Zentrum für Technomathematik

Fachbereich 3 – Mathematik und Informatik

A level set toolbox including reinitialization and mass correction algorithms for FEniCS

Mischa Jahn **Timo Klock**

Report 16–01

Berichte aus der Technomathematik

Report 16–01 **February 2016**

A LEVEL SET TOOLBOX INCLUDING REINITIALIZATION AND MASS CORRECTION ALGORITHMS FOR FENICS

M. JAHN AND T. KLOCK

Abstract. In this article, an overview of the level set method is given and a toolbox for the numerical solution of level set problems is presented. Mainly based on the work of [8], we mention various aspects of the level set method including discretization and stabilization aspects, as well as the reinitialization of the level set function. Additionally, global and local mass resp. volume correction approaches adapted from [8] respectively [16] are presented for maintaining the level set function during its evolution in time. All described models and methods are implemented into a toolbox for the `FEniCS` framework [14].

1. Introduction

Many engineering processes include time dependent movements of discontinuities, e.g. a solid-liquid interface in melting and solidification processes or a surface within a two-phase flow. For the modeling and simulation of such a process, a representation of the discontinuity is needed.

There are many approaches available to characterize discontinuities mathematically, however, a very popular choice is the level set method introduced by Osher and Sethian [19]. The basic idea of the level set method is to represent a time dependent discontinuity implicitly by the zero level set of a continuous scalar function whose evolution in time is described by a transport equation.

Unfortunately, solving the level set equation numerically using standard finite elements may cause degeneration of the function's gradient and lack of mass resp. volume conservation. Therefore, a need for methods preserving these properties arises, i.e. the level set function has to be reinitialized and the mass resp. volume enclosed by a level set has to be corrected. In connection with maintaining the function, the construction of a discrete representation of the discontinuity has to be considered.

In this article, a level set toolbox including reinitialization and mass conserving methods for the finite element framework `FEniCS` [14] is presented. Started in 2003, the idea of the `FEniCS` Project is to automate the solution of mathematical models based on PDEs. By using different software libraries that are integrated into one package, the user can specify a PDE-based problem in weak form and `FEniCS` generates the code automatically. Using this automated code generation approach, the numerical simulation of different physical processes and engineering applications can be easily implemented.

This article is organized as the following: In Section 2, a short introduction to the level set method including comments on the derivation of the weak formulation of the level set problem is given. Section 3 deals with the discretization and stabilization of the level set equation. Following [8], a discrete representation of the discontinuity is derived and a reinitialization technique is presented. To conserve mass during the evolution of the level set function, the approaches of global [8] and local [16] mass resp. volume correction are presented. Aspects of the implementation of the level set toolbox in `FEniCS` are given in detail in Section 4 and numerical results are shown in Section 5.

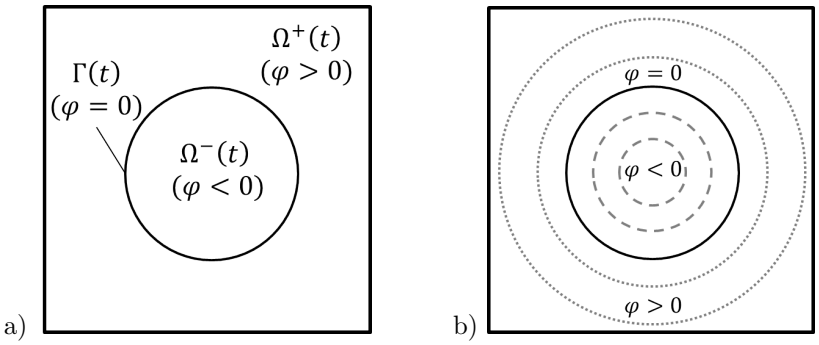


FIGURE 2.1. Visualization of the idea of the level sets method using a scalar function φ: a) Domains $\Omega^+(t)$ and $\Omega^-(t)$ are separated by the zero level set $\Gamma(t)$ of φ. b) Visualization of some level sets of φ.

2. The level set method: Review

2.1. **Background.** The basic idea of the level set method is to define a continuous scalar function $\varphi\colon \Omega \times [t_0, t_f] \to \mathbb{R}$ on a given domain $\Omega \subset \mathbb{R}^d$, $d = 2, 3$, whereas the zero level set of φ

$$\Gamma(t) = \{x \in \Omega : \varphi(x, t) = 0\}, \quad t \in [t_0, t_f],$$

represents a time dependent discontinuity, for example an interface, in an implicit way. By using the sign of $\varphi = \varphi(\cdot, t)$, the hold-all domain Ω can be divided into the subdomains

$$\Omega(t) = \Omega^+(t) \cup \Omega^-(t) \cup \Gamma(t),$$

with $x \in \Omega^+(t) \Leftrightarrow \varphi(x, t) > 0$ and $x \in \Omega^-(t) \Leftrightarrow \varphi(x, t) < 0$. An exemplary sketch of a 2D situation where a hold-all domain Ω is divided by the sign of the function φ into subdomains $\Omega^+(t)$ resp. $\Omega^-(t)$ is given in Fig. 2.1a and some level sets of φ are indicated in Fig. 2.1b.

In regards to geometrical properties, the level set method allows for an easy computation of the normal $\vec{n}$ to Γ

$$\vec{n} = \frac{\nabla\varphi}{||\nabla\varphi||},$$

and the curvature of Γ reads as

$$\kappa = -\mathrm{div}\vec{n} = -\mathrm{div}\frac{\nabla\varphi}{||\nabla\varphi||}.$$

These properties are useful in many applications, e.g. if considering two-phase flow including surface tension on the interface or the two-phase Stefan problem.

There are various functions φ which can be defined and used within the level set method, however, from a numerical point of view it is important, e.g. for a stable computation of $\vec{n}$ and κ, that the gradient $||\nabla\varphi||$ does neither vanish nor become too big. Due to this, literature suggest to use a so called *signed distance function*, i.e.

$$\varphi(x, t) = \begin{cases} -\min\limits_{y \in \Gamma(t)} ||x - y||_2, & \text{if } x \in \Omega^-(t) \\ \min\limits_{y \in \Gamma(t)} ||x - y||_2, & \text{if } x \in \Omega^+(t) \end{cases},$$

which satisfies $||\nabla\varphi|| = 1$. This condition is important from a numerical point of view, as it guarantees a stable computation of $\vec{n}$ and κ.

Following [8], a transport equation for the function φ can be derived, if we consider the movement of a particle $X(t)$ in a sufficiently smooth and divergence free velocity field $\vec{u} = \vec{u}(x,t)$ that is given by

$$\frac{d}{dt}X(t) = \vec{u}(x,t), \quad t \in [t_0, t_f]. \tag{2.1}$$

We want the values of the level set function $\varphi(x,t)$ to be constant for the particle $X(t)$, $t \in [t_0, t_f]$ and therefore define

$$\varphi(X(t),t) = \varphi(X(t_0),t_0) = const \tag{2.2}$$

Total differentiation leads to the transport equation

$$\varphi_t + \vec{u} \cdot \nabla\varphi = 0, \tag{2.3}$$

which describes the evolution respectively the motion of the discontinuity Γ in time.

To get a proper problem formulation of the level set problem, initial and boundary conditions have to be defined. For t_0 let $\varphi_0(x) = \varphi(x,t_0)$ be a sufficiently smooth function, e.g. a signed distance function, with the zero level set $\Gamma(t_0)$. Additionally, we define an *inflow boundary*

$$\partial\Omega_{\text{in}}(t) = \{x \in \partial\Omega : \vec{u}(x,t) \cdot \vec{n}(x) < 0\} \tag{2.4}$$

and a continuous function $\varphi_D \colon \partial\Omega \times [t_0, t_f] \to \mathbb{R}$. The level set problem in strong formulation is then given by:
Find $\varphi(x,t) \in C^1(\Omega \times [t_0,t_f]) \cap C^0(\bar{\Omega} \times [t_0,t_f])$, s.t.

$$\begin{aligned} \varphi_t + \vec{u} \cdot \nabla\varphi &= 0 && \text{in } \Omega \times [t_0,t_f], \\ \varphi(x,t_0) &= \varphi_0(x) && \text{in } \Omega, \\ \varphi(x,t) &= \varphi_D(x,t) && \text{on } \partial\Omega_{\text{in}}(t) \times [t_0,t_f]. \end{aligned} \tag{2.5}$$

2.2. **Weak formulation.** To get a weak formulation of the level set problem (2.5), we define the time dependent function space

$$V_{u,D} = \{v \in L^2(\Omega) : u \cdot \nabla v \in L^2(\Omega) \wedge v|_{\partial\Omega_{\text{in}}} = \varphi_D\}. \tag{2.6}$$

By multiplying (2.3) with an arbitrary test function $v \in L^2(\Omega)$ and integrating over Ω, we get the weak formulation of the level set problem (2.5):
For $t \in [t_0, t_f]$ find $\varphi(\cdot,t) \in V_{u,D}$ with $\varphi_t \in L^2(\Omega)$ s.t. $\varphi(\cdot,t_0) = \varphi_0$ and

$$(\varphi_t, v)_{L^2} + (\vec{u} \cdot \nabla\varphi, v)_{L^2} = 0, \quad \forall v \in L^2(\Omega). \tag{2.7}$$

3. Numerical method

3.1. **Discretization.**

3.1.1. *Discretization in space.* Let $\{\mathcal{S}_h\}_{h>0}$ be a family of shape regular triangulations consisting of d-simplices with d denoting the dimension and h is the maximum diameter $h = \max_{S \in \mathcal{S}_h} \operatorname{diam}(S)$. For simplicity, we restrict ourselves to quasi-uniform triangulations $\mathcal{S}_h$. For each triangulation we define the standard Lagrangian finite element space

$$V_h^k = \{v_h \in C(\Omega) : v_h|_S \in \mathcal{P}_k, \ \forall S \in \mathcal{S}_h\}, \tag{3.1}$$

and for functions with Dirichlet boundary conditions we introduce

$$V_{h,\text{D}}^k = \{v_h \in C(\Omega) : v_h|_S \in \mathcal{P}_k, \ \forall S \in \mathcal{S}_h, v(x) = \varphi_D(x), \ \forall x \in \partial\Omega_{\text{in},h}\}, \tag{3.2}$$

with $k \geq 1$ and $\partial\Omega_{\text{in},h}$ being the discrete influx boundary. Using this function spaces, (2.7) discretized in space reads: For $t \in [t_0, t_f]$ find $\varphi(\cdot, t) \in V_{h,\text{D}}^k$ with $\vec{u} \cdot \nabla\varphi_h \in L^2(\Omega)$ such that

$$\sum_{S \in \mathcal{S}_h} \left(\frac{\partial \varphi_h}{\partial t} + \vec{u} \cdot \nabla \varphi_h, v_h \right)_{L^2(S)} = 0, \quad \forall v_h \in V_h. \tag{3.3}$$

In many applications, e.g. multi-phase flow, the polynomial degree $k = 2$ is chosen for the finite-dimensional function space (3.2). This is due to different reasons, for example the quality of the curvature approximation of the level set function containing second derivatives, as pointed out in [8]. Moreover, using quadratic basis functions has the additional advantage that the degrees of freedom coincide the the degrees of freedom of linear basis functions on a regularly refined mesh. This will be extensively exploited for characterizing the interface Γ discretely and by the reinitialization technique.

