Analysis of Contour Crossings in Contour-Advective Simulations - Part 1 - Algorithm

T. M. Schaerf and C. Macaskill

June 10, 2008

Abstract

This is the first of two papers devoted to the analysis of contour crossing errors that occur in contour-advective simulations of fluid motion. Here an algorithm is presented for quantifying the error due to contour crossings. The method for analysing contour crossing errors works in several stages. The first step is to determine the relative proximity of all possible pairs of contours. A digital representation of each contour is produced to aid in the corresponding calculation. Simple analysis of functions is then used to find any crossings between contours deemed close to each other by the digital representation method. Next, the area in error of a pair of crossing contours is calculated by identifying the polygon or polygons that approximately bound the erroneous region. Finally, some preliminary results of analysis of contour crossings that occur in Contour-Advective Semi-Lagrangian (CASL) simulations of single layer quasigeostrophic turbulence are presented. It is shown that the error due to contour crossings is small in the simulations considered here.

1 Introduction

The Contour-Advective Semi-Lagrangian (CASL) algorithm is a powerful hybrid of contour and grid based techniques for simulating the evolution of inviscid fluid flows. The CASL technique was first described in Dritschel and Ambaum (1997) with an application to simulating quasigeostrophic fluid motion in a doubly periodic domain. The method has since been further developed to be used for a broad range of applications including the simulation of quasigeostrophic flows in a cylindrical domain (Macaskill et al., 2003), on the surface of a sphere (Dritschel, 1999), and in a periodic channel (Benilov et al., 2004). A CASL method now exists for the study of fluid motions governed by the shallow-water equations in Dritschel et al. (1999) and has been used in the development of an explicit potential vorticity conserving approach to modelling non-linear internal gravity waves in Viúdez and Dritschel (2002). The efficiency of the CASL method is such that it allows much higher resolution simulations to be performed than standard spectral or finite difference simulations of equivalent computational cost. One of the exciting recent uses of the CASL method has allowed deeper insight into some aspects of unforced two-dimensional turbulence, (Dritschel et al., 2008). In particular, the high resolution achievable by the CASL technique allows determination of the high wavenumber behaviour of the enstrophy and energy spectra and the corresponding time decay rates. Given the rapid development and uptake of the CASL algorithm it is important to assess the accuracy of the technique. This paper details a systematic method for quantifying a form of numerical error that occurs in CASL simulations, as well as simulations performed by the related techniques of contour dynamics and contour advection with surgery. This error

manifests itself in the form of contour crossings. To put this problem in context a brief summary of the development and features of contour dynamics, contour advection with surgery and the CASL method is provided here.

Contour dynamics is used commonly in the simulation of inviscid vortex motion. The method of contour dynamics is based on the earlier water-bag model for solving the Vlasov equation that appears in plasma physics (Berk and Roberts, 1970). Contour dynamics was initially presented in the context of simulating two-dimensional fluid motions governed by the barotropic vorticity equation in a domain of infinite extent by Zabusky et al. (1979). Since then it has been modified to deal with fluid motions in a number of different geometries and flow regimes, including vortical motions on the surface of a sphere or a cylinder of infinite extent (Dritschel, 1989) and multilayer quasigeostrophic flows (Dritschel and Saravanan, 1994). One of the appealing features of contour dynamics and related techniques is the ability to resolve fine-scale features such as steep gradients in the vorticity or potential vorticity (PV) field.

In contour dynamics the potential vorticity field of the flow is represented by a series of contours. Associated with each contour is a jump in potential vorticity. Between contours the potential vorticity is assumed to be uniform. The contours are represented by nodes, usually most densely placed in regions of high local curvature, and by some interpolating function between the nodes. Both the method of node distribution and the type of interpolating functions used vary from author to author (Zabusky et al., 1979; Dritschel, 1988, 1989; Vosbeek and Mattheij, 1997; Vera and Rebollo, 2001). The nodes on the contours are advected at each time step according to the prevailing velocity field, which is calculated through the evaluation of contour integrals. For n nodes the computational cost of calculating the advecting velocities is $\mathcal{O}(n^2)$. As a flow evolves, and often becomes increasingly complex, it is necessary to insert and redistribute nodes to adequately resolve the features of the flow.

There are two major drawbacks to the method of contour dynamics. The first is the potentially rapid growth in the number of nodes needed to represent the contours. This rapid node growth is often associated with the formation of long filamentary structures in the potential vorticity field that are believed to contribute very little to overall flow dynamics. The second drawback is the $\mathcal{O}(n^2)$ dependence of the velocity field calculation which is required every time step. In combination, the rapid growth in nodes and the cost of the velocity calculation can cause calculations to grind to a virtual standstill.

Contour surgery (Dritschel, 1988, 1989) was introduced to treat the development of fine scale structures, and thus help to reduce the growth in the number of nodes in a consistent manner. This method effectively removes features in the potential vorticity field that form below a predefined surgical scale, δ, which is chosen consistently with the node distribution scheme as described in Dritschel (1988) and Dritschel (1989). Contour surgery deletes suitably small contours, acts to shorten filaments that appear in the PV field and joins together contours that bound equal levels of potential vorticity provided they are sufficiently close to each other. Contour surgery is similar in some respects to the procedure of trimming implemented in the water-bag method in Berk and Roberts (1970).

Moment accelerated contour surgery, as in Dritschel (1993), and the Hierarchical-Element method of Vosbeek et al. (2000) are amongst a group of techniques developed with the intent of increasing the speed of contour dynamical simulations by reducing the computational cost of determining the velocity field. The last decade has also seen the introduction and continued development of the Contour-Advective Semi-Lagrangian (CASL) algorithm of Dritschel and Ambaum (1997) which has retained the advantageous features of both contour dynamics and

contour surgery, whilst removing the $\mathcal{O}(n^2)$ dependence of the velocity field calculation. The CASL method was developed after the success of the diagnostic tool "contour advection with surgery" (Waugh and Plumb, 1994), where a gridded known wind distribution is used to advect a set of material contours that are treated as passive tracers. As with contour dynamics, the potential vorticity field is represented by a series of contours in the CASL method. The velocity field, however, is not calculated by evaluation of contour integrals. Rather, for each time step required for temporal integration, the potential vorticity field is transferred from the contours on to a fine grid. The potential vorticity values at each grid point are then averaged, perhaps several times, onto a coarser grid where the velocity is calculated using some grid based method such as a finite difference scheme or fast Fourier transforms. The gridded velocities are then interpolated back to the nodes on the contours, which are advected according to an appropriate time integration scheme. The essential technical requirement needed for such a method to work efficiently is a fast and accurate method for transferring from the Lagrangian contoured representation of the potential vorticity field to the Eulerian gridded representation of the same, and then the transfer of the velocities from the grid points back to the nodal points of the contours. This issue was resolved in Dritschel and Ambaum (1997) where an $\mathcal{O}(n)$ scheme for potential vorticity contour to grid conversion was described. Bilinear interpolation is used to obtain nodal velocities from the gridded velocity field.

Contour crossings are a form of numerical error common to simulations performed by contour dynamics, contour advection with surgery and the CASL algorithm. The severity of contour crossings is dependent on the type of fluid motion simulated. They can range in size and severity from tiny and simple, as illustrated in figure 1, to relatively large and quite complex, as illustrated in figure 2. Although the problem of contour crossings has existed since the original development of contour dynamics it is more important to study the problem now. This is the case because the CASL algorithm is capable of performing long timescale calculations whereas contour dynamics was developed to perform short timescale calculations. Any cumulative numerical error is more likely to be significant in a CASL simulation.

The formation of contour crossings during contour advective simulations has been discussed in several articles (Dritschel, 1989; Waugh and Plumb, 1994; Vosbeek and Mattheij, 1997; Vera and Rebollo, 2001). In Dritschel (1989) inadequate resolution of contours in regions where they closely approach other contours is identified as the principal cause of contour crossings. To combat this, a method of node redistribution based on both the local curvature of individual contours and a non-local measure of the curvature of other contours was developed. The potential for contour crossings that occur with the close approach of individual contours is discussed again in Vosbeek and Mattheij (1997) where an explicit geometric technique for preventing crossings is described. The technique involves placing nodes along contours at points of close approach to other contours.

A different form of contour crossing is discussed in Waugh and Plumb (1994) where the importance of periodically redistributing nodes along contours is emphasized. In the absence of some form of node redistribution a contour that has developed increased complexity will no longer be adequately resolved and can become horribly tangled, as illustrated in figure 11 of Waugh and Plumb (1994).

In Vera and Rebollo (2001) an explicit example of contour crossings between two vortices is given. The time at which the crossing first occurs is referred to as the singular time. For a small study which involved varying the total number of nodes, n, and the time step, Δt, it was shown that the singular time could be delayed by both increasing n and decreasing Δt, although the parameters were not varied independently. In Dritschel (1989) it is suggested

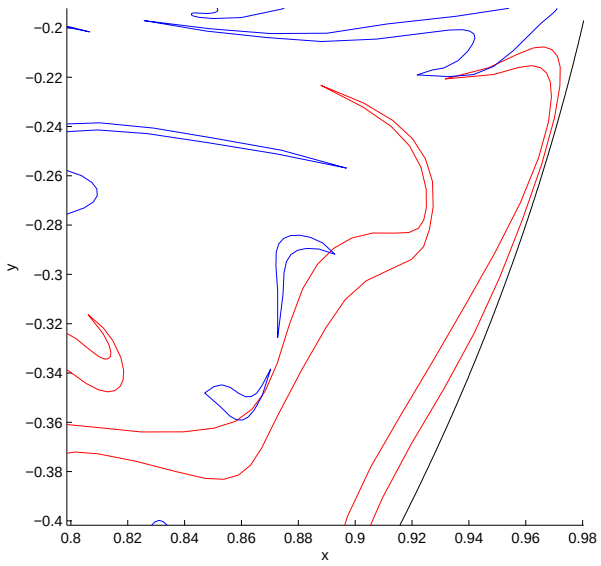

Figure 1: Detail of three simple contour crossings that have occurred during the period of peak complexity of quasigeostrophic turbulence in a cylindrical domain as simulated by the CASL algorithm.

