Detecting contour crossings in contour dynamical and contour-advective semi-Lagrangian simulations

T. M. Schaerf* C. Macaskill†

(Received 9 August 2003; revised 23 January 2004)

Abstract

Contour dynamics and contour-advective methods are commonly used numerical techniques for simulating inviscid fluid motions. In these methods the vorticity or potential vorticity of a flow is represented by a series of contours which are advected according to the prevailing velocity field. In some circumstances the contours may cross, eroding the accuracy of the numerical solution and violating the equations of motion. This paper describes an automated method for explicitly revealing such crossings, first considering the case of determining if two contours cross and then later the more general case of determining if and where an arbitrary number of contours cross.

*School of Mathematics and Statistics, University of Sydney, Sydney, NSW 2006, Australia. mailto:tschaerf@maths.usyd.edu.au

†School of Mathematics and Statistics, University of Sydney, Sydney, NSW 2006, Australia.

See http://anziamj.austms.org.au/V45/CTAC2003/Scha/home.html for this article, Published July 29, 2004. ISSN 1446-8735

Contents

1 Introduction

Contour dynamics is a commonly used numerical method for simulating inviscid fluid motions, particularly vortex motion. This method is based on the earlier "water-bag" model for solving the Vlasov equation that appears in plasma physics [1] and was originally introduced in the context of simulating two dimensional fluid motions governed by the barotropic vorticity equation in a domain of infinite extent [12]. It has since been modified to deal with

vortical motions on the surface of a sphere or cylinder as well as multilayer quasigeostrophic flows [5, 6], amongst others. One of the most appealing features of contour dynamics is its ability to resolve fine-scale features, such as steep gradients in the vorticity or potential vorticity (PV) of a flow.

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 the PV and between the contours the PV is uniform. The contours are discretely represented by nodes, usually most densely placed in regions of high local curvature, and by some interpolating function between the nodes. Both node distribution and the type of interpolating function used vary [3, 4, 5, 10, 12]. The nodes on the contours are advected at each time step according to the prevailing velocity field of a flow, which is calculated by evaluating 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 flow.

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

Contour surgery [4, 5] was introduced to treat the development of fine-scale structures, and thus help reduce the growth in the number of nodes, in a consistent manner. It effectively removes features in the PV field that form below a predefined surgical scale, δ, which is chosen consistently with the method of node distribution described in [4] and [5].

The Contour-Advective Semi-Lagrangian (CASL) algorithm [7] seeks to retain the advantageous features of contour dynamics and contour surgery,

whilst removing the $\mathcal{O}(n^2)$ dependence of the velocity calculation. It was developed after the success of the diagnostic tool known as contour advection with surgery [11]. As with contour dynamics the PV field is represented by a series of contours, however the velocity field of a flow is not calculated by evaluation of contour integrals. Rather, for each time step required for temporal integration, the information about the PV field is transferred from the contours to a fine grid. The PV 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, which are advected according to a 4th order Runge-Kutta time integration scheme. The essential technical issue needed for such a method to work efficiently has been resolved in [7] which is an $\mathcal{O}(n)$ scheme for transferring from the Lagrangian contoured representation of the PV 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 on the contours.

The CASL method is rapidly gaining popularity as an efficient method for simulation of various fluid flows with recent advances such as the development of the CASL algorithm for cylindrical geometry [9]. However, an error that was originally present in contour dynamics has persisted. In some cases contours may cross, see Figure 1, most often after contours that represent the boundaries of different levels of potential vorticity have become very close to each other. Methods for trying to prevent such crossings from occurring are given in [4, 5, 10] and an explicit example of contour crossing in a contour dynamical simulation is given in [3]. The problem seems to be a result of inadequate spatial resolution of the contours, particularly at points of close approach with other contours. In some cases the effects of contour crossings may be considered benign, but in other cases, such as in simulations of vortical motion on the β-plane this is almost certainly not the case.

