Archive of Numerical Software
vol. 1, no. 2, pages 1–19

A Framework for the Implementation of Coupled Multiphase Flow Problems Using the `deal.II` Library and Its Application to Electrowetting

Abner J. Salgado *[1]

[1]Department of Mathematics, University of Maryland, College Park, MD 20742, USA.
abnersg@math.umd.edu

Received: December 28th, 2011; **final revision:** September 3rd, 2012; **published:** July 2nd, 2013.

Abstract: The purpose of this work is to introduce some ideas aimed at simplifying the implementation of schemes for the solution of multiphysics and multidomain problems and describe their realization in the `deal.II` library. Namely, we introduce a general interface to describe problems and how different sub-problems might depend on each other. We illustrate the applicability of these concepts on the discretization of the phenomenon known as electrowetting.

1 Introduction

Most interesting physical phenomena involve the interaction between disparate physical processes which are, generally, governed by quite different laws. We will call them multiphysics problems. Such problems can be posed as a system of differential equations which are, mathematically, of different type. Variable density ([Lions, 1996]) and multiphase ([Liu and Shen, 2003]) flows can serve as examples of such type of problems. Multidomain problems are those where the quantities of interest lie in different (possibly time-dependent) parts of a bigger domain. As examples of these one can mention fluid-structure interactions [Lefrançois and Boufflet, 2010, Causin et al., 2005, Badia et al., 2008a,b, Tezduyar et al., 2006, Bazilevs et al., 2008, Richter and Wick, 2010] and contact problems, with [Krause and Wohlmuth, 2002] or without friction [Krause and Wohlmuth, 2003, Kornhuber and Krause, 2001].

The term electrowetting refers to the local modification of the surface tension between two immiscible fluids via electric actuation. This allows to change the shape and wetting behavior of a two fluids system and, thus, it is possible to manipulate it. This phenomenon was originally discovered by Lippmann [Lippmann, 1875] more than a century ago, and it has recently found a wide spectrum of applications, such as: reprogrammable lab-on-chip systems [Lee et al., 2002, Saeki et al., 2001], auto-focus cell phone lenses [Berge and Peseux, 2000], colored oil pixels and video speed smart paper [Hayes and Feenstra, 2003, Roques-Carmes et al., 2004a,b].

*This material is based on work supported by NSF grants CBET-0754983 and DMS-0807811 and an AMS-Simons Grant.

Figure 2.1: The basic configuration of an electrowetting on dielectric device [Cho et al., 2001, 2003]. The solid black region depicts the dielectric plates and the white region denotes a droplet of one fluid (say water), which is surrounded by another (air). We will denote by Ω the fluid domain and by $\Omega^\star$ the region occupied by the fluids and the plates. The boundary between the fluid and the plates is denoted by Γ and $\partial^\star\Omega = \partial\Omega^\star \setminus \Gamma$.

Given the wide range of applications this phenomenon has, it is important to develop reliable computational tools for the simulation of these effects. As it is the case with any real world problem, a complete description of all the involved processes might become too expensive or it is simply impossible due to the fact that the phenomena at hand are not completely understood. For this reason one must often deal with approximate models, which must be complete enough, so that they can reproduce the most important physical effects, yet sufficiently simple that it is possible to extract from them meaningful information in a reasonable amount of computing time. This is the main motivation for our current research program. We have proposed and analyzed a new model of electrowetting and obtained efficient discretization techniques for it ([Salgado, 2013, Nochetto et al., 2014]). In this work, we shall be concerned with the implementation of the aforementioned numerical techniques.

This work is organized as follows. In Section 2 the model of electrowetting on dielectric is described, as well as the discretization technique that we shall apply. Section 3 will describe the main ideas and data structures behind the general framework for the solution of multiphysics and multidomain problems and some details in their implementation. This is the core of this work and the application of these ideas to the problem of electrowetting, was our main motivation in developing these tools. Computational results are presented in Section 4. Section 5 provides instructions for compilation and execution; as well as details on the run-time parameters. Finally, some concluding remarks and possibilities for extensions are discussed in Section 6.

2 The problem and its discretization

The purpose of this Section is to introduce the problem we shall be concerned with and describe the discretization technique we shall apply. For details the reader is referred to [Nochetto et al., 2014].

2.1 Notation

Figure 2.1 shows the basic configuration for the electrowetting on dielectric problem. We will use the symbol Ω to denote the domain occupied by the fluid and $\Omega^\star$ for the fluid and dielectric plates. Thus, $\Omega \subset \Omega^\star$. With this notation, Ω and $\Omega^\star$ are bounded connected domains of $\mathbb{R}^d$, for $d = 2$ or 3. The boundary of Ω will be denoted by Γ and $\partial^\star\Omega^\star = \partial\Omega^\star \setminus \Gamma$. We will denote by $[0, T]$ with $0 < T < \infty$ the time interval of interest. For any vector valued function $\mathbf{w} : \Omega \to \mathbb{R}^d$ that is smooth enough so as to have a trace on Γ, we define

$$\mathbf{w}_\tau|_\Gamma := \mathbf{w}|_\Gamma - (\mathbf{w}|_\Gamma \cdot \mathbf{n})\mathbf{n}, \tag{2.1}$$

where $\mathbf{n}$ is the outer normal to Γ.

Let $D \subset \mathbb{R}^d$. To denote the classical Lebesgue and Sobolev spaces, we use the standard notation $L^p(D)$ and $W_p^k(D)$, respectively. In addition $H^k(D) = W_2^k(D)$. Spaces of vector valued functions and its elements will be denoted by boldface characters. For $S \subset \mathbb{R}^d$, by $\langle \cdot, \cdot \rangle_S$ we denote, indistinctly, the $L^2(S)$- or $\mathbf{L}^2(S)$-inner product. If no subscript is given, we assume that the domain is Ω. If $S \subset \mathbb{R}^{d-1}$, then the inner product is denoted by $[\cdot, \cdot]_S$ and if no subscript is given, the manifold must be understood to be Γ. We define the spaces

$$H^1_\star(\Omega^\star) := \left\{ v \in H^1(\Omega^\star) : v|_{\partial^\star \Omega^\star} = 0 \right\}, \tag{2.2}$$

and

$$\mathbf{V} := \left\{ \mathbf{v} \in \mathbf{H}^1(\Omega) : \mathbf{v} \cdot \mathbf{n}|_\Gamma = 0 \right\}. \tag{2.3}$$

2.2 Description of the problem

In this work we shall be concerned with the following system of equations:

$$\begin{cases} \phi_t + \mathsf{u} \cdot \nabla \phi = \nabla \cdot (M(\phi) \nabla \mu), & \text{in } \Omega, \\ \mu = \gamma \left(\dfrac{1}{\delta} \mathcal{W}'(\phi) - \delta \Delta \phi \right) - \dfrac{1}{2} \varepsilon'(\phi) |\nabla V|^2 + \dfrac{1}{2} \rho'(\phi) |\mathsf{u}|^2, & \text{in } \Omega, \\ \alpha \left(\phi_t + \mathsf{u}_\tau \partial_\tau \phi \right) + \gamma \left(\Theta'_{fs}(\phi) + \delta \partial_{\mathbf{n}} \phi \right) = 0, \; M(\phi) \partial_n \mu = 0, & \text{on } \Gamma, \end{cases} \tag{2.4}$$

$$\begin{cases} \dfrac{D(\rho(\phi)\mathsf{u})}{Dt} - \nabla \cdot \left(\eta(\phi) \mathbf{S}(\mathsf{u}) \right) + \nabla \mathsf{p} = \mu \nabla \phi - q \nabla \left(V + \lambda q \right) + \dfrac{1}{2} \rho'(\phi) \phi_t \mathsf{u} & \text{on } \Omega, \\ \nabla \cdot \mathsf{u} = 0, & \text{in } \Omega, \\ \mathsf{u} \cdot \mathbf{n} = 0, & \text{on } \Gamma, \\ \beta(\phi) \mathsf{u}_\tau + \eta(\phi) \mathbf{S}(\mathsf{u})_{\mathbf{n}\tau} = \gamma \left(\Theta'_{fs}(\phi) + \delta \partial_{\mathbf{n}} \phi \right) \partial_\tau \phi, & \text{on } \Gamma, \end{cases} \tag{2.5}$$

