Journal of Computational Physics 228 (2009) 4379–4399

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

Accurate interface-tracking for arbitrary Lagrangian–Eulerian schemes

Tormod Bjøntegaard, Einar M. Rønquist *

Norwegian University of Science and Technology, Department of Mathematical Sciences, N-7491 Trondheim, Norway

ARTICLE INFO

Article history:
Received 10 March 2008
Received in revised form 25 February 2009
Accepted 9 March 2009
Available online 20 March 2009

Keywords:
Arbitrary Lagrangian–Eulerian
Interface-tracking
Spectral elements

ABSTRACT

We present a new method for tracking an interface immersed in a given velocity field which is particularly relevant to the simulation of unsteady free surface problems using the arbitrary Lagrangian–Eulerian (ALE) framework. The new method has been constructed with two goals in mind: (i) to be able to accurately follow the interface; and (ii) to automatically achieve a good distribution of the grid points along the interface. In order to achieve these goals, information from a pure Lagrangian approach is combined with information from an ALE approach. Our implementation relies on the solution of several pure convection problems along the interface in order to obtain the relevant information. The new method offers flexibility in terms of how an "optimal" point distribution should be defined. We have proposed several model problems, each with a prescribed time-dependent velocity field and starting with a prescribed interface; these problems should be useful in order to validate the accuracy of interface-tracking algorithms, e.g., as part of an ALE solver for free surface flows. We have been able to verify first, second, and third order temporal accuracy for the new method by solving these two-dimensional model problems.

1. Introduction

The ability to accurately follow time-dependent surfaces is very important in many areas of computational science and engineering. An important class of such problems is free surface flows, with the free surface representing the interface between two fluids, e.g., air and water. Computational methods for solving such problems can typically be classified into two categories: methods which explicitly track the free surface (interface-tracking methods; e.g., [18]) and methods where the interface is more implicitly defined (e.g., level set methods [17,19,16] or volume-of-fluid methods [9]); we will here focus on the former class.

Interface-tracking methods (or sometimes also referred to as front-tracking methods) comprise a few essential steps. At any particular point in time, a velocity field is typically determined from the governing equations within the fluid(s), e.g., by solving the Navier–Stokes equations. By integrating this velocity field, it is possible to obtain a new position of the interface.

A pure Lagrangian approach applied to an evolving interface is simply based on integrating the velocities of the fluid particles along the surface to obtain the position of the surface at a later point in time. However, in the context of a numerical approximation (e.g., using finite-element-based methods), a pure Lagrangian approach is often not a very attractive approach since it typically results in large deformations of the computational domain.

In the context of free surface flows, the arbitrary Lagrangian–Eulerian (ALE) formulation of the governing equations has been very successful as a point of departure for a numerical approximation [8,4,12]. A typical approach to updating the free surface is to enforce a kinematic condition along the surface. This condition has its origin in a continuum description, and says that the normal fluid velocity has to be equal to the normal domain velocity at any point along the surface. Hence, a

* Corresponding author. Tel.: +47 73 59 35 47; fax: +47 73 59 35 24.
E-mail address: ronquist@math.ntnu.no (E.M. Rønquist).

0021-9991/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2009.03.012

fluid particle which is present somewhere along the free surface at a particular time will also be present at the free surface at a later time.

While the kinematic condition enforces a normal condition, a tangential domain velocity also needs to be specified along the surface; a common choice is to enforce a homogeneous Dirichlet condition for the tangential component [20,2]. This choice typically reduces the deformation of the computational domain compared to a pure Lagrangian approach, however, it offers limited control over the quality of the grid used to represent the free surface. In particular, the *distribution* of the grid points along the free surface may deteriorate over time, which may ultimately result in severe loss of accuracy (or even breakdown of the simulation). This latter issue may be dealt with in various ways, e.g., through remeshing or other mesh update strategies [13,5]. However, the temporal accuracy will typically suffer using such a strategy.

The issue of a non-optimal evolution of the surface representation is particularly acute in the context of using high order finite elements or spectral elements. The reason for this is related to the fact that such methods depend on locally regular mappings between a reference domain and the corresponding physical element. If the distribution of the physical surface points along the free surface becomes very distorted, this mapping may not be so regular anymore, resulting in a loss of spatial accuracy. This will again affect the calculation of tangent and normal vectors, as well as the local curvature, since the computation of these quantities depends on the coupling between many surface points [11,23].

One could also imagine enforcing the kinematic condition together with a tangential component of the domain velocity in such a way that the integration of the total domain velocity would: (i) result in an accurate representation of the free surface; and (ii) maintain a good distribution of the grid points along the surface [5,3]. An obvious challenge with this approach is how to define the overall domain velocity in such a way that not only good spatial accuracy is achieved (with no need for remeshing), but in a way that will also ensure good temporal accuracy (better than first order). The goal of this paper is to propose a way to achieve these two objectives at the same time.

The paper is organized as follows: We first discuss some key aspects associated with the two-dimensional problems we will focus on, including the notation we will use. We will only discuss the evolution of a surface when it is "immersed" in a known two-dimensional velocity field; no partial differential equation will be solved to obtain this velocity field. We will let the surface evolve in time, and different computational strategies for predicting the surface evolution will be tested and compared. In particular, we will compare two well-known methods with the new approach proposed in this work. Numerical tests will illustrate the similarities and differences between the methods, and conclusions will be made based on these. This study is part of an ongoing research project on solving partial differential equations in time-dependent geometries.

2. Two-dimensional interface-tracking

Consider first the front depicted in Fig. 1. Assume that we know the front at time t^n. Assume also that a numerical approximation of the front is used, something which requires a surface parameterization. A typical way to achieve this is to use piecewise polynomial approximations (e.g., finite-element-based methods), which typically introduces grid points.

Assume now that we have an interface with "optimally" distributed grid points at time level t^n. At each point $\mathbf{x}$ along the surface there is an associated velocity field $\mathbf{u}$. This velocity field can be explicitly known (as in our study here), or it can be given as the solution of an underlying partial differential equation (e.g., the solution of the Navier–Stokes equations in a free surface problem).

We assume that the velocity at a point along the surface represents the velocity of the corresponding "fluid particle". If we integrate the velocity of all the fluid particles along the surface, we obtain the position of the surface at a later time. This is what a pure Lagrangian description will give us; the motion of a particle is simply governed by the equation

$$\frac{d\mathbf{x}}{dt} = \mathbf{u}(\mathbf{x}, t). \tag{1}$$

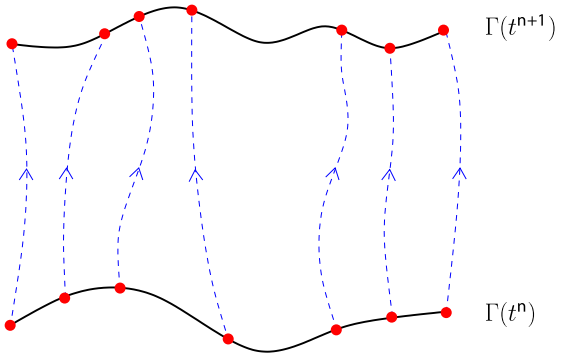


Fig. 1. A front Γ at time t^n with an "optimal" point distribution. For example, the points can be the nodes along an edge of a deformed spectral element. The front is "immersed" in a velocity field and the particles follow the paths of the dashed lines between t^n and t^{n+1}.

In a computational setting, we can limit the integration of (1) to the grid points, and then use the underlying surface parameterization to represent the entire surface at a later time; see Fig. 1. A severe problem with this approach is obvious: we have no control over the distribution of the grid points at a later time t^{n+1}. This will again result in a loss of accuracy in the calculation of surface quantities, e.g., tangent and normal vectors, as well as the local curvature.

A great advantage with the ALE formulation is that it introduces a separate domain velocity $\mathbf{w}$ (also referred to as the grid velocity in the context of the discrete problem), which limits the deformation of the computational domain. In an ALE-framework, the position of the interface is advanced according to

$$\frac{d\mathbf{x}}{dt} = \mathbf{w}(\mathbf{x}, t) \tag{2}$$

instead of the pure Lagrangian approach (1).

A continuum description dictates that $\mathbf{w}\cdot\mathbf{n} = \mathbf{u}\cdot\mathbf{n}$ (the kinematic condition), where $\mathbf{n}$ is the unit normal along the interface. However, no particular condition is required for the tangential component $\mathbf{w}\cdot\mathbf{t}$ of the domain velocity ($\mathbf{t}$ is the unit tangent vector). A common choice is to set $\mathbf{w}\cdot\mathbf{t} = 0$ along the surface, although this is often not an optimal choice; see Figs. 2 and 3.

Let us also comment on the issue of temporal accuracy. First, in the context of simulating unsteady free surface flows, the velocity field is determined from the governing equations, e.g., using an ALE formulation. The associated discrete spatial operators of the Navier–Stokes equations are all time-dependent, and the computation of the velocity, pressure and geometry represents, in principle, a fully coupled system. Due to the complexity of solving such a fully coupled system, a segregated approach is preferred. To this end, a new velocity field is computed based on the most recent geometry configuration(s), while a new geometry is computed by integrating the computed velocity field [12,1]. Integration of (1) and (2) is therefore commonly done using an explicit method; often an explicit multi-step method (e.g., Adams–Bashforth) is preferred [10,2]. If the velocity fields ($\mathbf{u}$ and $\mathbf{w}$) are sufficiently regular, we expect to achieve higher order temporal accuracy (second and third) in terms of the *location* of individual points along the front. As mentioned above, this approach may yield limited control over the *distribution* of the points along the front. If the point distribution is non-optimal, the resulting loss of spatial accuracy *will* affect the accuracy of surface quantities such as normal and tangent vectors, local curvature, and length/area, and this again may affect the accuracy of the interface tracking.

In order to construct a computational approach which will yield both high order temporal accuracy (e.g., second or third order), as well as good spatial accuracy in the calculation of surface quantities, we need to solve the problem of automatically obtaining a good point distribution in a satisfactory way. This problem is particularly acute in the context of using high order methods. Despite the importance of this issue, very limited discussion or results appear to be available in the literature.

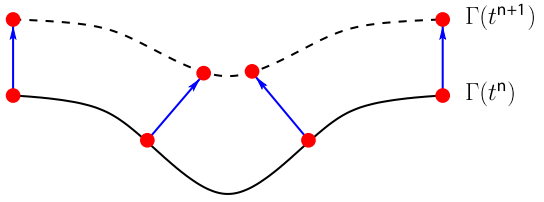


Fig. 2. A front at time t^n with an "optimal" point distribution. This interface is advanced by honoring the kinematic condition, while imposing a zero tangential grid velocity. The resulting point distribution at a later time t^{n+1} is obviously no longer optimal.

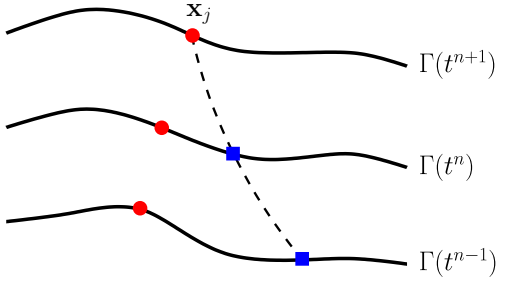


Fig. 3. The plot depicts the position of a single grid point along the front at three different time levels (red circles). The point moves according to (2), with $\mathbf{w}\cdot\mathbf{n} = \mathbf{u}\cdot\mathbf{n}$ and $\mathbf{w}\cdot\mathbf{t} = 0$. The position at time level t^{n+1} is $\mathbf{x}_j^{n+1}$. Note that the corresponding positions at time levels t^n and t^{n-1} do *not* correspond to the same fluid particle; the path of the fluid particle (e.g., a particle which moves according to (1)) ending up at position $\mathbf{x}_j^{n+1}$ at time t^{n+1} follows the dashed line. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

We also mention a complicating factor in the development and assessment of various approaches for interface-tracking: the lack of analytic solutions. In Section 4, we propose a few examples of test problems which we will use for verification purposes.

