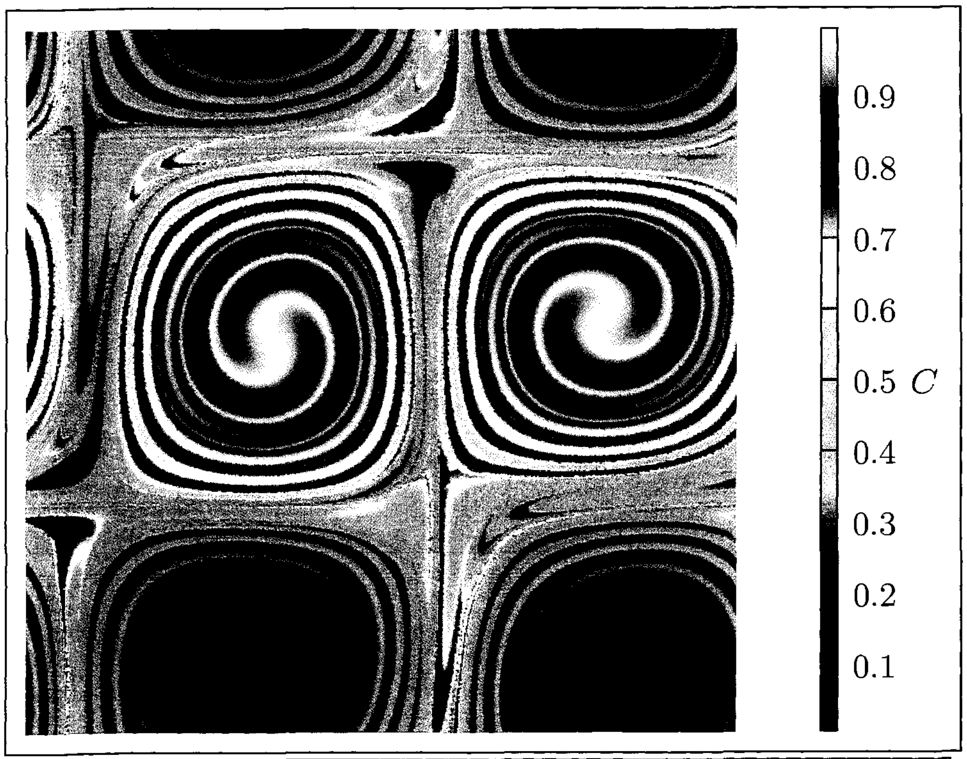


Figure 4.15: Semi-Lagrangian vs. LR after 150 time steps, with diffusion.



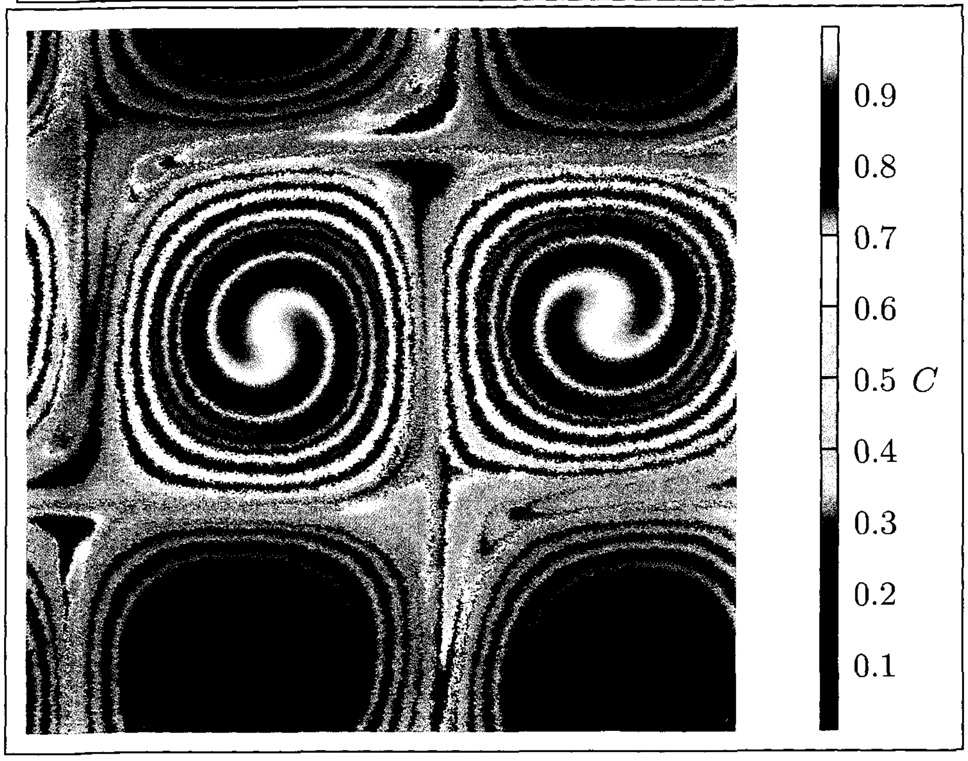
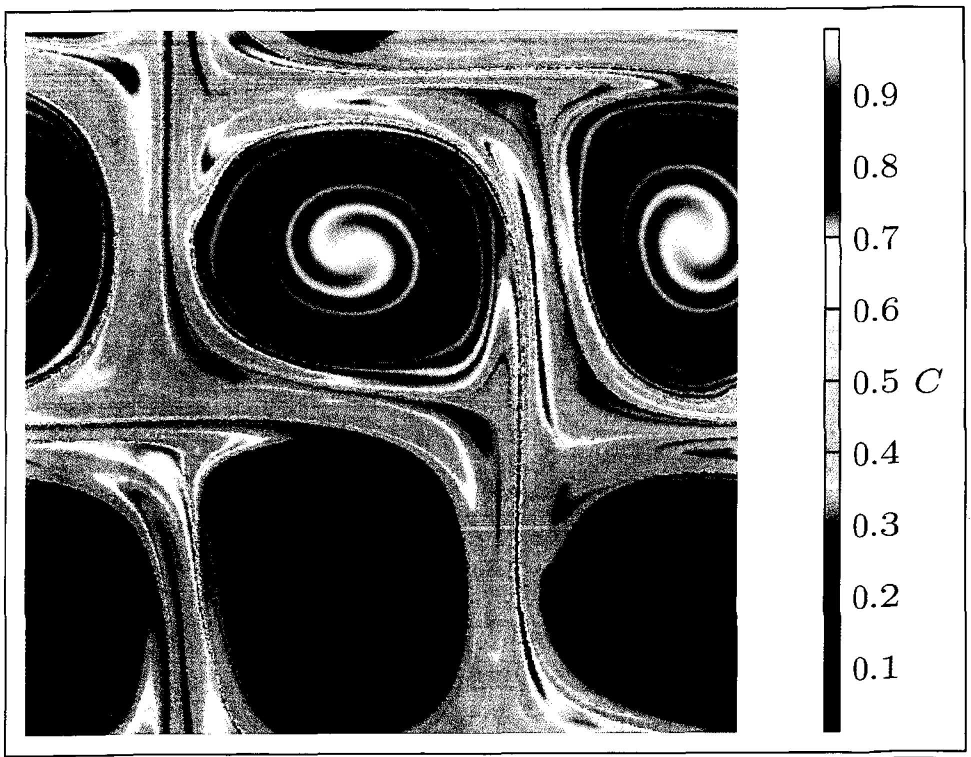


Figure 4.16: Semi-Lagrangian vs. LR after 250 time steps, with diffusion.