where $\frac{D(\rho(\phi)\mathsf{u})}{Dt} := \sigma(\phi)(\sigma(\phi)\mathsf{u})_t + \rho(\phi)\mathsf{u} \cdot \nabla \mathsf{u} + \frac{1}{2} \nabla \cdot (\rho(\phi)\mathsf{u})\mathsf{u}$, and $\sigma := \sqrt{\rho}$, $\mathbf{S}(\mathsf{u}) = \nabla \mathsf{u} + \nabla^\intercal \mathsf{u}$ is the symmetric gradient,

$$\begin{cases} q_t + \nabla \cdot (q\mathsf{u}) = \nabla \cdot \left[K(\phi) \nabla \left(\lambda q + V \right) \right], & \text{in } \Omega, \\ K(\phi) \nabla \left(\lambda q + V \right) \cdot \mathbf{n} = 0, & \text{on } \Gamma, \end{cases} \tag{2.6}$$

$$\begin{cases} -\nabla \cdot \left([\![\varepsilon(\phi)]\!] \nabla V \right) = q \chi_\Omega, & \text{in } \Omega^\star, \\ V = V_0, & \text{on } \partial^\star \Omega^\star, \\ \partial_{\mathbf{n}} V = 0, & \text{on } \partial \Omega^\star \cap \Gamma, \end{cases} \tag{2.7}$$

where

$$[\![\varepsilon(\phi)]\!] = \begin{cases} \varepsilon(\phi), & \Omega, \\ \varepsilon^\star, & \Omega^\star \setminus \Omega, \end{cases}$$

with $\varepsilon^\star$ constant.

The unknowns in system (2.4)–(2.7) are: ϕ – the so-called phase field variable, μ – the chemical potential, u – the velocity of the fluid, p – its pressure, q – the charge distribution and V – the voltage (which is defined in $\Omega^\star$). The material parameters M, ρ, η, K and ε are the mobility, density, viscosity, conductivity and permittivity; respectively. They are assumed to be smooth functions of the phase field, so that we allow them to depend on the phase. A possible definition of the density, for instance, is

$$\rho(\phi) = \frac{\rho_1 - \rho_2}{2} \phi + \frac{\rho_1 + \rho_2}{2},$$

where ρ_i, $i = 1, 2$ is the density of each one of the phases. The interface thickness is denoted by δ, the surface tension coefficient is γ. The constants α and λ are regularization parameters. The function

$$\mathcal{W}(\xi) = \begin{cases} (\xi + 1)^2, & \xi < -1, \\ \frac{1}{4}\left(1 - \xi^2\right)^2, & |\xi| \leq 1, \\ (\xi - 1)^2, & \xi > 1, \end{cases}$$

is known as the Ginzburg-Landau potential. $\gamma\Theta_{fs}$ is the interface energy density, where

$$\Theta_{fs}(\phi) = \frac{\cos\theta_s}{2} \sin\left(\frac{\pi\phi}{2}\right),$$

and θ_s is the static contact angle, i.e., the angle the interface between the two fluids makes at equilibrium. This is regarded as a material property. The system is supplemented with initial conditions $(\phi^0, \mathsf{u}^0, q^0)$ for the phase field, velocity and charge, respectively.

Remark 2.1 *Without going into details, let us briefly describe the rationale behind this model. The externally applied voltage V_0 induces an electric field (described by its potential V) and a charge distribution q which, in turn, act on the fluid (described by its velocity u and pressure p). The movement of the fluid transports the charge distribution and changes the material parameters (say ε and ρ), thus inducing a change in the voltage V.*

Remark 2.2 *We must remark that this is not the only nor most complete model for electrowetting. The simplifying assumptions that led to system* (2.4)–(2.7) *as well as comparison with other models are detailed in [Nochetto et al., 2014]. Let us contempt here with noticing that the introduction of the diffuse interface is at the level of the continuous model and it is related with the fact that an accurate description of the three phase contact line poses severe difficulties from the modeling, as well as the analysis, points of view. Although at the computational level this might seem similar to interface capturing techniques like level set methods, the two approaches are quite different as here the "diffuse interface" is given at the continuous level, whereas in interface capturing methods it is introduced for computational convenience. Nevertheless, the advantages and disadvantages inherent to interface capturing techniques are also present in our model.*

2.3 Discretization

Let us briefly describe the discretization technique. To discretize in time, we divide the time interval $[0, T]$ into subintervals of length $\Delta t > 0$ (for simplicity assumed constant) and look for sequences $\psi_{\Delta t} = \{\psi^n\}$ which approximate $\psi^n \approx \psi(n\Delta t)$. For any sequence $\psi_{\Delta t}$, we define the time increment operator $\mathfrak{d}$ as

$$\mathfrak{d}\psi^n = \psi^n - \psi^{n-1},$$

and the time average operator

$$\psi^{\star,n} = \frac{1}{2}\left(\psi^n + \psi^{n-1}\right).$$

To discretize in space, we introduce a parameter $h > 0$ and let $\mathbb{W}_h \subset H^1_\star(\Omega^\star)$, $\mathbb{Q}_h \subset H^1(\Omega)$, $\mathbb{X}_h \subset \mathbf{V}$ and $\mathbb{M}_h \subset L^2_{\int=0}(\Omega)$ be finite element spaces. We require that the polynomial degree of $\mathbb{W}_h$ is not bigger than that of $\mathbb{Q}_h$ and, moreover, that the pair of spaces $(\mathbb{X}_h, \mathbb{M}_h)$ satisfies the so-called LBB condition (see [Girault and Raviart, 1986, Ern and Guermond, 2004]), that is, there exists a constant c independent of h such that

$$c\|\bar{p}_h\|_{L^2} \leq \sup_{\mathbf{v}_h \in \mathbb{X}_h} \frac{\int_\Omega \bar{p}_h \nabla\cdot\mathbf{v}_h}{\|\mathbf{v}_h\|_{\mathbf{H}^1}}, \quad \forall \bar{p}_h \in \mathbb{M}_h. \tag{2.8}$$

The space $\mathbb{W}_h$ will be used to approximate the voltage; $\mathbb{Q}_h$ the charge, phase field and chemical potential; and $\mathbb{X}_h$, $\mathbb{M}_h$ the velocity and pressure, respectively. Finally, to account for the boundary conditions on the voltage, we denote

$$\mathbb{W}_h(\bar{V}_0^{k+1}) = \mathbb{W}_h + \bar{V}_0^{k+1}.$$

The starting point for our discretization scheme is problem (2.4)–(2.7) which, being nonlinear, we linearize (in time) with lagging of the variables. Moreover, for the Cahn Hilliard Navier Stokes part we employ the fractional time-stepping technique developed in [Salgado, 2013]. In other words, at each time step we know

$$(V_h^n, q_h^n, \phi_h^n, \mu_h^n, \mathbf{u}_h^n, p_h^n, \xi_h^n) \in \mathbb{W}_h(\bar{V}_0^n) \times (\mathbb{Q}_h)^3 \times \mathbb{X}_h \times (\mathbb{M}_h)^2,$$

with $\xi_h^0 := 0$ and, to advance in time, solve the following sequence of discrete and linear problems:

- Potential: Find $V_h^{n+1} \in \mathbb{W}_h(\bar{V}_0^{n+1})$ that solves:

$$\left\langle [\![\varepsilon(\phi_h^n)]\!] \nabla V_h^{n+1}, \nabla W_h \right\rangle_{\Omega^\star} = \left\langle q_h^n, W_h \right\rangle, \quad \forall W_h \in \mathbb{W}_h, \tag{2.9}$$

- Charge: Find $q_h^{n+1} \in \mathbb{Q}_h$ that solves:

$$\left\langle \frac{\mathfrak{d} q_h^{n+1}}{\Delta t}, r_h \right\rangle - \left\langle q_h^n \mathbf{u}_h^n, \nabla r_h \right\rangle + \left\langle K(\phi_h^n) \nabla \left(\lambda q_h^{n+1} + V_h^{n+1} \right), \nabla r_h \right\rangle = 0, \quad \forall r_h \in \mathbb{Q}_h, \tag{2.10}$$