3.1.2. *Discretization in time.* For time discretization, a $\theta-$scheme is used here. We discretize the interval $[t_0, t_f]$ by $N+1$ time steps $t_n = t_0 + n\Delta t$, $n = 0, \ldots, N$ with Δt denoting the time step. Let $\theta \in [0, 1]$ be a parameter and $\varphi_h^n(\cdot) \approx \varphi(\cdot, t_n)$ be an approximation of the level set function φ at time t_n. The completely discretized level set problem reads

$$\sum_{S \in \mathcal{S}_h} \left(\frac{\varphi_h^{n+1} - \varphi_h^n}{\Delta t} + \theta \vec{u}^{n+1} \nabla \varphi_h^{n+1} + (1-\theta) \vec{u}^n \nabla \varphi_h^n, v_h \right)_{L^2(S)} = 0, \tag{3.4}$$

for all $v_h \in V_h$. Note that $\theta = 0$ leads to the explicit Euler-scheme while $\theta = 1$ results in the implicit Euler-scheme.

3.2. **Stabilization.** It is well known, that solving hyperbolic PDEs with standard finite element methods can be instable, especially for high velocities $\vec{u}$. An approach to overcome this issue is using a stabilization method [21] to slightly reformulate the discretized problem to enforce stability. A method well known in literature is the *Streamline-Upwind/Petrov-Galerkin* (SUPG) stabilization [5].

As proposed in [8], special test functions $\tilde{v}_h \in L^2(\Omega)$ of the form

$$\tilde{v}_h|_S := v_h + \delta_S \vec{u} \cdot \nabla v_h, \quad S \in \mathcal{S}_h,\ v_h \in V_h$$

are used in eq. (3.3) for stabilization, where $\delta_S \in [0, 1]$ is a parameter. Then, the fully discretized and stabilized weak formulation of (2.3) is given by

$$\sum_{S \in \mathcal{S}_h} \left(\frac{\varphi_h^{n+1} - \varphi_h^n}{\Delta t} + \theta \vec{u}^{n+1} \nabla \varphi_h^{n+1} + (1-\theta) \vec{u}^n \nabla \varphi_h^n, v_h + \delta_S \vec{u} \cdot \nabla v_h \right)_{L^2(S)} = 0, \tag{3.5}$$

for all test functions $v_h \in V_h$.

In literature, it is suggested to use a δ_S that depends on the velocity $\vec{u}$ and the diameter of the simplex $h_S = \text{diam}(S)$, for $S \in \mathcal{S}_h$,

$$\delta_S = c \frac{h_S}{\max\{\delta_0, ||u||_{\infty,S}\}}, \tag{3.6}$$

with $0 < \delta_0 \ll 1$ and $c \in [0, 1]$. Note, if we choose $c = 0$, then no stabilization is applied and we get eq. (3.3) resp. eq. (3.4).

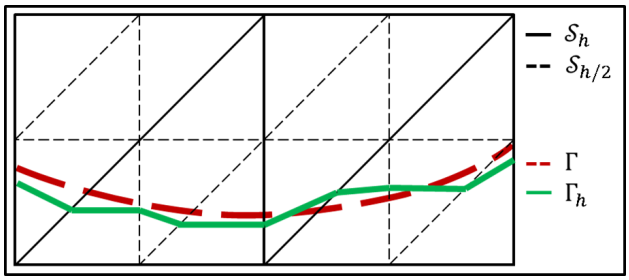


FIGURE 3.1. Construction of the discrete representation Γ_h of a interface Γ

3.3. **Representation of Γ_h.** For the computation, a discrete representation of Γ is needed. In this paper, we follow the approach of [8] and approximate the interface linearly on a refined triangulation.

For $t_n = t_0 + n\Delta t$, $n = 0, \dots, N$, let $\varphi_h(\cdot, t_n) \in V_h^2$ be the finite element approximation of the level set function φ, $\tilde{\Gamma}_h$ its zero level, and

$$\mathcal{S}_h^\Gamma := \left\{ S \in \mathcal{S}_h : \mathrm{meas}_{\mathrm{d}-1}(S \cap \tilde{\Gamma}_h) > 0 \right\} \tag{3.7}$$

the set of d-simplices containing $\tilde{\Gamma}_h$. We drop the time t_n as an argument in this paragraph in our notation for simplicity and define $\mathcal{S}_{h/2}^\Gamma$ as the set consisting of all d-simplices that are obtained, if the elements in $\mathcal{S}_h^\Gamma$ are regularly refined.

The finite element approximation φ_h of φ is then linearly interpolated by $I\varphi_h$ on the patch of refined elements $S \in \mathcal{S}_{h/2}^\Gamma$ and the discrete approximation of Γ is given by

$$\Gamma_h := \{ x \in \Omega : I\varphi_h(x) = 0 \}, \tag{3.8}$$

as shown in Fig. 3.1. A detailed investigation about the approximation quality and the discretization error of this discrete interface representation is given in [8].

While a high order approximation of the interface Γ is possible, cf. [6], using the presented approach to get a discrete representation Γ_h has several advantages. First of all, the degrees of freedom (DOFs) of φ_h and $I\varphi_h$ coincide due to the choice of $\varphi_h \in V_h^2$, allowing for a fast interpolation. More important, the segments of $S \cap \Gamma_h$, $S \in \mathcal{S}_{h/2}$ are straight resp. planar which makes the computation of intersection points very easy. This fact is heavily utilized during the computation of the distances in the reinitialization method presented in the following section. Another advantage of this approach is that the discrete counterparts

$$\Omega_h^- := \Omega_h^-(\varphi_h) = \{ x \in \Omega : I\varphi_h(x, t_n) < 0 \} \tag{3.9}$$

resp.

$$\Omega_h^+ := \Omega_h^+(\varphi_h) = \{ x \in \Omega : I\varphi_h(x, t_n) > 0 \} \tag{3.10}$$

at time $t_n = t_0 + n\Delta t$, $n = 0, \dots, N$, can be easily (re-) constructed.

3.4. **Reinitialization.** We assume that our level set function φ is a signed distance function whose finite element approximation is given by $\varphi_h \in V_h^2$. During the evolution of the level set function φ and its discrete counterpart φ_h in time, the signed distance property may be lost due to variations of $\vec{u}$, discretization errors, insufficient approximation of the curvature and topological changes [23]. In order to regain the signed distance property, a so called reinitialization $\tilde{\varphi}_h$ of the level function φ_h is required whose zero level is (approximately) the same as the zero level of φ_h and which satisfies the condition $||\nabla \tilde{\varphi}_h|| \approx 1$.

There are many reinitialization methods known in literature [10], however, in this article, we cite the variant of [8] of the Fast Marching Method [22], which is only applicable to linear functions.

3.4.1. *Fast Marching Method.* Given a triangulation $\mathcal{S}_h$ and a level set function $\varphi_h \in V_h^2$, we compute the linear interpolation $I\varphi_h$ of φ_h on the regularly refined triangulation $\mathcal{S}_{h/2}$, cf. Section 3.3. Let $\mathcal{D}(S)$ denote the set of degrees of freedom (DOFs) that are given on a simplex $S \in \mathcal{S}_{h/2}$ and $\mathcal{D} := \mathcal{D}(\mathcal{S}_{h/2})$ is the (discrete) set of all DOFs on the discretized domain.

The patch of elements related to a DOF $v \in \mathcal{D}(S),\ S \in \mathcal{S}_{h/2}$ is given by

$$\mathcal{P}(v) := \{S \in \mathcal{S}_{h/2} : v \in \mathcal{D}(S)\} \tag{3.11}$$

and the set of *(direct)* neighbors to $v \in \mathcal{D}(S)$, i.e. all $w \in \mathcal{D}$ that are connected to v via an edge of a d-simplex, is defined by

$$\mathcal{D}_{\mathcal{P}}(v) = \left(\bigcup_{S \in \mathcal{P}(v)} \mathcal{D}(S) \right) \setminus \{v\}. \tag{3.12}$$

Please note that since we have linear basis functions, the degrees of freedom within an element $S \in \mathcal{S}_{h/2}$ can be identified with its vertices. Therefore, we simplify the notation and use both meanings synonymously.

Initialization phase: Firstly, all degrees of freedom resp. vertices of intersected elements are considered, i.e. $v \in \mathcal{D}^\Gamma := \{v \in \mathcal{D}(S) : S \in \mathcal{S}_{h/2}^\Gamma\}$, where $\mathcal{S}_{h/2}^\Gamma$ denotes the d-simplices containing the interface Γ_h, cf. equation (3.7). A geometrical approach and standard linear algebra are used to compute the distance between $v \in \mathcal{D}(S) \subset \mathcal{D}^\Gamma$ and the straight (2D) resp. planar (3D) interface segments $\Gamma_{h,S} = S \cap \Gamma_h$. The computed distances are then used to set the values of a distance function $d\colon \mathcal{D}^\Gamma \to \mathbb{R}$. This function d is now extended to the remaining DOFs resp. vertices $v \in \mathcal{D} \setminus \mathcal{D}^\Gamma$ in an iteration phase.