3. The new approach

We propose a method which gives an accurate representation of the interface (i.e., the individual grid points end up on the correct surface), as well as the flexibility of specifying a user defined point distribution. This method will be "automatic" in the sense of fulfilling both goals in an "integrated" fashion, i.e., with no need for remeshing etc.

Our strategy is based on a combination of a pure Lagrangian approach and an ALE approach, in which both (1) and (2) are integrated in order to find the new grid points. The algorithm we propose has two important ingredients: (i) we need to define what we mean by a good distribution of the grid points; (ii) we need to identify the fluid particles at time t^n which will end up at the valid locations (induced by (i)) at time t^{n+1}. There is flexibility in the choices made with respect to both (i) and (ii). We propose two ways to define a good point distribution (see Section 3.2) and we explain how we efficiently deal with (ii) in this work. We also discuss other possible choices which we do not implement.

Before we present the first version of the algorithm, we describe in some detail the way we represent all surface information, i.e., the surface parameterization; this is done in Section 3.1. In Section 3.2, we propose two alternatives for how to define a good distribution of the grid points, while in Section 3.3 we present the first version of the new method for tracking interfaces (with particular relevance to ALE schemes). The remaining part of Section 3 explains in detail the choices we have made in order to realize our objectives.

3.1. Surface parameterization

In order to make the algorithm concrete, as well as reasonably simple, we limit our discussion to the case where the entire front $\Gamma(t)$ corresponds to an edge in a single spectral element. However, we remark that the ideas behind the proposed method is equally applicable to the case where the front is composed of a number of finite elements (low or high order).

A basic assumption we make is that the interface (or front) Γ at any given point in time is sufficiently smooth to warrant a high order polynomial representation. This assumption is often fulfilled for free surface problems where surface-tension effects play a significant role. This is also the type of problems motivating this study.

Following a standard spectral element discretization [14], the front $\Gamma(t)$ is parameterized as follows: for a given $\xi \in \widehat{\Gamma} = [-1, 1]$, the corresponding point $\mathbf{x}$ on $\Gamma(t)$ is given as $\mathbf{x} = (x_1, x_2) = (x_1(\xi), x_2(\xi))$. In general, any field variable φ associated with the front can be represented in terms of the reference variable ξ. In particular, an Nth order polynomial approximation φ^n of φ at time t^n can be expressed in terms of the following nodal basis:

$$\varphi^n(\xi) = \sum_{j=0}^{N} \varphi_j^n \ell_j(\xi). \tag{3}$$

Here, φ_j^n represents an approximation of $\varphi(\xi_j, t^n)$, ξ_j is the jth Gauss–Lobatto Legendre (GLL) point, and $\ell_j(\xi)$ is the Nth order Lagrangian interpolant through the GLL points; as usual, $\ell_j(\xi_i) = \delta_{ij}$. For example, the coordinates $x_i,\ i = 1, 2$, along the front are represented as

$$(x_i)^n(\xi) = \sum_{j=0}^{N} (x_i)_j^n \ell_j(\xi), \quad i = 1, 2, \tag{4}$$

where $(x_i^n)_j$ is the ith coordinate of the jth point at time t^n. In order for this high order polynomial approximation to be accurate, we rely on a good point distribution, i.e., a good distribution of the nodal coordinates $(x_i)_j^n,\ j = 0, \ldots, N$, $i = 1, 2$, for all times $t^n,\ n = 0, 1, 2, \ldots$. A representation similar to (4) is done for the velocity components $u_i,\ i = 1, 2$, as well as for the domain velocity $w_i,\ i = 1, 2$. We use underscore to denote a vector comprising *all* the nodal values associated with a field variable, e.g., the vector $\underline{x}_1^n$ represents all the values $(x_1)_j^n$, $j = 0, \ldots, N$, in (4). The following surface variables are assumed to be known for a second order temporal scheme:

$$\underline{x}_i^n,\ \underline{x}_i^{n-1}, \quad \underline{u}_i^n,\ \underline{u}_i^{n-1}, \quad \underline{w}_i^{n-1},\ \underline{w}_i^{n-2}, \quad i = 1, 2.$$

3.2. Defining a good point distribution

We consider two alternative ways to define a good point distribution. The first strategy is illustrated in Fig. 4. Starting from the front at time t^n, we move the end points to time level t^{n+1}. How these points are moved will be problem dependent. For example, if we consider a closed system where no particles enter or leave our computational domain, the movement will correspond to a Lagrangian motion where the two end points follow the path of the fluid particles from t^n to t^{n+1} (see (1)). Alternatively, the end points may move according to a standard ALE-formulation (see (2)).

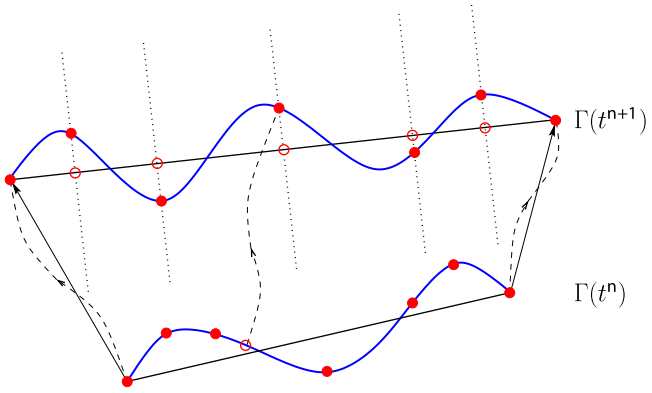


Fig. 4. Strategy 1 for advancing the interface Γ from time level t^n to the new time level t^{n+1}. First, the two end points are advanced. Next, the points along the chord connecting the new end points of the interface are distributed according to a GLL distribution (the open circles). We now define normals to the chord going through these points (the dotted lines). The intersection between these normals and the interface (the closed circles) define the coordinates of the nodal points along $\Gamma(t^{n+1})$. Except for the end points of the interface, each nodal point along $\Gamma(t^{n+1})$ corresponds to a fluid particle which at time t^n had a location somewhere along $\Gamma(t^n)$, and typically not corresponding to a nodal point at time t^n (open circle on $\Gamma(t^n)$).

When the position of the end points are updated, we construct the chord between the two end points, and distribute grid points along this chord according to the desired distribution (in our case, a GLL distribution). Next, the normal from the chord is constructed at each of these points. We then search for the particle at time t^n which, when advanced through a pure Lagrangian motion, is located somewhere along this normal. The advantage of this approach is that we are in full control of the distribution of all the grid points at each time step. The disadvantage is that it may not be so easy to extend the approach to three dimensions.

The second strategy is illustrated in Fig. 5. Again, the end points are first advanced to t^{n+1}. Next, for each end point, a vector which connects the end point at t^n to the corresponding end point at t^{n+1} is constructed. Then, directional vectors for the interior nodes are constructed through a linear interpolation of the end point vectors. Finally, the requirement is to find the particle at time t^n which, when advanced to t^{n+1}, is located along this interpolated vector starting at the grid point at time t^n. This strategy has the advantage that it is more easily extended to three dimensions. However, we have less explicit control over the point distribution at each time level compared to Strategy 1.

3.3. First version of a second order temporal scheme

We now discuss the key ingredients in a second order temporal scheme; we will later discuss all the details, as well as the extension to a third order scheme. In Algorithm 1, we present the first version of the new algorithm. Note that this version is meant to represent the general idea and that all the steps are not necessarily implemented in this form (although this is also an alternative).

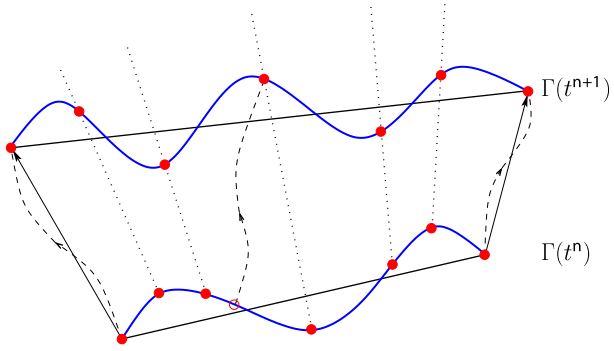


Fig. 5. Strategy 2 for advancing the interface Γ from time level t^n to the new time level t^{n+1}. First, two vectors connecting the end points at t^n to the corresponding end points at t^{n+1} are constructed (solid lines with arrow heads). Next, directional vectors for the remaining nodes along the interface are constructed based on a linear interpolation of the end vectors at the GLL points on the reference domain, $\widehat{\Gamma}$. The dotted lines are these directional vectors started at the nodal points at time t^n. The intersection between these directional vectors and the interface (the closed circles) define the coordinates of the nodal points along $\Gamma(t^{n+1})$. Except for the end points of the interface, each nodal point along $\Gamma(t^{n+1})$ corresponds to a fluid particle which at time t^n had a location somewhere along $\Gamma(t^n)$, and typically not corresponding to a nodal point at time t^n (open circle).

Algorithm 1. First version of a second order temporal scheme

for $j = 0$ to N **do**
1. Guess ξ^n.
repeat
2. Find $\xi^{n-1}(\xi^n)$.
3. Compute
$$\begin{aligned} \hat{x}_i^1 &= x_i^n(\xi^n) \qquad i = 1,2, \\ \hat{u}_i^1 &= u_i^n(\xi^n) \qquad i = 1,2, \\ \hat{u}_i^2 &= u_i^{n-1}(\xi^{n-1}) \quad i = 1,2. \end{aligned}$$
4. Compute $(x_i^{n+1})_j = \hat{x}_i^1 + \Delta t(\frac{3}{2}\hat{u}_i^1 - \frac{1}{2}\hat{u}_i^2), \quad i = 1,2.$
5. Update ξ^n.
until convergence
6. Compute $(w_i^n)_j$ from $(x_i^{n+1})_j = (x_i^n)_j + \Delta t(\frac{3}{2}w_i^n - \frac{1}{2}w_i^{n-1})_j, \ i = 1,2.$
end for

The position $(x_i)_j^{n+1}$ computed in Step 4 represents the integration of (1) for a single fluid particle using a second order Adams–Bashforth scheme. The position of this fluid particle at time t^n is given by $x_i^n(\xi^n)$ for some reference coordinate ξ^n. The velocity of this fluid particle at time t^n is given by $u_i^n(\xi^n)$. The same fluid particle is at a position $x_i^{n-1}(\xi^{n-1})$ for some reference coordinate ξ^{n-1}. However, initially we don't know if the computed grid point represents a valid point on $\Gamma(t^{n+1})$, e.g., consistent with Strategy 1 or Strategy 2 considered in this work. An iterative process in order to determine which fluid particle ends up in a valid position is therefore necessary. We obtain convergence when we have chosen the "correct" particle, which amounts to choosing the "correct" ξ^n at time t^n and a corresponding ξ^{n-1} at time t^{n-1}.

For a given ξ^n, the computation of ξ^{n-1} may be found explicitly through a process involving integration backwards in time (i.e., by computing the "foot of the characteristic"); we will discuss this alternative in Section 3.8. However, from Step 3 and 4 we see that what we actually need is the *velocity* the particle in question had at time t^{n-1} (denoted as $\hat{u}_i^2$), and not necessarily the position it had at time t^{n-1}; we will later explain how we may exploit this fact.

When we have converged to the correct position $(x_i^{n+1})_j$ of grid point j on $\Gamma(t^{n+1})$, see Step 4, we also compute the corresponding grid velocity $(w_i^n)_j$ such that an integration of (2) using a second order Adams–Bashforth method will result in the *same* position; this is done in Step 6. Here, $(x_i^n)_j, (x_i^{n+1})_j$, and $(w_i^{n-1})_j$ are known, while $(w_i^n)_j$ is the quantity we compute. The reasons for computing the grid velocity in this way are: (i) it gives consistency with a pure Lagrangian approach; (ii) we indirectly satisfy the kinematic condition; and (iii) we indirectly and "automatically" specify a tangential grid velocity such that we obtain a good point distribution.

