Horizon 2020

ROMSOC

Reduced Order Modelling, Simulation and Optimization of Coupled systems

Validation of Fluid-Structure Interaction Simulations in Membrane-Based Blood Pumps

Deliverable number: D5.2

Version 0.1

Funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 765374

Project Acronym:	ROMSOC
Project Full Title:	Reduced Order Modelling, Simulation and Optimization of Coupled systems
Call:	H2020-MSCA-ITN-2017
Topic:	Innovative Training Network
Type of Action:	European Industrial Doctorates
Grant Number:	765374

Editor:	Marco Martinolli, MOX, Dipartimento di Matematica, PoliMi
Deliverable nature:	Report (R)
Dissemination level:	Public (PU)
Contractual Delivery Date:	30/04/2020
Actual Delivery Date	19/03/2020
Number of pages:	16
Keywords:	Fluid-Structure Interaction, Blood Pumps, Model Validation, HQ curves
Methodology:	Coupling Methods
Authors:	**Marco Martinolli**, MOX, Dipartimento di Matematica, PoliMi **Christian Vergara**, LaBS, Dipartimento di Chimica, Materiali e Ingegneria Chimica, PoliMi **Luc Polverelli**, CorWave Inc.
Peer review:	ITMATI

Abstract

The benchmark consists in the validation of a numerical model for the fluid-structure interaction arising in membrane-based blood pumps against experimental data obtained by *in vitro* testings at CorWave Inc. The goal is to numerically reproduce the pump system under the same working conditions of the documented experimental sessions, in order to predict the pump outflow rates and the hydraulic power for different pressure conditions in the pump and finally compare the results with the experimental HQ curves. The software for the solution of the benchmark will be implemented in the C++ parallel library of finite elements LIFEV using the Extended Finite Element Method. The report includes the plan for the Docker installation of the LIFEV environment and the third-part packages, information on the future online availability of the software, the licences of use of the main libraries and the computer requirements to run the benchmark.

Disclaimer & acknowledgment
This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 765374.

Contents

List of Acronyms

CIP	Continuous Interior Penalty
DG	Discontinuous Galerkin
FSI	Fluid-Structure Interaction
HQ	Head pressure-Flow
LIFEV	LIbrary of Finite Elements V
LVADs	Left Ventricular Assist Devices
NS	Navier-Stokes
PDEs	Partial Differential Equations
XFEM	Extended Finite Element Method
3D	three dimensions

1. Introduction

Blood pumps are medical devices used to support cardiac function in patients affected by end-stage heart failure. These devices are implanted at the apex of the heart, where they expel the oxygenated blood collected in the left ventricle into the ascending aorta via a flexible outlet cannula. Hence, these devices are called Left Ventricular Assist Devices (LVADs). In this case, we will focus on a novel prototype of blood pumps that is under development at CorWave Inc.(Paris), said progressive wave blood pumps [1]. Progressive wave blood pumps are based on the interaction between an undulating elastic membrane and the blood, resulting in a pumping mechanism that ejects blood in a physiologic pulsatile regime without exerting high forces on the blood cells [2, 3]. In the left panel of Figure 1, we show the cross sectional view of the pump device that works as follows: the electromagnetic actuator at the center drives the oscillatory motion of the mobile magnet ring; the latter transfers the motion to the most external part of the membrane; as a consequence, the motion propates along the elastic membrane causing a wave-like displacement that propels the blood, coming from the inlet channel, towards the outlet. Thanks to this progressive wave propagation, the membrane can push and tranport the blood without causing blood trauma (unlike standard rotary blood pumps). Notice that, being a pump system, the pressure at the outlet is higher than at the inlet; therefore the role of the wave membrane is to win the adverse pressure gradient acting across the pump. The possibility to simulate this complex dynamics inside the pump allows to predict the device performance under different operating conditions (changing parameters like the amplitude or the frequency of the membrane oscillation or the velocity of the blood entering in the device) and in different pump designs (varying geometrical features, like membrane diameter or thickness, or inlet/outlet section). In addition, computational simulations can be used to predict blood adverse events, like hemolysis or thrombosis, and improve the hemocompatibility of the device.

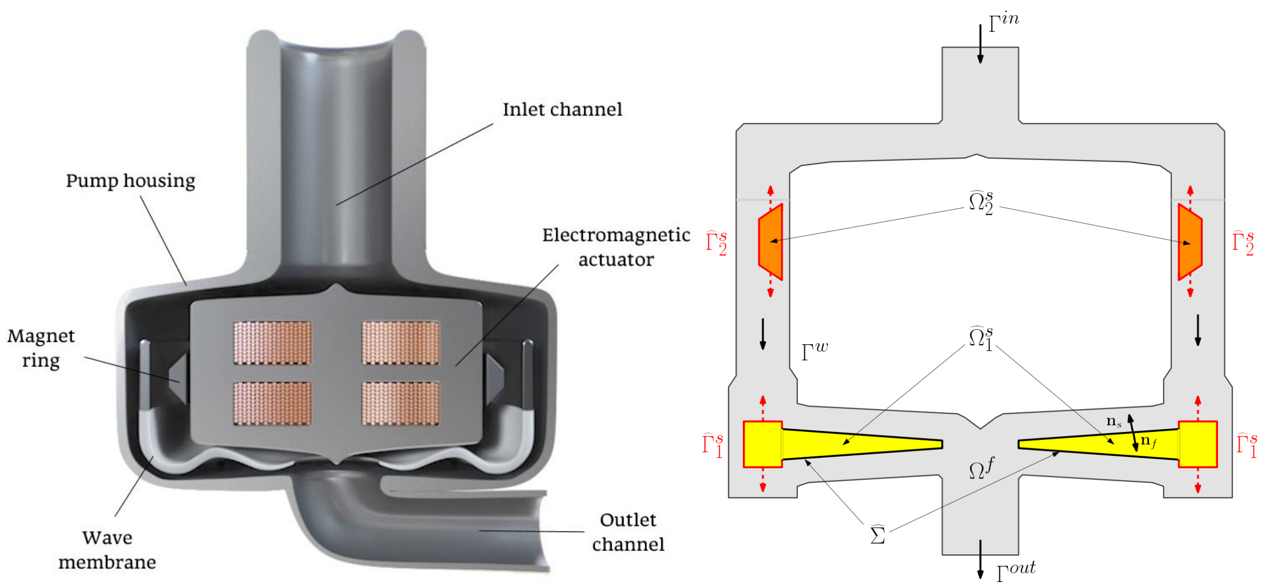

Figure 1: Left: Cross section of the implantable progressive wave pump. The blood enters from the superior inlet channel, it flows down along the sides of the central actuator body, it interacts dynamically with the wave silicon membrane and it is ejected into the inferior outlet channel. Right: Sketch of the pump domain.

