MOX-Report No. 56/2022

lifex: a flexible, high performance library for the numerical solution of complex finite element problems

Africa, P.C.

MOX, Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

mox-dmat@polimi.it http://mox.polimi.it

`life`^{x}: a flexible, high performance library for the numerical solution of complex finite element problems

P. C. Africa

MOX, Department of Mathematics, Politecnico di Milano
Piazza Leonardo da Vinci, 32, 20133, Milano (Italy)

Abstract

Numerical simulations are ubiquitous in mathematical and computational modeling, where many industrial and clinical applications are required to deal with multiphysics problems and with complex systems characterized by multiple spatial and temporal scales.

This document introduces the design and the capabilities of `life`^{x}, an open source `C++` library for high performance finite element simulations of multiphysics, multiscale and multidomain problems. `life`^{x} offers a versatile solution to answer the emerging need for efficient computational tools that are also easily approachable by a wide community of users and developers. We showcase illustrative examples of use, benchmarks, advanced application scenarios and demonstrate its parallel performance up to thousands of cores.

Keywords: mathematical software, high performance computing, scientific computing, finite elements, numerical simulations, multiphysics problems
PACS: 02.30.Jr, 02.60.Cb, 02.70.-c, 02.70.Dh
2020 MSC: 35-04, 65-04, 65Y05, 65Y20, 68-04, 68N30

Code repository	`https://gitlab.com/lifex/lifex`
Homepage	`https://lifex.gitlab.io/`
Documentation	`https://lifex.gitlab.io/lifex/`
License	LGPLv3
Programming language and tools	`C++` (standard $\geq$ 17) `MPI` `CMake` $\geq$ 3.12.0
Third-party dependencies	`deal.II` $\geq$ 9.3.0 `VTK` $\geq$ 9.0.0 `Boost` $\geq$ 1.76.0

Table 1: `life`^{x} metadata.

1. Background

A broad range of applications in biology, medicine, physics, engineering, astronomy, energy, environmental and material sciences can be described by multiple complex physical processes interacting at different spatial and temporal scales [1]. Such models can be seen as agglomerations of well-defined physics referred to as *core models*. Therefore, it is fundamental to develop new computational frameworks for the numerical

Email address: `pasqualeclaudio.africa@polimi.it` (P. C. Africa)

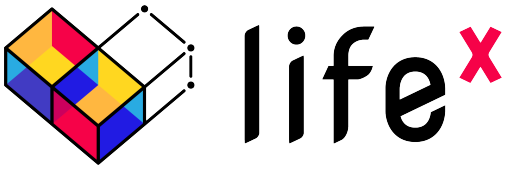

Figure 1: `life`^{x} official logo. This image is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

solution of multiphysics, multiscale, multidomain problems. Such tools should aim at reproducibility of *in silico* experiments while providing at the same time a stable and intuitive simulation environment without compromising the computational accuracy, efficiency and universality.

The development of tool of this kind plays a central role in decoupling the software development phase from the time-consuming process of performing different kinds of analysis – from forward simulations to sensitivity analysis, optimization, and uncertainty quantification – and in enabling the simulation of each core model both as standalone and in various coupled configurations [2].

In this work we introduce `life`^{x} (pronounced /ˌlaɪfˈɛks/, metadata listed in Tab. 1 and official logo shown in Fig. 1), an open source library for the numerical solution of Partial Differential Equations (PDEs) and related coupled problems. It is written in `C++` using modern programming techniques available in the `C++17` standard and builds upon the `deal.II 9.3.1` [3] Finite Element (FE) core. `life`^{x} aims at providing a flexible, intuitive, yet robust and high performance tool simplifying the definition of complex physical models and parameters, coupling and post-processing.

`life`^{x} enables its users to shift the focus from technical numerical details to the plain mathematical formulation of the problem of interest. The library provides extensive documentation and a number of test cases covering a wide range of numerical applications and coupling strategies.

While serving similar purposes as other multiphysics libraries already available in the open source community, such as `FEniCS` [4, 5], `MFEM` [6], `MOOSE` [7] and `preCICE` [8], `life`^{x} offers several unique features, including:

- an intuitive user interface with extreme ease of use;
- a open source license that allows for unrestricted academic use;
- modern programming paradigms by design, leveraging the `C++17` standard, and up-to-date versions of third-party dependencies;
- high parallel scalability;
- interoperability, *i.e.* the possibility to import (export) data and meshes from (to) common file formats, with reference in particular to `VTK`;
- support for arbitrary finite elements and the possibility to import meshes with either hexahedral or tetrahedral elements [9];
- a clean and meticulously documented code base.

All of these features will be highlighted hereafter in this manuscript.

2. Software description

`life`^{x} has been conceived in 2019 as an academic research library in the framework of the iHEART project (see Sec. Acknowledgements) at Politecnico di Milano, with a primary focus on mathematical models and numerical schemes for integrated simulations of the cardiac function.

Since its initial design, many modules for the simulation of different core models were added to the code base. The development of `life`^{x} has been founded on strict coding conventions and practices [10]. The fast

increase of the number developers and users is a testament to the fact that its kernel is intuitive, fast and general enough to be used for diverse applications and worthy of being built and released as a standalone library.

Third-party dependencies of `life`^{x} include: `deal.II` (configured with support to `PETSc` [11] and `Trilinos` [12]), `VTK`, and `Boost`. `life`^{x} can be configured to use, by default, linear algebra data structures and algorithms from `PETSc`, `Trilinos` (through the interfaces exposed by `deal.II`) or `deal.II` itself; where needed, a specific datatype or solver provided by one of the three backends may also be hard-coded, disregarding the default type `life`^{x} was configured with. All the code is natively parallel through the Message Passing Interface (`MPI`): following a distributed memory paradigm, the global mesh is decomposed so that each `MPI` process owns and stores only a subset of cells.

This library aspires for maximum portabilty, having being deployed successfully on `Linux`, `Windows` and `macOS` operating systems. This builds upon advanced deployment technology, more specifically: `mk` (a set of portable, pre-compiled scientific packages for `x86-64` `Linux` systems), `lifex-env` (a set of *build-from-source* shell scripts, deliberately inspired from `candi`), or via `Spack`. Pre-built `Docker` images with all dependencies installed are also ready for download and use. More details can be found on `life`^{x} website.

2.1. Overview

Structurally, the key features of `life`^{x} can be grouped in three main components:

1. an **abstraction layer** built on top of the `deal.II` FE library, exposing abstract numerical *helpers* as essential building blocks that foster the development of advanced data structures and numerical schemes for time integration, linearization, solving and preconditioning linear systems, imposing boundary conditions and mesh handling;

2. a framework for **multiphysics coupling**, with functionalities enabling to transfer solution fields and data from one core model to the other, either in the same domain or across multiple domains;

3. a seamless **user interface** through several advanced Input/Output (I/O) capabilities, with a focus on importing data coming from the post-processing of experimental results, imaging techniques or other numerical simulations, *e.g.* with the help of the `VTK` library.

The main code components, classes and their interaction falling into these three categories are depicted in Fig. 2.

All `life`^{x} executables belong to one of three main categories.

`apps`:
: generic applications that are not model-specific, such as tools for printing mesh statistics or for converting between compatible file formats.

`examples`:
: problems and solvers that define specific model or geometrical parameters, such as boundary conditions, initial conditions, domain, ...

`tests`:
: executables used for automatic testing (run via `CTest`), automatically run on continuous integration services at each `git push` on `GitLab` remote. Tests also include a number of *tutorials*, which can be used as prototypes for building new applications. All tests and tutorials are used to determine the overall code coverage, *i.e.* a metric that determines the number of lines of code that are successfully validated by the test procedure.

