Level-Set Method to Solve the 2D Fisher–Stefan Model

Alexander K. Y. Tam[*1] **and Matthew J. Simpson**[1]

[1]School of Mathematical Sciences, Queensland University of Technology, Brisbane, Queensland 4001, Australia.

November 16, 2021

In this document, we describe numerical methods used to solve the Fisher–Stefan model in two spatial dimensions. We have implemented the methods described in a Julia package that is publicly available on GitHub.

1 Level-Set Method

We implement the level-set method [1, 2] to compute solutions to the two-dimensional Fisher–Stefan model. The model involves solving the Fisher–KPP equation on a moving boundary, whose position is determined by a Stefan condition. We denote the region inside the boundary as $\Omega(t)$, and the boundary itself as $\partial\Omega(t)$. The region $\Omega(t)$ exists in a two-dimensional computational domain, denoted by $\mathcal{D}$, and we assume that the interface $\partial\Omega(t)$ does not cross the boundary of $\mathcal{D}$. We then consider the two configurations illustrated in Figure 1.1, depending on whether $\Omega(t)$ crosses the boundary of $\mathcal{D}$. If not (as in Figure 1.1a), we refer to this as a *spreading/extinction* problem. If so (as in Figure 1.1b), we refer to this as a *hole closing* problem.

In general, the dimensional Fisher–Stefan model reads

$$\frac{\partial u}{\partial t} = D\nabla^2 u + \lambda u\,(1-u) \quad \text{on} \quad \Omega(t), \tag{1.1a}$$

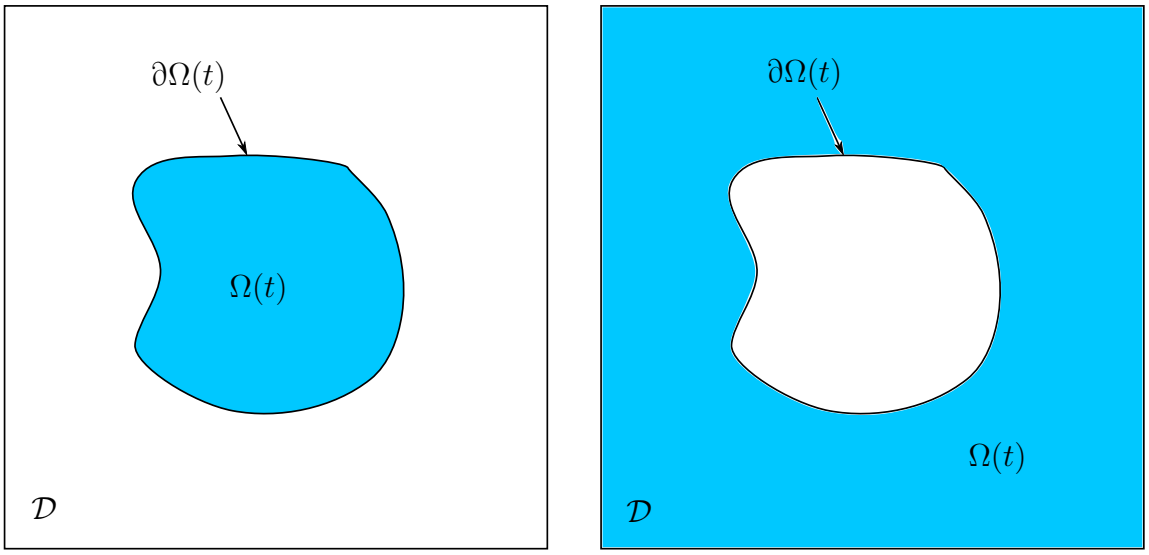
$$u(\boldsymbol{x}, t) = u_f \quad \text{on} \quad \partial\Omega(t), \tag{1.1b}$$

$$V = -\kappa \nabla u \cdot \hat{\boldsymbol{n}} \quad \text{on} \quad \partial\Omega(t), \tag{1.1c}$$

$$u(\boldsymbol{x}, 0) = u_0(\boldsymbol{x}) \quad \text{on} \quad \Omega(0), \tag{1.1d}$$

where $u(\boldsymbol{x}, t)$ is the density of the relevant quantity (*e.g.* cells). The parameter D is the diffusivity, λ is the reaction rate, and u_f is the constant density at the interface. For cell problems with sharp-fronted profiles, we take $u_f = 0$. The condition (1.1c) is the multidimensional Stefan condition. The scalar function V, defined on $\partial\Omega(t)$, describes the outward normal speed of the interface. The parameter κ is the Stefan parameter, which describes loss or leakage at the interface. The unit outward normal vector to the interface is expressed in terms of the level-set function, $\hat{\boldsymbol{n}} = \nabla\phi/|\nabla\phi|$ on $\partial\Omega(t)$.

*Contact: alexander.tam@qut.edu.au



(a) Domain and interface for a spreading/extinction problem.

(b) Domain and interface for a hole-closing problem.

Figure 1.1: The two-dimensional computational domain for solving the Fisher–Stefan model using the level-set method. The density $u(\boldsymbol{x}, t)$ is defined in the region $\Omega(t)$ (shaded in blue), on which we solve the Fisher–KPP equation. The boundary of this region, $\partial\Omega(t)$, evolves according to the Stefan condition.

