9-month Research Report

Marco Agnese
PhD student in Applied Mathematics
AMMP Section, Department of Mathematics, Imperial College London
London SW7 2AZ, UK
`m.agnese13@imperial.ac.uk`

1 Research topic

1.1 Two-phase Navier–Stokes problem

My Ph.D. research project consists in the analytical study and numerical implementation of a finite element method to solve a mathematical model of two-phases Navier–Stokes flow in 2-D and 3-D. Numerical methods for these flows have many important applications, which range from fuel injection in engines to biomedical engineering. Two-phase flow is a free boundary problem since the evolution of the interface is not prescribed but it is an unknown of the problem.

In the literature there are different computational techniques to deal with the numerical treatment of the unknown interface. One class of approaches is based on interface capturing methods using an indicator function to describe the interface. The *volume of fluid method* and the *level set method* fall into this category. In the former, the characteristic function of one of the phases is approximated numerically, see e.g. [23, 31, 30]; whereas in the latter, the interface is given as the level set of a function, which has to be determined, see e.g. [22].

In *phase field methods* the interface is assumed to have a small, but positive, thickness and an additional parabolic equation, defined in the whole domain, has to be solved in these so-called diffuse interface models, see e.g. [24, 2, 29, 17, 28, 1] for details.

Lastly, in *front tracking methods* the interface is approximated by a polyhedral surface, [14], and equations on the surface mesh have to be coupled to quantities defined on the bulk mesh, see e.g. [34, 3, 33, 19].

Recently in [7, 8] a novel front-tracking method was developed which has good mesh properties. We note that in [7, 8] an *unfitted approach* was considered, which means that the numerical approximation of the interface is totally independent of the bulk mesh. The aim of my research project is to extend the ideas from [7, 8] to the *fitted approach*, where the discrete interface approximation is given as an union of edges/faces of the bulk mesh elements.

We consider two-phase flows in a given domain $\Omega \subset \mathbb{R}^d$, where $d = 2$ or $d = 3$. The domain Ω contains two different immiscible incompressible phases (liquid-liquid or liquid-gas) which for all $t \in [0,T]$ occupy time dependent regions $\Omega_+(t)$ and $\Omega_-(t) := \Omega \setminus \overline{\Omega}_+(t)$ and which are separated by an interface $(\Gamma(t))_{t\in[0,T]}$, $\Gamma(t) \subset \Omega$.

Moreover we assume that $(\Gamma(t))_{t\in[0,T]}$ is a sufficiently smooth evolving hypersurface.

Let $\rho(t) = \rho_+ \mathbf{1}_{\Omega_+(t)} + \rho_- \mathbf{1}_{\Omega_-(t)}$, with $\rho_\pm \in \mathbb{R}_{\geq 0}$, denote the fluid densities, where here and throughout $\mathbf{1}_{\mathcal{A}}$ defines the characteristic function for a set $\mathcal{A}$. Denoting by $\vec{u} : \Omega \times [0,T] \to \mathbb{R}^d$ the fluid velocity, by $\underline{\underline{\sigma}} : \Omega \times [0,T] \to \mathbb{R}^{d\times d}$ the stress tensor, and by $\vec{f} : \Omega \times [0,T] \to \mathbb{R}^d$ a possible forcing, the incompressible Navier–Stokes equations in the two phases are given by

$$\rho\,(\vec{u}_t + \vec{u}\cdot\nabla\,\vec{u}) - \nabla\cdot\underline{\underline{\sigma}} = \vec{f} := \rho\,\vec{f}_1 + \vec{f}_2 \qquad \text{in } \Omega_\pm(t)\,, \tag{1a}$$

$$\nabla\cdot\vec{u} = 0 \qquad \text{in } \Omega_\pm(t)\,, \tag{1b}$$

$$\vec{u} = \vec{0} \qquad \text{on } \partial_1\Omega\,, \tag{1c}$$

$$\vec{u}\cdot\vec{n} = 0\,, \quad [\underline{\underline{\sigma}}\,\vec{n} + \beta\,\vec{u}]\cdot\vec{t} = 0 \quad \forall\,\vec{t} \in \{\vec{n}\}^\perp \qquad \text{on } \partial_2\Omega\,, \tag{1d}$$

where $\partial\Omega = \partial_1\Omega \cup \partial_2\Omega$, with $\partial_1\Omega \cap \partial_2\Omega = \emptyset$, denotes the boundary of Ω with outer unit normal $\vec{n}$ and $\{\vec{n}\}^\perp := \{\vec{t} \in \mathbb{R}^d : \vec{t}\cdot\vec{n} = 0\}$. Hence (1c) prescribes a no-slip condition on $\partial_1\Omega$, while (1d) prescribes a general slip condition on $\partial_2\Omega$. Here we assume that $\beta \geq 0$ and note that $\beta = 0$ corresponds to the so-called free-slip conditions.

Moreover, the stress tensor in (1a) is defined by

$$\underline{\underline{\sigma}} = \mu\,(\nabla\,\vec{u} + (\nabla\,\vec{u})^T) - p\,\underline{\underline{Id}}\,, \tag{2}$$

where $\underline{\underline{Id}} \in \mathbb{R}^{d\times d}$ denotes the identity matrix, $p : \Omega \times [0,T] \to \mathbb{R}$ is the pressure and $\mu(t) = \mu_+ \mathbf{1}_{\Omega_+(t)} + \mu_- \mathbf{1}_{\Omega_-(t)}$, with $\mu_\pm \in \mathbb{R}_{>0}$, denotes the dynamic viscosities in the two phases.