2.2. Implementation

All `life`^{x} applications are associated with a set of common attributes, such as: user-specified command line flags; the name of a parameter file, *i.e.* a file with a tree-like structure containing all configurations, parameters and settings used to run the executable; a *subsection path*, *i.e.* the path in the parameter file

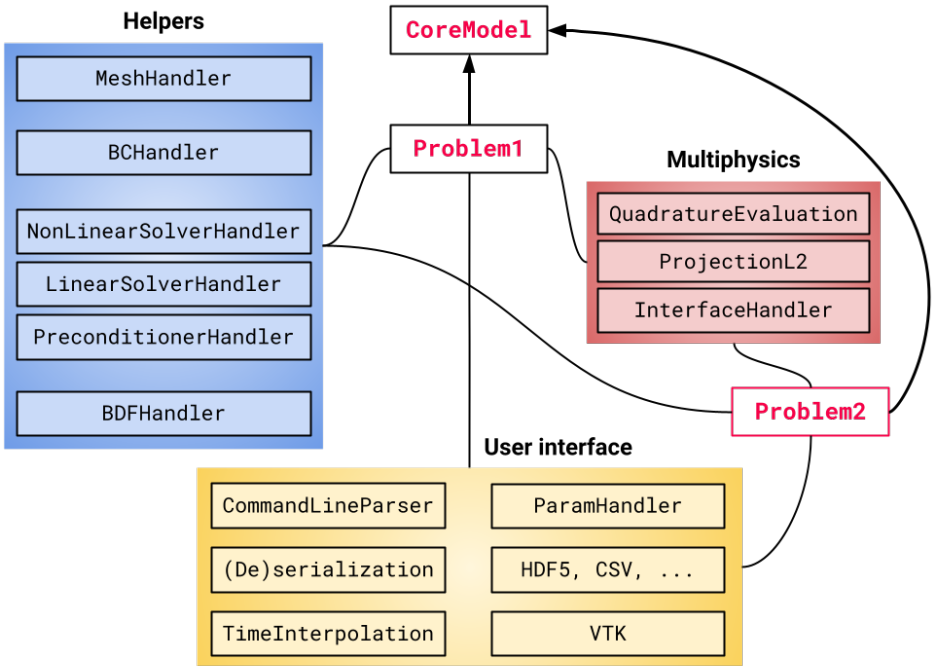

Figure 2: Overview of `life`^{x} main components. The main classes and their interaction are shown, grouped in three categories: abstract numerical helpers (blue), multiphysics coupling (red), user interface (yellow).

associated with the currently open subsection; an *execution mode* flag that specifies whether to generate a new parameter file or actually execute the app; an output directory, which will contain all output files; `MPI` rank and size used for parallel computations. Upon running, such attributes are shared among instances of all classes.

Moreover, all main classes are designed in such a way to expose their own specific parameters, such as geometry, physical parameters, discretization schemes, numerical settings, I/O options, from the parameter file.

The following three main classes are used to define a global, common interface for all `life`^{x} applications:

`Core`:
: a class implemented following the *singleton* design pattern that stores attributes that are global and common to all other classes, such as the ones mentioned above.

`CoreModel`:
: an abstract class that inherits from `Core` and extends it with pure virtual methods that define the interface exposed by each *core model* or *solver* throughout `life`^{x}. Classes within the `CoreModel` hierarchy expose a set of parameters that configure their behavior. Such parameters are exposed to the user through the parameter file (see Sec. 2.2.3). A sample code snippet will be provided and discussed in Sec. 2.3.

`lifex_init`:
: a lifespan handler that takes care of properly initializing all attributes and dependencies needed by each run, such as the instance of the singleton `Core` and `MPI`; an instance of this class is typically constructed at the very beginning of the `main()` function and destroyed at the program end.

More specific, high level data structures are described below.

2.2.1. Abstract numerical helpers

A huge part of `life`^{x} consists of abstract wrappers and helpers: most of these classes explicitly invoke or refer to `deal.II` design and features [13], with the goal of providing a higher level interface to them and facilitating the implementation of advanced numerical schemes for a given problem. The main classes are described below.

`MeshHandler`:
: a wrapper around `deal.II` distributed meshes. The user can select whether to import a mesh with hexahedral or tetrahedral elements; depending on this choice, this class owns an instance of a `distributed` or a `fullydistributed` triangulation from `deal.II`; the latter is a recent introduction that adds support to tetrahedral meshes [3], which at the time of writing is still to be consolidated. The `MeshHandler` class tightly interacts with `MeshInfo`, that parses information from the input mesh such as volume and surface tags to be used, for instance, to impose different boundary conditions on different parts of the boundary or to differentiate material properties in different sub-regions. Helper functions (implemented in the `geometry/mesh_info` and `geometry/finders` modules) allow the computation of (sub)domain volumes and boundary surfaces or to locate, *e.g.*, the closest degrees of freedom, mesh vertex or boundary face to a given input point.

`BCHandler`:
: a helper class to impose boundary conditions; Dirichlet boundary conditions can be applied directly to a FE vector or imposed as linear constraints to the linear systems arising from a FE discretization. For vector problems, the normal flux or tangential flux can also be imposed. A helper method to assemble Neumann and Robin-like contributions to a local system right-hand side is also provided.

`LinearSolverHandler`:
: for a (sparse, distributed) linear system, this class provides a simple interface that enables the user to select at run-time which linear solver to use and all of its options (for instance, maximum number of iterations, tolerances, stopping criteria, history log, ...), parsed from parameter files. Many common solvers are already included, such as `CG`, `GMRES`, `BiCGStab`, `MinRes`, `FGMRES`, but in principle any solver exposed by `deal.II` (including those from `PETSc` and `Trilinos`) is supported. The complete suite of solvers from `PETSc` is still accessible via the `-options_file` command-line flag, forwarded from `life`^{x} to `PETSc`.

`PreconditionerHandler`:
: analogously to `LinearSolverHandler`, this class exposes parameters that are used for the preconditioning of linear systems. It supports many types of preconditioner, such as Algebraic Multi-Grid (AMG), block Jacobi, additive Schwarz (SOR, SSOR, block SOR, block SSOR, ILU, ILUT), and can be easily extended to support more.

`BDFHandler`:
: for time dependent problems, semi-implicit Backward Difference Formula (BDF) time discretization schemes [14] are implemented in this class, which deals with storing the information to advance the problem from one time step to the next one. This class stores and exposes the BDF solution and its extrapolation and can be easily extended to different time advancing schemes.

`NonLinearSolverHandler`:
: for solving non-linear problems, a family of Newton methods is provided. An abstract implementation requires the user to specify an *assemble function*, that assembles the Jacobian matrix and the residual vector, and a *solve function* that assembles the preconditioner and solves the linear system associated with each non-linear iteration; the two functions must return the norms of residual, solution and Newton increment, to be used as possible stopping criteria. The *frozen Jacobian* (or *Jacobian lagging*) approach [15], which consists of re-assembling the Jacobian only once every n time steps, can be toggled to increase the computational efficiency. Two specializations for the quasi-Newton method with the Jacobian matrix approximated via finite differences [16] and for the inexact Newton method [17] are also supplied.

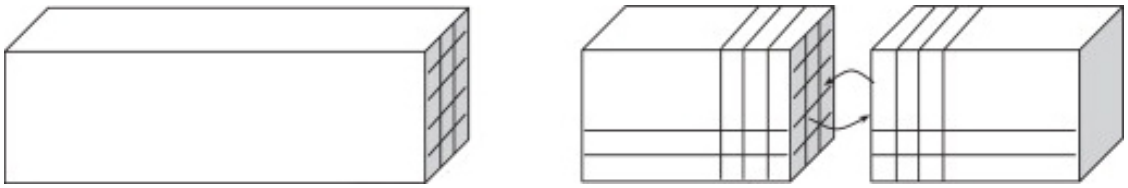

Figure 3: Possible solution schemes for a geometrically coupled problem: monolithic (left) solution scheme vs. partitioned (right). Reprinted from [21]. The original image is licensed under a CC BY 3.0 License.