Hence, from the mathematical point of view, the problem at hand is a time-dependant Fluid-Structure Interaction (FSI) problem with two moving structures: the wave membrane and the magnet ring. Therefore, we need to study the mutual interaction dynamics between the blood flow and the two structures. In particular, we are interested to check computationally the membrane wave pumping technology under different working conditions of the pump device. The operating point of the pump is identified by three parameters: (i) the excitation frequency f of the magnet ring and the membrane; (ii) the amplitude Φ of their oscillations; and (iii) the pressure difference $\Delta P < 0$ between the pump inlet and the outlet.

However, notice that the solution of this problem can be very tricky. In particular, during the wave motion of the membrane, the structure approaches very closely to the pump housing, potentially reaching a contact configuration with the surrounding pump walls. Therefore, our solution for the benchmark needs to address the possible conditions and the numerical issues arising during the mesh deformation in standard fitted methods for FSI problems. In the next section, we will propose a possible solution to these problems, based on the unfitted Extended Finite Element Method (XFEM).

The benchmark consists in the validation of a numerical method to solve the FSI problem in membrane-based blood pumps against real hydraulic data provided by CorWave Inc. Specifically, the goal is to predict the outflow volume rate Q^{sim} given a certain operating point of the pump $(f, \Phi, \Delta P)$ and compare the numerical result with the measured flow data Q^{data}.

The remaining of the report is structured as follows: in Section 2, we describe the FSI model, with a focus on its hierarchical features, and we propose a numerical strategy to solve the benchmark; in Section 3, we detail the required numerical libraries and software packages and the installation concept via Docker; in Section 4, we report the discuss the technical details to run the benchmark, such as environment settings, hardware and sofware requirements; and finally in Section 5, we present the structure and the format of the experimental data provided by CorWave Inc. (5.1), we depict the plan for the benchmark for the model validation (5.2). Licences of use and configuration files can be found in the Apendix A.

2. Mathematical Formulation

2.1. The Fluid-Structure Interaction Model

The intertwined dynamics arising inside a progressive wave pumps can be mathematically described in the framework of FSI modeling, where a system of Partial Differential Equations (PDEs) describes separately the behaviour of the fluid and of the structure in the respective domains, while proper coupling conditions define their interaction at the interface. In the following, we simplify the formulation of the problem by considering a unique structural domain $\Omega^s = \Omega^s_1 \cup \Omega^s_2$, where Ω^s_1 is the membrane domain and Ω^s_2 is the magnet ring domain.

The FSI problem reads as follows: for each time $t > 0$, find fluid velocity and pressure0 $(\mathbf{u}(t), p(t))$ in the fluid domain $\Omega^f(t)$ and membrane displacement $\widehat{\mathbf{d}}(t)$ in the reference structure domain $\widehat{\Omega}^s = \Omega^s(0)$, such that:

$$\rho_f\left(\partial_t \mathbf{u} + \mathbf{u} \cdot \nabla \mathbf{u}\right) - \nabla \cdot \mathbf{T}^f(\mathbf{u}, p) = \mathbf{0} \qquad \text{in } \Omega^f(\mathbf{d}^f), \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0 \qquad \text{in } \Omega^f(\mathbf{d}^f), \tag{2}$$

$$\rho_s \partial_{tt} \widehat{\mathbf{d}} - \nabla \cdot \widehat{\mathbf{T}}^s(\widehat{\mathbf{d}}) = \mathbf{0} \qquad \text{in } \widehat{\Omega}^s, \tag{3}$$

$$\mathbf{d}^f = \mathbf{d} \qquad \text{on } \Sigma(\mathbf{d}^f), \tag{4}$$

$$\mathbf{u} = \partial_t \mathbf{d} \qquad \text{on } \Sigma(\mathbf{d}^f), \tag{5}$$

$$\mathbf{T}^f(\mathbf{u}, p)\, \mathbf{n} = \mathbf{T}^s(\mathbf{d})\, \mathbf{n} \qquad \text{on } \Sigma(\mathbf{d}^f). \tag{6}$$

with proper boundary and initial conditions. Equations (1) and (2) are the Navier-Stokes (NS) equations modeling the conservation of momentum and mass of an incompressible viscous Newtonian fluid. Equation (3), written in the Lagrangian formulation with the quantities defined in the reference configuration ($\widehat{\cdot}$ supercript), is the elastodynamic equation that models the structure deformation. Equation (4) provides the adherence condition between the fluid and the solid domains, and Equations (5) and (6) guarantee the continuity of velocity and of stresses respectively at the fluid-structure interface Σ. Notice that both the fluid domain Ω^f and the interface Σ depend over time on the fluid displacement $\mathbf{d}^f$ and thus on the structure displacement $\mathbf{d}$ due to the geometric condition (4). In particular, the physical quantities presented in the mathematical formulation are:

- ρ_f and ρ_s are the mass densities of fluid and structure;
- $\mathbf{n}^f$ and $\mathbf{n}^s$ are the external normal vectors from fluid and structure domains, satisfying $\mathbf{n}^f = -\mathbf{n}^s = \mathbf{n}$;
- $\mathbf{T}^f(\mathbf{u}, p) = -p\mathbf{I} + 2\mu_f \mathbf{D}(\mathbf{u})$ is the Cauchy stress tensor for a viscous Newtonian fluid with dynamic viscosity μ_f and symmetric operator $\mathbf{D}(\mathbf{w}) = \frac{1}{2}(\nabla \mathbf{w} + \nabla \mathbf{w}^T)$
- $\widehat{\mathbf{T}}^s(\hat{\mathbf{d}})$ is the first Piola-Kirkhhoff tensor of the structure, such that $\widehat{\mathbf{T}}^s(\widehat{\mathbf{d}}) = J\,\mathbf{T}^s\,(\mathbf{d})\mathbf{F}^{-T}$ with $\mathbf{T}^s$ being the solid Cauchy stress tensor depending on the constitutive law of the structure, $\mathbf{F} = \nabla \mathbf{x}$ the gradient of deformation and $J = \det \mathbf{F}$ its determinant. According with previous works [4], we assume linear elasticity for both structures, so that we can use the Hooke Law and write the Cauchy tensor as $\mathbf{T}^s(\mathbf{d}) = \lambda^s\,(\nabla \cdot \mathbf{d})\,\mathbf{I} + 2\mu^s\,\mathbf{D}(\mathbf{d})$, where λ^s and μ^s are the Lamé parameters for each material.

In the right panel of Figure 1, we see the mathematical domain of the blood pump. We specify that the geometrical differences with respect to the view in the right panel of Figure 1 are due to the fact that we work on an alternative design of the pump. In addition we have omitted the rigid titanium apparatus that connects the magnet ring to the membrane, because it is not significant for our analysis. Indeed, we apply the oscillatory motion imposed by the actuator directly on the structure boundaries Γ_1^s and Γ_2^s.. Hence, the imposed oscillations are modeled by means of a sinusoidal Dirichlet conditions on $\Gamma^s = \Gamma_1^s \cup \Gamma_2^s$ as:

$$\mathbf{d}\,(t) = \Phi\,\sin\,(2\pi\,f\,t)\,\mathbf{e}_z \qquad \text{on } \Gamma^s \tag{7}$$

where Φ and f are respectively the amplitude and the frequency of the forced oscillation of both the membrane and the magnet ring.