On the free surface $\Gamma(t)$, the following conditions need to hold:

$$[\vec{u}]_-^+ = \vec{0} \qquad \text{on } \Gamma(t)\,, \tag{3a}$$

$$[\underline{\underline{\sigma}}\,\vec{\nu}]_-^+ = -\gamma\,\varkappa\,\vec{\nu} \qquad \text{on } \Gamma(t)\,, \tag{3b}$$

$$\mathcal{V} = \vec{u}\cdot\vec{\nu} \qquad \text{on } \Gamma(t)\,, \tag{3c}$$

where $\gamma > 0$ is the surface tension coefficient where we have adopted the sign convention that $\varkappa$ is negative where $\Omega_-(t)$ is locally convex. Moreover, $\vec{\nu}$ is the unit normal to the interface and (3c) is a geometric PDE, see Sec.1.2 for more details. As usual, $[\vec{u}]_{-}^{+} := \vec{u}_{+} - \vec{u}_{-}$ and $[\underline{\underline{\sigma}}\,\vec{\nu}]_{-}^{+} := \underline{\underline{\sigma}}_{+}\,\vec{\nu} - \underline{\underline{\sigma}}_{-}\,\vec{\nu}$ denote the jumps in velocity and normal stress across the interface $\Gamma(t)$.

The system (1a–d), (2), (3a–c) is closed with the initial conditions

$$\Gamma(0) = \Gamma_0\,, \qquad \rho(\cdot,0)\,\vec{u}(\cdot,0) = \rho(\cdot,0)\,\vec{u}_0 \qquad \text{in } \Omega\,, \tag{4}$$

where $\Gamma_0 \subset \Omega$ and $\vec{u}_0 : \Omega \to \mathbb{R}^d$ are given initial data.

My project goal is to extend [7, 8] to the fitted case which will require combining [7, 8] with ideas from the *ALE* approach, see e.g. [25, 18, 20].

1.2 Mathematical description of the interface

As starting point, I consider the simple problem of purely geometric PDEs. Several different methods to approximate evolving surfaces can be found in the literature, these methods can be divided in explicit and implicit, see e.g. [14]. The parametric approach is an example of an explicit method which involves seeking a parametrization over a base surface (e.g. a sphere). On the other hand, the surface can be represented as the zero level set of a certain function giving rise to the implicit method known as level set method. Another implicit method is the so called phase field method which approximates the interface by a zero level set of a phase field satisfying a PDE depending on a new parameter.

We choose to use a parametric approach which consists of a description of the hypersurface $\Gamma(t)$ as

$$\Gamma(t) = \vec{x}(\cdot,t)(M) \tag{5}$$

where $M \subset \mathbb{R}^d$ is the reference manifold and

$$\vec{x} : M \times [0,T) \to \mathbb{R}^d \tag{6}$$

is one unknown of the problem. Finally $\vec{x}(\vec{p},t)$, $\vec{p} \in M$, is the position vector at a certain time t and at a certain point $\vec{p}$ of the reference manifold. All the geometrical quantities of the hypersurface (e.g. curvature) can be expressed as derivatives of the parametrization.

An equation which describes the motion of a surface prescribing the normal velocity is called *geometric PDE*. More precisely a general geometric evolution equation has the following form

$$V = f(\vec{x}, \vec{\nu}, \varkappa), \qquad \text{on } \Gamma(t), \tag{7}$$

where $\vec{\nu}$ is the normal direction and $\varkappa$ is the sum of the principal curvatures of $\Gamma(t)$. One of the simplest geometric PDEs is the one which arises from motion by *mean curvature*, see e.g. [14],

$$V = -\varkappa, \qquad \text{on } \Gamma(t). \tag{8}$$

This equation describes a surface evolving in such a way that its own normal velocity is equal to the sum of the $d-1$ principal curvatures of $\Gamma(t)$.

Another important geometric PDE is the one which arises from motion by *surface diffusion*

$$V = -\Delta_{\Gamma(t)}\varkappa, \qquad \text{on } \Gamma(t) \tag{9}$$

where $\Delta_{\Gamma(t)}$ is the *Laplace–Beltrami* operator, also called surface Laplacian, defined as

$$\Delta_{\Gamma(t)} f(\vec{p}) = \nabla_{\Gamma(t)} \cdot \nabla_{\Gamma(t)} f(\vec{p}), \qquad \vec{p} \in \Gamma(t). \tag{10}$$

Here $\nabla_{\Gamma(t)}$ is the *tangential gradient* defined as

$$\nabla_{\Gamma(t)} f(\vec{p}) = \nabla f(\vec{p}) - \nabla f(x) \cdot \vec{\nu}(\vec{p})\vec{\nu}(\vec{p}), \qquad \vec{p} \in \Gamma(t). \tag{11}$$

It can be shown that both (8) and (9) decrease the surface area of $\Gamma(t)$ over time. Moreover (9) conserves the volume enclosed by $\Gamma(t)$, which means that a sphere remains exactly the same sphere when it evolves according to surface diffusion. See [14] for more details on geometric evolution equations and their numerical approximation.

1.3 Finite element formulation of the interface

The *Finite Element Method* which I use is based on the seminal paper [16] and I refer to the ones described in [5, 4, 6]. The surface diffusion problem (9) can be rewritten as a system of second order equations

$$\begin{cases} \vec{x}_t \cdot \vec{\nu} = -\Delta_{\Gamma(t)}\varkappa, \\ \varkappa\vec{\nu} = \Delta_{\Gamma(t)}\vec{x}, \end{cases} \tag{12}$$

where we have used the fact that the *normal velocity* can be expressed as a function of the parametrization

$$V = \vec{x}_t \cdot \vec{\nu}. \tag{13}$$

The second equation, instead, is a well-known identity from surface geometry.