- Phase Field and Potential: Find ϕ_h^{n+1}, $\mu_h^{n+1} \in \mathbb{Q}_h$ that solve:

$$\left\langle \frac{\mathfrak{d} \phi_h^{n+1}}{\Delta t}, \bar{\phi}_h \right\rangle + \left\langle \mathbf{u}_h^n \cdot \nabla \phi_h^n, \bar{\phi}_h \right\rangle + \left\langle M(\phi_h^n) \nabla \mu_h^{n+1}, \nabla \bar{\phi}_h \right\rangle = 0, \quad \forall \bar{\phi}_h \in \mathbb{Q}_h, \tag{2.11}$$

$$\begin{aligned} \left\langle \mu_h^{n+1}, \bar{\mu}_h \right\rangle = {} & \frac{\gamma}{\delta} \left\langle \mathcal{W}'(\phi_h^n) + \mathcal{A} \mathfrak{d} \phi_h^{n+1}, \bar{\mu}_h \right\rangle + \gamma \delta \left\langle \nabla \phi_h^{n+1}, \nabla \bar{\mu}_h \right\rangle \\ & - \frac{1}{2} \left\langle (\varepsilon'(\phi_h^n) + C \mathfrak{d} \phi_h^{n+1}) |\nabla V_h^{n+1}|^2, \bar{\mu}_h \right\rangle + \frac{1}{2} \left\langle \rho'(\phi_h^n) |\mathbf{u}_h^n|^2, \bar{\mu}_h \right\rangle \\ & + \alpha \left[\frac{\mathfrak{d} \phi_h^{n+1}}{\Delta t} + \mathbf{u}_{h\tau}^n \partial_\tau \phi_h^n, \bar{\mu}_h \right] + \gamma \left[\Theta'_{fs}(\phi_h^n) + \mathcal{B} \mathfrak{d} \phi_h^{n+1}, \bar{\mu}_h \right] \quad \forall \bar{\mu}_h \in \mathbb{Q}_h, \end{aligned} \tag{2.12}$$

- Velocity: Define $p_h^\sharp = p_h^n + \xi_h^n$, then find $\mathbf{u}_h^{n+1} \in \mathbb{X}_h$ such that

$$\begin{aligned} & \left\langle \frac{\rho^{\star,n+1}(\phi_h) \mathbf{u}_h^{n+1} - \rho(\phi^n) \mathbf{u}_h^n}{\Delta t}, \mathbf{w}_h \right\rangle + \left\langle \rho(\phi_h^n) \mathbf{u}_h^n \cdot \nabla \mathbf{u}_h^{n+1}, \mathbf{w}_h \right\rangle + \frac{1}{2} \left\langle \nabla \cdot (\rho(\phi_h^n) \mathbf{u}_h^n) \mathbf{u}_h^{n+1}, \mathbf{w}_h \right\rangle \\ & \quad + \left\langle \eta(\phi_h^n) \mathbf{S}(\mathbf{u}_h^{n+1}), \mathbf{S}(\mathbf{w}_h) \right\rangle - \left\langle p_h^\sharp, \nabla \cdot \mathbf{w}_h \right\rangle + \left[\beta(\phi_h^n) \mathbf{u}_{h\tau}^{n+1}, \mathbf{w}_{h\tau} \right] + \alpha \left[\mathbf{u}_{h\tau}^{n+1} \partial_\tau \phi_h^n, \mathbf{w}_{h\tau} \partial_\tau \phi_h^n \right] \\ & \quad = \left\langle \mu_h^{n+1} \nabla \phi_h^n, \mathbf{w}_h \right\rangle - \left\langle q_h^n \nabla (\lambda q_h^{n+1} + V_h^{n+1}), \mathbf{w}_h \right\rangle + \frac{1}{2} \left\langle \rho'(\phi_h^n) \frac{\mathfrak{d} \phi_h^{n+1}}{\Delta t} \mathbf{u}_h^n, \mathbf{w}_h \right\rangle \\ & \quad \quad - \alpha \left[\frac{\mathfrak{d} \phi_h^{n+1}}{\Delta t}, \mathbf{w}_{h\tau} \partial_\tau \phi_h^n \right] \quad \forall \mathbf{w}_h \in \mathbb{X}_h. \end{aligned} \tag{2.13}$$

- Penalization and Pressure: Finally, ξ_h^{n+1} and p_h^{n+1} are computed via

$$\left\langle \nabla \xi_h^{n+1}, \nabla \bar{p}_h \right\rangle = -\frac{\varrho}{\Delta t} \left\langle \nabla \cdot \mathbf{u}_h^{n+1}, \bar{p}_h \right\rangle, \quad \forall \bar{p}_h \in \mathbb{M}_h, \tag{2.14}$$

  where $\varrho := \min\{\rho_1, \rho_2\}$ and

$$p_h^{n+1} = p_h^n + \xi_h^{n+1}. \tag{2.15}$$

The study of the stability and convergence properties of this scheme is well beyond the scope of this work. We refer the interested reader to [Nochetto et al., 2014, Salgado, 2013] for details. Let us only mention that the constants $\mathcal{A}$, $\mathcal{B}$ and C play the rôle of stabilization parameters. Their admissible values can be found in the aforementioned references.

3 A general framework

The algorithm presented in §2.3, i.e., (2.9)–(2.15), involves only the solution of linear problems. However, it presents several difficulties for implementation. First of all, the equations are posed in different parts of a common domain. In addition, each of the equations depend on the solution of the others at previous time steps, hence we must develop an efficient way to be able to use the solution of one problem as a coefficient or forcing term in another.

To address the first difficulty we take advantage of the `hp`-framework as detailed in [Bangerth et al., step-46]. The second issue, on the other hand, requires some more attention. It can certainly be circumvented by keeping in a master class the `hp::DoFHandler` objects for each one of the subproblems. Most, if not all, of the tutorial programs for the `deal.II` library proceed this way (see [Bangerth et al.]). While this helps in keeping matters simple, it is our belief that this approach is contrary to one of the key concepts in object oriented programming – *encapsulation*.

Let us explain this in more detail. To assemble, say, the local system matrices associated with problem (2.9) one only needs to know, whether the cell belongs to Ω or $\Omega^\star \setminus \Omega$ and, if we are in Ω, the values at the quadrature points of the expression $[\![\varepsilon(\phi_h^n)]\!]$. It is of no relevance to this procedure that this actually comes from the solution of a finite element problem with, say, globally continuous d-linear elements. Maybe, for reasons of implementation or solving, the variables ϕ_h and μ_h are discretized as a vector valued problem. This is of no consequence to the assembly procedure of problem (2.9) and, thus, all these, problem specific, nuisances must be kept hidden from this procedure. The situation becomes much more complicated when we look at (2.13). To be able to assemble the local matrices and right hand sides we would need: the phase field at the current and previous time steps: ϕ_h^{n+1}, ϕ_h^n; the velocity at the previous time step: $\mathbf{u}_h^n$; the extrapolation of the pressure: $p_h^\sharp$; the chemical potential μ_h^{n+1}; the charge at the previous and current time steps: q_h^n q_h^{n+1}; and the voltage V_h^{n+1}. In addition, we need to be able to determine when a face in a cell is at the interface between solid and fluid and, if it is, add the corresponding boundary integrals which also depend on discrete functions from other problems. Again, it is of no consequence to this assembly procedure where all these other discrete functions are coming from, what finite element space is used for their representation or what solution scheme the problem that defines them is using. The only relevant information is the values at the quadrature points of the function or its derivatives.

The idea of encapsulation is implemented in the following class.

3.1 The `AsFunction` class template

Finite element functions are represented in the computer via their coefficient vectors, i.e., the coefficients of their decomposition in the canonical finite element basis. This is done by using an object of the type `Vector` (in any of its avatars). To be able to interpret it as a function one needs a `DoFHandler` (in any of its flavors), which is the object that stores all the relevant information concerning the meaning of the degrees of freedom and basis functions. To extract local (i.e., relevant to a cell or face) information, one needs an object of the type `FEValues`, which is the one that does the actual computations of the values of the basis functions (or its derivatives) at the quadrature points.

