ANOTHER LOOK AT VELOCITY EXTENSIONS IN THE LEVEL SET METHOD *

DAVID L. CHOPP†

Abstract. The level set method [8] has become a widely used numerical method for moving interfaces, e.g. see the many examples in [11, 7]. For many applications, the velocity of the interface is known only on the interface, while the level set method requires information about the interface speed at least in a neighborhood of grid points near the interface. To address this issue, velocity extensions are used to map the velocity information on the interface into the rest of the computational domain [2]. This allows the level set method to proceed.

The velocity extension method presented in [2] uses the fast marching method [12, 13], and is only a first order approximation for the velocity field near the interface. Furthermore, it can lead to unexpected behavior in some cases. This is primarily due to the strictly local solution for the characteristics of the flow near the interface. In this paper, we look more closely at the characteristics near the interface, and then present a modified velocity extension method, also based on the fast marching method, which handles the characteristcs more accurately. In turn, this new method will lead to some interesting new possibilities for the fast marching method.

Key words. fast marching method, level set method, velocity extension, bicubic interpolation

AMS subject classifications. 65M06, 65D05, 65D10

1. Introduction.

The level set method is a widely used numerical method for moving interfaces [11, 7]. For many moving interface problems, the speed of the interface is known only on the interface itself. However, the level set method requires the interface velocity on grid points, which generally will not lie directly on the interface. One commonly used strategy for mapping the interface velocity onto the neighboring grid points is through a velocity extension [2].

Briefly, the method for velocity extensions presented in [2] assumes a first order approximation for the velocity field near the interface by extending the characteristics for the resulting flow in straight lines normal to the interface. This is equivalent to a first-order Taylor expansion of the resulting flow. For applications where the interface moves only short distances between velocity extensions, the errors in using the linear velocity extensions may not be important. However, for some applications, the cost of determining the velocity is computationally expensive. In those instances, it is advantageous to compute the velocity field less often, extend the velocity field, then take many interface evolution time steps with the calculated velocity. This is the strategy that was employed in [5]. In this paper, we show that the standard velocity extension techniques suffer greater errors when this strategy is employed. To address this issue, we present here a modified fast marching method for computing velocity extensions which respects the characteristics for the resulting flow, and consequently, will allow for more accurate level set method computations.

Since the Fast Marching Method was first introduced [12], a number of improvements, extensions, and variations have been published. Higher order Fast Marching Methods were introduced in [13], which assumed sufficient accuracy in the initialization. An improved initialization procedure was presented in [4], where a bicubic interpolant was used to improve the accuracy of the initial data for the Fast Marching

*This work was supported in part by the National Institute of General Medical Sciences of the National Institutes of Health under grant #1R01-GM67248-01.

† Engineering Sciences and Applied Mathematics Dept., Northwestern University, Evanston, Illinois, 60208-3125 (chopp@northwestern.edu).

Method. A more general class of methods called Ordered Upwind Methods, which includes the Fast Marching Method, were designed in [14]. For Ordered Upwind Methods, the speed function is still required to be monotonic, but now can also depend on the local gradient of ϕ. It is not clear whether the present work can likewise be extended to the more general Ordered Upwind Methods. An iterative method, called the Fast Sweeping Method, for solving problems similar to those presented in [14] has been proposed in [6]. None of these existing modifications address the curving characteristics related to moving interfaces we present in this paper.

In the present work, the fast marching method will first be modified so that velocity extensions are computed simultaneously with the evolving interface. This will highlight some important differences between the velocity extensions used in [2] and the present work. The two methods will be compared with a marker particle approach to verify that the new velocity extension method is the proper way to do velocity extensions. Next, the initialization procedure presented in [4] will be extended to apply to the present work. The resulting modified method will then be used to propagate pieces of interface where the speed function is zero at the endpoints.

The remainder of this paper is organized as follows. In section 2, we review the fast marching method. In section 3, we describe velocity extensions and the new algorithm which couples velocity extensions to the fast marching method. In section 4, we modify this algorithm to consider the evolution of initial segments. In section 5, we show how this segment algorithm can be reassembled to do the fast marching method for non-monotonic speed functions. We conclude with some final remarks in section 6.

2. The Fast Marching Method. The fast marching method is an optimal method for solving an equation of the form

$$\|\nabla\phi\| = 1/F(\mathbf{x}) \tag{2.1}$$

where $F(\mathbf{x})$ is a monotonic speed function for an advancing interface. If $\Gamma = \phi^{-1}(0)$ represents the initial interface, and ϕ solves (2.1), then $\phi^{-1}(t)$ gives the location of the interface evolving with normal velocity $F(\mathbf{x}) > 0$ at time t. The advantage of this method over the original level set method is that the entire evolution of the front is computed in one pass over the mesh with an operation count of $O(N \log N)$ for N mesh points. It is also advantageous over other front tracking algorithms in that it uses techniques borrowed from hyperbolic conservation laws to properly advance fronts with sharp corners and cusps. We present here a basic description of the method, the interested reader is referred to [11, 12, 13].

In the fast marching method, (2.1) is solved numerically by using upwind finite differences to approximate $\nabla\phi$. The use of upwind finite differences indicates a causality, or a direction for the flow of information propagating from the initial contour $\phi^{-1}(0)$ outward to larger values of ϕ. This causality means that the value of $\phi(\mathbf{x})$ depends only on values of $\phi(\mathbf{y})$ for which $\phi(\mathbf{y}) \leq \phi(\mathbf{x})$. Thus, if we solve for the values of ϕ in a monotonically increasing fashion, then the upwind differences are always valid and all the mesh points are eventually computed. This sequential procession through the mesh points is maintained by a heap sort which controls the order in which the mesh points are computed.

To begin, the mesh points are separated into three disjoint sets, the set of *accepted* points A, the set of *tentative* points T, and the set of *distant* points D. The mesh points in the set A are considered computed and are always closer to the initial interface than any of the remaining mesh points. The mesh points in T are all potential candidates to be the next mesh point to be added to the set A. The mesh points in T are always

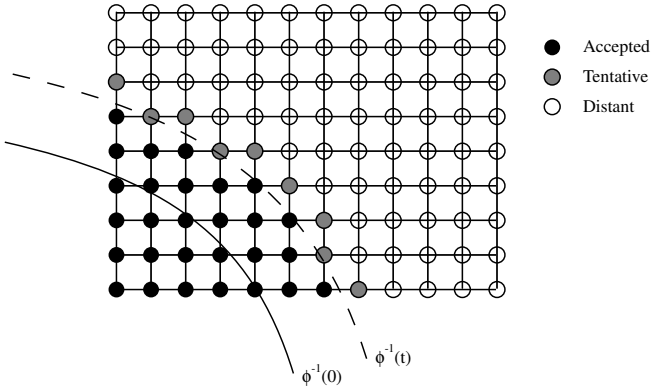

FIG. 2.1. *Illustration of the sets A, T, and D*

kept sorted in a heap sort so that the best candidate is always easily found. The mesh points in D are considered too far from the initial interface to be possible candidates for inclusion in A. Thus, if $\mathbf{x} \in A$, $\mathbf{y} \in T$, and $\mathbf{z} \in D$, then $\phi(\mathbf{x}) < \phi(\mathbf{y}) < \phi(\mathbf{z})$. Figure 2.1 shows the relationship between the different sets of mesh points.