The FEM scheme for (12) from [6] assumes the following formulation

$$\begin{cases} \langle \frac{\vec{X}^{m+1} - \vec{X}^m}{\tau_m}, \chi\vec{\nu}^m \rangle^h_{\Gamma^m} - \langle \nabla_{\Gamma^m}\kappa^{m+1}, \nabla_{\Gamma^m}\chi \rangle^h_{\Gamma^m} = 0, \\ \langle \kappa^{m+1}\vec{\nu}^m, \vec{\eta} \rangle^h_{\Gamma^m} + \langle \nabla_{\Gamma^m}\vec{X}^{m+1}, \nabla_{\Gamma^m}\vec{\eta} \rangle_{\Gamma^m} = 0 \end{cases} \tag{14}$$

$\forall \vec{\eta} \in \underline{V}(\Gamma^m)$ and $\forall \chi \in W(\Gamma^m)$. These spaces of test functions are defined as

$$\underline{V}(\Gamma^m) := \{\vec{\chi} \in C(\Gamma^m, \mathbb{R}^d) : \vec{\chi}|_{\sigma_j^m} \text{ is linear } \forall j = 1 \to J\} := [W(\Gamma^m)]^d \subset H^1(\Gamma^m, \mathbb{R}^d) \tag{15}$$

where $W(\Gamma^m) \subset H^1(\Gamma^m)$ is the space of scalar continuous piecewise linear functions on Γ^m. Here Γ^m is the polyhedral surface which approximates the surface $\Gamma(t^m)$ at time t^m and $\{\sigma_j^m\}_{j=1}^J$ is a family of mutually disjoint open triangles such that $\Gamma^m = \cup_{j=1}^J \overline{\sigma_j^m}$.

Moreover $\langle \cdot, \cdot \rangle_{\Gamma^m}$ is the L^2 inner product over the current polyhedral surface

$$\langle u, v \rangle_{\Gamma^m} = \int_{\Gamma^m} uv \, \mathrm{d}s, \qquad \forall u, v \in L^2(\Gamma^m), \tag{16}$$

and similarly for $\vec{u}, \vec{v} \in L^2(\Gamma^m, \mathbb{R}^d)$. In addition, $\langle \cdot, \cdot \rangle_{\Gamma^m}^h$ is a mass lumped inner product with the vertices of each triangle of the mesh as quadrature points.

For what concerns the mean curvature flow, the system of second order equations is

$$\begin{cases} \vec{x}_t \cdot \vec{\nu} = -\varkappa, \\ \varkappa \vec{\nu} = \Delta_{\Gamma(t)} \vec{x}, \end{cases} \tag{17}$$

and the FEM formulation from [6] is

$$\begin{cases} \langle \frac{\vec{X}^{m+1} - \vec{X}^m}{\tau_m}, \chi \vec{\nu}^m \rangle_{\Gamma^m}^h - \langle \kappa^{m+1}, \chi \rangle_{\Gamma^m}^h = 0, \\ \langle \kappa^{m+1} \vec{\nu}^m, \vec{\eta} \rangle_{\Gamma^m}^h + \langle \nabla_{\Gamma^m} \vec{X}^{m+1}, \nabla_{\Gamma^m} \vec{\eta} \rangle_{\Gamma^m} = 0. \end{cases} \tag{18}$$

1.4 Algebraic formulation of the interface

As regards the mean curvature flow, (18), the corresponding algebraic system of equations can be written as

$$\begin{pmatrix} \tau_m M_m & -\vec{N}_m^T \\ \vec{N}_m & \vec{A}_m \end{pmatrix} \begin{pmatrix} \kappa^{m+1} \\ \delta \vec{X}^{m+1} \end{pmatrix} = \begin{pmatrix} 0 \\ -\vec{A}_m \vec{X}^m \end{pmatrix}, \tag{19}$$

while, for the system of equations describing the surface diffusion, (14), the algebraic system is

$$\begin{pmatrix} \tau_m A_m & -\vec{N}_m^T \\ \vec{N}_m & \vec{A}_m \end{pmatrix} \begin{pmatrix} \kappa^{m+1} \\ \delta \vec{X}^{m+1} \end{pmatrix} = \begin{pmatrix} 0 \\ -\vec{A}_m \vec{X}^m \end{pmatrix}. \tag{20}$$

We can notice that the only difference between the above systems occurs in the upper-left entry of the matrix.

The entries of the matrix are defined as

$$
\begin{aligned}
[M_m]_{kl} &:= \langle \phi_k^m \,, \phi_l^m \rangle_m^h, &(21)\\
\left[\vec{N}_m\right]_{kl} &:= \int_{\Gamma^m} \pi^h \, [\phi_k^m, \phi_l^m] \, \vec{\nu}^m \mathrm{d}s, &(22)\\
[A_m]_{kl} &:= \langle \nabla_s \phi_k^m \,, \nabla_s \phi_l^m \rangle_m, &(23)
\end{aligned}
$$

where $\{\phi_k\}$ are the basis functions of the finite element space of piecewise linear continuous functions $W(\Gamma^m)$ and

$$[\vec{A}_m]_{kl} := [A_m]_{kl} \vec{Id}, \tag{24}$$

where $\vec{Id}$ is the identity matrix.

The linear systems (19) and (20) are invertible under a suitable assumption on the triangulation at each time step, see [6]. Hence they can be solved with a sparse factorization package such as `UMFPACK`, see [13].

Alternately, the algebraic system (19) can be solved with a *Schur complement* approach:

$$
\begin{aligned}
\kappa^{m+1} &= \frac{1}{\tau_m} M_m^{-1} \vec{N}_m^T \, \delta \vec{X}^{m+1}, &(25a)\\
(\vec{A}_m + \frac{1}{\tau_m} \vec{N}_m M_m^{-1} \vec{N}_m^T) \, \delta \vec{X}^{m+1} &= -\vec{A}_m \, \vec{X}^m, &(25b)
\end{aligned}
$$

where (25b) is symmetric and positive definite.

1.5 Software used

In order to simulate numerically the evolution governed by mean curvature flow and surface diffusion I have chosen to use the `DUNE` framework (acronym for `Distributed and Unified Numerics Environment`)), see [10, 9, 11, 12].