The problem addressed by this paper is as follows. Given N closed contours, from a time step of a contour dynamics or CASL simulation, each

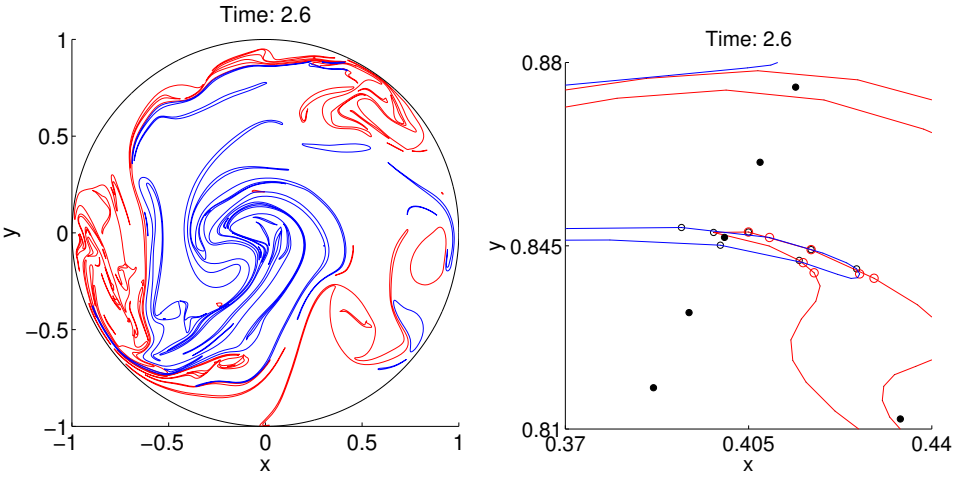


FIGURE 1: The leftmost diagram illustrates a time step from a two dimensional turbulence simulation in a cylinder. The rightmost diagram illustrates some contour crossings detected at the top right of the leftmost diagram using the method described in the text. Black dots indicate the grid points where the velocity is determined. Black circles represent nodes to the left of a crossing, red circles represent nodes to the right.

discretely represented by n_j nodes, $j = 1, \ldots, N$, whose positions are given in Cartesian (x, y) coordinates, and assuming linear interpolation between the nodes, to explicitly determine the location of all contour crossings, if any exist. The purpose of addressing this problem is to attempt to determine the cause of such crossings and ultimately develop a method to prevent or at least reduce the number of them.

Section 2 describes a method for finding any crossings that occur between two contours. In Section 3 the general case of finding all crossings that occur between N contours is considered. The methods described in Section 3 involve using a rough test to determine which pairs of contours are close enough that they may potentially cross, and then utilising the method of Section 2 on pairs of contours considered close enough to explicitly report any crossings. Section 4 contains brief comments on possible improvements and extensions of the algorithms described in Section 3.

2 The Two Contour Problem

The method for determining if two contours cross utilises the following idea. Consider two functions, $f(x)$ and $g(x)$. If the sign of $f(x) - g(x)$ changes at any point, then $f(x)$ and $g(x)$ must cross. When dealing with numerical data, the values of $f(x)$ and $g(x)$ must be known for the same values of x to determine if two functions cross. The difficulty in using this idea to determine if two contours cross arises from two things. Firstly, in general the contours that are being tested for crossings are not functions, but rather closed curves. Secondly, there is no regularity in the positioning of nodes in the x or y directions, so the y coordinates of each contour are not in general known for the same set of x coordinates.

The first step in determining if two contours cross is to split the jth contour, into s_j parts that are single valued functions ('segments') of x, for $j = 1, 2$. This is achieved by looking for sign changes in the differences of

the x coordinates of consecutive nodes on each contour.

Only those pairs of single valued segments for which both the ranges of x and y coordinates overlap have the potential to cross, see Figure 2(c). To determine if there is any such overlap in the ranges of x and y coordinates of a given pair of segments, their extreme x and y values are compared in turn. Segments that are suitably well separated cannot cross, as in Figure 2(a) and (b). 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 segments may cross based on the extreme value comparison. For a given pair of segments, k' and l', $m_{k'l'}$ is set to one if the ranges of x and y values of k' and l' intersect, or zero otherwise.

The range of intersection in the x direction is calculated for those pairs of segments, k' and l', with $m_{k'l'} = 1$. It is then necessary to obtain the y values of each pair of segments for the same x values, using linear interpolation at a number of grid points chosen within the range of intersection of the two segments. The range of intersection of two segments, $x_{\text{left}} \leq x \leq x_{\text{right}}$, and a choice of x coordinates at which to obtain the y coordinates are illustrated in Figure 2(d).