To describe the main algorithm for the fast marching method, we will use the notation for discrete derivatives given by

$$\begin{aligned}
\phi_{i,j} &= \phi(\mathbf{x}_{i,j}) \\
D_x^+\phi_{i,j} &= \frac{1}{\Delta x}(\phi_{i+1,j} - \phi_{i,j}) \\
D_x^-\phi_{i,j} &= \frac{1}{\Delta x}(\phi_{i,j} - \phi_{i-1,j}) \\
D_y^+\phi_{i,j} &= \frac{1}{\Delta y}(\phi_{i,j+1} - \phi_{i,j}) \\
D_y^-\phi_{i,j} &= \frac{1}{\Delta y}(\phi_{i,j} - \phi_{i,j-1})
\end{aligned}$$

where Δx, Δy are the space step sizes in the x and y directions respectively. One of the key components in the fast marching method is the computation of the estimate of ϕ for points in T. Suppose, for example, mesh points $\mathbf{x}_{i-1,j}$, $\mathbf{x}_{i,j+1} \in A$, and $\mathbf{x}_{i,j} \in T$. Given the values of $\phi_{i-1,j}$, $\phi_{i,j+1}$, we must estimate the value of $\phi_{i,j}$. This is accomplished by looking at the discretization of (2.1) given by

$$(D_x^-\phi_{i,j})^2 + (D_y^+\phi_{i,j})^2 = \frac{1}{F_{i,j}^2}. \tag{2.2}$$

(2.2) reduces to a quadratic equation in $\phi_{i,j}$ given by

$$\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right)\phi_{i,j}^2 - 2\left(\frac{\phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1}}{\Delta y^2}\right)\phi_{i,j} + \frac{\phi_{i-1,j}^2}{\Delta x^2} + \frac{\phi_{i,j+1}^2}{\Delta y^2} - \frac{1}{F^2} = 0. \tag{2.3}$$

The new estimate for $\phi_{i,j}$ is given by the largest of the two roots of (2.3).

The remaining configurations and the resulting quadratic equations can be derived

in a similar fashion and result in the following formulation:

$$\begin{aligned}(\max(D_x^- \phi_{i,j} + s_{x,-1}\frac{\Delta x}{2} D_x^- D_x^- \phi_{i,j}, -D_x^+ \phi_{i,j} + s_{x,1}\frac{\Delta x}{2} D_x^+ D_x^+ \phi_{i,j}, 0)^2 \\ + \max(D_y^- \phi_{i,j} + s_{y,-1}\frac{\Delta y}{2} D_y^- D_y^- \phi_{i,j}, -D_y^+ \phi_{i,j} + s_{y,1}\frac{\Delta y}{2} D_y^+ D_y^+ \phi_{i,j}, 0)^2 = \frac{1}{F_{i,j}^2}.\end{aligned} \tag{2.4}$$

Here, the coefficient $s_{x,-1}$ is defined so that

$$s_{x,-1} = \begin{cases} 1 & \mathbf{x}_{i-1,j} \in A \cup T \\ 0 & \mathbf{x}_{i-1,j} \in D \end{cases}.$$

The other coefficients are similarly defined.

Now the fast marching method can be assembled as an algorithm:

1. Initialize all the points adjacent to the initial interface with an initial value, put those points in A. A discussion about initialization follows in §3.3. All points $\mathbf{x}_{i,j} \notin A$, but are adjacent to a point in A are given initial estimates for $\phi_{i,j}$ by solving (2.4). These points are tentative points and put in the set T. All remaining points are placed in D and given initial value of $\phi_{i,j} = +\infty$.
2. Choose the point $\mathbf{x}_{i,j} \in T$ which has the smallest value of $\phi_{i,j}$ and move it into A. Any point which is adjacent to $\mathbf{x}_{i,j}$ (i.e. the points $\mathbf{x}_{i-1,j}$, $\mathbf{x}_{i,j-1}$, $\mathbf{x}_{i+1,j}$, $\mathbf{x}_{i,j+1}$) which is in T has its value $\phi_{i,j}$ recalculated using (2.4). Any point adjacent to $\mathbf{x}_{i,j}$ and in D has its value $\phi_{i,j}$ computed using (2.4) and is moved into the set T.
3. If $T \neq \emptyset$, go to step 2.

Note that (2.2) is a first order approximation of (2.1). A description of higher order methods can be found in [4].

3. Coupling Velocity Extensions to the Fast Marching Method. For any moving interface problem, the key quantity to be determined is the speed of the interface. For some applications, the speed can be passively obtained from a larger flow field in which the interface is embedded, e.g. two-phase fluid flow. For other applications, the speed is computed locally on the interface and it is not easily expressed analytically away from the interface, e.g. crack propagation.

For level set methods, when the speed is obtained passively, it seems natural to use the flow field velocity, $\mathbf{v}$, in the level set evolution equation

$$\phi_t + \mathbf{v} \cdot \nabla \phi = 0. \tag{3.1}$$

However, it was discovered early in the development of the level set method, that this does not produce very stable results, leading to the use of reinitialization [3]. The reinitialization process itself leads to some additional diffusive error, though the process has been refined more recently [4, 9, 10, 15]. It was observed in [2], that the use of velocity extensions would eliminate the need for reinitialization, and hence improve the accuracy of the resulting computation.

3.1. Velocity Extensions for the Level Set Method. One of the fundamental differences between the Level Set Method and Lagrangian type methods is that the evolution of the interface is embedded in the evolution of a higher dimensional function ϕ, through the level set evolution equation [8]

$$\phi_t + F\|\nabla \phi\| = 0. \tag{3.2}$$

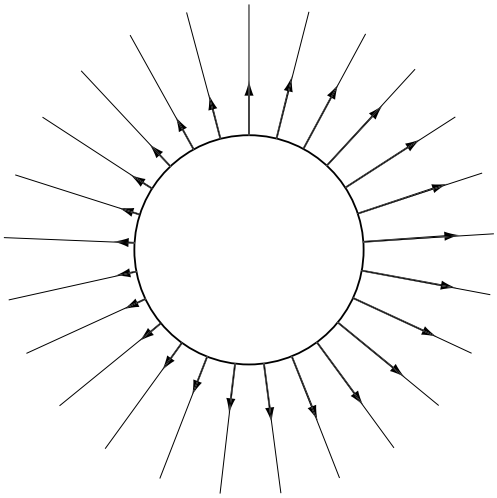

FIG. 3.1. *Lines of constant speed F for a circular interface. Arrows indicate magnitude of F at different points on the interface.*

In order to use this equation, the speed function F must be defined not only on the moving interface, but in the entire domain of ϕ.

Furthermore, it is known from [3], that for stability, it is ideal to preserve ϕ as a signed distance function, characterized by the property $\|\nabla\phi\| = 1$. However, this will happen only under very special circumstances: (1) ϕ is allowed to deviate from the signed distance function, but then corrected via reinitialization, or (2), the velocity field is constructed specially so that $\|\nabla\phi\| = 1$ is preserved. These two different solutions characterize the key difference between different implementations of the Level Set Method. A discussion about the relative merits of these different approaches is best left to a survey paper, and beyond the scope of this paper. Instead, we focus on the solution (2) above as described in [2].