Iteration phase: We denote the set of *finished* DOFs resp. vertices, i.e. $v \in \mathcal{D}$ for which the function d has already be defined, by $\mathcal{D}_{\mathrm{F}}$. Obviously, $\mathcal{D}_{\mathrm{F}} = \mathcal{D}^\Gamma$ after the initialization phase is completed. The corresponding patch is shown in Fig. 3.2. Now, the set of *active* DOFs resp. vertices is defined by

$$\mathcal{A} := \{v \in \mathcal{D} \setminus \mathcal{D}_{\mathrm{F}} : \mathcal{D}_{\mathcal{P}}(v) \cap \mathcal{D}_{\mathrm{F}} \neq \emptyset\}. \tag{3.13}$$

Using this set, the iteration phase is as follows: For each $v \in \mathcal{A}$, a temporary approximated distance function $\tilde{d}(v)$ is computed by

$$\tilde{d}(v) = \min\{\tilde{d}_S(v) : S \in \mathcal{P}(v) \text{ with } \mathcal{D}(S) \cap \mathcal{D}_F \neq \emptyset\}, \tag{3.14}$$

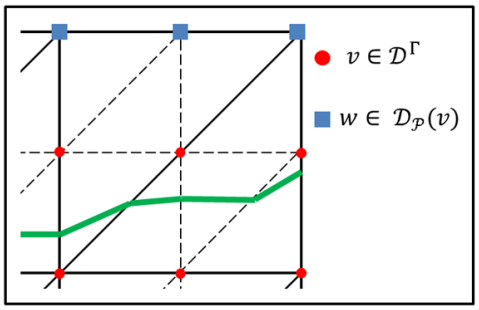


FIGURE 3.2. Sketch of all $v \in \mathcal{D}^\Gamma$ and their neighbors $w \in \mathcal{D}_{\mathcal{P}}(v)$

where the value $\tilde{d}_S(v)$ is calculated by

$$\tilde{d}_S(v) = \begin{cases} d(w) + \|v - w\|, & \text{for } \mathcal{D}(S) \cap \mathcal{D}_F = \{w\}, \\ d(P_W(v)) + \|v - P_W v\|, & \text{for } \mathcal{D}(S) \cap \mathcal{D}_F = \{w_i,\ i = 2 \text{ or } 3\} \end{cases} \tag{3.15}$$

with P_W as the orthogonal projection of v on the line $W := conv(w_1,\ w_2)$ resp. on the triangle $W := conv(w_1,\ w_2,\ w_3)$ for $D(S) \cap \mathcal{D}_F = \{w_1,\ w_2\}$ resp. $D(S) \cap \mathcal{D}_F = \{w_1,\ w_2,\ w_3\}$. The value $d(P_W(v))$ is therefore interpolated from the already calculated values $d(w_i)$ by using the barycentric coordinates of $P_W(v)$ on the line and triangle, respectively.

Afterwards, the DOF resp. vertex $v_0 \in \mathcal{A}$ with

$$\tilde{d}(v_0) = \min_{v \in \mathcal{A}} \tilde{d}(v) \tag{3.16}$$

is added to the finalized set $\mathcal{D}_\mathrm{F}$ and the active set $\mathcal{A}$, the patches $\mathcal{P}(v)$, $v \in \mathcal{D}$ as well as the set of direct neighbors $\mathcal{D}_\mathcal{P}(v)$ are updated accordingly. Furthermore, the temporary function $\tilde{d}(v)$ has to be re-computed for the new setting. This loop is performed until $\mathcal{A} = \emptyset$.

After completing the iteration phase, the (linear) distance function $d(v) \approx |I\varphi_h|$ is defined for all $v \in \mathcal{D}$ and a reinitialized *linear* level set function, approximating a signed distance function on $\mathcal{S}_{h/2}$, is given by $I\tilde{\varphi}_h := d \cdot \mathrm{sign}(I\varphi_h)$. Using the values of the function, a piecewise quadratic function $\tilde{\varphi}_h$ is then defined on $\mathcal{S}_h$, which is our new reinitialized level set function.

In practice, it turns out that this variant of the Fast Marching Method is more stable and leads to better results, if we do not use the patch $\mathcal{P}(v)$ but the extended patch $\mathcal{P}_{\mathrm{ext}}(v)$ considering also all *second* neighbor cells S of v in eq. (3.14) and eq. (3.15).

3.5. **Mass conservation.** For a divergence-free velocity field $\vec{u}(x,t)$, the mass in each subdomain $\Omega^+(t)$ and $\Omega^-(t)$ is conserved for all $t \in [t_0, t_f]$ from an analytical point of view. However, this is not necessarily true for the discretized subdomains $\Omega_h^-(t_n)$ resp. $\Omega_h^+(t_n)$, $t_n = t_0 + n\Delta t$, $n = 0, \ldots, N$, that are obtained by solving the discretized level set problem in stabilized form using the discrete representation of Γ_h, cf. Section 3.3.

As analyzed in [20], the loss of mass will decrease only with decreasing mesh size h and for smaller time steps Δt since φ_h and resp. $I\varphi_h$ converges to φ. Since reinitialization methods as the one presented 3.4 are also not mass conserving, an approach to enforce this property is advisable. For simplicity, we here assume that no phase transitions occur and that the subdomains separated by the level set function do not mix. Thus, the mass of each phase should be constant. Moreover, by using the notation old resp. new, we indicate that $I\varphi_h^{\mathrm{old}}$ and $I\varphi_h^{\mathrm{new}}$ can be the level set function at different time steps or the functions before and after reinitialization. Due to this, the time as argument is dropped as in the sections before.

We adapt two mass conserving strategies, which are based on a global and a local consideration of elements. Both mass correction methods take advantage of the signed distance property of the level set function by shifting the zero level set using a function $\psi_h(\cdot, t) \in V_{h/2}^1$ such that[1]

$$\Delta V_h^- = V_h^-(I\varphi_h^{\mathrm{old}}) - V_h^-(I\varphi_h) = 0 \tag{3.17}$$

holds, with

$$I\varphi_h := I\varphi_h^{\mathrm{new}} + \psi_h \tag{3.18}$$

[1]Instead of $V_h^-(I\varphi_h^{\mathrm{old}})$ we could also write $V_h^-(I\varphi(\cdot, t_0)$ due to our simplifying assumptions that no phase transitions occurs and the subdomains do not mix.

and $V_h^-(\phi)$ is defined for a $\phi_h \in V_{h/2}^k$ by

$$V_h^-(\phi_h) := V_h^-(\phi_h(\cdot,t)) = \int_{\{x\in\Omega:\phi_h(x,t)<0\}} 1dV, \quad t \in [t_0, t_f]. \tag{3.19}$$

Please note that if (3.17) is true, the same applies to ΔV_h^+ since we have

$$V_h^+(I\varphi_h) = V_h - V_h^-(I\varphi_h). \tag{3.20}$$

Equation (3.17) describes a non-linear problem which has to be solved by an iterative approach. From a numerical point of view, it is important to compute the solution using as few iteration steps and function evaluations as possible. In this paper, we use the Anderson/Björck variant [3] of the regula falsi method, see Section 4.3.3 for details.

In the following paragraphs, we describe the mass correction methods for the reinitialization step. Therefore, we can omit the time variable. Please note that correcting the mass defect resulting from time evolution (and without reinitialization) can easily be adapted.

3.5.1. *Global approach for mass conservation.* A simple approach for correcting the mass defect is based on using a globally constant function $\psi_h := \psi_h^{\text{const}} \in \mathcal{P}_0(\Omega_h)$, $\psi_h^{\text{const}}(x) = \epsilon,\ \forall x \in \Omega_h$ in (3.17). Thereby, the value ϵ can be obtained by finding the root of the non-linear equation

$$Z(\epsilon) := V_h^-(I\varphi_h^{\text{old}}(\cdot)) - V_h^-(I\varphi_h^{\text{new}}(\cdot) + \epsilon) = 0 \tag{3.21}$$

by using the previously mentioned regula falsi method.

Due to the fact that the level set function is shifted globally by ϵ, the corresponding function ψ_h^{const} can be added to $I\varphi^{\text{new}}$. Furthermore, this method is independent from a reinitialization process and does not alter the gradient. Therefore, if using a reinitialization technique, the mass correction can be computed after the level set function is reinitialized.

3.5.2. *Local approach for mass conservation.* A more complex method to conserve mass in the level set method, also utilizing the signed distance property, is adapted from [16]. In contrast to the global method, this approach is based on considering the mass defects on all elements $S \in \mathcal{S}_{h/2}^\Gamma$, i.e. all elements intersected by Γ_h, individually. For these elements, the non-linear equation

$$Z_S(\epsilon_S) : = V_{h,S}^-(I\varphi_h^{\text{old}}(\cdot)) - V_{h,S}^-(I\varphi_h^{\text{new}}(\cdot) + \epsilon_S) = 0, \tag{3.22}$$

with

$$V_{h,S}^-(I\varphi_h) = \int_{\Omega_h^-\cap S} 1dV, \tag{3.23}$$

is solved using the same Anderson/Björck variant of the regula falsi method as in the global mass conservation approach. Using the values $\epsilon_S \in \mathbb{R}$, a piecewise constant function $\tilde{\psi}_h \in \mathcal{P}_0(S),\ S \in \mathcal{S}_{h/2}^\Gamma$, that is discontinuous across element boundaries is defined by $\tilde{\psi}_h(S) = \epsilon_S$.