We mentioned earlier that it is quite common to combine the kinematic condition with a zero tangential domain velocity, which implies that the grid points along the interface only moves in the normal direction; see Figs. 2 and 3. In this case, a second order discretization of (2) can also be expressed as indicated in Step 6 of Algorithm 1, but with $(x_i^n)_j, (w_i^n)$, and $(w_i^{n-1})_j$ as the known quantities, and $(x_i^{n+1})_j$ as the unknown. However, as also discussed earlier, this approach does not ensure a good point distribution.

3.4. Finding a fluid particle's earlier velocities

The second order scheme presented in Algorithm 1 requires information about the velocities at time levels t^n and t^{n-1} for those fluid particles along the interface which end up at the grid points $\mathbf{x}_j^{n+1}, j = 0,\ldots,N$, along $\Gamma(t^{n+1})$. One way to obtain the necessary information is by tracking the characteristics backwards in time, and we will return to a discussion of this alternative later. We will here focus on a different approach which is inspired by the work in [15]. In either case, we need to deal with the fact that the interface Γ changes shape as a function of time.

Since we assume that the same particles remain on the surface at all time, it is sufficient to solve a one-dimensional convection problem *forward* in time in order to obtain the required information; we now explain this procedure. Consider the following pure time-dependent convection problem along the interface $\Gamma(t^* + \tau)$, where t^* is either t^{n-1} or t^{n-2}: Find $\varphi(s,\tau)$ such that

$$\frac{\partial \varphi}{\partial \tau} + u_s \frac{\partial \varphi}{\partial s} = 0, \quad \text{on } \Gamma(t^* + \tau),$$

$$\varphi(s, \tau = 0) = \varphi_0(s), \quad \text{on } \Gamma(t^*).$$

Here, s is an arc-length variable, $u_s = \mathbf{u} \cdot \mathbf{t}$ is the tangential component of the fluid velocity, and appropriate boundary conditions are assumed on $\partial\Gamma(t^* + \tau)$. The ALE formulation of this problem reads: find $\varphi(s;\tau) \in X$ such that

$$\begin{aligned} &\frac{\mathrm{d}}{\mathrm{d}\tau}(\varphi, v) + c(\varphi, v) = 0, \quad \forall v \in X, \\ &\varphi(\xi; \tau = 0) = \varphi_0(\xi), \end{aligned} \tag{5}$$

where

$$(\varphi, v) = \int_{\widehat{\Gamma}} v\varphi J_s \,\mathrm{d}\xi, \tag{6}$$

$$c(\varphi, v) = \int_{\widehat{\Gamma}} v(u_s - w_s)\frac{\partial \varphi}{\partial \xi}\,\mathrm{d}\xi - \int_{\widehat{\Gamma}} v\varphi \frac{\partial w_s}{\partial \xi}\,\mathrm{d}\xi. \tag{7}$$

Here, X is an appropriate function space, e.g., $X = H^{1/2}(\Gamma)$ if all the field variables correspond to the trace of H^1-functions in the adjacent fluid domains. Furthermore, $J_s = \left(\left(\frac{\partial x_1}{\partial \xi}\right)^2 + \left(\frac{\partial x_2}{\partial \xi}\right)^2\right)^{1/2}$ is the surface Jacobian, and $w_s = \mathbf{w}\cdot\mathbf{t}$ is the tangential component of the domain velocity. Note that $u_s - w_s$ is zero on $\partial\Gamma$ for all the model problems in our study. We also remark that we have used the same symbol φ for a field variable expressed both in the arc-length variable and in the reference coordinate.

We discretize the convection problem (5) using a standard spectral element method in space [14] (with a single element), and arrive at the following set of ordinary differential equations:

$$\begin{aligned} &\frac{\mathrm{d}(\underline{B}^s\,\underline{\varphi})}{\mathrm{d}\tau} = -\underline{C}^s\underline{\varphi}, \\ &\underline{\varphi}(\tau = 0) = \underline{\varphi}_0. \end{aligned} \tag{8}$$

Here, $\underline{B}^s$ is the (diagonal) surface mass matrix with matrix elements

$$B^s_{ij} = (\ell_j, \ell_i)_{\mathrm{GLL}},$$

$\underline{C}^s$ is the discrete convection operator along the surface (including the "surface divergence" of the domain velocity) with matrix elements

$$C^s_{ij} = c(\ell_j, \ell_i)_{\mathrm{GLL}}.$$

Here, subscript GLL refers to evaluation of the bilinear forms (6) and (7) by Gauss–Lobatto Legendre quadrature. Note that both $\underline{B}^s$ and $\underline{C}^s$ are time-dependent.

If we integrate (8) from 0 to Δt with $\underline{\varphi}_0 = \underline{u}_i^{n-1}, i = 1, 2$, we observe that $\underline{\varphi}(\tau = \Delta t)$ will be an approximation to the ith velocity component at time t^{n-1} of the fluid particles which at time t^n are located at the grid points along $\Gamma(t^n)$. This approach is inspired by the ideas presented in [15] in the context of constructing convection-Stokes splitting schemes in fixed geometries, and extended to time-dependent domains in [1].

Note that $\hat{u}_i^2$ in Algorithm 1 generally represents the velocity of a fluid particle at time t^{n-1} which does *not* coincide with a grid point along $\Gamma(t^{n-1})$. However, by computing the velocities at time t^{n-1} of the fluid particles which coincide with the grid points of $\Gamma(t^n)$ (i.e., by solving (8)), we can use the polynomial expansion (3) to find the velocity at *any* value of the parameter ξ. Moreover, a *single* value of ξ will now give us information about the velocity of a fluid particle at two different time levels (t^n and t^{n-1}). This is also exactly the type of information we need in our algorithm; in fact, access to this information avoids entirely the need to find $\xi^{n-1}(\xi^n)$ in Step 2 of Algorithm 1 and thus represents a simplification.

The approach is readily extended to velocities at earlier time levels as well. For instance, if we choose $\underline{\varphi}_0 = \underline{u}_i^{n-2}$ in (8), $\underline{\varphi}(\tau = 2\Delta t)$ will be an approximation to the ith velocity component at time t^{n-2} of the fluid particles which at time t^n are located at the grid points along $\Gamma(t^n)$.

For the integration of (8) we have chosen the classical explicit fourth order Runge–Kutta scheme. In practice, the time step Δt will be given by the associated Navier–Stokes solver. Similar to the convection subproblem treated in [15], the tangential fluid velocity $u_s(\xi, t)$, the tangential domain velocity $w_s(\xi, t)$, as well as the surface coordinates $x_i(\xi, t),\ i = 1, 2$, are each approximated as a polynomial in time of one order lower than the Adams–Bashforth scheme used in Step 4 and Step 6 of Algorithm 1. Thus, for a second order temporal scheme, a first order polynomial interpolation (extrapolation for w_s) in time is used for these quantities when solving (8), e.g.,

$$\underline{x}_i(t) = \underline{x}_i^{n-1} + \frac{(\underline{x}_i^n - \underline{x}_i^{n-1})}{\Delta t}\,(t - t^{n-1}), \quad t^{n-1} \leqslant t \leqslant t^n, \qquad i = 1, 2.$$

The extension to a third order scheme will use a second order polynomial approximation in time for u_s, w_s and $x_i,\ i = 1, 2$, in the interval $t^{n-2} \leqslant t \leqslant t^n$.

To summarize this section: in Algorithm 1 we are interested in the velocities that particular fluid particles had at time levels t^n and t^{n-1}. We choose $\underline{\varphi}_0 = \underline{u}_i^{n-1}$ in (8) and set $\underline{\tilde{u}}_i^n = \underline{\varphi}(\tau = \Delta t), i = 1, 2$. Note that the total cost of solving (8) is only $\mathcal{O}(N^2)$. We now have six sets of nodal values, $\underline{x}_i^n, \underline{u}_i^n$, and $\underline{\tilde{u}}_i^n$, $i = 1, 2$, all associated with the interface $\Gamma(t^n)$. By using the polynomial expansion (3) we have thus six polynomial approximations, and all of these are defined along $\Gamma(t^n)$. Hence, one particular value of ξ corresponds to the *same* fluid particle. Thus, there is no longer a need to compute $\xi^{n-1}(\xi^n)$ in Step 2 of

Algorithm 1. We may now only focus on finding the appropriate ξ^n which will allow us to achieve convergence in Step 4. The modified version of the algorithm is summarized in Algorithm 2.

Algorithm 2. Second version of a second order temporal scheme

1. Solve (8) with $\underline{\varphi}_0 = \underline{u}_i^{n-1}$ and set $\underline{\tilde{u}}_i^n = \varphi(\tau = \Delta t)$ for $i = 1, 2$.
for $j = 0$ to N **do**
  2. Guess ξ^n.
  repeat
    3. Compute

$$\hat{x}_i^1 = x_i^n(\xi^n) \quad i = 1,2,$$
$$\hat{u}_i^1 = u_i^n(\xi^n) \quad i = 1,2,$$
$$\hat{u}_i^2 = \tilde{u}_i^n(\xi^n) \quad i = 1,2.$$

    4. Compute $(x_i^{n+1})_j = \hat{x}_i^1 + \Delta t(\frac{3}{2}\hat{u}_i^1 - \frac{1}{2}\hat{u}_i^2), \quad i = 1,2.$
    5. Update ξ^n.
  until convergence
  6. Compute $(w_i^n)_j$ from $(x_i^{n+1})_j = (x_i^n)_j + \Delta t(\frac{3}{2}w_i^n - \frac{1}{2}w_i^{n-1})_j,\ i = 1,2.$
end for

3.5. Finding the "correct" particle

We are searching for the position of a fluid particle at time level t^n, as well as its velocities at time levels t^n and t^{n-1}, such that the computed position of the fluid particle at time level t^{n+1} according to Step 4 in Algorithm 1 or Algorithm 2 is a "valid" position according to our desired point distribution (in our case, using either Strategy 1 or Strategy 2). As explained in the previous section, the problem can be reduced to finding one particular value $\xi^n \in \widehat{\Gamma}$; see Algorithm 2. Let us discuss two alternative ways to do this. A comparison of the cost of the two alternatives will be made at the end of this section.

3.5.1. Direct search

One alternative is to do a direct search via a Newton iteration. Let us briefly explain this procedure in the very simple case when the two end points are moving only in the x_2-direction and Strategy 2 is employed. In this case, all the interpolated direction vectors point in the x_2-direction and the x_1-coordinate of each grid point should remain fixed as the interface evolves. Note that, even though the x_1-component of the fluid velocity may be non-zero, we still want the grid points to move only in the x_2-direction through a discretization of (2); this is one of the features exploited when using the ALE-formulation. Our objective with the front tracking algorithm is to automatically honor the kinematic condition *and* achieve a good point distribution at the same time.

Combining Steps 3 and 4 of Algorithm 2 we can define

$$f_j(\xi) = (x_1)_j^{n+1} - \left((x_1)^n(\xi) + \Delta t\left(\frac{3}{2}(u_1)^n(\xi) - \frac{1}{2}(\tilde{u}_1)^n(\xi)\right)\right), \tag{9}$$

where $(x_1)_j^{n+1} = (x_1)_j^0$, i.e., equal to the initial x_1-coordinate of grid point j. We can use a Newton iteration to find ξ^n such that $f_j(\xi^n) = 0$. To this end, we also need information about $f_j'(\xi)$. However, such derivative information is available by differentiating $(x_1)^n(\xi)$, $(u_1)^n(\xi)$ and $(\tilde{u}_1)^n(\xi)$ using the expansion (3) for each of these variables. Once we have determined ξ^n (which in general does not correspond to a GLL point), we find the new (physical) coordinates for grid point j from

$$(x_i)_j^{n+1} = (x_i)^n(\xi^n) + \Delta t\left(\frac{3}{2}(u_i)^n(\xi^n) - \frac{1}{2}(u_i)^{n-1}(\xi^n)\right), \quad i = 1,2. \tag{10}$$

Note that $(x_i)^n(\xi^n),\ i = 1,2$, is the position of the fluid particle at time t^n which ends up in grid point j at time t^{n+1}. For a more general situation, e.g., see Fig. 4, the one-dimensional function $f_j(\xi)$ will be slightly more complicated, but the approach will involve a Newton iteration as in this simple case.