`DUNE` is a modular toolbox for solving partial differential equations with grid-based methods. It implements slim interfaces allowing an efficient use of legacy and new libraries. Moreover it is coded with modern `C++` programming techniques levering heavily on template meta-programming. These techniques enable very different implementations of the same concept (like grids, solvers) using a common interface at a very low overhead. For this reasons, this framework ensures efficiency in scientific computations and supports high-performance computing applications.

`DUNE` is based on the following main principles:

- separation of data structures and algorithms by abstract interfaces;
- efficient implementation of these interfaces using generic programming techniques;

- reuse of existing finite element packages with a large body of functionality.

For what concern the FEM discretisation, I use the module `dune-fem`, based on the `dune-grid` interface library, see [15]. This module aims at extending `dune-grid` by a number of discretisation algorithms for the resolution of non-linear systems of partial differential equations.

The main concept is that of a *spatial discrete operator* which models a mapping between two discrete function spaces: $L_h : U_h \to V_h$. Basic operators can be combined to construct more complex operators.

Finally, `dune-fem` offers different solvers, both direct and iterative; unfortunately the direct solvers are only sequential up to now.

In order to construct the discrete operators, the user has to choose a continuous function space, a set of basis functions over it and a view of the underlying grid. The view determines the part of the grid on which the functions are defined. So far, it is possible to use discontinuous and Lagrange finite element spaces.

1.6 Preliminary numerical results

I implemented a `DUNE` module, called `dune-interface`, to simulate surface diffusion, (14), and mean curvature flow, (18).

This module can simulate surfaces embedded both in 2-D and in 3-D. Moreover, it supports `ALBERTA`, see [32], and `ALUGrid`, see [35], as grid managers. More precisely, `ALUGrid` manages only 2-D surfaces but it can be used both sequentially and in parallel (using `MPI`). On the other hand, `ALBERTA` supports 1-D and 2-D surfaces but it can be run only sequentially.

When running the code sequential, the module uses the `UMFPACK` direct solver. Finally, the initial mesh is created using `GMSH`, see [21], and it is read by the module from a `.msh` file.

To test my module, I performed some simulations reported in the following section.

1.6.1 Mean curvature flow with a sphere as initial condition

As regards the mean curvature flow, the simplest case is to use a sphere as the initial condition, see [27]. Indeed, the sphere shrinks without changing shape since the curvature is the same all around. In this case (8) can be rewritten as

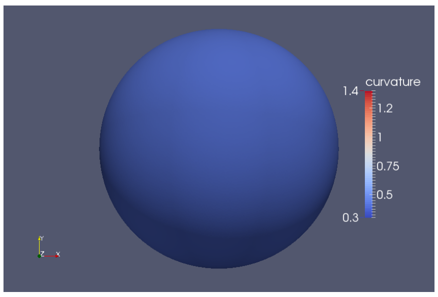
$$\frac{\mathrm{d}R}{\mathrm{d}t} = -\frac{d-1}{R}, \qquad R(0) = R_0 \tag{26}$$

where $R(t)$ is the radius of the sphere and d is the dimension of the world where the sphere is embedded. Equation (26) has the following analytical solution

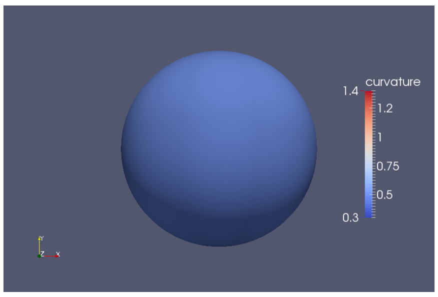
$$R(t) = \sqrt{R_0^2 - 2(d-1)t}, \tag{27}$$

so the sphere disappears at time

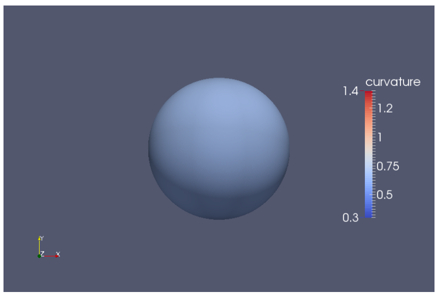
$$t_e = \frac{R_0^2}{2(d-1)}. \tag{28}$$



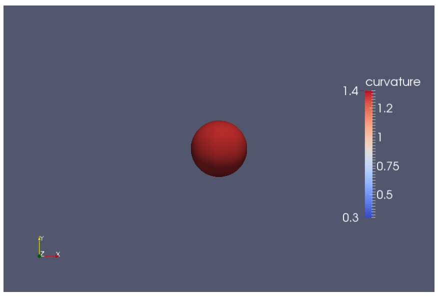
(a)



(b)



(c)



(d)

Figure 1: Mean curvature flow with a sphere as initial condition.

In Fig.1 there are 4 snapshots of the simulation at time $t = 0.1$, $t = 2.1$, $t = 4.1$ and $t = 6.1$ respectively. The initial sphere has radius $R_0 = 5$; obviously $d = 3$ since the simulation is 3-D. In this case, the extinction time, (28), is $t_e = \frac{25}{4} = 6.25$. The number of mesh elements is 74770 and there are 37387 nodes. The time step is constant and equal to $\tau = 0.1$.

We can see that the evolution is consistent to what was expected.

1.6.2 Mean curvature flow with a cube as initial condition

Since mean curvature flow (8) is a parabolic equation it will have a short-term smoothing effect on the surface. However, the surface can become singular later. It has been proven, [26], that if $\Gamma(0)$ is a bounded convex surface, than $\Gamma(t)$ becomes more and more nearly spherical as it shrinks, and at the instant it vanishes, it is asymptotic to the shrinking sphere given above.