Moreover, each non-linear solution scheme can be equipped with proper acceleration strategies (static relaxation, Aitken [18], Anderson [19] acceleration) to accelerate convergence. In addition to the non-linear solver handler, the user can benefit from the use of *automatic differentiation* (with support to `Sacado` and `ADOL-C` interfaces exposed by `deal.II`), demonstrated on `Tutorial04_AD` and `Tutorial07_AD`, which allows to compute the derivatives of often complicated functions to a very high accuracy.

2.2.2. Multiphysics coupling

The complexity of multiphysics, multiscale and multidomain problem can be eased by the help of three hierarchies of classes. They all serve the purpose of transferring solution fields and data from one core model to the other, or across internal interfaces.

In order to keep the code as general as possible, we assume that different core models can be solved using arbitrarily independent discretization schemes, such as different finite element degrees or mesh resolutions. This allows to better capture all the physical phenomena involved, whose dynamics can be characterized by extremely different spatial and temporal scales.

We remark that problems involving more than one physical model (possibly on multiple domains sharing a common interface, such as in the case of Fluid-Structure Interaction) can be generally solved using either monolithic or partitioned algorithms [20], as schematized in Fig. 3. In the former case, a global system involving all the unknowns from all problems is assembled and solved (at each time step); in the latter, each sub-problem is solved independently and coupling conditions are imposed, *e.g.* using explicit schemes or sub-iterating with a fixed-point scheme until convergence of coupling conditions is reached. Both choices are possible within `life`^{x} and illustrated on a number of examples and tests.

`QuadratureEvaluation`**:**
this class provides a high level interface for the evaluation of arbitrary analytic functions or more complex data structures at a given quadrature point. User-defined classes deriving from `QuadratureEvaluation` can be easily implemented for scalar, vector or tensor fields. Furthermore, the `QuadratureEvaluationFEM` hierarchy of classes is implemented to enable the coupling of multiple physical models solved in the same domain. The different problems can be discretized using different finite element degrees and the integrals arising from the weak formulation can be approximated using quadrature formulas of different type and/or accuracy. The `QuadratureEvaluationFEM` classes provide an interface similar to that of `FEValues` from `deal.II`: such objects are constructed using the `DoFHandler` associated with the finite element field to evaluate and the quadrature rule used for the target problem. By re-initializing such objects on each mesh cell, the input field can be evaluated at the corresponding quadrature points. `life`^{x} provides specializations to automatically evaluate finite element solutions, gradients and divergence of a given solution vector.

`ProjectionL2`**:**
instead of the exact numerical evaluation allowed by `QuadratureEvaluation` classes, a *smoothed* L^2 projection approach can be considered. Given a function $f(\mathbf{x})$, this class computes a finite element solution $f_h(\mathbf{x})$ that satisfies $(\varepsilon\nabla f_h, \nabla\varphi_i)_\Omega + (f_h, \varphi_i)_\Omega = (f, \varphi_i)_\Omega$ for each basis function φ_i in the chosen finite element space. The numerical solution to this problem clearly involves a mass matrix: its lumping can be toggled and the regularization parameter ε can be tuned in order to avoid numerical oscillations,

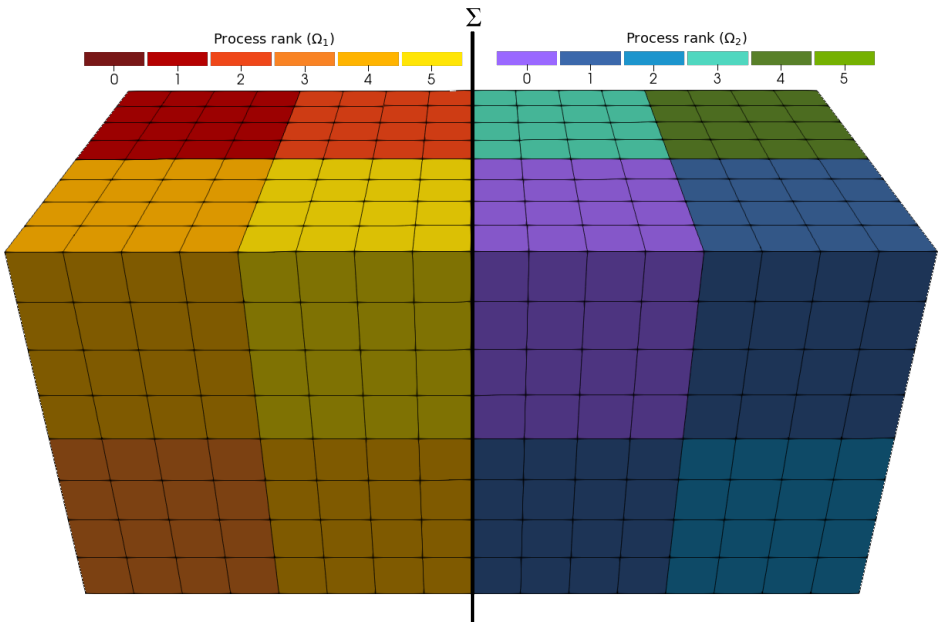

Figure 4: Example of handling two domains Ω_1 (left) and Ω_2 (right) sharing a common interface with conforming mesh discretizations. The `InterfaceHandler` is able to properly deal with non-conforming parallel partitioning.

e.g. in the case of coarse meshes [22]. The solution f_h thus obtained can be easily evaluated at quadrature nodes associated with the target problem.

`InterfaceHandler`:
consider two subdomains Ω_1 and Ω_2, sharing a common interface Σ, with conforming discretizations and let u_1 and u_2 be finite element functions defined on the two subdomains, typically representing solutions to differential problems defined on the two subdomains. Suppose that the problem defined on Ω_1 (Ω_2) involves conditions on Σ that depend on u_2 (u_1) [23, 20]. This class manages, for each of the subdomains, the extraction of interface data on Σ from the other subdomain and its application as a boundary condition on Σ. `InterfaceHandler` also deals with *interface maps*, *i.e.* two mappings of degrees of between the local interface Σ and the global domains Ω_1 and Ω_2. This is of critical importance in parallel simulations, where the parallel partitioning on both domains can be different, such as the example shown in Fig. 4. This class only deals with conforming meshes, whereas extensions to non-conforming discretizations, such as the `INTERNODES` technique [24], are still under development.

The last case to be considered is transferring solutions between multiple core models solved using the same finite element discretization but with different mesh resolutions. For nested hexahedral grids, the `VectorTools` namespace of `deal.II` already provides functions that do exactly what is needed. This procedure is hardly generalizable as it tightly depends on how the different meshes have been generated and on the mesh element type. For instance, transfer operator built upon Radial Basis Function (RBF) interpolators could be used in the case of non-conforming discretizations [25, 26] but are still to be implemented in `life`^{x}.

Clearly, both approaches can be combined in order to couple different models solved with different finite element approximations and different mesh resolutions.