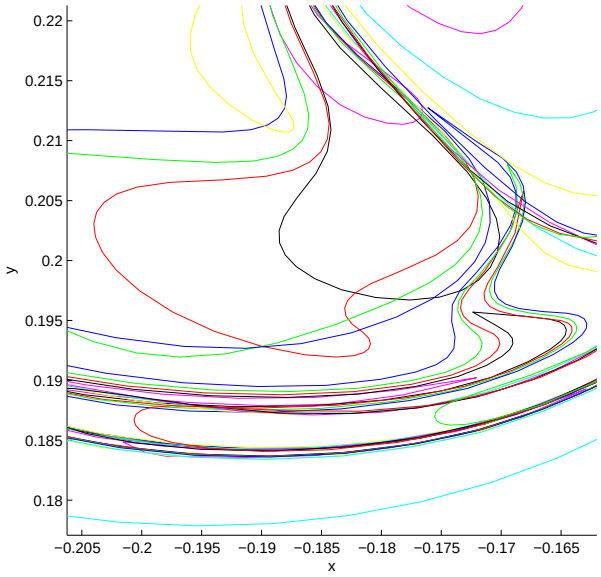

Figure 2: Detail of contour crossings that occur during the late stages of evolution of the motion of a vortex patch on the β-plane as simulated by the CASL algorithm. The contours pictured represent part of the regular component of the flow, which are also referred to as β-contours.

that reducing Δt alone cannot prevent a contour crossing as the contours must also be sufficiently resolved to deform correctly according to the prevailing velocity field. This appears

to be true in general with CASL simulations as well. Provided that Δt is sufficiently small, based on considerations such as the rate of rotation of the vortices, it is the spatial resolution of individual contours that determines if crossings occur. After the singular time Vera and Rebollo (2001) treats the numerical solution obtained as incorrect. The view of the present paper is slightly different. Although contour crossings are clearly undesirable, they represent a form of numerical error resulting from the limitations of finite resolution. As is the case with other forms of numerical error, a method for quantifying contour crossing errors should be produced and in cases where the error is considered too severe steps should be made to remedy the problem.

This paper is devoted to describing a method for quantifying numerical error due to contour crossings that occur in CASL simulations of geophysical fluid dynamics. Section 2 describes an algorithm for explicitly quantifying numerical error due to contour crossings. Section 3 follows with some initial analysis of contour crossing errors that occur in CASL simulations of quasigeostrophic turbulence in a domain with a cylindrical boundary, anticipating a more complete analysis in part 2 of this work. Concluding remarks are made in section 4.

2 An Algorithm for the Quantification of Contour Crossing Errors

Associated with the problem of contour crossings is the necessity to develop a method for quantifying the numerical error due to such crossings. An obvious first step is to create a technique for finding any crossings that occur during a contour advective simulation. One measure for quantifying the error due to contour crossings would then be to count the number of crossings. It turns out that in many cases the number of crossings by itself is not an accurate reflection of the magnitude of crossing related errors. A simple example can be used to illustrate this point. Consider two closed circular contours, initially separated by some distance, that have crossed during the course of a vortex simulation. The crossing contours will have two points of intersection, regardless of how far one contour has pushed through the other, but the error will be more severe if the area of intersection of the two contours is large. It turns out that the combination of the number of contour crossings with the area of intersection of any crossing contours gives a good indication of the true magnitude of error. If the areas in error become larger than the smallest scale being resolved in a particular simulation then the error could have significance. Errors that appear at the grid scale of a CASL simulation, which is usually ten times larger than the smallest resolved scales of the contours, would be considered more severe and might influence the future behaviour of the simulated flow.

This section is devoted to describing an algorithm for quantifying error due to contour crossings that appear in contour advective simulations. This is the first time that an explicit technique for analysing contour crossings has been proposed, developed and implemented. The algorithm first determines any contour crossings through the explicit identification of any points of intersection, as is described in section 2.1 below. Following the detection of any crossings, the area of intersection of crossing contours is used to further quantify the numerical error. A technique for determining the area of intersection of crossing contours is discussed in section 2.2. In the case of nested contours, such as those used for the discrete representation of continuous distributions of potential vorticity, the area of intersection often

does not represent the actual area in error. Rather, it is the area of the protrusion made by an inner contour through an outer contour that correctly quantifies the associated error. Section 2.3 describes the necessary modifications to the technique of 2.2 to calculate areas of extrusion of crossing contours instead of areas of intersection.

2.1 Identification of contour crossings

In the following sections a method for explicitly identifying contour crossings that occur between an arbitrary number of contours is described. An early version of this algorithm was described in Schaerf and Macaskill (2004) and so we only provide a summary of the final version here.

For the purpose of renoding in the CASL method a contour is represented by a locally defined cubic polynomial between each pair of nodes. Each cubic polynomial's departure from linearity is small because nodes are clustered in regions of high curvature.

To simplify the problem of detecting contour crossings the parts of contours between nodes are treated as straight line segments; this is justified by the small departure from linearity of the cubic splines. The problem then becomes one of finding the points of intersection between an arbitrary number of simple polygons (closed contours), each comprised of an arbitrary number of vertices (nodes) and edges (the interpolating functions between nodes). The description of the algorithm that follows is limited to the analysis of a single layer two dimensional flow, but it can be generalised to deal with multilayer flows by using the single layer algorithm to analyse each layer in turn.

2.1.1 Overview of the algorithm to identify contour crossings

The algorithm developed here to identify contour crossings analyses data from a CASL simulation at any given time step. For each time step, CASL output provides the number of contours, the total number of nodes used to represent all the contours, the number of nodes used to represent each individual contour and the (x, y) coordinates of the nodes. Once the data has been stored in memory a relatively inexpensive preliminary calculation is performed to determine the relative proximity of the contours to each other. The method presented in this paper involves producing a digital representation of each contour and identifying contour pairs that are close to each other based on their shape and the coordinates of their nodes. This prelimnary calculation significantly reduces the computational cost of identifying any contour crossings and is discussed in section 2.1.2. Contour pairs that are deemed to be close enough to each other that there is a potential for crossing are then flagged for further testing. These contours are divided up into segments that are single valued functions of x. The segments of any one contour are then compared with those from nearby contours, checking for any points of intersection as in section 2.1.4.

2.1.2 Preliminary calculations to determine the relative proximity of contour pairs

It is clearly inefficient to check every possible pairing of contours in detail for crossings. For N contours there are a total of $(N^2 - N)/2$ contour pairs. Rather, a quick test for the relative proximity of contours to each other is used to avoid unnecessary detailed comparison of contours that are sufficiently spatially separated.

The method involves the construction of a digital picture of each of the contours on an $n_g \times n_g$ grid. From experience using a value for n_g between about 200 and 300 works well, and allows fairly accurate depiction of each contour without producing very large matrices. The construction of the digital representation of each contour will be discussed in two parts. The first part will deal with forming digital pictures for use in an algorithm that performs pairwise comparison of contours. The second part will address issues with making a much more efficient algorithm, particularly in terms of storage.

First we provide a description of an easily implemented method for checking the relative proximity of contours. We then address the issues involved in making the method more efficient in terms of storage, and thus more practical for analysing CASL data.

Digital matrix representations of each contour are produced and then stored in the three dimensional array T, with elements t_{rcj}, for $r = 1, \ldots, n_g$, $c = 1, \ldots, n_g$, $j = 1, \ldots, N$. Layer j of T contains the digital representation of contour j. Initially, T is set to be the $n_g \times n_g \times N$ zero array.

The digital representations of each contour are produced as follows, with reference to figure 3. The x and y coordinates of the nodes, (x_i, y_i), are scaled and shifted so that they lie between 1 and n_g. For contours from the standard CASL algorithm in a 2π doubly periodic domain where $-\pi \leq x \leq \pi$, $-\pi \leq y \leq \pi$ an appropriate scaling and shifting is

$$\hat{x}_i = \frac{(x_i + \pi)(n_g - 1)}{2\pi} + 1, \qquad \hat{y}_i = \frac{(y_i + \pi)(n_g - 1)}{2\pi} + 1,$$

where $(\hat{x}_i, \hat{y}_i)$ are the coordinates of node i when shifted and scaled to the domain $1 \leq x \leq n_g$, $1 \leq y \leq n_g$. The nearest integers to both $\hat{x}_i$ and $\hat{y}_i$ are calculated and stored as $\bar{x}_i$ and $\bar{y}_i$ respectively. For contour j, $t_{\bar{y}_i \bar{x}_i j}$ is set to 1. See figure 3, panels (a) to (d), to see the transition from original contour to digital representation of the nodes.