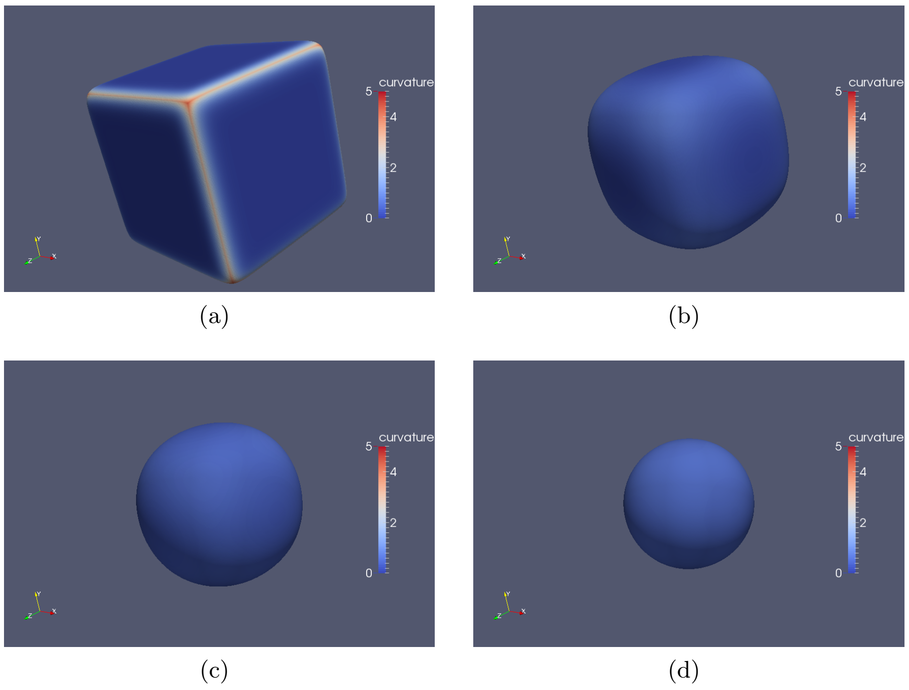


Figure 2: Mean curvature flow with a cube as initial condition.

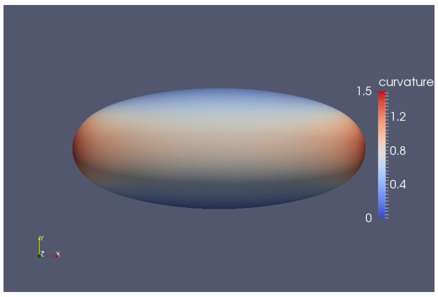
In Fig. 2 there are 4 snapshots of the simulation at time $t = 0.1$, $t = 2.1$, $t = 4.1$ and $t = 6.1$ respectively. The initial cube has edges of length $L = 10$. The number of mesh elements is 158420 and there are 79212 nodes. The time step is constant and equal to $\tau = 0.1$.

As expected, there is a smoothing effect and the cube shrinks becoming slowly a sphere.

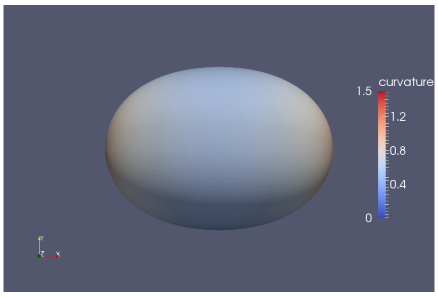
1.6.3 Surface diffusion with an ellipsoid as initial condition

In order to test the surface diffusion I have used as initial condition an ellipsoid. The ellipsoid should evolve towards a sphere and then become stationary. Moreover the enclosed volume should remain constant during all the computation.

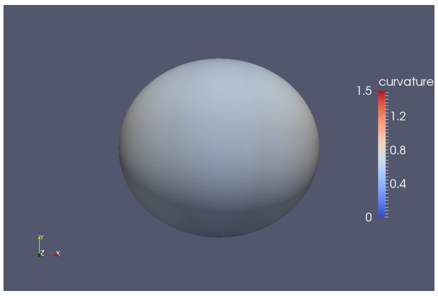
In Fig. 3 there are 4 snapshots of the simulation at time $t = 0.1$, $t = 5.1$, $t = 10.1$ and $t = 20.1$ respectively. The initial ellipsoid has semi-axes of



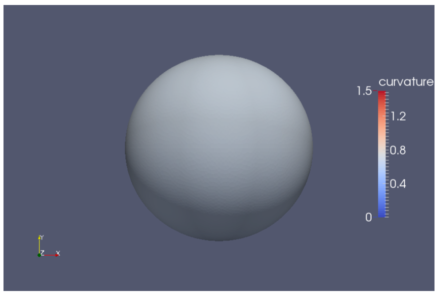
(a)



(b)



(c)



(d)

Figure 3: Surface diffusion with an ellipsoid as initial condition.

length 5, 2 and 3 along the x-axis, y-axis and z-axis, respectively. The number of mesh elements is 31464 and there are 15734 nodes. The time step is constant and equal to $\tau = 0.1$.

Also in this case the numerical result is consistent.

1.6.4 Surface diffusion with an ellipse as initial condition

Finally, in order to testing the case of a 1-D surface (curve), I have used as initial condition an ellipse that then evolves accord to surface diffusion. Also for the ellipse we expect an evolution towards a circle. Moreover the enclosed area should remain constant during all the computation.

In Fig. 4 there are 4 snapshots of the simulation at time $t = 0.1$, $t = 10.1$, $t = 20.1$ and $t = 30.0$ respectively. The initial ellipse has semi-axes of length 5 and 3 along the x-axis and y-axis respectively. The number of mesh elements is 232 and there are 232 nodes. The time step is constant and equal to $\tau = 0.1$.

