CardioMechanics

User's Manual & Documentation

Version 1.0

Dr.-Ing. Tobias Gerach

January 8, 2024

Contents

Listings

1 Introduction

CardioMechanics is a finite element analysis software for biomechanics problems specifically developed for cardiac mechanics at the Institute of Biomedical Engineering, Karlsruhe Institute of Technology (KIT). It supports passive and active tissue mechanics for heart geometries with 1-, 2-, or 4-chambers. The use of a flexible plugin interface gives access to additional functionality such as the recovery of the reference configuration, passive parameter optimization, circulatory system coupling, and coupling to a solver for cardiac electrophysiology. A direct interface to the now retired cardiac electrophysiology simulator *acCELLerate* developed by Gunnar Seemann et al. is provided but coupling to other software packages is possible if you want to put in the work required.

2 Getting Started

CardioMechanics is the main component of this software package. To make electromechanical simulations possible, the additional components *acCELLerate* and *CellModelTest* (with subcomponents *ElphyModelTest* and *ForceModelTest*) were added and can be utilized via the plugin interface of *CardioMechanics*. Currently, no precompiled version is available and the entire software has to be build from source. Make sure to follow the build instructions (docs/BUILD.md) to install from source using CMake or read the following section.

2.1 System Requirements

The following requirements have to be installed before trying to build CardioMechanics from source. We recommend using a package manager (we use macports on our macOSX systems) whenever possible.

- C and C++ compilers (e.g. gcc/g++ or clang/clang++)
- CMake
- zlib
- gfortran
- git
- make
- PETSc
- VTK
- Open MPI
- Python3 (optional, if you want to use some of the provided tools)

With installRequirements.sh we provide a script to compile Open MPI, PETSc, and VTK from source with the most recently tested versions to ensure compatibility. Building with CMake as described in the next step requires the location and version of the tools as set in the script. If you do want to use alternative locations/versions you have to link them as required. By default, the script installRequirements.sh compiles all components using a single process, which takes a considerable amount of time. If you want to speed up this process, use the command `make` with the option `-j X` where X is the number of processes you want to use. Additionally, adjust `export AUTOMAKE_JOBS=X` specifically for Open MPI.

2.2 Installation Instructions

Before continuing with compiling CardioMechanics using CMake, add the following environmental variables to your systems configuration file (e.g. .bashrc or .zshrc)

```
export kaRootDir=$HOME/CardioMechanics
export THIRDPARTY_HOME=$kaRootDir/thirdparty
export PETSC_DIR=$THIRDPARTY_HOME/macosx
export PETSC_ARCH=petsc-v3.19.1
```

and add the location of the executables to your PATH variable (replace macosx with linux if you are on a linux machine).

```
PATH="$PATH:$THIRDPARTY_HOME/macosx/openMPI-64bit/bin"
PATH="$PATH:$kaRootDir/_build/bin/macosx"
PATH="$PATH:$kaRootDir/tools/python"
export PATH
```

We assume here that you copied the repository to your `$HOME` directory. If you chose a different root directory, adjust the `kaRootDir` variable accordingly. If you used a different PETSc version in installRequirements.sh, adjust the `PETSC_ARCH` variable. Now run

```
cmake -S . -B _build
```

to create the `_build` folder and compile the code using

```
cmake --build _build
```

3 Mathematical Model

A general overview of the equations that are solved by *CardioMechanics* is given in this section. This is a shortened version of the explanation given in [1].

3.1 Modeling Cardiac Mechanics

Under normal conditions, the heart undergoes large deformations during the contraction and relaxation phases of the cardiac cycle. The associated displacement of the myocardium can be found by relating the strain of the tissue with the stress generated by the contraction of myocytes together with the pressure that is exerted onto the endocardium by the blood inside the chambers.

Considering a continuous body $\Omega_0 \subset \mathbb{R}^3$ in its reference configuration, a deformation map $\varphi\colon (0,T] \times \Omega_0 \to \mathbb{R}^3$ can be defined that relates the reference coordinates $\mathbf{X} \in \Omega_0$ to the current coordinates $\mathbf{x} \in \Omega$ such that $\mathbf{x} = \varphi(\mathbf{X},t)$ at time t. Next, the displacement field of the body is defined by $\mathbf{d}(\mathbf{X},t) = \varphi(\mathbf{X},t) - \mathbf{X}$ together with the deformation gradient $\mathbf{F}(\mathbf{X},t) = \boldsymbol{\nabla}\varphi(\mathbf{X},t) = \mathbb{1} - \boldsymbol{\nabla}\mathbf{d}$. The deformation map is injective and orientation preserving, i.e. its Jacobian satisfies $J = \det \mathbf{F} > 0$.

The fundamental equation that describes the mechanical behavior and kinematics of a solid body is based on Newton's second law of motion. The so-called momentum equation:

$$\rho \frac{\partial^2 \mathbf{d}}{\partial t^2} - \boldsymbol{\nabla} \cdot \boldsymbol{\sigma} - \rho \mathbf{b} = \mathbf{0}\,, \tag{1}$$

where ρ is the body's density, σ is the Cauchy stress, and $\mathbf{b}$ are body forces. Cauchy stress corresponds to the real or physical stress, since it refers to the current configuration. However, various stress measures can be delineated, exhibiting distinctions based on their respective frames of reference. The nominal stress tensor $\mathbf{P}$ is an asymmetric tensor defined as

$$\mathbf{P} = J\mathbf{F}^{-1}\boldsymbol{\sigma}\,, \tag{2}$$

and can be used to express the current force $\mathrm{d}\mathbf{f}$ acting on a surface Γ in terms of the normal vector $\mathbf{n}_0$ and that surface in the reference configuration Γ_0:

$$\mathrm{d}\mathbf{f} = \mathbf{n}_0 \mathbf{P}\, \mathrm{d}\Gamma_0\ . \tag{3}$$

However, due to its asymmetry, $\mathbf{P}$ is less attractive to work with. Cardiac tissue is considered an orthotropic material with fibers that have a certain orientation. Typically we are interested in the stress along the fibers. This can be done by using the second Piola-Kirchhoff stress tensor:

$$\mathbf{S} = J\mathbf{F}^{-1}\boldsymbol{\sigma}\mathbf{F}^{-\mathsf{T}}\,. \tag{4}$$

Contrary to the nominal stress $\mathbf{P}$, $\mathbf{S}$ is symmetric and compared to the Cauchy stress does not change during rigid body rotations.

Passive material properties of the myocardium are assumed to be hyperelastic and quasi-incompressible. As a consequence, the second Piola-Kirchhoff stress $\mathbf{S}$ can be defined by a stored strain energy density function Ψ after introducing the right Cauchy-Green tensor $\mathbf{C} = \mathbf{F}^{\mathsf{T}}\mathbf{F}$ and the Green-Lagrange strain tensor $\mathbf{E} = \frac{1}{2}(\mathbf{C} - \mathbb{1})$:

$$\mathbf{S} = 2\frac{\partial \Psi(\mathbf{C})}{\partial \mathbf{C}} = \frac{\partial \Psi(\mathbf{E})}{\partial \mathbf{E}}\,. \tag{5}$$

Different constitutive laws have been introduced and used in the context of cardiac modeling. Due to the presence of fibers in cardiac tissue, these constitutive laws account for anisotropy by imposing a different elastic response along the directions of fibers $\mathbf{f}_0$, sheets $\mathbf{s}_0$, and sheet-normals $\mathbf{n}_0$.