To complete the representation of each contour the straight lines joining consecutive nodes are represented by filling appropriate entries in the array T. The Bresenham line algorithm, see Bresenham (1965) and Hoff (1995), is used to determine efficiently which are the appropriate array entries to fill, if any. In essence, the entries of T lying on the straight line segment from $t_{\bar{y}_i \bar{x}_i j}$ to $t_{\bar{y}_{(i+1)} \bar{x}_{(i+1)} j}$ are set to 1, as illustrated in figure 3, panel (e).

Once all the digital representations of the contours are finalised, the relative proximity of the contours is tested as follows. Consider any two layers of T; each contains an $n_g \times n_g$ matrix representation of a different contour. The two matrices are added together and the result is searched for any entries with value 2, which would imply an overlap in the two digital representations. A pair of contours, j' and j'', are explicitly tested for crossings using the method to follow in section 2.1.4 if the matrix obtained by adding layers j' and j'' of T contains any entries equal to 2. There are $(N^2 - N)/2$ such matrix additions and subsequent search operations to perform for N contours.

In practice, comparing digital representations of contours selects many fewer pairs of contours for further scrutiny than the maximum number of pairings, $(N^2 - N)/2$. One of the advantages of the method is that it includes information about the general shape of the contours. It is also quite efficient in comparison with other methods for checking the relative proximity of contours. Figure 4 illustrates the average number of contour pairs

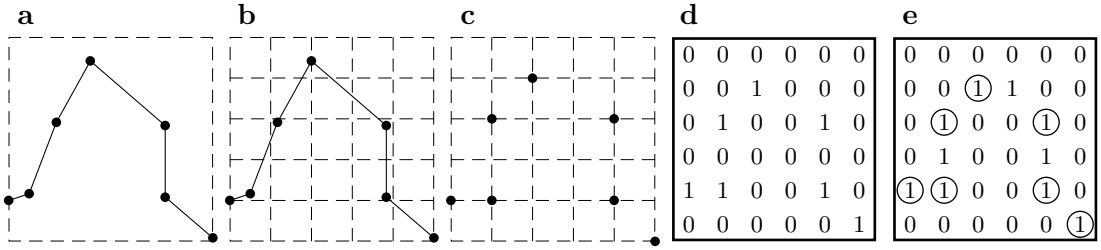

Figure 3: Part of a contour is depicted in (a). Dots are used to represent the positions of nodes. The coordinates of the nodes are scaled and shifted so that they lie between 1 and n_g (here $n_g = 6$) in both the x and y directions. The grid lines in (b) represent the integer coordinates 1 to 6 in both the x and y directions. The nearest integers to the scaled coordinates are taken, effectively moving the nodes to the nearest vertices of the grid, as depicted in (c). Then the corresponding entries in T are set to 1, as in (d). Appropriate intermediate entries are set to 1 with the aid of the Bresenham line algorithm in (e). Nodes in (e) are represented by circled entries of 1. To aid with visual representation, the rows of the matrices in (d) and (e) are drawn in ascending order from the bottom of the panels to the top.

picked out for further analysis by the present method in comparison to a method developed in Dritschel and Ambaum (1997) across twenty simulations of quasigeostrophic turbulence in a cylindrical domain. The method of Dritschel and Ambaum (1997) appears on page 1105 of their publication and for our purposes we set $\delta = 0$ in the inequalities at the top of the page in their section 2 (f). Each vortex simulation started with twenty randomly distributed circular vortices of equal radius, ten with potential vorticity 4π and the other ten vortices with potential vorticity -4π. The duration of each simulation was 20 core rotation periods. A good measure of the general efficacy of either method in reducing the number of contour pairs that require further testing is the number of contour pairs selected as a fraction of all possible contour pairing at each time step. Figure 5 provides a plot of the mean fraction of contour pairs selected by each method across all 20 simulations. It is clear from these figures that both methods are successful in substantially reducing the number of contour pairs that are further tested, with the digital representation technique consistently choosing fewer pairs. The drawback of the method of drawing digital pictures is that due to finite resolution it is possible that a pair of contours that should be explicitly tested for crossings may be missed. However, this rarely occurs and in circumstances where it does any crossings missed from one time step of CASL data are usually detected in the next. By contrast, Dritschel and Ambaum's method is a robust approach and guarantees identification of all potentially crossing contour pairs. Table 1 lists the maximum number of contours and corresponding number of contour pairs in each simulation, along with the maximum number of contour pairs selected by each technique. Computational timing tests have shown that the calculations made by both the digital representation method and the method of Dritschel and Ambaum (1997) are performed over roughly equal periods of time. In addition, both preliminary calculation techniques take up a relatively small portion of an overall calculation to identify contour crossings, with approximately 10% of the total CPU time spent on them.

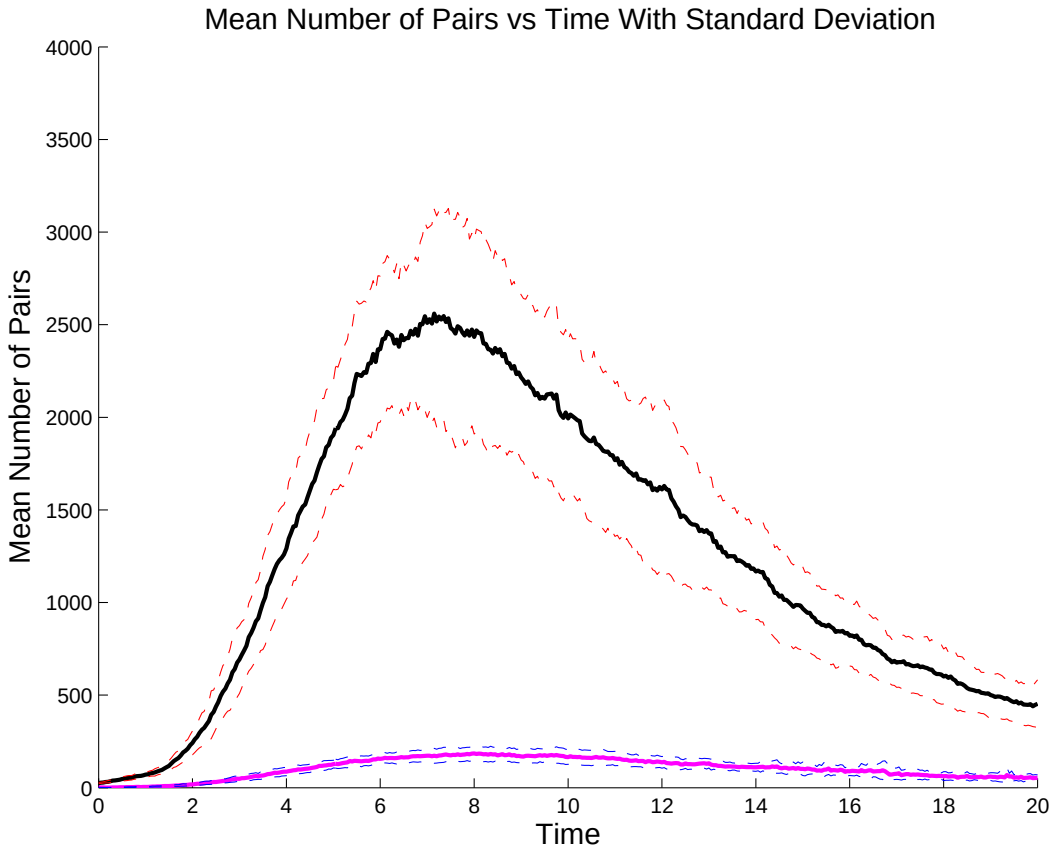

Figure 4: Average number of contour pairs selected by the method of Dritschel and Ambaum (1997) (in black) and the present digital representation method (in magenta). Dashed red and blue lines are plotted one standard deviation above and below each of these averages respectively.

2.1.3 Improved efficiency of the digital representation method

The method of producing digital representations of each contour described in section 2.1.2 is more efficient than the method of Dritschel and Ambaum (1997) at reducing the number of pairs of contours that undergo the final tests for crossings. The digital representation method as described above is quite poor in terms of storage efficiency. To be able to use this method for analysis of CASL output improvements need to be made.

Storage efficiency of the digital representation method may be improved in the following manner. An $n_g \times n_g$ matrix, $\tilde{T}$, with elements $\tilde{t}_{rc}$ is used to temporarily store the digital representation of each contour. Prior to the construction of the picture of each contour, $\tilde{T}$ is set to be the $n_g \times n_g$ zero matrix. The columns and rows of all the entries of $\tilde{T}$ that are set to 1 during the construction of a digital picture of a contour are recorded in the vectors **c** and **r** respectively. Another vector, **b**, is used to record the indices of the entries in **c** and **r** that correspond to the first point on each contour. Once the digital picture of a contour