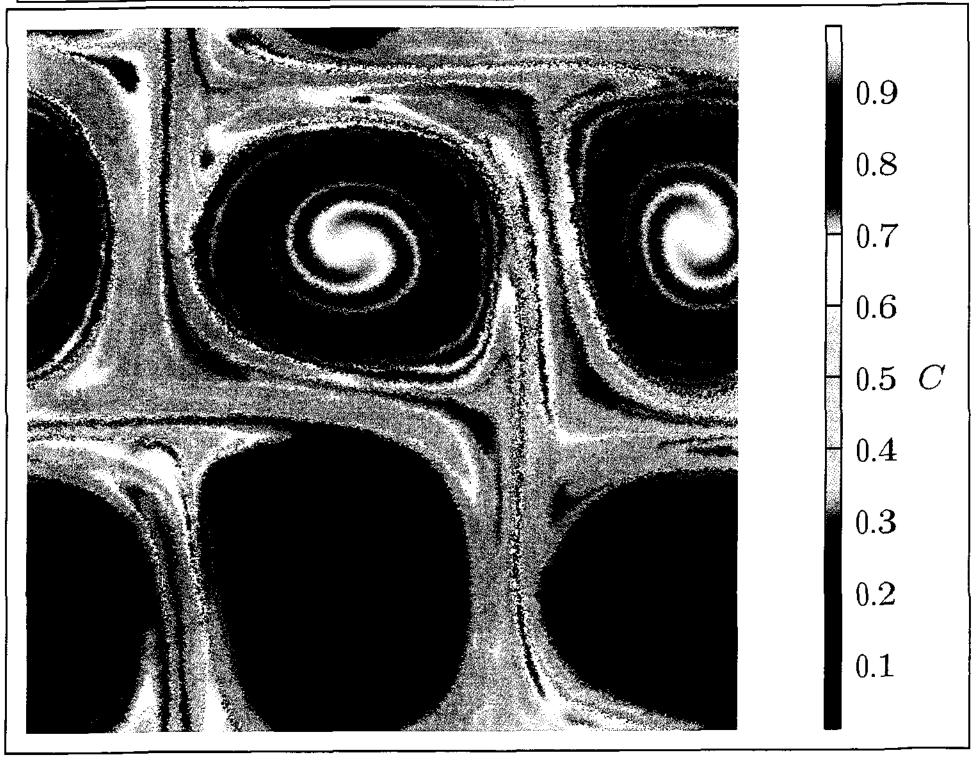
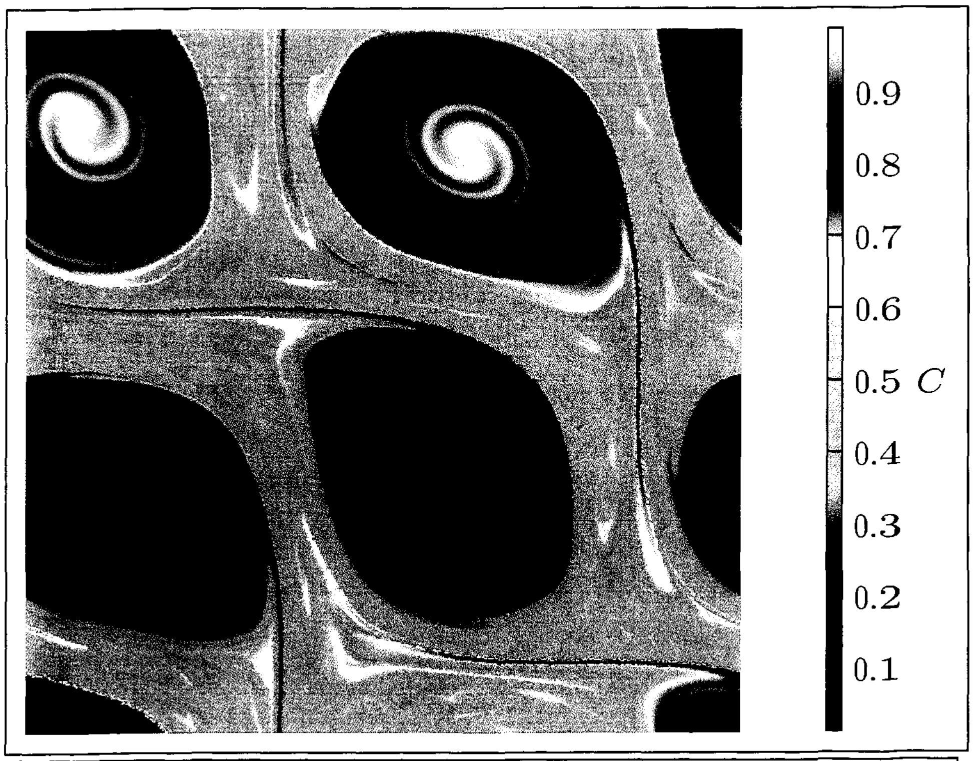


Figure 4.17: Semi-Lagrangian vs. LR after 500 time steps, with diffusion.



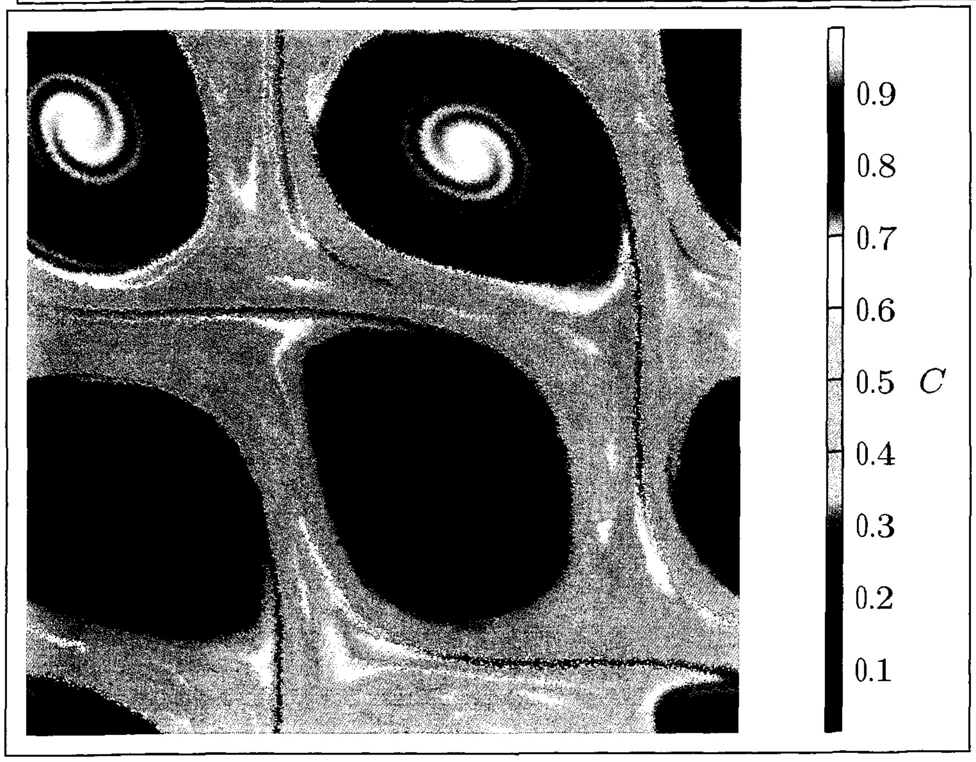
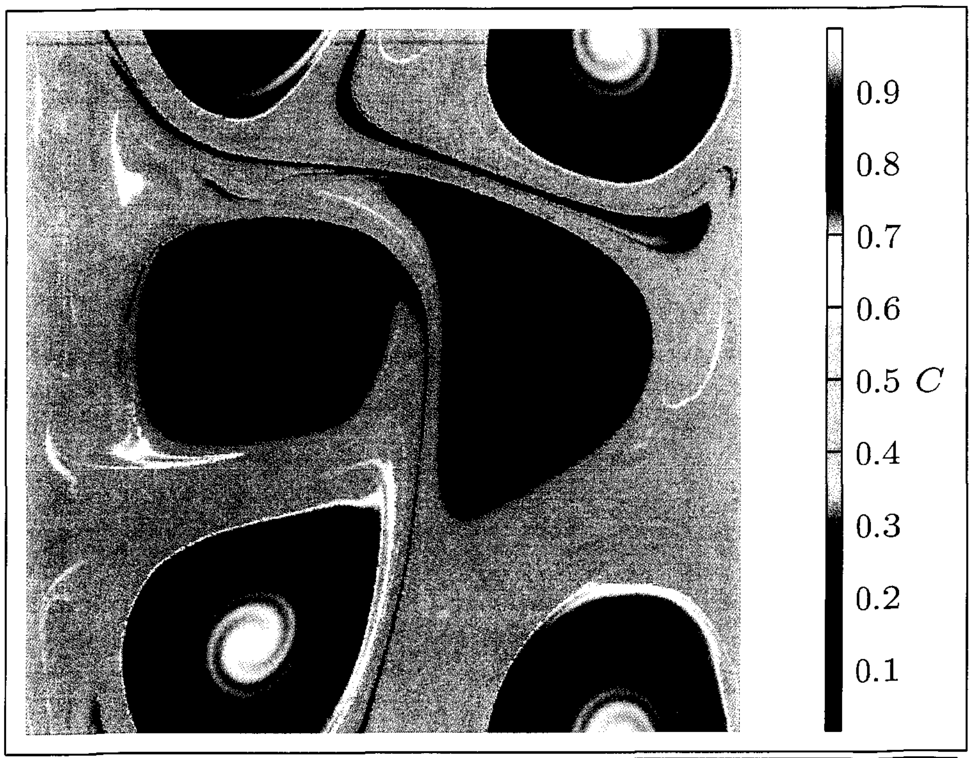


Figure 4.18: Semi-Lagrangian vs. LR after 750 time steps, with diffusion.