To account for the active stress generated during cardiac contraction, $\mathbf{S}$ is additively decomposed into an active ($\mathbf{S}_\mathrm{a}$) and a passive ($\mathbf{S}_\mathrm{p}$) component such that

$$\begin{aligned}\mathbf{S}(\mathbf{d}, T_\mathrm{a}(\mathbf{F})) &= \mathbf{S}_\mathrm{p} + \mathbf{S}_\mathrm{a} \\ &= 2\frac{\partial \Psi(\mathbf{C})}{\partial \mathbf{C}} + T_\mathrm{a}(\mathbf{F})\frac{\mathbf{f}_0 \otimes \mathbf{f}_0}{\sqrt{\mathbf{F}\mathbf{f}_0 \cdot \mathbf{F}\mathbf{f}_0}}\,,\end{aligned} \tag{6}$$

where T_a is the contractile force. Active stress is modeled by upscaling the microscopic force T_a generated by cardiac muscle fibers to the tissue level. Thus, expressing it in terms of an internal stress that acts in the first principal direction of the tissue, namely the fibers $\mathbf{f}_0$.

3.2 Spatial Discretization

We use finite elements to transform Equation 1 into its discrete form. The notation $\dot{u} = \frac{\partial u}{\partial t}$ and $\ddot{u} = \frac{\partial^2 u}{\partial t^2}$ is adopted for time derivatives. Using index notation in terms of the displacement $\mathbf{d}$, Equation 1 is given by

$$\rho \ddot{d}_i - \frac{\partial}{\partial x_j} \sigma_{ji} - \rho b_i = 0 \tag{7}$$

The fundamental concept of the finite element method involves the approximation of d_i within the domain of interest Ω using a trial function $\tilde{d}_i$. This trial function comprises a defined finite set of appropriate basis functions N_i, referred to as element shape functions, which exhibit vanishing behavior on the displacement boundaries. The approximation of the displacement in component i at node I reads

$$d_i \approx \tilde{d}_i = d_{iI} N_I \,. \tag{8}$$

Since $\tilde{d}_i$ is an approximation to the solution, we are left with a residual R. Following Galerkin's Method of weighted residuals, we use an arbitrary weighting function $w_i = w_{iI} N_I$ (test function) from the same function space as the trial function and build the inner product with R:

$$\int_\Omega w_{iI} \left(\rho N_I N_J \ddot{d}_{jJ} - \frac{\partial N_I}{\partial x_j} \sigma_{ji} - \rho N_I b_i \right) \mathrm{d}\Omega = 0 \,. \tag{9}$$

This is the weak form of the linear momentum equation. Using the relations

$$\begin{aligned} J \sigma_{ji} &= F_{jk} P_{ki} = \frac{\partial x_j}{\partial X_k} P_{ki} \\ \mathrm{d}\Omega &= J \, \mathrm{d}\Omega_0 \\ \rho b_i &= \rho_0 b_i \Omega_0 \end{aligned}$$

we can transform the weak form into the reference domain Ω_0 (also known as total Lagrangian form):

$$\int_{\Omega_0} w_{iI} \left(\rho_0 N_I N_J \ddot{d}_{jJ} - \frac{\partial N_I}{\partial X_j} P_{ji} - \rho N_I b_i \right) \mathrm{d}\Omega_0 = 0 \,. \tag{10}$$

Finally, we use the derivative product formula in order to get rid of the derivative of the nominal stress and apply the Gauss's theorem on the result. Then we use that w_i vanishes on the traction boundary and obtain discrete equations of the weak form of the total Lagrangian formulation:

$$w_{iI} \int_{\Omega_0} \rho_0 N_I N_J \ddot{d}_{jJ} \, \mathrm{d}\Omega_0 + w_{iI} f_{iI}^{\text{int}} - w_{iI} f_{iI}^{\text{ext}} = 0 \,, \tag{11}$$

with the internal and external nodal forces

$$f_{iI}^{\text{int}} = \int_{\Omega_0} \frac{\partial N_I}{\partial X_j} P_{ji} \, \mathrm{d}\Omega_0 \quad \text{and} \quad f_{iI}^{\text{ext}} = \int_{\Omega_0} N_I \rho_0 b_i \, \mathrm{d}\Omega_0 + \int_{\Gamma_0} N_I t_i^0 \, \mathrm{d}\Gamma_0 \,, \tag{12}$$

respectively. Since the test functions w_{iI} are arbitrary, it follows

$$M_{ijIJ} \ddot{d}_{jJ} + f_{iI}^{\text{int}} - f_{iI}^{\text{ext}} = 0 \,, \tag{13}$$

$$\text{where} \quad M_{ijIJ} = \delta_{ij} \int_{\Omega_0} \rho_0 N_I N_J \, \mathrm{d}\Omega_0 \,, \tag{14}$$

is the so-called mass matrix. Rewriting everything in matrix vector notation gives

$$\mathbf{M}\ddot{\mathbf{d}} + \mathbf{f}^{\text{int}}(\mathbf{d}(t), t) - \mathbf{f}^{\text{ext}}(\mathbf{d}(t), t) = \mathbf{0}\,. \tag{15}$$

If it is desired to take friction into account, the equation can be extended with a term for damping

$$\mathbf{M}\ddot{\mathbf{d}} + \mathbf{C}\dot{\mathbf{d}} + \mathbf{f}^{\text{int}}(\mathbf{d}(t), t) - \mathbf{f}^{\text{ext}}(\mathbf{d}(t), t) = \mathbf{0}\,. \tag{16}$$

The type of damping used in *CardioMechanics* is called Rayleigh damping with the damping matrix

$$\mathbf{C} = \alpha\mathbf{M} + \beta\mathbf{K} \tag{17}$$

using the linearization of the internal forces $\mathbf{f}^{\text{int}}(\mathbf{d}(t), t) \approx \mathbf{K} \cdot \mathbf{d}(t)$ and the stiffness matrix $\mathbf{K} := \boldsymbol{\nabla}_{\mathbf{d}}\mathbf{f}^{\text{int}}(\mathbf{d}(t), t)$.

In the case of static equilibrium, where mass inertia can be neglected, all time derivatives vanish and Equation 16 reduces to

$$\mathbf{f}^{\text{int}}(\mathbf{d}) - \mathbf{f}^{\text{ext}}(\mathbf{d}) = \mathbf{0}\,. \tag{18}$$

3.3 Time Discretization

Now that the equations are discretized in space, the temporal derivatives $\dot{\mathbf{d}}$ and $\ddot{\mathbf{d}}$ have to be discretized in time as well. For better readability, we redefine $\mathbf{a} = \ddot{\mathbf{d}}$ and $\mathbf{v} = \dot{\mathbf{d}}$ and use $\tilde{()}$ for incremental solutions.

Since Equation 18 omits all time derivatives and yields a solution in static equilibrium with respect to the current boundary conditions, there is no need for time integration and the system

$$\mathbf{r}(\mathbf{d}) := \mathbf{f}^{\text{int}}(\mathbf{d}) - \mathbf{f}^{\text{ext}}(\mathbf{d}) = \mathbf{0}\,, \tag{19}$$

can be solved directly using Newton's method.