The level-set method enables Eulerian treatment of moving-boundary problems on arbitrarily-shaped domains, avoiding the need to parameterise the domain itself. The key feature is embedding the interface position as the zero of a scalar function $\phi(\boldsymbol{x}, t)$, termed the level-set function. That is,

$$\partial\Omega(t) = \{\boldsymbol{x} \mid \phi(\boldsymbol{x}, t) = 0\}, \tag{1.2}$$

where $\phi(\boldsymbol{x}, t)$ is defined everywhere in the computational domain, $\mathcal{D}$. The function $\phi(\boldsymbol{x}, t)$ describes the position of $\partial\Omega$ implicitly, because its zero level-sets evolve with time according to the level-set equation

$$\frac{\partial\phi}{\partial t} + F\,|\nabla\phi| = 0, \tag{1.3}$$

where $F(\boldsymbol{x}, t)$ is a scalar function defined on $\mathcal{D}$, and is equal to the normal speed V on the interface. After introducing the level-set function, the numerical problem becomes

$$\frac{\partial u}{\partial t} = D\nabla^2 u + \lambda u\,(1-u) \quad \text{on} \quad \Omega(t), \tag{1.4a}$$

$$\frac{\partial\phi}{\partial t} + F\,|\nabla\phi| = 0 \quad \text{on} \quad \mathcal{D}, \tag{1.4b}$$

$$F = -\kappa\nabla u\cdot\hat{\boldsymbol{n}} \quad \text{on} \quad \partial\Omega(t), \tag{1.4c}$$

$$u = u_f \quad \text{on} \quad \partial\Omega(t), \tag{1.4d}$$

$$u(\boldsymbol{x}, 0) = u_0(\boldsymbol{x}) \quad \text{on} \quad \Omega(0). \tag{1.4e}$$

In level-set schemes, it is desirable for ϕ to denote the signed-distance from the interface [2], such that $\phi(\boldsymbol{x}, t) < 0$ at all points $\boldsymbol{x} \in \Omega(t)$, and $\phi(\boldsymbol{x}, t) > 0$ at all points $\boldsymbol{x} \notin \Omega(t)$. The zero level-set, $\phi = 0$, defines the interface $\partial\Omega(t)$.

2 Numerical Algorithm

We solve (1.4) on a computational domain $\mathcal{D}$. In two-dimensions, we use the rectangular domain $[0, L_x] \times [0, L_y]$, where L_x and L_y are the widths in the x and y-directions respectively. We compute the solution on an equispaced discrete grid (x_i, y_j). The discrete grid points (or *nodes*) are given by $x_i = i\Delta x$, and $y_j = j\Delta y$, for $i = 0, \dots N_x$, and $j = 0, \dots N_y$, where N_x and N_y are the number of grid intervals in the x and y-directions respectively, and Δx and Δy are the (constant) grid spacing in the respective direction. We discretise the temporal domain similarly, defining $t_k = k\Delta t$, where $k = 0, \dots N_t$, and Δt is constant. The symbols $u^k_{i,j} = u(x_i, y_j, t_k)$, $\phi^k_{i,j} = \phi(x_i, y_j, t_k)$, and $F_{i,j} = F(x_i, y_j)$, then denote the value of each quantity at discrete points in time and space.

The procedure to solve the Fisher–Stefan model using the level-set method consists of the following steps:

1. Initialise the problem by defining the model parameters D, λ, and κ, and initial conditions $u_0(\boldsymbol{x})$ and $\phi_0(\boldsymbol{x})$.

2. Advance in time to solve for $u^{k+1}_{i,j}$, $\phi^{k+1}_{i,j}$, and $F^{k+1}_{i,j}$, for $i = 0, \dots N_x$, $j = 0, \dots N_y$, and $k = 0, \dots, N_t - 1$. At each time step, perform the following:

   (a) Identify the domain, $\Omega(t)$, and the interface position, $\partial\Omega(t)$.

   (b) Solve the Fisher–KPP equation (1.4a) on $\Omega(t)$ to obtain $u^{k+1}_{i,j}$, subject to the boundary condition (1.4d).

   (c) Compute the interface velocity, V, using the Stefan condition (1.4c). Subsequently, extend the velocity field to obtain a speed function $F(\boldsymbol{x}, t)$ defined for all $\boldsymbol{x} \in \mathcal{D}$.

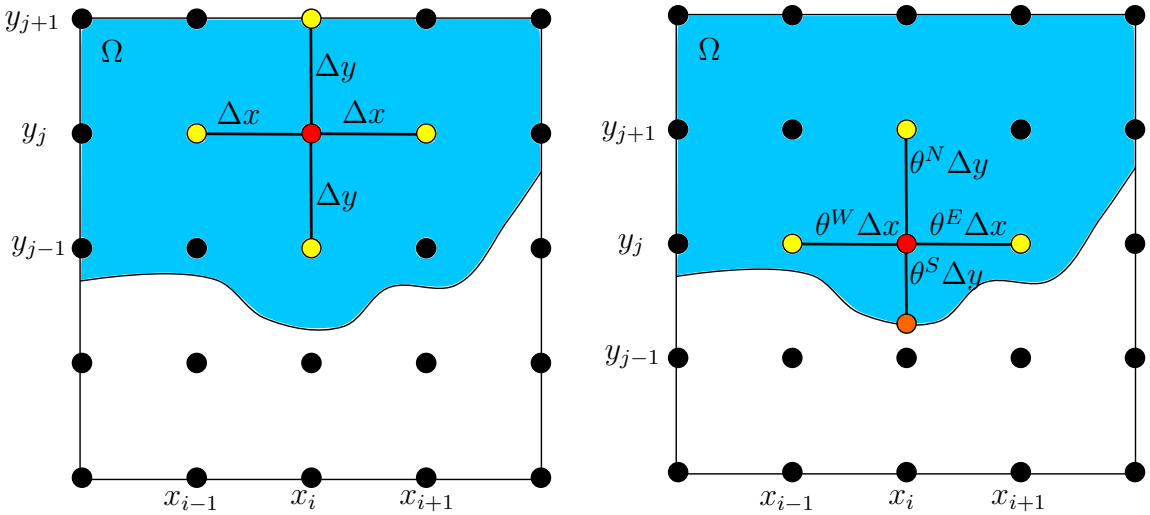
   (d) Solve the level-set equation (1.4b) to obtain $\phi^{k+1}_{i,j}$, which updates the interface position.

   (e) (Optional) Reinitialise the level-set function to a signed-distance function in an appropriate neighbourhood of the interface.

We implement this procedure using code developed in Julia. A detailed description of the methods used for steps 2.(a)–(e) is provided in subsequent subsections.