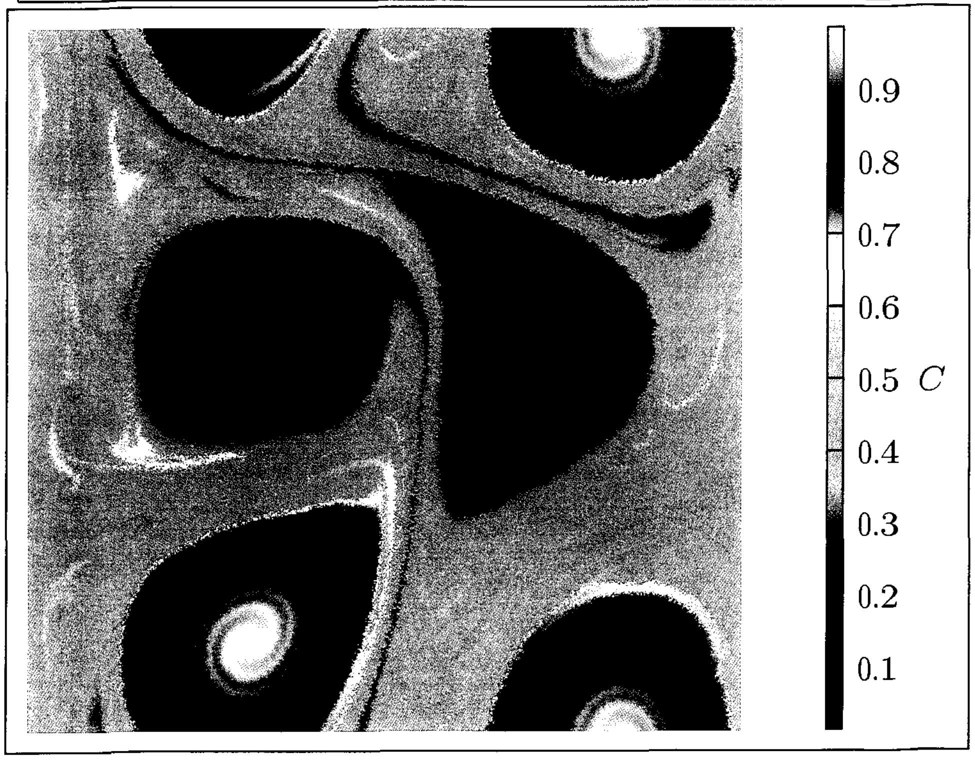


Figure 4.19: Semi-Lagrangian vs. LR after 1000 time steps, with diffusion.

In comparing the LR and semi-Lagrangian methods, we now emphasize the conservation of Casimir invariants. In the absence of diffusion, our numerical approximation to the *concentration energy* $\frac{1}{2}\int C^2\,d\boldsymbol{x}$ and the *enstrophy* $Z = \frac{1}{2}\int \omega^2\,d\boldsymbol{x}$ should be conserved, just as for an inviscid fluid. In Figs. 4.20 and 4.21 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 due to unwanted numerical diffusion. Moreover, in the case of a viscous fluid consider the energy equation obtained by multiplying both sides of 3.1 by $\boldsymbol{U}$ and integrating over the domain:

$$\begin{aligned} & \int C\frac{\partial C}{\partial t}\,d\boldsymbol{x} + \int C\boldsymbol{v}\cdot\boldsymbol{\nabla}C\,d\boldsymbol{x} = \int CD\nabla^2 C\,d\boldsymbol{x} \\ \Rightarrow\quad & \frac{1}{2}\frac{\partial}{\partial t}\int C^2\,d\boldsymbol{x} + \frac{1}{2}\int \boldsymbol{v}\cdot\boldsymbol{\nabla}C^2\,d\boldsymbol{x} = D\int C\nabla^2 C\,d\boldsymbol{x}. \end{aligned}$$

On integrating by parts, we find that

$$\int \boldsymbol{v}\cdot\frac{\boldsymbol{\nabla}C^2}{2}\,d\boldsymbol{x} = \int \boldsymbol{\nabla}\cdot\left(\boldsymbol{v}\frac{C^2}{2}\right) - \int \frac{C^2}{2}\boldsymbol{\nabla}\cdot\boldsymbol{v} = 0.$$

The first integral on the right-hand side is zero due to integration over the doubly periodic boundary conditions, and the second one is zero because of the incompressibility condition. Therefore we have:

$$\frac{1}{2}\frac{\partial}{\partial t}\int C^2\,d\boldsymbol{x} = D\int C\nabla^2 C\,d\boldsymbol{x}.$$

Furthermore, $\boldsymbol{\nabla}\cdot(C\boldsymbol{\nabla}C) = C\nabla^2 C + \boldsymbol{\nabla}C\cdot\boldsymbol{\nabla}C$, so

$$\begin{aligned}\int C\nabla^2 C\,d\boldsymbol{x} &= \int \boldsymbol{\nabla}\cdot(C\boldsymbol{\nabla}C)\,d\boldsymbol{x} - \int \boldsymbol{\nabla}C\cdot\boldsymbol{\nabla}C\,d\boldsymbol{x}\\ &= 0\text{ (periodic boundary conditions)} - \int \boldsymbol{\nabla}C\cdot\boldsymbol{\nabla}C\,d\boldsymbol{x}\\ &= -\int |\boldsymbol{\nabla}C|^2\,d\boldsymbol{x}.\end{aligned}$$

Finally,

$$\frac{1}{2}\frac{\partial}{\partial t}\int C^2\,d\boldsymbol{x} = -D\int |\boldsymbol{\nabla}C|^2\,d\boldsymbol{x}.$$

We obtain an analogous equation for the evolution of the enstrophy. Thus,

$$\frac{1}{2}\frac{\partial}{\partial t}\int C^2\,d\boldsymbol{x} = -D\int |\boldsymbol{\nabla}C|^2\,d\boldsymbol{x}, \tag{4.1}$$

$$\frac{1}{2}\frac{\partial}{\partial t}\int \omega^2\,d\boldsymbol{x} = -D\int |\boldsymbol{\nabla}\omega|^2\,d\boldsymbol{x}. \tag{4.2}$$

We now introduce the normalized energy decay rates

$$\frac{\dfrac{\partial}{\partial t}\displaystyle\int C^2\,d\boldsymbol{x}}{\displaystyle\int C^2\,d\boldsymbol{x}} \quad\text{and}\quad \frac{-2D\displaystyle\int |\boldsymbol{\nabla}C|^2\,d\boldsymbol{x}}{\displaystyle\int C^2\,d\boldsymbol{x}}.$$

Similarly we define

$$\frac{\dfrac{\partial}{\partial t}\displaystyle\int \omega^2\,d\boldsymbol{x}}{\displaystyle\int \omega^2\,d\boldsymbol{x}} \quad\text{and}\quad \frac{-2\nu\displaystyle\int |\boldsymbol{\nabla}\omega|^2\,d\boldsymbol{x}}{\displaystyle\int \omega^2\,d\boldsymbol{x}}.$$

According to (4.1) and (4.2), the energy decay rates for each field, as calculated by the two corresponding expressions, should agree.