When the y coordinates of a pair of segments, k' and l', are known for the same set of x values, it is then possible to perform the simple analysis described at the beginning of this section, that is, observing the sign of $f(x) - g(x)$, to determine if the two segments cross. For example, let $Y_{k'}$ and $Y_{l'}$ be the y values of segments k' and l' that have been obtained for the same set of x coordinates using linear interpolation. If the sign of $Y_{k'} - Y_{l'}$ changes from negative to positive, or vice versa, then the two segments cross. If the segments cross it is then possible to determine which nodes lie to either side of a crossing by examining the indices where there are sign changes in $Y_{k'} - Y_{l'}$, and therefore explicitly determine the point(s) of intersection of the two segments and hence the two contours to which they belong, if desired. Technical details of the procedures for splitting contours into single valued segments, determining if the ranges of x and y values of two segments intersect, determining the range of intersection of two segments in the x direction

and choosing the set of x coordinates at which the y values of two segments should be calculated are given in Appendix A.

3 The N-Contour Problem

This section describes how the method for determining if, and where, two contours cross may be extended to search for all crossings amongst an arbitrary number of contours. It would be inefficient to immediately compare all pairs of contours in a search for crossings using the technique of Section 2. Two methods for determining the relative closeness of contours, and thereby reducing the number of pairs of contours that should be checked using the two contour method, are described in this section.

The first method is derived directly from part of the fusion surgery subroutine of the CASL algorithm [7]. The extreme x and y coordinates for each contour j, $j = 1, \ldots, N$, are determined. From these, the contour half-widths, $w = (x_{\max} - x_{\min})/2$, half-heights, $h = (y_{\max} - y_{\min})/2$ and centres $(X, Y) = \big((x_{\min} + x_{\max})/2, (y_{\min} + y_{\max})/2\big)$ are calculated. Two contours, j' and j'', are then to be tested for crossings using the method of Section 2 if both

$$|X' - X''| \leq w' + w'' \quad \text{and} \quad |Y' - Y''| \leq h' + h'' \tag{1}$$

are satisfied, where dashes are used to denote the quantities associated with contour j', and double dashes are used to denote those associated with contour j''.

The second method for roughly determining if two contours are close enough that they may potentially cross involves constructing a digital picture of each of the contours on an $n_g \times n_g$ grid. The choice of n_g is arbitrary; however, from experience, using a value for n_g between about 200 and 300 works well, and allows a fairly accurate depiction of each contour without consuming excessive computational resources. The information regarding