In addition we have the boundary conditions for the fluid subproblem. We use a pair of Neumann conditions to prescribe the pressure conditions at the inlet Γ^{in} and at the outlet Γ^{out} of the pump domain and a non-slip condition at the pump walls Γ^w. In fact, we have that:

$$\mathbf{T}^f(\mathbf{u}, p)\,\mathbf{n}^f = \Delta P\,\mathbf{n}^f \qquad \text{on } \Gamma^{in}, \tag{8}$$

$$\mathbf{T}^f(\mathbf{u}, p)\,\mathbf{n}^f = \mathbf{0} \qquad \text{on } \Gamma^{out}, \tag{9}$$

$$\mathbf{u} = \mathbf{0} \qquad \text{on } \Gamma^w, \tag{10}$$

where $\Delta P < 0$ is the negative pressure parameter that defines the pressure difference between the inlet and the outlet (having set the zero pressure level at the outlet).

We finally consider null initial conditions for both velocity and displacement variables.

2.2. Numerical Method

In our solution of the benchmark, we will try to numerically reproduce the experimental results mimiking the same working conditions of the pump in the experimental setup and solving the mathematical problem above in three dimensions (3D) using the XFEM [5, 6]. XFEM is an unfitted technique which has two main advantages compared with other approaches for FSI problems: i) since the fluid mesh is kept fixed on the background, it avoids the remeshing procedure normally occurring in case of element distortion; ii) the accuracy of the solution is maintained at the interface, thanks to the local enrichment of the functional space of the extended finite elements. However, since the structure mesh moves in the foreground cutting the underlying fluid mesh, XFEM requires to compute the mesh intersections at each time istant to identify the fluid elements that are cut in multiple subportions (called *split elements*), leading to a higher computational cost. For more details on the numerical formulation of the XFEM with a Discontinuous Galerkin (DG) mortaring at the interface, the reader can find an exaustive explanation in the reference paper [7].

We linearize the fluid subproblem by using the semi-implicit approach for the convective term. In addition, in order to solve efficiently the geometric coupling, we take the first order extrapolation of the fluid domain from previous timestep. We use first order finite elements for all variables and we consider the Continuous Interior Penalty (CIP) stabilization to satisfy the inf-sup condition [8]. We also apply a ghost penalty stabilization to solve the spurious oscillations coming from XFEM, when the size of the split elements becomes too small [9].

The linear system obtained at each timestep after the XFEM-based discretization is solved with a monolythic approach with a GMRES solver preconditioned by a block Gauss-Seidel preconditioner.

2.3. Hierarchical Modeling

Due to the multi-component and multiphysics nature of our problem, the strong coupling of the physical system and the computational needs of the unfitted method [10], most of the times the solution of the fluid-structure dynamics can be very tricky and the complexity of the problem can represent a real barrier to in-depth analysis of the system.

However, in our application, there exist several modeling options, both at the geometric and numerical level, that can enable faster computations but at the cost of a decreased accuracy, and viceversa. In fact, the FSI model of our study is actually the result of a sequence of modeling options (e.g. geometry detail, modeling assumptions, numerical approximations, etc.), that act on distinct levels and that contribute in different ways to the global complexity of the model. Hence, the model is considered to be hierarchical. This means that we can tailor the model to fit the needs of our study by acting independently of multiple levels, requiring, for instance, a high level of detail of geometric features but relaxing specific numerical coupling conditions.
In the following, we provide the hierarchical levels that we identified in the FSI problem in progressive wave blood pumps.

2.3.1. Geometry of the domain

The complete domain of the blood pump is composed by several subcomponents: the pump housing, the membrane assembly, the actuator, the magnet ring and the membrane frame (see Figure 1). It is clear that model all the components and their interaction can be very complicated and, in most cases, not necessary. Therefore, it is possible to reduce the complexity of the geometry of the domain by omitting the rigid components or either by simplifying the geometrical features of the system.
For instance, if we are interested in the study of the fluid-structure dynamics between the blood and the wave membrane, we can limit the computational domain to the so-called pump head region. Therefore, we can simply cut the domain limiting ourselves to the study of the lower part of the pump (see blue square in Figure 2). In this way, we heavily reduced the number of nodes and consequently the computational time for the simulations. Alternatively, we can reduce the geometric detail of the pump or the membrane, so that the mesh intersection step in the XFEM is easier and faster to solve.

2.3.2. Meshing

Another standard way to modulate the complexity of the problem is by working at the mesh step h of the computational domain. Indeed, you can either choose to have a coarse mesh (large mesh step h) to have fast simulations with small accuracy or finer mesh (small mesh step h) to get accurate results at higher computational cost. In fact, when you decrease the discretization step h, you automatically increase the number of degrees of freedom of the problem and consequently the dimension of the linear system. In case of unfitted

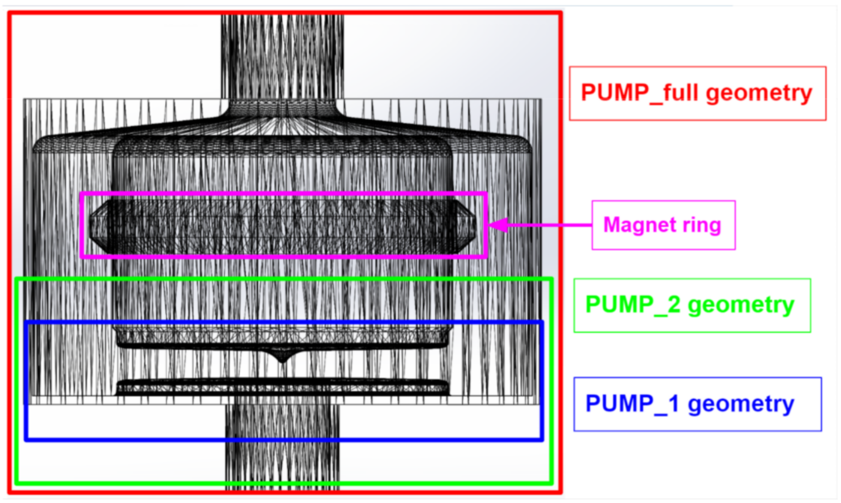

Figure 2: Different levels of geometric extensions of the blood pump domain. Blue and green boxes identifiy reduced pump domains; the red box identifies the full pump geometry, including the inlet and the outlet channels and the magnet ring (violet).

methods, the choice of the discretization step is even more important, because it affects as well the number of fluid elements that are split by the structure mesh resulting in a higher complexity of the mesh intersection step.