These values for both the semi-Lagrangian and LR methods are plotted in Figs. 4.22 and 4.23, respectively. As seen in the graphs, the decay rates predicted by the semi-Lagrangian method (denoted by the subscript I) do not agree, where 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 ∇C_I (green curve) relative to the other predictions. This shows that the term $\boldsymbol{v}\cdot\boldsymbol{\nabla U}$ is not modelled by the semi-Lagrangian method to respect the correct energy decay rate for a real fluid.

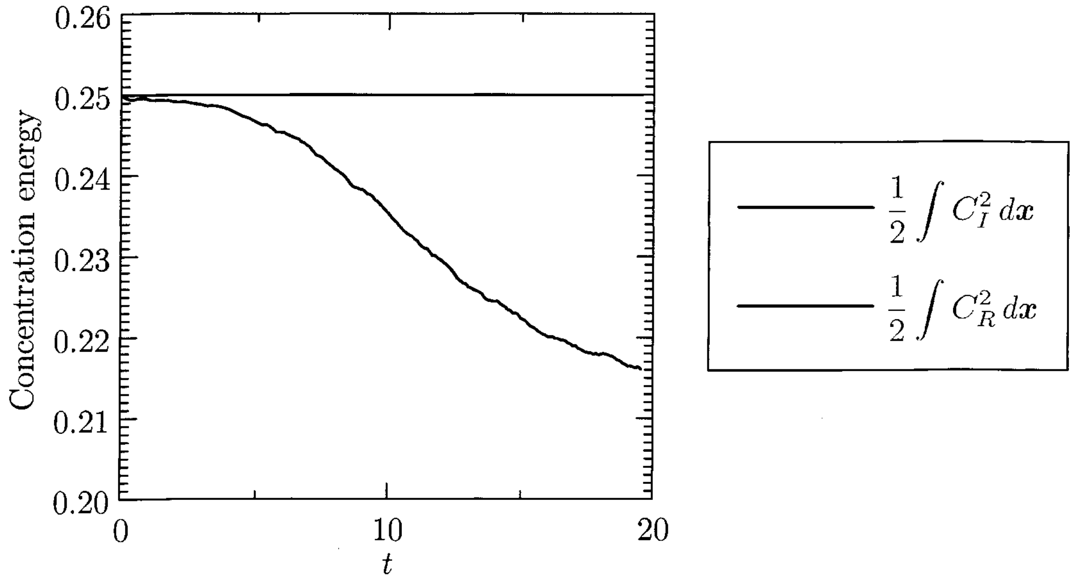


Figure 4.20: Evolution of the concentration field energy predicted by the semi-Lagrangian (I) and rearrangement (R) methods.

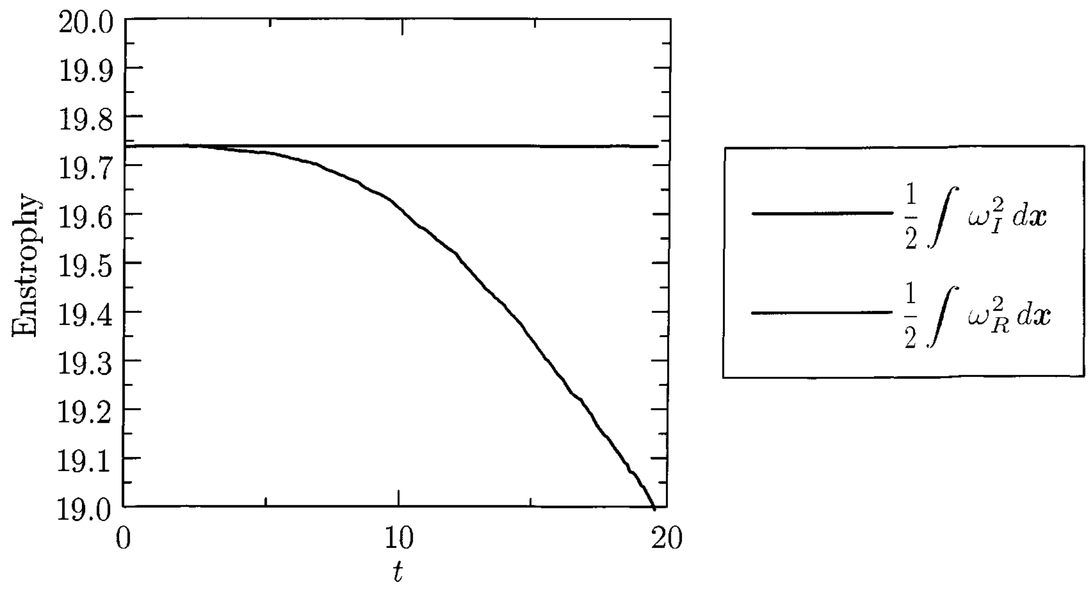


Figure 4.21: Evolution of the enstrophy predicted by the semi-Lagrangian (I) and rearrangement (R) methods.

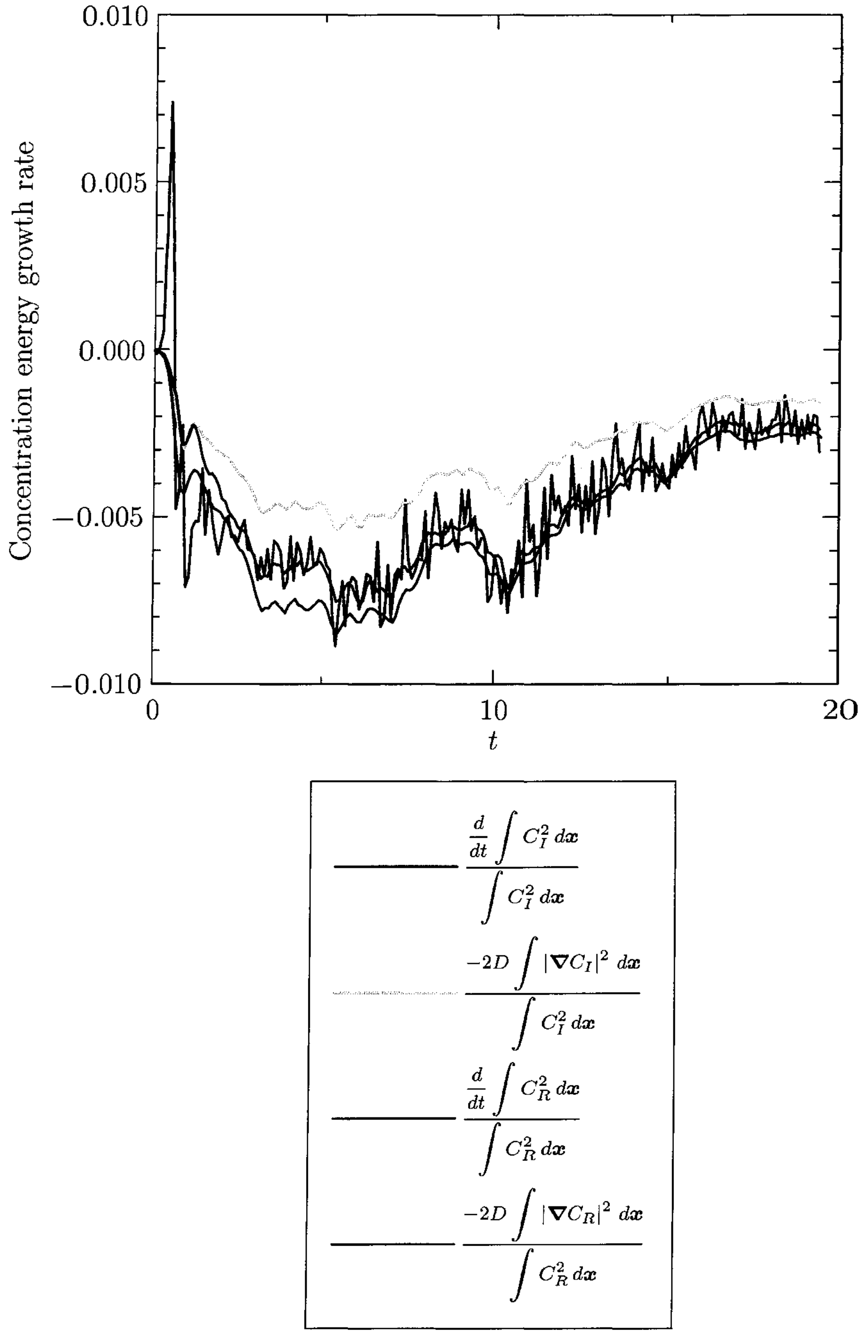
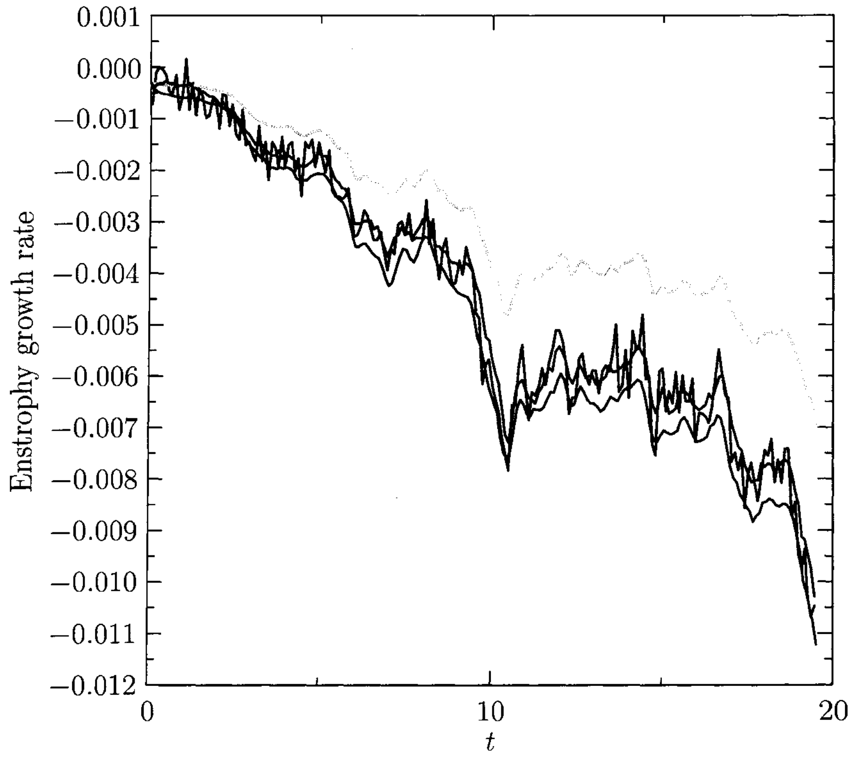