The velocity field F which preserves the signed distance function is characterized by

$$\nabla F \cdot \nabla\phi = 0. \tag{3.3}$$

This equation is easily obtained by assuming $\|\nabla\phi\| \equiv 1$, differentiating with respect to t, and then substituting for ϕ_t using (3.2) [2, 16]. Equation (3.3) is easily interpreted geometrically by noting that F is constant along lines orthogonal to the interface described by $\phi^{-1}(0)$ (see Figure 3.1).

Numerically, the velocity extension can be computed using the fast marching method. First, using some initialization process, e.g. [4], the values of ϕ and F are placed on grid points which are in a neighborhood of the initial interface. The fast marching method is now used to compute the remaining values of ϕ using speed one, i.e. $F \equiv 1$ in (2.1). This alone leads to $\|\nabla\phi\| = 1$. The extended velocity is now computed by discretizing (3.3) using the same upwind finite differences as for computing ϕ. Of course, if ϕ is already a signed distance function, then the fast marching solution for ϕ can be skipped provided care is taken to ensure the values of F are computed in the proper order, from $|\phi|$ small to $|\phi|$ large, and the upwind finite differences are taken in the direction of decreasing values of $|\phi|$. The result of this is

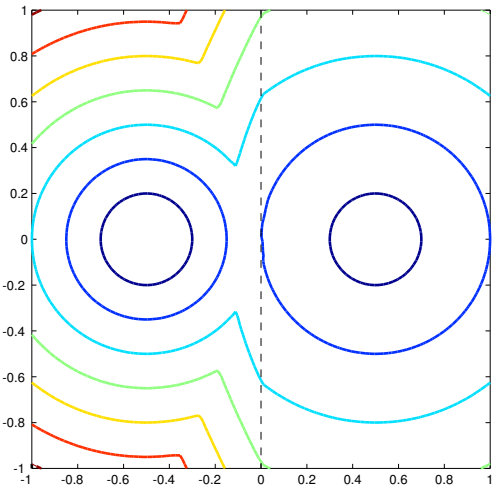

FIG. 3.2. *Example of two circles, $F = 1$ on left, $F = 2$ on right, using old velocity extension method*

illustrated by Figure 3.1, where the speed function is constant along normals to the interface regardless of the change in speed along the interface.

This approach works well for the Level Set Method because a small time step is taken with this extended velocity field, the interface evolves, and the velocity is recomputed and again extended using the new velocity. However, as will be shown in the next section, this approach does not work so well for the Fast Marching Method, nor for the case of a fixed velocity field for multiple Level Set Method steps.

3.2. Velocity Extensions and the Fast Marching Method. The velocity extension approach used above is not so well suited to the fast marching method. The primary difficulty arises from the fact that the velocity extension is not recomputed as the interface evolves, resulting in the speed function not coinciding with the characteristic paths of the points on the interface. This can lead to some erroneous results.

For example, consider the problem illustrated in Figure 3.2. Here, the initial surface consists of two circles. The circle on the left has interface speed one, and the circle on the right has speed two. According to the velocity extension method from [2], the speed to be used at a given point $\mathbf{x}$ is determined by the speed at the point $\mathbf{y}$ on the initial interface nearest to $\mathbf{x}$. This effectively divides the plane in half with the left half using speed one, and the right half speed two. Now the fast marching method is invoked to compute the time of crossing map for this flow. So long as the interfaces do not cross the centerline, everything proceeds as expected, with the right circle expanding twice as fast as the left circle. However, the right circle reaches the centerline first, and then is forced to slow down artificially because it is using the extended velocity from the left circle. This leads to an incorrect solution, which persists through the remainder of the calculation.

The reason this velocity extension failed is because the velocity is computed *a priori*. Instead, the velocity should be computed dynamically as the surface evolves, just like how it is done in the level set method. To compute the velocity dynamically, we must therefore solve a pair of equations simultaneously in the fast marching

method:

$$F\|\nabla\phi\| = 1 \tag{3.4}$$
$$\nabla F \cdot \nabla\phi = 0. \tag{3.5}$$

Again, these equations are discretized using the upwind finite differences discussed in Section 2. Following the example given by (2.2), we have the pair of equations

$$F_{i,j}^2((D_x^-\phi_{i,j})^2 + (D_y^+\phi_{i,j})^2) = 1 \tag{3.6}$$
$$(D_x^- F_{i,j})(D_x^-\phi_{i,j}) + (D_y^+ F_{i,j})(D_y^+\phi_{i,j}) = 0. \tag{3.7}$$

Solving for $F_{i,j}$, $\phi_{i,j}$ in (3.6), (3.7) results in a quartic equation for $F_{i,j}$:

$$\begin{aligned}(\phi_{i-1,j} - \phi_{i,j+1})^2(\Delta x^2 + \Delta y^2)F_{i,j}^4 \\ - 2(\phi_{i-1,j} - \phi_{i,j+1})^2(\Delta x^2 F_{i,j+1} + \Delta y^2 F_{i-1,j})F_{i,j}^3 \\ ((\phi_{i-1,j} - \phi_{i,j+1})^2(\Delta x^2 F_{i,j+1}^2 + \Delta y^2 F_{i-1,j}^2) - (\Delta x^2 + \Delta y^2)^2)F_{i,j}^2 \\ + 2(\Delta x^2 + \Delta y^2)(\Delta x^2 F_{i,j+1} + \Delta y^2 F_{i-1,j})F_{i,j} \\ - (\Delta x^2 F_{i,j+1} + \Delta y^2 F_{i-1,j})^2 = 0.\end{aligned} \tag{3.8}$$

Note that since the leading coefficient is positive, and the constant term is negative, we are guaranteed at least two real roots. In practice, our observation is that typcially four real roots are obtained from (3.8). Once (3.8) is solved, $\phi_{i,j}$ is easily obtained:

$$\phi_{i,j} = \frac{\Delta x^2(F_{i,j+1} - F_{i,j})\phi_{i,j+1} + \Delta y^2(F_{i-1,j} - F_{i,j})\phi_{i-1,j}}{\Delta x^2(F_{i,j+1} - F_{i,j}) + \Delta y^2(F_{i-1,j} - F_{i,j})}. \tag{3.9}$$

Of the four roots, the one to choose is the one which produces the smallest value of $\phi_{i,j} > \max\{\phi_{i-1,j}, \phi_{i,j+1}\}$.

Equation (3.8) can be solved either using a direct quartic polynomial solver [1], or by using an iterative scheme. In our code, we obtained the most consistent results using Newton's method to solve (3.6), (3.7) with initial data

$$F_{i,j}^0 = \frac{F_{i-1,j} + F_{i,j+1}}{2}, \tag{3.10}$$
$$\phi_{i,j}^0 = \min\left\{\phi_{i-1,j} + \frac{\Delta x}{F_{i-1,j}}, \phi_{i,j+1} + \frac{\Delta y}{F_{i,j+1}}\right\}. \tag{3.11}$$

To obtain sufficient accuracy with the direct solver, an iterative refinement method must be employed to obtain consistent results anyway, so using Newton's method with the above initial data does not present a significant degredation in performance compared to a direct solver.