A suitable time integration method for Equation 16 is Newmark-beta time integration. It was designed for elasticity problems and can be set up as an explicit or an implicit method depending on the choice of parameters β and γ. Equation 16 is an initial value problem, which can be discretized using $\Delta t = t^{n+1} - t^n$, $\mathbf{d}(t^0) = \mathbf{d}^0$, $\mathbf{v}(t^0) = \mathbf{v}^0$, $\mathbf{a}(t^0) = \mathbf{a}^0$:

$$\mathbf{r}(\mathbf{d}^{n+1}) = \mathbf{M}\mathbf{a}^{n+1} + \mathbf{C}\mathbf{v}^{n+1} + \mathbf{f}^{\text{int}}(\mathbf{d}^{n+1}, t^{n+1}) - \mathbf{f}^{\text{ext}}(\mathbf{d}^{n+1}, t^{n+1}) \overset{!}{=} 0 \tag{20}$$

To solve this equation with the Newmark-beta scheme, we first calculate predictor values for the displacement and the velocity which depend on the previous time step

$$\tilde{\mathbf{d}}^{n+1} = \mathbf{d}^n + \Delta t\mathbf{v}^n + \frac{\Delta t^2}{2}(1 - 2\beta)\mathbf{a}^n\,, \tag{21}$$

$$\tilde{\mathbf{v}}^{n+1} = \mathbf{v}^n + (1 - \gamma)\Delta t\mathbf{a}^n\,, \tag{22}$$

and substitute $\mathbf{d}^{n+1}$, $\mathbf{v}^{n+1}$, and $\mathbf{a}^{n+1}$ in Equation 20 with the following formulas

$$\mathbf{d}^{n+1} = \mathbf{d}^n + \Delta t\mathbf{v}^n + \frac{\Delta t^2}{2}\left((1 - 2\beta)\mathbf{a}^n + 2\beta\mathbf{a}^{n+1}\right)\,, \tag{23}$$

$$\mathbf{v}^{n+1} = \tilde{\mathbf{v}}^{n+1} + \gamma\Delta t\mathbf{a}^{n+1}\,, \tag{24}$$

$$\mathbf{a}^{n+1} = \frac{1}{\beta\Delta t^2}(\mathbf{d}^{n+1} - \tilde{\mathbf{d}}^{n+1})\,, \tag{25}$$

which leads to a system of nonlinear equations that only depends on $\mathbf{d}^{n+1}$:

$$\begin{aligned}\mathbf{r}(\mathbf{d}^{n+1}) =& \mathbf{M}\left[\frac{1}{\beta\Delta t^2}(\mathbf{d}^{n+1} - \tilde{\mathbf{d}}^{n+1})\right] + \mathbf{C}\left[\frac{\gamma}{\beta\Delta t}(\mathbf{d}^{n+1} - \tilde{\mathbf{d}}^{n+1}) + \tilde{\mathbf{v}}^{n+1}\right] \\ &+ \mathbf{f}^{\text{int}}(\mathbf{d}^{n+1}, t^{n+1}) - \mathbf{f}^{\text{ext}}(\mathbf{d}^{n+1}, t^{n+1}) = 0 \end{aligned} \tag{26}$$

The most robust method to solve non-linear algebraic equations such as Equation 26 is Newton's method. Linearizing Equation 26 using a Taylor expansion around the current value of the displacement $\mathbf{d}^{n+1}$ and dropping terms of higher order results in the linear model

$$\mathbf{r}(\mathbf{d}^{n+1}, t^{n+1}) + \mathbf{A}\Delta\mathbf{d}^{n+1} = 0\,, \tag{27}$$

with the system Jacobian matrix

$$\mathbf{A} = \frac{1}{\beta\Delta t^2}\mathbf{M} + \frac{\gamma}{\beta\Delta t}\mathbf{C} + \mathbf{K}\,, \tag{28}$$

where internal and external forces $\mathbf{K}$ are linearized in each Newton iteration:

$$\mathbf{K}\mathbf{d}^{n+1} \approx \mathbf{f}^{\text{int}}(\mathbf{d}^{n+1}, t^{n+1}) - \mathbf{f}^{\text{ext}}(\mathbf{d}^{n+1}, t^{n+1})\,. \tag{29}$$

Finally, we calculate the updated values for velocity and acceleration using Equation 24 and Equation 25.

3.4 Units

In general, all units are given in the international system of units (SI). For the most important units you can refer to Table 1. If you are unsure about the unit of parameters not listed in Table 1, use base SI units without prefix.

Table 1: Units used in *CardioMechanics*.

Symbol	Input unit	Internal unit	Conversion	Comment
x, y, z	mm	m	10^{-3}	-
t	s	s	1	-
$\mathbf{d}$	m	m	1	-
ρ_0	$\mathrm{kg\,m^{-3}}$	$\mathrm{kg\,m^{-3}}$	1	-
T_a	Pa	Pa	1	-
$\mathbf{S}$	Pa	Pa	1	-
p	Pa	Pa; mmHg	1	decision by plugin

4 File Formats

4.1 Input Files

4.1.1 CardioMechanics Config File (*.xml)

This is the main configuration file that controls the mechanical simulations. It uses the extensible markup language (XML) to store and define all simulation parameters. The top level tags are defined as follows: `General`, `Mesh`, `Materials`, `Export`, `Solver`, and `Plugins`. All available settings inside these tags are described in detail within the Simulation Framework.

```
<settings>A general description</settings>

<General>
    <LogFile>   filename.txt   </LogFile>
    <Verbose>   false/true       </Verbose>
    <Debug>   false/true    </Debug>
</General>

<Mesh>
...
</Mesh>

<Materials>
...
</Materials>

<Export>
...
</Export>

<Solver>
...
</Solver>

<Plugins>
...
</Plugins>
```

Listing 1: CardioMechanics Config File (*.xml)

4.1.2 Element file (*.ele)

The element file contains only the elements for the mechanical mesh and has the following structure:

```
    <# elements> <Nodes per Element> <# Attributes>
    <Element Index> <node 1> <node 2> ... <node n> <material index>
    ...
```

Listing 2: Element file (*.ele)

Only P1 and P2 elements are supported, hence each element needs either $n = 4$ or $n = 10$ nodes.

4.1.3 Node File (*.node)

The node file contains only the vertices for the mesh and has the following structure:

```
<# nodes> <Dimension (must be 3)> <Boundary Conditions> (0: no boundary
 condition, 1: Dirichlet)> <BoundaryMarker (must be 0)>
<Node Index> <x> <y> <z> <Boundary Condition>
...
```

Listing 3: Node File (*.node)

The boundary condition is set as an integer number between 0 and 7. These numbers translated to their respective binary representation describe the Dirichlet boundary condition in the axes z,y,x, i.e. 001 (dec: 1) -> x-direction fixed, 010 (dec: 2) -> y-direction fixed, 100 (dec: 4) -> z-direction fixed, 111 (dec: 1+2+4 = 7) all directions fixed. Combinations of the aforementioned fixations are possible as well, i.e. you can fix a node in the x,y-plane by using the integer number 3 (binary: 011) etc.

4.1.4 Surface Elements File (*.sur)

This file contains a list of all surface triangle node indices of the mechanical mesh and its attributes.

```
<# elements> <Nodes per Element> <# Attributes (min 2; max 3)>
<surface element Index> <node 1> <node 2> ... <node n> <material index>
 <surface index> <surface traction scaling>
...
```

Listing 4: Surface Elements File (*.sur)

Only P1 and P2 elements are supported, hence $n = 3$ or $n = 6$. You need a minimum of two or a maximum of three attributes defined for each surface element. Attribute 1 is the material index, which is used as a material tag during export. Attribute 2 is the surface index, which is used by boundary conditions. Attribute 3 is optional and sets a scaling factor which can be applied to epicardial boundary conditions (ContactHandling, RobinBoundary, RobinBoundaryGeneral).

4.1.5 Fiber Orientation File (*.bases)

This file contains a list of all fiber orientations of the mechanical mesh. If you intend to use P2 elements, you have to submit the fiber orientations for each quadrature point as well. Either way, the first orientation is always the centroid of the element. This means it is possible to use a file for P2 elements with P1 elements as well, since the program will only read the number of quadrature points given in the file.

```
<# elements> <# qp>
<element index> <fiber_x> <fiber_y> <fiber_z> <sheet_x> <sheet_y> <
sheet_z> <normal_x> <normal_y> <normal_z>
...
```

Listing 5: Fiber Orientation File (*.bases)

4.1.6 Local Activation Time File (*.txt)

This file contains activation times for each element of the mesh that defines the start of the contraction cycle in the tension model. If a tension model requires a calcium transient, you need to make sure to provide one. This is only relevant to the paragraph 5.2.2 model, which has multiple calcium transients to choose from already implemented. Leave an empty line at the end of the file.

```
    <# elements> 1
    <element 1> <activation time in s>
    <element 2> <activation time in s>
    <element 3> <activation time in s>
    ...
    <element n> <activation time in s>
```

Listing 6: Local Activation Time File (*.txt)

4.2 Output Files

4.2.1 VTK Unstructured Grid Files (*.vtu)

This file contains all data specified by the Export options and the `SolverPlugin::Export()` function of each active plugin as long as the plugin supports exporting to VTK. Refer to Table 3 for an overview of export options to the VTK file type. A file is written at each `Export.TimeStep`.