2.2.3. User interface

`CommandLineParser`:
`life`^{x} makes use of the lightweight `clipp` interface for parsing command-line arguments. All executables expose a set of command-line options, which can be printed using the `-h` (or `--help`) flag:

```
./executable_name -h
```

ParamHandler:
each `life`^{x} application or example defines a set of parameters that are required in order to be run. They involve problem-specific parameters (such as constitutive relations, geometry, time interval, boundary conditions, ...), numerical parameters (types of linear/non-linear solvers, tolerances, maximum number of iterations, ...), I/O options, ... In case an application has sub-dependencies (such as a linear solver), also the related parameters are included (typically in a proper subsection). Parameters are organized in a tree-like structure following the functionalities exposed by the `ParameterHandler` class from `deal.II`. The first step before running any executable is to generate the default parameter file(s). This is done via the `-g` (or `--generate-params`) flag:

```
./executable_name -g -f filename.ext
```

At user's option, in order to guarantee a flexible interface to external file processing tools, the parameter file extension `ext` can be chosen among three different interchangeable file formats `prm`, `json` or `xml`, sorted from the most human-readable to the most machine-readable.

An excerpt of a `prm` file is the following:

```
subsection Problem
  subsection Mesh and space discretization
    # Here goes the parameter description.
    set Element type = Hex
    # ...
  end
  # ...
  subsection Linear solver
    set Type = GMRES

    subsection GMRES
      set Max. number of temporary vectors = 100
      # ...
    end
  end
  # ...
  subsection Preconditioner
    set Type = AMG

    subsection AMG
      set W-cycle = true
      # ...
    end
end
```

Listing 1: Example of parameter file in `prm` format.

Omitting the `-g` flag reads an existing parameter file and runs the simulation.

The `ParamHandler` class of `life`^{x} extends the `deal.II` class by two main functionalities:

verbosity control:
by default, only parameters considered with a *standard* verbosity are printed. In order to customize the user experience, the verbosity of each parameter can be decreased (*minimal*) or increased (*full*) from the source code. A parameter file containing a minimal (full) set of parameters can be generated by passing the optional flag `minimal` (`full`) to the `-g` flag, respectively:

```
./executable_name -g [minimal,full] \
                  -f filename.ext
```

If the `-g` is provided without any further specification, the intermediate level of verbosity is assumed.

multiple default values:
in principle, each application could be run to simulate different scenarios or simply with different pre-defined sets of parameters; `life`^{x} gives the possibility to provide multiple default parameter files out of the box. The `ParamHandler` class can read user-provided files in `json` format specifying a list of parameter names and their (new) default values, which will be appended to the complete set of parameters and written to file, ready to use (see, *e.g.*, the `time_interpolation` test).

Utilities for parsing *lists of values* are also provided in the `param_handler_helpers` module for convenience of use.

(De-)serialization:
`life`^{x} includes a checkpointing system that allows for all aspects of a simulation to be serialized to file. This allows to recover a simulation state after an unexpected failure, to restart it after maximum computational wall time has been reached, or simply to initialize a simulation with custom input data. Convenient tools for (de-)serializing (distributed) meshes and solution vectors is provided in the `io/serialization` module, with interface to `deal.II`-compatible binary files.

`CSV` readers and writers:
the simplicity of use of Comma-Separated Values (CSV) files is widespread to process data organized as fields. Many utility functions and classes are present in `life`^{x} to read and write `CSV` files by converting number and text values into `deal.II` data structures (vectors, matrices, ...). This enables to easily post-process simulation results, *e.g.* by exporting point-wise variables at each time step.

`TimeInterpolation`:
many applications require to resample discrete sets of data at arbitrary points, such as time dependent variables that need to be interpolated in correspondence of the time steps performed by the numerical simulation. The `TimeInterpolation` class provides methods based on: linear interpolation, cubic splines, smoothing cubic splines, trigonometric interpolation (discrete Fourier transform) and linear and spline interpolation of the derivative of the input data.

`VTKFunction` and `VTKPreprocess`:
many physical problems are characterized by coefficients that are derived from experimental data or imaging techniques, such as segmented geometries of organs from Magnetic Resonance Imaging (MRI) or Computer Tomography (CT) scans [27, 28], or from post-processing of other numerical simulation steps [29]. `VTK` toolkit certainly defines some of the most common data formats to deal with data defined over volumes (`vtkUnstructuredGrids`) or surfaces (`vtkPolyData`). Moreover, it is also used in sophisticated pipelines for surface processing and mesh generation [30]. `life`^{x} provides a class named `VTKFunction`, inheriting from `dealii::Function`, that imports a `VTK` file containing a cell or point data field and evaluates it at an arbitrary point, possibly belonging to a related computational mesh. Three possible evaluation methods are available, namely closest-point, linear projection and signed distance. Finally, `VTKPreprocess` exploits `VTKFunction` to interpolate input `VTK` data onto finite element vectors, which are serialized to file for later importing and reuse in numerical simulations.

2.3. Sample code

The following code illustrates a sample code snippet with comments, containing the minimal interface exposed by the vast majority of all `life`^{x} classes, *i.e.* those inheriting from `CoreModel`. In particular, the `declare_parameters` and `parse_parameters` methods are *pure virtual* and must be overridden, whereas the `run` method is virtual and has an empty definition by default. An example on how to locally adjust the verbosity of some parameters is also shown. Finally, this sample class makes use of a `LinearSolverHandler`, for which we also declare and parse related parameters.

```
namespace lifex
{
  class Problem : public CoreModel
  {
  public:
    // Specify the subsection path where to
    // declare current parameters.
    Problem(const std::string &subsection_path)
    : CoreModel(subsection_path)

    // Specify a "relative" subsection.
    // Subpaths are separated by a "/".
    , linear_solver(
        prm_subsection_path + " / Linear solver",
        /* ... */)
    {}

    virtual void
    declare_parameters(ParamHandler &params) const override
    {
      // Navigate subsections and declare parameters.
      params.enter_subsection_path(prm_subsection_path);
      {
        // Problem-dependent parameters.
        // ...

        params.set_verbosity(VerbosityParam::Full);
        {
          // If -g full is *not* specified,
          // the parameters declared here will
          // be hidden from the parameter file.
          // ...
        }
        params.reset_verbosity();
      }
      params.leave_subsection_path();

      linear_solver.declare_parameters(params);
    }

    virtual void
    parse_parameters(ParamHandler &params) override
    {
      // Actually parse parameter file.
      params.parse();

      // Analogously to declare_parameters,
      // navigate subsections, read parameters
      // and possibly store them into class
      // members.
      // ...
    }

    virtual void
    run() override
    {
      // Create mesh.
      // Setup system.
      // Assemble system.
      // Solve system.
      // Output solution.
    }

  private:
```

```
    LinearSolverHandler linear_solver;
    // ...
  };
}
```

3. Applications

`life`^{x} is capable of solving complex multiphysics problems. The functionalities described in the previous section are pointed out in a series of *tutorials* that are found in the source code as tests. The tutorials are sorted by increasing complexity and involve different kind of scalar equations and coupled problems solved either monolithically or partitioned, namely:

1. linear elliptic equation;
2. non-linear elliptic equation;
3. linear parabolic equation;
4. non-linear parabolic equation;
5. non-linear parabolic equation, with Jacobian matrix assembled via automatic differentiation;
6. parabolic system of equations (solved monolithically);
7. parabolic system of equations (solved using a partitioned scheme and exploiting `QuadratureEvaluationFEM` capabilities);
8. Cahn-Hilliard equation.

The mathematical and numerical formulation for all of these problems is detailed in the package documentation. The results of some of these simulations are showcased below.

Scalability study

We perform a strong scaling test on `Tutorial06`, where the following equations are solved:

$$\begin{cases} \dfrac{\partial u}{\partial t} - \Delta u + u^2 = f, & \text{in } \Omega \times (0,T], \\ \dfrac{\partial v}{\partial t} - \Delta v + uv = g, & \text{in } \Omega \times (0,T], \end{cases}$$

where $\Omega = (-1,1)^3$, with suitable data, boundary and initial conditions (please refer to the documentation for further details).