We propose to encapsulate these three objects into a class that can be used to interpret the coefficient vector as a function, and create a wrapping interface around them that is going to operate in the spirit of the `FEValues` object, i.e., it will provide all the relevant information via

methods of the form `AsFunction::get_function_values`. The specification of this class should be as follows:

C++ code

```
class AsFunction{
  public:
    AsFunction( DoFHandler _dh, Vector _x ); // The DoFHandler interprets
        the vector
    ~AsFunction(); // Clear data
    void set_quadrature( Quadrature _quad ); // Change quadrature and
        update flags
    void set_update_flags( UpdateFlags _flags );
    void reinit( DoFHandler::active_cell_iterator _cell ); // maps to
        <FEValues::reinit>
    // Map to FEValues::get_function_* using <x> as function
    void get_function_values( std::vector<double> value );
    void get_function_gradients( std::vector<Tensor> value );
  protected:
    DoFHandler dh; // References to the passed (during construction)
        objects.
    Vector x;
    FEValues fe_val; // An <FEValues> object together with update flags and
        quadrature
    Quadrature quad;
    UpdateFlags flags;
};
```

Remark 3.1 *There is a similar class in the* `deal.II` *library, namely the* `FEFieldFunction` *class template. However, we found that this class was not suitable for our purposes. The documentation for this class clearly states that support for* `hp` *is a planned (but not yet implemented) feature. In addition, it does not provide the flexibility of allowing one to specify the quadrature formula to be used.*

3.2 A class for the description of a problem

Once discretized in time, all the problems appearing in (2.9)–(2.15) are basically matrix-vector problems and many of the steps needed to assemble the system, solve it and transfer the solution (or any other auxiliary quantity) between meshes after coarsening or refinement are independent of the specifics of the problem at hand. In an effort to promote *code reuse*, we propose a class that captures these common features.

All problems, independently of their physical significance must keep:

- A reference to the triangulation of the domain where the problem is posed or, for that matter, of a larger domain.
- A sparse matrix, a vector of solutions and a right hand side.
- A constraint matrix that deals with hanging node constraints and, possibly, implements essential boundary conditions or any other type of constraints.
- An object of type `hp::DoFHandler` and a specification of the finite element space that we are using, i.e., a child of the class `FiniteElement`.

In addition, all problems must implement a method that:

- Sets the degrees of freedom and builds the constraint matrices.

- Allocates the matrices and vectors of the correct size.
- Performs the loop over all cells to compute the local system matrix and right hand side and assemble the global ones.
- Detects that the triangulation is about to be coarsened or refined and stores the vectors that need to be transferred in a `SolutionTransfer` object.
- Once the triangulation has been coarsened or refined use the `SolutionTransfer` object, transfer the needed vectors to the new mesh. Note that this last two items are easily implemented using the `signals` mechanism that the `Triangulation` class has.
- Reinitializes (if needed) the preconditioner and calls the iterative solver.

It must be noted that it might be possible that a problem needs to do something specific before or after the system is solved, or add constraints to the `ConstraintMatrix` object due to some of its specifics, etc. We will account for this possibilities using a series of prefixes/suffixes for each one of the common methods.

Putting together these ideas, the prototype for this class looks as follows:

C++ code

```
1  class Problem{
2    public:
3      Problem( Triangulation tria ); // Store a reference to triangulation
           and init objects
4      ~Problem(); // Clear data
5      void init(); // Distribute DoFs and set Vector sizes
6      virtual void solve( vector< AsFunction> data ); // Assemble and solve
7    protected:
8      Triangulation tri; // reference to the triangulation
9      DoFHandler dh; // The DoF handler object
10     ConstraintMatrix constraints; // The hanging node (and other)
           constraints
11     FiniteElement fe; // The finite element
12     Vector sol, rhs; // Solution and the RHS
13     SparseMatrix K; // System matrix
14     void SetupDoFs(); // Setup Degrees of Freedom
15     virtual void SetupDoFsSuffix(); // Called after the degrees of freedom
           have been setup
16     void InitLAData(); // Initialize Linear Algebra data
17     virtual void InitLADataSuffix(); // Initialize problem specific LA data
18     struct PerTaskData; // Assembly data structure
19     virtual void AssembleSystem( vector< AsFunction > data ) = 0; //
           Assemble the system
20     void CopyToGlob( const PerTaskData &data ); // Copy local to global
21     void SolveSystem(); // Solve the system
22     virtual void SolveSystemPreffix(); // Called before solving the system
23     virtual void ReinitPrec() = 0; // Reinitialize the preconditioner
24     virtual void DoSolve( SolverControl &control ) = 0; // Call the actual
           solver
25     virtual void SolveSystemSuffix(); // Called after the system is solved
26     virtual void SetInitialData() = 0; // Set the initial data
27     vector<Vector> x_sol; // The vectors that are going to be transferred
28     SolutionTransfer transfer; // The transferring mechanism
29     void PreRefinement(); // Called before refinement
30     virtual void PreRefinementPreffix() = 0; // Fills <x_sol>
31     void PostRefinement(); // Called after refinement
```



```
32      virtual void PostRefinementSuffix( vector<Vector> &sol_tmp ) = 0; //
            Transfer x_sol
33  };
```

Notice that:

- The public method `solve` takes as an argument a list of `AsFunction` objects which, depending on the child class, will be interpreted as terms in the right hand side or coefficients in the equation.
- The `PerTaskData` structure defined in line 18 will contain the local matrix, local vector and the indices of the local degrees of freedom.
- There is no way that one can in advance know how a particular problem will interpret, during the assembly, the `AsFunction` objects, or how many scratch vectors will be needed. For this reason we declare, in line 19, the `AssembleSystem` method to be pure virtual, and defer to the child classes its implementation and declaration of the structures that define the scratch data.
- The same reasoning applies to the preconditioner and iterative solver. See lines 23 and 24.

3.3 An application. The `Pressure` class template

To illustrate the application of the ideas mentioned above, while keeping matters simple, let us detail the implementation of the penalization and pressure steps: (2.14)–(2.15). The description of the class is as follows:

C++ code

```
1   class Pressure: public Problem{
2     public:
3       Pressure( Triangulation tria, Material_Parameters params, unsigned deg,
            double ee, double thres, unsigned sweeps );
4       virtual ~Pressure();
5     protected:
6       const double rho_min;
7       Preconditioner prec;
8       PreconditionerData prec_data;
9       Vector p, pres_extr;
10      virtual void SetupDoFsSuffix();
11      virtual void InitLADataSuffix();
12      struct ScratchData{
13        Quadrature quad;
14        UpdateFlags flags;
15        FEValues fe_val;
16        AsFunction velocity;
17        /* scratch vectors and constructors */
18      };
19      virtual void AssembleSystem( vector< AsFunction > data, double time,
20                                   bool m_threaded = true );
21      void AssembleCell( Iterator Its, ScratchData scratch, PerTaskData data
            );
22      virtual void ReinitPrec();
23      virtual void DoSolve( SolverControl &control );
24      virtual void SolveSystemSuffix();
25      virtual void SetInitialData();
26      virtual void PreRefinementPreffix();
27      virtual void PostRefinementSuffix( vector< Vector > sol_tmp );
```



```
28    public:
29      AsFunction get_pressure, get_extrapolated_pressure;
30  };
```

Let us go over the members and methods with more detail:

- In the *constructor*, we pass the needed data to the parent class and store, from `Material_Parameters` the value ϱ, which is represented by `rho_min`. In addition we obtain data related to the polynomial degree of the finite element space and preconditioner (number of sweeps and threshold).
- The *destructor* clears the data. Notice that these are the only two public methods in this class, the other ones being inherited from `Problem`.
- The `Problem` class has already a vector destined to represent the solution of a problem, in this case this is ξ_h^{n+1}. In this class there are two vectors which will represent each one of the views of the problem, the pressure p_h^{n+1} and the extrapolated pressure $p_h^\sharp$.
- The preconditioner is given by `prec` which is an algebraic multigrid method and is initialized with the `ReinitPrec` method.
- The `SetupDoFsSuffix` adds a constraint to remove the singularity associated with homogeneous Neumann conditions.
- The `InitLADataSuffix` just sets p_h^{n+1} and $p_h^\sharp$ to the correct size.
- The assembly procedure, given by `AssembleSystem` needs to know the velocity, which is the first entry in the parameter `data`. In addition it needs some scratch data, which is defined in line 13. The actual assembly, i.e., the computation of

$$\langle \nabla\Phi_{h,\ell_1}, \nabla\Phi_{h,\ell_2}\rangle_T\,, \quad \left\langle \nabla\cdot\mathbf{u}_h^{k+1}, \Phi_{h,\ell_2}\right\rangle_T\,,$$

  where T is a cell and $\Phi_{h,\ell}$ are the local basis functions, is performed in `AssembleCell`.
- The solution (using CG) is done in `DoSolve`. After that `SolveSystemSuffix` updates p_h^{n+1} and the extrapolated pressure $p_h^\sharp$.
- Finally, there are two public `AsFunction` classes, which is how this class will communicate to the world the values of p_h^{n+1} and $p_h^\sharp$. These are attached to each one of the extra vectors that this class has declared.