4.2.2 ParaView Data File (*.pvd)

This file can be used to open, view, and post process the time series of VTK Unstructured Grid Files (*.vtu) files in ParaView. Refer to the ParaView Wiki on data formats for more information.

4.3 acCELLerate Specific Input Files

The input files listed in the following sections are specific to *acCELLerate* and are only required if you intend to use the solver plugin acCELLerate to run electromechanically coupled simulations.

4.3.1 Config File (*.aclt)

This is the main configuration file for EP tissue simulations. Not all listed entries are needed for, e.g. Monodomain simulations, but possible entries are

```
Resprefix      The prefix of the result data
Condition      Name of the condition file
Sensor         Name of the sensor file
CalcLength     Calculation length [s]
DTCell         Time step of cell model calculation [s]
DTExtra        Time step of extracellular calculation [s]
DTIntra        Time step of intracellular calculation [s]
```

```
BeginSave       Time at which saving of results starts [s]
DTSave          Time step of saving [s]
CellModelFile   Name of the cell model file
Material        Name of vector describing tissue classes
MaterialFibro   Name of vector describing fibroblast classes
MatrixIntra     Name of intracellular conductivity matrix
MatrixExtra     Name of extracellular conductivity matrix
MatrixExtraCombined       Name of extracellular conductivity matrix that is
     already combined with Intra (and Fibro)
MatrixFibro     Name of fibroblast conductivity matrix
BetaMyoFib      Number of Myo-Fibro gap junctions [1/m^3]
RMyoFib         Resistor of single myo-fibro gap junction [Ohm]
VolumeMyo       Relative amount of myocyte volume
VolumeExtra     Relative amount of extracellular volume
VolumeFibro     Relative amount of fibroblast volume
Domain          Type, either "Mono", "Bi", or "Tri"
Verbose         Print verbose information into terminal
Protocol        Name of verbose information file
HeteroFileIntra      Name of the heterogeneous cell model file (entries: <
    vector file name> <parameter name>)
HeteroFileFibro     Name of the heterogeneous fibro model file (entries: <
    vector file name> <parameter name>)
LoadBackup      Loads backup from standard file of <file> if declared
SaveBackup      Saves backup to standard file of <file> if declared
DTBackup        Time step of backup
Results         List of variables to be saved (e.g., Vm, Ve, AT, m, Na_i,
    I_Na, and any variable of the cell model)
Implicit        Number of Jacobi iterations for implicit calculation (!Only
    for Monodomain)
Force           Also calculate and output force
MassMatrixIntra         Name of intracellular mass matrix
ThetaIntra      Theta for intra PDE solver scheme. [0.0..1.0] 0=explicit, 1=
    implicit (default: 0 (FD) or 0.5 (FE))
MembraneCapacitance      Membrane capacitance per unit area, number or
    vector (F/m^2)
SurfaceToVolume       Myocyte surface to volume ratio, number or vector (1/
    m, default: cell model)
Gauss           Matrices for gauss interpolation and integration
ActivationThreshold       TMV threshold for activation time (default: 0V).
CurrentScheme   Godunow, SVI, ICI, None. SVI and None require Gauss matrices
    .
IntraIndexSet   PETSc IndexSet of the intracellular domain (if smaller than
    the xtracellular domain)
Compress        Compress output vectors using deflate/gzip.
NoExport        Deactivate Export
```

Listing 7: acCELLerate configuration file (*.aclt)

4.3.2 Condition File (*.acnd)

This file contains the stimuli and initial/boundary conditions for an EP simulation.

```
    <node index> <amplitude> <cycle length> <duration> <temporal offset> Ii
    |Ie|If|Vm|Ve|Vf
    ...
```

Listing 8: acCELLerate condition file (*.acnd)

The type of stimulus (last column) is given as a combination of first letter: either I(current) or V(voltage); domain of stimulus as second letter: intracellular=i, extracellular=e or fibroblastic=f.

4.3.3 Cell Model File (*.cmf)

Within the cell model file, the tissue classes from the geometry file are assigned to cell model types and force models.

```
    <first tissue class> <elphpy model> <force model>
    <second tissue class> <elphpy model> <force model>
    <third tissue class> <elphpy model> <force model>
    ...
```

Listing 9: acCELLerate cell model file (*.cmf)

Elphy model and force model are given as the path to the .ev-file and .fv-file you want to use.

4.3.4 Material File (*.def)

Defines material properties, such as conductivities. Each material that exists in the *Material* array of the geometry has to be defined in this file with the following entries:

```
[NAME MATERRIALNUMBER COLORCODE]
Kappa           FLOAT FLOAT
AnisotropyX     FLOAT
AnisotropyY   FLOAT
AnisotropyZ   FLOAT
```

Listing 10: acCELLerate material file (*.def)

Kappa is the conductivity (second column, in S/m) at a given frequency (first column). The COLORCODE value is currently not in use and can be set to an arbitrary number. MATERIALNUMBER should be of type INT and should stay in the range 1 to 255 (same in the geometry array!). You can use the anisotropy parameters to define different conductivites in fiber (AnisotropyX), sheet (AnisotropyY), and normal (AnisotropyZ) directions. Kappa by itself defines the transversal conductivities, such that only AnisotropyX would need to be defined in case of transverse isotropy. A second option to define a material is by using a definition of another material by referencing it with a hashtag followed by the material number if the conductivities are the same. In the following example, we set a general material for the ventricles and atria to define the left/right atrium and ventricle.

```
[Ventricle_myo 100 0]
Kappa           0 0.28
AnisotropyX     1.0
AnisotropyY   0.65
AnisotropyZ   0.35

[Atria_myo 200 0]
Kappa           0 0.1823
AnisotropyX     3.75

[Ventrikel_rechts 2 0]
#100
```

```

[Ventrikel_links 3 0]
#100

[right_atrium 32 0]
#200

[left_atrium 33 0]
#200
```

Listing 11: acCELLerate material file example

4.3.5 Sensor File (*.txt)

The sensor files contain the information about the sensors that are placed in a simulated tissue. A sensor records any output parameter of the cell model in the applied node and saves the values in a textfile.

```
    <node index> <file name> <offset> <t_increment> Ii|Ie|If|Vm|Ve|Vf (or
   any output parameter of the applied cellmodel)
    ...
```

Listing 12: acCELLerate sensor file (*.txt)

4.3.6 CellModel Parameter File (*.ev)

Cellmodel files define parameters and initial values for ionic models. The general structure is given below. Parameter files for all ionic models can be found in electrophysiology/data/.

```
CELLMODEL*IBT*KA
CELLMODELNAME
# of parameters to read
...
...
...
```

Listing 13: Ionic model parameter file (*.ev)

4.3.7 ForceModel Parameter File (*.fv)

Files with *.fv* ending contain parameters and initial values for tension models. Their structure is the same as for *.ev* files described above.