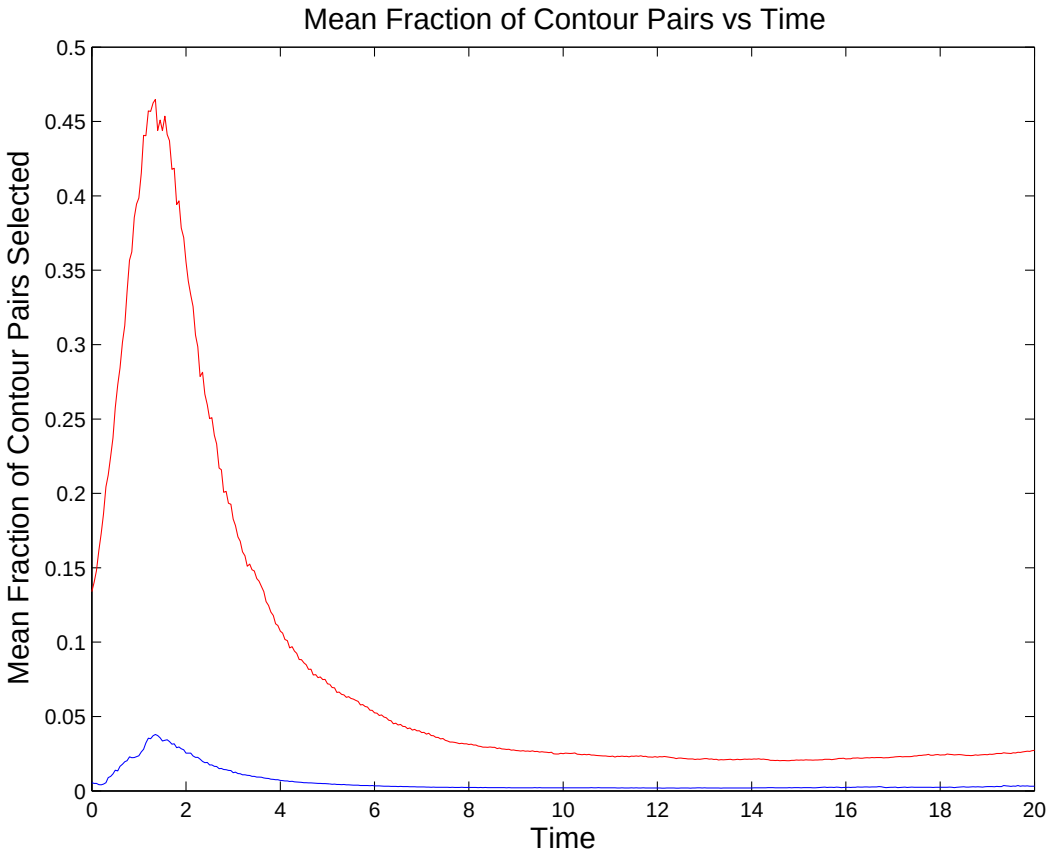

Figure 5: Mean fractional number of contour pairs selected by the method of Dritschel and Ambaum (1997) (in red) and the digital representation method (in blue).

is complete, $\tilde{T}$ is added to S, an $n_g \times n_g$ matrix which is initially set to be a zero matrix and is to hold the sum of all the digital representations of the contours. $\tilde{T}$ is then reset to an $n_g \times n_g$ zero matrix before the construction of the next contour's picture. After all the contours have been treated in this way, S is searched for entries with value greater than 1. If any such entries are found, **c**, **r** and **b** are used to determine which contours have part of their digital representation at these grid points. All pairs of contours that are found to share grid points in their digital form are then tested for crossings according to the method of section 2.1.4, which follows.

2.1.4 Points of intersection of two contours

The method used for determining if two contours cross is based on some elementary analysis to determine if two functions intersect. Consider two continuous functions, $f(x)$ and $g(x)$. If the two functions cross, then the sign of $f(x) - g(x)$ must change. For the calculation of $f(x) - g(x)$ to make sense when performed by a numerical package both functions must be known for the same set of x coordinates. There are two difficulties in implementing

Simulation Number	Maximum Contours	Maximum Pairs	Digital Rep.	Dritschel & Ambaum
1	452	101 926	244	3 953
2	540	145 530	273	3 343
3	419	87 571	218	2 969
4	402	80 601	179	3 207
5	399	79 401	227	2 789
6	335	55 945	150	2 087
7	430	92 235	230	2 678
8	385	73 920	182	2 692
9	403	81 003	211	2 399
10	442	97 461	214	3 116
11	403	81 003	182	2 911
12	339	57 291	206	1 988
13	442	97 461	238	3 948
14	442	97 461	205	2 491
15	498	123 753	230	2 837
16	501	125 250	256	3 733
17	531	140 715	305	3 404
18	449	100 576	187	2 752
19	427	90 951	269	2 676
20	331	54 615	146	2 100

Table 1: Comparison of maximum number of contour pairs over the length of each turbulence simulation with the maximum number of contour pairs selected by the digital representation method and the method of Dritschel and Ambaum (1997). The columns of the table are, from left to right, the turbulence simulation number, maximum number of contours during a simulation, the corresponding maximum number of contour pairs, the maximum number of contour pairs selected by the digital representation method and the maximum number of contour pairs selected by the method of Dritschel and Ambaum.

this approach. Firstly, in general the contours that are being tested for crossings are not functions but rather are closed curves. The second problem is that there is no regularity in the positioning of nodes in the x or y directions.

The first step is to split each contour, j, into s_j segments that are functions of x, where $j = 1, 2$. This is achieved by observing sign changes in the differences of the x coordinates of consecutive nodes on each contour. Using x_i to denote the x coordinate of node i on one of the contours, a particular node i' is identified as the last node of one segment and the first node on the next segment if $\mathrm{sgn}(x_{i'+1} - x_{i'}) \neq \mathrm{sgn}(x_{i'} - x_{i'-1})$. Here $i = 1, \ldots, n_j$ for contour j, and $x_{n_j+1} = x_1$ since all contours are closed.

Segments of contour 1 are then compared to segments of contour 2 in the search for potential crossings. Only pairs of segments for which the ranges of both x and y coordinates intersect have the potential to cross. An $s_1 \times s_2$ matrix M with elements m_{kl}, $k = 1, \ldots, s_1$, $l = 1, \ldots, s_2$, is used to record if such an intersection occurs. $m_{k'l'} = 1$ if the ranges of x and

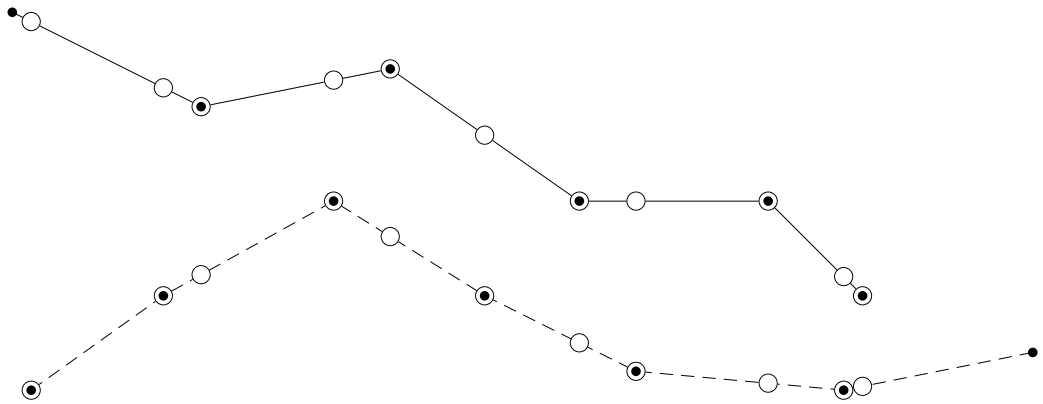

Figure 6: An illustration of two piecewise linear functions. To determine if the two functions cross as described in section 2.1.4 the y coordinates for both functions must be calculated for the same set of x coordinates. Identify the curve with the solid line as k' and the curve with the dashed line as l'. The original nodes on each curve are represented by filled black circles. The set of nodes with x coordinates given by the ordered set $X_{k'} \cup X_{l'}$ is marked with open circles on both curves.

y coordinates of segments k' and l' overlap, otherwise $m_{k'l'} = 0$.

For pairs of segments, k' and l', with $m_{k'l'} = 1$, the range of intersection in the x direction is $x_{\text{left}} \leq x \leq x_{\text{right}}$ where $x_{\text{left}} = \max(x_{k'\,\text{min}}, x_{l'\,\text{min}})$, $x_{\text{right}} = \min(x_{k'\,\text{max}}, x_{l'\,\text{max}})$, and $x_{k'\,\text{min}}$, $x_{l'\,\text{min}}$, $x_{k'\,\text{max}}$, $x_{l'\,\text{max}}$ are used to denote the minimum and maximum x values of each segment. Linear interpolation is used to find the y coordinates of both segments k' and l' for the ordered set $X_{k'} \cup X_{l'}$ where $X_{k'}$ is the set of x coordinates of the nodes belonging to k' in the interval $x_{\text{left}} \leq x \leq x_{\text{right}}$ and $X_{l'}$ is the corresponding set of nodes belonging to l'. Figure 6 illustrates a pair of contour segments where y coordinates corresponding to the set of x coordinates $X_{k'} \cup X_{l'}$ have been calculated using linear interpolation.

Once the linear interpolation has been performed the y coordinates of both segments are known for the same set of x coordinates in the interval $x_{\text{left}} \leq x \leq x_{\text{right}}$. The analysis described at the beginning of this section can then be used to determine if the two segments, and hence their corresponding contours, cross. If the segments cross it is possible to determine which nodes lie to either side of a crossing by locating the indices where there are sign changes in $f(x) - g(x)$. It is then possible to explicitly calculate the coordinates of all points of intersection of the two contours, as described below.

2.2 Area of intersection

As noted before, the assumption is made here that although contours are locally defined cubic polynomials between contiguous nodes, the departure from linearity of these cubic splines is small. The shape of the contours as well as the area of intersection can thus be well represented by simple polygons. The procedure to calculate the area in error of two crossing contours, described below, works by identifying the polygon or polygons that bound the area of intersection and then determining the area of these shapes.