3.5.2. Solving an artificial convection problem

Instead of pursuing a direct search to find ξ^n, we here wish to explore the possibility of using a completely different way of finding the information we need. As mentioned earlier, all the pertinent surface information is parameterized using the reference variable ξ.

Assume that we are currently interested in obtaining information about the fluid particle which ends up in grid point j at time t^{n+1}. Instead of doing a direct search for this fluid particle as explained above, we propose: (i) to define an *artificial* convecting velocity U on $\widehat{\Gamma}$; and (ii) to find a corresponding artificial time τ^n which will convect all the pertinent surface

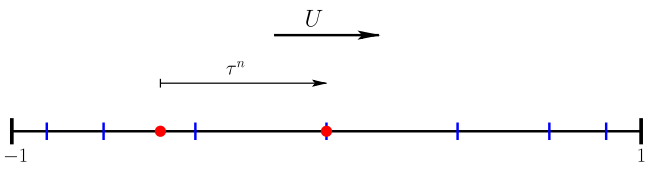


Fig. 6. Six sets of nodal values, $\underline{x}_i^n, \underline{u}_i^n$, and $\underline{\tilde{u}}_i^n, i = 1, 2$, are associated with the reference domain $\widehat{\Gamma}$. By integrating (12) from $\tau = 0$ to $\tau = \tau^n$, we can artificially convect the information associated with a particular $\xi^n \in \widehat{\Gamma}$ (i.e., associated with a particular fluid particle) and check if Step 4 in Algorithm 2 will give us a "valid" new position for a grid point along $\Gamma(t^{n+1})$.

information about this particle to ξ_j (the jth GLL point); see Fig. 6. Specifically, we consider the one-dimensional convection problem

$$\begin{aligned} &\frac{\partial \varphi}{\partial \tau} + U\frac{\partial \varphi}{\partial \xi} = 0, \quad \text{on } \widehat{\Gamma}, \\ &\varphi(\tau = 0) = \varphi_0. \end{aligned} \tag{11}$$

We assume that $U = 0$ on $\partial\widehat{\Gamma}$ (a similar condition was assumed for u_s in the previous section), and we will thus not specify any particular boundary condition for φ. We discretize (11) using a spectral element method [14] (with a single element) and arrive at the following set of ordinary differential equations

$$\begin{aligned} &\widehat{\underline{B}}\frac{\mathrm{d}\underline{\varphi}}{\mathrm{d}\tau} + \widehat{\underline{C}}\,\underline{\varphi} = \underline{0}, \\ &\underline{\varphi}(\tau = 0) = \underline{\varphi}_0. \end{aligned} \tag{12}$$

In (12), $\widehat{\underline{B}}$ and $\widehat{\underline{C}}$ are the mass matrix and discrete convection operator, respectively; the hat indicates that both matrices are associated with $\widehat{\Gamma}$. The idea is that, instead of searching for a suitable ξ^n as we did in the first alternative, we search for a τ^n such that we obtain the information we need when we integrate (12) from $\tau = 0$ to $\tau = \tau^n$. The vector $\underline{\varphi}_0$ in (12) can be one of six sets of nodal values, $\underline{x}_i^n, \underline{u}_i^n$, and $\underline{\tilde{u}}_i^n, i = 1, 2$, i.e., we solve (12) six times. In summary, instead of searching for a particle as in the first alternative, we "sit" at a fixed point (ξ_j) and (artificially) convect the pertinent information about the particle to this point.

Let us denote the solution vector of (12) at time τ as $\underline{\varphi}(\tau; \underline{\varphi}_0)$ where we have emphasized that the initial condition for this problem is equal to $\underline{\varphi}_0$; in particular, let $(\underline{\varphi}(\tau; \underline{\varphi}_0))_j$ denote the jth element of this solution vector. With this notation, (9) can be modified to read

$$\tilde{f}_j(\tau) = (x_1)_j^{n+1} - \left((\underline{\varphi}(\tau; \underline{x}_1^n))_j + \Delta t\left(\frac{3}{2}(\underline{\varphi}(\tau; \underline{u}_1^n))_j - \frac{1}{2}(\underline{\varphi}(\tau; \underline{\tilde{u}}_1^n))_j\right)\right). \tag{13}$$

This time we use a Newton iteration to find τ^n such that $\tilde{f}_j(\tau^n) = 0$. To this end, we also need information about $\tilde{f}_j'(\tau)$, which necessitates differentiating $\underline{\varphi}$ with respect to τ. However, using (12), the necessary information can be computed precisely from

$$\left(\frac{\mathrm{d}\underline{\varphi}}{\mathrm{d}\tau}(\tau; \underline{\varphi}_0)\right)_j = \left(\widehat{\underline{B}}^{-1}\widehat{\underline{C}}\;\underline{\varphi}(\tau; \underline{\varphi}_0)\right)_j, \tag{14}$$

where $\underline{\varphi}_0$ is either $\underline{x}_1^n, \underline{u}_1^n$, or $\underline{\tilde{u}}_1^n$. For the test problems considered in this study, (12) is solved using an explicit fourth order Runge–Kutta scheme. Thus, the derivative information we need is automatically available from the vector constructed in the last stage of the Runge–Kutta scheme (to be precise, the jth element of this vector). For the test problems explored in this study, convergence in the Newton iteration is achieved in approximately 2 or 3 iterations.

At convergence, the particular value of τ^n is actually of no interest to us; all we need is $\underline{\varphi}(\tau^n)$ and then exploit the fact that $(\underline{\varphi}(\tau^n; \underline{\varphi}_0))_j = (\varphi_0)(\xi^n)$. Through this equivalence, we can directly update the new grid position using (10).

Note that there is great flexibility in the choice of the convective velocity, U. This is because, for a specific particle somewhere along the surface, and for a reasonable choice of U, there will always be a corresponding τ^n (perhaps negative) which convects this particle to the current grid point (GLL point ξ_j). An easy and convenient choice is $U(\xi) = 1 - \xi^2$ since this velocity is very smooth, it does not change sign, and it is also compatible with the condition $u_s - w_s = 0$ at $\partial\Gamma$ (which is the case for all our numerical examples). Note that the artificial convecting velocity U in (12) is *time-independent*; the main purpose of U is just to provide a mechanism by which pertinent information about a point is transported (as an alternative to a plain search).

3.5.3. A comparison of the two alternatives

We now compare the two alternatives, in particular, the computational cost associated with finding the correct particles. Note that this discussion does not include the cost of solving the problem (8) which scales like $\mathcal{O}(N^2)$.

The alternative based on a direct search requires the evaluation of the six quantities x_i^n, u_i^n, and $\tilde{u}_i^n, i = 1, 2$, at an arbitrary value of the reference coordinate ξ. Using an expansion (3) for each quantity, each evaluation ostensibly requires approximately $2N$ floating point operations since there are $N + 1$ terms to sum up. However, the computation of $\ell_k(\xi)$ for a general value of ξ requires at least $2N$ operations for each $k = 0, \ldots, N$. Hence the total cost is approximately $2N^2$ for each evaluation of $f_j(\xi)$.

The basis (3) is also used in order to compute derivative information at an arbitrary value of the reference coordinate ξ; this is required in the Newton iteration. To this end, we first need to apply the differentiation matrix associated with the GLL points at a cost of $2N^2$ operations (a single matrix–vector product), and then evaluate the derivative $f_j'(\xi)$ at an arbitrary value of the reference coordinate ξ at a cost of about $6N$ operations (assuming that we store the earlier computed values of $\ell_k(\xi), k = 0, \ldots, N$).

A direct search is used for all the internal GLL points. Hence, the total cost for the first alternative is about $2N^3$ operations per Newton iteration.

Let us now discuss the second alternative; see Section 3.5.2. For each (internal) point we also need to do a Newton iteration. At each Newton iteration we need to solve six convection problems of the type (12). We solve each convection problem using an explicit fourth order Runge–Kutta scheme and using a single time step. Hence, the cost of these six convection problems is about 48 N^2 operations (6 problems, 4 stages per problem, and one matrix–vector product evaluation per stage). We remark that no additional computation is needed to obtain derivative information in the Newton iteration since this information is automatically available from the last stage in the Runge–Kutta scheme. We also remark that, although our initial experience has been very encouraging, a careful stability analysis of this alternative is still lacking.

Again, we need to apply this alternative approach for all the internal GLL points. Hence, the total cost for the second alternative is about $48N^3$ operations per Newton iteration.

In the numerical tests we consider later, we have only implemented the second alternative. The direct search approach certainly has a smaller constant in front of the $\mathcal{O}(N^3)$ scaling. However, there are also several reasons for choosing the second alternative: (i) the extension to the multi-element case avoids any need to know in which element the fluid particle we are searching for is located; (ii) easier and more convenient implementation on parallel computers (due to point (i)); (iii) the scaling of the cost is similar; (iv) we would like to verify that an approach based on an artificial convection problem actually works; (v) the cost of the entire front-tracking algorithm presented here is generally subdominant the cost of the solution of the associated Navier–Stokes problem; see the discussion in Section 3.8 of the overall approach.

3.6. Final version of a second order temporal scheme

The discussion from the preceding sections leads us to our final version of an algorithm for a second order temporal scheme; see Algorithm 3 below. We remark that the loop $j = 0, \ldots, N$ over the grid points includes the two end points of the interface. These two end points may be treated differently compared to the inner points; this depends on the particular information which is available for the end points. For instance, for our numerical test problems, we consider a "closed system" where no particles enter or leave the domain (i.e., the interface), and the end points are moved in a pure Lagrangian fashion according to (1).

Algorithm 3. Final version of a second order temporal scheme

1. Solve (8) with $\underline{\varphi}_0 = \underline{u}_i^{n-1}$ and set $\underline{\tilde{u}}_i^n = \varphi(\tau = \Delta t)$ for $i = 1, 2$.
for $j = 0$ to N **do**
  2. Guess τ^n.
  repeat
    3. Integrate (12) from 0 to τ^n six times with $\underline{\varphi}_0 = \underline{x}_i^n$, $\underline{\varphi}_0 = \underline{u}_i^n$, $\underline{\varphi}_0 = \underline{\tilde{u}}_i^n$, for $i = 1, 2$. Define the results as $\underline{\hat{x}}_i$, $\underline{\hat{u}}_i^1$, $\underline{\hat{u}}_i^2$, respectively.
    4. Compute $(x_i^{n+1})_j = (\hat{x}_i)_j + \Delta t\left(\frac{3}{2}\hat{u}_i^1 - \frac{1}{2}\hat{u}_i^2\right)_j$ for $i = 1, 2$.
    5. Update τ^n.
  until convergence
  6. Compute $(w_i^n)_j$ from $(x_i^{n+1})_j = (x_i^n)_j + \Delta t\left(\frac{3}{2}w_i^n - \frac{1}{2}w_i^{n-1}\right)_j$ for $i = 1, 2$.
end for

3.7. Extension to a third order temporal scheme

Algorithm 4 represents a third order version of this method. We now have to first solve two problems of the type (8) for each velocity component. During the integration of these problems, we need to use a second order polynomial approximation in time of the grid, the fluid velocity, and the grid velocity in order to maintain a third order temporal convergence rate. Obviously, the overall integration scheme for integrating (8) must also be at least of third order in time. As before, we use an explicit fourth order Runge–Kutta scheme, so this will not cause any problems. The solution of (12) will be the same as for

the second order scheme except for the fact that we now have to solve 8 convection problems instead of 6. The cost of the third order scheme will therefore be about 8/6 of the cost of a second order scheme since the cost associated with the loop over the grid points will dominate. In Step 4 and Step 6, we use a third order Adams–Bashforth scheme to compute the new position and the new grid velocity of each grid point, respectively.