2.1 Identifying the Domain and Interface

After specifying the parameters and initial conditions, the first step is to identify the domain $\Omega(t)$ and interface $\partial\Omega(t)$. To locate $\Omega(t)$, we classify all interior grid points (x_i, y_j), for $i = 1, \dots N_x - 1$, and $j = 1, \dots, N_y - 1$, as either inside or outside $\Omega(t)$, such that $(x_i, y_j) \in \Omega(t)$ if $\phi(x_i, y_j, t) < 0$, and $(x_i, y_j) \notin \Omega(t)$. To locate the interface $\partial\Omega(t)$, we identify all grid points (x_i, y_j) such that ϕ at any adjacent grid point has the opposite sign. We refer to these as *irregular* grid points (and points where ϕ has the same sign at all adjacent nodes as *regular* grid points).



(a) Regular grid point. (b) Irregular grid point.

Figure 2.1: Regular and irregular grid points in the computational domain. Black dots represent grid points, the red dot is the point (x_i, y_j), and yellow dots are nodes adjacent to (x_i, y_j) that are in Ω. For the irregular point in (b), the orange dot represents a ghost node on the interface $\partial\Omega$, in the southern direction from (x_i, y_j).

Figure 2.1 illustrates the difference between regular and irregular grid points. In Figure 2.1a, the red node is a regular grid point, because $\phi_{i,j} < 0$, and $\phi < 0$ at all adjacent (yellow) nodes. However, in Figure 2.1b the red node is irregular, because its southern neighbouring node (x_i, y_{j-1}) is outside $\Omega(t)$. To distinguish between regular and irregular grid points, we define $\theta^W_{i,j}$, $\theta^E_{i,j}$, $\theta^S_{i,j}$, and $\theta^N_{i,j}$ to be the distance (relative to grid spacing) between the grid point (x_i, y_j) and a neighbouring node (western, eastern, southern, or northern respectively) or the interface, whichever is smaller. For example,

$$\theta^S_{i,j} = \begin{cases} 1 & \text{if} \quad \operatorname{sgn}(\phi_{i,j}) = \operatorname{sgn}(\phi_{i,j-1}) \\ \dfrac{\phi_{i,j}}{\phi_{i,j} - \phi_{i,j-1}} & \text{otherwise,} \end{cases} \tag{2.1}$$

where we apply linear interpolation at irregular grid points to find the zero level-set ($\phi = 0$) if the interface passes between (x_i, y_j) and its southern neighbour. If so, the interface ghost node (orange dot in Figure 2.1b) (zero level-set) position is (x_i, y^G), where

$y^G = y_j - \theta^S_{i,j}\Delta y$. Similar interpolation formulae can be developed for $\theta^W_{i,j}$, $\theta^E_{i,j}$, and $\theta^N_{i,j}$ as necessary, and we store their values at each interior grid point in $\mathcal{D}$. These enable development of finite-difference methods for the Fisher–KPP equation and extension velocity field that respect the shape of Ω and the interface location.

2.2 Fisher–KPP Equation

We solve the Fisher–KPP equation using the explicit finite-difference method of lines. At each interior grid point in $\mathcal{D}$, we discretise the right-hand side of (1.4a) using second-order accurate finite-difference approximations for the derivatives. This yields a semi-discrete system of ordinary differential equations (ODEs) of the form

$$\frac{\mathrm{d}u_{i,j}}{\mathrm{d}t} = \begin{cases} D\left(\nabla_d^2 u_{i,j}\right) + \lambda u_{i,j}\left(1 - u_{i,j}\right) & \text{if} \quad \phi(x_i, y_j) < 0 \\ 0 & \text{if} \quad \phi(x_i, y_j) \geq 0, \end{cases} \tag{2.2}$$

for $i = 0, \ldots, N_x$, and $j = 0, \ldots, N_y$. We apply Dirichlet boundary conditions at all nodes that lie on boundaries of the computational domain. For spreading/extinction problems (Figure 1.1a), we use the Dirichlet conditions $u_{0,j}(t) = u_{N_x,j}(t) = u_{i,0}(t) = u_{i,N_y}(t) = u_f$, whereas for hole-closing problems (Figure 1.1b) we set $u_{0,j}(t) = u_{N_x,j}(t) = u_{i,0}(t) = u_{i,N_y}(t) = 1$. The term $\nabla_d^2 u_{i,j}$ denotes the second-order central finite-difference approximation for the Laplacian, $\nabla^2 u = u_{xx} + u_{yy}$.

The finite-difference discretisation of $\nabla^2 u$ differs depending on whether (x_i, y_j) is regular or irregular. At regular grid points, we apply the standard five-point central finite-difference stencil,

$$\nabla_d^2 u_{i,j} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2}. \tag{2.3}$$

However, we cannot use (2.3) at irregular grid points, because one or more of the grid points on the standard stencil lies outside Ω. Instead, we introduce ghost nodes on the interface (for example the orange dot in the southern direction in Figure 2.1b) to replace nodes outside Ω that would otherwise be in the standard stencil. The density at these ghost nodes adheres to the interface condition (1.4d), $u = u_f$. With our definitions of $\theta^K_{i,j}$, $K \in \{W, E, S, N\}$ above, this gives rise to the non-standard finite-difference stencil (2.4)

$$\begin{aligned} \nabla_d^2 u_{i,j} = {} & \frac{2}{(\Delta x)^2}\left[\frac{u^E_{i,j}}{\theta^E_{i,j}(\theta^W_{i,j} + \theta^E_{i,j})} - \frac{u_{i,j}}{\theta^W_{i,j}\theta^E_{i,j}} + \frac{u^W_{i,j}}{\theta^W_{i,j}(\theta^W_{i,j} + \theta^E_{i,j})}\right] \\ & + \frac{2}{(\Delta y)^2}\left[\frac{u^N_{i,j}}{\theta^N_{i,j}(\theta^S_{i,j} + \theta^N_{i,j})} - \frac{u_{i,j}}{\theta^S_{i,j}\theta^N_{i,j}} + \frac{u^S_{i,j}}{\theta^S_{i,j}(\theta^S_{i,j} + \theta^N_{i,j})}\right], \end{aligned} \tag{2.4}$$

where the superscripts W, E, S, and N denote the western, eastern, southern, and northern neighbours to (x_i, y_j) respectively. The values $u^W_{i,j}$, $u^E_{i,j}$, $u^S_{i,j}$, and $u^N_{i,j}$ depend on whether