The two equations are discretized in time using the `BDFHandler` of order 1 for u and 3 for v, decoupled using an explicit partitioned scheme and linearized using the `NonLinearSolverHandler` class. Finally, the space discretization makes use of linear (u) and quadratic (v) finite elements, respectively. The solution u appearing in the second equation is evaluated using the capabilities of `QuadratureEvaluationFEM`. The mesh size consists of 2'097'152 cells (average cell diameter: $h \approx 0.027$) and 19'121'282 degrees of freedom (2'146'689 for u, 16'974'593 for v), the time step is chosen equal to $\Delta t = 0.1$ and the simulation is run until $T = 1$.

The scalability test has been run on the `GALILEO100` supercomputer available at `CINECA` (Intel Cascade-Lake 8260, 2.40GHz). We have recorded the total simulation time as well as partial times spent in the assembly and in the linear solving phases; the speedup for the three quantities shown in Fig. 5 confirms that on such benchmark problem `life`^{x} main data structures scale approximately linearly up to 4096 cores. The linear solver performances slowly degrades starting from about 512 cores probably due to the limited problem size; however, the additional overhead from the `LinearSolverHandler` wrapper is negligible compared to the time spent in applying the `GMRES` solver from the `Trilinos` library.

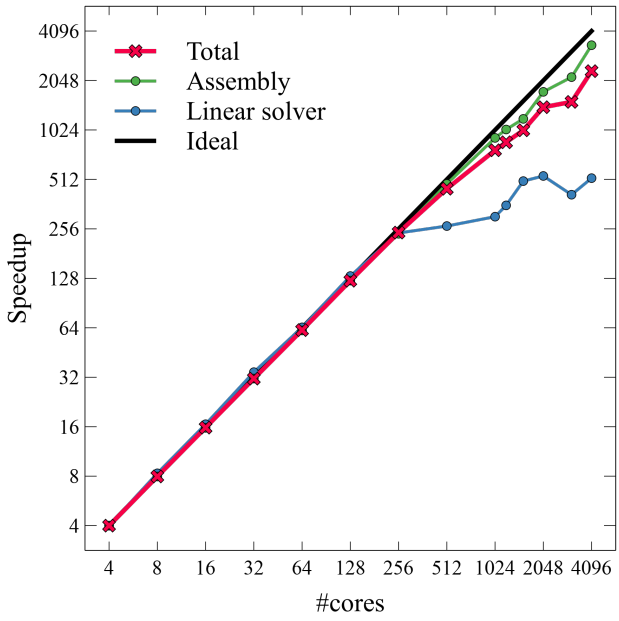

Figure 5: Parallel speedup of `life`^{x}, demonstrated on `Tutorial06`. The speedup has been computed on the total time (red), the time spent in assembling the linear system at each time step (green) and the time spent in solving the linear system at each time step (blue).

Cahn-Hilliard multiphase system

To demonstrate the multiphysics capabilities of `life`^{x} we present a results of the spinodal decomposition of a binary fluid undergoing shear flow using the advective Cahn-Hilliard equation, a stiff, nonlinear, parabolic equation characterized by the presence of fourth-order spatial derivatives [31]. Spinodal decomposition consists in the separation of a mixture of two or more components to the bulk regions of both, which occurs, *e.g.*, when a high temperature mixture of two or more alloys is cooled rapidly.

The equation has been discretized using finite elements in mixed form, by splitting it into a system of two parabolic-elliptic equations:

$$\begin{cases} \dfrac{\partial c}{\partial t} - \Delta\mu = 0, & \text{in } \Omega \times (0,T], \\ \mu - \dfrac{\mathrm{d}f}{\mathrm{d}c}(c) + \lambda\Delta c = 0, & \text{in } \Omega \times (0,T], \end{cases}$$

with homogeneous natural conditions and suitable initial condition (please refer to the documentation for further details).

The solver is implemented in `Tutorial07_AD` by exploiting *automatic differentiation* along with the numerical schemes and multiphysics capabilities described above. Fig. 6 shows the steady state solution over the unit cube.

4. Discussion

4.1. Impact

The impact and the wide applicability of `life`^{x} is demonstrated by the high number of journal articles, preprints, conference abstracts and Ph.D. theses citing it.

Computational studies carried out with `life`^{x} have recently appeared in a variety of fields, mostly originated from but not limited to cardiovascular modeling, such as: cardiac electrophysiology [32, 33, 34, 35],

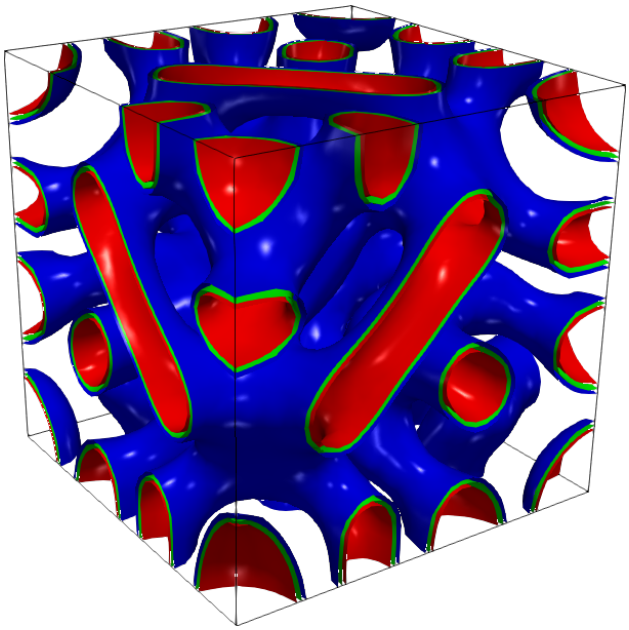

Figure 6: Solution of the Cahn-Hilliard equations implemented in `Tutorial07`. Isosurfaces corresponding to values of the solution c equal to 0.35 (red), 0.5 (green), 0.65 (blue) are shown.

cardiac mechanics, electromechanics and blood circulation [36, 29, 37, 38, 39, 40], fluid dynamics [41, 42, 28], Fluid-Structure Interaction [20], poromechanics coupled with blood perfusion [43, 44], hemodynamics in patients affected by COVID-19 [45].

A recent remarkable result is a comprehensive and biophysically detailed computational model of the whole human heart electromechanics [46], as displayed in Fig. 7.

Other studies oriented towards numerical methods have addressed the development of a high-order matrix-free solver for cardiac electrophysiology [47] and of reduced order methods for real-time simulations [48, 49, 50].

A comprehensive and up-to-date list of publications making use of `life`^{x} can be found at `https://lifex.gitlab.io/lifex/publications.html`.

`life`^{x} has provided a fast and stable environment with a gentle learning curve, enabling to obtain unprecedented results in terms of model reliability, numerical accuracy and computational efficiency. We expect that future directions for `life`^{x} will lead towards expanding its developer and user bases, keeping an active and friendly community that welcomes new contributions and making new advanced features openly available to the wider public (see also [51, 52]).

4.2. Conclusions

`life`^{x} is a parallel `C++` library for simulations of multiphysics, multiscale and multidomain problems based on the `deal.II` finite element core. `life`^{x} shows a low computational footprint and seamless parallel performances enhanced by advanced numerical solvers, thus realizing an invaluable tool that can be run on diverse architectures, ranging from laptop computers to HPC facilities and cloud platforms.

On the one hand, `life`^{x} provides a robust and friendly interface enabling easily accessible and reproducible *in silico* experiments, yet without any compromise on computational efficiency and numerical accuracy. On the other hand, being conceived as a research library, `life`^{x} can be exploited by scientific computing experts to address new modeling and numerical challenges within an easily approachable development framework.

We expect `life`^{x} to attract a sizable community of users and developers. Any contribution is highly appreciated, from code commits to bug reports or by suggesting new ideas for improving `life`^{x}.