Algorithm 4. Final version of a third order temporal scheme

1a. Solve (8) with $\underline{\varphi}_0 = \underline{u}_i^{n-1}$ and set $\tilde{\underline{u}}_i^n = \varphi(\tau = \Delta t), \quad i = 1,2.$
1b. Solve (8) with $\underline{\varphi}_0 = \underline{u}_i^{n-2}$ and set $\tilde{\tilde{\underline{u}}}_i^n = \varphi(\tau = 2\Delta t), \quad i = 1,2.$
for $j = 0$ to N
  2. Guess τ^n.
  repeat
    3. Integrate (12) from 0 to τ^n eight times with $\underline{\varphi}_0 = \underline{x}_i^n, \underline{\varphi}_0 = \underline{u}_i^n, \underline{\varphi}_0 = \tilde{\underline{u}}_i^n, \underline{\varphi}_0 = \tilde{\tilde{\underline{u}}}_i^n, i = 1,2$. Define the results as $\hat{\underline{x}}_i, \hat{\underline{u}}_i^1, \hat{\underline{u}}_i^2, \hat{\underline{u}}_i^3$, respectively.
    4. Compute $(x_i^{n+1})_j = (\hat{x}_i)_j + \Delta t\left(\frac{23}{12}\hat{u}_i^1 - \frac{4}{3}\hat{u}_i^2 + \frac{5}{12}\hat{u}_i^3\right)_j, \ i = 1,2.$
    5. Update τ^n.
  until convergence
  6. Compute $(w_i^n)_j$ from $(x_i^{n+1})_j = (x_i^n)_j + \Delta t\left(\frac{23}{12}w_i^n - \frac{4}{3}w_i^{n-1} + \frac{5}{12}w_i^{n-2}\right)_j, i = 1,2.$
end for

3.8. Discussion of the overall approach

A key ingredient of the proposed algorithm is to represent the values of the fluid velocity at earlier times (t^{n-1} and t^{n-2}) in terms of the polynomial basis associated with $\Gamma(t^n)$. Having chosen what we mean by a good point distribution along the new interface $\Gamma(t^{n+1})$, it is then sufficient to search for the "correct" particles along $\Gamma(t^n)$ which end up at valid locations along $\Gamma(t^{n+1})$. In Section 3.5.3, we discussed two alternative ways to do this, and we explained our choice among these two. The cost of the entire interface-tracking algorithm is $\mathcal{O}(N^3)$, dominated by the cost of finding the correct particles. Note that this cost is independent of whether we choose Strategy 1 or Strategy 2.

We should mention that a completely different overall approach is also possible. Following the ideas behind semi-Lagrangian schemes [6,22,21,7], we could have embarked directly on Algorithm 1. In this case, we would have to integrate the characteristics backward in time. For example, starting at a reference coordinate value ξ^n, i.e., a fluid particle located at position $x_i^n(\xi^n), \ i = 1,2$, on $\Gamma(t^n)$, we could then obtain access to the location of this particle at an earlier time t^{n-1}. From this location ($x_i^{n-1}, \ i = 1,2$) we could determine the reference coordinate ξ^{n-1} associated with the mapping between $\widehat{\Gamma}$ and $\Gamma(t^{n-1})$, which again would allow us to find the value of the fluid particle's earlier velocity, $u_i^{n-1}(\xi^{n-1}), \ i = 1,2$.

There are a couple of issues worth mentioning here. First, depending on how $x_i^{n-1}, \ i = 1,2$, is obtained from $x_i^n(\xi^n), \ i = 1,2$, the location $x_i^{n-1}, \ i = 1,2$, could possibly correspond to a point which is not exactly on the interface $\Gamma(t^{n-1})$; see [21]. In this case, we would need to find the point along $\Gamma(t^{n-1})$ which is closest to $x_i^{n-1}, \ i = 1,2$, before finding the corresponding reference coordinate ξ^{n-1} through an iterative process.

Second, the search for the correct ξ^n could be done similarly to the direct search approach discussed in Section 3.5.1. However, in the simple example discussed there, the function $f_j(\xi^n)$ in (9) would have to be modified to read

$$f_j(\xi^n) = (x_1)_j^{n+1} - \left((x_1)^n(\xi^n) + \Delta t\left(\frac{3}{2}(u_1)^n(\xi^n) - \frac{1}{2}(u_1)^{n-1}(\xi^{n-1}(\xi^n))\right)\right). \tag{15}$$

Similar to the approach we have chosen in this work, the overall cost for a semi-Lagrangian approach is also $\mathcal{O}(N^3)$ operations per Newton iteration.

There are several reasons why we have focused on the approach summarized in Algorithms 3 and 4. First and foremost, our goal has been to develop a method for tracking an interface which is *verifiably accurate* and which automatically ensures a *good point distribution* throughout the simulation; such a method seems to be lacking in the literature. Our motivation derives from being able to track an interface in the context of an ALE scheme, e.g., solving a free surface problem where surface-tension effects are important. In this context, the fluid velocity (which we assume is given here) is computed by solving the Navier–Stokes equations. In the two-dimensional case, the computational cost of computing $\underline{u}_i, \ i = 1,2$, will at best scale like $\mathcal{O}(N^3)$ with a large factor in front; this factor will be a product of the number of spectral elements in the two-dimensional domain, the number of iterations needed to solve the associated linear system of equations, a factor associated with operator evaluation per element and per type of discrete operator. Compared to this cost, the cost of the proposed algorithm will be subdominant. Our choices have therefore emphasized other aspects of the various alternatives, such as avoiding the need for a search and interpolation algorithm in the multiple element case, and avoiding any issue related to "hitting" the front in the context of integrating the characteristics backwards in time. Again, the main focus has been on demonstrating that the overall objectives can be achieved; the particular choices considered here for the various components of the overall approach can, of course, be substituted with other alternatives.

4. Numerical results

In this section, we will perform numerical experiments in order to validate and compare the different algorithms for tracking the interface. For all the test problems we will define a two-dimensional, time-dependent velocity field, and at time $t = 0$ we will specify an initial interface. With the interface "immersed" in this two-dimensional velocity field, we will then monitor:

(1) how accurately we are able to follow the exact interface; and
(2) the quality of the corresponding point distribution.

The velocity fields will be prescribed in such a way that we are able to obtain analytic solutions for the exact interfaces at all times.

4.1. Error computation

The way we compute the error, E_1, in following the interface is illustrated in Fig. 7. We first compute the chord between the end points of our numerical solution, and then compute the normal, $\mathbf{n}_c$, to this chord. For each grid point, $(\mathbf{x}^n)_j, j = 0, \ldots, N$, along our numerically approximated interface, we find the intersection between the analytical front and the line originating from $(\mathbf{x}^n)_j$ and moving in the $\mathbf{n}_c$-direction. We call this intersection $(\mathbf{x}^e)_j$ and compute

$$e_j = \sqrt{((x_1^e)_j - (x_1^n)_j)^2 + ((x_2^e)_j - (x_2^n)_j)^2}.$$

Finally, we define the error E_1 as

$$E_1 = \frac{1}{N+1} \sum_{j=0}^{N} e_j. \tag{16}$$

The way we will measure the grid quality is by computing the error, E_2, in the length of the interface. For a high order approximation, this measure will generally give an indication of the quality of the point distribution (although there are special situations when this is not the case). Thus, we first compute the length, S_n, of our numerically computed interface. Using GLL quadrature we compute

$$S_n = \sum_{\alpha=0}^{N} \rho_\alpha (J_s^n)_\alpha = \sum_{\alpha=0}^{N} \rho_\alpha \left(\left(\frac{\partial x_1^n}{\partial \xi} \right)^2 + \left(\frac{\partial x_2^n}{\partial \xi} \right)^2 \right)_\alpha^{1/2}, \tag{17}$$

where J_s^n is the surface Jacobian at time $T = t^n$, and $\rho_\alpha, \alpha = 0, \ldots, N$, are the GLL quadrature weights. We then define the error

$$E_2 = |S_n - S_e|, \tag{18}$$

where S_e is the length of our exact interface.

4.2. Convergence tests

In order to more clearly see the strengths and the weaknesses of the different approaches, we will first consider three examples where the velocity field is chosen such that the front keeps it shape. For these three tests, we will compare three different approaches:

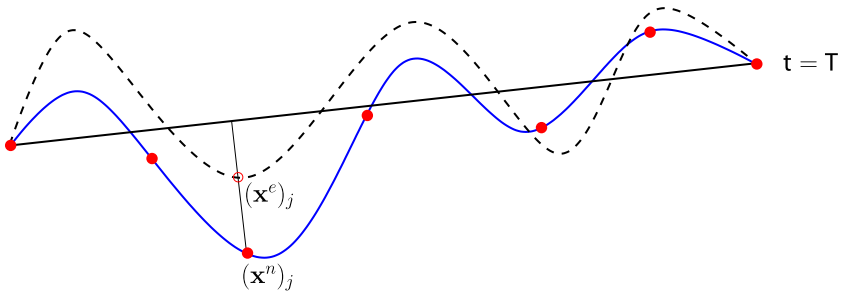


Fig. 7. The solid line corresponds to our numerically computed interface at time $T > 0$, while the dashed line represents the corresponding exact interface.

(i) the proposed method using Strategy 1 (see Fig. 4);
(ii) enforcing the kinematic condition as well as a zero tangential grid velocity (denoted as the "Normal" method in the following);
(iii) a pure Lagrangian approach.

We have also tested the proposed method using Strategy 2 (see Fig. 5). However, Strategy 1 and Strategy 2 give almost identical results for these test problems, so we only report convergence results using Strategy 1.

The fourth and last test will involve a more complex interface and a more complex velocity field; in this test we will only compare (i) and (iii).

4.3. Test 1

In the first example, the interface motion is only in the x_2-direction. The initial front given by

$$x_2(x_1) = \frac{1}{2}\left(1 - \cos\left(\frac{\pi(x_1 - 1)}{3}\right)\right), \quad 1 \leqslant x_1 \leqslant 4,$$

while the prescribed velocity field is given as

$$u_1(x_1, t) = 0$$

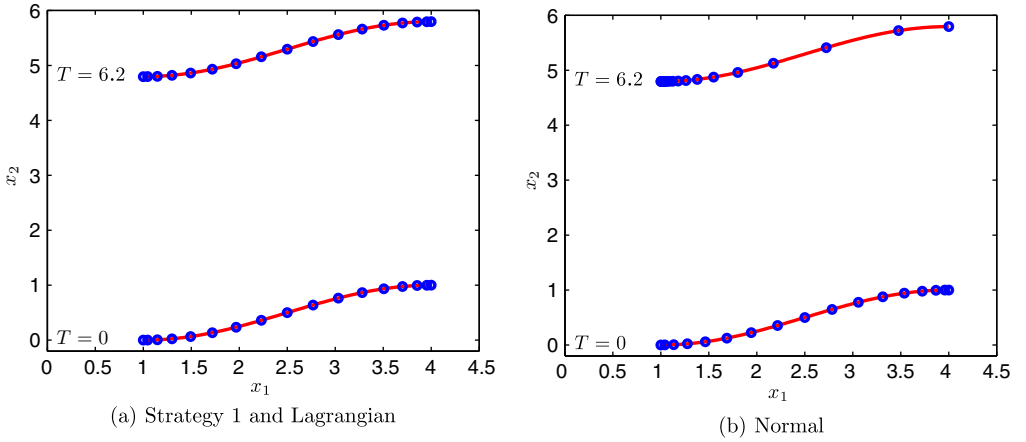
$$u_2(x_1, t) = -\frac{1}{2} + \frac{1}{2}e^{t/4}(1 + \cos(\pi t)).$$

Thus, each particle on the initial front will only move in the x_2-direction. Note that, for the Normal approach, we compute the normals *analytically* (which we can do since we know the analytical expression of the front at all times), and not numerically. A numerical computation of the normals will lead to the Normal algorithm breaking down due to the bad interpolation properties when the point distribution becomes poor. Fig. 8 shows the initial point distribution and the point distribution at the final time $T = 6.2$ for the three different methods. In Figs. 9 and 10, we report the errors E_1 and E_2, respectively. We observe that all three methods perform well in terms of "hitting" the front. However, for the Normal approach, the point distribution is poor at $T = 6.2$ due to the shape of the front; this again leads to a poor approximation of the length of the front. For Strategy 1 and the Lagrangian method, the error E_2 reaches machine precision since u_2 does not depend on x_1, and we use a sufficiently large polynomial degree, N.