the neighbouring nodes lie inside $\Omega(t)$. For example, we have

$$u_{i,j}^{W} = \begin{cases} u_{i-1,j} & \text{if} \quad \phi_{i-1,j} < 0 \\ u_f & \text{if} \quad \phi_{i-1,j} \geq 0, \end{cases} \tag{2.5}$$

and similar for $u_{i,j}^{E}$, $u_{i,j}^{S}$, and $u_{i,j}^{N}$. We note that if $\theta_{i,j}^{K} = 1$, then the irregular stencil (2.4) is identical to the regular stencil (2.3). The approximation (2.4) can involve division by small quantities if any $\theta_{i,j}^{K}$ ($K \in \{N, E, S, W\}$) is small. If so, small time-steps are required in the method of lines to retain numerical stability. Following Morrow et al. [3], we circumvent this difficulty by setting $u_{i,j} = u_f$ if any $\theta_{i,j}^{K} < \theta_{\partial\Omega}$, for some threshold $\theta_{\partial\Omega}$. In all computed solutions, we use $\theta_{\partial\Omega} = 0.01$.

After constructing the system of ODEs (2.2) at all interior grid points in the computational domain, we integrate in time using the fifth-order Tsitouras Runge–Kutta method [4]. The calculation is performed using the in-built ODE solver in `DifferentialEquations.jl` [5]. At each time step in our algorithm, we solve to advance in time by an increment of Δt. Depending on the numerical stability of (2.2), the solver might take multiple intermediate time steps. We retain this adaptive time-stepping for accuracy, and apply the absolute tolerance of 1×10^{-3} and relative tolerance of 1×10^{-6}.

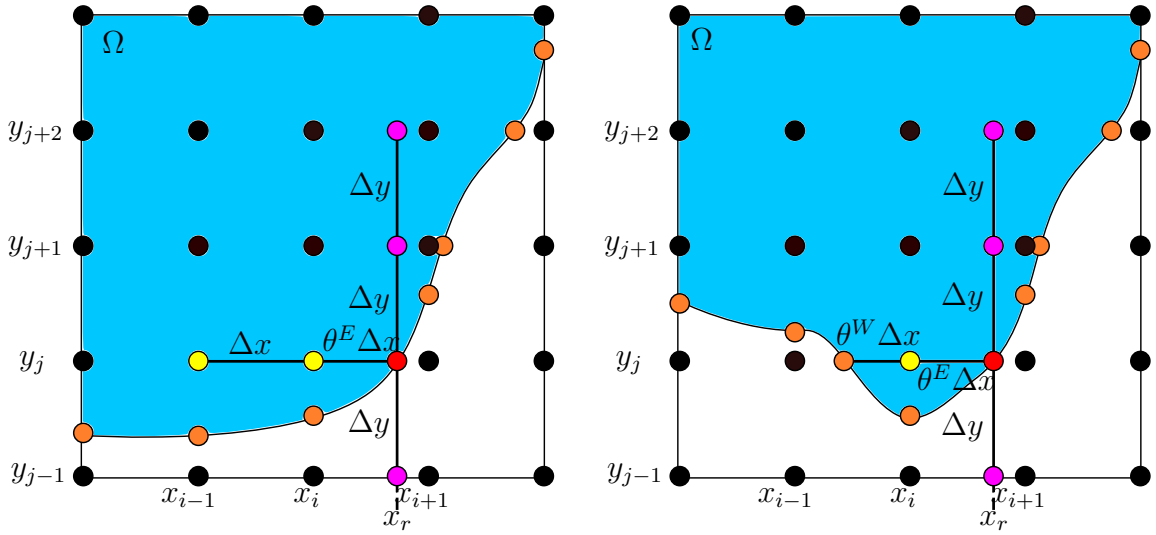
2.3 Constructing the Extension Velocity Field

Once the density $u(\boldsymbol{x}, t)$ has been updated according to Fisher–KPP, the next step is to compute an extension velocity field, $F(\boldsymbol{x}, t)$, which is used to evolve the interface. Constructing $F(\boldsymbol{x}, t)$ is a two-step process. First, we need to obtain the normal speed on the interface, V, by implementing the Stefan condition (1.4c). Next, we extend this velocity to construct the field defined for all $\boldsymbol{x} \in \mathcal{D}$. We use non-standard, second-order finite-differences and linear interpolation to implement the Stefan condition. Then, we extrapolate the velocity field in the direction orthogonal to the interface by solving a partial differential equation (PDE).

2.3.1 Computing the Interface Velocity

We apply the Stefan condition at all ghost nodes on the interface (orange nodes in Figure 2.1 and subsequent). To apply (1.4c), we implement second-order accurate finite-differences for the first spatial derivatives of u and ϕ, denoted here by the subscripts u_x, u_y, ϕ_x, and ϕ_y. The finite-difference scheme depends on the location of the interface and the adjacent nodes. Without loss of generality, here we describe the procedure used if the ghost node lies in the x-direction between two grid points, and if the region $\Omega(t)$ lies directly west and north of the ghost node (see Figure 2.2). Although this is not the only possible configuration, where necessary different approximations can be developed

similarly. For example, if Ω lies in the eastern rather than western direction, we apply forward differencing to compute u_x instead of the backward differences developed here.



(a) Regular. (b) Irregular.

Figure 2.2: Finite-difference stencils used to compute the interface normal speed for grid points with $\theta^E_{i,j} > \theta_{\partial\Omega}$. Black dots denote nodes of the computational grid. The red dots denote the ghost node at which we compute V using the Stefan condition, and orange dots denote other ghost nodes on the interface. Yellow nodes denote grid points used in the stencils for u_x, and pink dots denote ghost nodes used in the stencils for u_y.