4 Results

We now show the capabilities of the developed program in a series of testcases. These were presented originally in [Nochetto et al., 2014]. Unless stated otherwise, the density ratio between the two fluids is $\rho_1/\rho_2 = 100$, the viscosity ratio $\eta_1/\eta_2 = 10$ and the surface tension coefficient is $\gamma = 50$. The conductivity ratio is $K_1/K_2 = 10$ and the permittivity ratio $\varepsilon_1/\varepsilon_2 = 5$ and $\varepsilon^\star/\varepsilon_2 = 100$. We have set the mobility parameter to be constant $M = 10^{-2}$, and $\alpha = 10^{-3}$. The slip coefficient is taken constant $\beta = 10$, and the equilibrium contact angle between the two fluids is $\theta_s = 120°$. The interface thickness is $\delta = 5\cdot 10^{-2}$ and the regularization parameter $\lambda = 0.5$. The applied voltage is $V_{00} = 20$.

The time-step is set constant and $\Delta t = 10^{-3}$ and every 10 time-steps the mesh is coarsened and refined using as refinement indicator the magnitude of $|\nabla\phi|$. The discrete spaces are such that $\deg \mathbb{W}_h = 1$, i.e., the polynomial degree is one in each coordinate direction; $\deg \mathbb{Q}_h = 2$; $\deg \mathbb{X}_h = 2$ and $\deg \mathbb{M}_h = 1$.

4.1 Movement of a droplet

The first and simplest example shows that we can use electric actuation used to manipulate a droplets. The fluid occupies the domain $\Omega = (-5,5)\times(0,1)$ and above and below there are dielectric plates of thickness $1/2$, so that $\Omega^\star = (-5,5)\times(-1/2,3/2)$. A droplet of a heavier fluid with an initial shape of half a circle of radius $1/2$ is centered at the origin and initially at rest. To the right half of the lower plate we apply a voltage, so that

$$V_0 = V_{00}\chi_D, \qquad D = \left\{(x,y)\in\mathbb{R}^2 : x\geq 0,\ y=-\frac{1}{2}\right\}.$$

The initial mesh consists of 5364 cells with two different levels of refinement. Away from the two-fluid interface the local mesh size is about 0.125 and, near the interface, the local mesh size is about 0.03125. Figure 4.1 shows the evolution of the interface. An animation of these results is available at `media/merge.avi`.

4.2 Splitting of a droplet

According to the literature, one of the main advantages in using diffuse interface models is that topological changes can be handled without any special care. Let us illustrate this by showing how, using electrowetting, one can split a droplet. Initially a drop of heavier material occupies

$$\chi_{\rho_2} = \left\{(x,y)\in\mathbb{R}^2 :\ \frac{x^2}{2.5^2}+\frac{y^2}{0.5^2}\leq 1\right\}.$$

To be able to split the droplet, the externally applied voltage is

$$D = \left\{(x,y)\in\mathbb{R}^2 : |x|\geq\frac{3}{2},\ y=-\frac{1}{2}\right\}.$$

Figure 4.2 shows the evolution of the system. An animation of these results is available at `media/split.avi`.

4.3 Merging of two droplets

To finalize let us show the inverse process to §4.2, i.e., the merging of two droplets of the same material via electric actuation. The geometry is the same as before but, in this case, there are initially two droplets of heavier material, each one of radius 0.5 and centered at $(\pm 0.7, 0)$, respectively. The material parameters are the same in the previous cases, except the interfacial thickness, which is set to $\delta = 10^{-2}$. We apply an external voltage so that

$$D = \left\{(x,y)\in\mathbb{R}^2 :\ |x|\leq\frac{1}{2},\ y=-\frac{1}{2}\right\}.$$

To be able to capture the fine interfacial dynamics that merging possesses, we set the initial level of refinement to 4, with 3 extra refinements near the interface, so that the number of cells is 48,696 with a local mesh size away of the interface of about 0.02875 and near the interface of about $6\cdot 10^{-3}$. This amounts to a total of 147,249 degrees of freedom. The time-step, again, is set to $\Delta t = 10^{-3}$.

Figure 4.3 shows the evolution of the two droplets under the action of the voltage. Again, other than properly resolving the interfacial layer, we did not need to do anything special to handle the topological change. An animation of these results is available at `media/merge.avi`.