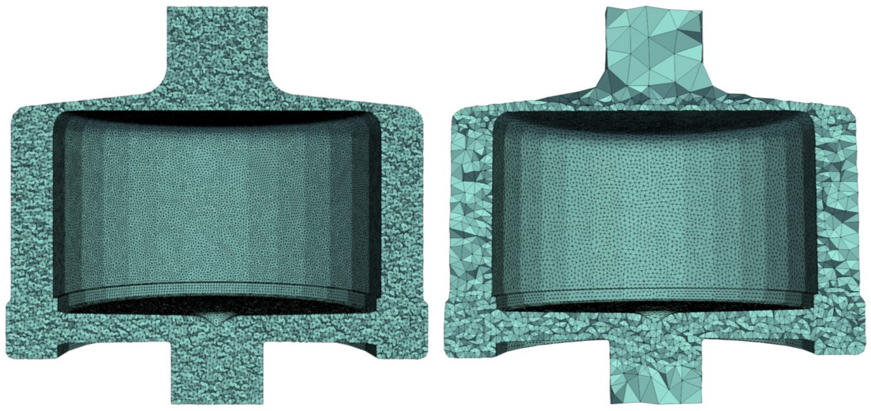

Figure 3: Different levels of meshing of the fluid mesh. Left: uniform meshing with small meshsize. Right: coarse fluid mesh with local refinement in the specific regions of interest.

In case of the multi-component systems like our blood pump, it is possible to make more sophisticated choices of the mesh step depending on the different properties of specific regions. In fact, instead of having a uniform mesh (same h in the whole domain), once can start from a coarse mesh and refine the meshing in the regions where we expect the dynamics are more complex or more interesting for our study - like in the proximity of the membrane or the magnet ring structures - or in regions that present geometrical peculiarities (like narrowing of the flow path or sharp angles), as shown in Figure 3.

2.3.3. Geometric coupling

In FSI problems we have two types of coupling conditions at the interface: physical and geometrical coupling conditions. Physical conditions impose the continuity of the velocities (kinematic condition) and of the normal stresses (dynamic condition) at the fluid-structure interface. The geometric condition requires that the adherence of the fluid and the solid domain is maintained during their motion in time, without the creation of inner holes. The latter coupling condition is normally formulated by asking that the displacement of the structure $\mathbf{d}$ is equal to the fluid domain displacement $\mathbf{d}_f$ in all the points of the interface (see Equation 4). We want to remind the reader that in the XFEM framework the fluid mesh is fixed on the background and the fluid domain corresponds to the sole region that is not overlapped by the structure mesh moving on the foreground. Therefore, the fluid domain changes in time according with $\mathbf{d}_f$, that depends directly on the motion of structure by means of the coupling condition.

In XFEM applications the satisfaction of the geometric adherence condition is actually a source of high computational costs. In fact, the fluid domain displacement $\mathbf{d}_f$ has to be computed from the intersection of the moving structure mesh with the underlying fluid mesh. At each time instant, the structure moves in the space region overlapping the fluid mesh in different positions and thus the mesh intersection step has to be repeated at each iteration. The costs for this operation are definitely not negligible and they increase with the refinement of both meshes.

As a consequence, in our study we propose the so-called *fixed geometry approximation*, for which, in case of small displacement regime ($\mathbf{d}$ small and bounded), we neglect the fluid domain displacement $\mathbf{d}_f$. This means that the fluid domain is actually not updated at each timestep, but it is approximated to remain fixed to its original configuration. Therefore, the mesh intersection step has to be performed only once during the first time iteration, saving a lot of computational time. We want to emphasize that the displacement of the structure $\mathbf{d}$ is still computed and it is still active for the physical coupling conditions; we only neglect its effect as modifier of the fluid domain. Thus, the fixed geometry approximation can be a smart way in case of small displacement regime, to decrease the computational cost and avoid the additional complications in the splitting of the fluid elements coming from atypical geometrical configurations.

A second level of approximation relies on moving the fluid domain, but in an explicit way, by using extrapolation of information at previous time steps. This strategy will introduce an error with respect to the fully implicit geometric coupling, which is however smaller than that produced by the previous strategy, at the price of an increased computational cost.

2.3.4. Modeling the Contact

We consider to have contact when two solid bodies are sufficiently close so that their motions are mutually affected by each other. Therefore it is not strictly necessary that the two bodies touch each other to have contact and under some conditions proper impact cannot even occur. In our application, contact can occur between the moving membrane and the walls of the pump housing. Thus, only one of the two bodies is mobile, simplifying the implementation of the contact condition. Such contact-like situation is actually very important for a correct pumping performance for two reasons: i) near the contact region, the membrane wave generates a pressure gradient that propels the blood towards the outlet, ii) the contact between the membrane and the walls works as a valve that avoids the backflow from the outlet towards the inlet (see Figure 4).

Therefore, in our study we can add an additional level complexity by investigating the contact between the membrane and the pump wall, finding a way to prohibit the penetration of the former in the latter.

In FSI problems, a possible approach to model the contact derives directly from the combination of non-slip coupling interface conditions and the incomprimibility of the fluid. In fact, previous theoretical studies [11] proved that collisions between a body moving in a viscous incompressible fluid and a wall can never occur in finite time. This is justified by the fact that the incompressible fluid opposes a resistance to the motion of the

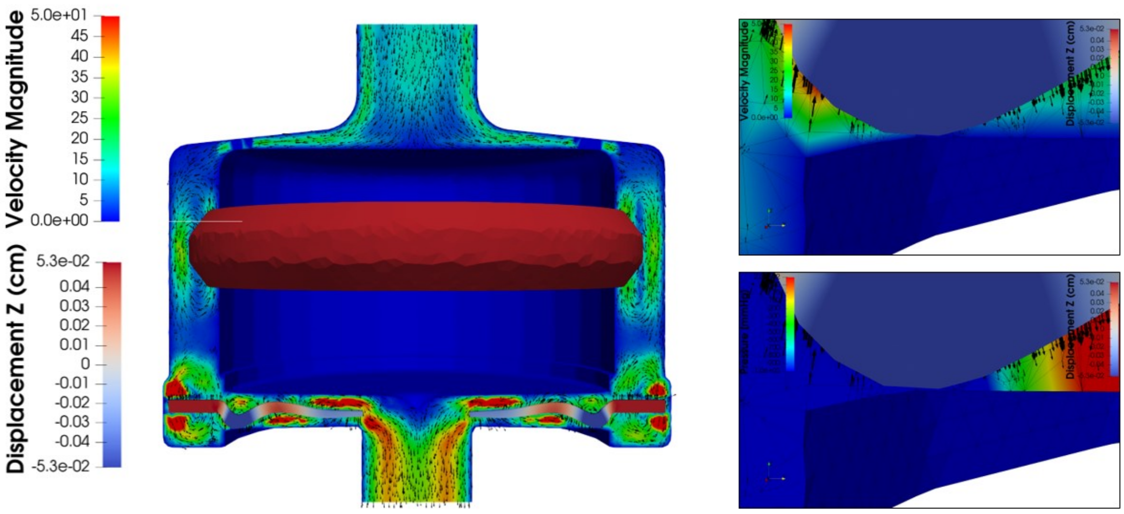

Figure 4: Detail of the contact point between the membrane and the lower pump wall. Left: velocity field in the whole pump domain. Right: zoom on the velocity (above) and pressure (below) fields in the contact area.

structure with an intensity that increases more and more they get closer. Since there is no possibility to have tangential slip at the interface, then there will always remain a small layer of fluid separating the two bodies. This is confirmed by physical and numerical studies. This approach has the advantage that it is simple to implement, but it requires to have small discretization steps both in space and in time, in order that the model can capture the dynamics generated when the membrane approaches the wall.