As before, we have obtained what expected.

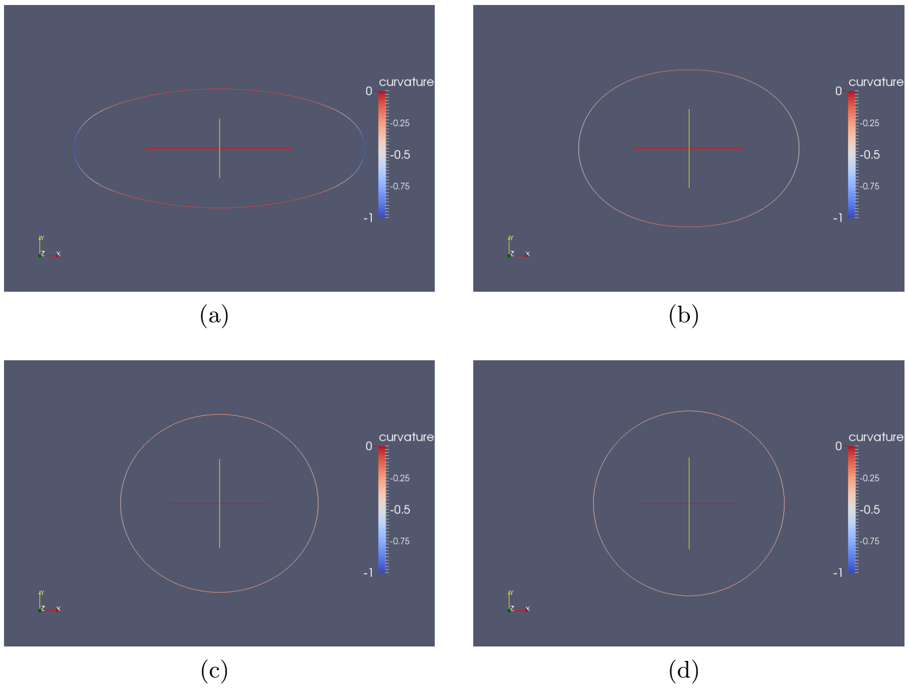


Figure 4: Surface diffusion with an ellipse as initial condition.

1.7 Grid and sub–grid

So far I have described how to solve the geometric PDE over the interface. However, for the project, we also need to couple an equation for the bulk (e.g. Navier–Stokes).

Unfortunately the core modules of `DUNE` do not allow to have a grid with a problem defined on it and a sub-grid, of lower dimension, with another problem. To tackle this issue I wrote another module to manage these complex types of grids called `dune-dynamic-grid`.

This module offers different functionalities which are essential for my project. The most important are:

- dynamic grids;
- coupled grid–sub-grid;
- re-meshing.

In the following sections I will describe very briefly these features.

1.7.1 Dynamic grids

`dune-grid` is the `DUNE` core module which provides the interface for every grid. All the grids, once created, cannot be modified.

The only operation which modifies the grid is the refinement, performed usually by splitting one edge. When the user performs a refinement, a level in the hierarchical grid is added for all the elements which are refined.

Since the problems which I am dealing with modify the grid at every time step, I need an efficient way to change the values of the grid without recreating the mesh at each time step (very inefficient).

`dune-dynamic-grid` implements a new type of grid which maps a wrapped grid over a host grid using a linear projection of the nodes. In this way, every time the grid evolves, changing only the coefficients of the projection (which are simply the values of the new nodes), the wrapped grid evolves accordingly.

The problems implemented in `dune-interface` are defined always over a wrapped grid and therefore the module uses `dune-dynamic-grid`.

1.7.2 Coupled grid–sub-grid

Another missing feature of `dune-grid` is the possibility to extract a sub-grid of lower dimension from a grid.

My modules offers an infrastructure to extract and store a certain submesh of lower dimension. The important part is the coupling since for `DUNE` the two grids are two independent objects. The module therefore creates a map (with linear complexity for what concerns the random access) between elements of the two grids. This map is essential to couple the interface problem with the bulk problem.

1.7.3 Re-meshing

Sometimes, it can be necessary to rebuild the mesh completely. A typical example occurs when there is an interface which moves in the bulk. In this case the size of the elements decrease downstream and increase upstream. At a certain point, it is impossible to adapt the mesh in order to improve the quality and the only solution is to rebuild the bulk grid.

Obviously the boundaries of the bulk grid and the mesh of the interface need to be kept unchanged during this process since it is meaningless to add or remove nodes. More precisely, only the original nodes which describe the boundaries are prescribed while the other nodes are recomputed to avoid accumulation effects.

I have implemented a class which loads the initial mesh and stores the geometrical informations describing the boundaries. The user can rebuild the mesh prescribing an interface.

This class uses the API of `Gmsh` [21] to create the mesh. There are available different meshing algorithms both for 2-D and 3-D.

When the mesh is rebuilt, the class uses the `DUNE` grid factory to create a `DUNE` grid.

1.8 Coupling the bulk with the interface

So far, my modules `dune-interface` and `dune-dynamic-grid` allow to solve certain geometric PDEs over a surface and enrich the features of the basic grid offered by `DUNE`. Nevertheless, my project consists of a coupled problem in the bulk and over the interface.

I have implemented a third module, `dune-coupled`, which couples the PDEs over the interface and the PDEs in the bulk. Up to now, I haven't coded the module `dune-bulk` yet, therefore `dune-coupled` manages only the coupling of the grid–sub-grid and solves the PDEs over the surface. In the bulk no PDE is solved.

In the following section I will show a simulation using this module to understand better how it works.

1.8.1 Surface diffusion with an ellipse as initial condition coupled with the bulk

In this numerical simulation I want to test the code with a coupled grid–sub-grid with a surface diffusion problem for the interface. The bulk domain is a rectangle while the initial interface is an ellipse. The interface partitions the bulk in two regions: inner region and outer region. In the bulk I do not solve any problem therefore the coupling is only geometrical.