Since $\tilde{\psi}_h(S)$ is a discontinuous function, we cannot add $\tilde{\psi}_h$ to the level set function $I\varphi_h^{\text{new}}$. Instead, we first define a function $\tilde{\tilde{\psi}}_h \in V_{h/2}^1$, cf. (3.2), that is piecewise linear on $S \in \mathcal{S}_{h/2}^\Gamma$ and continuous on Ω_h. For this purpose, we compute the average μ of the correction value ϵ_S of each DOF $v \in \mathcal{D}^\Gamma$ by

$$\mu(v) = \frac{1}{|S \in \mathcal{P}^\Gamma(v)|} \sum_{S\in\mathcal{P}^\Gamma(v)} \epsilon_S, \tag{3.24}$$

where $|S \in \mathcal{P}^\Gamma(v)|$ is the number of elements of the patch

$$\mathcal{P}^\Gamma(v) := \{S \in \mathcal{S}^\Gamma_{h/2} : v \in \mathcal{D}^\Gamma(S)\},$$

and define

$$\tilde{\tilde{\psi}}_h(v) = \begin{cases} \mu(v), & \text{for } v \in \mathcal{D}^\Gamma \\ 0, & \text{for } v \notin \mathcal{D}^\Gamma \end{cases}. \tag{3.25}$$

Now, we solve the non-linear problem for a constant $C \in \mathbb{R}$ such that

$$Z(C) : = V_h^-(I\varphi_h^{\text{old}}(\cdot)) - V_h^-(I\varphi_h^{\text{new}}(\cdot) + C \cdot \tilde{\tilde{\psi}}_h) = 0, \tag{3.26}$$

by using the regula falsi method again and, finally, define the correction function

$$\psi_h = C \cdot \tilde{\tilde{\psi}}_h, \tag{3.27}$$

so that $I\varphi_h = I\varphi_h^{\text{new}} + \psi_h$ is the new, mass conserving, level set function.

Similar to our observations in regards to the reinitialization method, the local mass correction approach is more stable, if the average $\mu(v)$ is computed considering more elements. Therefore, we use the extended patch $\mathcal{P}^\Gamma_{\text{ext}}(v)$ instead of $\mathcal{P}^\Gamma(v)$ in eq. (3.24).

Note, that (3.24) alters the gradient of φ_h and we may have $||\nabla\varphi_h|| \not\approx 1$. Consequently, this mass conservation method should be integrated into the reinitialization, more precisely, between initialization and iteration phase so that instead of $\hat{d}$ we use $\hat{d}_\epsilon$ in the iteration phase.

4. Implementation

The previously discussed methods for solving level set problems with or without using the streamline diffusion stabilization technique numerically have been implemented as a toolbox into the `FEniCS` framework [14].

4.1. **FEniCS.**

The `FEniCS`-Project is a collaborative project of researchers who develop tools for automated scientific computing, especially in the field of finite element methods for the solution of partial differential equations [14]. It consists of a collection of core components such as

(1) the Unified Form Language `UFL` [2], which is a domain-specific language to specify finite element discretizations of differential equations using variational formulations.
(2) the `FEniCS` Form Compiler `FFC` [12, 18], which analyzes given `UFL` code and, in combination with `Instant` and `FIAT` [11], generates `UFC` [1] code for arbitrary finite elements on simplices based on the variational forms specified in the `UFL` file
(3) `DOLFIN` [15], the main problem solving environment and user interface whose functionality integrates the other `FEniCS` components and handles communication with external libraries.

There are various articles describing the different modules and extensions of `FEniCS`, e.g. `Unicorn` [9], a massively parallel adaptive finite element solver for problems in the field of fluid and structure mechanics, or `dolfin-adjoint` [7], that facilitate the automated development of reliable adjoint models which can be used in optimization methods for problems with PDE constraints.

4.2. **Design principles.**

The level set toolbox consists of different form files and classes that are implemented into the `FEniCS` framework. Basically the implementation can be divided into three parts, cf. Fig. 4.1.

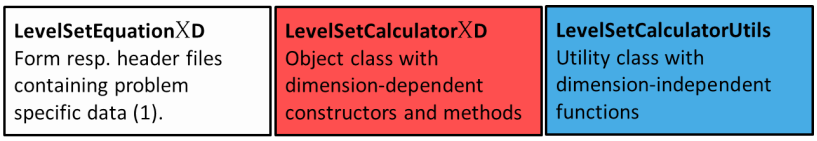


FIGURE 4.1. Implementation structure: Form (header) files, object classes and utility functions

Form files. Using the `UFL` syntax, the stabilized level set problem[2], cf. eq. (3.5), is formulated in the files `LevelSetEquation2D.ufl` and `LevelSetEquation3D.ufl` for two resp. three dimensions. After running the `FFC` compiler on these files, the corresponding C++ header files are automatically generated, containing most of the problem specific data and therefore being the basis to solving the transport equation.

Object class. The classes `LevelSetCalculator2D` and `LevelSetCalculator3D` contain dimension depending constructors that create the object and routines for solving the transport equation named `updateLevelSetFunction`. Therefore, the previously mentioned Form files are included providing the level set specific background, e.g. the bilinear form a and the linear form L. Furthermore, the basics of the re-distancing method are implemented here, which rely mostly on dimension independent algorithms that are specified in the `LevelSetCalculatorUtils`. Due to their dimension dependence, the reinitialization and mass conservation are integrated in the update-routine. Parameters like the frequency of reinitializing the level set function are passed to the methods using a list. The default variant is to perform a reinitialization step and the local mass conservation procedure after every time step.

Utility class. Aside from the automatically generated problem header files, most implementation aspects are covered in the utility class `LevelSetCalculatorUtils` which consists of various static methods, i.e. methods that do not belonging to any object. In detail, this class contains a routine implementing the iteration phase of the reinitialization approach, mass conservation methods and some helper functions. In this paragraph, we want to present the helper methods briefly:

`hasSignSwitchOnCell:` A method to check whether the zero level set Γ_h intersects the cell. It can be easily done by checking for sign switches of the level set DOFs corresponding to the cell.
`build_IntersectedNeighborPatch:` A method to build the patch consisting of the first and second neighbors of any cell (input parameter), which are intersected by Γ_h.
`find_PlanarLevelsetSegment:` This method is used to calculate the intersection points given by the zero level set Γ_h and the edges of any given cell (input data).
`determine_NodeToLineDistance:` This is used to calculate distances between a given point and a finite line, which is given by the endpoints of the line. Additionally it stores the barycentric coordinates to the orthogonal projection of the vertex on the line.
`determine_NodeToTriangleDistance:` Same as above, only with a triangle given by three corner points instead of the finite line.

4.3. **Level set toolbox.** For using the level set toolbox, the user has to create an object of the type `LevelSetCalculator2D` or `LevelSetCalculator3D` in his

[2]Note that by setting $\delta = 0$, the unstabilized problem can be reconstructed.

Algorithm 1 Using the Levelset Toolbox

```
[...] // Initialization etc.
// Creating and extending the parameters structure: Since all methods are
// hidden within the object class, the reinitialization frequency, the volume
// correction method etc. are added and defined within the parameters structure.
dolfin::Parameters parameters; parameters.add("foo", foo);
// Creating an object of type LevelSetCalculatorXD
LevelSetCalculatorXD lc(mesh, u(x,t), φ_h(t_0), parameters);
// Update LevelSetCalculatorXD lc object based on the specified parameters.
lc.updateLevelSetFunction();
[...]
```

code, depending on the geometrical dimension. This is done by calling the class constructor with the input data

(1) a triangulation $\mathcal{S}_h$,
(2) the (initial) velocity field $\vec{u}(x,t)$,
(3) an initial function $\varphi_h \in V_h^2$,
(4) a parameter list including e.g.. the time stepping scheme, the stabilization parameter, and the reinitialization method as well as its frequency
(5) optional: an influx boundary $\partial\Omega_{\text{in}}(t)$ and its boundary condition,

as shown in Algorithm 1.

4.3.1. *Solving the level set equation.* After creating the object, the next evolution step of the level set function resp. the surface can be computed by calling the `updateLevelSetFunction` member function without any input. The routine uses the data provided by the object and solves the level set problem with `DOLFIN`-methods. Due to the fact that the level set function is given by a reference to the calculator object, the user automatically gets access to the updated level set function.

4.3.2. *Reinitialization of the level set function.* When creating an object of the type `LevelSetCalculator2D` resp. `LevelSetCalculator3D`, the triangulation $\mathcal{S}_h$ is regularly refined and the piecewise quadratic level set function φ_h is linearly interpolated by $I\varphi_h$ on the generated triangulation $\mathcal{S}_{h/2}$. Note that the degrees of freedom of the linear approximation $I\varphi_h$ coincide with the DOFs of φ_h on $\mathcal{S}_h^\Gamma$ which allows for a fast and efficient interpolation.

For $I\varphi_h$, the reinitialization method described in Section 3.4 is separated into an initialization phase and an iteration phase[3]. Basically, the implementation of the reinitialization method uses three containers:

(1) $\mathcal{D}_\Gamma$, the set of DOFs belonging to elements $S \in \mathcal{S}_{h/2}$ intersected by the interface Γ_h. In the code $\mathcal{D}_\Gamma$ is realized as a vector of booleans.
(2) $\mathcal{D}_F$, the set of DOFs whose distance to Γ_h have already been computed. Like $\mathcal{D}_\Gamma$ it is realized as a vector of booleans.
(3) $\mathcal{A}$, a set of active DOFs for which a temporary distance $\tilde{d}$ is computed.

For a practical point of view, one has to consider that the initialization phase depends heavily on the geometrical dimension and, consequently, is implemented as a private function within in the object class. The principal algorithm for the 3D situation is shown in Algorithm 2. In contrast to the initialization phase, the iteration phase can be implemented almost independently from the dimension in the utility class.

[3]If a local mass conservation algorithm shall be applied, this would be called after the initialization phase with the $d(v)$-values as an input, c.f. 4.3.3.

Algorithm 2 Fast marching method: initialization phase (3D)

Require: Refined mesh $\mathcal{S}_{h/2}$, linear level set function $I\varphi_h$.
Set up containers $\mathcal{D}_F = \mathcal{D}^\Gamma$ (false as default) and d (-1 as default) for the finished DOFs and the d-values.