To approximate u_x, in general we apply a second-order accurate finite-difference approximation. If the grid point to the left of the ghost node are in Ω, as in Figure 2.2, this takes the form

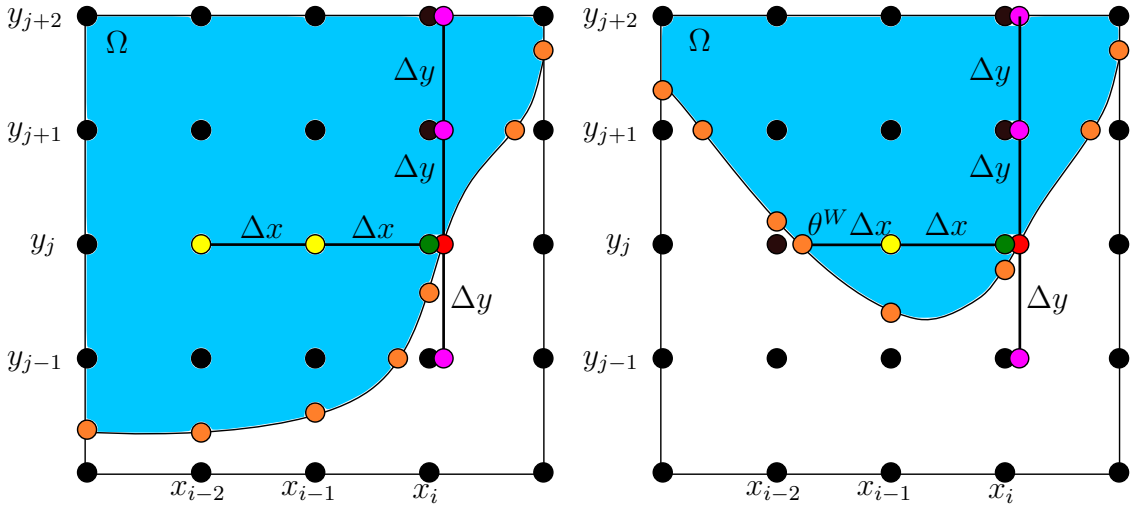
$$u_x = \frac{1}{\Delta x}\left[\frac{(2\theta^E_{i,j} + \theta^W_{i,j})u_f}{\theta^E_{i,j}(\theta^W_{i,j} + \theta^E_{i,j})} - \frac{(\theta^W_{i,j} + \theta^E_{i,j})u_{i,j}}{\theta^W_{i,j}\theta^E_{i,j}} + \frac{\theta^E_{i,j}u^W_{i,j}}{\theta^W_{i,j}\theta^W_{i,j} + \theta^E_{i,j})}\right]. \tag{2.6}$$

The value $u^W_{i,j}$ depends on whether the node (x_{i-1}, y_j) is in Ω, and is given by (2.5). Note, that if $\theta^W_{i,j} = \theta^E_{i,j} = 1$, then (2.6) is the standard second-order one-sided difference for u_x.

Like the non-standard difference formulae (2.4) used when solving Fisher–KPP, the scheme (2.6) can encounter division by small quantities if $\theta^W_{i,j} \ll 1$ and/or $\theta^E_{i,j} \ll 1$. Thus, if $\theta^E_{i,j} < \theta_{\partial\Omega}$, we instead assume that the interface lies exactly on (x_i, y_j). Provided $(x_{i-1}, y_j) \in \Omega$ and $(x_{i-2}, y_j) \in \Omega$, we apply the standard formula

$$u_x = \frac{3u_f - 4u_{i-1,j} + u_{i-2,j}}{2\Delta x}. \tag{2.7}$$

This scenario is illustrated in Figure 2.3a. If $(x_{i-1}, y_j) \notin \Omega$, we set $u_x = 0$. If $(x_{i-1}, y_j) \in \Omega$ but $(x_{i-2}, y_j) \notin \Omega$ (as in Figure 2.3b), we adjust (2.7) to obtain an appropriate non-standard three-point stencil. The formula is given by (2.6) evaluated at the point (x_{i-1}, y_j), with $\theta^E_{i-1,j} = 1$, and $u^W_{i-1,j} = u_f$. If $\theta^W_{i-1,j} < \theta_{\partial\Omega}$, we instead set $u_x = 0$.



(a) A (b) A

Figure 2.3: Finite-difference stencils used to compute the interface normal speed for grid points with $\theta^E_{i,j} < \theta_{\partial\Omega}$. Orange and red nodes denote ghost nodes on the interface, at which we compute the speed using the Stefan condition. Yellow nodes denote grid points used in the stencils for u_x, and pink dots denote ghost nodes used in the stencils for u_y. The green dot indicates the grid point close to the interface, where we assume $u_{i,j} = u_f$. Black dots denote nodes of the computational grid.

After approximating u_x, the next step is to obtain the derivatives of the level-set function, ϕ_x and ϕ_y. To compute ϕ_x, we first apply second-order central finite-differences at the two grid points adjacent to the interface ghost node. This yields

$$(\phi_{i,j})_x = \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2\Delta x}, \tag{2.8a}$$

$$(\phi_{i+1,j})_x = \frac{\phi_{i+2,j} - \phi_{i,j}}{2\Delta x}. \tag{2.8b}$$

To approximate the value of ϕ_x on the interface ghost node, we then interpolate linearly to obtain

$$\phi_x = \theta^E_{i,j} (\phi_{i+1,j})_x + \left(1 - \theta^E_{i,j}\right) (\phi_{i,j})_x . \tag{2.9}$$

To approximate ϕ_y, we define two additional ghost nodes directly above and below the interface ghost node (pink dots in Figures 2.2 and 2.3). We then use linear interpolation to approximate ϕ at these nodes, which gives

$$\phi_{r,j+1} = \theta^E_{i,j}\phi_{i+1,j+1} + \left(1 - \theta^E_{i,j}\right) \phi_{i,j+1}, \tag{2.10a}$$

$$\phi_{r,j-1} = \theta^E_{i,j}\phi_{i+1,j-1} + \left(1 - \theta^E_{i,j}\right) \phi_{i,j-1}. \tag{2.10b}$$