5 Simulation Framework

CardioMechanics is a robust object-oriented programming framework featuring meticulously crafted interfaces to facilitate extensible implementations. This framework seamlessly couples various model components while ensuring swift solutions through efficient utilization of computational resources via software parallelization using MPI and adaptive time increments. In this section, we delve into the pivotal elements of *CardioMechanics* , the framework tailored for conducting cardiac biomechanics (CB) simulations. This framework is intricately connected to two external libraries: the VTK library, employed for efficient data export, and the PETSc library, used for matrix vector operations and to solve the dynamic linear and non-linear equation systems. Programmed in C++, *CardioMechanics* adopts a comprehensive object-oriented programming structure, emphasizing its role as a multiscale electromechanical simulation framework. It uses the method of finite elements, to solve the equilibrium of forces within the simulation domain. The fundamental class structure of *CardioMechanics* is shown in Figure 1 and consists of the following components:

CBCardioMechanics presents a user-friendly command line interface, proficient in parsing user input and initiating core components based on the provided instructions. Configuration files, adhering to the XML format, serve as conduits for parameters that configure the Model/Mesh, Solver, Exporter, Materials, and all SolverPlugins. Notably, all parameters are subject to override via the command line, facilitating swift testing of various configurations, such as adjusting time steps. Additionally, users have the flexibility to select different levels of verbosity for logging data.

CBModel manages all data encompassing geometric information, undergoing population during mesh loading courtesy of CBModelLoader. Its primary role lies in conveying geometric details to the solver. Additionally, it plays a pivotal role in exporting simulation data to VTK Unstructured Grid Files (*.vtu) files. Consequently, it undergoes updates with the most recent node positions and is enriched with supplementary information, including active stress, relative fiber length, and the Cauchy deformation tensor. This augmented dataset is then handed over to CBModelExporter.

CBSolver tackles a nonlinear problem within each time step, leveraging PETSc's SNES-Solve function to address nonlinear equation systems. It incorporates time integration in one of its specialized derived classes, namely CBSolverStatic and CBSolverNewmarkBeta. Responsible for overarching tasks such as step size computation, invocation of solver plugins, and mesh exporting, the CBSolver's Run() function orchestrates these operations. Featuring distinct data structures formatted for efficient computation, including parallel PETSc matrices and vectors, CBSolver necessitates conversion to a CBModel object before exporting. Instead of directly computing new node coordinates, the SolverStep() function calculates displacements in each time step, subsequently adding them to node positions from the preceding time step.

CBSolverPlugin serves as a comprehensive interface to extend the solver's capabilities. The class defines essential functions such as Init(), Apply(), StepBack(), and Export(). These functions are invoked before commencing the simulation, during each solver step for updating internal information, after each unsuccessful solver step for reverting to the previous state, and after each successful solver step, complying with an export time condition, for writing information into the model and/or a text file. SolverPlugins wield influence over the subsequent time step through return codes like CBStatus::SUCCESS (indicating permission for the solver to advance to the next time step), CBStatus::FAILED (suggesting a reduction in time step due to plugin dissatisfaction with results), or CBStatus::REPEAT (prompting the solver to recompute the current time step, enabling plugins to iterate multiple times).

CBElement The element structure comprises two distinct types: CBElementSolid and CBElementSurface. CBElementSolid is equipped with functions that populate the stiffness matrix with node derivative entries in a PETSc-compatible format. These functions are implicitly invoked by PETSc's SNESSolve() function, which employs the solver's Nodal-ForcesHelperFunction() to call each solid element's CalcNodalForces(). This calculation determines the element's contribution to the system matrix, necessitated by the intricate structure dictated by PETSc's SNESSolve(). Unfortunately, this core functionality of the solver had to be implemented in a somewhat convoluted manner due to these constraints. Each CBElementSolid is self-aware, maintaining knowledge of its current and initial node positions, fiber directions, material model, and tension model. This information is instrumental in computing its deformation tensor and determining the resulting contribution to the global energy derivatives. Each solid element is associated with a CBConstitutiveModel for passive stress computation and a CBTensionModel for active stress computation during its CalcNodalForces() operation. In contrast, CBElementSurface is designed to handle surface forces application, volume computation, and identification of contact partners for the contact handling algorithm.

CBElementAdapter serves as a central hub, holding pointers to critical global and element-specific components within *CardioMechanics* for the sake of efficiency. Examples include pointers to the solver, constitutive model, time-stepping object, and other essential elements. The primary purpose is twofold: firstly, to grant each element read access to information that is technically not available within the element, such as global variables. To minimize memory consumption, node positions are stored only once in memory, facilitated by the solver's solution vector, with CBElementAdapter providing the location in memory. Secondly, this global object enables elements to directly contribute to mass and stiffness matrices within the solver's data objects, fostering seamless integration of element-specific computations. This approach optimizes both memory usage and computational efficiency throughout *CardioMechanics*.

CBConstitutiveModel provides the functions CalcEnergy() and CalcPK2Stress(), empowering each solid element to compute its deformation energy based on a provided deformation tensor, considering the passive material properties.

CBTensionModel offers functions for computing active energy contributions arising from contracting muscle fibers, utilizing a given deformation tensor. Specifically, CalcActiveTension() calculates the tension in the fiber direction, while CalcActiveStressTensor() determines the contribution to the PK2 stress. The structure closely aligns with CBConstitutiveModel, reflecting a consistent approach to handling active stress computations within the framework.

The object-oriented nature of the codebase allows for the seamless interchangeability and extension of specialized modules. For example, specializations of the CBElement class include CBElementSolidT4 and CBElementSolidT10. Similarly, specializations of the CBConstitutiveModel class encompass CBConstitutiveModelGuccione and CBConstitutive-ModelMooneyRivlin. Despite their unique functionalities, all these specializations adhere to the same interface functions, ensuring compatibility and ease of integration within the system.

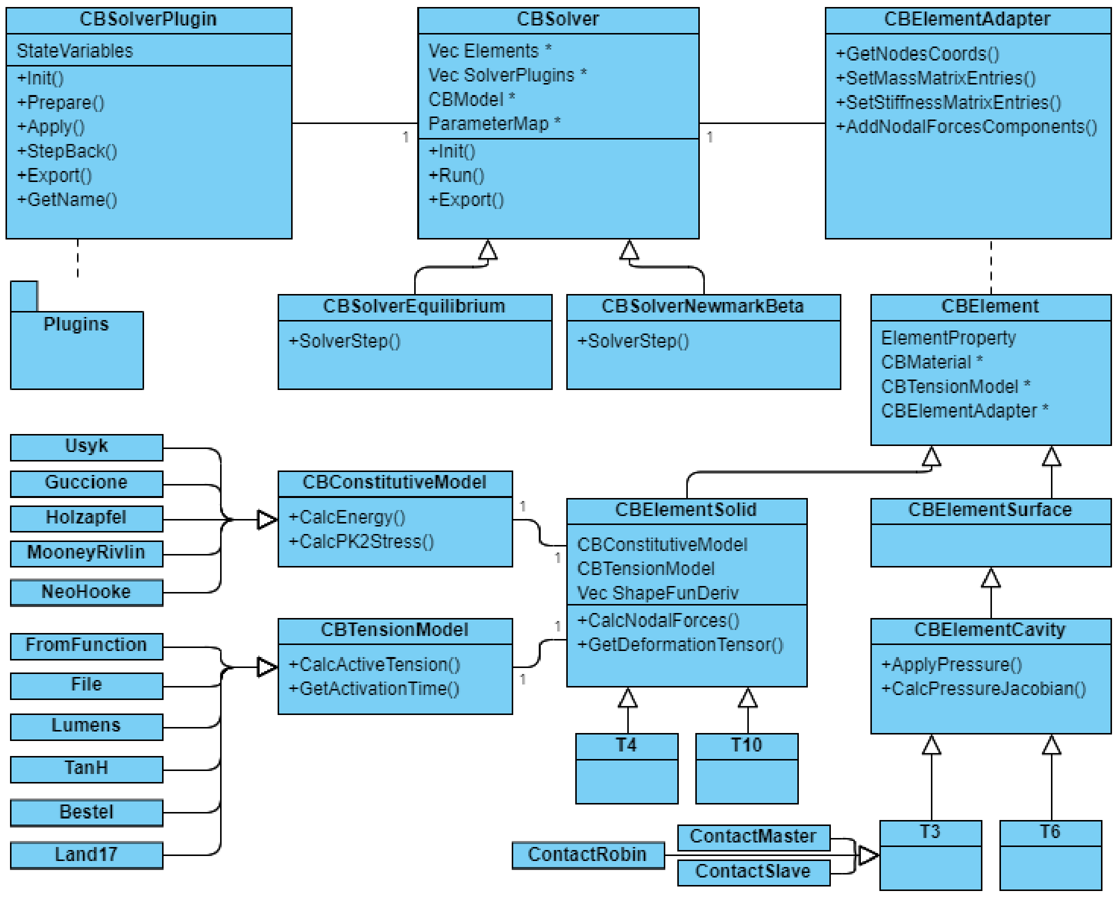

Figure 1: Fundamental structure of the simulation framework *CardioMechanics*.