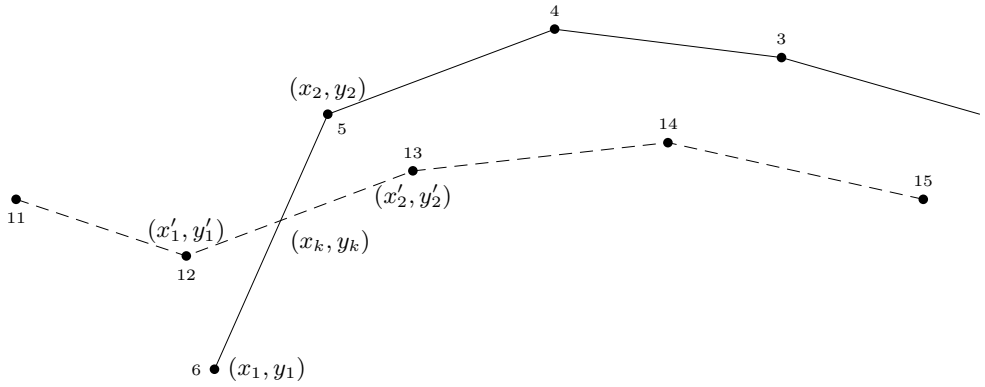

Figure 7: Illustration of notation used to represent the coordinates of the nodes to either side of a point of intersection. Identifying contour j as the curve with the solid line and contour j' as the curve with the dashed line, the algorithm to detect contour crossings would return $\mathbf{n}^1 = [6]$, $\mathbf{n}^2 = [5]$, $\mathbf{n}^3 = [12]$ and $\mathbf{n}^4 = [13]$ for the particular crossing illustrated.

2.2.1 An algorithm for calculating areas of intersection

The algorithm to detect contour crossings described in section 2.1 gives the following output. First it identifies the indices of a pair of crossing contours, j and j'. The indices of the nodes (and hence the coordinates of the nodes) immediately to the left and right of the crossing point are also provided. The indices of the nodes to the left and the right of a crossing point on contour j are stored in the vectors $\mathbf{n}^1$ and $\mathbf{n}^2$ respectively. The indices of the nodes to the left and right of a crossing point on contour j' are stored in the vectors $\mathbf{n}^3$ and $\mathbf{n}^4$ respectively. The entries in each of the vectors $\mathbf{n}^1$ to $\mathbf{n}^4$ are ordered so that element k of each vector corresponds to the kth point of intersection of contours j and j'.

The points of intersection of contours j and j' are calculated. With reference to figure 7, let (x_1, y_1) and (x_2, y_2) be the coordinates of the nodes immediately to the left and right respectively of a crossing on contour j. Let (x'_1, y'_1) and (x'_2, y'_2) be the coordinates of the nodes to the left and right of the same crossing on contour j'. The coordinates of the point of intersection, (x_k, y_k), are then given by

$$x_k = \frac{b' - b}{m - m'} \qquad \text{and} \qquad y_k = \frac{mb' - m'b}{m - m'}.$$

where

$$m = \frac{y_2 - y_1}{x_2 - x_1}, \quad b = y_1 - mx_1, \quad m' = \frac{y'_2 - y'_1}{x'_2 - x'_1} \quad \text{and} \quad b' = y'_1 - m'x'_1.$$

It is necessary to insert dummy nodes a small distance ε to either side of each crossing on both contours to avoid some problems associated with calculating the area of overlap that occur with specific types of crossings. The indices in $\mathbf{n}^1$, $\mathbf{n}^2$, $\mathbf{n}^3$ and $\mathbf{n}^4$ are adjusted to

identify the newly inserted dummy nodes as the nodes that are immediately before and after a crossing.

The polygon that bounds the area of intersection of contours j and j' is made up of the nodes on contour j that lie inside contour j', the nodes on contour j' that lie inside j and the points of intersection of the two contours. To determine if a point (x_i, y_i) lies inside or outside a contour, which is treated as a polygon, a horizontal line is drawn from (x_i, y_i) to some point (x_e, y_e) which lies to the right of the maximum x coordinates of both j and j'. If the horizontal line crosses a contour an odd number of times then (x_i, y_i) lies inside the contour, an even number of crossings indicates that (x_i, y_i) lies outside the contour. The number of crossings of the horizontal line is determined using the same algorithm that is used to find contour crossings. Information about the nodes on one contour lying inside or outside the other contour is stored in the vectors $\mathbf{L}_1$ and $\mathbf{L}_2$. If node i of contour j is inside contour j' then $\mathbf{L}_1(i) = 1$, otherwise $\mathbf{L}_1(i) = 0$. Similarly if node i of contour j' lies inside contour j then $\mathbf{L}_2(i) = 1$, otherwise $\mathbf{L}_2(i) = 0$. The notation $\mathbf{a}(i)$ is being used here to denote the ith element of the vector $\mathbf{a}$.

From the vectors $\mathbf{L}_1$ and $\mathbf{L}_2$ it is possible to identify "chains" of consecutive nodes that lie inside either one of the contours. The polygon that bounds the area of intersection is made by pasting these chains of nodes together at crossing points. The "head" of a chain of nodes is identified as the node with the lowest index in a set of consecutive nodes that lie inside a contour. The "tail" of a chain of nodes is identified as the node with the highest index in a set of consecutive nodes that lie inside a contour. The indices of head and tail nodes on each contour are identified by forming the differences $\mathbf{L}_1(i+1) - \mathbf{L}_1(i)$ for $i = 1, \ldots, n_j - 1$ and $\mathbf{L}_2(i+1) - \mathbf{L}_2(i)$ for $i = 1, \ldots, n_{j'} - 1$ where n_j is the number of nodes on contour j and $n_{j'}$ is the number of nodes on contour j'. If $\mathbf{L}_k(i+1) - \mathbf{L}_k(i) = 1$ then node $i+1$ is the head of a chain of nodes, if $\mathbf{L}_k(i+1) - \mathbf{L}_k(i) = -1$ then node i represents the tail of a chain of nodes (this is true for either contour, hence the use of the subscript k). The indices of heads and tails of node chains on contour j are stored in the vectors $\mathbf{h}_1$ and $\mathbf{t}_1$ respectively. Indices of heads and tails on contour j' are stored in $\mathbf{h}_2$ and $\mathbf{t}_2$. In general the index of the head of a chain should be less than that of the tail, except in the case when a chain includes the last and first listed nodes on a contour (contours are closed, so that node $n_j + 1$ corresponds to the first node on contour j for example).

The information in the vectors $\mathbf{n}^1$, $\mathbf{n}^2$, $\mathbf{n}^3$, $\mathbf{n}^4$, $\mathbf{h}_1$, $\mathbf{t}_1$, $\mathbf{h}_2$ and $\mathbf{t}_2$ is used to identify a correct order for joining all the node chains together. The vital parts of this information are stored in a matrix, A, that will be referred to as the connection matrix here. Initially the connection matrix is set to be a $n_k \times 4$ matrix of zeroes where n_k is the number of points of intersection between contours j and j'. Non-zero entries in the connection matrix are filled according to the following rules. The notation A_{nm} is used to denote the entry in row n, column m of the matrix A.

$$
\begin{array}{llll}
\text{If} & \mathbf{n}^1(k) = \mathbf{h}_1(l) & \text{then} & A_{k1} = 2l - 1 \\
\text{If} & \mathbf{n}^1(k) = \mathbf{t}_1(l) & \text{then} & A_{k1} = 2l \\
\text{If} & \mathbf{n}^2(k) = \mathbf{h}_1(l) & \text{then} & A_{k2} = 2l - 1 \\
\text{If} & \mathbf{n}^2(k) = \mathbf{t}_1(l) & \text{then} & A_{k2} = 2l \\
\text{If} & \mathbf{n}^3(k) = \mathbf{h}_2(l) & \text{then} & A_{k3} = 2l - 1 \\
\text{If} & \mathbf{n}^3(k) = \mathbf{t}_2(l) & \text{then} & A_{k3} = 2l \\
\text{If} & \mathbf{n}^4(k) = \mathbf{h}_2(l) & \text{then} & A_{k4} = 2l - 1 \\
\text{If} & \mathbf{n}^4(k) = \mathbf{t}_2(l) & \text{then} & A_{k4} = 2l
\end{array}
\tag{1}
$$

Columns 1 and 2 of the connection matrix contain information about contour j, columns 3 and 4 contain information about contour j'. Element k of $\mathbf{n}^i$ corresponds to the kth point of intersection of contours j and j'.

A polygon that bounds an area of intersection should be made up of an even number of node chains and intersection points. An equal number of node chains should come from each contour, and the node chains should be joined together at intersection points such that a chain from contour j should be followed immediately by a chain from contour j' and a chain from contour j' should be followed by a chain from contour j. If a pair of contours cross at two points, for example, then the polygon that bounds the area of intersection will be made up of two points of intersection and two chains of nodes, one from contour j and the other from contour j'.

The connection matrix is used to determine the order in which the various chains of nodes are put together and whether each chain should be connected in ascending or descending nodal order. Even numbers in the connection matrix correspond to the tails of node chains, odd numbers correspond to the heads of node chains. An important point to note is that it is often the case that a pair of crossing contours, j and j', intersect in several places and that the area of intersection of the two contours is made up of more than just one polygon, as illustrated in figure 8. This is taken into account in the following algorithm presented as pseudocode for traversing the connection matrix. As the matrix is traversed the x and y coordinates of each polygon that bounds the area of intersection are stored in the vectors $\mathbf{p}_x$ and $\mathbf{p}_y$ respectively.