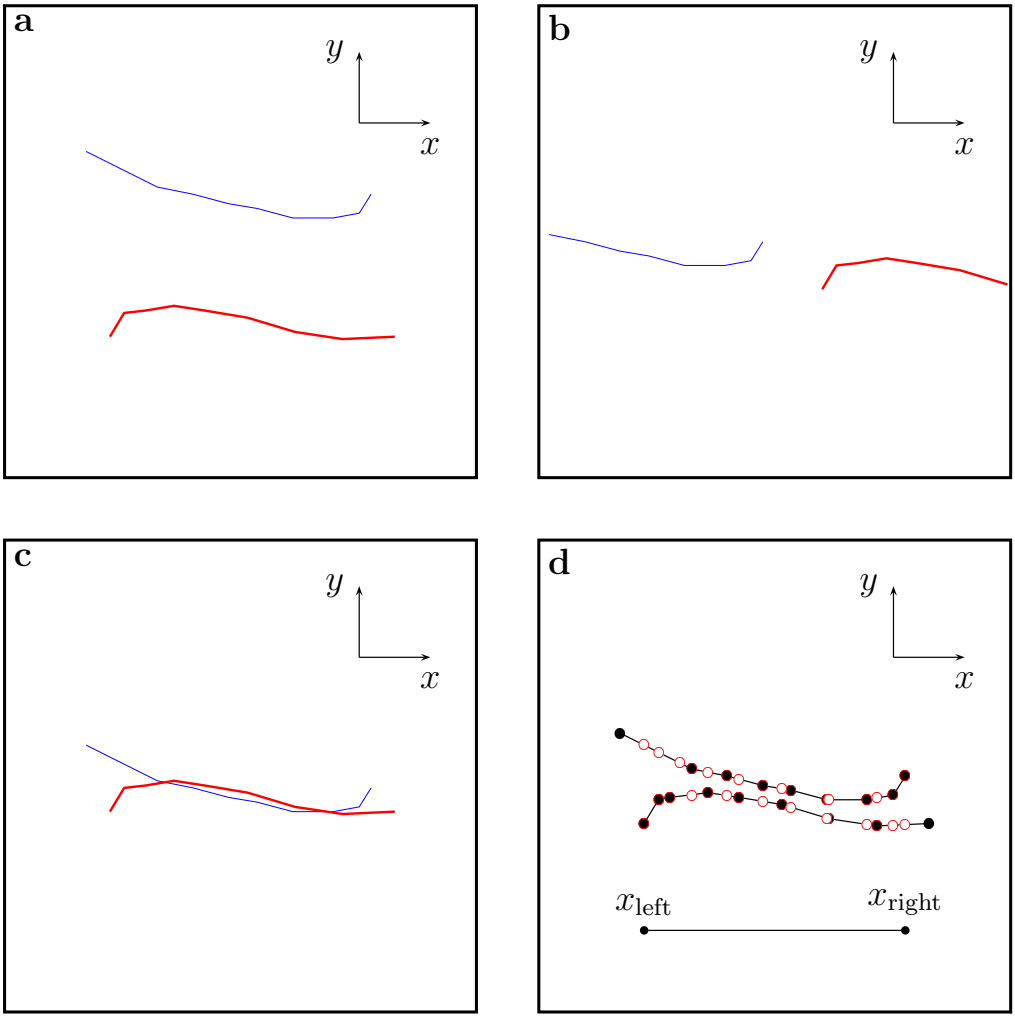


FIGURE 2: Two segments cannot cross if they are suitably separated in the (a) y or (b) x directions, or both. For two segments to cross the ranges of their x and y coordinates must overlap as in (c). In (d) closed black circles denote the nodes that make up each of the segments. Open red circles are used to denote the points at which the y coordinates of each segment are determined in order to find if the two segments cross.

the digital representation of each contour is 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 drawing of contour j. Initially, T is set to be the $n_g \times n_g \times N$ zero array. The production of the digital versions of each of the contours is done as follows. 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 example if the coordinates of all the nodes from all the contours are confined to the domain $-\pi \le x \le \pi$, $-\pi \le y \le \pi$ then an appropriate scaling and shifting would be

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

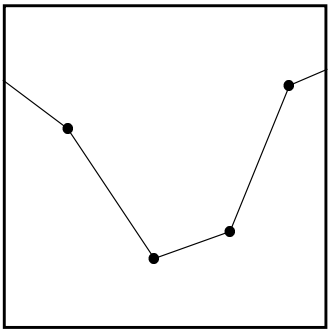
where $(\hat{x}_i, \hat{y}_i)$ are the coordinates of node i when shifted and scaled to the domain $1 \le x \le n_g$, $1 \le y \le n_g$. The nearest integers to both $\hat{x}_i$ and $\hat{y}_i$ are calculated and stored as $\bar{x}_i$ and $\bar{y}_i$ respectively. For contour j, $t_{\bar{y}_i \bar{x}_i j}$ is set to 1.

To construct an accurate representation of each contour, consecutive nodes in their scaled, integer form need to be joined by the equivalent of straight line segments. That is for contour j, the entries of T lying on the equivalent straight line segment between $t_{\bar{y}_i \bar{x}_i j}$ and $t_{\bar{y}_{(i+1)} \bar{x}_{(i+1)} j}$ are set to 1. The Bresenham line algorithm [2, 8] is used to determine efficiently which are the appropriate array entries to fill, if any. Bresenham's algorithm was originally developed for computer control of digital plotters and is now commonly used to draw lines on computer displays. Figure 3 illustrates the process of constructing the digital representation of a curve.

Pairs of layers of T are added together in turn, and the resulting matrices are searched for entries with value greater than 1. This process requires $(N^2 - N)/2$ matrix additions and search operations. Two contours, j' and j'', are to be further scrutinised for potential crossings if the matrix obtained by adding layers j' and j'' of T contains any entries with value 2.