Alternatively, we can relax the coupling conditions by allowing the tangential slip at the interface in order to have contact. In case of slip coupling conditions, the kinematic condition requires the continuity of the velocities only in normal direction, without constraints in the tangential one. Under these conditions, contact can formally occur and we need to add a new term to the weak formulation acting on the portion of the boundary that is proximal to the wall [12]. This additional term represents the contact force exerted on the membrane in case of contact. This more sophisticated approach needs to identify the portion of the membrane boundary where to apply the contact force and calibrate the intensity of such force by tuning a penalty contact parameter.

As a result, we have identified four independent hierarchical levels of our FSI model, that can be set in order to build up simulations with a certain complexity, depending on the degree of accuracy or the computational time of our study. In our solution for this benchmark we decided to consider the full pump geometry because the experimental data referred to the whole pump domain, but we made small changes to the geometry, such as the omission of the rigid holder apparatus and of a small step in the most external region of the membrane. We considered a uniform mesh size for the background mesh, while for the membrane mesh we refined the mesh step h more and more approaching the tip region, where the membrane thickness is smaller. Concerning the geometric coupling we have solved it by taking the first order extrapolation of the fluid domain from the previous timestep. And finally for the contact subproblem, we assumed non-slip condition both at the interface and at the pump walls, in agreement with experimental observations in similar problems [13].

3. Implementation

The software for the benchmark will be implemented in LIbrary of Finite Elements V (LIFEV) (https://bitbucket.org/lifev-dev/lifev-env.git) [14], a C++ parallel finite element library for the solution of PDEs. In particular, the library is suitable for solving real problems in the field of cardiovascular applications. The XFEM module requires external libraries to handle the geometric intersections between the fluid and the solid meshes and the treatment of the split elements. Therefore, Triangle (version 1.6) and TetGen (version 1.15.0) [15] are used to mesh the polihedra generated by the cut of the fluid mesh in 2D and 3D respectively. In particular, the source code of TetGen library has been modified to make it compliant with the LIFEV environment. The licence of use is reported in the Appendix A.1 for each of these libraries.

Despite a release version of LIFEV is available online (accessible with a free bitbucket account), it does not include the module for the XFEM numerical approach that is needed for the benchmark. The branch of the library including the XFEM method will be made public soon, so that the codes will be open access.

LifeV requires some third-part libraries to be built. In particular, the following packages are required:

- cmake (latest version): to configure and create the necessary makefiles for building the source code;
- openblas (v. 0.2.17): low level linear algebra package, providing both BLAS and LAPACK bindings;
- trilinos (v. 11.14.3): parallel linear algebra;
- metis (v. 5): graph partitioning library;
- parmetis (v. 4.0.3): parallel version of metis;
- hdf5 (v. 1.8.16): management of the HDF5 file format for storing data;
- suitesparse (v. 4.5.1): linear algebra library with linear solvers and utilities.

The software will be made publically available soon on Bitbucket. Moreover, in order to simplify the configuration of the system and increase the cross-platform compatibility with other benchmarks in ROMSOC, the installation of the LIFEV environment and the necessary third-part libraries will be set via Docker. Therefore free Docker and Bitbucket accounts will be necessary to run the software to run the benchmark.

Here we present the general idea of the installation procedure, detailing in the following list the steps required for the Docker installation ansd the configuration procedure:

1. Set the Docker container with the required modules installed by typing in the terminal (it requires docker):

```
docker build -f Dockerfile
```

where the Dockerfile contains the instructions as follows:

```
# Use Ubuntu 16.04 as parent image
FROM ubuntu:xenial

# Install LifeV dependencies
RUN apt-get update && \
apt-get install -y \
g++ \
cmake \
git \
libblacs-mpi-dev \
libscalapack-mpi-dev \
libsuitesparse-dev \
trilinos-all-dev \
libboost-program-options-dev \del
libparmetis-dev \
libmetis-dev \
libhdf5-openmpi-dev \
libmumps-dev \
libsuperlu-dev \
libtbb-dev \
libptscotch-dev \
binutils-dev \
libiberty-dev \
libtriangle-dev && \
groupadd -r lifev && \
useradd -l -m -g lifev lifev
# Set user 'lifev'
USER lifev
# Set working directory
WORKDIR /home/lifev
# Copy the content of the current dir into the WORKDIR
ADD --chown=lifev:lifev . /home/lifev
```

Listing 1: Dockerfile

2. Define the paths inside the container as in the *defs.sh* file below:

```
#!/bin/bash
LifeV_DIR=$PWD
LifeV_SRC=$LifeV_DIR/lifev-src
LifeV_BUILD=$LifeV_DIR/lifev-build
LifeV_LIB=$LifeV_DIR/lifev-install
TetGen_DIR=$LifeV_DIR/tetgen1.5.0
mkdir -p $LifeV_SRC
mkdir -p $LifeV_BUILD
mkdir -p $LifeV_LIB
```

Listing 2: defs.sh

3. Clone the source codes of LIFEV and TetGen from xx bitbucket repository into the container by executing `./clone.sh` file:

```
#!/bin/bash
source defs.sh
git clone lifev-xfem_REPOSITORY ${LifeV_SRC}
git clone tetgen4lifev_REPOSITORY ${TetGen_DIR}
```

Listing 3: clone.sh

Notice that lifev-xfem_REPOSITORY and tetgen4lifev_REPOSITORY will be the url to the public repository of the LIFEV software and to the modified version of TetGen library.

4. Build TetGen and configure LIFEV typing `./config.sh` from the container:

```
#!/bin/bash
source defs.sh
cd $TetGen_DIR
make tetlib
cd $LifeV_BUILD
cmake \
-D BUILD_SHARED_LIBS:BOOL=OFF \
-D CMAKE_BUILD_TYPE:STRING=RELEASE \
-D CMAKE_INSTALL_PREFIX:PATH=${LifeV_LIB}\
-D CMAKE_C_COMPILER:STRING="mpicc" \
-D CMAKE_CXX_COMPILER:STRING="mpicxx" \
-D CMAKE_CXX_FLAGS:STRING="-O3 -msse3
    -Wno-unused-local-typedefs -Wno-literal-suffix"\
-D CMAKE_C_FLAGS:STRING="-O3 -msse3" \
-D CMAKE_Fortran_COMPILER:STRING="mpif90" \
-D CMAKE_Fortran_FLAGS:STRING="-Og -g" \
-D CMAKE_AR:STRING="ar" \
-D CMAKE_MAKE_PROGRAM:STRING="make" \
[...]
${LifeV_SRC}
cd $LifeV_DIR
```

Listing 4: config.sh

Here the *config.sh* is shown in a shorter form with the most important compiling options (`mpicc` compiler for C, `mpicxx` compiler fo C++, `O3` optimization). For a complete version with all compiling options and dependancies we refer the reader to Appendix A.2).