Initialisation

Initialise a vector $\mathbf{v}$ with n_k elements as a zero vector.

Start with row 1 of the connection matrix, this corresponds to the first recorded point of intersection of j and j'. Store the (x, y) coordinates of the first listed crossing in $\mathbf{p}_x$ and $\mathbf{p}_y$. Record that row 1 of A has been visited by setting $\mathbf{v}(1) = 1$ and that the starting row, $s_r = 1$.

Search columns 1 and 2, row 1 of A for a non-zero entry, m.

$(*)$
if m is even (if $m \mod 2 = 0$)

> Store the coordinates of chain $m/2$ of contour j in $\mathbf{p}_x$ and $\mathbf{p}_y$ in descending order.
>
> Search columns 1 and 2 of A for $m - 1$. Denote the row that contains $m - 1$ as n.

else if m is odd (if $m \mod 2 = 1$)

> Store the coordinates of chain $(m + 1)/2$ of contour j in $\mathbf{p}_x$ and $\mathbf{p}_y$ in ascending order.
>
> Search columns 1 and 2 of A for $m + 1$. Denote the row that contains $m + 1$ as n.

end if

Insert the coordinates of crossing n in $\mathbf{p}_x$ and $\mathbf{p}_y$.

Set $\mathbf{v}(n) = 1$, indicating that row n has been visited.

Search columns 3 and 4, row n of A for a non-zero entry, m'.

if m' is even (if $m' \mod 2 = 0$)

> Store the coordinates of chain $m'/2$ of contour j' in $\mathbf{p}_x$ and $\mathbf{p}_y$ in descending order.
>
> Search columns 3 and 4 of A for $m' - 1$. Denote the row that contains $m' - 1$ as n'.

else if m' is odd (if $m' \mod 2 = 1$)

> Store the coordinates of chain $(m' + 1)/2$ of contour j' in $\mathbf{p}_x$ and $\mathbf{p}_y$ in ascending order.
>
> Search columns 3 and 4 of A for $m' + 1$. Denote the row that contains $m' + 1$ as n'.

end if

Insert the coordinates of crossing n' in $\mathbf{p}_x$ and $\mathbf{p}_y$.

Set $\mathbf{v}(n') = 1$, indicating that row n' has been visited.

if $n' = s_r$ (the algorithm has returned to the starting row)

> A closed polygon has been formed. The area of the polygon is then calculated and recorded.
>
> if all the entries of $\mathbf{v}$ equal 1
>
> > All the rows of A have been visited, and the algorithm terminates.
>
> else
>
> > Find the index, n'', of the first zero entry of $\mathbf{v}$, corresponding to an unvisited row of the matrix A. A new polygon is to be constructed.
> >
> > Set $s_r = n''$.
> >
> > Insert the coordinates of crossing n'' in $\mathbf{p}_x$ and $\mathbf{p}_y$.
> >
> > Search columns 1 and 2, row n'' of A for a non-zero entry, m, then return to $(*)$.
>
> end

else

> Search columns 1 and 2, row n' of A for a non-zero entry m, then return to $(*)$.

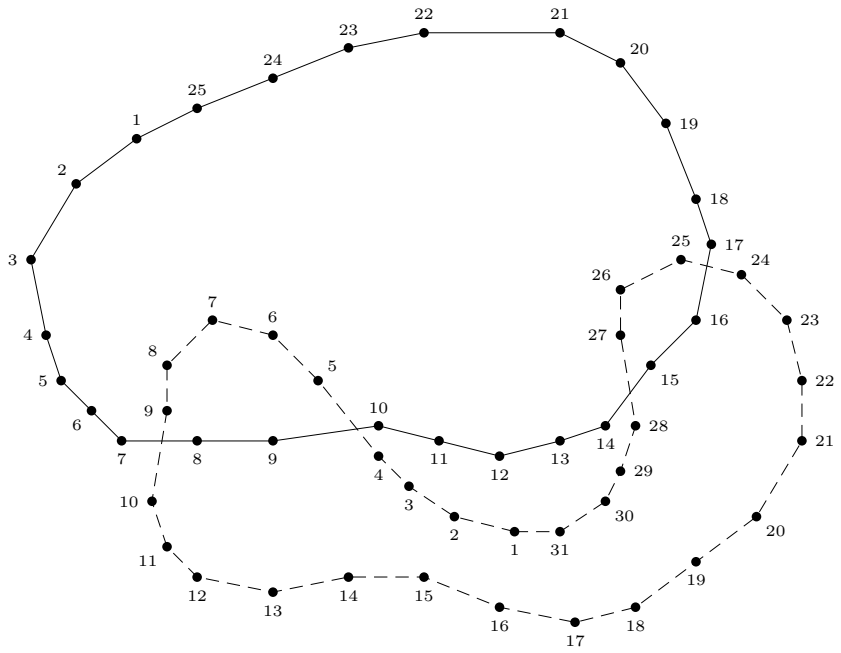

Figure 8: An example of crossing contours where the area of intersection of the two contours is made up of two polygons. Contour j is represented by the solid line. Contour j' is represented by the dashed line.

end if

Using p_x^i to denote the ith element of $\mathbf{p}_x$ and p_y^i to denote the ith element of $\mathbf{p}_y$, the area of each polygon that describes an area of intersection, P_A, is given by

$$P_A = \left| -\frac{1}{2} \sum_{i=1}^{n_p} \left(p_x^{i+1} - p_x^i\right) \left(p_y^{i+1} + p_y^i + 2c\right) \right| \tag{2}$$

where n_p is the number of vertices of the polygon and c is a constant chosen such that $p_y^i + c \geq 0$ for $i = 1, \ldots, n_p$. It is assumed that node indices increase as the polygon is traversed in the anticlockwise direction, however it is possible that polygons with node indices increasing in the clockwise direction will be identified using the above algorithm so the absolute value of the area is calculated.

The following example shows how the algorithm described in this section is utilised to find the area of intersection of the contours illustrated in figure 8. Identify contour j as the contour with the solid line and contour j' as the contour with the dashed line. There are four points of intersection between the two contours and the area of intersection is made up of two polygons.

The algorithm that identifies contour crossings stores the node indices to the left and right of the points of intersection on contour j in $\mathbf{n}^1$ and $\mathbf{n}^2$ respectively. The corresponding node

indices for contour j' are stored in $\mathbf{n}^3$ and $\mathbf{n}^4$, so that

$$\mathbf{n}^1 = \begin{bmatrix} 7 \\ 9 \\ 14 \\ 16 \end{bmatrix}, \quad \mathbf{n}^2 = \begin{bmatrix} 8 \\ 10 \\ 15 \\ 17 \end{bmatrix}, \quad \mathbf{n}^3 = \begin{bmatrix} 10 \\ 5 \\ 27 \\ 25 \end{bmatrix}, \quad \text{and} \quad \mathbf{n}^4 = \begin{bmatrix} 9 \\ 4 \\ 28 \\ 24 \end{bmatrix}$$

and the coordinates of each point of intersection are calculated. Whether a node lies to the left or right of a crossing is determined by examining its x coordinate.

The heads and tails of chains of nodes of contour j that lie inside contour j' and the heads and tails of chains of nodes of j' that lie inside j are stored in $\mathbf{h}_1$, $\mathbf{t}_1$, $\mathbf{h}_2$ and $\mathbf{t}_2$ respectively. They are

$$\mathbf{h}_1 = \begin{bmatrix} 8 \\ 15 \end{bmatrix}, \quad \mathbf{t}_1 = \begin{bmatrix} 9 \\ 16 \end{bmatrix}, \quad \mathbf{h}_2 = \begin{bmatrix} 5 \\ 25 \end{bmatrix}, \quad \mathbf{t}_2 = \begin{bmatrix} 9 \\ 27 \end{bmatrix}$$

Chain 1 of contour j is comprised of nodes 8 and 9, chain 2 of j is made up of nodes 15 and 16. Chain 1 of contour j' extends from node 5 to node 9, chain 2 of j' is made up of the nodes from 25 to 27.

Using (1), the connection matrix A is

$$A = \begin{bmatrix} 0 & 1 & 0 & 2 \\ 2 & 0 & 1 & 0 \\ 4 & 0 & 3 & 0 \\ 0 & 3 & 4 & 0 \end{bmatrix}$$

Polygons are then constructed by traversing the connection matrix.

The x and y coordinates of the first listed crossing are stored in $\mathbf{p}_x$ and $\mathbf{p}_y$. $\mathbf{v}(1)$ is set to 1, indicating that row 1 has been visited and $s_r = 1$, indicating that the starting row is row 1. Columns 1 and 2 of row 1 are searched for a non-zero entry, in this case $A_{12} = 1$, corresponding to the head of chain 1 of contour j. Chain 1 of contour j is inserted in $\mathbf{p}_x$ and $\mathbf{p}_y$ with nodes in ascending order. Columns 1 and 2 of A are searched for the entry 2 which corresponds to the tail of chain 1 of j. $A_{21} = 2$, so the second listed crossing is inserted in $\mathbf{p}_x$ and $\mathbf{p}_y$. $\mathbf{v}(2)$ is set to 1, indicating that row 2 has been visited. Columns 3 and 4, row 2 are searched for a non-zero entry. $A_{23} = 1$, corresponding to the head of chain 1 of contour j'. Chain 1 of j' is inserted in $\mathbf{p}_x$ and $\mathbf{p}_y$ with nodes in ascending order. Columns 3 and 4 are searched for the entry 2, corresponding to the tail of chain 1 of j'. $A_{14} = 2$. The coordinates of crossing 1 are inserted in $\mathbf{p}_x$ and $\mathbf{p}_y$. The current row, row 1, is the same as the starting row, $s_r = 1$, indicating that a closed polygon has been formed. The area of this polygon is calculated according to equation (2).