In practice, the method of producing digital representations of each contour picks out many fewer pairs of contours for further testing using the

a



b

0	0	0	0	0	0
0	0	0	0	0	1
0	1	0	0	0	0
0	0	0	0	0	0
0	0	1	0	1	0
0	0	0	0	0	0

c

0	0	0	0	0	0
1	0	0	0	0	1
0	1	0	0	0	1
0	1	0	0	1	0
0	0	1	1	1	0
0	0	0	0	0	0

FIGURE 3: Part of a contour is depicted in (a). Black 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 in both the x and y directions. The nearest integers to these values are taken, and the corresponding entries in T are set to 1, as in (b). Appropriate intermediate entries are set to 1 with the aid of the Bresenham line algorithm in (c). To aid with visual representation, the rows of the matrices in (b) and (c) are drawn in ascending order from the bottom of the panels to the top.

method of Section 2 than the method derived from fusion surgery. As an example, the two methods have been applied to a data set containing 201 time steps from the simulation of two dimensional turbulence in a cylindrical domain using the CASL algorithm (the same data set from which the images in Figure 1 were obtained). Figure 4 shows the number of pairs of contours selected for further scrutiny by each method. The reason that fewer pairs are picked out by the method of producing digital representations is related to the fact that this method includes information about the general shape of the contours, whereas the method derived from fusion surgery does not. Due to its finite resolution the digital representation method may miss some pairs of contours that could cross. This rarely happens and in cases that it does any crossings missed from one time step are usually detected in the next. By contrast, application of equations (1) guarantees identification of all potentially crossing contour pairs.

Although the method of producing digital representations of each contour is better than application of equations (1) at reducing the number of pairs of contours that undergo the final tests for crossings, as it has been described above it is not efficient in terms of storage. It is necessary to reduce the space taken up by the digital representations of the contours to make many calculations tractable. The storage efficiency of the digital representation method may be vastly improved in the following way. An $n_g \times n_g$ matrix, $\widetilde{T}$, with elements $\tilde{t}_{rc}$, is used to temporarily store the digital representation of each contour. Prior to the construction of the representation of each contour, $\widetilde{T}$ is set to be the $n_g \times n_g$ zero matrix. The columns and rows of all the entries of $\widetilde{T}$ that are set to 1 during the process of constructing a digital representation of a contour are recorded in the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively. Another vector, $\boldsymbol{b}$, is used to record the indices of the entries in $\boldsymbol{x}$ and $\boldsymbol{y}$ that correspond to the first point on each contour. Once the digital representation of a contour is complete, $\widetilde{T}$ 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. $\widetilde{T}$ is then set to O before the construction of the next contour representation. After all the contours have

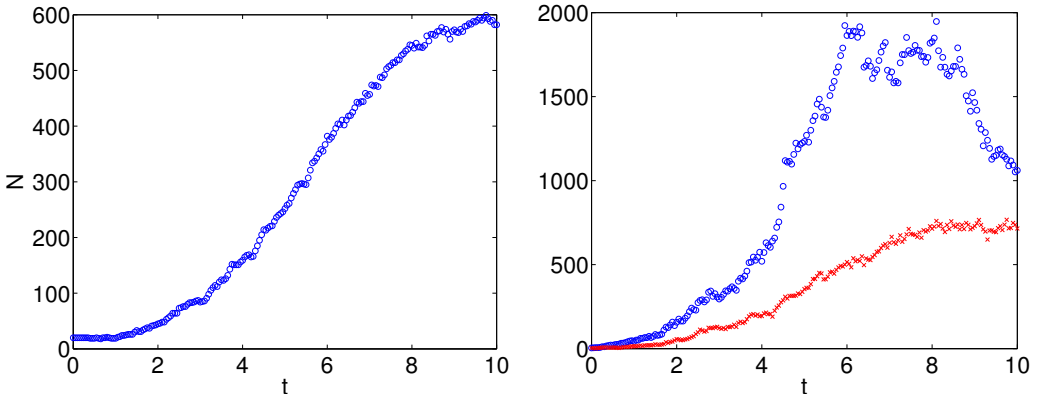


FIGURE 4: The leftmost plot shows the number of contours, N, for a series of time steps from a CASL simulation. The number of contours varies due to the application of contour surgery. The right hand diagram illustrates the number of pairs of contours that are to be further tested for crossings after application of equations (1) (blue circles), or the method of producing digital pictures (red crosses). The number of pairs in the right hand panel have been selected from as many as 179101 contour pairs corresponding to the peak value of 599 contours.

been treated in this way, S is searched for entries with value greater than 1. If any such entries are found, $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{b}$ 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 using the method of Section 2. A brief summary of the steps used to find any contour crossings that occur between N contours is given in Table 1. Appendix B contains estimates for the computational cost of detecting contour crossings with the aid of producing digital representations of the contours and improved storage efficiency.