Figure 4.22: Energy decay rates for the concentration field predicted by the semi-Lagrangian (I) and rearrangement (R) methods.



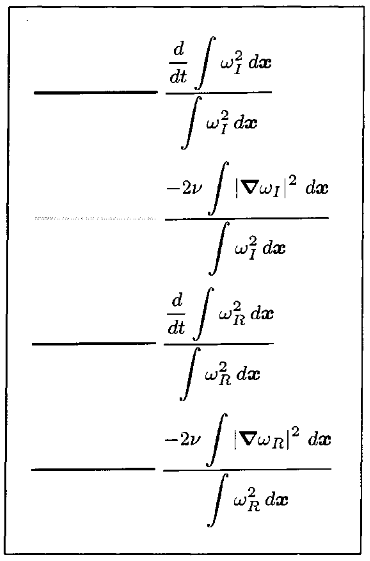

$\frac{\frac{d}{dt}\int \omega_I^2\,dx}{\int \omega_I^2\,dx}$
$\frac{-2\nu\int |\nabla\omega_I|^2\,dx}{\int \omega_I^2\,dx}$
$\frac{\frac{d}{dt}\int \omega_R^2\,dx}{\int \omega_R^2\,dx}$
$\frac{-2\nu\int |\nabla\omega_R|^2\,dx}{\int \omega_R^2\,dx}$

Figure 4.23: Enstrophy decay rates predicted by the Lagrangian semi-Lagrangian (I) and rearrangement (R) methods.

Chapter 5

Conclusion

5.A Discussion

We have proposed a numerical method for solving the advection equation, using Lagrangian advection. This method preserves Casimir invariants such as energy and momentum, just as inviscid fluids do. In this work we argue that in the inviscid case, the discretized values of the concentration field, when viewed on an Eulerian grid, should only be rearranged rather than changed, thereby enforcing a discretized version of Casimir invariance by the nonlinear advection term, treating pixels like infinitesimal parcels. At all times, the outputted concentration field is thus just a 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 our 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 must find a path from a pile to the nearest hole and push the chain of parcels (values) toward the hole.

This path is calculated using a weighted version of the Bresenham algorithm for drawing digitized 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 a energy-respecting filter for viewing the current Lagrangian solution in an Eulerian frame. To deal with self-advection, we interpolate the advected Lagrangian vorticity field on the Eulerian grid, avoiding any pushing error being transmitted to the velocity field. We then calculate the new velocity from the projected vorticity by inverting a Laplacian with a multigrid solver.

This method is used to view the conserved concentration field after it is advected by the velocity field. In the inviscid case there will be no artificial numerical diffusion. In the case of a viscous fluid we use operator splitting to account for the effects of diffusion.

5.B Future work

The algorithm can be generalized further to handle flow in a three-dimensional space in addition to the two-dimensional domain discussed here. The most significant changes (for example, the vortex stretching term) will arise from relaxing the assumption of two dimensionality in the calculation and simplification of the vorticity equation (see Appendix A). The doubly periodic boundary conditions adopted throughout this work also need to be generalized.

The algorithm might also be sped up by using data structures other than the list and vector structures used in our program. For example, the use of search trees might reduce the complexity of searching for holes and/or memory usage. Moreover, in the pushing algorithm, the simultaneous treatment of piles that are competing for a single hole may lead to further improvements. It would be preferable, and more consistent, to choose the parcel with the minimal, rather than maximal, weight as the initial parcel in a chain. The feasibility of using an adaptive grid could also be explored. Improvements should be made in dealing with diffusion and self-advected flow, by considering the advection of parcel vertices rather than the parcel centroid, using a modified area-weighted interpolation based on polygonal clipping algorithms, to account for an arbitrary advected quadrilateral parcel.

The question also arises whether one can prove that our rearrangement scheme actually converges to the exact solution as the grid is refined. One would need to show that the rearranged parcel weights all converge to zero in this limit. Finally, the upper bound in Theorem 2.1 is optimal only for short distances and could be tightened for large distances.

Appendix A

The Vorticity Equation and the Stream Function

Here we take the curl of the Navier–Stokes equation to find an equation for the vorticity $\boldsymbol{\omega} = \boldsymbol{\nabla}\times\boldsymbol{v}$. The advantage of the vorticity formulation is that the pressure term will vanish since $\boldsymbol{\nabla}\times\boldsymbol{\nabla}P = 0$ for any scalar P. On using the Einstein notation of implied summation of repeated indices, the cross product of $\boldsymbol{A} = (A_1, A_2, A_3)$ and $\boldsymbol{B} = (B_1, B_2, B_3)$ can be written $\boldsymbol{A}\times\boldsymbol{B} = \epsilon_{ijk}A_iB_j\hat{\boldsymbol{x}}_k$, where $\hat{\boldsymbol{x}}_k$ is the kth unit vector, where, for $i, j, k \in \{1, 2, 3\}$

$$\epsilon_{ijk} = \begin{cases} 1 & \text{if } ijk \text{ is an even permutation of 123,} \\ -1 & \text{if } ijk \text{ is an odd permutation of 123,} \\ 0 & \text{otherwise.} \end{cases}$$