The first derivative approximation is then obtained using the central scheme

$$\phi_y = \frac{\phi_{r,j+1} - \phi_{r,j-1}}{2\Delta y}. \tag{2.11}$$

Finally, to apply the Stefan condition (1.4c), we also need u_y. If the ghost node on the interface is in the x-direction, we require linear interpolation to approximate u_y. To apply a one-sided difference, we define two additional ghost nodes directly above the ghost node on the interface (see Figures 2.2 and 2.3), such that

$$u_{r,j} = u_f, \tag{2.12a}$$
$$u_{r,j+1} = \theta^E_{i,j} u_{i+1,j+1} + (1 - \theta^E_{i,j}) u_{i,j+1}, \tag{2.12b}$$
$$u_{r,j+2} = \theta^E_{i,j} u_{i+1,j+2} + (1 - \theta^E_{i,j}) u_{i,j+2}. \tag{2.12c}$$

The second-order one-sided finite-difference approximation is then

$$u_y = \frac{-3u_f + 4u_{r,j+1} - u_{r,j+2}}{2\Delta y}. \tag{2.13}$$

The approximation (2.13) assumes that both ghost nodes (x_r, y_{j+1}) and (x_r, y_{j+2}) lie within Ω, that is $\phi_{r,j+1} < 0$ and $\phi_{r,j+2} < 0$. The value of $\phi_{r,j+1}$ can be checked using (2.10b), and the value of $\phi_{r,j+2}$ can be estimated using similar linear interpolation,

$$\phi_{r,j+2} = \theta^E_{i,j} \phi_{i+1,j+2} + \left(1 - \theta^E_{i,j}\right) \phi_{i,j+2}. \tag{2.14}$$

If both $\phi_{r,j+1} < 0$ and $\phi_{r,j+2} < 0$, we use (2.13). If $\phi_{r,j+1} < 0$ but $\phi_{r,j+2} \geq 0$, we introduce another ghost node (see Figure 2.4) and define

$$\theta^G = \frac{\phi_{r,j+1}}{(\phi_{r,j+1} - \phi_{r,j+2})} \tag{2.15}$$

to be the distance from the x_r, y_{j+1} ghost node to the interface, scaled by Δy. For ghost points sufficiently far from the interface, that is $\theta^G \geq \theta_{\partial\Omega}$, then we use the approximation

$$u_y = \frac{1}{\Delta y}\left[-\frac{(2+\theta^G)u_f}{1+\theta^G} + \frac{(1+\theta^G)u_{r,j+1}}{\theta^G} - \frac{u_f}{\theta^G(1+\theta^G)}\right]. \tag{2.16}$$

To prevent division by small quantities in (2.16), we set $u_y = 0$ if $\theta^G < \theta_{\partial\Omega}$. Similar formulae can be developed if Ω is below the interface, that is $\phi_{r,j+1} \geq 0$, but $\phi_{r,j-1} < 0$. If both $\phi_{r,j-1} \geq 0$ and $\phi_{r,j+1} \geq 0$, we set $u_y = 0$. Once numerical approximations to u_x, u_y, ϕ_x, and ϕ_y are found at the interface, we apply the Stefan condition to obtain $V = -\kappa(u_x\phi_x + u_y\phi_y)$, and store the normal speed at each interface ghost node.

To summarise, the procedure for computing the velocity at a ghost node on the interface is:

1. Determine whether the ghost node lies in the x or y-direction of two grid points, and determine which point (left or right/bottom or top) lies in $\Omega(t)$.

2. Compute the first derivative of u in the direction in which the interface ghost point

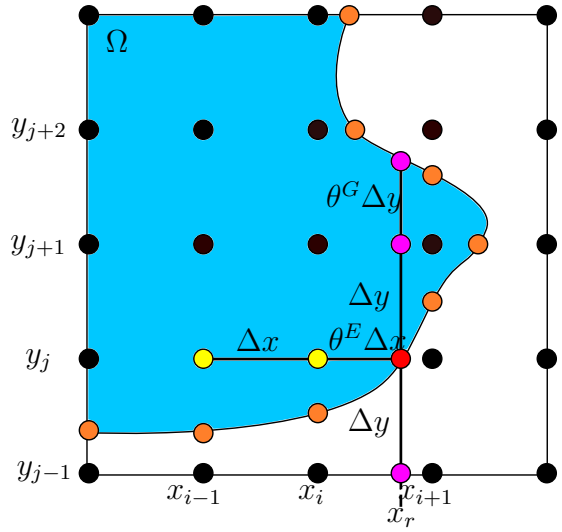


Figure 2.4: Finite-difference stencil used to compute u_y, where $\phi_{r,j+1} < 0$ but $\phi_{r,j+2} \geq 0$. The derivative is computed at the red ghost node, and pink nodes indicate the stencils used to compute u_y and ϕ_y. Black and yellow dots denote nodes in the computational grid, and orange dots are other ghost nodes on the interface.

lies, using a second-order accurate one-sided finite-difference. This involves applying either (2.7) or (2.6), depending on the interface position.

3. Approximate ϕ_x and ϕ_y at the ghost node using central differencing and linear interpolation, (2.9) and (2.11).

4. Compute the first derivative of u along the non-grid direction using a second-order accurate one-sided difference. Depending on the interface location, this involves applying either (2.13) or (2.16).

5. Once all derivatives of u and ϕ are computed, store the ghost node position and velocity $V = -\kappa(u_x\phi_x + u_y\phi_y)$.

2.3.2 Velocity Extrapolation Away From Interface

To obtain an extension velocity field, $F(x, y, t)$, defined throughout $\mathcal{D}$, we extrapolate V away from the interface in the orthogonal direction. We achieve this by solving the PDE

$$\frac{\partial F}{\partial \tau} + \nabla F \cdot \hat{\boldsymbol{n}} = 0, \tag{2.17}$$

subject to the boundary condition $F = V$ on $\partial\Omega$, where $\hat{\boldsymbol{n}}$ is the unit outward normal to the interface. To obtain F, we first solve (2.17) with the initial condition $F(\boldsymbol{x}, 0) = 0$ everywhere except $\partial\Omega$. This extrapolates the velocity in the outward normal direction.