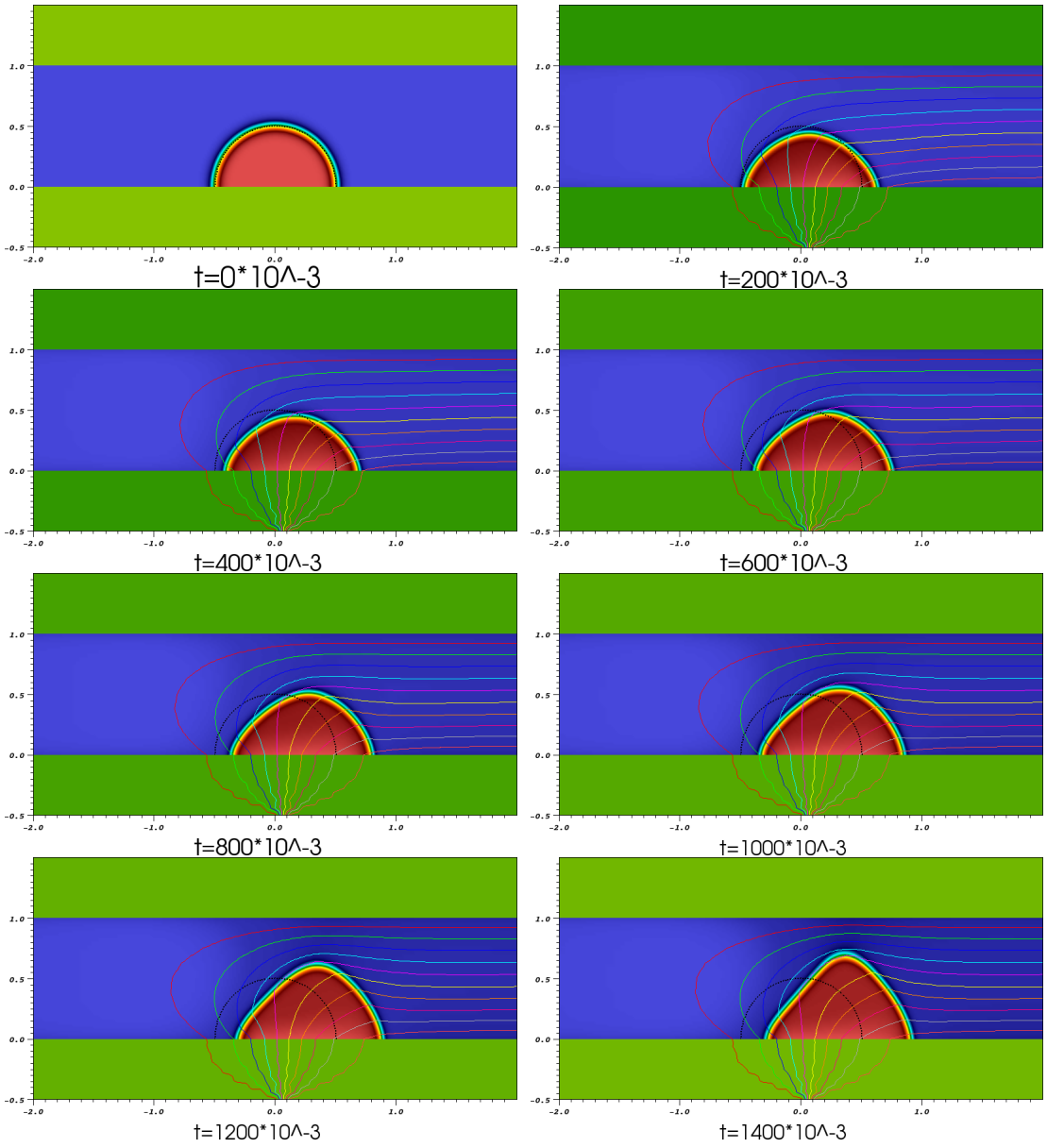

Figure 4.1: Movement of a droplet under the action of an external voltage. The material parameters are $\rho_1/\rho_2 = 100$, $\eta_1/\eta_2 = 10$, $\gamma = 50$, $K_1/K_2 = 10$, $\varepsilon_1/\varepsilon_2 = 5$, $\varepsilon^\star/\varepsilon_2 = 100$, $M = 10^{-2}$, $\alpha = 10^{-3}$, $\beta = 10$, $\theta_s = 120°$, $\delta = 5 \cdot 10^{-2}$, $\lambda = 0.5$ and $V_{00} = 20$. The interface is shown at times 0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2 and 1.4. Colored lines are used to represent the iso-values of the voltage. The black dotted line is the position of the interface at the beginning of the computations.

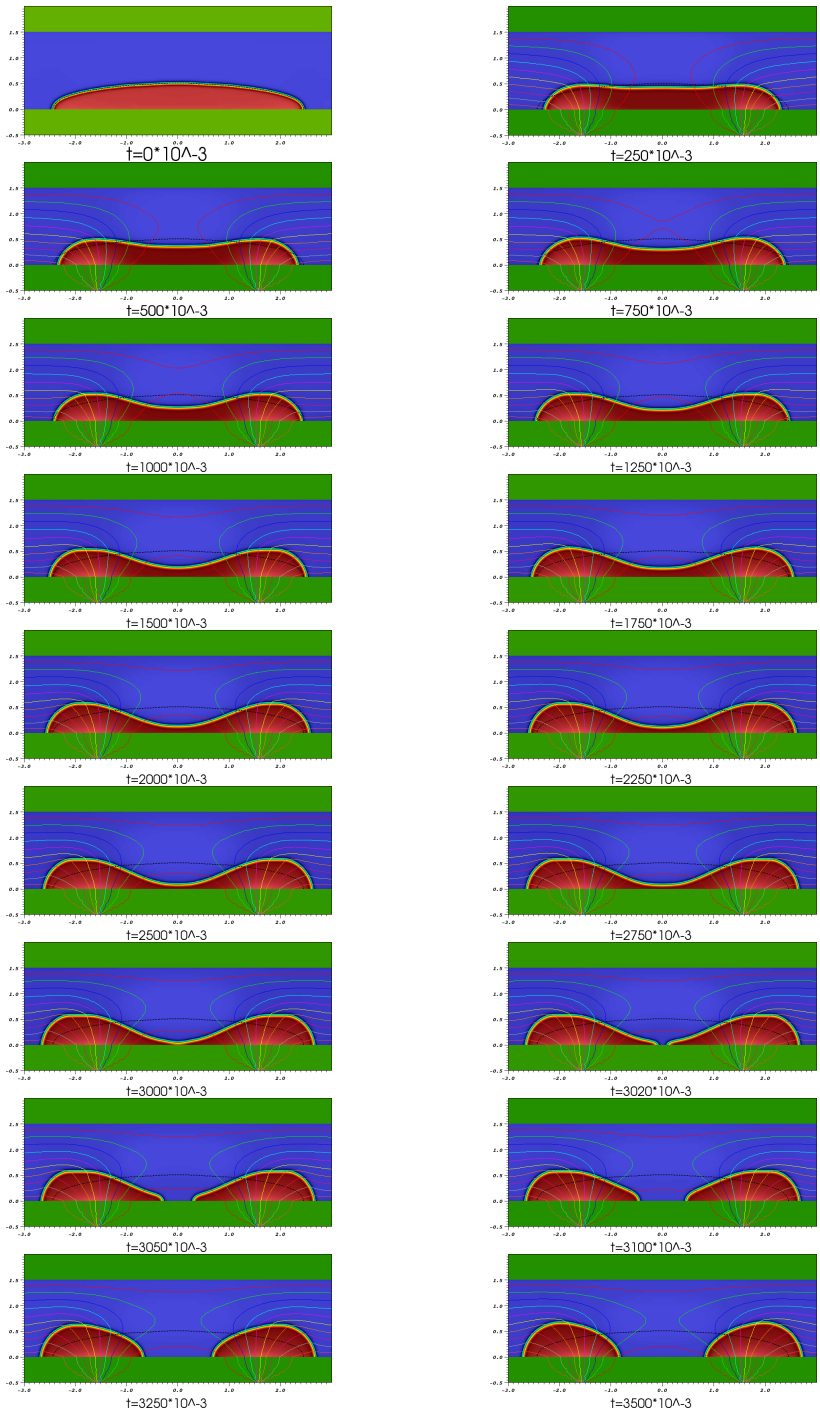

Figure 4.2: Splitting of a droplet under the action of an external voltage. The material parameters are $\rho_1/\rho_2 = 100$, $\eta_1/\eta_2 = 10$, $\gamma = 50$, $K_1/K_2 = 10$, $\varepsilon_1/\varepsilon_2 = 5$, $\varepsilon^\star/\varepsilon_2 = 100$, $M = 10^{-2}$, $\alpha = 10^{-3}$, $\beta = 10$, $\theta_s = 120°$, $\delta = 5 \cdot 10^{-2}$, $\lambda = 0.5$ and $V_{00} = 20$. The interface is shown at times 0, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.25, 2.5, 2.75, 3.0, 3.02, 3.05, 3.10, 3.25 and 3.5. Colored lines are used to represent the iso-values of the voltage. The black dotted line is the position of the interface at the beginning of the computations.