We will need the identity

$$\begin{aligned}\boldsymbol{v}\times(\boldsymbol{\nabla}\times\boldsymbol{v}) &= \epsilon_{ijk}v_i(\boldsymbol{\nabla}\times\boldsymbol{v})_j\hat{\boldsymbol{x}}_k = \epsilon_{ijk}v_i(\epsilon_{lmj}\partial_l v_m)_j\hat{\boldsymbol{x}}_k \\ &= -\epsilon_{ikj}v_i(\epsilon_{lmj}\partial_l v_m)_j\hat{\boldsymbol{x}}_k = -v_i\partial_i(v_k\hat{\boldsymbol{x}}_k) + v_i\partial_k v_i\hat{\boldsymbol{x}}_k \\ &= -(\boldsymbol{v}\cdot\boldsymbol{\nabla})\boldsymbol{v} + \frac{1}{2}\boldsymbol{\nabla}v^2.\end{aligned}$$

Therefore, $(\boldsymbol{v}\cdot\boldsymbol{\nabla})\boldsymbol{v} = \frac{1}{2}\boldsymbol{\nabla}v^2 - \boldsymbol{v}\times\boldsymbol{\nabla}\times\boldsymbol{v}$, so that

$$\begin{aligned}\boldsymbol{\nabla}\times(\boldsymbol{v}\cdot\boldsymbol{\nabla})\boldsymbol{v} = \boldsymbol{\nabla}\times\left[\frac{1}{2}\boldsymbol{\nabla}v^2 - \boldsymbol{v}\times(\boldsymbol{\nabla}\times\boldsymbol{v})\right] &= -\boldsymbol{\nabla}\times(\boldsymbol{v}\times\boldsymbol{\omega}) \\ &= -\epsilon_{ijk}\partial_i(\epsilon_{lmj}v_l\omega_m)_j\hat{\boldsymbol{x}}_k = \epsilon_{ikj}\partial_i(\epsilon_{lmj}v_l\omega_m)_j\hat{\boldsymbol{x}}_k \\ &= \partial_i v_i\omega_k\hat{\boldsymbol{x}}_k - \partial_i v_k\omega_i\hat{\boldsymbol{x}}_k = v_i\partial_i\omega_k\hat{\boldsymbol{x}}_k - \omega_i\partial_i v_k\hat{\boldsymbol{x}}_k \\ &= \boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{\omega} - \boldsymbol{\omega}\cdot\boldsymbol{\nabla}\boldsymbol{v}.\end{aligned}$$

since $\partial_i v_i = 0$ and $\partial_i\omega_i = 0$. The curl of the Navier–Stokes equation becomes

$$\begin{aligned} & \boldsymbol{\nabla}\times\frac{\partial\boldsymbol{v}}{\partial t} + \boldsymbol{\nabla}\times(\boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{v}) = -\frac{1}{\rho}\boldsymbol{\nabla}\times\boldsymbol{\nabla}\boldsymbol{P} + \boldsymbol{\nabla}\times\nu\nabla^2\boldsymbol{v} \\ \Rightarrow\quad & \frac{\partial\boldsymbol{\omega}}{\partial t} + \boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{\omega} = \boldsymbol{\omega}\cdot\boldsymbol{\nabla}\boldsymbol{v} + \nu\nabla^2\boldsymbol{\omega}.\end{aligned}$$

For two-dimensional flow, $\boldsymbol{\omega} = \boldsymbol{\nabla}\times\boldsymbol{v} = \omega\hat{\boldsymbol{z}}$, where $\hat{\boldsymbol{z}}$ is the normal to the plane of motion, since $\boldsymbol{v}$ has no z component or z dependence. Thus, the *vortex stretching* term $\boldsymbol{\omega}\cdot\boldsymbol{\nabla}\boldsymbol{v}$ is zero and the vorticity equation becomes

$$\frac{\partial\omega}{\partial t} + \boldsymbol{v}\cdot\boldsymbol{\nabla}\omega = \nu\nabla^2\omega.$$

Moreover, for a C^1 vector field $\boldsymbol{v}$ in a simply connected domain we know that $\boldsymbol{\nabla}\cdot\boldsymbol{v} = 0 \iff \boldsymbol{v} = \boldsymbol{\nabla}\times\boldsymbol{A}'$, where $\boldsymbol{A}'$ is a vector potential. We can always introduce $\boldsymbol{A} = \boldsymbol{A}' + \boldsymbol{\nabla}\phi$, where ϕ satisfies $\nabla^2\phi = -\boldsymbol{\nabla}\boldsymbol{\cdot}\boldsymbol{A}'$. Then

$$\boldsymbol{\nabla}\boldsymbol{\cdot}\boldsymbol{v} = 0 \Longleftrightarrow \boldsymbol{v} = \boldsymbol{\nabla}\boldsymbol{\times}\boldsymbol{A} \quad \text{with} \quad \boldsymbol{\nabla}\boldsymbol{\cdot}\boldsymbol{A} = 0,$$

so that $\boldsymbol{\omega} = \boldsymbol{\nabla}\boldsymbol{\times}\boldsymbol{v} = \boldsymbol{\nabla}\boldsymbol{\times}(\boldsymbol{\nabla}\boldsymbol{\times}\boldsymbol{A}) = \boldsymbol{\nabla}(\boldsymbol{\nabla}\boldsymbol{\cdot}\boldsymbol{A}) - \nabla^2\boldsymbol{A} = -\nabla^2\boldsymbol{A}$. Thus $\nabla^2 A_x = \nabla^2 A_y = 0$ (where A_x means the x component of $\boldsymbol{A}$). Given zero boundary conditions at infinity or periodic boundary conditions, we can assume, without loss of generality, that $A_x = A_y = 0$. That is, $\boldsymbol{A} = A_z\hat{\boldsymbol{z}}$. It is conventional to define the *stream function* $\psi = -A_z$, in terms of which $\omega = \nabla^2\psi$ and

$$\boldsymbol{v} = \boldsymbol{\nabla}\boldsymbol{\times}\boldsymbol{A} = \boldsymbol{\nabla}\boldsymbol{\times}(-\psi\hat{\boldsymbol{z}}) = \hat{\boldsymbol{z}}\boldsymbol{\times}\boldsymbol{\nabla}\psi = \left(-\frac{\partial\psi}{\partial y}, \frac{\partial\psi}{\partial x}\right).$$

Without loss of generality we can thus assume that ψ has no z dependence. For further details, see for example Bowman [2004].

Appendix B

Optimized Bresenham Algorithm

As described in Sect. (2.A.1), the Bresenham algorithm is designed to discretize a line between two points. There are eight cases to consider, depending on the location of the destination point. The first case, where the final point lies in the first octant relative to the initial point, is discussed in Sect. 2.A.1. The following is `Asymptote` code (a vector graphics language for technical drawing [Hammerlindl *et al.* 2004]) for efficiently handling all eight cases, by mapping them to the first case. The use of integer arithmetic avoids floating point roundoff error. Test cases for all eight octants are shown in Fig. B.1, where the green dot indicates the starting point and the red dot is the end point.

```
pair[] Bresenham(int x, int y, int x2, int y2)
{
  pair[] line;
  int signx=x2 >= x ? 1 : -1;
  int signy=y2 >= y ? 1 : -1;
  int dx=signx*(x2-x), dy=signy*(y2-y), eps=0;
  if(dx >= dy) {
    dy *= 2;
    int deps=2*dx;
    for(; x != x2; x += signx) {
      line.push((x,y));
      eps += dy;
      if(dx <= eps) {
        y += signy;
        eps -= deps;
      }
    }
  } else {
    dx *= 2;
    int deps=2*dy;
    for(; y != y2; y += signy) {
      line.push((x,y));
      eps += dx;
      if(dy <= eps) {
        x += signx;
        eps -= deps;
      }
    }
  }
  line.push((x2,y2));
  return line;
}
```

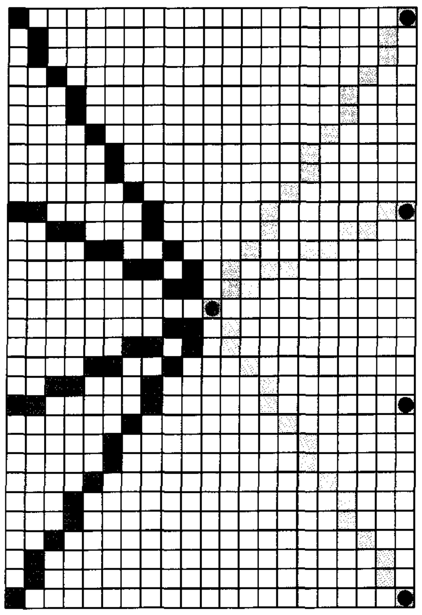


Figure B.1: Sample lines for each octant drawn by the Bresenham algorithm.