As in the original fast marching method, if $|\phi_{i-1,j} - \phi_{i,j+1}|$ is sufficiently large, then the discretization in (3.6), (3.7) can erroneously produce a solution $\phi_{i,j} < \min\{\phi_{i-1,j}, \phi_{i,j+1}\}$. This violates the causality assumption of the fast marching method. The remedy here is similar to that used in the fast marching method. If

$$\phi_{i-1,j} + \frac{\Delta x}{F_{i-1,j}} < \phi_{i,j+1}, \tag{3.12}$$

then the value of $\phi_{i,j+1}$ is sufficiently lagging that it is assumed $\phi_{i,j+1}$ will have no influence on the value of $\phi_{i,j}$. Thus, the $D_y^+\phi_{i,j}$ terms in (3.6), (3.7) are discarded. Similarly, the $D_x^-\phi_{i,j}$ terms are discarded if

$$\phi_{i,j+1} + \frac{\Delta y}{F_{i,j+1}} < \phi_{i-1,j}. \tag{3.13}$$

Aside from forcing (3.6), (3.7) to be solved simultaneously, the rest of the fast marching method remains unchanged from the one described in Section 2.

3.3. Initializing the Fast Marching Method. In [4], a more accurate method for initializing the fast marching method was introduced. The method used a Newton-type method to locate the nearest point $\mathbf{y}$ on the interface $\phi(\mathbf{y}) = 0$ from a given grid point $\mathbf{x}$. This results in solving the pair of equations

$$\phi(\mathbf{y}) = 0 \tag{3.14}$$
$$\nabla\phi(\mathbf{y}) \times (\mathbf{x} - \mathbf{y}) = 0 \tag{3.15}$$

This solution is valid under the assumption that F is fixed in the radial direction. This was true using the old velocity extension methods. However, for the modified fast marching method, (3.15) is no longer valid. We derive here a replacement for (3.15).

A general solution to (2.1) for varying F has not yet been found, however, a solution using characteristics can be found under certain simplifying assumptions. If we assume Γ is a straight line, and F is a linear function on Γ, then an explicit solution can be found.

THEOREM 3.1. *Suppose* $\Gamma = \{(x,y) : ax + by = c\}$ *and* $F_0(x,y) = dx + ey + f$ *for* $(x,y) \in \Gamma$ *with* F_0 *not identically zero on* Γ*, then the equations*

$$F\|\nabla\phi\| = 1 \tag{3.16}$$
$$\nabla F \cdot \nabla\phi = 0 \tag{3.17}$$

with $\phi(x,y) = 0$, $F(x,y) = F_0(x,y)$ *for* $(x,y) \in \Gamma$ *has a solution of the form*

$$F(x,y) = \frac{db - ea}{\sqrt{a^2+b^2}}\sqrt{X(x,y)^2 + Y(x,y)^2}, \tag{3.18}$$
$$\phi(x,y) = \frac{\sqrt{a^2+b^2}}{db-ea}\tan^{-1}\left(\frac{Y(x,y)}{X(x,y)}\right), \tag{3.19}$$

if $db - ea \neq 0$, *where,*

$$X(x,y) = \frac{b}{\sqrt{a^2+b^2}}(x - A) - \frac{a}{\sqrt{a^2+b^2}}(y - B), \tag{3.20}$$
$$Y(x,y) = \frac{a}{\sqrt{a^2+b^2}}(x - A) + \frac{b}{\sqrt{a^2+b^2}}(y - B), \tag{3.21}$$

and where $A = \frac{ec+fb}{ae-bd}$, $B = \frac{af+cd}{bd-ae}$. *The solution is valid in the set* $\mathbb{R}^2 \setminus L$, *where* L *is an arbitrary line passing through the point* (A,B).

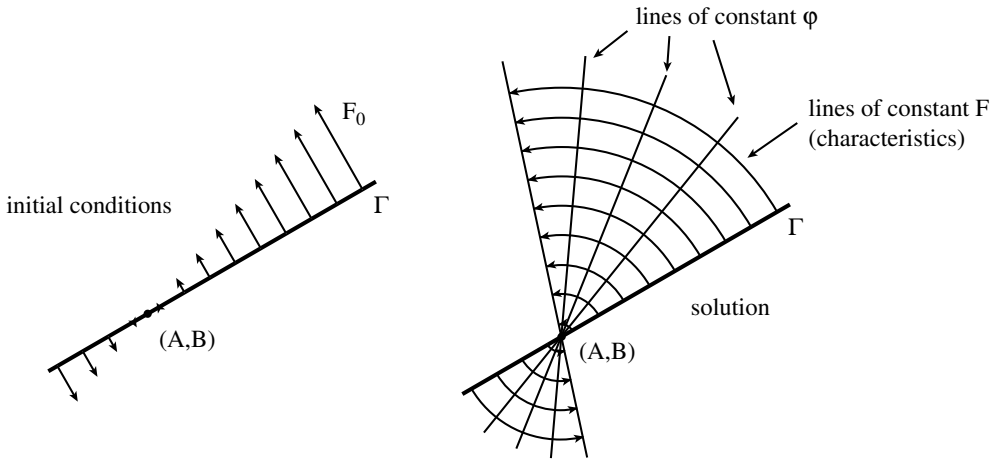

FIG. 3.3. *Illustration of a sample initial condition and corresponding solution.*

If $db - ea = 0$*, then* $F_0(x, y) = F_0$ *is constant on* Γ*, and the solution becomes*

$$F(x, y) = F_0, \tag{3.22}$$

$$\phi(x, y) = \frac{\pm 1}{F_0\sqrt{a^2 + b^2}}(ax + by - c), \tag{3.23}$$

valid on all $\mathbb{R}^2$*.*

Proof: Before doing the general case, we first assume Γ is the x-axis, and $F_0(0, 0) = 0$, hence we take $a = c = e = f = 0$, and assume b, $d \neq 0$. As a result, F_0 simplifies to $F_0(x, y) = dx$.

Recall that rigid body rotation of a straight line results in a speed function which varies linearly from the center. This suggests that a rigid body rotation about the origin, where $F_0(x, y) = 0$, may be a solution. The angle of rotation would then be $\theta(x, y) = \tan^{-1}(y/x)$, and the radius of rotation would be $r(x, y) = \sqrt{x^2 + y^2}$.

The solution can now be written down explicitly. To get the speed F, we note that the speed is constant along circles, so we define $F(x, y) = F_0(r(x, y), 0)$. The value of $\phi(x, y)$ is then simply the time it takes for the rotation of Γ to reach the point (x, y), hence we get

$$F(x, y) = d\sqrt{x^2 + y^2} \tag{3.24}$$

$$\phi(x, y) = \frac{r(x, y)\theta(x, y)}{F(x, y)} = \frac{\sqrt{x^2 + y^2}\tan^{-1}(y/x)}{d\sqrt{x^2 + y^2}} \tag{3.25}$$

A quick verification shows that the pair of functions (3.24), (3.25) solve (3.16), (3.17) with $\phi(x, y) = 0$, $F(x, y) = F_0(x, y)$ for $(x, y) \in \Gamma$. This solution is illustrated in Figure 3.3.

Under the assumption that $bd - ae \neq 0$, the general solution is obtained by using rigid body rotation and translation to transform Γ into the x-axis, and the corresponding values of F_0 so that $F_0(0, 0) = 0$. This transformation is encoded in (3.20), (3.21). Applying the solution (3.24), (3.25) and then inverting the transformation yields (3.18), (3.19). The solution (3.18), (3.19) is then easily verified.