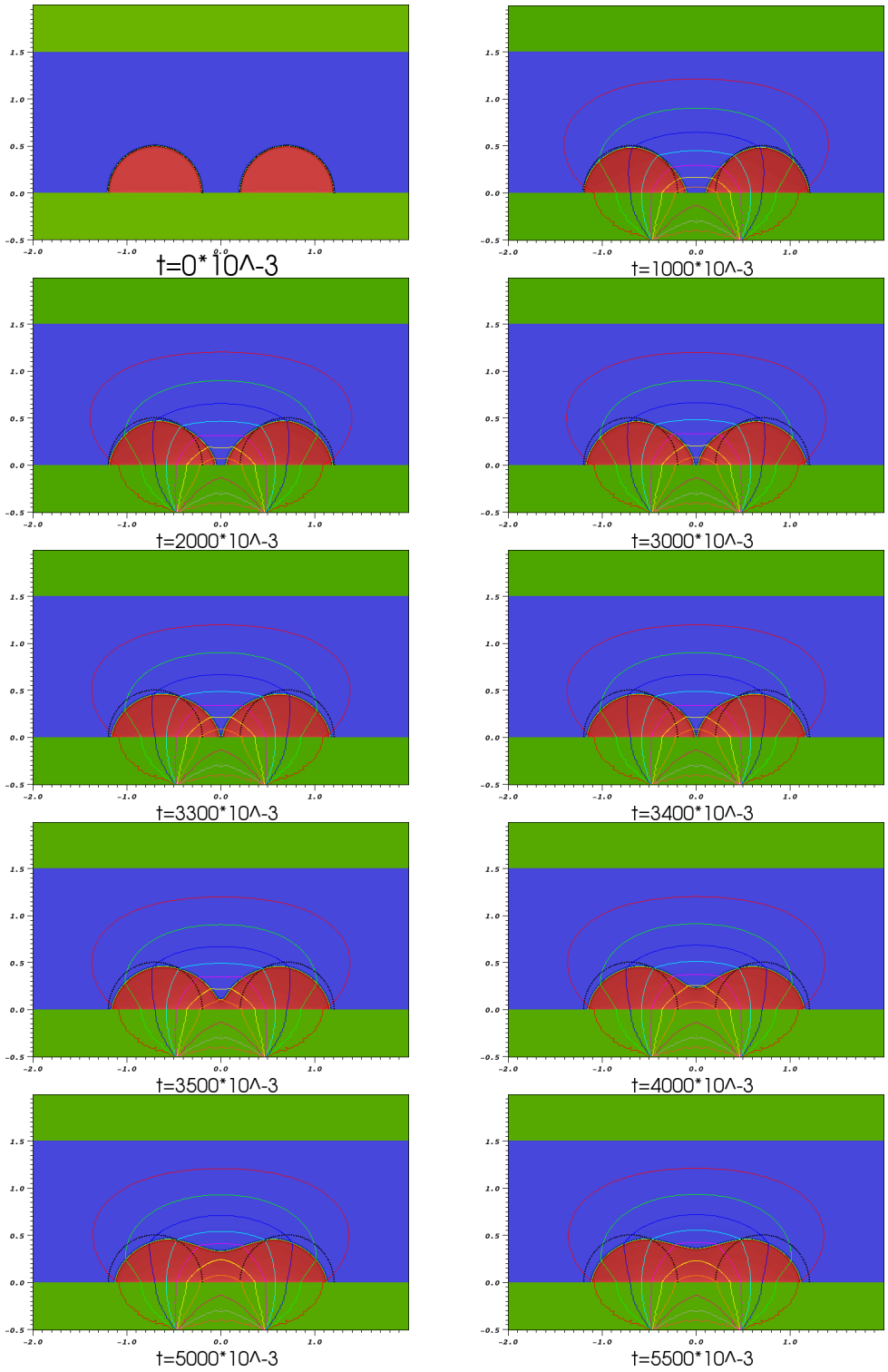

Figure 4.3: Merging of two droplets under the action of an externally applied voltage. The material parameters are $\rho_1/\rho_2 = 100$, $\eta_1/\eta_2 = 10$, $\gamma = 50$, $K_1/K_2 = 10$, $\varepsilon_1/\varepsilon_2 = 5$, $\varepsilon^\star/\varepsilon_2 = 100$, $M = 10^{-2}$, $\alpha = 10^{-3}$, $\beta = 10$, $\theta_s = 120°$, $\delta = 10^{-2}$, $\lambda = 0.5$ and $V_{00} = 20$. The interface is shown at times 0, 1, 2, 3, 3.3, 3.4, 3.5, 4, 5 and 5.5. Colored lines are used to represent the iso-values of the voltage. The black dotted line is the position of the interface at the beginning of the computations.

5 The program

5.1 Instructions for compilation and execution

The code must be compiled with the `deal.II` library, version 7.2 configured with `Trilinos`. Together with the code we have bundled a `Makefile` which is, basically, a copy of the large project file that the `deal.II` library provides and, thus, it can take the usual arguments (`clean`, `run`, etc.) We have added the option `doc`, which creates an online documentation for the program. The documentation can be accessed at `doc/doxygen/index.html`. The `Makefile` assumes that `deal.II` is located at `../../deal.II`. This location can be changed by modifying the variable `D` on line 33.

Three parameter files are included, one for each of the test cases presented here. Their names are `parameter-#.prm`, where `#` stands for `move/split/merge`. These are the parameters used in each of the experiments of Section 4. Their content is described below.

5.2 Handling the run-time parameters

To be able to specify the many physical and discretization parameters our problem has, as well as to be able to select a test case, we use a `ParameterHandler` class, as discussed in many of the tutorials of [Bangerth et al.]. The default parameter file as well as the meaning of each of the parameters is as follows:

Code

```
# Listing of Parameters
# ---------------------
# The test case: move merge split
set test_case = move
# How many time steps we print the solution.
set output  = 10
# The verbosity of the solution process.
set verbose = true
subsection Data solve charge
  # The stopping criterion.
  set eps         = 1e-6
  # The number of sweeps the AMG smoother does.
  set sweeps      = 3
  # The aggregation threshold for AMG.
  set threshold   = 1e-4
  # How often we update the preconditioner.
  set update_prec = 10
end
subsection Data solve penalty
  # The stopping criterion.
  set eps       = 1e-6
  # The number of sweeps the AMG smoother does.
  set sweeps    = 3
  # The aggregation threshold for AMG.
  set threshold = 1e-4
end
subsection Data solve phase
  # The size of the Krylov subspace to be used.
  set Krylov_size = 30
  # The stopping criterion.
  set eps         = 1e-6
  # The number of extra fills.
  set ilut_fill      = 50
  # The drop parameter.
```

```
35    set drop   = 1e-4
36    # How often we update the preconditioner.
37    set update_prec = 10
38  end
39  subsection Data solve velocity
40    # The size of the Krylov subspace to be used.
41    set Krylov_size = 60
42    # The stopping criterion.
43    set eps         = 1e-6
44    # The number of sweeps the AMG smoother does.
45    set sweeps      = 6
46    # The aggregation threshold for AMG.
47    set threshold   = 1e-3
48    # How often we update the preconditioner.
49    set update_prec = 10
50  end
51  subsection Data solve voltage
52    # The stopping criterion.
53    set eps         = 1e-6
54    # The number of sweeps the AMG smoother does.
55    set sweeps      = 3
56    # The aggregation threshold for AMG.
57    set threshold   = 1e-4
58    # How often we update the preconditioner.
59    set update_prec = 10
60  end
61  subsection Geometry Data
62    # The lower corner of the box. x-coordinate
63    set xl = -5
64    # The upper corner of the box. x-coordinate
65    set xm = 5
66    # The lower corner of the box. y-coordinate
67    set yl = 0
68    # The upper corner of the box. y-coordinate
69    set ym = 1
70    # The lower corner of the box. z-coordinate
71    set zl = 0
72    # The upper corner of the box. z-coordinate
73    set zm = 0
74    # The plate width
75    set width = 0.5
76  end
77  subsection Physical Data
78    # The conductivity of fluid 1.
79    set K_1          = 10
80    # The conductivity of fluid 2.
81    set K_2          = 1
82    # The mobility of fluid 1.
83    set M_1          = 0.01
84    # The mobility of fluid 2.
85    set M_2          = 0.01
86    # The parameter for CH.
87    set alpha        = 1e-3
88    # The slip coefficient of fluid 1.
89    set beta_1       = 10
90    # The slip coefficient of fluid 2.
91    set beta_2       = 10
92    # The interface thickness.
```



```
  set delta         = 0.05
  # The permitivity of fluid 1.
  set epsilon_1     = 5
  # The permitivity of fluid 2.
  set epsilon_2     = 1
  # The permitivity of the plates
  set epsilon_plate = 100
  # The final time of computation.
  set final_time    = 3.5
  # The surface tension coeffcient.
  set gamma         = 50
  # The initial time of computation.
  set initial_time = 0.
  # The regularization coefficient for the charge.
  set lambda        = 0.5
  # The density of fluid 1.
  set rho_1         = 100.
  # The density of fluid 2.
  set rho_2         = 1.
  # The static contact angle (in degrees).
  set theta_s       = 120
  # The viscosity of fluid 1.
  set viscosity_1   = 10
  # The viscosity of fluid 2.
  set viscosity_2   = 1
end
subsection Space discretization
  # Number of extra refines.
  set n_of_extra_refines   = 2
  # Number of global refines.
  set n_of_initial_refines = 3
  # The degree for the phase field space.
  set ph_deg               = 2
  # The polynomial degree for the pressure space.
  set pres_deg             = 1
  # The degree for the voltage space.
  set volt_deg             = 1
end
subsection Time step data
  # Time step size.
  set dt = 1e-3
end
```

6 Conclusions

In this work we have proposed a way to encapsulate the specifics of a problem in a class, as well as provided a way to handle the communication between problems. We applied these ideas to the implementation of a discretization scheme for the problem of electrowetting, as well as illustrated the performance of the developed code in a series of examples. We are confident that the simple ideas presented here are powerful enough to allow for rapid development of code. For instance, once this framework had been developed, a solver for a class of non-Newtonian fluids (which will be reported elsewhere) takes not more than a few hundred lines of code in which, basically, one needs to define the assembly of the local matrices.