Appendix C

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. C.1 shows the same test cases as in Fig. B.1. All cases have been mapped to the first quadrant. 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 (see Fig. 2.6 in 2.B.1). The following `Asymptote` code for the weighted Bresenham algorithm has been optimized in the manner described for the Bresenham algorithm in Appendix B.

```
pair[] discreteline(int x, int y, int x2, int y2, int[][] count)
{
  pair[] line;
  while(true) {
    int signx,signy;
    int dx=x2-x, dy=y2-y;
    if(dx < 0) {
      signx=-1; dx=-dx;
    } else signx=1;
    if(dy < 0) {
      signy=-1; dy=-dy;
    } else signy=1;
    line.push((x,y));
    if(dx >= dy) {
      if(dx <= 1) {
        if(dx > 0) line.push((x2,y2));
        return line;
      }
      int N=count[x][y+signy];
      int NE=count[x+signx][y+signy];
      int E=count[x+signx][y];
      if(dx == dy) {
        if(E >= NE && E >= N) {                    // NE or N
          y += signy;
          if(NE <= N) x += signx;                  // favour NE
        } else if(N >= NE && N >= E) {             // NE or E
          x += signx;
          if(NE <= E) y += signy;                  // favour NE
        } else {                                   // E or N
          if(E <= N) x += signx;                   // favour E
          else y += signy;
        }
      } else {
        int SE=count[x+signx][y-signy];
        x += signx;
        if(dy == 0) {
          if(SE >= E && SE >= NE) {                // E or NE
            if(NE < E) y += signy;                 // favour E
          } else if(NE >= E && NE >= SE) {         // E or SE
            if(SE < E) y -= signy;                 // favour E
          } else {                                 // NE or SE
            if(NE <= SE) y += signy;               // favour NE
            else y -= signy;
          }
        } else if(NE < E || (NE == E && dx <= 0)) // E or NE
```

```
            y += signy;                                 // favour Bresenham
          }
        } else {
          if(dy == 1) {
            line.push((x2,y2));
            return line;
          }
          int N=count[x][y+signy];
          int NE=count[x+signx][y+signy];
          int NW=count[x-signx][y+signy];
          y += signy;
          if(dx == 0) {
            if(NW >= N && NW >= NE) {                  // N or NE
              if(NE < N) x += signx;                   // favour N
            } else if(NE >= N && NE >= NW) {           // N or NW
              if(NW < N) x -= signx;                   // favour N
            } else {                                   // NE or NW
              if(NE <= NW) x += signx;                 // favour NE
              else x -= signx;
            }
          } else if(NE < N || (NE == N && dy <= 0))    // N or NE
            x += signx;                                // favour Bresenham
        }
      }
      return line;
    }
```

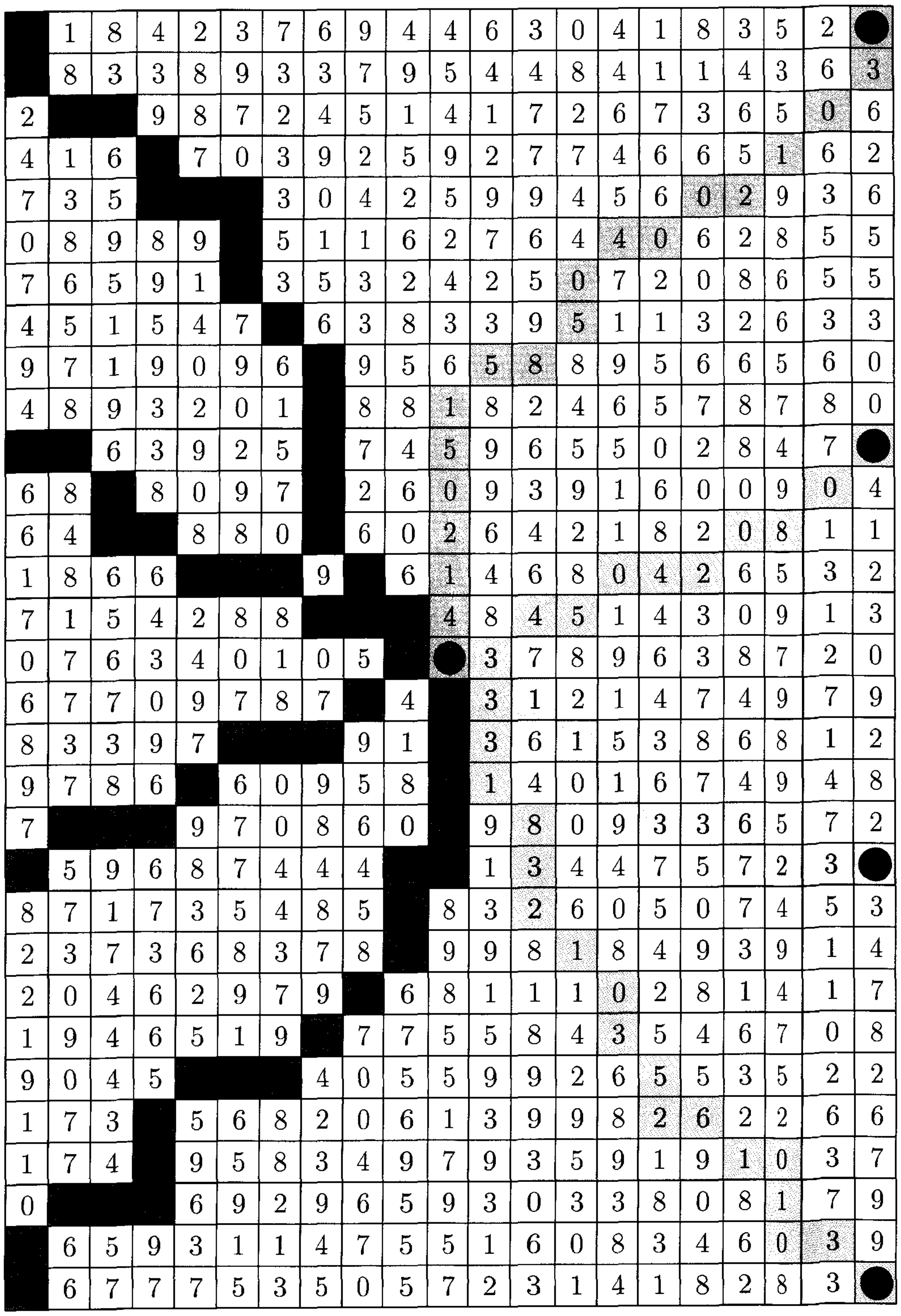


Figure C.1: Weighted Bresenham algorithm.

Appendix D

Multigrid Method

One can discretize a differential equation as a system of linear equations $\boldsymbol{Au} = \boldsymbol{f}$ on a grid, where $\boldsymbol{u}$ denotes the exact solution. Using iterative methods, one can find the ith iterative approximation to $\boldsymbol{u}$, represented by $\boldsymbol{v}^i$. For instance, on a one-dimensional grid in $[0, 1]$, the differential equation $-u'' = f$ can be written as

$$-u_{j-1} + 2u_j - u_{j+1} = f_i h^2, \quad 1 \leq j \leq N-1,$$

where $h = 1/N$, is the grid size. For example, one could use the *Jacobi iteration*

$$v_j^i = \frac{1}{2}(v_{j-1}^{i-1} + v_{j+1}^{i-1} + hf_i).$$

The relaxation scheme or iteration can be represented in matrix form as well. For example, the matrix $\boldsymbol{A}$ can be written as $\boldsymbol{A} = \boldsymbol{D} - \boldsymbol{L} - \boldsymbol{U}$, where $\boldsymbol{D}$, $-\boldsymbol{L}$, and $-\boldsymbol{U}$ represent the diagonal, the strictly lower, and strictly upper triangular parts of $\boldsymbol{A}$ respectively. Then we have:

$$(\boldsymbol{D}-\boldsymbol{L}-\boldsymbol{U})\boldsymbol{u}=\boldsymbol{f} \quad \Rightarrow \quad \boldsymbol{u}=\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{u}+\boldsymbol{D}^{-1}\boldsymbol{f}.$$