Note that the solution is the graph of a spiral helix with axis of rotation in the z-axis direction centered at (A, B). For the solution to be correct in a classical sense, we must make a branch cut through the center of the helix, hence the solution is made valid in $\mathbb{R}^2 \setminus L$ where L is a straightline branch cut through the point (A, B).

Finally, if $db - ae = 0$, then ∇F_0 is aligned with the normal of Γ so that $F_0(x, y) = F_0$ is constant on Γ. In this case, the solution is the line Γ translating in the direction of its normal at speed F_0. The resulting time of crossing map is then easily obtained to be (3.22), (3.23). These equations are also easily verified to be a solution. □

We can now use this theorem to derive a general condition equivalent to (3.15) for the case where F varies along the initial interface. As before, given a grid point $\mathbf{x}$, we need to find $\mathbf{y}$ such that $\mathbf{y}$ is on the initial interface, i.e. $\phi(\mathbf{y}) = 0$, and such that the characteristic solution starting at $\mathbf{y}$ passes through the point $\mathbf{x}$. One way to test for this is to check whether the computed value of $F(\mathbf{x})$ from (3.18) matches the value at $F(\mathbf{y})$. To this end, we linearize the interface at a point $\mathbf{y}$ to get $\Gamma = \{\mathbf{x}' : \nabla\phi(\mathbf{y}) \cdot \mathbf{x}' = \nabla\phi(\mathbf{y}) \cdot \mathbf{y}\}$, and the corresponding initial speed function becomes $F_0 = \nabla F(\mathbf{y}) \cdot \mathbf{x}' + F(\mathbf{y}) - \nabla F(\mathbf{y}) \cdot \mathbf{y}$.

Applying the theorem with these values for Γ and F_0 gives

$$F(\mathbf{x}) = \frac{1}{\|\nabla\phi(\mathbf{y})\|}\Big(\|\mathbf{x}-\mathbf{y}\|^2(\mathbf{k}\cdot(\nabla F(\mathbf{y})\times\nabla\phi(\mathbf{y})))^2 - 2F(\mathbf{y})(\nabla\phi(\mathbf{y})\times(\mathbf{x}-\mathbf{y}))(\mathbf{k}\cdot(\nabla F(\mathbf{y})\times\nabla\phi(\mathbf{y}))) + F(\mathbf{y})^2\|\nabla\phi\|^2\Big)^{1/2}. \quad (3.26)$$

Here, $\mathbf{k}$ is the unit vector in the z-direction. We now set the criterion for the correct $\mathbf{y}$ to be that $F(\mathbf{x}) = F(\mathbf{y})$, i.e. both $\mathbf{x}$ and $\mathbf{y}$ lie on the same characteristic. For this equation to hold, it follows from (3.26), that the first two terms in the parentheses must sum to zero:

$$\|\mathbf{x}-\mathbf{y}\|^2(\mathbf{k}\cdot(\nabla F(\mathbf{y})\times\nabla\phi(\mathbf{y})))^2 - 2F(\mathbf{y})(\nabla\phi(\mathbf{y})\times(\mathbf{x}-\mathbf{y}))(\mathbf{k}\cdot(\nabla F(\mathbf{y})\times\nabla\phi(\mathbf{y}))) = 0. \quad (3.27)$$

Setting this as the criterion, and simplifying, produces the final equation which we take to be the equivalent to (3.15):

$$\nabla\phi(\mathbf{y})\times(\mathbf{x}-\mathbf{y}) = \frac{\|\mathbf{x}-\mathbf{y}\|^2(\mathbf{k}\cdot(\nabla F(\mathbf{y})\times\nabla\phi(\mathbf{y})))}{2F(\mathbf{y})}. \quad (3.28)$$

As should be expected, this expression simplifies to (3.15) when $\nabla F(\mathbf{y}) = 0$. This solution is illustrated in Figure 3.4.

As in [4], the functions ϕ, F are approximated locally by a bicubic interpolant to evaluate (3.28) at a subgrid level. A Newton method similar to that used in [4] is then employed to solve the condition $\phi(\mathbf{y}) = 0$ with (3.28) for $\mathbf{y}$ given $\mathbf{x}$. Once $\mathbf{y}$ is found which satisfies these conditions, the values of $F(\mathbf{x})$, $\phi(\mathbf{x})$ are then evaluated directly from (3.18), (3.19). Again, note that if $\nabla F(\mathbf{y}) \times \nabla\phi(\mathbf{y})$ is determined to be zero, then the solution in (3.22), (3.23) is used instead.

3.4. Comparing Velocity Extensions Methods. We next show some examples to compare the two velocity extension methods. Returning to the example of the two circles of the previous section, we recompute ϕ using the new velocity extension method to compare. The results are shown in Figure 3.5. Note how the interface propagating from the right continues at the proper speed even after the two interfaces have merged.

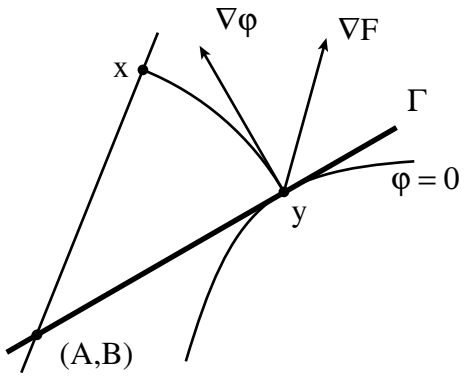

FIG. 3.4. *Illustration of local search for* **y** *given* **x** *satisfying (3.28)*

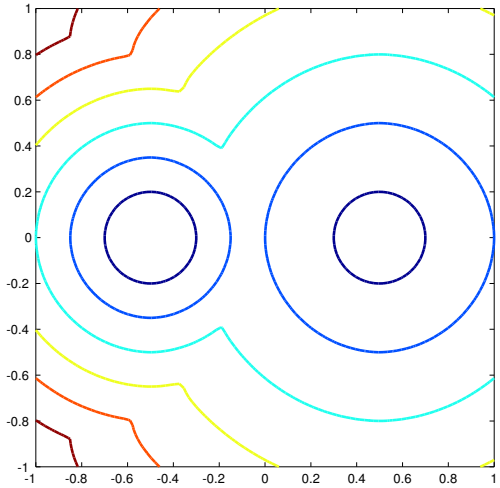

FIG. 3.5. *Example of two circles, $F = 1$ on left, $F = 2$ on right, using new velocity extension method.*

In Figure 3.6, we show what happens when a single circular interface, with a varying speed function around the interface is computed. Here, F is linear in x, with F near zero on the left side of the circle. Using the old method, everything slows down to the left of the circle, while using the new method, the interface propagates as it should.

To verify the accuracy of the new method, we compare the new method with a corresponding marker particle method. In this case, each marker particle is assigned a normal speed which does not change over time. The interface is then allowed to evolve. We show the results of this comparison in Figure 3.7. Note how the interface locations agree even after the interface has merged with itself, resulting in a non-physical loop in the marker particle solution.