Initialize the set $\mathcal{D}_F$ of finished DOFs:
for $S \in \mathcal{S}_{h/2}$ **do**
  if `hasSignSwitchOnCell`(S) == TRUE **then**
    for $v \in \mathcal{D}(S)$ **do**
      Mark DOF v as TRUE in $\mathcal{D}_F$.

Calculate the distance to the (linear) zero level set Γ_h for every $v \in \mathcal{D}_F$:
for $v \in \mathcal{D}$ **do**
  if $\mathcal{D}_F(v)$ == TRUE **then**
    Build the intersected neighbor patch and store the cells in $\mathcal{N}$ (by using `build_IntersectedNeighborPatch`).
    while $\mathcal{N} \neq$ **do**
      Take the first cell from $\mathcal{N}$ and store it to S. Calculate the intersected zero level set $\Gamma_h(S)$ and store it into a vector of points called $\mathcal{V}$.
      if $|\mathcal{V}| == 2$ **then**
        Use `determine_NodeToLineDistance` to calculate the distance of v to the line given by the two nodes and store it in d_{temp}.
      if $|\mathcal{V}| == 3$ **then**
        Use determine_NodeToTriangleDistance to calculate the distance of v to the line given by the two nodes and store it in d_{temp}.
      if $|\mathcal{V}| == 4$ **then**
        Separate the quadrilateral into two triangles and calculate the distance to both. Set the smaller one as d_{temp}.
      if $d(v) = -1$ (default value) or $d_{\text{temp}} < d(v)$ **then**
        $d(v) = d_{\text{temp}}$.
      Erase S from the `patch` $\mathcal{N}$.

return $d(v)$ for every $v \in \mathcal{D}^\Gamma(S),\ S \in \mathcal{S}^\Gamma_{h/2}$.

For the computation of the temporary values $\tilde{d}(v)$, the method called `calculateDistanceMethod` is implemented as shown in Algorithm 3. However, to account for the permanently changing values $\tilde{d}$ in an efficient way, we save the DOFs, which are neighbors of the current processed DOF v and only recalculate the $\tilde{d}$-value for this DOFs. Every other $\tilde{d}$-value cannot be influenced by adding v to the finished set. The re-computation of the whole active set increases the calculation time a lot. Additionally, we want to point out, that a c++ `multimap` with a double as the key and a unsigned int as the value is a good way to implement an ordered active set $\mathcal{A}$ because the DOF with minimal $\tilde{d}$-value is always at position 0. The complete iteration phase is presented as pseudo code in Algorithm 4.

4.3.3. *Volume/mass conservation.* The mass correction methods presented in this paper take advantage of the signed distance property and are based on the computation of the volume of patches of elements. As mentioned in Section 3.5, it is important to use an efficient method to compute the quantities ϵ in eq. (3.21) resp. ϵ_S in (3.22). For both methods this is done by using an advanced regula falsi algorithm that is presented in [3]. The idea of the method is given in Algorithm 5.

Except for this method, the most important routines used in the implementation of the mass correction methods are `calculateLevelSetDifference` and

Algorithm 3 Fast marching method: Algorithm to calculate $\tilde{d}(v)$

Require: A DOF v, the finished set $\mathcal{D}_F$ and $d(v)$ for $v \in \mathcal{D}_F$.
Create a container for the return value $r = -1$.

for $S \in \mathcal{P}_{\text{ext}}(v)$ **do**

Initialize a temporary container $\mathcal{W}$ representing $\mathcal{D}(S) \cap \mathcal{D}_F$.
for $\tilde{v}$ in $\mathcal{D}(S)$ **do**
if $\mathcal{D}_F(\tilde{v}) ==$ TRUE **then**
Save the DOF to $\mathcal{W}$.

if $\mathcal{W} = \{w_1\}$ (contains one DOF) **then**
Save a temporary value $d_{temp} = d(w_1) + \|v - w_1\|$.
if $\mathcal{W} = \{w_1,\ w_2\}$ (contains two DOFs) **then**
Calculate the distance $dist(v, \mathcal{W})$ of v to the line given by w_1 and w_2 and the barycentric coordinates of the projection by using `determine_NodeToLineDistance`. Save a temporary value $d_{temp} = d(P_W(v)) + dist(v, \mathcal{W})$.
if $\mathcal{W} = \{w_1,\ w_2,\ w_3\}$ (contains three DOFs) **then**
Calculate the distance $dist(v, \mathcal{W})$ of v to the triangle given by w_1, w_2 and w_2 and the barycentric coordinates of the projection by using `determine_NodeToTriangleDistance`. Save a temporary value $d_{temp} = d(P_W(v)) + dist(v, \mathcal{W})$.

if $d_{temp} < r$ or $r = -1$ (default value) **then**
Set $r = d_{temp}$.

return The $\tilde{d}(v)$ value given by the return value r.

`calculatePatchVolume`. The routine `calculateLevelSetDifference` is used to compute the value of the objective functional Z resp. Z_S. It relies heavily on the method `calculatePatchVolume` that computes the volume $V_h^- \cap \mathcal{P}$ of a patch of elements. This patch can be an arbitrary collection of simplices so that the method can be used for both mass conserving methods. Both routines are implemented in the utility class `LevelSetCalculatorUtils`.

Within the local mass correction approach, computing the correction value ϵ_S on a simplex using the presented regula falsi method may result in numerical issues, if the zero level set is very close to a DOF. Hence, we ignore the adjustment and set ϵ_S to zero in these cases. Please note that different approaches, e.g. transforming the search for ϵ_S into an optimization problem and using a very robust optimizer like the Nelder-Mead algorithm [17], could solve this problem making the algorithm much more stable. However, these approaches are much more expensive from a numerical point of view.

5. Results

The level set toolbox is tested by computing the numerical solutions for different examples in both, two and three dimensions. For analyzing the quality of the solutions, the following errors are defined:

(1) The standard L^2-error between two functions

$$e_{L^2}(f, g) = \|f - g\|_{L^2(\Omega)} = \sqrt{\int_\Omega |f - g|^2 \, dx}, \tag{5.1}$$

Algorithm 4 Fast marching method: iteration phase

Require: The finished set $\mathcal{D}_f$ after the initialization phase and the corresponding d-values.

Initialize the active set $\mathcal{A}$:
for every DOF v of the mesh **do**
  if $\mathcal{D}_F(v)$ is true **then**
    for every DOF n directly connected to v **do**
      Calculate $\tilde{d}(n)$ by calling `calculateDistanceFunction` and save it ordered to the active set.

while $\mathcal{A}$ is not empty **do**
  Store the first DOF of $\mathcal{A}$ in v and erase it from $\mathcal{A}$.
  Set $\mathcal{D}_F(v)$ to true and $d(v)$ to $\tilde{d}(v)$.
  Create a boolean container $\mathcal{N}$ with false as default to save direct neighbors of v.

  for every direct neighbor n of v **do**
    if n is not in $\mathcal{A}$ nor in $\mathcal{D}_F$ **then**
      Save n to $\mathcal{A}$ and with respect to the correct order by calling `calculateDistanceFunction`.
    else if n is not in $\mathcal{D}_F$ **then**
      Mark n by adding it to $\mathcal{N}$ for the calculation of a possibly new $\tilde{d}$-value.
  for every DOF n of the mesh **do**
    if $n \in \mathcal{N}$ **then**
      Recalculate $\tilde{d}(n)$ by calling `calculateDistanceFunction`.
      if $\tilde{d}(n)$ has changed **then**
        Erase n from $\mathcal{A}$ and reinsert it to restore the right order.

for every DOF v of the mesh **do**
  Set the linearized level set DOF according to $I\varphi_h(v) = \operatorname{sign}(\varphi_h(v))d(v)$.

Interpolate the linearized level set function $I\varphi_h$ by the quadratic level set function φ_h by using the one-to-one DOF relation.
return A reinitialized level set function φ_h

Algorithm 5 Regula falsi method in the Anderson/Björck variant

Require: The objective function f, left and right initial values z_1 and z_2.
repeat
  Calculate $f_i = f(z_i),\ i \in \{1,\ 2\}$.
  Calculate $z = z_1 - f_1 \cdot (f_2 - f_1)/(z_2 - z_1)$ and $f_z = f(z)$.
  if $f_z \cdot f_2 < 0$ **then**
    Set $z_1 = z_2,\ f_1 = f_2,\ z_2 = z$ and $f_2 = f_z$.
  else
    Calculate m according to

$$m = \begin{cases} 1 - f_z/f_2 & \text{if } 1 - f_z/f_2 \geq 0, \\ 1/2 & \text{else.} \end{cases}$$

    Set $f_1 = mf_1,\ z_2 = z,\ f_2 = f_z$ and $z_1 = z_1$.
until $|z_1 - z_2| < \mathrm{Tol}_z$ and $|f_z| < \mathrm{Tol}_{f_z}$.
return An approximation z to the root of f.

(2) the relative volume error between two functions

$$e_{\mathrm{Vol}}(f, g) = \frac{|V^-(f) - V^-(g)|}{V^-(f)}, \tag{5.2}$$

with

$$V^-(f) = \int_{\{x\in\Omega: f(x,t_n)<0\}} 1dx.$$

(3) the maximum euclidean distance error between the zero level set Γ_g of a function g to the nearest point of the zero level set Γ_f of a function f

$$e_\infty(f,g) = \max_{x_2\in\Gamma_g} \min_{x_1\in\Gamma_f} \|x_1 - x_2\|_2. \tag{5.3}$$

In the following, the function f will always represent the initial signed distance function to a certain shape $\Omega(0)$ or a reference solution, respectively, while g will always be our numerical computed solution φ_h at $t = t_f$. Thereby, it will be mentioned, if reinitialization and mass conservation methods are used within the computation. Please note that all examples are chosen so that $\varphi_h(\cdot, t_f)$ should coincide with the original zero level set at $t = t_0$.