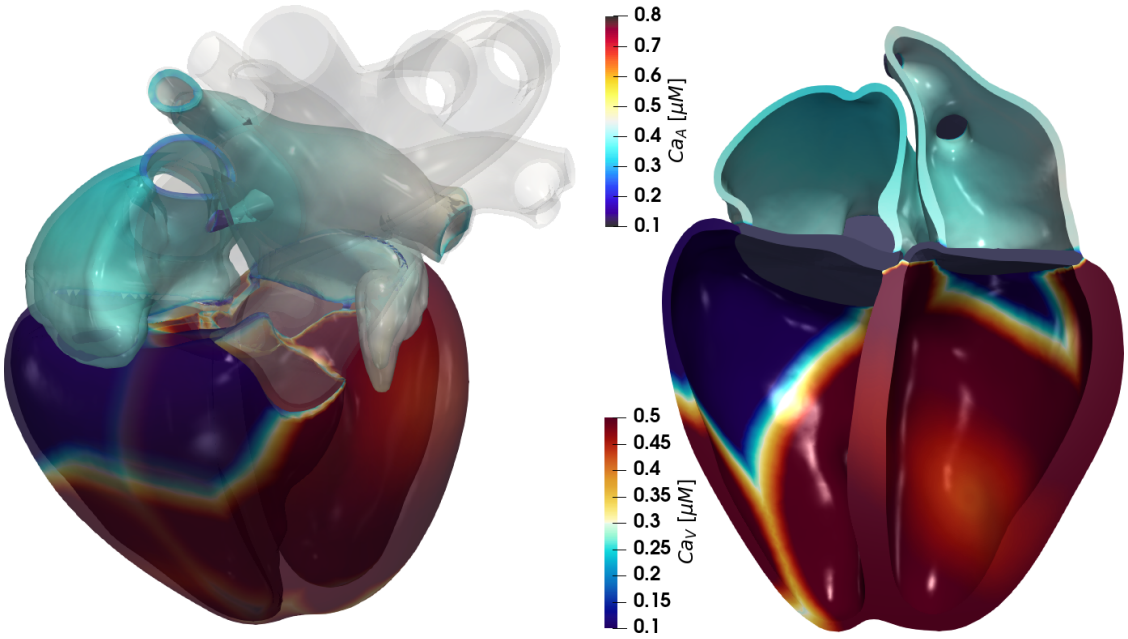

Figure 7: Snapshot of a cardiac electromechanics simulation on a whole-heart geometry during ventricular systole. The color map show the intracellular concentration of calcium ions.

As we approach the exascale era dominated by high-end supercomputers, numerical simulations are expected to be one of the main computational workloads [53]: `life`^{x} will contribute towards this direction by offering transparency, accessibility, reproducibility and reusability of *in silico* experiments, within a flexible, high performance software tool.

5. Conflict of Interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

Acknowledgements

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 740132, iHEART - An Integrated Heart Model for the simulation of the cardiac function, P.I. Prof. A. Quarteroni). We acknowledge the CINECA award `MathBeat` under the ISCRA initiative, for the availability of high performance computing resources and support. `life`^{x} logo has been designed by S. Pozzi.

`life`^{x} would not have been possible without a large and loyal team working on software development and review, contributing code and fixes or reporting bugs and suggestions: N. Barnafi, L. Bennati, M. Bucelli, L. Cicci, S. Di Gregorio, M. Fedele, I. Fumagalli, S. Pagani, R. Piersanti, F. Regazzoni, M. Salvador, S. Stella, E. Zappon, A. Zingaro and many others.

Special thanks go to Profs. A. Quarteroni, L. Dede', L. Formaggia, P. Gervasio, A. Manzoni, C. Vergara, P. Zunino for all the stimulating and inspiring discussions, and to L. Paglieri for the endless patience and support.

`life`^{x} follows all the enthusiasm, passion, experience and dedication to scientific computing brought in by several people who contributed to the `LifeV` library [54]. The name itself is inspired by `LiFE` (Library of Finite Elements), conceived by Prof. Fausto Saleri.

References

[1] D. Groen, S. J. Zasada, P. V. Coveney, Survey of multiscale and multiphysics applications and communities, Computing in Science Engineering 16 (2) (2014) 34–43. doi:10.1109/MCSE.2013.47.

[2] D. E. Keyes, L. C. McInnes, C. Woodward, W. Gropp, E. Myra, M. Pernice, J. Bell, J. Brown, A. Clo, J. Connors, E. Constantinescu, D. Estep, K. Evans, C. Farhat, A. Hakim, G. Hammond, G. Hansen, J. Hill, T. Isaac, X. Jiao, K. Jordan, D. Kaushik, E. Kaxiras, A. Koniges, K. Lee, A. Lott, Q. Lu, J. Magerlein, R. Maxwell, M. McCourt, M. Mehl, R. Pawlowski, A. P. Randles, D. Reynolds, B. Rivière, U. Rüde, T. Scheibe, J. Shadid, B. Sheehan, M. Shephard, A. Siegel, B. Smith, X. Tang, C. Wilson, B. Wohlmuth, Multiphysics simulations: Challenges and opportunities, The International Journal of High Performance Computing Applications 27 (1) (2013) 4–83. doi:10.1177/1094342012468181.

[3] D. Arndt, W. Bangerth, B. Blais., M. Fehling, R. Gassmöller., T. Heister., L. Heltai, U. Köcher, M. Kronbichler, M. Maier, P. Munch, J. P. Pelteret, S. Proell, S. Konrad, B. Turcksin, D. Wells, J. Zhang, The deal.II library, version 9.3, Journal of Numerical Mathematics 29 (3) (2021) 171–186. doi:10.1515/jnma-2021-0081.
URL https://www.dealii.org/

[4] M. S. Alnaes, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, G. N. Wells, The FEniCS project version 1.5, Archive of Numerical Software 3 (2015). doi:10.11588/ans.2015.100.20553.

[5] M. W. Scroggs, I. A. Baratta, C. N. Richardson, G. N. Wells, Basix: a runtime finite element basis evaluation library, Journal of Open Source Software 7 (73) (2022) 3982. doi:10.21105/joss.03982.

[6] R. Anderson, J. Andrej, A. Barker, J. Bramwell, J.-S. Camier, J. Cerveny, V. Dobrev, Y. Dudouit, A. Fisher, T. Kolev, W. Pazner, M. Stowell, V. Tomov, I. Akkerman, J. Dahm, D. Medina, S. Zampini, MFEM: A modular finite element methods library, Computers & Mathematics with Applications 81 (2021) 42–74, development and Application of Open-source Software for Problems with Numerical PDEs. doi:10.1016/j.camwa.2020.06.009.

[7] C. J. Permann, D. R. Gaston, D. Andrš, R. W. Carlsen, F. Kong, A. D. Lindsay, J. M. Miller, J. W. Peterson, A. E. Slaughter, R. H. Stogner, R. C. Martineau, MOOSE: Enabling massively parallel multiphysics simulation, SoftwareX 11 (2020) 100430. doi:10.1016/j.softx.2020.100430.

[8] H.-J. Bungartz, F. Lindner, B. Gatzhammer, M. Mehl, K. Scheufele, A. Shukaev, B. Uekermann, preCICE – a fully parallel library for multi-physics surface coupling, Computers & Fluids 141 (2016) 250–258, advances in Fluid-Structure Interaction. doi:10.1016/j.compfluid.2016.04.003.

[9] A. Quarteroni, A. Valli, Numerical approximation of partial differential equations, Vol. 23, Springer Science & Business Media, 2008. doi:10.1007/978-3-540-85268-1.