4.4. Test 2

Next, we will consider a trivial interface, but a velocity field with a more substantial tangential contribution. The initial interface is given by

$$x_2(x_1) = 0, \quad 1 \leqslant x_1 \leqslant 4,$$



(a) Strategy 1 and Lagrangian (b) Normal

Fig. 8. The interface and the point distribution at the initial time $t = 0$ and at the final time $T = 6.2$ for Test 1 using the three different approaches. A polynomial degree $N = 16$ is used for the spectral approximation.

while the prescribed velocity field is

$$u_1(x_1,t) = \frac{2}{5}\sin\left(\frac{\pi(x_1-1)}{3}\right),$$

$$u_2(x_1,t) = -\frac{1}{2}+\frac{1}{2}e^{t/4}(1+\cos(\pi t)).$$

Hence, the front has no curvature, and all the normals point in the x_2-direction. Thus, $u_1(x,t)$ corresponds to a pure tangential component. Fig. 11 shows the initial point distribution and the point distribution at the final time $T = 6.2$ for the three different methods. In Fig. 12, we report the error E_1. We get first, second and third order temporal convergence in capturing the interface for all three methods.

In Fig. 11, we see that the pure Lagrangian approach leads to a poor point distribution. However, this is not reflected in the computation of the length due to the simplicity of the front. For this simple interface, (17) reduces to

$$S_n = \sum_{\alpha=0}^{N} \rho_\alpha \left(\frac{\partial (x_1)_N}{\partial \xi}\right)_\alpha = \int_{-1}^{1} \frac{\partial (x_1)_N}{\partial \xi}\,\mathrm{d}\xi,$$

since $\frac{\partial (x_1)_N}{\partial \xi}$ is an $(N-1)$th degree polynomial which is integrated exactly using GLL quadrature. In addition, since

$$\int_{-1}^{1} \frac{\partial (x_1)_N}{\partial \xi}\,\mathrm{d}\xi = (x_1)_N(1) - (x_1)_N(-1),$$

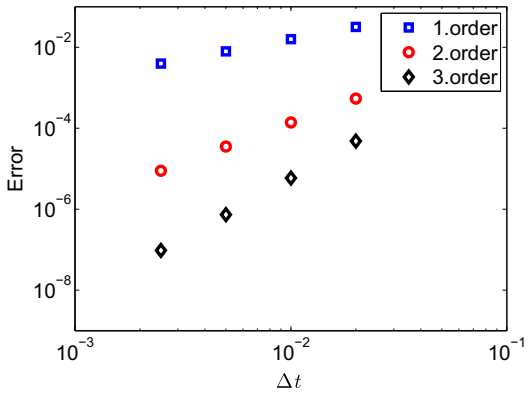
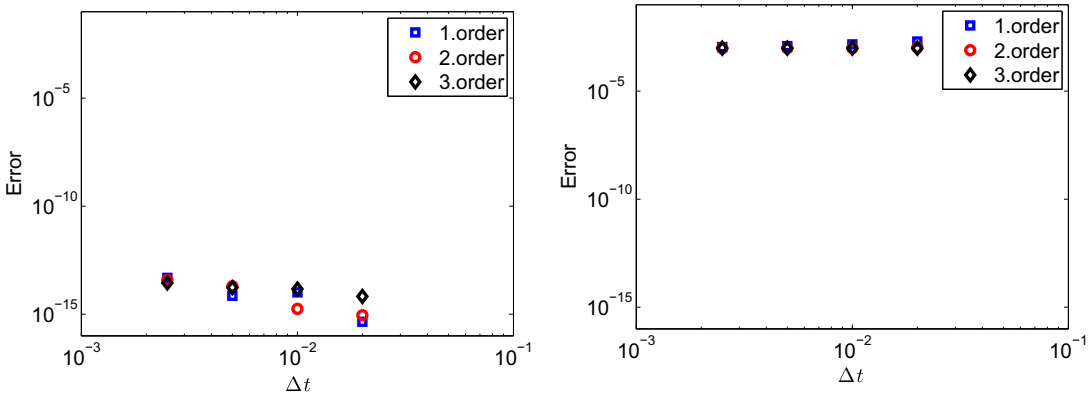
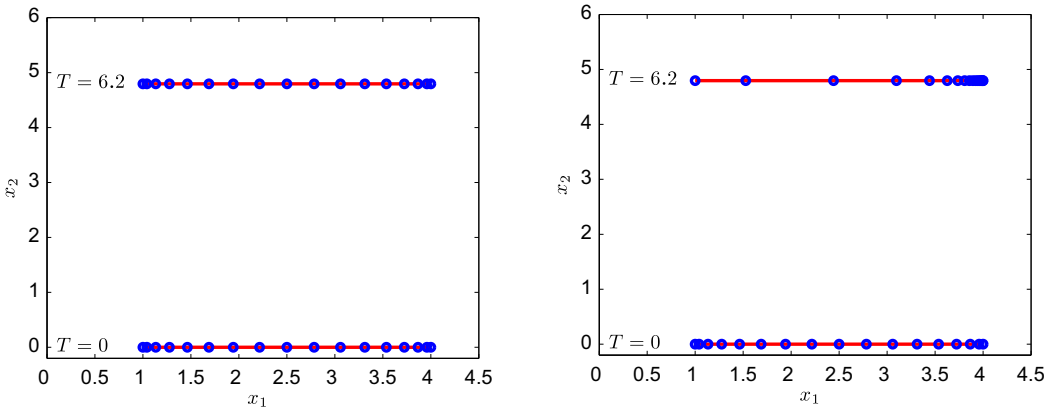


Fig. 9. The error E_1 at time $T = 6.2$ for Test 1 using the three different approaches. A polynomial degree $N = 16$ is used for the spectral approximation. Only a single convergence plot is shown since all three methods give almost identical results.



(a) Strategy 1 and Lagrangian (b) Normal

Fig. 10. The error E_2 at time $T = 6.2$ for Test 1 using the three different approaches. A polynomial degree $N = 16$ is used for the spectral approximation. The computed value for the length of the interface, S_n, does not converge for the Normal method due to an incorrect point distribution. The results for Strategy 1 and the Lagrangian method are almost identical (the error is close to machine precision); hence, only a single convergence plot is shown for these methods.



(a) Strategy 1 and Normal (b) Lagrangian

Fig. 11. The interface and the point distribution at the initial time $t = 0$ and at the final time $T = 6.2$ for Test 2 using the three different approaches. A polynomial degree $N = 16$ is used for the spectral approximation.

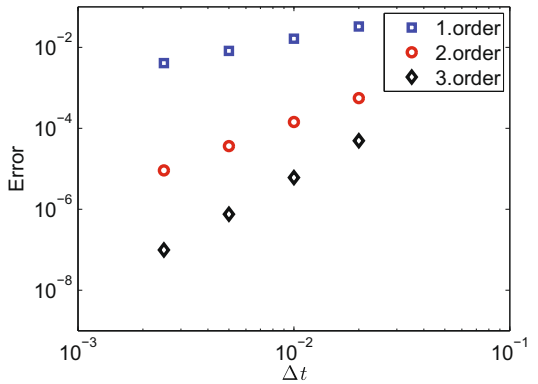


Fig. 12. The error E_1 at time $T = 6.2$ for Test 2 using the three different approaches. A polynomial degree $N = 16$ is used for the spectral approximation. Only a single convergence plot is shown since all three methods give almost identical results.

and since the end points are the same for all strategies, we achieve machine precision for all three methods regardless of the quality of the grid. It should be emphasized that this is, indeed, a very special case.

4.5. Test 3

We now consider a combination of the previous two tests. The initial front is the same as in Test 1, but we prescribe a velocity field with non-zero components in both the x_1- and the x_2-direction. In particular, we choose the same velocity field as in Test 1, but with the modification that we also add another tangential component. Hence, the interface will still keep its shape during the simulation. Our initial interface is then given by

$$x_2(x_1) = \frac{1}{2}\left(1 - \cos\left(\frac{\pi(x_1 - 1)}{3}\right)\right), \quad 1 \leqslant x_1 \leqslant 4,$$

while the prescribed velocity field is given as

$$u_1(x_1, t) = u_1^t,$$

$$u_2(x_1, t) = -\frac{1}{2} + \frac{1}{2}e^{t/4}(1 + \cos(\pi t)) + u_2^t,$$

with

$$u_1^t = \frac{2}{5}\sin\left(\frac{\pi(x_1 - 1)}{3}\right),$$

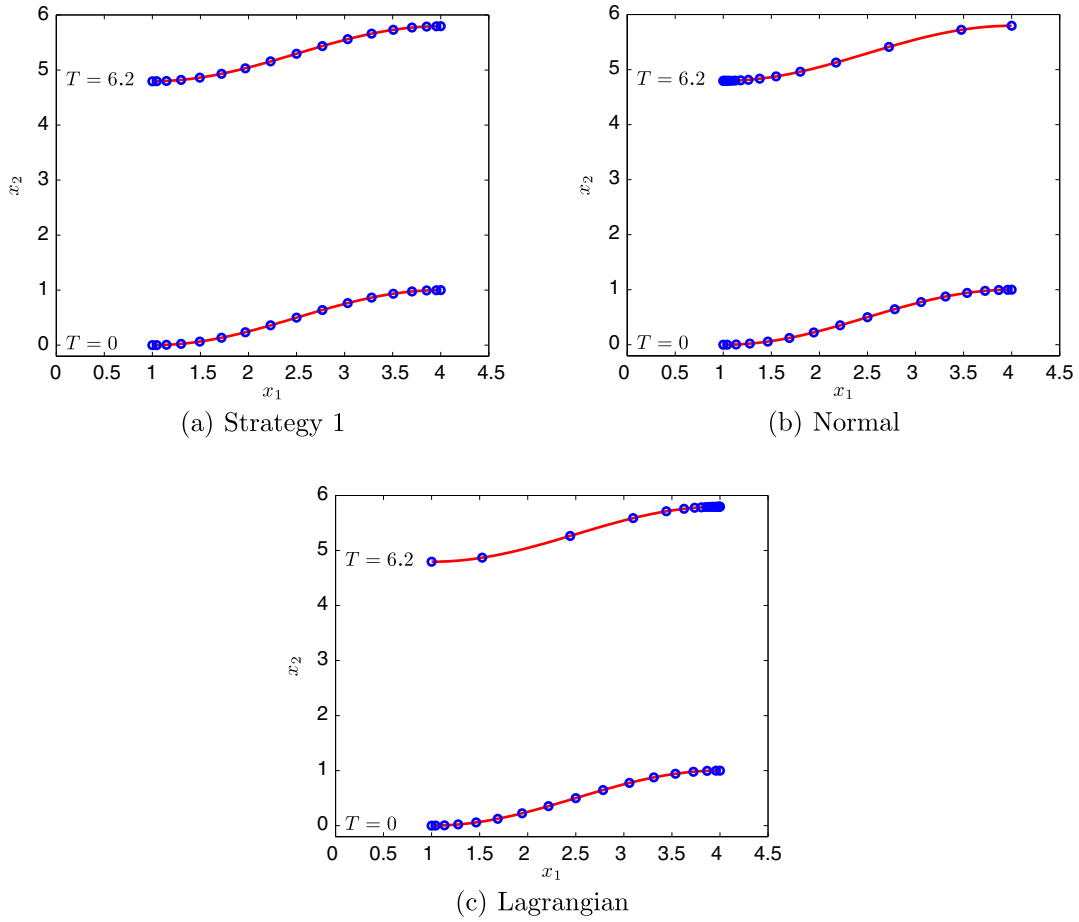


Fig. 13. The interface and the point distribution at the initial time $t = 0$ and at the final time $T = 6.2$ for Test 3 using the three different approaches. A polynomial degree $N = 16$ is used for the spectral approximation.