5.1. **2D-Example: Deformation flow.** On $\Omega = [0,1]^2$ consider a circle centered at $(0.5, 0.75)$ with a radius of 0.15. The initial level set Γ at $t = 0$ is given by the signed distance function φ_0 to the circle

$$d(x,y) = \sqrt{(x-0.5)^2 + (y-0.75)^2} - 0.15 \tag{5.4}$$

and a time dependent velocity field $u(t,x,y)$ is given by

$$u(t,x,y) = \begin{pmatrix} -\sin^2(\pi x)\sin(2\pi y)\cos(\pi t/t_f) \\ \sin(2\pi x)\sin^2(\pi y)\cos(\pi t/t_f) \end{pmatrix}, \tag{5.5}$$

with $t \in [0,2]$. The characteristic of this example, which has been published by [13] and is also considered as a benchmark example in [4], is that during the period $0 < t < 1$, the circle is deformed and stretched while for $1 < t < 2$ a reversal phase takes place so that at $t = 2$, the initial shape of a circle is recovered, see Fig. 5.1

Time stepping. Our first test addresses the error introduced by the time discretization. Here, we consider the implicit Euler scheme, which means $\theta = 1$ in eq. (3.4), and the Crank-Nicolson scheme corresponding to $\theta = 0.5$.

Since we are only interested in errors resulting from time discretization, a coarse uniform mesh with $2 \times (10 \times 10)$ elements is chosen. The stabilization parameter δ_T is time dependent and computed with (3.6) and $c = 0.1$. Apart from that, no other method, e.g. reinitialization or mass conservation methods are used during this test.

Please note that if no stability, reinitialization and mass conservation methods are used, the numerical solution computed using the Crank-Nicolson time discretization scheme will be the interpolation of d onto the space V_h^2 for $t = 2$ due to the symmetry of the example. cf. [8]. This analytical result is reproduced if solving

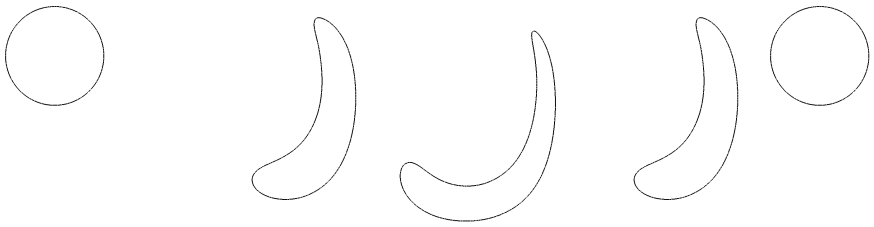


FIGURE 5.1. 2D Example deformation flow: Reference solution $\varphi_h(x,t)$ with $\varphi_h(x,0) = \varphi_h(x,2)$ at $t = 0,\ t = 0.5,\ t = 1,\ t = 1.5$ and $t = 2$.

Δt	Implicit Euler ($\theta = 1$)	Crank-Nicolson ($\theta = 0.5$)
$2^0/10$	$3.25e-2$	$6.10e-3$
$2^{-1}/10$	$1.86e-2$	$1.54e-3$
$2^{-2}/10$	$1.01e-2$	$3.87e-4$
$2^{-3}/10$	$5.36e-3$	$9.68e-5$
$2^{-4}/10$	$2.71e-3$	$2.42e-5$
$2^{-5}/10$	$1.32e-3$	$5.99e-6$
$2^{-6}/10$	$5.92e-4$	$1.45e-6$

TABLE 1. L_2-errors $\|d^{ref}(\frac{1}{2}t_f) - \varphi_h^{\text{ie}}(\frac{1}{2}t_f)\|_{L^2(\Omega_h)}$ and $\|d^{ref}(\frac{1}{2}t_f) - \varphi_h^{\text{cn}}(\frac{1}{2}t_f)\|_{L^2(\Omega_h)}$ for different time step sizes on a mesh consisting of $2 \times 10 \times 10 = 200$ elements.

the example with our level set toolbox [4]. Therefore, we use the Crank-Nicolson time discretization scheme without stabilization, reinitialization or mass conservation methods to compute the reference solution using a very small time step size of $\Delta t = 2^{-5}/100$. This reference solution $d^{ref}(x,t)$ is then used for the computation of the L_2-errors and the analysis of the convergence behavior.

The results in Table 1 show that the solution computed with the toolbox converges as one would expect, i.e. it converges linearly, if the implicit Euler scheme is used, and quadratically, if applying the Crank-Nicolson scheme to the problem. Additionally, it can be noted that for a fixed time step size, the overall error when using the Crank-Nicolson scheme is smaller compared to the implicit Euler discretization.

Stabilization. As mentioned in Section 3.2, the level set problem often has to be stabilized. On two uniformly constructed meshes with $2 \times 40 \times 40 = 3200$ and $2 \times 80 \times 80 = 12800$ elements consider problem (3.5) with $\theta = 1$, i.e. implicit Euler time discretization[5]. Now, we compare the differences in the L_2 norm at $t = t_f$ between our initial signed distance function $\varphi_h(0) = d$ and the numerical solutions without applying stabilization $\varphi_h^0(t_f)$ and with factor $c = 0.5$, i.e. $\varphi_h^{0.5}(t_f)$.

As for the constant δ_0 of (3.6), we choose $\delta_0 = h_S$, which is globally constant given by $\sqrt{2}/40$ resp. $\sqrt{2}/80$ for the used meshes. Aside from this, no additional algorithms like reinitialization or mass conservation are applied for this scenario.

Examining the results which are given in Table 2, one can see that the influence of an applied stabilization parameter is very small in this scenario[6]. Therefore, we do not use this technique for the other examples. However, please note that the influence of the stabilization parameter depends highly on the velocity field and the example.

Reinitialization and mass conservation. The most important aspects of the level set toolbox are reinitialization and mass conservation. As previously described, the numerical solution of the level set problem does not preserve the signed distance property and, moreover, lack mass conservation, especially if reinitialization techniques are used. Additionally it has to be noted that the L_2-error of a reinitialized level set function does not converge to zero for finer meshes. This is due to the

[4]The L_2-error between the interpolation of d onto V_h^2 and the calculated $\varphi_h(t = 2)$ using the the Crank-Nicolson scheme is within the range of computational accuracy, i.e. smaller than 10^{-15}.

[5]As mentioned, using the Crank-Nicolson time discretization scheme would only be reasonable in this scenario for $t \in [t_0, \frac{t_f}{2}]$.

[6]We tested varying the constant $c \in [0, 2]$ on several mesh sizes and time step sizes with similar results

Δt	No stability ($c = 0$)		With stability ($c = 0.5$)	
	$2 \times 40 \times 40$	$2 \times 80 \times 80$	$2 \times 40 \times 40$	$2 \times 80 \times 80$
0.1	$5.02e-2$	$4.13e-2$	$5.02e-2$	$5.02e-2$
0.05	$3.21e-2$	$3.21e-2$	$3.21e-2$	$3.21e-2$
0.025	$1.91e-2$	$1.91e-2$	$1.91e-2$	$1.91e-2$
0.01	$9.09e-3$	$9.09e-3$	$9.11e-3$	$9.09e-3$
0.005	$5.05e-3$	$5.05e-3$	$5.10e-3$	$5.05e-3$
0.0025	$2.76e-3$	$2.76e-3$	$2.87e-3$	$2.77e-3$

TABLE 2. Errors $\|\varphi_h(0) - \varphi_h^0(t_f)\|_{L_2}$ (without stabilization) and $\|\varphi_h(0) - \varphi_h^{0.5}(t_f)\|_{L_2}$ (with stabilization) with respect to different time step and mesh sizes.

fact that the fast marching method only guarantees the property $\|\nabla\varphi_h\| \approx 1$ in a close range to the zero level set while the reinitialized function differs from the real signed distance function d in the far field. Therefore, we compare the e_{vol} and e_∞ errors in this section instead of the difference in the L_2 norm.

On different meshes with diam $h = \sqrt{2}/32, \ldots, \sqrt{2}/512$, we use the Crank-Nicolson scheme without stabilization, i.e. means $\theta = 0.5$ and $\delta_S = 0$. The time step size is $\Delta t = 0.01$ and we will compare the situations where

(1) only reinitialization ("R"),
(2) reinitialization with global mass conservation (called "RGM"), and
(3) reinitialization with local mass conservation (called "RLM")

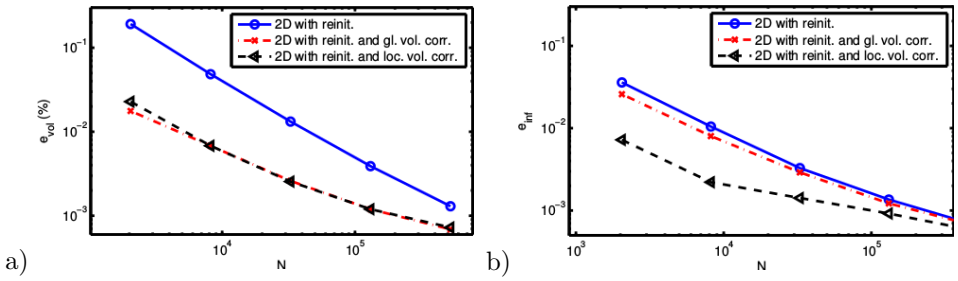
methods are used after every time step, which means that the respective method is used in every single computation step. All results are presented in Table 3 and visualized in Figure 5.2. Exemplary results of all methods on meshes with $2 \times 32 \times 32 = 2048$ and $2 \times 64 \times 64 = 8192$ elements are given in Figure 5.3.

According to this test scenario, only using a reinitialization technique without correcting the mass defect gives the worst results, the case where reinitialization and local mass conservation methods are used gives in contrast the smallest errors. If instead of local mass conservation we employ the global strategy, the volume errors are still very small. However, the drawback of this rather simple approach is that the shape of the zero level set Γ_h is also conserved, i.e. the zero level set does not change its shape as it should do according to the velocity field. This leads to a higher e_∞ error.

In brief, our main conclusions regarding reinitialization and mass correction are: Firstly with applied reinitialization, mass conservation should be used as well because the reinitialization has a negative effect on the zero level set reconstruction.