In Fig. 5 there are 4 snapshots of the simulation at time $t = 5$, $t = 10$, $t = 20$ and $t = 30$, respectively. The edges of the rectangle have length 6 and 4, respectively. The initial ellipse has semi-axes of length 5 and 3 along the x-axis and the y-axis, respectively. The number of mesh elements for the interface is constant and equal to 52 elements. Obviously the number of nodes is 52. The time step is constant and equal to $\tau = 0.1$.

Since over the interface we are solving the same problem of Example 1.6.4 the evolution of the interface is the same. Nevertheless, the sub-grid is coupled to the bulk now. Therefore when it evolves, it moves also the

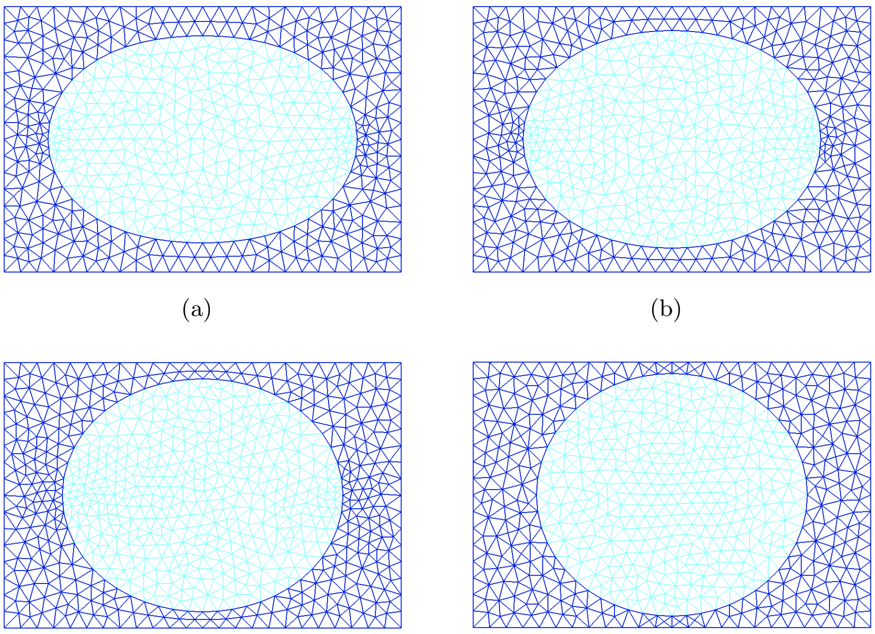

(a)
(b)
(c)
(d)

Figure 5: Surface diffusion with an ellipse as initial condition coupled with the bulk.

nodes of the grid. For this reason the quality of the mesh changes at each time-step.

The module performs a complete “re-meshing” of the bulk at certain time-steps in order to improve the quality of the grid. Obviously the boundary of the bulk and the interface are kept fixed during the re-mesh process.

2 Future work

2.1 Project plan

Up to now, there are several things which I have to do in the following months. Some of them should be quite straightforward while other will be more challenging. A very rough plan is the following.

First of all I am going to extend the module `dune-dynamic-grid` adding some mesh quality controls and some algorithms to perform a mesh smoothing.

These two features are necessary because it is not feasible to rebuild the mesh at each time step, since that is computationally very expensive compared to a smoothing.

After that, the next step will be solution of the Navier–Stokes equation in the bulk. This will result in a new module called `dune-bulk`. The simulator for the bulk problem will interact with the simulator for the interface problem thanks to the module `dune-coupled`.

So far, the module `dune-coupled` manages only the geometrical coupling and the simulator for the interface. For this reason I need also to work on this module.

Finally, an optional feature which I would like to add is a full support for parallel running for all the modules at least for 3-D problems. Actually some part are already in parallel and some others are very simple to parallelize. Nevertheless, this goal is quite difficult to achieve globally due to the complex mesh management. I am confident that also a partial support to parallel runs will boost up the code very much allowing faster computations.

3 Courses and conferences

3.1 Courses for academic credits

This year I took the following courses:

- **Manifolds**, Master Course, 30 hours;
- **Theory of PDE's**, TCC, 20 hours;
- **Advanced Finite Element Theory**, TCC, 20 hours;
- **Pseudo Differential Operators**, TCC, 20 hours;

completing 90 hours over the 100 hours which I have to do in my first two years of Ph.D. .

3.2 Courses for Graduate School

In compliance with College regulations about Transferable Skills, I have already attended four courses given by the Graduate School, precisely the following:

- **Research Skills & Development**, (RSD) Course, 3 courses;
- **Negotiation Skills for Researchers**, 1 course.

Moreover, I have completed the **Plagiarism Awareness** course and I have met College Postgraduate Research English requirement.

3.3 Conferences

I attended the conference Numerical Analysis for Partial Differential Equations, held by University of Sussex, 18 - 20 June 2014.

References

[1] H. Abels, H. Garcke, and G. Grün, *Thermodynamically consistent, frame indifferent diffuse interface models for incompressible two-phase flows with different densities*, Math. Models Methods Appl. Sci., 22 (2012), p. 1150013.

[2] D. M. Anderson, G. B. McFadden, and A. A. Wheeler, *Diffuse-interface methods in fluid mechanics*, in Annual review of fluid mechanics, Vol. 30, Annual Reviews, Palo Alto, CA, 1998, pp. 139–165.

[3] E. Bänsch, *Finite element discretization of the Navier–Stokes equations with a free capillary surface*, Numer. Math., 88 (2001), pp. 203–235.

[4] J. W. Barrett, H. Garcke, and R. Nürnberg, *On the variational approximation of combined second and fourth order geometric evolution equations*, SIAM J. Sci. Comput., 29 (2007), pp. 1006–1041.