[10] G. Wilson, D. A. Aruliah, C. T. Brown, N. P. C. Hong, M. Davis, R. T. Guy, S. H. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, et al., Best practices for scientific computing, PLOS Biology 12 (1) (2014) 1–7. doi:10.1371/journal.pbio.1001745.

[11] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. M. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M. G. Knepley, F. Kong, S. Kruger, D. A. May, L. C. McInnes, R. T. Mills, L. Mitchell, T. Munson, J. E. Roman, K. Rupp, P. Sanan, J. Sarich, B. F. Smith, S. Zampini, H. Zhang, H. Zhang, J. Zhang, PETSc Web page (2022).
URL https://petsc.org/

[12] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, K. S. Stanley, An overview of the Trilinos project, ACM Trans. Math. Softw. 31 (3) (2005) 397–423. doi:10.1145/1089014.1089021.

[13] D. Arndt, W. Bangerth, D. Davydov, T. Heister, L. Heltai, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, D. Wells, The deal.II finite element library: Design, features, and insights, Computers & Mathematics with Applications 81 (2021) 407–422, development and Application of Open-source Software for Problems with Numerical PDEs. doi:10.1016/j.camwa.2020.02.022.

[14] D. Forti, L. Dedè, Semi-implicit BDF time discretization of the Navier–Stokes equations with VMS-LES modeling in a high performance computing framework, Computers & Fluids 117 (2015) 168–182. doi:10.1016/j.compfluid.2015.05.011.

[15] J. Brown, P. Brune, Low-rank quasi-Newton updates for robust Jacobian lagging in newton methods, in: Proceedings of the 2013 International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering, 2013, pp. 2554–2565.
URL https://jedbrown.org/files/BrownBrune-LowRankQuasiNewtonRobustJacobianLagging-2013.pdf

[16] P. E. Gill, W. Murray, Quasi-Newton methods for unconstrained optimization, IMA Journal of Applied Mathematics 9 (1) (1972) 91–108. doi:10.1007/BF01585529.

[17] S. C. Eisenstat, H. F. Walker, Choosing the forcing terms in an inexact Newton method, SIAM Journal of Scientific Computing 17 (1) (1996) 16–32. doi:10.1137/0917003.

[18] U. Küttler, W. A. Wall, Fixed-point fluid–structure interaction solvers with dynamic relaxation, Computational Mechanics 43 (1) (2008) 61–72. doi:10.1007/s00466-008-0255-5.

[19] H. F. Walker, P. Ni, Anderson acceleration for fixed-point iterations, SIAM Journal on Numerical Analysis 49 (4) (2011) 1715–1735. doi:10.1137/10078356X.

[20] M. Bucelli, L. Dede', A. Quarteroni, C. Vergara, Partitioned and monolithic algorithms for the numerical solution of cardiac fluid-structure interaction (2021).
URL https://www.mate.polimi.it/biblioteca/add/qmox/78-2021.pdf

[21] J. Borgdorff, M. Mamonski, B. Bosak, K. Kurowski, M. Ben Belgacem, B. Chopard, D. Groen, P. Coveney, A. Hoekstra, Distributed multiscale computing with MUSCLE 2, the Multiscale Coupling Library and Environment, Journal of Computational Science 5 (5) (2014) 719–731. doi:10.1016/j.jocs.2014.04.004.

[22] A. Quarteroni, R. Sacco, F. Saleri, Numerical mathematics, Vol. 37, Springer Science & Business Media, 2010. doi:10.1007/b98885.
[23] A. Quarteroni, A. Valli, Domain decomposition methods for partial differential equations, Oxford University Press, 1999. URL http://infoscience.epfl.ch/record/140704
[24] S. Deparis, D. Forti, P. Gervasio, A. Quarteroni, INTERNODES: an accurate interpolation-based method for coupling the Galerkin solutions of PDEs on subdomains featuring non-conforming interfaces, Computers & Fluids 141 (2016) 22–41, advances in Fluid-Structure Interaction. doi:10.1016/j.compfluid.2016.03.033.
[25] S. Deparis, D. Forti, A. Quarteroni, A rescaled localized Radial Basis Function interpolation on non-Cartesian and non-conforming grids, SIAM Journal on Scientific Computing 36 (6) (2014) A2745–A2762. doi:10.1137/130947179.
[26] M. Salvador, L. Dede', A. Quarteroni, An intergrid transfer operator using radial basis functions with application to cardiac electromechanics, Computational Mechanics 66 (2) (2020) 491–511. doi:10.1007/s00466-020-01861-x.
[27] N. Paliwal, R. L. Ali, M. Salvador, R. O'Hara, R. Yu, U. A. Daimee, T. Akhtar, P. Pandey, D. D. Spragg, H. Calkins, N. A. Trayanova, Presence of left atrial fibrosis may contribute to aberrant hemodynamics and increased risk of stroke in atrial fibrillation patients, Frontiers in Physiology 12 (2021). doi:10.3389/fphys.2021.657452.
[28] I. Fumagalli, M. Fedele, C. Vergara, L. Dede', S. Ippolito, F. Nicolò, C. Antona, R. Scrofani, A. Quarteroni, An image-based computational hemodynamics study of the systolic anterior motion of the mitral valve, Computers in Biology and Medicine 123 (2020) 103922. doi:10.1016/j.compbiomed.2020.103922.
[29] F. Regazzoni, M. Salvador, P. Africa, M. Fedele, L. Dede', A. Quarteroni, A cardiac electromechanical model coupled with a lumped-parameter model for closed-loop blood circulation, Journal of Computational Physics 457 (2022) 111083. doi:10.1016/j.jcp.2022.111083.
[30] M. Fedele, A. Quarteroni, Polygonal surface processing and mesh generation tools for the numerical simulation of the cardiac function, International Journal for Numerical Methods in Biomedical Engineering 37 (4) (2021) e3435. doi:10.1002/cnm.3435.
[31] J. Liu, L. Dede', J. A. Evans, M. J. Borden, T. J. Hughes, Isogeometric analysis of the advective Cahn–Hilliard equation: Spinodal decomposition under shear flow, Journal of Computational Physics 242 (2013) 321–350. doi:10.1016/j.jcp.2013.02.008.
[32] C. Vergara, S. Stella, M. Maines, P. C. Africa, D. Catanzariti, C. Demattè, M. Centonze, F. Nobile, A. Quarteroni, M. Del Greco, Computational electrophysiology of the coronary sinus branches based on electro-anatomical mapping for the prediction of the latest activated region, Medical & Biological Engineering & Computing (2022). doi:10.1007/s11517-022-02610-3.
[33] R. Piersanti, P. C. Africa, M. Fedele, C. Vergara, L. Dede', A. F. Corno, A. Quarteroni, Modeling cardiac muscle fibers in ventricular and atrial electrophysiology simulations, Computer Methods in Applied Mechanics and Engineering 373 (2021) 113468. doi:10.1016/j.cma.2020.113468.
[34] S. Pagani, L. Dede', A. Frontera, M. Salvador, L. R. Limite, A. Manzoni, F. Lipartiti, G. Tsitsinakis, A. Hadjis, P. Della Bella, A. Quarteroni, A computational study of the electrophysiological substrate in patients suffering from atrial fibrillation, Frontiers in Physiology 12 (2021). doi:10.3389/fphys.2021.673612.
[35] S. Stella, C. Vergara, M. Maines, D. Catanzariti, P. C. Africa, C. Dematté, M. Centonze, F. Nobile, M. Del Greco, A. Quarteroni, Integration of activation maps of epicardial veins in computational cardiac electrophysiology, Computers in Biology and Medicine 127 (2020) 104047. doi:10.1016/j.compbiomed.2020.104047.
[36] R. Piersanti, F. Regazzoni, M. Salvador, A. Corno, C. Vergara, A. Quarteroni, et al., 3D–0D closed-loop model for the simulation of cardiac biventricular electromechanics, Computer Methods in Applied Mechanics and Engineering 391 (2022) 114607. doi:10.1016/j.cma.2022.114607.
[37] M. Salvador, F. Regazzoni, S. Pagani, L. Dede, N. Trayanova, A. Quarteroni, The role of mechano-electric feedbacks and hemodynamic coupling in scar-related ventricular tachycardia, Computers in Biology and Medicine 142 (2022) 105203. doi:10.1016/j.compbiomed.2021.105203.
[38] M. Salvador, M. Fedele, P. C. Africa, E. Sung, L. Dede', A. Prakosa, N. Trayanova, J. Chrispin, A. Quarteroni, Electromechanical modeling of human ventricles with ischemic cardiomyopathy: numerical simulations in sinus rhythm and under arrhythmia, Computers in Biology and Medicine 136 (2021) 104674. doi:10.1016/j.compbiomed.2021.104674.
[39] L. Dedè, A. Quarteroni, F. Regazzoni, Mathematical and numerical models for the cardiac electromechanical function, Atti della Accademia Nazionale dei Lincei, Classe di Scienze Fisiche, Matematiche e Naturali. Rendiconti Lincei - Matematica e Applicazioni 32 (2) (2021) 233–272. doi:10.4171/rlm/935.
[40] A. Quarteroni, L. Dedè, F. Regazzoni, Modeling the cardiac electromechanical function: A mathematical journey, Bulletin of the American Mathematical Society 59 (3) (2022) 371–403. doi:10.1090/bull/1738.
[41] A. Zingaro, I. Fumagalli, M. Fedele, P. C. Africa, L. Dede', A. Quarteroni, A. F. Corno, A geometric multiscale model for the numerical simulation of blood flow in the human left heart, Discrete and Continuous Dynamical Systems - S 15 (8) (2022) 2391–2427. doi:10.3934/dcdss.2022052.
[42] I. Fumagalli, P. Vitullo, C. Vergara, M. Fedele, A. Corno, S. Ippolito, R. Scrofani, A. Quarteroni, Image-based computational hemodynamics analysis of systolic obstruction in hypertrophic cardiomyopathy, Frontiers in Physiology (2022) 2437doi:10.3389/fphys.2021.787082.
[43] N. A. Barnafi Wittwer, S. D. Gregorio, L. Dede', P. Zunino, C. Vergara, A. Quarteroni, A multiscale poromechanics model integrating myocardial perfusion and the epicardial coronary vessels, SIAM Journal on Applied Mathematics 82 (4) (2022) 1167–1193. doi:10.1137/21M1424482.
[44] S. Di Gregorio, M. Fedele, G. Pontone, A. F. Corno, P. Zunino, C. Vergara, A. Quarteroni, A computational model applied to myocardial perfusion in the human heart: From large coronaries to microvasculature, Journal of Computational Physics 424 (2021) 109836. doi:https://doi.org/10.1016/j.jcp.2020.109836.