When implemented in MATLAB the version of the algorithm that uses equations (1) to select pairs of contours took 119 000 seconds of CPU time on a 2200 MHz Pentium Xeon processor to find all contour crossings in the two dimensional turbulence data set. The version of the algorithm that utilises digital representations of the contours took 62 000 seconds of CPU time to analyse the same data set. The results obtained by both methods were in good agreement. The bulk of the calculation time for both cases was spent in the part of the algorithm that compares extreme values of single valued segments and then explicitly searches for crossings where appropriate. Over 94% of the CPU time for the first calculation, and over 88% of the second was spent in this part of the calculation. In contrast, a maximum of 3.34 seconds of CPU time per time step of data analysed was consumed picking contour pairs by application of equations (1) and a maximum of 2.37 seconds was used by the digital representation of contours method.

4 Concluding Remarks

The efficiency of the algorithm described in Section 3 may be improved by reordering some of the steps outlined in Table 1. After reading the data the contours could be split into single valued segments, and digital representations of the segments rather than the entire contours should be produced.

TABLE 1: Summary of Procedures for N Contour Algorithm

1. Input number of grid points, n_g, and number of time steps to be analysed.
2. For each time step
   (a) read data
   (b) construct digital representations of contours
   (c) determine pairs of contours to be tested using two contour procedure
   (d) If there are any pairs of contours requiring further testing
      i. Split contours into single valued segments (see appendix A)
      ii. For pairs of contours identified as being close to each other, compare extreme x and y values of single valued segments
      iii. Calculate range of intersection in x direction of segments that may cross. Use linear interpolation to obtain the y coordinates of each segment for the same set of x coordinates
      iv. Perform subtraction and look for sign changes to identify contour crossings

This would eliminate the process of comparing extreme values of segments (for a given pair of contours the cost is $2s_{j_i'}s_{j_i''}$, see Appendix B) and immediately identify which segments need to be explicitly checked for crossings.

The methods described in Sections 2 and 3 may be easily extended to include a search for self-intersection of a contour. They are also capable of dealing with contours which are closed in the periodic sense, such as those used to represent the regular part of the potential vorticity field in β-plane flows.

Acknowledgements: We thank Les Farnell for his helpful suggestions regarding the construction of digital pictures of the contours with the aid of the Bresenham line algorithm [2].

References

[1] H. L. Berk and K. V. Roberts. The water-bag model. *Methods Comput. Phys.*, 9:87–134, 1967. C694

[2] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965. C702, C708

[3] E. Chacón Vera and T. Chacón Rebollo. On cubic spline approximations for the vortex patch problem. *App. Num. Math.*, 36:359–387, 2001. C695, C696

[4] D. G. Dritschel. Contour surgery: A topological reconnection scheme for extended integrations using contour dynamics. *J. Comp. Phys.*, 77:240–266, 1988. C695, C696

[5] 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. C695, C696

[6] 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. C695

[7] 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. C695, C696, C700

[8] K. Hoff. Derivation of Bresenham's line algorithm. 1995. [Online] http://www.cs.unc.edu/~hoff/projects/comp235/bresline/bresen.html C702

[9] C. Macaskill, W. E. P. Padden and D. G. Dritschel. The CASL algorithm for quasi-geostrophic flow in a cylinder. *J. Comp. Phys.* 188:232–251, 2003. C696

[10] P. W. C. Vosbeek and R. M. M. Mattheij. Contour dynamics with symplectic time integration. *J. Comp. Phys.* 133:222–234, 1997. C695, C696

[11] 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. C696

[12] N. J. Zabusky, M. H. Hughes and K. V. Roberts. Contour dynamics for the Euler equations in two dimensions. *J. Comp. Phys.* 30:96–106, 1979. C694, C695

A Details

A.1 Splitting of contours into single valued segments

Let x_i denote the x coordinate of node i. The splitting of each contour, j, into s_j segments is achieved by looking at the sign of $x_{i+1} - x_i$ for $i = 1, \ldots, n_j$ on each contour, where n_j is the number of nodes used to discretely represent contour j, and $x_{n_j+1} = x_1$, since all contours are closed. A particular node, i', is identified as the last node of one segment and the first node of the next if the sign of $x_{i'+1} - x_{i'}$ is different to that of $x_{i'} - x_{i'-1}$.