5. Build LIFEV from the container by executing `./build.sh`:

```
#!/bin/bash
source defs.sh
cd $LifeV_BUILD
make -j 3
cd $LifeV_DIR
```

Listing 5: build.sh

4. Computer Requirements

The operative system has to be UNIX-based (like Oracle Solaris, Darwin or macOS), equipped with C++ compiler and all basic system libraries required for software development, like FORTRAN, C++, HDF5 or MPI. It is sufficient to type on the terminal

```
sudo apt-get install build_essentials
```

, to check if your operative system already has such fundamental libraries or, in case they are not present, install them.

The minimum requirements for the hardware are:

- CPU: 1 processor is sufficient to run the benchmark, but multi-thread CPU is needed in case you want to employ parallel computing;
- rigid disk: at least 3 GB of available space in the rigid disk, considering the memory required for building lifev and all third-part libraries, as well as the space required by data and results;
- RAM: a minimum of 40 GB of RAM is predicted to be required the benchmark in serial. Be aware that in case of parallel runs, the demand of RAM increases.

5. Numerical Example

5.1. Experimental Data

The experimental data used for the benchmark are Head pressure-Flow (HQ) curves, which are standard pump performance curves obtained during the *in vitro* testings of the pump system in a static pipe loop machine. Blood is replaced by a mixture of water and glycerin (39% in weight). The bench system is equipped with pressure sensors placed both at the inlet and at the outlet sections, to measure the pressure gradient arisen over the pump, or head pressure, and an ultrasonic flowmeter clamped at the outlet pipe, to measure the outflow volume rate of the pump. The hydraulic resistance in the circuit is varied by means of centrifugal pumps placed in series with the blood pump, that can work in favor or against it. Therefore, the HQ curves are obtained by diplaying the different measurements in the pressure-flow plan for the different flow conditions. Figure 5 shows the HQ curves of the pump system when the frequency of oscillation of the membrane is fixed to 120 Hz, while the amplitude of oscillation - that is a function of the input voltage V_p of the pump actuator - passes from 0.53 mm (grey triangles), to 0.63 mm (yellow crosses) and finally to 0.73 mm (blue dots).

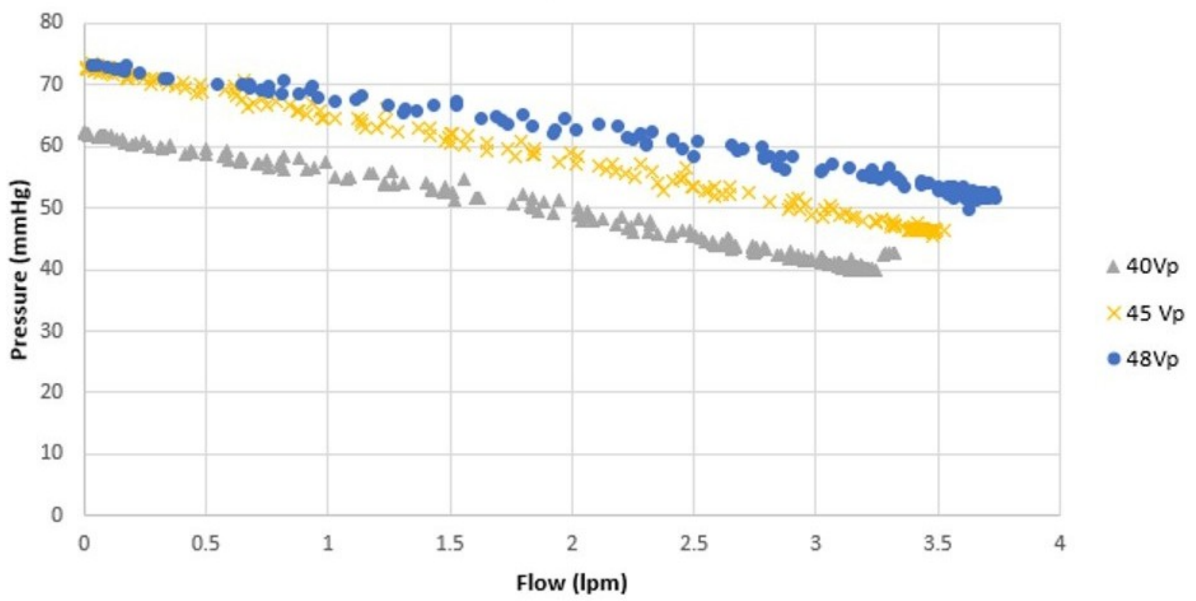

Figure 5: HQ experimental curves for three different values of the voltage of the actuator.

Raw data are provided in Excel format for each experiment session and will be made publically available in open-source spreadsheet applications. Each document consists of three separate data columns with the measurements of:

- head pressure H, corresponding to mean pressure difference between the outlet and the inlet, expressed in millimeters of mercury [mmHg] (1 mmHg = 1333.22 g /cm s^2).
- outflow volume rates Q, expressed in liters per minute [lpm] (1 lpm = 0.06 cm^3/s).

The software uses quantities expressed in the *cgs* system.

A separate table explicits the settings of each experimental sessions, including in particular the correspondence between the values of the input voltage of the actuator V_p, expressed in Volt [V], shown in the Figure above, and the respective amplitude of the induced membrane vibrations Φ, expressed in millimeters [mm].

5.2. Benchmark Plan

The aim of the proposed benchmark is to validate the FSI model via comparison of the experimental HQ curves with the numerical results. Analogously to the approach for model validation used by Perschal et al. in a similar scenario [4], the pump system is simulated under the same operative conditions of the experimental sessions, so that we can measure the error between the real data and the predicted quantities and quantify the goodness of the model. In particular, we set the pressure difference betweeen the pump outlet and inlet, so that we can predict the corresponding pump outflow rate and compare the numerical result with its experimental counterpart.

The plan for the model validation is described in the Algorithm 1. Specifically:

1. First, the experimental data coming from the HQ curves are stored in separated variables: H_data collects the data of the head pressure between outlet and inlet, Q_data contains the measurements of the corresponding outflow volume rate, and M_data stores the input settings for the parameters of membrane displacement; moreover, additional physical parameters are contained in the settings variables (line 1).
2. Only N data points are extracted by the curves to be reproduced via simulation and used for the model validation (line 2).