5.1 Mesh

The tag `Mesh`, contains all parameters for the configuration of the model/geometry. The mesh has to be provided in the TetGen format. See Node File (*.node), Element file (*.ele), Surface Elements File (*.sur), and Fiber Orientation File (*.bases) for details about this format. Only linear (`T4`) and quadratic tetrahedral (`T10`) elements are supported as volume elements. During the initialization process, you can choose the method of domain decomposition. This can either be done by node index (`None`) or by principal component analysis (`PCA`) of initial node coordinates. A third variant is available using the `Solver.DomainDecomposition` tag, which uses information from the stiffness matrix to minimize communication cost (becomes more relevant in HPC applications).

Surfaces and surface types for boundary condition applications have to be defined using the `Surface` tag. Only linear (`T3`) and quadratic (`T6`) triangular elements are supported. However, most plugins supplying traction forces are only implemented for linear triangles. See the code example below for available options.

The tag `Transform` can be used to transform linear to quadratic elements on the fly. This is only recommended for testing purposes.

Finally, a Local Activation Time File (*.txt) can be defined, if you want to run mechanical simulations with precalculated electrophysiological activations, e.g. from solutions to the eikonal equation.

```xml
<Mesh>
    <Type>    T4/T10    </Type>
    <Format>  Tetgen    </Format>
    <Sorting> None/PCA  </Sorting>
    <Tetgen>
        <Unit>  1e-3      </Unit>
        <Nodes> filename.node </Nodes>
        <Elements>  filename.ele  </Elements>
        <Surfaces>  filename.sur  </Surfaces>
        <Bases> filename.bases  </Bases>
        <DetermineNeighbors>    true/false    </DetermineNeighbors>
        <EnforceOrthonormalBases> true/false    </EnforceOrthonormalBases>
    </Tetgen>

     <!-- Define a surface with material number XXX and one of the
    following surface types:
     CAVITY - for closed surfaces used in the circulatory system
     T3, T6 - arbitrary surfaces
     CONTACT_ROBIN - for a generalized Robin boundary condition
     CONTACT_MASTER/CONTACT_SLAVE - for the contact handling plugin based
    on Fritz et al. 2013 -->
    <Surfaces>
        <Surface_XXX> surfaceType </Surface_XXX>
    </Surfaces>

    <!-- Transform linear elements to quadratic elements on the fly -->
    <Transform>
      <T4toT10>   false </T4toT10>
      <T3toT6>  false </T3toT6>
    </Transform>

    <!-- Define a filename with local activation times (LAT) -->
    <LAT>
        <FilePath> filename.txt </FilePath>
    </LAT>
</Mesh>
```

Listing 14: .xml settings for mesh initialization

5.2 Materials

The tag `Materials` contains all parameters for the definition of material properties. Using the tag `Global.Damping`, the user should define the parameters for Rayleigh damping of the form $\mathbf{C} = \alpha\mathbf{M} + \beta\mathbf{K}$. Furthermore, the user can set default material parameters using the tag `Mat_Default`. The parameters defined under this scope are valid for all materials unless specific material definitions are available. You can define different constitutive models and tension models for each individual material if you wish to do so.

In the example below, default parameters for the density and the Usyk constitutive law are set. If the tag `IgnoreCorruptElements` is set to `true`, elements with $\det(\mathbf{F}) \leq 0$ are ignored and do not contribute to the stiffness matrix. For obvious reasons, this setting is not recommended for use. Additionally, a material defined by number 30 is defined. It uses the Usyk constitutive law with additional scaling applied to the default parameters as well as the Land17 tension model. `TensionMax` is set to 1000, since tension in the Land model is in kPa and we need to scale it to Pa.

In the following sections, the configurations for all available constitutive laws and tension models are discussed.

```xml
<Materials>
    <Global>
        <Damping>
            <Rayleigh>
                <Alpha> 500 </Alpha>
                <Beta> 0.005 </Beta>
            </Rayleigh>
        </Damping>
    </Global>

    <Mat_Default>
        <IgnoreCorruptElements> true/false </IgnoreCorruptElements>
        <Density> 1082 </Density>
        <Usyk>
            <a>880</a> <!-- Pa -->
            <bff>8</bff>
            <bss>6</bss>
            <bnn>3</bnn>
            <bfs>12</bfs>
            <bfn>3</bfn>
            <bns>3</bns>
            <k>650e3</k> <!-- Pa -->
        </Usyk>
    </Mat_Default>

    <Mat_30>
        <Type>Usyk</Type>
        <Usyk>
            <aScale>0.07</aScale>
            <bScale>3.55</bScale>
        </Usyk>

        <TensionMax>1e3</TensionMax>
        <TensionModel>Land17</TensionModel>
        <Land17>
            <CalciumTransientType>Elphy</CalciumTransientType>
            <rateDependancy>ON</rateDependancy>
            <cycleLength>0.8</cycleLength>
            <beta0>0.7</beta0>
            <beta1>-1.2</beta1>
            <Ca50>0.805</Ca50>
            <Tref>440</Tref>
        </Land17>
    </Mat_30>

    ...
</Materials>
```

Listing 15: .xml settings for material definitions

5.2.1 Constitutive Models

The material constitutive laws that are used in the balance of momentum equation relate the strain of the body with its stress. However, in cardiac mechanics, the focus is on elastic materials or in particular hyperelastic ones, i.e. the behaviour of the material is path-independent. As a consequence, we can define a stored strain energy density function Ψ to express the first Piola-Kirchhoff stress tensor as

$$\mathbf{P}(\mathbf{F}(\mathbf{X}),\mathbf{X}) = \frac{\partial\Psi(\mathbf{F}(\mathbf{X}),\mathbf{X})}{\partial\mathbf{F}}\,, \tag{30}$$

or in terms of the symmetric second Piola-Kirchhoff stress tensor

$$\mathbf{S}(\mathbf{C}(\mathbf{X}),\mathbf{X}) = 2\frac{\partial\Psi(\mathbf{C}(\mathbf{X}),\mathbf{X})}{\partial\mathbf{C}} = \frac{\partial\Psi(\mathbf{E}(\mathbf{X}),\mathbf{X})}{\partial\mathbf{E}}\,. \tag{31}$$