[45] L. Dedè, F. Regazzoni, C. Vergara, P. Zunino, M. Guglielmo, R. Scrofani, L. Fusini, C. Cogliati, G. Pontone, A. Quarteroni, Modeling the cardiac response to hemodynamic changes associated with COVID-19: a computational study, Mathematical Biosciences and Engineering 18 (4) (2021) 3364–3383. doi:10.3934/mbe.2021168.

[46] M. Fedele, R. Piersanti, F. Regazzoni, M. Salvador, P. C. Africa, M. Bucelli, A. Zingaro, L. Dede', A. Quarteroni, A comprehensive and biophysically detailed computational model of the whole human heart electromechanics (2022). doi:10.48550/ARXIV.2207.12460.

[47] P. C. Africa, M. Salvador, P. Gervasio, L. Dede', A. Quarteroni, A matrix-free high-order solver for the numerical solution of cardiac electrophysiology (2022). doi:10.48550/ARXIV.2205.05136.

[48] L. Cicci, S. Fresca, S. Pagani, A. Manzoni, A. Quarteroni, Projection-based reduced order models for parameterized nonlinear time-dependent problems arising in cardiac mechanics, Mathematics in Engineering 5 (2) (2023) 1–38. doi:10.3934/mine.2023026.

[49] F. Regazzoni, M. Salvador, L. Dedè, A. Quarteroni, A machine learning method for real-time numerical simulations of cardiac electromechanics, Computer Methods in Applied Mechanics and Engineering 393 (2022) 114825. doi:10.1016/j.cma.2022.114825.

[50] F. Regazzoni, A. Quarteroni, Accelerating the convergence to a limit cycle in 3D cardiac electromechanical simulations through a data-driven 0D emulator, Computers in Biology and Medicine 135 (2021) 104641. doi:10.1016/j.compbiomed.2021.104641.

[51] P. C. Africa, R. Piersanti, M. Fedele, L. Dede', A. Quarteroni, `life`^{x} – `heart` module: a high-performance simulator for the cardiac function. Package 1: Fiber generation (software documentation), Zenodo (01 2022). doi:10.5281/zenodo.5810268.

[52] P. C. Africa, R. Piersanti, M. Fedele, L. Dede', A. Quarteroni, life^{x} - heart module: a high-performance simulator for the cardiac function. Package 1: Fiber generation (2022). doi:10.48550/ARXIV.2201.03303.

[53] S. Alowayyed, D. Groen, P. V. Coveney, A. G. Hoekstra, Multiscale computing in the exascale era, Journal of Computational Science 22 (2017) 15–25. doi:https://doi.org/10.1016/j.jocs.2017.07.004.

[54] L. Bertagna, S. Deparis, L. Formaggia, D. Forti, A. Veneziani, The LifeV library: engineering mathematics beyond the proof of concept (2017). doi:10.48550/ARXIV.1710.06596.

MOX Technical Reports, last issues

Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

55/2022 Cavinato, L.; Pegoraro, M.; Ragni, A.; Ieva, F.
Imaging-based representation and stratification of intra-tumor Heterogeneity via tree-edit distance

54/2022 Bucelli, M.; Zingaro, A.; Africa, P. C.; Fumagalli, I.; Dede', L.; Quarteroni, A.
A mathematical model that integrates cardiac electrophysiology, mechanics and fluid dynamics: application to the human left heart

51/2022 Losapio, D.; Scotti, A.
Local Embedded Discrete Fracture Model (LEDFM)

52/2022 Fedele, M.; Piersanti, R.; Regazzoni, F.; Salvador, M.; Africa, P. C.; Bucelli, M.; Zingaro, A.; I
A comprehensive and biophysically detailed computational model of the whole human heart electromechanics

50/2022 Elías, A.; Jiménez, R.; Paganoni, A.M.; Sangalli, L.M.
Integrated Depths for Partially Observed Functional Data

53/2022 Antonietti, P.F; Cauzzi, C.; Mazzieri, I.; Melas L.; Stupazzini, M.
Numerical simulation of the Athens 1999 earthquake including simplified models of the Acropolis and the Parthenon: initial results and outlook

47/2022 Botti, M.; Di Pietro, D.A.; Salah, M.
A serendipity fully discrete div-div complex on polygonal meshes

49/2022 Botti, M.; Fumagalli, A.; Scotti, A.
Uncertainty quantification for mineral precipitation and dissolution in fractured porous media

48/2022 Gregorio, C.; Barbati, G.; Ieva, F.
A wavelet-mixed landmark survival model for the effect of short-term oscillations in longitudinal biomarker's profiles

45/2022 Franco, N.; Fresca, S.; Manzoni, A.; Zunino, P.
Approximation bounds for convolutional neural networks in operator learning