Currently,

$$\mathbf{v} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

indicating that rows 1 and 2 of the connection matrix have been visited, but that rows 3 and 4 have not.

Row 3 is selected as the new starting row with, $s_r = 3$, and $\mathbf{v}(3)$ is set to 1. A new polygon is started with the coordinates of the third listed crossing point inserted as the first

point in $\mathbf{p}_x$ and $\mathbf{p}_y$. Columns 1 and 2, row 3, are searched for a non-zero entry and in this case $A_{32} = 3$ which corresponds to the head of chain 2 of j. The nodes of chain 2 of j are inserted in $\mathbf{p}_x$ and $\mathbf{p}_y$ in ascending order. The algorithm continues with the remaining parts of the second polygon made up of crossing 4, the nodes of chain 2 of j' in ascending order, then crossing 3 again. The area of the second closed polygon is calculated. As all the rows of A have been visited,

$$\mathbf{v} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

and the algorithm terminates.

2.3 Area of intrusion or extrusion

The algorithm described in section 2.2 deals specifically with the case where two contours are crossing and the area in error is the area of intersection of the two contours. In other words the area of intrusion of one contour into the area bounded by another contour is the area in error. In more general cases the area in error when two contours cross is not necessarily the area of intersection, but could be the area of the region where one contour protrudes through another. This is the case in particular when dealing with nested contours, such as those used to give a discrete representation of parabolic or Gaussian distributions of potential vorticity. Consider for example the pair of crossing contours illustrated in figure 9. Identify contour j as the contour with the solid line and contour j' as the contour marked with a dashed line. Assuming that contour j' was initially entirely inside contour j the area in error is the solid shaded region in the diagram. The polygon that bounds this area in error is comprised of the two points of intersection of the contours, the nodes of contour j that lie inside contour j' and the nodes of contour j' that lie *outside* contour j. In general, when calculating an area of extrusion node chains belonging to one of the contours (in this example, contour j') are made up of nodes that lie outside the other contour (contour j).

In practice it is not easy to determine whether the solid shaded region or the crosshatched region depicted in figure 9 represent the true area in error. To deal with this problem the area of both the shaded polygons depicted is calculated, it is then assumed that the smaller of the two areas calculated corresponds to the correct area in error. As described in section 2.2, it is possible that the area in error for a pair of crossing contours will be comprised of multiple polygons. When this is the case and the area in error is an area of extrusion more care needs to be taken in determining which is most likely to be the region in error. First the sum of the areas of the polygons comprised of nodes from contour j that lie inside j' and the nodes from j' that lie outside j, denoted by $\mathcal{A}_1$, is calculated. The sum of the areas of the polygons made up of nodes from j that lie outside j' and the nodes from j' that lie inside j, $\mathcal{A}_2$, is also calculated. The area in error is most likely to be the minimum of $\mathcal{A}_1$ and $\mathcal{A}_2$, and is described by the polygons that bound the minimum total area.

3 Some Results - Analysis of CASL Data

In figures 10 and 11 we present some preliminary results of analysis of contour crossings that occur in two CASL simulations of quasigeostrophic turbulence in a single layer fluid of finite

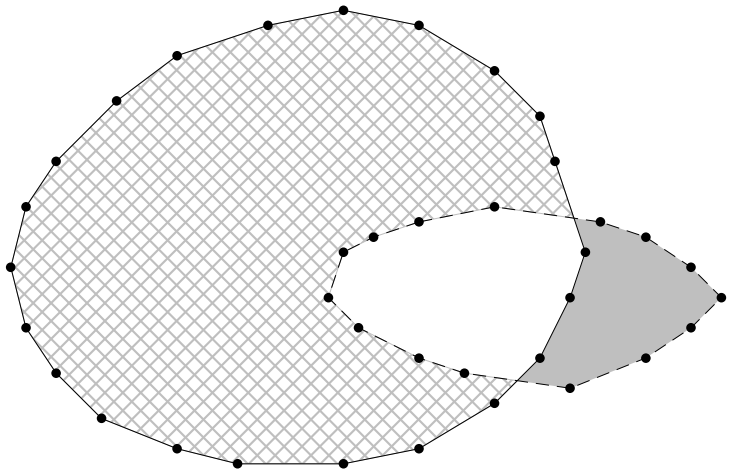

Figure 9: A pair of crossing contours used to illustrate the process of calculating the area of extrusion of contour j', represented by the dashed line, through contour j, represented by the solid line. For the purpose of this example it has been assumed that contour j' lay entirely inside contour j initially. The area in error is the solid shaded region.

depth in a cylindrical domain. For the two cases analysed here turbulence was simulated by observing the motion of twenty vortices initially randomly distributed throughout the cylindrical domain. Ten of the vortices were cyclonic with corresponding potential vorticity 4π, the other ten were anticyclonic with potential vorticity -4π. The vortices were initially circular, all with radii of approximately 0.1118 units so that initially they covered 25% of the domain. The vortices were also initially distributed so that the minimum separation of vortex centres was twice the vortex radius. In addition, the vortices were positioned randomly subject to the condition that the initial radial angular momentum profile be zero throughout the domain. Time stepping was performed by the CASL algorithm's 4th order Runge-Kutta time integration scheme with time step $\Delta t = 0.01$, which corresponds to 1/100th of a core rotation period for the vortices. Each simulation was run for a period of twenty core rotations, with contour information output every fifth timestep, or every 1/20th of a core rotation period.

Determination of the velocity field was performed on a grid of points equally spaced in the radial and azimuthal directions. In the radial direction $n_r = 64$ grid points were used and $n_\theta = 128$ grid points were used in the azimuthal direction. Potential vorticity contour to grid conversion was performed on a grid twice as fine in both radial and azimuthal directions. The large scale characteristic length scale of the flow, L, was chosen to be the initial diameter of each of the vortices, that is $L = 0.2236$. The dimensionless node separation parameter, μ, was chosen so that the surgical scale, δ, was one eighth of the radial inversion grid spacing. δ is related to μ and L through the equation $\delta = \frac{1}{4}\mu^2 L$. This leads to $\mu = 0.132714$.

The quantities illustrated in panels (a) to (h) of figures 10 and 11 are as follows: (a) the total number of contours, (b) the total number of nodes, (c) the total are bounded by the contours, (d) the number of intersecting line segments detected, (e) the number of polygons that describe the area in error, (f) the mean area of the polygons that describe the area in error, (g) the total area in error and (h) the total area in error as a fraction of the total area

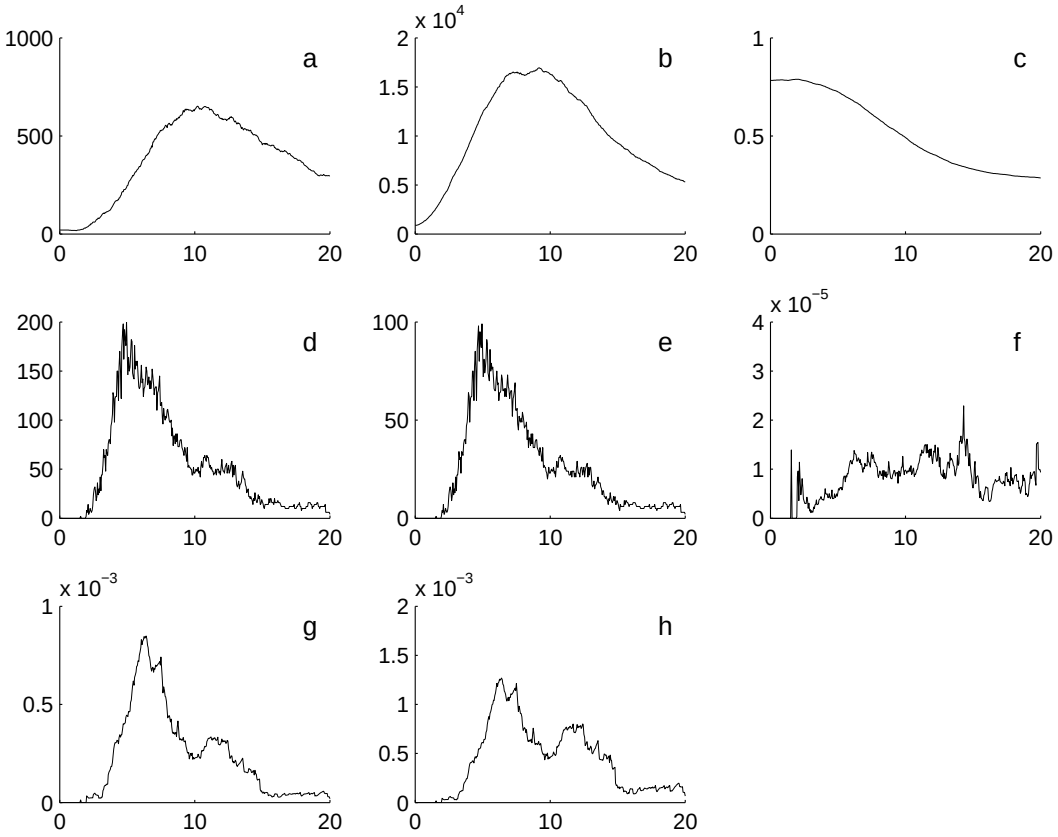

Figure 10: Crossing analysis results for a CASL simulation of single layer quasigeostrophic turbulence in a cylindrical domain.

bounded by the contours, all as a function of time. The areas in error for the turbulence simulations calculated here are treated as an area of intersection or extrusion according to the following rules that have been determined based on the initial orientation of the contours and the potential effects of contour surgery. If a pair of crossing contours are oriented in the same sense, either clockwise or anticlockwise, then the area of intersection is treated as the area in error. If the crossing contours are oriented in opposite senses then the smallest area of extrusion, as discussed in section 2.3, is treated as the area in error. A more detailed discussion of the choice of the correct area in error is provided in Schaerf (2006) on pages 89 to 96.