If the relationship between Ψ and $\mathbf{C}$ is independent of the material axes, Ψ can be expressed as a function of the invariants of $\mathbf{C}$ as

$$\Psi(\mathbf{C}(\mathbf{X}),\mathbf{X}) = \Psi(\mathrm{I}_\mathbf{C},\mathrm{II}_\mathbf{C},\mathrm{III}_\mathbf{C},\mathbf{X})\,, \tag{32}$$

where the invariants are defined as

$$\mathrm{I}_\mathbf{C} = \operatorname{tr}\mathbf{C} = \mathbf{C} : \mathbb{1} \tag{33}$$

$$\mathrm{II}_\mathbf{C} = \frac{1}{2}\left((\operatorname{tr}\mathbf{C})^2 - \operatorname{tr}\mathbf{C}^2\right) \tag{34}$$

$$\mathrm{III}_\mathbf{C} = \det\mathbf{C} = J^2\,. \tag{35}$$

It can also be convenient to consider isochoric invariants, replacing $\mathbf{C}$ with $\bar{\mathbf{C}} = J^{-2/3}\mathbf{C} = \mathrm{III}_\mathbf{C}^{-1/3}\mathbf{C}$. Furthermore, pseudo invariants can be introduced that describe the squared stretch along microstructural directions in anisotropic materials such as the myocardium:

$$\mathrm{I}_{ab} = \mathbf{C} : \operatorname{sym}(\mathbf{e}_a \otimes \mathbf{e}_b), \quad a,b \in \{f,s,n\}\,, \tag{36}$$

where $\operatorname{sym}(\mathbf{A}) = \frac{1}{2}(\mathbf{A} + \mathbf{A}^\mathrm{T})$ is the symmetric transformation. Near incompressibility can be achieved in hyperelastic materials by adding a volumetric energy component $U(J)$ to the distortional component $\Psi(\bar{\mathbf{C}})$:

$$\Psi(\mathbf{C}) = \Psi(\bar{\mathbf{C}}) + U(J)\,, \tag{37}$$

and

$$\mathbf{S} = 2\frac{\partial\Psi(\bar{\mathbf{C}})}{\partial\mathbf{C}} + 2\frac{\mathrm{d}U(J)}{\mathrm{d}J}\frac{\partial J}{\partial\mathbf{C}}\,. \tag{38}$$

In the following, the constitutive models that are implemented in *CardioMechanics* are described with their respective parameters.

Guccione The incompressible strain energy density by Guccione et al. [2] is given by

$$\begin{aligned}\Psi^{(\mathrm{G})}(\mathbf{C}) &= \frac{\kappa}{2}(J-1)^2 + \frac{\mu}{2}(\exp(Q)-1)\,,\\ Q &= b_\mathrm{f}E_\mathrm{ff}^2 + b_\mathrm{t}(E_\mathrm{ss}^2 + E_\mathrm{nn}^2 + E_\mathrm{ns}^2 + E_\mathrm{sn}^2) + b_\mathrm{fs}(E_\mathrm{fs}^2 + E_\mathrm{sf}^2 + E_\mathrm{fn}^2 + E_\mathrm{nf}^2)\,,\end{aligned} \tag{39}$$

where $E_{\mathrm{ij}} = \mathbf{E}\mathbf{i}_0 \cdot \mathbf{j}_0$ for $i, j \in \{f, s, n\}$ are the entries of the Green-Lagrange strain tensor $\mathbf{E} = \frac{1}{2}(\mathbf{C} - \mathbb{1})$. The 2nd Piola-Kirchhoff stress is given by

$$\mathbf{S} = \mu \exp(Q)(b \odot E)_{ij} + \kappa(J-1)J\mathbf{C}^{-1} \quad \text{with} \quad \mathbf{b} = \begin{pmatrix} b_{\mathrm{f}} & b_{\mathrm{fs}} & b_{\mathrm{fs}} \\ b_{\mathrm{fs}} & b_{\mathrm{t}} & b_{\mathrm{t}} \\ b_{\mathrm{fs}} & b_{\mathrm{t}} & b_{\mathrm{t}} \end{pmatrix}, \tag{40}$$

where $(A \odot B)_{ij} = (A)_{ij}(B)_{ij}$ is the Hadamard or Schur product of the matrices $\mathbf{A}$ and $\mathbf{B}$.

```
    <Type>Guccione</Type>
    <Guccione>
        <C>FLOAT</C> <!-- shear modulus [Pa] -->
        <bf>FLOAT</bf>
        <bt>FLOAT</bt>
        <bfs>FLOAT</bfs>
        <K>FLOAT</K> <!-- bulk modulus [Pa] -->
        <aScale>FLOAT</aScale> <!-- optional scaling for C (default 1) -->
        <bScale>FLOAT</bScale> <!-- optional scaling for b (default 1) -->
    </Guccione>
```

Listing 16: .xml settings for Guccione material

MooneyRivlin The Mooney-Rivlin material is a commonly used isotropic material law. The following more general form is implemented in *CardioMechanics*:

$$\Psi^{(\mathrm{MR})}(\mathrm{I}_{\mathbf{C}}, \mathrm{II}_{\mathbf{C}}) = \sum_{i=0}^{2}\sum_{j=0}^{2} \bar{c}_{ij}(\mathrm{I}_{\mathbf{C}} - 3)^i(\mathrm{II}_{\mathbf{C}} - 3)^j \quad \text{with} \quad \bar{c}_{00} = 0\,, \tag{41}$$

using a penalty function expressed as $\ln(\mathrm{III}_{\mathbf{C}}) = 0$ and the modified strain energy potential

$$\hat{\Psi} = \Psi^{(\mathrm{MR})} - (c_{10} + 2c_{01})\ln(\mathrm{III}_{\mathbf{C}}) + \frac{1}{2}\kappa(\ln(\mathrm{III}_{\mathbf{C}}))^2\,. \tag{42}$$

The 2nd Piola-Kirchhoff stress tensor is given by

$$\mathbf{S} = 2\frac{\partial\hat{\Psi}}{\partial\mathbf{C}} = 2\frac{\partial\Psi^{(\mathrm{MR})}}{\partial\mathbf{C}} + 2(-(c_{10} + 2c_{01}) + \kappa\ln(\mathrm{III}_{\mathbf{C}}))\mathbf{C}^{-1}\,, \tag{43}$$

where

$$2\frac{\partial}{\partial\mathbf{C}}\Psi^{(\mathrm{MR})} = 2\left(\frac{\partial\Psi^{(\mathrm{MR})}}{\partial\mathrm{I}_{\mathbf{C}}}\frac{\partial\mathrm{I}_{\mathbf{C}}}{\partial\mathbf{C}} + \frac{\partial\Psi^{(\mathrm{MR})}}{\partial\mathrm{II}_{\mathbf{C}}}\frac{\partial\mathrm{II}_{\mathbf{C}}}{\partial\mathbf{C}}\right) \tag{44}$$

$$= 2\left(\frac{\partial\Psi^{(\mathrm{MR})}}{\partial\mathrm{I}_{\mathbf{C}}} + \mathrm{I}_{\mathbf{C}}\frac{\partial\Psi^{(\mathrm{MR})}}{\partial\mathrm{II}_{\mathbf{C}}}\right)\mathbb{1} - 2\frac{\partial\Psi^{(\mathrm{MR})}}{\partial\mathrm{II}_{\mathbf{C}}}\mathbf{C}\,, \tag{45}$$

with

$$\frac{\partial\Psi^{(\mathrm{MR})}}{\partial\mathrm{I}_{\mathbf{C}}} = c_{10} + 2c_{20}(\mathrm{I}_{\mathbf{C}} - 3) + c_{11}(\mathrm{II}_{\mathbf{C}} - 3)\,, \tag{46}$$

$$\frac{\partial\Psi^{(\mathrm{MR})}}{\partial\mathrm{II}_{\mathbf{C}}} = c_{01} + 2c_{02}(\mathrm{II}_{\mathbf{C}} - 3) + c_{11}(\mathrm{I}_{\mathbf{C}} - 3)\,. \tag{47}$$

```
    <Type>MooneyRivlin</Type>
    <MooneyRivlin>
        <C10>FLOAT</C10> <!-- [Pa] -->
        <C01>FLOAT</C01> <!-- [Pa] -->
        <C20>FLOAT</C20> <!-- default 0 -->
        <C02>FLOAT</C02> <!-- default 0 -->
        <C11>FLOAT</C11> <!-- default 0 -->
        <B>FLOAT</B> <!-- bulk modulus [Pa] -->
    </MooneyRivlin>
```