A.2 Determining if the ranges of x and y values of two segments intersect

The extreme y values of the single valued segments are considered first. For a given pair of segments from different contours, k' and l' with $k' \in \{1, 2, \ldots, s_1\}$ and $l' \in \{1, 2, \ldots, s_2\}$, if the minimum y value of either segment exceeds the maximum y value of the other segment, then the two segments cannot possibly cross, $m_{k'l'}$ is set to zero, and the extreme y values of the next pair of segments should be examined. Otherwise the ranges of y values of the two segments intersect and their extreme x values should be examined. Similar to the test of extreme y values, if the minimum x value of either segment exceeds the maximum x value of the other segment, then the two segments cannot cross, $m_{k'l'}$ is set to zero, and the extreme y values of the next pair of segments should be considered. Passing both the tests of extreme x and y values, the ranges of x and y values of segments k' and l' intersect, it is possible that the two segments may cross, and $m_{k'l'}$ is set to one.

A.3 Range of intersection of two segments in the x direction

The range of intersection in the x direction is calculated for those pairs of segments, k' and l', with $m_{k'l'} = 1$. Let $x_{k'\,\min}$ and $x_{k'\,\max}$ be the extreme x values of segment k', and let $x_{l'\,\min}$ and $x_{l'\,\max}$ be the corresponding values of segment l'. Then the range of intersection of the two segments in the x direction is $x_{\text{left}} \le x \le x_{\text{right}}$, where $x_{\text{left}} = \max(x_{k'\,\min}, x_{l'\,\min})$ and $x_{\text{right}} = \min(x_{k'\,\max}, x_{l'\,\max})$.

A.4 Obtaining the y coordinates of the segments at the same grid points

Conceptually the easiest way to do this is to calculate the y values of the two segments for a set of equally spaced x coordinates in the range $x_{\text{left}} \le x \le x_{\text{right}}$ using linear interpolation. It is more efficient to obtain the y values of two segments at the ordered set of x coordinates $X_{k'} \cup X_{l'}$ using linear interpolation, where $X_{k'}$ is the set of x coordinates of nodes belonging to segment k' that lie in the range $x_{\text{left}} \le x \le x_{\text{right}}$ and $X_{l'}$ is the corresponding set of nodes belonging to segment l'.

B Operations count for components of the N contour algorithm utilising digital representation of contours and improved storage efficiency

Production of the matrix S requires approximately $22n+5\eta+N$ calculations, where n is the total number of of nodes used to discretely represent the

contours, η is the number of intermediate points filled with the aid of the Bresenham line algorithm and N is the total number of contours.

The cost of determining which pairs of contours need to be further tested for crossings after the production of the matrix S is roughly $n_g^2 + A(n + \eta + 2N) + \sum_{i=1}^{A} \left(B_i(N + 3/2) + B_i^2/2\right)$, where n_g is the number of grid points used in both the x and y directions to represent the contours digitally, A is the number of elements of S with value greater than 1 and B_i is the number of contours that pass through the ith entry of the matrix S that has value greater than 1 ($i = 1, \ldots, A$).

Splitting all the contours into single valued segments uses $4n + 3s + 3N$ operations, where s is the total number of single valued segments produced. The jth contour is split into s_j segments, so $s = \sum_{j=1}^{N} s_j$.

The cost of comparing the extreme values of the single valued segments of the ith pair of contours, j_i' and j_i'', is $2s_{j_i'}s_{j_i''}$ (one set of comparisons for both the x and y directions). If there are D pairs of contours picked out for further scrutiny by examination of their digital representations, then at most $2\sum_{i=1}^{D} s_{j_i'}s_{j_i''}$ calculations are required, since those pairs of segments that fail the extreme y value test do not require examination of their x coordinates.

The cost of explicitly checking if two segments cross by determining the y coordinates of the two segments at the same set of x coordinates and then examining the sign of the difference of the two curves is at worst $n_{\bar{s}}^2 + n_{\hat{s}}^2 + n_{\text{interp}}^2 + 2n_{\bar{s}} + 2n_{\hat{s}} + 11n_{\text{interp}}$ in addition to a few calculations to identify the nodes on either side of any crossings detected. $n_{\bar{s}}$ is the number of nodes on one segment and $n_{\hat{s}}$ is the number of nodes on the other. n_{interp} is used to denote the number of interpolation points used. Several parts of the calculation require x coordinates to be sorted in ascending order, the cost of such an operation being dependent on the square of the number of nodes in the worst case.