Algorithm 1 Benchmark structure: Model validation against N data points.

```
1: [Q_data, H_data, M_data] = read_data(HQcurves, phys_settings);
2: [Q_Ndata, H_Ndata, M_Ndata] = extract_Ndata($N$);          // $N \geq 1$ data points
3: for n = 1 : N do
4:     input_pressure = H_Ndata[n];
5:     input_frequency = M_Ndata[n].f;
6:     input_amplitude = M_Ndata[n].phi;
7:     define_bc(input_pressure, input_frequency, input_amplitude);   // Boundary conditions
8:     [$\mathbf{u}_0$, $\mathbf{d}_0$] = initialization()
9:     while $t < T_{max}$ do                                   // Time loop
10:        [$\mathbf{u}_t$, $p_t$, $\mathbf{d}_t$] = solve_FSI();             // Solve the FSI problem
11:        Q_out[t] = f($\mathbf{u}_t|_{out}$);                       // Outflow computation
12:        t = t + dt;
13:    end while
14:    Q_output[n] = mean($Q|_{out}$);                            // Average in time
15: end for
16: plot(Q_data, H_data; Q_output, H_Ndata);                   // Validation plot
17: err_Q = norm2(Q_Ndata - Q_output);
```

3. Therefore, for each of the N selected cases, the input parameters for the inlet flow (line 4) and for the membrane oscillation (lines 5-6) are used to define the boundary conditions of the problem (line 7).
4. In this way, the pump system is simulated in the time interval $[0, T_{max}]$, so that we obtain the numerical values of the blood velocity $\mathbf{u}_t$, blood pressure p_t and structure displacement $\mathbf{d}_t$ at each time t; afterwords, the numerical outflow Q_out is computed by taking the integral over the outlet surface of the blood normal velocity (lines 8-13).
5. At the end of each simulation, the pump outflow rate is averaged in time over the last period of membrane oscillation (line 14).
6. Finally, the HQ curves can be plotted together with the numerical output and an error measure can be computed as the 2-norm of the deviation of the numerical results from the experimental data (lines 16-17).

Therefore, the input for the benchmark are:

- the experimental data, i.e. the HQ curves,
- $N \geq 1$: number of data points used for validation (it coincides with the number of simulations),
- input_pressure: head pressure values for the n^{th} simulation,
- input_frequency: frequency of membrane vibration for the n^{th} simulation,
- input_amplitude: amplitude of membrane vibration for the n^{th} simulation,
- physical parameters of the blood pump system, e.g. blood and structure properties;

while the output are:

- Q_output: predictions of the outflow rate from the N simulations,
- err_Q: 2-norm error between the estimated and the experimental outflow vectors,
- validation plot: HQ curve and numerical points.

In conclusion, the proposed benchmark can be used for training in the fields of mathematical modeling of a coupled system, model testing and error estimation.

References

[1] C. N. Botterbusch, S. Lucquin, P. Monticone, J. B. Drevet, A. Guignabert, and P. Meneroud. Implantable pump system having an undulating membrane, May 15 2018. US Patent 9,968,720.

[2] C. N. Botterbusch, P. Monticone, E. Illouz, B. Burg, L. Polverelli, and T. Snyder. Getting past the spin: The CorWave LVAD, a membrane wave pump providing physiologic pulsatility without high shear. *The Journal of Heart and Lung Transplantation*, 38(4):5345, 2019.

[3] T. Snyder, A. Bourquin, F. Cornat, B. Burg, J. Biasetti, and C. N. Botterbusch. CorWave LVAD development update. *The Journal of Heart and Lung Transplantation*, 38(4):5341–5342, 2019.

[4] M. Perschall, J. B Drevet, T. Schenkel, and H. Oertel. The progressive wave pump: numerical multiphysics investigation of a novel pump concept with potential to ventricular assist device application. *Artificial organs*, 36(9), 2012.

[5] E. Burman and M. A. Fernández. An unfitted Nitsche method for incompressible fluid–structure interaction using overlapping meshes. *Computer Methods in Applied Mechanics and Engineering*, 279:497–514, 2014.

[6] A. Hansbo and P. Hansbo. An unfitted finite element method, based on Nitsche's method, for elliptic interface problems. *Computer methods in applied mechanics and engineering*, 191(47-48):5537–5552, 2002.

[7] S. Zonca, C. Vergara, and L. Formaggia. An unfitted formulation for the interaction of an incompressible fluid with a thick structure via an XFEM/DG approach. *SIAM Journal on Scientific Computing*, 40(1):B59–B84, 2018.

[8] E. Burman, M. A. Fernández, and P. Hansbo. Continuous interior penalty finite element method for Oseen's equations. *SIAM journal on numerical analysis*, 44(3):1248–1274, 2006.

[9] E. Burman. Ghost penalty. *Comptes Rendus Mathematique*, 348(21-22):1217–1220, 2010.

[10] C. Vergara and S. Zonca. *Extended Finite Elements Method for Fluid-Structure Interaction with an Immersed Thick Non-linear Structure*, pages 209–243. 01 2018.

[11] M. Hillairet and T. Takahashi. Collisions in three-dimensional fluid structure interaction problems. *SIAM Journal on Mathematical Analysis*, 40(6):2451–2477, 2009.

[12] E. Burman, M. A. Fernández, and S. Frei. A Nitsche-based formulation for fluid-structure interactions with contact. *arXiv preprint arXiv:1808.08758*, 2018.

[13] E. Lauga, M. P. Brenner, and H. A Stone. Microfluidics: the no-slip boundary condition. *arXiv preprint cond-mat/0501557*, 2005.

[14] L. Bertagna, S. Deparis, L. Formaggia, D. Forti, and A. Veneziani. The LifeV library: engineering mathematics beyond the proof of concept. *arXiv preprint arXiv:1710.06596*, 2017.

[15] H. Si. TetGen, a delaunay-based quality tetrahedral mesh generator. *ACM Transactions on Mathematical Software (TOMS)*, 41(2):11, 2015.

A. Appendix

A.1. Licences of Use

A.1.1. LIFEV (release version)