Finally, to illustrate how the lines of constant speed are not, in fact, straight, we overlay the lines of constant F onto the contours of ϕ in Figure 3.8. Note how the characteristic curves are not straight, as assumed by the old method, but bend according to the gradient of F along the interface. Furthermore, note how these

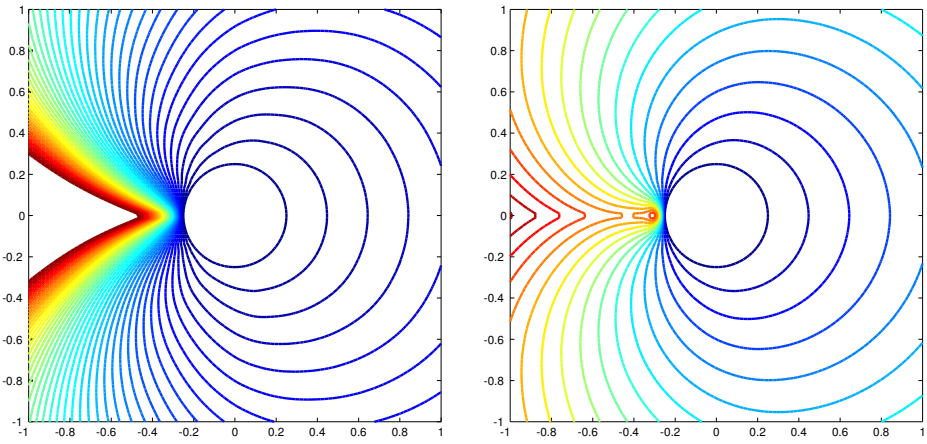

FIG. 3.6. *Comparison of old velocity extension method (left) with new velocity extension method (right).*

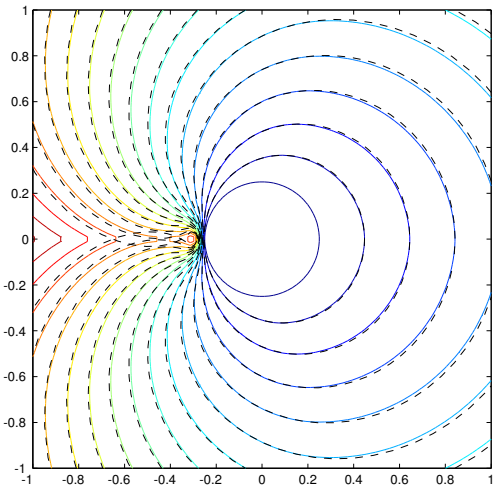

FIG. 3.7. *Comparison of new velocity extension method with marker particles method. Marker particles are indicated by dashed lines.*

characteristics remain orthogonal to the propagating interface as required.

4. Piecewise Fast Marching Method. We take the modified method described in the previous section, and now consider the case where the initial interface is no longer a closed loop, but a path with endpoints. We also require the normal speed of the path be fixed at zero at the endpoints. The problem then becomes, find

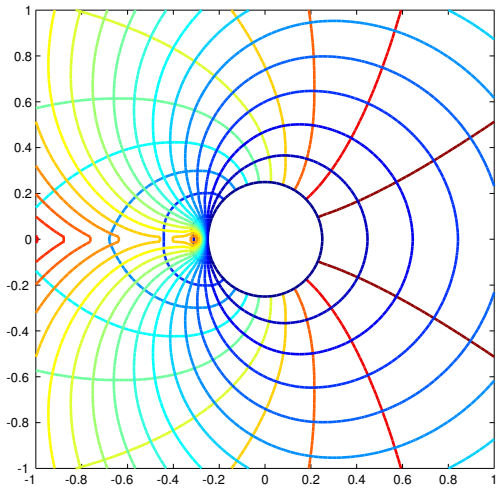

FIG. 3.8. *Lines of constant speed F overlayed on the isocontours of ϕ.*

functions ϕ, F such that

$$\begin{aligned} F\|\nabla\phi\| &= 1, \\ \nabla F \cdot \nabla\phi &= 0, \\ \phi(\mathbf{x}) &= 0, \text{ for } \mathbf{x} \in \Gamma, \\ F(\mathbf{x}) &> 0, \text{ given, for } \mathbf{x} \in \Gamma, \\ F(\mathbf{x}) &= 0, \text{ for } \mathbf{x} \in \partial\Gamma, \end{aligned} \tag{4.1}$$

where Γ is a given path in two-dimensional space, with $\partial\Gamma$ its endpoints. An analogous description can be written for higher dimensions.

4.1. Description of the Method. With the velocity now being computed dynamically with the evolving interface, a relatively small modification to the original method is required to solve system (4.1). The modification is contained entirely in the way the nodes, which are chosen to be initially accepted, are identified.

To begin, we assume there is a closed loop, Γ', such that $\Gamma \subset \Gamma'$, so that Γ' could serve as the initial contour for the original fast marching method. Also, assume that F can be extended to F' on Γ' in such a way that $F' = F$ on Γ and $F'(\mathbf{x}) < 0$ for $\mathbf{x} \in \Gamma' \setminus \Gamma$. Recall the initialization procedure described in section 3.3. There, nodes which are immediately adjacent to the initial interface are identified as being accepted. To solve system (4.1), a node is initially accepted only if:

1. the node would be initially accepted using the initialization procedure of section 3.3, and
2. the computed extended velocity from the initialization procedure is positive.

These accepted nodes are further distinguished by labelling those nodes $\mathbf{x}$ for which $\phi(\mathbf{x}) < 0$ as *pre-accepted*.

At this points, the nodes are segregated into pre-accepted, accepted, and far. For the fast marching method to proceed, we must select points from the far set into the tentative set. In the original fast marching method, all accepted points would have each of their neighboring nodes become tentative, and assigning values for ϕ and F at

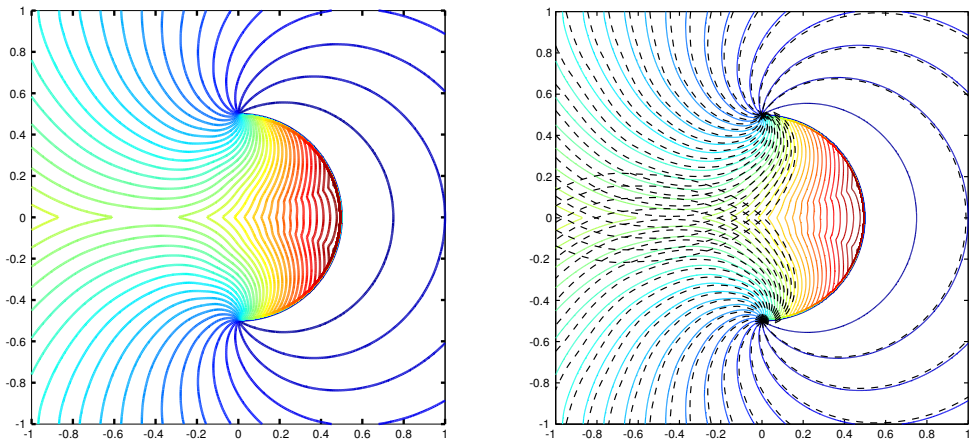

FIG. 4.1. *Example of the piecewise fast marching method for an initial semicircle and a comparison with a marker particle solution.*

those nodes according to (3.6), (3.7). For the piecewise fast marching method, this is also only done for the accepted points, and *not* for the pre-accepted points.

The distinction between accepted nodes and pre-accepted nodes is subtle, but important for this application. This effectively forces the fast marching method to advance *only* in the forward direction. Otherwise, it would advance in both the positive and negative directions.

Once the initially pre-accepted, accepted, and tentative points have been identified, the remainder of the method follows the algorithm presented in section 3.2.