The preliminary results presented in figures 10 and 11 show that the number of crossings and the total area in error grow during the earlier stages of the simulations, roughly following the growth in the number of nodes and hence the general complexity of the flow. Thereafter the number of crossings decreases, due in large part to the action of contour surgery in removing very small scale features from the simulation. For the simulations considered here the area in error due to contour crossings is small, both in absolute terms and in relative terms where the error is at most approximately 1/1000 of the total area bounded by the contours. Visual inspection of the crossing contours has been used to verify that the results presented here are accurate.

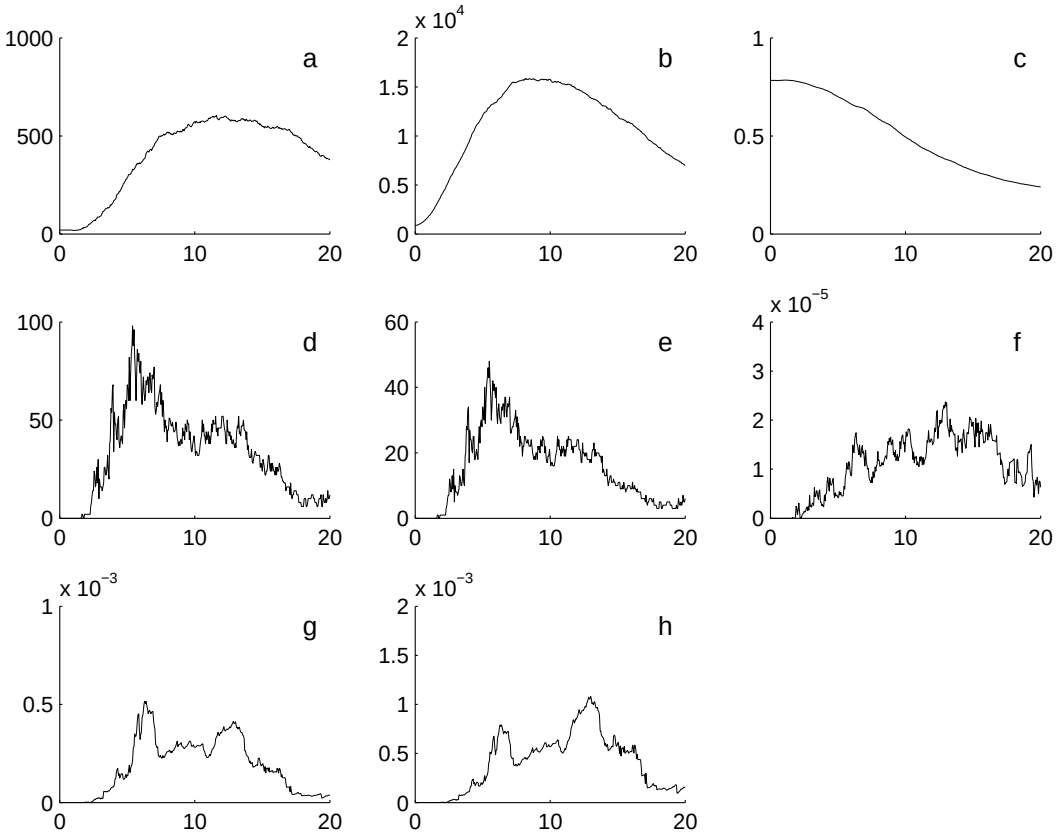

Figure 11: Crossing analysis results for a second CASL simulation of single layer quasigeostrophic turbulence in a cylindrical domain.

4 Conclusions

Here we have presented a general algorithm to quantify contour crossing errors that appear in various contour-advective simulations. The algorithm works in several stages. The first step is to perform a simple pairwise comparison of all contours to determine their relative proximity to each other. This comparison is achieved by producing digital representations of each contour, a method that has proven to be efficient compared to the method of Dritschel and Ambaum (1997). Contour pairs identified as being close to each other by the digital representation method are then closely scrutinised for contour crossings. Some elementary analysis of functions is used to identify any points of intersection and the nodes that lie to either side of a crossing point. The information relating to any points of intersection is then used to help identify polygons that closely approximate the area in error.

Some preliminary results have shown that the contour crossing errors that occur in CASL simulations of single layer quasigeostrophic turbulence in a cylindrical domain are small. However, this is not always the case. Further study has shown that depending on the flow being simulated the error due to contour crossings can be quite significant. This is particularly the case when simulating a flow that has many different layers of potential vorticity. Part 2 of this study provides an in depth analysis of the nature of contour crossing that occur in single

layer quasigeostrophic turbulence simulations and in simulations of vortex motion on the β-plane. The effects of varying the spatial resolution of CASL simulations on contour crossings are discussed as are two methods for preventing the onset and growth of such crossings.

References

E. S. Benilov, J. Nycander, and D. G. Dritschel. Destabilization of barotropic flows by small-scale topography. *J. Fluid Mech.*, 517:359–374, 2004.

H. L. Berk and K. V. Roberts. The Water-Bag Model. *Methods Comput. Phys.*, 9:87–134, 1970.

J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

D. G. Dritschel. Contour Surgery: A Topological Reconnection Scheme for Extended Integrations Using Contour Dynamics. *J. Comput. Phys.*, 77:240–266, 1988.

D. G. Dritschel. Contour Dynamics and Contour Surgery: Numerical Algorithms for Extended, High-Resolution Modelling of Vortex Dynamics in Two-Dimensional, Inviscid, Incompressible Flows. *Comput. Phys. Rep.*, 10:77–146, 1989.

D. G. Dritschel. A fast contour dynamics method for many-vortex calculations in two-dimensional flows. *Phys. Fluids A*, 5(1):173–186, 1993.

D. G. Dritschel. A fast hybrid algorithm for the simulation of fine-scale conservative fields on a sphere. 1999.

D. G. Dritschel and M. H. P. Ambaum. A contour-advective semi-lagrangian numerical algorithm for simulating fine-scale conservative dynamical fields. *Q. J. R. Meteorol. Soc.*, 123:1097–1130, 1997.

D. G. Dritschel, L. M. Polvani, and A. R. Mohebalhojeh. The contour-advective semi-Lagrangian algorithm for the shallow water equations. *Mon. Weather Rev.*, 127(7):1151–1165, 1999.

D. G. Dritschel and R. Saravanan. Three-dimensional quasi-geostrophic contour dynamics, with an application to stratospheric vortex dynamics. *Q. J. R. Meteorol. Soc.*, 120:1267–1297, 1994.

D. G. Dritschel, R. K. Scott, C. Macaskill, G. A. Gottwald, and C. V. Tran. A unifying scaling theory for vortex dynamics in two-dimensional turbulence. *submitted for publication to Phys. Rev. Lett.*, 2008.

K. Hoff. Derivation of Bresenham's line algorithm.
[Online] http://www.cs.unc.edu/∼hoff/projects/comp235/bresline/bresen.html, 1995.

C. Macaskill, W. E. P. Padden, and D. G. Dritschel. The CASL algorithm for quasi-geostrophic flow in a cylinder. *J. Comput. Phys.*, 188:232–251, 2003.

T. M. Schaerf. On Contour Crossings in Contour-Advective Simulations of Geophysical Fluid Flows. *PhD Thesis, University of Sydney, Australia*, 2006.

T. M. Schaerf and C. Macaskill. Detecting contour crossings in contour dynamical and contour-advective semi-lagrangian simulations. *ANZIAM Journal*, 45(E):C693–C712, 2004. Proceedings of the 11th Computational Techniques and Applications Conference, Electronic supplement, Jagoda Crawford and A. J. Roberts editors.

E. Chacón Vera and T. Chacón Rebollo. On cubic spline approximations for the vortex patch problem. *App. Num. Math.*, 36:359–387, 2001.

A. Viúdez and D. G. Dritschel. An explicit potential vorticity conserving approach to modelling nonlinear internal gravity waves. *J. Fluid Mech.*, 458:75–101, 2002.

P. W. C. Vosbeek, H. J. H Clercx, and R. M. M. Mattheij. Acceleration of Contour Dynamics Simulations with a Hierarchical-Element Method. *J. Comput. Phys.*, 161:287–311, 2000.

P. W. C. Vosbeek and R. M. M. Mattheij. Contour Dynamics with Symplectic Time Integration. *J. Comput. Phys.*, 133:222–234, 1997.

D. W. Waugh and R. A. Plumb. Contour Advection with Surgery: A Technique for Investigating Finescale Structure in Tracer Transport. *J. Atmos. Sci.*, 51(4):530–540, 1994.

N. J. Zabusky, M. H. Hughes, and K. V. Roberts. Contour Dynamics for the Euler Equations in Two Dimensions. *J. Comput. Phys.*, 30:96–106, 1979.