We then use the solution to (2.17) as the initial condition and solve

$$\frac{\partial F}{\partial \tau} + \nabla F \cdot (-\hat{\boldsymbol{n}}) = 0, \tag{2.18}$$

to extrapolate in the inwards normal direction. To solve (2.17) and (2.18) on $\mathcal{D}$, the unit outward normal vector on the interface can be extended to the entire domain via the level-set function, $\hat{\boldsymbol{n}} = \nabla\phi/|\nabla\phi|$. Thus, if $\nabla\phi$ is known we can rewrite both (2.17) and (2.18) as the linear advection equation

$$\frac{\partial F}{\partial \tau} + a\frac{\partial F}{\partial x} + b\frac{\partial F}{\partial y} = 0. \tag{2.19}$$

To solve (2.17), we use $a = \phi_x/|\nabla\phi|$ and $b = \phi_y/|\nabla\phi|$. To solve (2.18) and extrapolate in the inwards normal direction, we use $a = -\phi_x/|\nabla\phi|$ and $b = -\phi_y/|\nabla\phi|$.

We solve (2.19) using the method of lines, where we approximate derivatives of x and y using first-order upwind finite-differences. At all interior grid points, we approximate the derivatives of ϕ using the central schemes

$$(\phi_{i,j})_x = \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2\Delta x}, \quad (\phi_{i,j})_y = \frac{\phi_{i,j+1} - \phi_{i,j-1}}{2\Delta y}. \tag{2.20}$$

This is sufficient to determine the advection coefficients a and b in (2.19). Then, to apply the method of lines we use the semi-discrete form

$$\frac{\mathrm{d}\phi_{i,j}}{\mathrm{d}\tau} = a\,(\nabla_x F)_{i,j} + b\,(\nabla_y F)_{i,j}\,, \tag{2.21}$$

where ∇_x and ∇_y are first-order upwind approximations to the first derivatives. For example, at regular grid points $\nabla_x F$ takes the form

$$(\nabla_x F)_{i,j} = \begin{cases} (F_{i+1,j} - F_{i,j})/\Delta x & \text{if} \quad a < 0 \\ (F_{i,j} - F_{i-1,j})/\Delta x & \text{if} \quad a \geq 0. \end{cases} \tag{2.22}$$

At irregular grid points, if the sign of a or b requires differencing in a direction that crosses the interface, we construct these derivatives using the computed speed on the interface ghost node. For example, if the interface lies to the right of the point (x_i, y_j) and $a < 0$, then we use

$$(\nabla_x F)_{i,j} = \frac{v - F_{i,j}}{\theta^E_{i,j}\Delta x}, \tag{2.23}$$

where v is the computed speed at the interface ghost node that lies between (x_i, y_j) and (x_{i+1}, y_j). If $\theta^E_{i,j} < \theta_{\partial\Omega}$, then we instead set $F_{i,j} = v$, and $(\nabla_x F)_{i,j} = (\nabla_y F)_{i,j} = 0$. Again, this is to prevent division by small quantities in (2.23). Other cases at irregular grid points are handled similarly.

When solving (2.17) and (2.18), we apply the Dirichlet boundary conditions $F_{0,j} = F_{N_x,j} = F_{i,0} = F_{i,N_y} = 0$. This is valid if the interface $\partial\Omega$ remains far from the boundaries of $\mathcal{D}$, which we assume. We integrate in the 'pseudo-time' τ using `DifferentialEquations.jl`, again using the fifth-order Tsitouras Runge–Kutta method with the absolute tolerance of 1×10^{-3} and relative tolerance of 1×10^{-6}. For both equations (2.17) and (2.18), we compute solutions until $\tau = 10\Delta\tau$, where $\Delta\tau = 0.5(\Delta x/5 + \Delta y/5)$. This ensures that the velocity field $F(\boldsymbol{x}, t)$ is accurate in a neighbourhood of the interface. In turn, this ensures that the zero level-set of ϕ remains an implicit description of the interface position after solving the level-set equation.

2.4 Solving the Level-Set Equation

The level-set equation (1.4b) is a multidimensional Hamilton–Jacobi PDE of the form

$$\frac{\partial \phi}{\partial t} + H\left(\nabla\phi\right) = 0, \tag{2.24}$$

where $H = F|\nabla\phi|$ is the Hamiltonian. These equations are similar to hyperbolic conservation laws, and require similar numerical methods. We implement the second-order scheme described by Lin and Tadmor [6], and convert this to a non-staggered scheme using the averaging method of Jiang et al. [7]. The Lin–Tadmor method is based on the Nessyahu–Tadmor scheme for hyperbolic PDEs [8]. The method is a Godunov-type central scheme that does not require the approximate solution of a Riemann problem, and does not require upwind differencing.

The numerical scheme described by Lin and Tadmor [6] is a fully-discrete method that solves for ϕ^{k+1} on a staggered grid. The method is

$$\begin{aligned}
\phi^{k+1}_{i+1/2,j+1/2} = {} & \frac{1}{4}\left(\phi^k_{i,j} + \phi^k_{i+1,j} + \phi^k_{i,j+1} + \phi^k_{i+1,j+1}\right) \\
& + \frac{\Delta x}{16}\left[\left(\phi^k_{i,j}\right)_x - \left(\phi^k_{i+1,j}\right)_x + \left(\phi^k_{i,j+1}\right)_x - \left(\phi^k_{i+1,j+1}\right)_x\right] \\
& + \frac{\Delta y}{16}\left[\left(\phi^k_{i,j}\right)_y - \left(\phi^k_{i,j+1}\right)_y + \left(\phi^k_{i+1,j}\right)_y - \left(\phi^k_{i+1,j+1}\right)_y\right] \\
& - \frac{\Delta t}{2}\left[H\left(\frac{\phi^{k+1/2}_{i+1,j} - \phi^{k+1/2}_{i,j}}{\Delta x}, \frac{\phi^{k+1/2}_{i+1,j+1} - \phi^{k+1/2}_{i+1,j}}{\Delta y}\right)\right. \\
& \left. + H\left(\frac{\phi^{k+1/2}_{i+1,j+1} - \phi^{k+1/2}_{i,j+1}}{\Delta x}, \frac{\phi^{k+1/2}_{i,j+1} - \phi^{k+1/2}_{i,j}}{\Delta y}\right)\right],
\end{aligned} \tag{2.25}$$