A.1.2. TetGen

A.1.3. Triangle

A.2. Configuration files

```
#!/bin/bash
```

```
source defs.sh

cd $TetGen_DIR

make tetlib

cd $LifeV_BUILD

cmake \
-D BUILD_SHARED_LIBS:BOOL=OFF \
-D CMAKE_BUILD_TYPE:STRING=RELEASE \
-D CMAKE_INSTALL_PREFIX:PATH=${LifeV_LIB}\
-D CMAKE_C_COMPILER:STRING="mpicc" \
-D CMAKE_CXX_COMPILER:STRING="mpicxx" \
-D CMAKE_CXX_FLAGS:STRING="-O3 -msse3 -Wno-unused-local-typedefs -Wno-literal-suffix" \
-D CMAKE_C_FLAGS:STRING="-O3 -msse3" \
-D CMAKE_Fortran_COMPILER:STRING="mpif90" \
-D CMAKE_Fortran_FLAGS:STRING="-Og -g" \
-D CMAKE_AR:STRING="ar" \
-D CMAKE_MAKE_PROGRAM:STRING="make" \
\
-D TPL_ENABLE_AMD=ON \
-D   AMD_INCLUDE_DIRS=/usr/include/suitesparse/ \
-D   AMD_LIBRARY_DIRS=/usr/lib/x86_64-linux-gnu/ \
-D   AMD_LIBRARY_NAMES=amd \
-D TPL_ENABLE_BLACS=ON \
-D   BLACS_INCLUDE_DIRS=/usr/include/ \
-D   BLACS_LIBRARY_DIRS=/usr/lib/ \
-D   BLACS_LIBRARY_NAMES=blacs \
-D TPL_ENABLE_Boost=ON \
-D   Boost_INCLUDE_DIRS=/usr/include/boost/ \
-D TPL_ENABLE_BoostLib=ON \
-D   Boost_NO_BOOST_CMAKE:BOOL=ON \
-D   BoostLib_INCLUDE_DIRS=/usr/include/boost/ \
-D   BoostLib_LIBRARY_DIRS=/usr/lib/x86_64-linux-gnu/ \
-D TPL_ENABLE_HDF5=ON \
-D   HDF5_INCLUDE_DIRS=/usr/include/hdf5/openmpi/ \
-D   HDF5_LIBRARY_DIRS=/usr/lib/x86_64-linux-gnu/ \
-D TPL_ENABLE_SCALAPACK=ON \
-D   SCALAPACK_INCLUDE_DIRS= \
-D   SCALAPACK_LIBRARY_DIRS=/usr/lib/ \
-D   SCALAPACK_LIBRARY_NAMES=scalapack \
-D TPL_ENABLE_MPI=ON \
-D   MPI_BASE_DIR:PATH=/usr/ \
-D   ParMETIS_INCLUDE_DIRS=/usr/include/ \
-D   ParMETIS_LIBRARY_DIRS=/usr/lib/x86_64-linux-gnu/ \
-D TPL_ENABLE_UMFPACK=ON \
-D   UMFPACK_INCLUDE_DIRS=/usr/include/suitesparse/ \
-D   UMFPACK_LIBRARY_DIRS=/usr/lib/x86_64-linux-gnu/ \
-D   UMFPACK_LIBRARY_NAMES=umfpack \
-D   Trilinos_INCLUDE_DIRS:PATH="/usr/include/trilinos/" \
-D   Trilinos_LIBRARY_DIRS:PATH="/usr/lib/x86_64-linux-gnu/" \
-D   TetGen_INCLUDE_DIRS:PATH="${TetGen_DIR}" \
-D   TetGen_LIBRARY_DIRS:PATH="${TetGen_DIR}" \
-D   TetGen_LIBRARY_NAMES="tet" \
-D   Triangle_INCLUDE_DIRS:PATH="/usr/include" \
-D   Triangle_LIBRARY_DIRS:PATH="/usr/lib" \
-D   Triangle_LIBRARY_NAMES="triangle" \
\
-D   LifeV_ENABLE_DEBUG:BOOL=OFF \
-D   LifeV_ENABLE_TESTS:BOOL=ON \
\
-D   LifeV_ENABLE_ALL_PACKAGES:BOOL=ON \
-D   LifeV_ENABLE_Core:BOOL=ON \
-D   LifeV_ENABLE_ETA:BOOL=ON \
-D   LifeV_ENABLE_NavierStokes:BOOL=ON \
-D   LifeV_ENABLE_BCInterface:BOOL=ON \
-D   LifeV_ENABLE_Structure:BOOL=ON \
-D   LifeV_ENABLE_ZeroDimensional:BOOL=ON \
-D   LifeV_ENABLE_OneDFSI:BOOL=ON \
-D   LifeV_ENABLE_LevelSet:BOOL=ON \
-D   LifeV_ENABLE_Darcy:BOOL=ON \
-D   LifeV_ENABLE_Electrophysiology:BOOL=ON \
-D   LifeV_ENABLE_Heart:BOOL=ON \
-D   LifeV_ENABLE_FSI:BOOL=ON \
-D   LifeV_ENABLE_Multiscale:BOOL=ON \
-D   LifeV_ENABLE_XFEM:BOOL=ON \
-D   LifeV_ENABLE_Dummy:BOOL=ON \
\
-D   LifeV_STRUCTURE_ENABLE_ANISOTROPIC:BOOL=ON \
-D   LifeV_STRUCTURE_COLORING_MESH:BOOL=ON \
-D   LifeV_STRUCTURE_COMPUTATION_JACOBIAN:BOOL=ON \
-D   LifeV_STRUCTURE_EXPORTVECTORS:BOOL=ON \
\
-D   BCInterface_ENABLE_TESTS:BOOL=ON \
-D   Core_ENABLE_TESTS:BOOL=ON \
-D   Darcy_ENABLE_TESTS:BOOL=ON \
-D   Dummy_ENABLE_TESTS:BOOL=ON \
-D   Electrophysiology_ENABLE_TESTS:BOOL=ON \
-D   ETA_ENABLE_TESTS:BOOL=ON \
-D   FSI_ENABLE_TESTS:BOOL=ON \
-D   Heart_ENABLE_TESTS:BOOL=ON \
-D   LevelSet_ENABLE_TESTS:BOOL=ON \
-D   Multiscale_ENABLE_TESTS:BOOL=ON \
-D   NavierStokes_ENABLE_TESTS:BOOL=ON \
-D   OneDFSI_ENABLE_TESTS:BOOL=ON \
-D   Structure_ENABLE_TESTS:BOOL=ON \
-D   XFEM_ENABLE_TESTS:BOOL=ON \
-D   ZeroDimensional_ENABLE_TESTS:BOOL=ON \
\
-D   BCInterface_ENABLE_EXAMPLES:BOOL=ON \
-D   Core_ENABLE_EXAMPLES:BOOL=ON \
-D   Darcy_ENABLE_EXAMPLES:BOOL=ON \
-D   Dummy_ENABLE_EXAMPLES:BOOL=ON \
-D   Electrophysiology_ENABLE_EXAMPLES:BOOL=ON \
-D   ETA_ENABLE_EXAMPLES:BOOL=ON \
-D   FSI_ENABLE_EXAMPLES:BOOL=ON \
-D   Heart_ENABLE_EXAMPLES:BOOL=ON \
-D   LevelSet_ENABLE_EXAMPLES:BOOL=ON \
-D   Multiscale_ENABLE_EXAMPLES:BOOL=ON \
-D   NavierStokes_ENABLE_EXAMPLES:BOOL=ON \
-D   OneDFSI_ENABLE_EXAMPLES:BOOL=ON \
-D   Structure_ENABLE_EXAMPLES:BOOL=ON \
-D   XFEM_ENABLE_EXAMPLES:BOOL=ON \
-D   ZeroDimensional_ENABLE_EXAMPLES:BOOL=ON \
\
-D   LifeV_ENABLE_STRONG_CXX_COMPILE_WARNINGS:BOOL=ON \
${LifeV_SRC}

cd $LifeV_DIR
```

Listing 6: config.sh

ROMSOC

The ROMSOC project

April 27, 2020

ROMSOC-D5.2-0.1

Horizon 2020