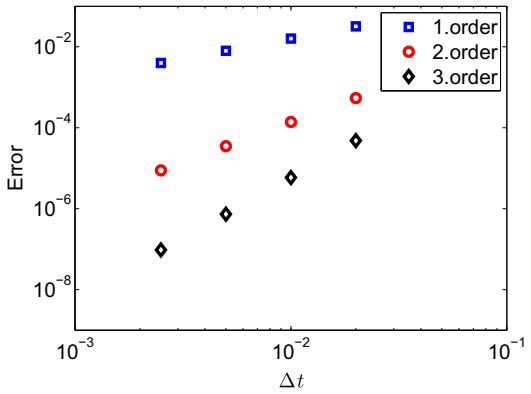


Fig. 14. The error E_1 at time $T = 6.2$ for Test 3 using the three different strategies. A polynomial degree $N = 16$ is used for the spectral approximation. Only a single convergence plot is shown since all three methods give almost identical results.

$$u_2^t = \frac{\pi}{6}\sin\left(\frac{\pi(x_1-1)}{3}\right)\frac{2}{5}\sin\left(\frac{\pi(x_1-1)}{3}\right).$$

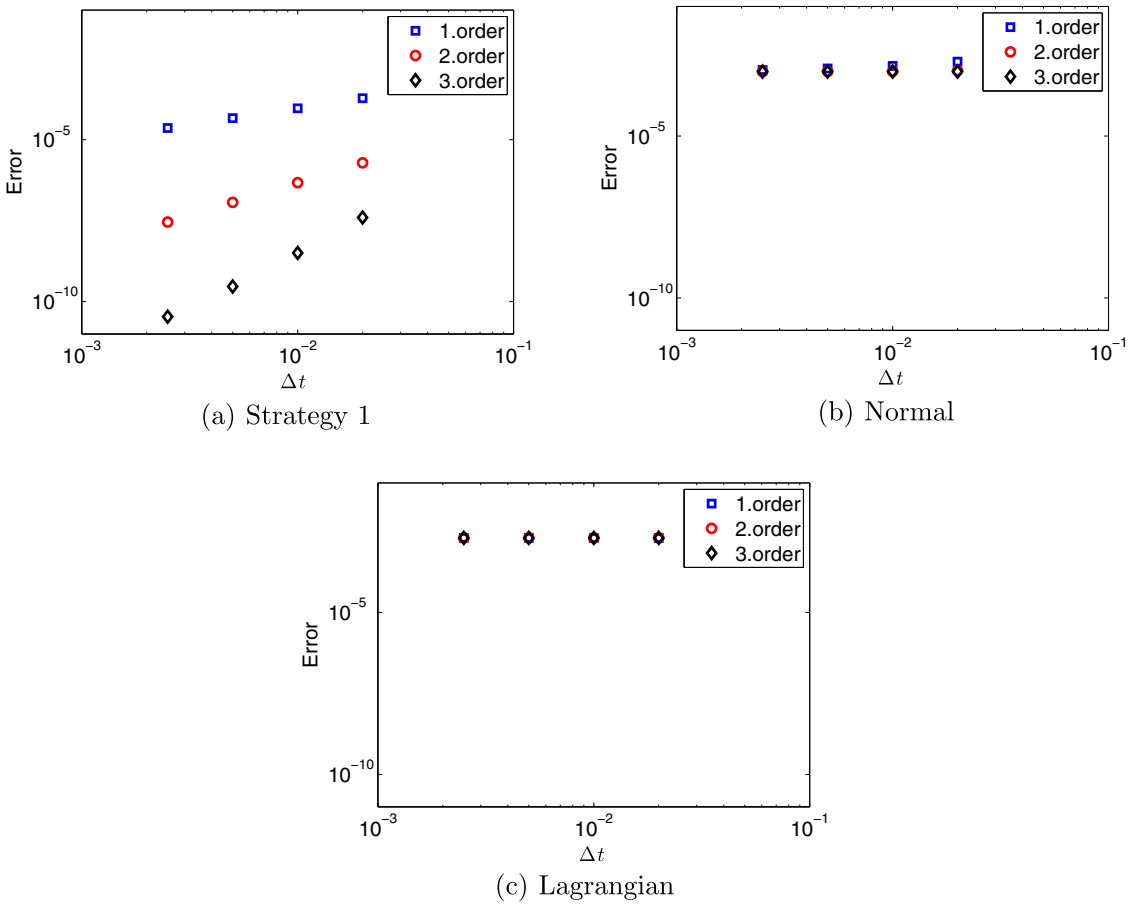


Fig. 15. The error E_2 at time $T = 6.2$ for Test 3 using the three different approaches. A polynomial degree $N = 16$ is used for the spectral approximation. The computed value for the length of the interface, S_n, does not converge for the normal method and for the Lagrangian method due to a highly incorrect point distribution.

Here, $\mathbf{u}^t = u_1^t \mathbf{e}_1 + u_2^t \mathbf{e}_2$ is a vector which points in the tangential direction of the interface. Fig. 13 shows the initial point distribution and the point distribution at the final time $T = 6.2$ for the three different methods.

In Figs. 14 and 15, we report the errors E_1 and E_2, respectively. These results are in agreement with the two previous numerical experiments. We observe that all three methods perform well in terms of "hitting" the front.

Strategy 1 still performs well with respect to both error measures. The Normal approach gives the same results as for Test 1 since the only difference with this example is the addition of a tangential velocity component; again, the point distribution is poor. The grid quality for a pure Lagrangian approach is also poor; this is because the added tangential velocity component will produce a (physical) point distribution which is far from the GLL point distribution on the reference domain.

Another thing we observe is that the error level for E_2 (the length computation) is about a factor $10^2 - 10^3$ smaller than the error level for E_1 (the interface error). The reason for this is that the interface error is rather uniform, which again is due to the simplicity of the problem. This makes the error in the spatial *derivative* of the interface substantially smaller than the interface error, which again leads to a better approximation of the length of the front.

4.6. Test 4

The three previous test cases were all constructed to illuminate some of the strengths and weaknesses of the different methods for tracking an interface; for this reason they were chosen to be rather simple. We now consider a more complicated numerical example in order to demonstrate the applicability of the new strategy to solve more general problems. We still choose a velocity field which depends on the initial shape of the front, and in such a way that we are able to derive an analytical expression for the shape of the front at all times. A major difference from the previous test cases is that we now choose a time-dependent front. In particular, we demand that the *shape* of the front is given by

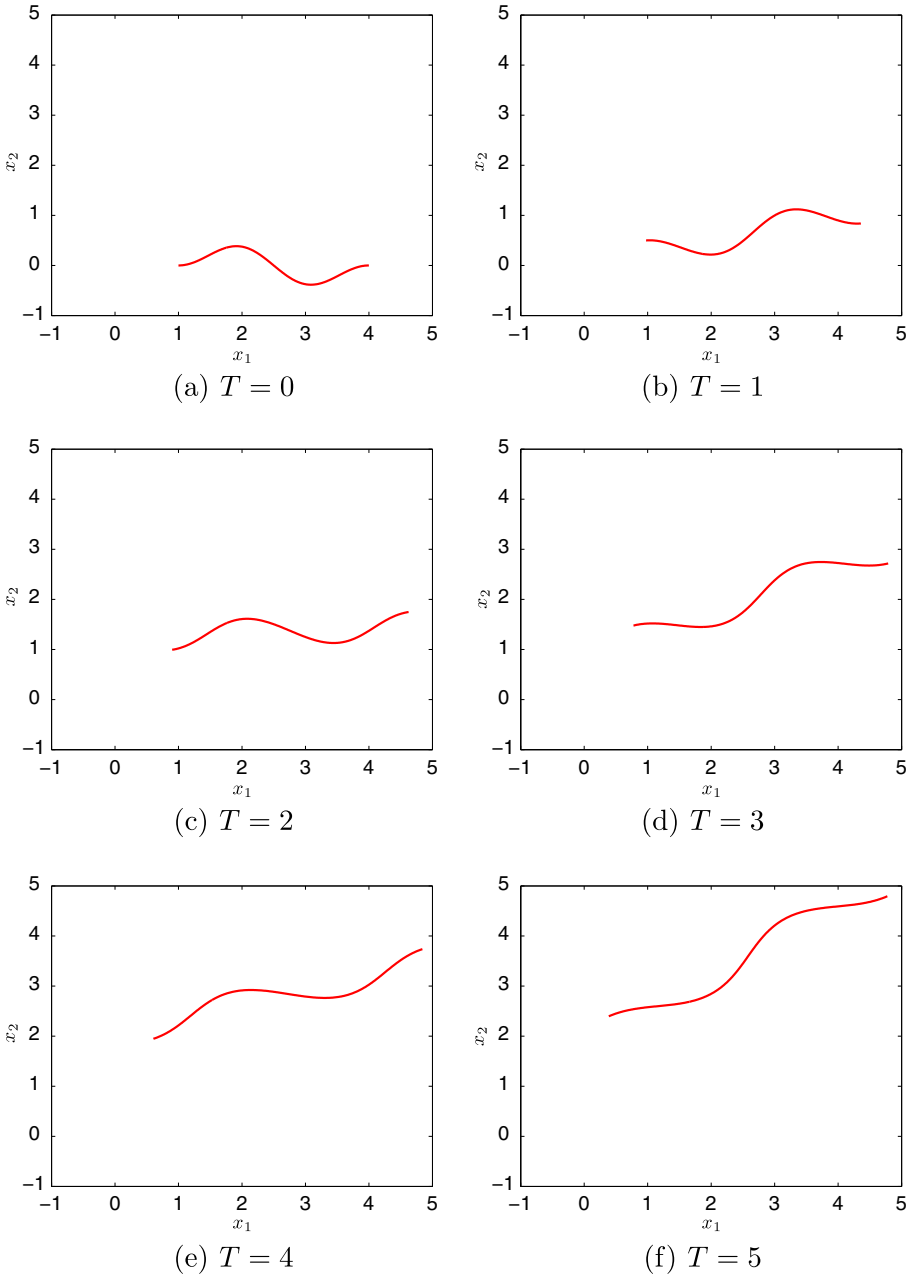


Fig. 16. The front $x_2(x_1, t)$ in (19) at different times when "immersed" in the prescribed velocity field.

$$x_2(x_1, t) = \frac{1}{4}\cos(\pi t)\left(\cos\left(\frac{\pi(x_1 - 1)}{3 + 0.4t}\right) - \cos\left(\frac{3\pi(x_1 - 1)}{3 + 0.4t}\right)\right). \tag{19}$$

Hence, the front will have a time-dependent amplitude and a time-dependent wavelength. We also wish to rotate the front in a circular motion, and in order to achieve this, the velocity field must be chosen in a careful manner. In particular, it consists of four contributions:

- an angular velocity which is responsible for a pure rotation of the initial front. The angle is computed with respect to a circle with center $(-4, 0)$, and a constant angular velocity $u_\theta = 0.1$ is imposed;
- a velocity field which accounts for the time-dependent amplitude in (19);
- a velocity field which "stretches" the front in accordance with the time-dependent wavelength in (19);
- an additional velocity field in the tangential direction on the front.