$\sqrt{2}/h$	R		RGM		RLM	
	$e_{\text{vol}}[\%]$	e_∞	$e_{\text{vol}}[\%]$	e_∞	$e_{\text{vol}}[\%]$	e_∞
32	19.14	$3.60e-2$	1.77	$2.59e-2$	2.28	$7.22e-3$
64	4.85	$1.05e-2$	0.68	$8.02e-3$	0.68	$2.21e-3$
128	1.32	$3.28e-3$	0.26	$2.91e-3$	0.255	$1.42e-3$
256	0.39	$1.36e-3$	0.12	$1.22e-3$	0.12	$9.2e-4$
512	0.13	$6.55e-4$	$6.89e-2$	$6.41e-4$	$7.26e-4$	$5.60e-4$

TABLE 3. Errors $e_{\text{vol}}(d, \varphi_h(t_f))$ and $e_\infty(d, \varphi_h(t_f))$ of the reinitialization and mass conservation algorithms with respect to h.



a) b)

FIGURE 5.2. Errors $e_{\text{vol}}(d, \varphi_h(t_f))$ (a) and $e_\infty(d, \varphi_h(t_f))$ (b) of the reinitialization and mass conservation algorithms with respect to the number of elements N.

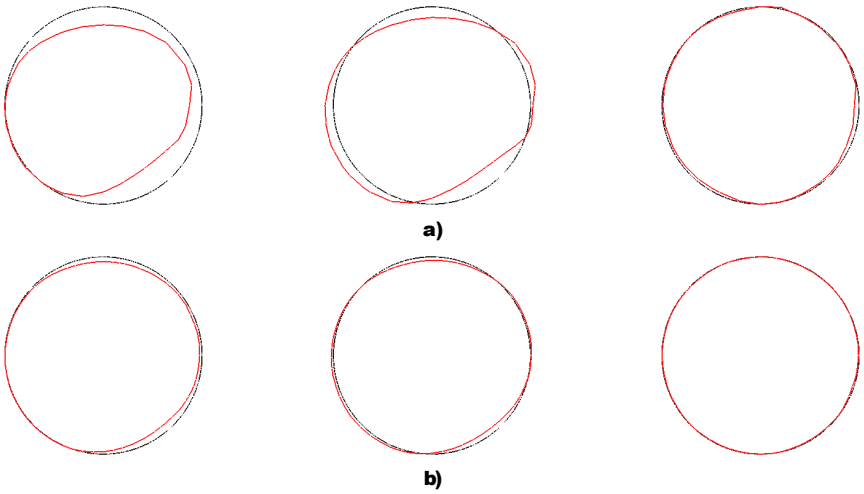


FIGURE 5.3. Method comparison for meshes with size a) $\sqrt{2}/32$ and b) $\sqrt{2}/64$ for methods "only reinitialization(R)", "reinit. and global mass correction (RGM)" and "reinit and local mass correction (RLM)" from left to right.

Secondly it points out the main problem of the global mass conservation, i.e. the distortion of the zero level. With applied local mass conservation, this effect can be reduced, especially if a coarse mesh is used.

5.2. **2D-Example: Deforming droplet.** The second example considered as a test scenario describes the movement and deformation of a circle and is adapted from [8]. For $t \in [0, 20]$ let $\Omega = [0, 1]^2$ be our domain and φ_0 be the zero level set of a circle with midpoint $(0.5, 0.25)$ and radius 0.15. A velocity field is given by

$$u(t, x, y) = \begin{cases} c(x, y) \cdot (\tilde{y}, -\tilde{x}), & \text{for } t \leq \frac{1}{2} t_f, \\ -c(x, y) \cdot (\tilde{y}, -\tilde{x}), & \text{for } t > \frac{1}{2} t_f, \end{cases} \tag{5.6}$$

with

$$c(x, y) = \begin{cases} 4||(\tilde{x}, \tilde{y})|| \ (0.5 - ||(\tilde{x}, \tilde{y})||), & \text{for } ||(\tilde{x}, \tilde{y})|| \leq 0.5, \\ 0, & \text{otherwise}, \end{cases}$$

and $(\tilde{x}, \tilde{y}) = (x - 0.5, y - 0.5)$. Figure 5.4 shows sketches the process.

In contrast to the extensive study of the previous example, we only analyze the convergence behavior of the global and the local mass corrections method for N

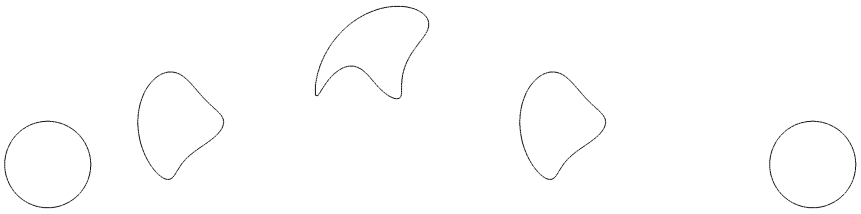


FIGURE 5.4. 2D Example rising deforming droplet: Reference solution $\varphi_h(x,t)$ with $\varphi_h(x,0) = \varphi_h(x,20)$ at $t = 0,\ t = 5,\ t = 10,$ $t = 15$ and $t = 20$.

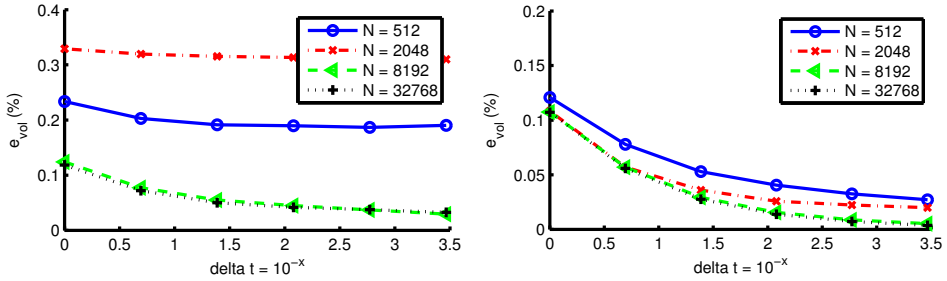


FIGURE 5.5. Errors $e_{\text{vol}}(d, \varphi_h(t_f))$ the global (left) and local (right) mass conservation algorithms.

varied between $2\times2^4\times2^4$ and $2\times2^7\times2^7$ and $\Delta t = 2^0, 2^{-1}, ..., 2^{-5}$. The convergence behavior of the L_2-errors are visualized is Figure 5.5.

As one can see, the error depends mainly on the spatial discretization. This is to be expected since reinitialization and mass correction is applied after every time step. Furthermore it can be observed that the error if using the local mass correction scheme is much smaller compared to the global mass conservation method.

5.3. **3D-Example: Deformation flow.** Now, we enhance the example 5.1 and choose a domain $\Omega = [0,1]^3$ containing a sphere centered at $(0.35, 0.35, 0.35)$ with a radius of 0.15. The initial condition φ_0 is given by

$$d(x,y,z) = \sqrt{(x-0.35)^2 + (y-0.35)^2 + (z-0.35)^2} - 0.15. \tag{5.7}$$

and the prescribed velocity field is given by

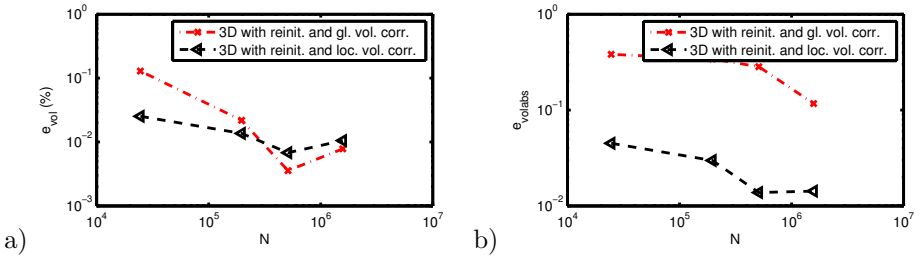
$$u(t,x,y,z) = \begin{pmatrix} 2\sin^2(\pi x)\sin(2\pi y)\sin(2\pi z)\cos(\pi t/t_f) \\ -\sin(2\pi x)\sin^2(\pi y)\sin(2\pi z)\cos(\pi t/t_f) \\ -\sin(2\pi x)\sin(2\pi y)\sin^2(\pi z)\cos(\pi t/t_f) \end{pmatrix} \tag{5.8}$$

where the time span is again given by $[t_0, t_f] = [0, 2]$ with a stretching phase until $t = 1$ and a reversal phase from 1 to 2. As before, this example, visualized in Figure 5.6 has been first published in [13] and used as a benchmark scenario in [4].

Just as in the 2D case, the difference between both mass conservation methods in the relative error e_{vol} is rather small. In fact, for higher mesh resolutions, the error of the global approach seems to be smaller than for the local conservation scheme. As for the e_∞ error, the local mass defect correction approach allows for a much better approximation of the shape. Exemplary results for $t = 2 = t_f$ are shown in Figure 5.8.

5.4. **3D-Example: Deforming droplet.** The last scenario considered in this paper extends example 5.2 to three dimensions [8]. Let $\Omega = [0,1]^3$, $t \in [0, 20]$ and φ_0

FIGURE 5.6. 3D Example deformation flow: Reference solution $\varphi_h(x,t)$ with $\varphi_h(x,0) = \varphi_h(x,2)$ at $t = 0$, $t = 0.25$, $t = 0.5$, $t = 1.0$, $t = 1.5$ and $t = 2.0$.



a) b)

FIGURE 5.7. Errors $e_{\text{vol}}(d, \varphi_h(t_f))$ (a) and $e_\infty(d, \varphi_h(t_f))$ (b) of the reinitialization and mass conservation algorithms with respect to the number of elements N.