Let $\boldsymbol{P}_J=\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})$. Then $\boldsymbol{v}=\boldsymbol{P}_J\boldsymbol{v}+\boldsymbol{D}^{-1}\boldsymbol{f}$. Also, the new Jacobi iterate, which is only an intermediate value, is computed as $v_j^*=\frac{1}{2}(v_{j-1}^i+v_{j+1}^i+hf_i)$. Alternatively, we can make the iteration a weighted or damped Jacobi method by introducing a weight factor w: $v_j=(1-w)v_j^i+wv_j^*$. The iteration then becomes

$$\boldsymbol{v}^{i+1}=[(1-w)I+\boldsymbol{P}_J]\boldsymbol{v}^i+\boldsymbol{D}^{-1}\boldsymbol{f}.$$

The *algebraic error* is $\boldsymbol{e}=\boldsymbol{u}-\boldsymbol{v}^i$. The *residual*, the measure of how much the approximated solution is unable to fulfill the system, is determined by $\boldsymbol{r}=\boldsymbol{f}-\boldsymbol{A}\boldsymbol{v}^i$. The equation $\boldsymbol{A}\boldsymbol{e}=\boldsymbol{f}$ is called the *residual equation*, which is useful to compute the new approximation to an already calculated approximation $\boldsymbol{v}^i$. Using the residual $\boldsymbol{r}=\boldsymbol{f}-\boldsymbol{A}\boldsymbol{v}^i$, we can solve the residual equation and the improved new approximation is then $\boldsymbol{u}=\boldsymbol{v}^i+\boldsymbol{e}$. An improvement that can be done is to use a good initial guess. This can be achieved by making use of only part of a grid. That is, only some of the grid points are considered, which is referred to as a coarse grid. One idea is to do some iteration on the coarse grid and use the approximation as an initial guess to iterate on the fine (original) grid. The transfer from the coarse grid to the fine grid is called an *extension* (or *prolongation*) and the reverse process is called a *restriction* (or *projection*). Another idea is to make use of the residual equation. Denote the values on the coarse grid and fine grid by the subscripts c and f, respectively. For example we can do a few iterations for $\boldsymbol{A}\boldsymbol{u}_f=\boldsymbol{0}$ on the fine grid, and then compute the residual and transfer to the coarse grid. One performs an iteration of $\boldsymbol{A}\boldsymbol{e}_c = \boldsymbol{r}_c$ on the coarse grid to compute the error there. Next, one extends the error to the fine grid, uses it to calculate the new approximation $\boldsymbol{v} = \boldsymbol{v} + \boldsymbol{e}_f$, and repeats the procedure. (For more details see Briggs [1987], Bramble [1993], and Hackbusch [1985]).

Appendix E

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 $r = 1/(e-1)$. Now since the probability of not having a hole in $8k$ cells is $(1-1/e)^{8k}$, we know that

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

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, one then deduces

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

so that the expected number of holes evaluates simply to

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

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.

Bibliography

[Alam & Bowman 2002]	J. Alam & J. Bowman, Theoretical and Computational Fluid Dynamics, **16**:1, 2002.
[Ames 1977]	W. Ames, *Numerical Methods for Partial Differential Equations*, Academic Press, San Diego, CA, 1977.
[Basse & Gelder 2000]	S. Basse & A. V. Gelder, *Computer Algorithms, Introduction to Design and Analysis*, Addison-Wesley, Ontario, 2000.
[Behrens & Mentrup 2005]	Behrens & Mentrup, "A conservative scheme for 2d and 3d adaptive semi-Lagrangian advection," in *Recent Advances in Adaptive Computation*, edited by Z.-C. Shi, Z. Chen, T. Tang, & D. Yu, volume 383, 2005.
[Behrens 1995]	J. Behrens, Modeling and Computation in Environmental Sciences, Proceedings of the First GAMM-Seminar at ICA Stuttgart, p. 49, 1995.
[Bowman 2004]	J. C. Bowman, *Math 655: Statistical theories of turbulence*, online lecture notes available at `http://www.math.ualberta.ca/~bowman`, 2004.
[Bramble 1993]	J. H. Bramble, *Multigrid Methods*, Longman Scientific and Technical, London, 1993.
[Bresenham 1965]	J. E. Bresenham, IBM Systems Journal, 4:25, 1965.
[Briggs 1987]	W. L. Briggs, *A Multigrid Tutorial*, SIAM, Philadelphia, 1987.

[Courant *et al.* 1967]	R. Courant, K. Friedrichs, & H. Lewy, IBM Journal of Research and Development, **11**:215, 1967.
[Evans 1998]	L. C. Evans, *Partial differential equations*, American Mathematical Society, Providence, Rhode Island, 1998.
[Fraccarollo *et al.* 2003]	L. Fraccarollo, H. Capart, & Y. Zech3, International Journal For Numerical Methods In Fluids, **41**:951, 2003.
[Frisch 2001]	U. Frisch, *Turbulence: The Legacy of A.N. Kolmogorov*, Cambridge University Press, New York, 2001.
[Godunov & Ryabenkii 1964]	S. Godunov & V. Ryabenkii, *Theory of Difference schemes*, John Wiley and Sons, New York, 1964.
[Godunov 1959]	S. Godunov, Sbornik. Mathematics, **47**:271, 1959.
[Grigoryev *et al.* 2002]	Y. Grigoryev, V. Vshivkov, & M. Fedoruk, *Numerical "Particle-in-Cell" Methods—theory and applications*, Brill Academic Publishers, 2002.
[Hackbusch 1985]	W. Hackbusch, *Multi-Grid Methods and Applications*, Series in Computational Mathematics, Springer, New York, 1985.
[Hammerlindl *et al.* 2004]	A. Hammerlindl, J. C. Bowman, & R. T. Prince, `ASYMPTOTE`: *A descriptive vector graphics language*, available online at `http://asymptote.sourceforge.net`, 2004.
[Lax & Wendroff 1960]	P. Lax & B. Wendroff, Communications on Pure and Applied Mathematics, **13**:217, 1960.
[Leboeuf *et al.* 1979]	J. Leboeuf, T. Tajima, & J. Dawson, Journal of Computational Physics, **31**:379, 1979.
[Leslie & Purser 1995]	L. M. Leslie & R. J. Purser, Monthly Weather Review, **123**:25, 1995.
[LeVeque 1990]	R. J. LeVeque, *Numerical Methods for Conservation Laws*, Birkhauser Verlag, Boston, 1990.

[Morrison 1998]	P. J. Morrison, Rev. Mod. Phys., **70**:467, 1998.
[Popinet & Zaleski 1999]	S. Popinet & S. Zaleski, International Journal for Numerical Methods in Fluids, p. 775, 1999.
[Press *et al.* 1992]	W. H. Press, S. A. Teukolsky, W. T. Vetterling, & B. P. Flannery, *Numerical Recipes, The Art of Scientific Computing,* Cambridge Univ. Press, Cambridge, 2nd edition, 1992.
[Quarteroni *et al.* 2002]	A. Quarteroni, A. Veneziani, & P. Zunino, SIAM Journal on Scientific Computing, p. 1959, 2002.
[Simmons 1991]	G. Simmons, *Differential equations with applications and historical notes*, McGraw-Hill, New York, 1991.
[Wang & Hutter 2001]	Y. Wang & K. Hutter, International Journal for Numerical Methods in Fluids, **37**:721, 2001.
[Woodward & Colella 1984a]	P. Woodward & P. Colella, Journal of Computational Physics, **54**:174, 1984.
[Woodward & Colella 1984b]	P. Woodward & P. Colella, Journal of Computational Physics, **54**:115, 1984.
[Yee & Harten 1985]	H. C. Yee & A. Harten, 7th Computational Fluid Dynamics Conference, p. 228, 1985.
[Yoko Takakura 1989]	S. O. Yoko Takakura, Tomiko Ishiguro, International Journal for Numerical Methods in Fluids, **9**:1011, 1989.

Index