where $(x_{i+1/2}, y_{j+1/2}) = (x_i + \Delta x/2, y_j + \Delta y/2)$ for $i = 0, \ldots N_x - 1$ and $j = 0, \ldots N_y - 1$ defines the staggered grid. Computing (2.25) involves evaluations of H at half-time-points $t_{k+1/2} = t_k + \Delta t/2$, and discrete derivatives of ϕ at non-staggered grid points (denoted by the subscripts x and y). The missing mid-values in time are obtained by Taylor expansion

about t_k, yielding

$$\phi_{i,j}^{k+1/2} = \phi_{i,j}^{k} - \frac{\Delta t}{2} H\left[\left(\phi_{i,j}^{k}\right)_x, \left(\phi_{i,j}^{k}\right)_y\right]. \tag{2.26}$$

The derivatives are computed using a suitable flux-limiter to ensure that the scheme remains non-oscillatory [6]. We use the min-mod limiter [6], such that

$$\left(\phi_{i,j}^{k}\right)_x = MM\left(\theta\frac{\phi_{i+1,j}^{k} - \phi_{i,j}^{k}}{\Delta x}, \frac{\phi_{i+1,j}^{k} - \phi_{i-1,j}^{k}}{2\Delta x}, \theta\frac{\phi_{i,j}^{k} - \phi_{i-1,j}^{k}}{\Delta x}\right) \tag{2.27a}$$

$$\left(\phi_{i,j}^{k}\right)_y = MM\left(\theta\frac{\phi_{i,j+1}^{k} - \phi_{i,j}^{k}}{\Delta y}, \frac{\phi_{i,j+1}^{k} - \phi_{i,j-1}^{k}}{2\Delta y}, \theta\frac{\phi_{i,j}^{k} - \phi_{i,j-1}^{k}}{\Delta y}\right), \tag{2.27b}$$

for some $\theta \in [1, 2]$. In our scheme, we use $\theta = 1.99$ [9], indicating a preference for central differencing where possible. The symbol MM denotes the min-mod function, which applied to a vector v_i is

$$MM(v_i) = \begin{cases} \min_i \{v_i\} & \text{if} \quad v_i > 0 \quad \forall i \\ \max_i \{v_i\} & \text{if} \quad v_i < 0 \quad \forall i \\ 0 & \text{otherwise.} \end{cases} \tag{2.28}$$

Once the staggered solution is computed using (2.25), it can be converted to a non-staggered scheme by applying the averaging [7],

$$\begin{aligned} \phi_{i,j}^{k+1} = {} & \frac{1}{4}\left(\phi_{i+1/2,j+1/2}^{k+1} + \phi_{i-1/2,j+1/2}^{k+1} + \phi_{i-1/2,j-1/2}^{k+1} + \phi_{i+1/2,j-1/2}^{k+1}\right) \\ & + \frac{\Delta x}{16}\left[\left(\phi_{i-1/2,j-1/2}^{k+1}\right)_x - \left(\phi_{i+1/2,j-1/2}^{k+1}\right)_x + \left(\phi_{i-1/2,j+1/2}^{k+1}\right)_x - \left(\phi_{i+1/2,j+1/2}^{k+1}\right)_x\right] \\ & + \frac{\Delta y}{16}\left[\left(\phi_{i-1/2,j-1/2}^{k+1}\right)_y - \left(\phi_{i-1/2,j+1/2}^{k+1}\right)_y + \left(\phi_{i+1/2,j-1/2}^{k+1}\right)_y - \left(\phi_{i+1/2,j+1/2}^{k+1}\right)_y\right], \end{aligned} \tag{2.29}$$

where again the appropriate derivatives are computed using the min-mod limiter (2.27). Together with (2.25), (2.29) describes a scheme to solve for $\phi_{i,j}^{k+1}$ given $\phi_{i,j}^{k}$. When solving (2.24), we apply the Dirichlet boundary conditions $\phi_{0,j}^{k+1} = \phi_{0,j}^{k}$, $\phi_{N_x,j}^{k+1} = \phi_{N_x,j}^{k}$, $\phi_{i,0}^{k+1} = \phi_{i,0}^{k}$, and $\phi_{i,N_y}^{k+1} = \phi_{i,N_y}^{k}$, for all k. This is appropriate provided the interface is far from the boundaries of $\mathcal{D}$, which we assume.

2.5 Level-Set Function Reinitialisation

Using orthogonal extrapolation to obtain the extension velocity field does not maintain the signed-distance property of the level-set function, ϕ [2]. Therefore, we develop a method to modify the level-set function to restore the signed-distance property. We achieve this by solving the reinitialisation equation [1], which is the PDE

$$\frac{\partial \phi}{\partial \tau} + S(\phi)\left(|\nabla \phi| - 1\right) = 0, \tag{2.30}$$

where $S(\phi)$ is a modified sign function defined as

$$S(\phi) = \frac{\phi}{\sqrt{|\nabla\phi|^2 + (\Delta x)^2}}. \tag{2.31}$$

Like the level-set equation (1.4b), the reinitialisation equation (2.30) is a Hamilton–Jacobi equation. It has the form $\phi_\tau + H(\phi, \nabla\phi) = 0$, with $H = S(\phi)(|\nabla\phi| - 1)$. Thus, we adapt the Lin–Tadmor method (2.25)–(2.27) and (2.29) used to solve the level-set equation to solve (2.30). Where required in the evaluation of H, we use $\phi^k_{i,j}$ instead of values at the half-time steps.

Although solving the reinitialisation equation at every time step is not required, it must be done intermittently to retain accuracy. In our solutions, we reinitialise the signed-distance function every ten time-steps, and apply ten τ-steps, with $\Delta\tau = 0.5(\Delta x/5 + \Delta y/5)$. To ensure the initial condition $\phi(\boldsymbol{x}, 0)$ is a signed-distance function, we solve (2.30) with 100 τ-steps after first introducing $\phi(\boldsymbol{x}, 0)$.