[5] ——, *A parametric finite element method for fourth order geometric evolution equations*, J. Comput. Phys., 222 (2007), pp. 441–462.

[6] ——, *On the parametric finite element approximation of evolving hypersurfaces in* $\mathbb{R}^3$, J. Comput. Phys., 227 (2008), pp. 4281–4307.

[7] ——, *Eliminating spurious velocities with a stable approximation of viscous incompressible two-phase Stokes flow*, Comput. Methods Appl. Mech. Engrg., 267 (2013), pp. 511–530.

[8] ——, *A stable parametric finite element discretization of two-phase Navier–Stokes flow*, J. Sci. Comp., 63 (2015), pp. 78–117.

[9] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, R. Kornhuber, M. Ohlberger, and O. Sander, *A Generic Grid Interface for Parallel and Adaptive Scientific Computing. Part II: Implementation and Tests in DUNE*, Computing, 82 (2008), pp. 121–138.

[10] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, M. Ohlberger, and O. Sander, *A Generic Grid Interface for Parallel and Adaptive Scientific Computing. Part I: Abstract Framework*, Computing, 82 (2008), pp. 103–119.

[11] M. Blatt and P. Bastian, *The Iterative Solver Template Library*, vol. 4699 of Lecture Notes in Computer Science, Springer, 2007.

[12] ——, *On the Generic Parallelisation of Iterative Solvers for the Finite Element Method*, Int. J. Comput. Sci. Engrg., 4 (2008), pp. 56–69.

[13] T. A. Davis, *Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method*, ACM Trans. Math. Software, 30 (2004), pp. 196–199.

[14] K. Deckelnick, G. Dziuk, and C. M. Elliott, *Computation of geometric partial differential equations and mean curvature flow*, Acta Numer., 14 (2005), pp. 139–232.

[15] A. Dedner, R. Klöfkorn, M. Nolte, and M. Ohlberger, *A Generic Interface for Parallel and Adaptive Scientific Computing: Abstraction Principles and the DUNE-FEM Module*, Computing, 90 (2010), pp. 165–196.

[16] G. Dziuk, *An algorithm for evolutionary surfaces*, Numer. Math., 58 (1991), pp. 603–611.

[17] X. Feng, *Fully discrete finite element approximations of the Navier–Stokes–Cahn–Hilliard diffuse interface model for two-phase fluid flows*, SIAM J. Numer. Anal., 44 (2006), pp. 1049–1072.

[18] S. Ganesan, *Finite element methods on moving meshes for free surface and interface flows*, PhD thesis, University Magdeburg, Magdeburg, Germany, 2006.

[19] S. Ganesan, G. Matthies, and L. Tobiska, *On spurious velocities in incompressible flow problems with interfaces*, Comput. Methods Appl. Mech. Engrg., 196 (2007), pp. 1193–1202.

[20] J.-F. Gerbeau, C. Le Bris, and T. Lelièvre, *Mathematical methods for the magnetohydrodynamics of liquid metals*, Numerical Mathematics and Scientific Computation, Oxford University Press, Oxford, 2006.

[21] C. Geuzaine and J.-F. Remacle, *Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities*, Internat. J. Numer. Methods Engrg., 79 (2009), pp. 1309–1331.

[22] S. Gross and A. Reusken, *An extended pressure finite element space for two-phase incompressible flows with surface tension*, J. Comput. Phys., 224 (2007), pp. 40–58.

[23] C. W. Hirt and B. D. Nichols, *Volume of fluid (VOF) method for the dynamics of free boundaries*, J. Comput. Phys., 39 (1981), pp. 201–225.

[24] P. C. Hohenberg and B. I. Halperin, *Theory of dynamic critical phenomena*, Rev. Mod. Phys., 49 (1977), pp. 435–479.

[25] T. J. R. Hughes, W. K. Liu, and T. K. Zimmermann, *Lagrangian–Eulerian finite element formulation for incompressible viscous flows*, Comput. Methods Appl. Mech. Engrg., 29 (1981), pp. 329–349.

[26] G. Huisken, *Flow by mean curvature of convex surfaces into spheres*, J. Differential Geom., 20 (1984), pp. 237–266.

[27] T. Ilmanen, *Lectures on Mean Curvature Flow and Related Equations*, 1998.

[28] D. Kay, V. Styles, and R. Welford, *Finite element approximation of a Cahn–Hilliard–Navier–Stokes system*, Interfaces Free Bound., 10 (2008), pp. 15–43.

[29] J. Lowengrub and L. Truskinovsky, *Quasi-incompressible Cahn–Hilliard fluids and topological transitions*, R. Soc. Lond. Proc. Ser. A Math. Phys. Eng. Sci., 454 (1998), pp. 2617–2654.

[30] S. Popinet, *An accurate adaptive solver for surface-tension-driven interfacial flows*, J. Comput. Phys., 228 (2009), pp. 5838–5866.

[31] Y. Renardy and M. Renardy, *PROST: a parabolic reconstruction of surface tension for the volume-of-fluid method*, J. Comput. Phys., 183 (2002), pp. 400–421.

[32] A. Schmidt and K. G. Siebert, *Design of Adaptive Finite Element Software: The Finite Element Toolbox ALBERTA*, vol. 42 of Lecture Notes in Computational Science and Engineering, Springer-Verlag, Berlin, 2005.

[33] G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, and Y.-J. Jan, *A front-tracking method for the computations of multiphase flow*, J. Comput. Phys., 169 (2001), pp. 708–759.

[34] S. O. Unverdi and G. Tryggvason, *A front-tracking method for viscous, incompressible multi-fluid flows*, J. Comput. Phys., 100 (1992), pp. 25–37.

[35] `DUNE ALUGrid`, *https://www.dune-project.org/modules/dune-alugrid/*.