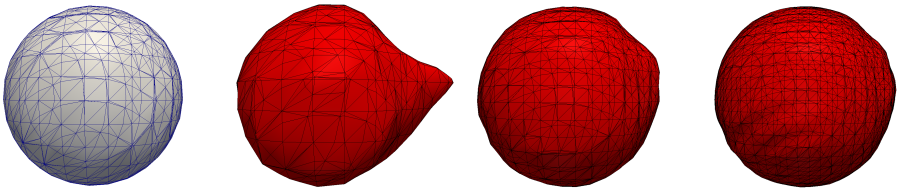


FIGURE 5.8. 3D Example deformation flow: Reference solution and numerical solution for mesh sizes $6 \times 24 \times 24$, $6 \times 44 \times 44$ and $6 \times 64 \times 64$ at $(t = 2)$using reinitialization and local volume correction.

be the zero level set of a sphere with radius 0.2 which is centered at $(0.5, 0.25, 0.5)$. The velocity field $u(t,x,y,z)$ is given by

$$u(t,x,y,z) = \begin{cases} c(x,y,z) \cdot (\tilde{y}, -\tilde{x}, 0), & \text{for } t \leq \frac{1}{2} t_f, \\ -c(x,y,z) \cdot (\tilde{y}, -\tilde{x}, 0), & \text{for } t > \frac{1}{2} t_f, \end{cases} \tag{5.9}$$

with

$$c(x,y,z) = \begin{cases} 4||(\tilde{x},\tilde{y},\tilde{z})|| \; (0.5 - ||(\tilde{x},\tilde{y},\tilde{z})||), & \text{for } ||(\tilde{x},\tilde{y},\tilde{z})|| \leq 0.5, \\ 0, & \text{otherwise,} \end{cases}$$

and $(\tilde{x},\tilde{y},\tilde{z}) = (x - 0.5, y - 0.5, z - 0.5)$. The movement and deformation of the droplet in this example is shown for different times in Figure 5.9.

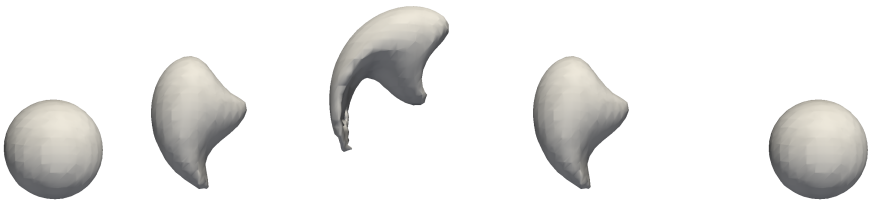


FIGURE 5.9. 3D Example rising deforming droplet: Reference solution $\varphi_h(x,t)$ with $\varphi_h(x,0) = \varphi_h(x,20)$ at $t = 0$, $t = 5$, $t = 10$, $t = 15$ and $t = 20$.

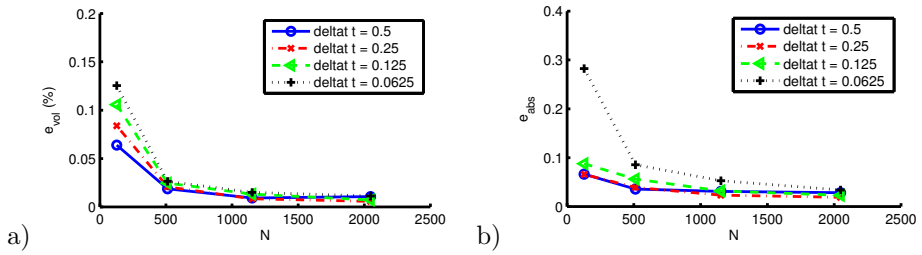


FIGURE 5.10. Errors $e_{\text{vol}}(d, \varphi_h(t_f))$ (a) and $e_\infty(d, \varphi_h(t_f))$ (b) of the global mass conservation algorithm with respect to different time step sizes and the number of elements N.

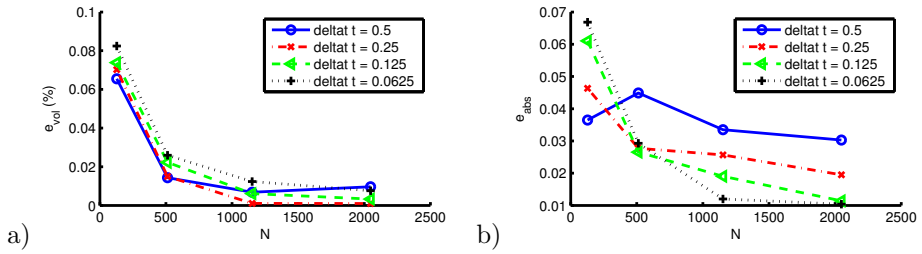


FIGURE 5.11. Errors $e_{\text{vol}}(d, \varphi_h(t_f))$ (a) and $e_\infty(d, \varphi_h(t_f))$ (b) of the local mass conservation algorithm with respect to different time step sizes and the number of elements N.

As one can see in the error plots Figure 5.10 and Figure 5.11, the local mass conservation approach performs best in both errors. Additionally, it can be observed that applying reinitialization and mass correction methods in every step may not be advisable. Instead, a criterion based on the quality of $\nabla\varphi_h$ should be used to decide whether we should reinitialize our level set function φ_h or continue with the current one.

6. Conclusions

We have given a brief overview of the level set method and presented approaches for reinitializing the level set function and correcting the mass defect. Therefore, we adapted known methods and implemented them into a toolbox for the `FEniCS` environment. The toolbox was tested using different scenarios in both, 2D and 3D. By using this toolbox, a user can easily implement more complicated problems that base on a level set formulation. In future, this toolbox will be used as foundation of an XFEM toolbox within the `FEniCS` framework.

Acknowledgement

The authors gratefully acknowledge the financial support by the DFG (German Research Foundation) for the subproject A3 within the Collaborative Research Center SFB 747 “Mikrokaltumformen - Prozesse, Charakterisierung, Optimierung”.

References

[1] M. S. Alnaes, A. Logg, K.-A. Mardal, O. Skavhaug, and H. P. Langtangen. Unified framework for finite element assembly. *International Journal of Computational Science and Engineering*, 4(4):231–244, 2009.

[2] M. S. Alnaes, A. Logg, K. B. Olgaard, M. E. Rognes, and G. N. Wells. Unified form language: A domain-specific language for weak formulations of partial differential equations. *ACM Trans. Math. Softw.*, 40(2):9:1–9:37, March 2014.

[3] Ned Anderson and Åke Björck. A new high order method of regula falsi type for computing a root of an equation. *BIT Numerical Mathematics*, 13(3):253–264, 1973.

[4] R. Ausas, E. Dari, and G. Buscaglia. A mass-preserving geometry-based reinitialization method for the level set function. *Mecánica Computacional*, 27:25–27, 2008.

[5] A. N. Brooks and T. J. R. Hughes. Streamline upwind/petrov-galerkin formulations for convection dominated flows with particular emphasis on the incompressible navier-stokes equations. *Comput. Methods Appl. Mech. Eng.*, pages 199–259, 1990.

[6] K. W. Cheng and T.-P. Fries. Higher-order xfem for curved strong and weak discontinuities. *International Journal for Numerical Methods in Engineering*, 82(5):564–590, 2010.

[7] P. E. Farrell, D. A. Ham, S. W. Funke, and M. E. Rognes. Automated derivation of the adjoint of high-level transient finite element programs. *SIAM Journal on Scientific Computing*, 35(4):C369–C393, 2013.

[8] S. Gross and A. Reusken. *Numerical Methods for Two-phase Incompressible Flows*. Springer Series in Computational Mathematics. Springer, 2011.

[9] J. Hoffman, J. Jansson, C. Degirmenci, N. Jansson, and M. Nazarov. *Unicorn: a Unified Continuum Mechanics Solver*, chapter 18. Springer, 2012.

[10] S. Hysing. A new implicit surface tension implementation for interfacial flows. *International Journal for Numerical Methods in Fluids*, 51(6):659–672, 2006.

[11] R. C. Kirby. Algorithm 839: Fiat, a new paradigm for computing finite element basis functions. *ACM Trans. Math. Softw.*, 30(4):502–516, December 2004.

[12] R. C. Kirby and A. Logg. A compiler for variational forms. *ACM Trans. Math. Softw.*, 32(3):417–444, September 2006.

[13] Randall J. Leveque. High-resolution conservative algorithms for advection in incompressible flow. *SIAM Journal on Numerical Analysis*, 33(2):627–665, 1996.

[14] A. Logg, K.-A. Mardal, and G. N. Wells, editors. *Automated Solution of Differential Equations by the Finite Element Method*, volume 84 of *Lecture Notes in Computational Science and Engineering*. Springer, 2012.

[15] A. Logg and G. N. Wells. DOLFIN: Automated finite element computing. *ACM Trans Math Software*, 37(2):20:1–20:28, 2010.

[16] F. Mut, G. Buscaglia, and E. Dari. New mass-conserving algorithm for level set redistancing on unstructured meshes. *Journal of Applied Mechanics*, pages 73–1011, 2004.

[17] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[18] K. B. Olgaard and G. N. Wells. Optimizations for quadrature representations of finite element tensors through automated code generation. *ACM Trans. Math. Softw.*, 37(1):8:1–8:23, January 2010.

[19] S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of Computational Physics*, 79(1):12 – 49, 1988.

[20] A. Reusken and E. Loch. *On the Accuracy of the Level Set SUPG Method for Approximating Interfaces*. Bericht. Inst. für Geometrie und Praktische Mathematik, 2011.

[21] H.G. Roos, M. Stynes, and L. Tobiska. *Robust Numerical Methods for Singularly Perturbed Differential Equations: Convection-Diffusion-Reaction and Flow Problems*. Springer Series in Computational Mathematics. Springer, 2008.

[22] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996.

[23] M. Sussman, P. Smereka, and S. Osher. A level set approach for computing solutions to incompressible two-phase flow. *Journal of Computational Physics*, 114(1):146 – 159, 1994.

The Center for Industrial Mathematics and MAPEX Center for Materials and Processes, University of Bremen, 28359 Bremen

E-mail address: mischa@math.uni-bremen.de

E-mail address: s_tm3ji4@uni-bremen.de