Of course, much can be done to improve the ideas and the implementation itself. For instance, an implementation of these concepts in a distributed environment will be of interest to a wider audience. Their implementation using the `MeshWorker` concept provided by the library might

simplify the code. Concerning the ideas themselves, it would be interesting to extend this framework to problems posed in domains of different dimension, for instance a surface immersed in a bulk object, etc. We will leave this for future study.

Acknowledgement

AJS would like to express his gratitude to Patrick Sodré (University of Maryland) for his help in finding a subtle bug in the code that threatened the author's, already feeble, sanity.

References

S. Badia, A. Quaini, and A. Quarteroni. Splitting methods based on algebraic factorization for fluid-structure interaction. *SIAM J. Sci. Comput.*, 30(4):1778–1805, 2008a. ISSN 1064-8275. doi: 10.1137/070680497. URL `http://dx.doi.org/10.1137/070680497`.

S. Badia, A. Quaini, and A. Quarteroni. Modular vs. non-modular preconditioners for fluid-structure systems with large added-mass effect. *Comput. Methods Appl. Mech. Engrg.*, 197(49-50): 4216–4232, 2008b. ISSN 0045-7825. doi: 10.1016/j.cma.2008.04.018. URL `http://dx.doi.org/10.1016/j.cma.2008.04.018`.

W. Bangerth, R. Hartmann, and G. Kanschat. `deal.II` *Differential Equations Analysis Library, Technical Reference*. URL `http://www.dealii.org`. `http://www.dealii.org`.

Y. Bazilevs, V. Calo, T. Hughes, and Y. Zhang. Isogeometric fluid-structure interaction: theory, algorithms, and computations. *Comput. Mech.*, 43(1):3–37, 2008. ISSN 0178-7675. doi: 10.1007/s00466-008-0315-x. URL `http://dx.doi.org/10.1007/s00466-008-0315-x`.

B. Berge and J. Peseux. Variable focal lens controlled by an external voltage: An application of electrowetting. *European Physical Journal E*, 3(2):159–163, 2000.

P. Causin, J. F. Gerbeau, and F. Nobile. Added-mass effect in the design of partitioned algorithms for fluid-structure problems. *Comput. Methods Appl. Mech. Engrg.*, 194(42-44):4506–4527, 2005. ISSN 0045-7825. doi: 10.1016/j.cma.2004.12.005. URL `http://dx.doi.org/10.1016/j.cma.2004.12.005`.

S. Cho, H. Moon, J. Fowler, S.-K. Fan, and C.-J. Kim. Splitting a liquid droplet for electrowetting-based microfluidics. In *International Mechanical Engineering Congress and Exposition*, New York, NY, Nov 2001. ASME Press. ISBN: 0791819434.

S. Cho, H. Moon, and C.-J. Kim. Creating, transporting, cutting, and merging liquid droplets by electrowetting-based actuation for digital microfluidic circuits. *Journal of Microelectromechanical Systems*, 12(1):70–80, 2003.

A. Ern and J.-L. Guermond. *Theory and practice of finite elements*, volume 159 of *Applied Mathematical Sciences*. Springer-Verlag, New York, 2004. ISBN 0-387-20574-8.

V. Girault and P.-A. Raviart. *Finite Element Methods for Navier-Stokes Equations. Theory and Algorithms*. Springer Series in Computational Mathematics. Springer-Verlag, Berlin, Germany, 1986. ISBN 3-540-15796-4.

R. Hayes and B. Feenstra. Video-speed electronic paper based on electrowetting. *Nature*, 425 (6956):383–385, 2003.

R. Kornhuber and R. Krause. Adaptive multigrid methods for Signorini's problem in linear elasticity. *Comput. Vis. Sci.*, 4(1):9–20, 2001. ISSN 1432-9360. doi: 10.1007/s007910100052. URL `http://dx.doi.org/10.1007/s007910100052`.

R. Krause and B. Wohlmuth. A Dirichlet-Neumann type algorithm for contact problems with friction. *Comput. Vis. Sci.*, 5(3):139–148, 2002. ISSN 1432-9360. doi: 10.1007/s00791-002-0096-2. URL http://dx.doi.org/10.1007/s00791-002-0096-2.

R. Krause and B. Wohlmuth. Monotone multigrid methods on nonmatching grids for nonlinear multibody contact problems. *SIAM J. Sci. Comput.*, 25(1):324–347 (electronic), 2003. ISSN 1064-8275. doi: 10.1137/S1064827502405318. URL http://dx.doi.org/10.1137/S1064827502405318.

J. Lee, H. Moon, J. Fowler, T. Schoellhammer, and C.-J. Kim. Electrowetting and electrowetting-on-dielectric for microscale liquid handling. In *Sensors and Actuators, A-Physics (95)*, pages 259–268, 2002.

E. Lefrançois and J.-P. Boufflet. An introduction to fluid-structure interaction: application to the piston problem. *SIAM Rev.*, 52(4):747–767, 2010. ISSN 0036-1445. doi: 10.1137/090758313. URL http://dx.doi.org/10.1137/090758313.

P.-L. Lions. *Mathematical topics in fluid mechanics. Vol. 1. Incompressible models*, volume 3 of *Oxford Lecture Series in Mathematics and its Applications*. The Clarendon Press, Oxford University Press, New York, 1996. ISBN 0-19-851487-5.

G. Lippmann. *Ann. Chim. Phys.*, 5(494), 1875.

C. Liu and J. Shen. A phase field model for the mixture of two incompressible fluids and its approximation by a Fourier-spectral method. *Phys. D*, 179(3-4):211–228, 2003. ISSN 0167-2789. doi: 10.1016/S0167-2789(03)00030-7. URL http://dx.doi.org/10.1016/S0167-2789(03)00030-7.

R. Nochetto, A. Salgado, and S. Walker. A diffuse interface model for electrowetting on dielectric with moving contact lines. *M3AS*, 01(24), 2014. (accepted).

T. Richter and T. Wick. Finite elements for fluid-structure interaction in ALE and fully Eulerian coordinates. *Comput. Methods Appl. Mech. Engrg.*, 199(41-44):2633–2642, 2010. ISSN 0045-7825. doi: 10.1016/j.cma.2010.04.016. URL http://dx.doi.org/10.1016/j.cma.2010.04.016.

T. Roques-Carmes, R. Hayes, B. Feenstra, and L. Schlangen. Liquid behavior inside a reflective display pixel based on electrowetting. *Journal of Applied Physics*, 95(8):4389–4396, 2004a.

T. Roques-Carmes, R. Hayes, and L. Schlangen. A physical model describing the electro-optic behavior of switchable optical elements based on electrowetting. *Journal of Applied Physics*, 96 (11):6267–6271, 2004b.

F. Saeki, J. Baum, H. Moon, J.-Y. Yoon, C.-J. Kim, and R. L. Garrell. Electrowetting on dielectrics (ewod): Reducing voltage requirements for microfluidics. *Polym. Mater. Sci. Eng.*, 85:12–13, 2001.

A. Salgado. A diffuse interface fractional time-stepping technique for incompressible two-phase flows with moving contact lines. *M2AN Math. Model. Numer. Anal.*, 47(3):743–769, 2013.

T. Tezduyar, S. Sathe, and K. Stein. Solution techniques for the fully discretized equations in computation of fluid-structure interactions with the space-time formulations. *Comput. Methods Appl. Mech. Engrg.*, 195(41-43):5743–5753, 2006. ISSN 0045-7825. doi: 10.1016/j.cma.2005.08.023. URL http://dx.doi.org/10.1016/j.cma.2005.08.023.