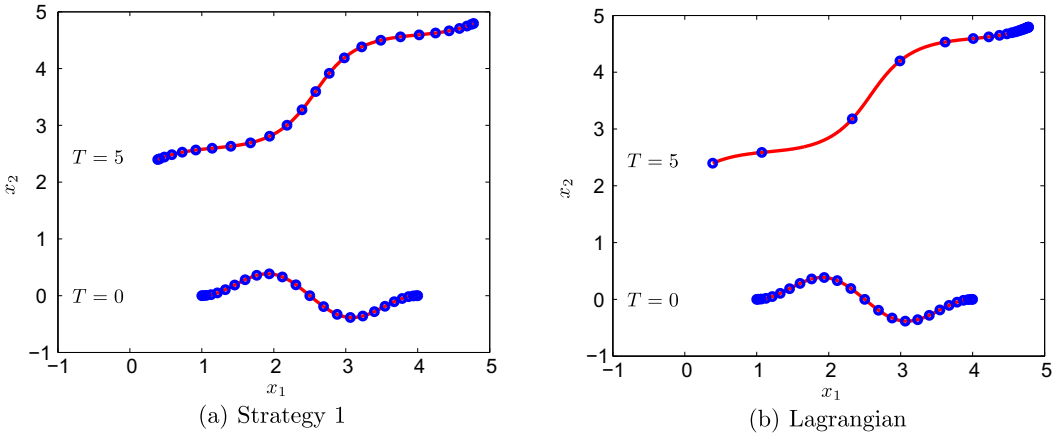


Fig. 17. The interface and the point distribution at the initial time $t = 0$ and at the final time $T = 5$ for Test 4 using Strategy 1 and the Lagrangian approach. A polynomial degree $N = 24$ is used for the spectral approximation.

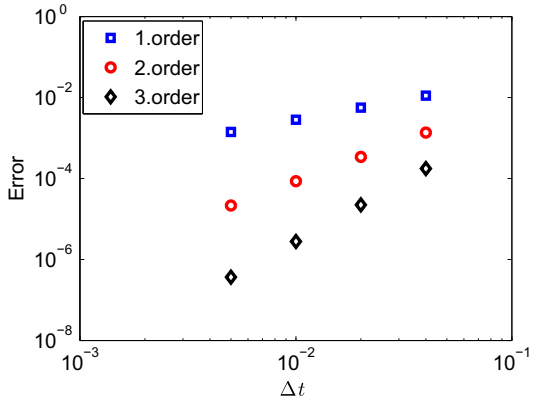


Fig. 18. The error E_1 at time $T = 5$ for Test 4 using Strategy 1 and the Lagrangian approach. A polynomial degree $N = 24$ is used for the spectral approximation. Only a single convergence plot is shown since both methods give almost identical results.

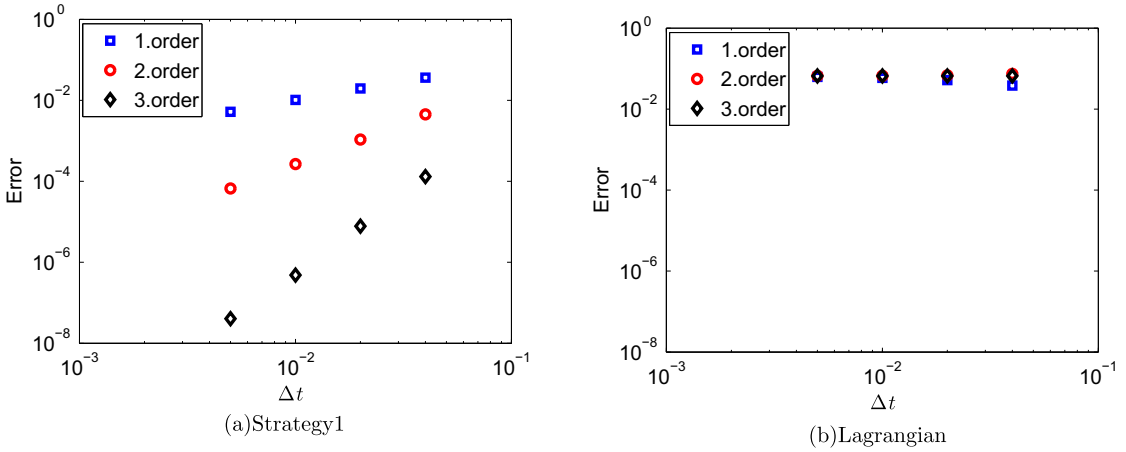


Fig. 19. The error E_2 at time $T = 5$ for Test 4 using Strategy 1 and the Lagrangian approach. A polynomial degree $N = 24$ is used for the spectral approximation. The computed value for the length of the interface, S_n, does not converge for the Lagrangian method due to an incorrect point distribution.

By adding these four contributions, we obtain a smooth, two-dimensional, time-dependent velocity field. Apart from spatial and temporal discretization errors, the initial front $x_2(x_1, t = 0)$ will keep the *shape* given by (19) when "immersed" in our velocity field; see Fig. 16.

In Fig. 17, we show the initial point distribution and the point distribution at the final time $T = 5$ using Strategy 1 and a Lagrangian approach. In Figs. 18 and 19, we report the errors E_1 and E_2 for the two methods, respectively. We see that Strategy 1 performs well both in terms of "capturing" the front and in terms of giving the correct length. The Lagrangian approach is also able to "capture" the front, but the point distribution is poor such that the error in the length of the front is large. Also, compared with Test 3, the difference between the error levels for E_1 and E_2 is now much smaller; this is due to the time-dependent amplitude in (19), which makes the interface error much less uniform.

5. Conclusions

We have presented a new approach for tracking an interface immersed in a given velocity field. The method is particularly relevant to the simulation of unsteady free surface problems using the arbitrary Lagrangian–Eulerian framework. The new method has been constructed with two goals in mind: (i) to be able to accurately follow the interface; and (ii) to automatically maintain a good distribution of the grid points along the interface. The method combines information from a pure Lagrangian approach with information from an ALE approach.

We have been able to construct two-dimensional model problems offering analytical expressions for both the interface as well as the prescribed velocity field in which the interface (or front) is "immersed". This has allowed us to verify and compare the temporal accuracy of different methods: the new approach, a pure Lagrangian approach, and an approach honoring the kinematic condition in the normal direction, but imposing a homogeneous Dirichlet condition for the tangential component of the grid velocity (called the Normal approach).

Using the new approach we have been able to achieve both of our primary objectives; in particular, we have verified first, second, and third order temporal accuracy for all four model problems. The new method is particularly important in the context of using high order spatial discretization schemes.

Both the Lagrangian approach and the Normal approach generally give a non-optimal point distribution along the interface, something which again may result in large errors in the computation of important surface quantities (e.g., normal and tangent vectors, local curvature, length etc.). Such errors may, in the worst case, result in a complete breakdown of the interface-tracking.

The new method should be extended to, and tested in, more general situations, in particular, by solving real free surface problems using an ALE approach, and by extending the approach to three dimensions.

Acknowledgment

The work has been supported by the Norwegian University of Science and Technology and the Research Council of Norway under Contract 185336/V30. The support is gratefully acknowledged.

References

[1] T. Bjøntegaard, E.M. Rønquist, A high order splitting method for time-dependent domains, Computer Methods in Applied Mechanics and Engineering 197 (2008) 4763–4773.
[2] R. Bouffanais, M. Deville, Mesh update techniques for free-surface flow solvers using spectral elements, Journal of Scientific Computing 27 (1–3) (2006) 137–149.
[3] W. Dettmer, D. Perić, A computational framework for free surface fluid flows accounting for surface tension, Computer Methods in Applied Mechanics and Engineering 195 (2006) 3038–3071.
[4] J. Donea, S. Giuliani, J. Halleux, An arbitrary Lagrangian–Eulerian finite element method for transient dynamic fluid–structure interactions, Computer Methods in Applied Mechanics and Engineering 33 (1982) 689–723.
[5] F. Duarte, R. Gormaz, S. Natesan, Arbitrary Lagrangian–Eulerian method for Navier–Stokes equations with moving boundaries, Computer Methods in Applied Mechanics and Engineering 193 (2004) 4819–4836.
[6] F. Giraldo, The Lagrange–Galerkin spectral element method on unstructured quadrilateral grids, Journal of Computational Physics 147 (1998) 114–146.
[7] F. Giraldo, Hybrid Eulerian–Lagrangian semi-implicit time-integrators, Computers and Mathematics with Applications 52 (2006) 1325–1342.
[8] C. Hirt, A. Amsden, J. Cook, An arbitrary Lagrangian–Eulerian computing method for all flow speeds, Journal of Computational Physics 14 (1974) 227–253.
[9] C.W. Hirt, B. Nichols, Volume of fluid (VOF) method for the dynamics of free boundaries, Journal of Computational Physics 39 (1981) 201–225.
[10] L. Ho, A. Patera, A Legendre spectral element method for simulation of unsteady incompressible viscous free-surface flows, Computer Methods in Applied Mechanics and Engineering 80 (1990) 355–366.
[11] L. Ho, A. Patera, Variational formulation of three-dimensional viscous free-surface flows: natural imposition of surface tension boundary conditions, International Journal for Numerical Methods in Fluids 13 (1991) 691–698.
[12] A. Huerta, A. Rodríguez-Ferran (Eds.), The Arbitrary Lagrangian–Eulerian Formulation, Computer Methods in Applied Mechanics and Engineering 193 (39–41) (2004) 4073–4456.
[13] A. Johnson, T. Tezduyar, Mesh update strategies in parallel finite element computations of flow problems with moving boundaries and interfaces, Computer Methods in Applied Mechanics and Engineering 119 (1994) 73–94.
[14] Y. Maday, A. Patera, Spectral element methods for the Navier–Stokes equations, in: A.K. Noor, J.T. Oden (Eds.), State of the Art Surveys in Computational Mechanics, ASME, New York, 1989, pp. 71–143.
[15] Y. Maday, A. Patera, E. Rønquist, An operator-integration-factor splitting method for time-dependent problems: application to incompressible fluid flow, Journal of Scientific Computing 5 (4) (1990) 263–292.
[16] S. Osher, R. Fedkiw, Level Set Methods and Dynamic Implicit Surfaces, Springer-Verlag, 2002.

[17] S. Osher, J. Sethian, Fronts propagating with curvature-dependent speed: algorithms based on Hamilton–Jacobi formulations, Journal of Computational Physics 79 (1988) 12–49.
[18] B. Ramaswamy, M. Kawahara, Arbitrary Lagrangian–Eulerian finite element method for unsteady convective incompressible viscous free surface flow, International Journal for Numerical Methods in Fluids 7 (1987) 1053–1074.
[19] J. Sethian, Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science, Cambridge University Press, 1999.
[20] M. Walkey, P. Gaskell, P. Jimack, M. Kelmanson, J. Summers, Finite element simulation of three-dimensional free-surface flow problems, Journal of Scientific Computing 24 (2) (2005) 147–162.
[21] D. Xiu, S. Sherwin, S. Dong, G. Karniadakis, Strong and auxiliary forms of the semi-Lagrangian methods for incompressible flows, Journal of Scientific Computing 25 (1–2) (2005) 322–346.
[22] J. Xu, D. Xiu, G. Karniadakis, Semi-Lagrangian method for turbulence simulations using mixed spectral discretizations, Journal of Scientific Computing 17 (1–4) (2002) 585–597.
[23] H. Zhou, J. Derby, An assessment of a parallel, finite element method for three-dimensional, moving-boundary flows driven by capillarity for simulation of viscous sintering, International Journal for Numerical Methods in Fluids 36 (2001) 841–865.