As an example of this piecewise fast marching method, consider the initial semicircle $\Gamma = \{(x,y) : \sqrt{x^2+y^2} = 1/2,\ x \geq 0\}$ with initial speed $F_0(x,y) = x$. The result of the piecewise method are shown in Figure 4.1. See how the surface wraps around the endpoints to collide with the initial surface. Also in Figure 4.1, we compare the solution with the marker particle method to verify the solution. Note that the comparison with marker particles is not as good on the left side after the interface has wrapped around. Part of this error can be attributed to the fact that the corresponding characterstic curves, which the marker particle method tracks, leave the computational domain on the right side, and reenter on the left side. This new fast marching method must use extrapolation to estimate the proper speed, and this introduces errors in the estimate of F on the outer boundary.

A second observation about the results is that the fast marching solution is not as good as it approaches the initial curve, Γ, from the left. Part of this error can be attributed to the problem of exiting characteristics, but part of it is can also be attributed to the influence of the initial curve Γ on the evolving front. To solve this problem, we can use branch cuts to allow the evolution to proceed for even longer times, as will be described next.

A second example is shown in Figure 4.2, where $F \leq 0$ on the left semicircle. Again, a comparison with a marker particle solution is provided. In both examples, aside from the two identified discrepancies between the fast marching and marker particle methods, the agreement is good.

4.2. Using Branch Cuts for Longer Times. The identification of pre-accepted nodes can also serve as an identifying marker for longer time computations for this

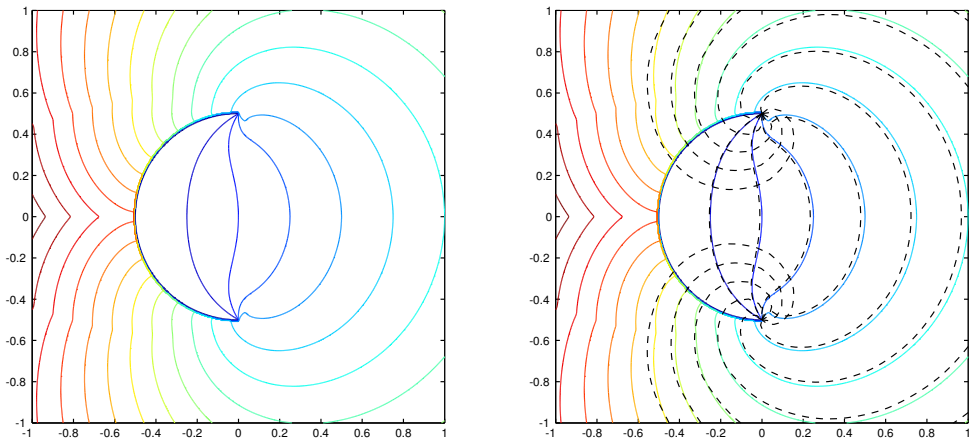

FIG. 4.2. *Example of the piecewise fast marching method for an initial semicircle and a comparison with a marker particle solution.*

problem. For system (4.1), it is expected that the interface will bend around the endpoints which are held fixed, and eventually wrap around and cross the original interface location. On a single mesh, this means the computation terminates because each node can only be crossed once. However, the computation can continue if the pre-accepted nodes are identified as the location for a branch cut. Additional meshes can be used, and the pre-accepted nodes indicate how two neighboring meshes are connected (think of a Riemann surface with a single branch cut).

For illustrative purposes, we include an example of using the branch cut idea. Suppose we allow the solution depicted in Figure 4.1 to continue beyond the point where it wraps around upon itself. By using the branch cut idea, we can continue the calculation. In Figure 4.3, we show what happens in this case. The location of the initial interface/branch cut is left in the figure, and two layers of solution ϕ are superimposed on the one plot. We have shown the later stages of this example so that the wrapping around can be seen to be smooth across the branch cut. By adding more layers, this calculation can continue. However, as noted earlier, the validity of the solution deteriorates with longer time due to the exit and reentry of the characteristic curves of the solution. Again, we show a comparison with the marker particle solution. Note that the dove-tailing which plagues marker particle type methods has been removed from the plot for clarity of the comparison.

5. The Bidirectional Fast Marching Method. We can now take the method described in section 4 to build a new bidirectional fast marching method. This new method can be used in a manner similar to how an implicit time-stepping level set method would be used. It allows for an arbitrarily large time step without losing stability. Of course, the same cautions also apply: taking time steps which are too large will result in degredation of accuracy. Nonetheless, this method provides a means of achieving the stability of implicit methods without having to solve a non-linear set of implicit equations, usually done using iterative methods. A brief description of how this algorithm can be used to circumvent explicit time-stepping restrictions is given at the end of this section.

5.1. Implementation Details. The implementation of the bidirectional fast marching method is straightforward from the piecewise fast marching method in the

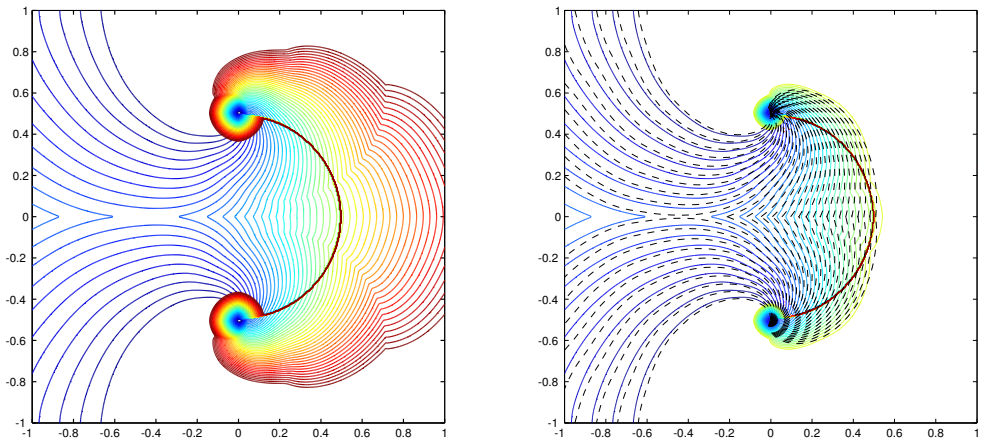

FIG. 4.3. *Example of using branch cuts to extend the time the evolution is allowed to proceed. Comparison with marker particles is on the right.*

previous section. Suppose now that we must find ϕ, F such that

$$\begin{aligned} F\|\nabla\phi\| &= 1, \\ \nabla F\cdot\nabla\phi &= 0, \\ \phi(\mathbf{x}) &= 0, \text{ for } \mathbf{x}\in\Gamma \\ F(\mathbf{x}) &\text{ given, for } \mathbf{x}\in\Gamma. \end{aligned} \tag{5.1}$$

Note that if F is monotonic, then the method of section 3.2 suffices.

We are now interested in the case where F is C^1 and is not monotonic. We break the initial front into two pieces, Γ^+, Γ^- where $\Gamma^+ = \{\mathbf{x}\in\Gamma : F(\mathbf{x})\geq 0\}$ and $\Gamma^- = \{\mathbf{x}\in\Gamma : F(\mathbf{x})\leq 0\}$. The piecewise fast marching method is now applied separately to Γ^+, Γ^-.

Let ϕ^+, ϕ^-, be the solutions from the piecewise fast marching method for with initial front Γ^+, Γ^-, respectively. Recombination of these two solutions cannot be done into a single function ϕ due to overlap: each point in the plane will have two first arrival times, one from ϕ^+ and one from ϕ^-. However, it is still possible to reconstruct the location of the moving interface, for the original problem (5.1), for any specified time $t\geq 0$ using the following formula:

$$\tilde{\phi}(\mathbf{x}) = t+\min(|\phi^+(\mathbf{x})-t|,|\phi^-(\mathbf{x})-t|)\,\text{sign}(\phi^+(\mathbf{x})-t)\,\text{sign}(\phi^-(\mathbf{x})-t)\,\text{sign}(\phi_0). \tag{5.2}$$

where ϕ_0 is a level set representation of Γ, $\Gamma=\{\mathbf{x}:\phi_0(\mathbf{x})=0\}$, and with the property

$$\nabla\phi_0(\mathbf{x})\cdot\nabla\phi^+(\mathbf{x})>0, \text{ for } \mathbf{x}\in\Gamma^+, \tag{5.3}$$

$$\nabla\phi_0(\mathbf{x})\cdot\nabla\phi^-(\mathbf{x})<0, \text{ for } \mathbf{x}\in\Gamma^-. \tag{5.4}$$

Note that $\text{sign}(y)$ is defined by

$$\text{sign}(y) = \begin{cases} 1 & y\geq 0 \\ -1 & y<0 \end{cases}. \tag{5.5}$$

5.2. Examples. As a simple example, we can take the two flows shown in Figure 4.1, 4.2, and combine them according to (5.2) to produce the results shown in Figure 5.1.

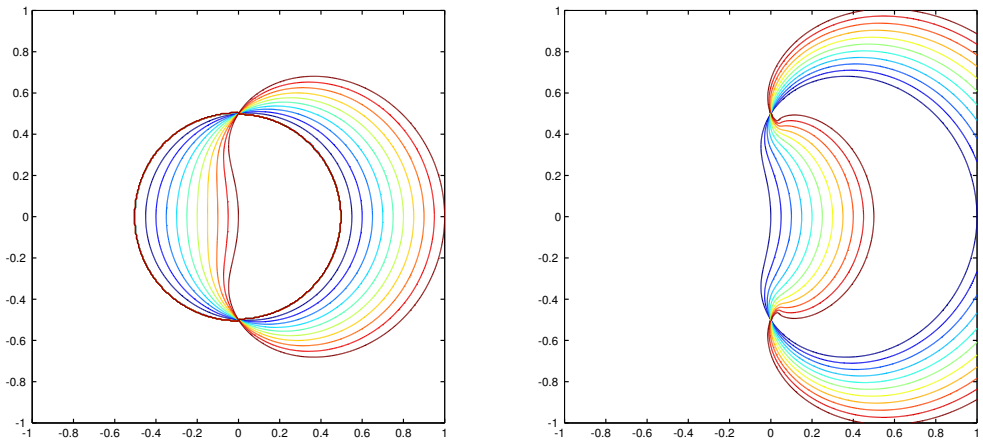

FIG. 5.1. *Example of bidirectional fast marching method, with* $F_0(x,y) = x$ *and* Γ *a circle of radius* $1/2$. *Evolution is shown in time steps of* $\Delta t = 0.1$, *with* $0 \le t \le 1$ *on the left, and* $1 \le t \le 2$ *on the right.*

5.3. Circumventing the CFL Condition. The formula in (5.2) is important because it allows the level set method to circumvent the explicit time-step restriction imposed by the Courant-Friedrichs-Lewy condition. Given an interface, Γ, with a prescribed speed function, F, then (5.2) gives the formula for the location of the interface at time t. To use this equation as an implicit time step for a level set method, then t is subtracted from the right hand side of (5.2) so that the zero level set of $\tilde{\phi}$ is the new location of the interface at time t. If desired, a reinitialization step can subsequently be applied so that the resulting level set function is again a signed distance function, e.g. see [4]. The size of t can be taken as large or small as desired without any restrictions for stability.

6. Conclusion. In this paper, we have presented a novel modification to the fast marching method, which allows it to solve a static Hamilton-Jacobi equation without the restriction that F be monotonic. This modification was achieved by solving both the velocity extension and the fast marching method equations simultaneously, then treating the cases of $F > 0$ and $F < 0$ separately. Those two pieces could then be recombined to produce the desired solution.

The primary application of this method will be to give level set methods the stability of implicit time-stepping methods without having to solve a large non-linear system of equations, which typically requires an iterative solver. This method will be most suitable for problems where the computation of F is too expensive for an explicit level set method implementation.

Another key result from this paper, is a description of how to properly do velocity extensions for fast marching methods. We demonstrated how the currently used velocity extension methods are inadequate for this application, and we have shown an alternative method, which more accurately extends the velocity so that it is sensitive to the evolving interface.

REFERENCES

[1] *A Source Book in Mathematics 1200–1800*, Princeton University Press, Princeton, New Jersey, 1990.
[2] D. Adalsteinsson and J. Sethian, *A fast level set method for propagating interfaces*, Journal of Computational Physics, 118 (1995), pp. 269–277.
[3] D. Chopp, *Computing minimal surfaces via level set curvature flow*, Journal of Computational Physics, 106 (1993), pp. 77–91.
[4] D. L. Chopp, *Some improvements of the fast marching method*, SIAM J. Sci. Comp., 23 (2001), pp. 230–244.
[5] H. Ji, D. Chopp, and J. E. Dolbow, *A hybrid extended finite element/level set method for modeling phase transformations*, International J. for Numerical Methods in Engineering, 54 (2002), pp. 1209–1233.
[6] C. Y. Kao, S. Osher, and Y. H. Tsai, *Fast sweeping methods for static Hamilton-Jacobi equations*, sinum, 42 (2005), pp. 2612–2632.
[7] S. Osher and R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*, Springer Verlag, Heidelberg, 2002.
[8] S. Osher and J. A. Sethian, *Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations*, Journal of Computational Physics, 79 (1988), pp. 12–49.
[9] D. Peng, B. Merriman, S. Osher, H.-K. Zhao, and M. Kang, *A PDE-based fast local level set method*, jcp, 155 (1999), pp. 410–438.
[10] G. Russo and P. Smereka, *A remark on computing distance functions*, jcp, 163 (2000), pp. 51–67.
[11] J. Sethian, *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision and Material Science*, Cambridge University Press, 1996.
[12] ———, *A marching level set method for monotonically advancing fronts*, Proceedings of the National Academy of Sciences, 93 (1996), pp. 1591–1595.
[13] ———, *Fast marching methods*, SIAM Review, 41 (1999), pp. 199–235.
[14] J. A. Sethian and A. Vladimirsky, *Order upwind methods for static Hamilton-Jacobi equations: Theory and algorithms*, sinum, 41 (2003), pp. 325–363.
[15] M. Sussman and E. Fatemi, *An efficient interface-preserving level set redistancing algorithm and its application to interfacial incompressible fluid flow*, SIAM J. Sci. Comput., 20 (1999), pp. 1165–1191.
[16] H. Zhao, T. Chan, B. Merriman, and S. Osher, *A variational level set approach to multiphase motion*, Journal of Computational Physics, 127 (1996), pp. 179–195.