Listing 17: .xml settings for general Rivlin material

Holzapfel The incompressible Holzapfel-Odgen strain energy density [3] can be written as a sum of isotropic and anisotropic components

$$\begin{aligned}\Psi^{(\mathrm{HO})}(\mathbf{C}) =& \frac{a}{2b}[\exp\big(b(\bar{\mathrm{I}}_{\mathbf{C}}-3)\big)-1] + \sum_{kl\in S}\frac{a_{kl}}{2b_{kl}}[\exp\big(b_{kl}(\mathrm{I}_{kl}-\delta_{kl})^2\big)-1]\\ &+\frac{\kappa}{4}(J^2-1-2\ln J)\,,\end{aligned}\tag{48}$$

where δ_{kl} denotes the Kronecker delta (zero unless $k = l$, in which case it is 1) and $S = \{ff, ss, fs\}$. The anisotropic invariants I_{ff} and I_{ss} do not support compression. Therefore, the associated strain energy density terms are zero if $\mathrm{I}_{ff}, \mathrm{I}_{ss} < 1$. The 2nd Piola-Kirchhoff stress tensor is given by

$$\begin{aligned}\mathbf{S} =& \frac{\kappa}{2}(J-\frac{1}{J})J\mathbf{C}^{-1} + \frac{a}{J^{2/3}}\exp\big(b(\bar{\mathrm{I}}_{\mathbf{C}}-3)\big)[\mathbb{1}-\frac{1}{3}\mathrm{I}_{\mathbf{C}}\mathbf{C}^{-1}]\\ &+\sum_{kl\in S}2a_{kl}(\mathrm{I}_{kl}-1)\exp\big(b_{kl}(\mathrm{I}_{kl}-\delta_{kl})^2\big)\mathrm{sym}(\mathbf{e}_k\otimes\mathbf{e}_l)\,.\end{aligned}\tag{49}$$

The terms with $kl \in S = \{ff, ss\}$ include an approximation to the Heavyside function $\chi(\mathrm{I}_{kl}) \approx \frac{1}{1+\exp\{-k(\mathrm{I}_{kl}-1)\}}$. With $k = 0$, we explicitly set $\chi(\mathrm{I}_{kl}) = 1$ if $\mathrm{I}_{kl} \le 1$, else $\chi(\mathrm{I}_{kl}) = 0$.

```
    <Type>Holzapfel</Type>
    <Holzapfel>
        <a>FLOAT</a> <!-- [Pa] -->
        <b>FLOAT</b> <!-- [Pa] -->
        <af>FLOAT</af>
        <bf>FLOAT</bf>
        <as>FLOAT</as>
        <bs>FLOAT</bs>
        <afs>FLOAT</afs>
        <bfs>FLOAT</bfs>
        <kappa>FLOAT</kappa> <!-- bulk modulus [Pa] -->
        <k>FLOAT</k> <!-- optional (default: 100) -->
    </Holzapfel>
```

Listing 18: .xml settings for Holzapfel material

Usyk The incompressible strain energy density by Usyk et al. [4] is given by

$$\begin{aligned}\Psi^{(\mathrm{U})}(\mathbf{C}) &= \frac{\kappa}{2}(\ln J)^2 + \frac{\mu}{2}(\exp(Q)-1)\,,\\ Q &= b_{\mathrm{ff}}E_{\mathrm{ff}}^2 + b_{\mathrm{ss}}E_{\mathrm{ss}}^2 + b_{\mathrm{nn}}E_{\mathrm{nn}}^2 + b_{\mathrm{fs}}(E_{\mathrm{fs}}^2+E_{\mathrm{sf}}^2) + b_{\mathrm{fn}}(E_{\mathrm{fn}}^2+E_{\mathrm{nf}}^2) + b_{\mathrm{ns}}(E_{\mathrm{ns}}^2+E_{\mathrm{sn}}^2)\,,\end{aligned}\tag{50}$$

where $E_{\text{ij}} = \mathbf{E}\mathbf{i}_0 \cdot \mathbf{j}_0$ for $i, j \in \{f, s, n\}$ are the entries of the Green-Lagrange strain tensor $\mathbf{E} = \frac{1}{2}(\mathbf{C} - \mathbb{1})$. The 2nd Piola-Kirchhoff stress is given by

$$\mathbf{S} = \mu \exp(Q)(b \odot E)_{ij} + \kappa \ln(J)\mathbf{C}^{-1} \quad \text{with} \quad \mathbf{b} = \begin{pmatrix} b_{\text{ff}} & b_{\text{fs}} & b_{\text{fn}} \\ b_{\text{sf}} & b_{\text{ss}} & b_{\text{sn}} \\ b_{\text{nf}} & b_{\text{ns}} & b_{\text{nn}} \end{pmatrix}, \tag{51}$$

where $(A \odot B)_{ij} = (A)_{ij}(B)_{ij}$ is the Hadamard or Schur product of the matrices $\mathbf{A}$ and $\mathbf{B}$. In general, the shear stress components are assumed to be of equal contribution

$$b_{ij} = b_{ji} \quad \text{for} \quad i \neq j\,. \tag{52}$$

```
    <Type>Usyk</Type>
    <Usyk>
        <a>FLOAT</a> <!-- shear modulus [Pa] -->
        <bff>FLOAT</bff>
        <bss>FLOAT</bss>
        <bnn>FLOAT</bnn>
        <bfs>FLOAT</bfs>
        <bfn>FLOAT</bfn>
        <bns>FLOAT</bns>
        <k>FLOAT</k> <!-- bulk modulus [Pa] -->
        <aScale>FLOAT</aScale> <!-- optional scaling for a (default 1) -->
        <bScale>FLOAT</bScale> <!-- optional scaling for b (default 1) -->
    </Usyk>
```

Listing 19: .xml settings for Usyk material

NeoHooke A nearly incompressible Neo-Hookean material is defined by the hyperelastic energy potential

$$\Psi^{(\text{NH})}(\mathbf{C}) = \frac{\mu}{2}(\operatorname{tr}\bar{\mathbf{C}} - 3) + \frac{\kappa}{2}(J-1)^2\,, \tag{53}$$

with the shear modulus μ, the bulk modulus κ, and the 2nd Piola-Kirchhoff stress

$$\begin{aligned} \mathbf{S} &= \mu \frac{\partial \operatorname{tr}\bar{\mathbf{C}}}{\partial \mathbf{C}} + \kappa \frac{\mathrm{d}(J-1)^2}{\mathrm{d}J}\frac{\partial J}{\partial \mathbf{C}} \\ &= \mu \frac{\partial(\mathrm{III}_{\mathbf{C}}^{-1/3}\mathbf{C} : \mathbb{1})}{\partial \mathbf{C}} + \kappa 2(J-1) \cdot \frac{1}{2} J\mathbf{C}^{-1} \\ &= \mu \left[\mathrm{III}_{\mathbf{C}}^{-1/3}\mathbb{1} - \frac{1}{3}\mathrm{III}_{\mathbf{C}}^{-1/3-1}\mathrm{III}_{\mathbf{C}}\mathbf{C}^{-1}(\mathbf{C} : \mathbb{1})\right] + \kappa(J-1)J\mathbf{C}^{-1} \\ &= \mu \mathrm{III}_{\mathbf{C}}^{-1/3}(\mathbb{1} - \frac{1}{3}\mathrm{I}_{\mathbf{C}}\mathbf{C}^{-1}) + \kappa(J-1)J\mathbf{C}^{-1}\,, \end{aligned} \tag{54}$$

where the relationship $\frac{\partial \mathrm{III}_{\mathbf{C}}}{\partial \mathbf{C}} = J^2\mathbf{C}^{-1}$ was used.

```
    <Type>NeoHooke</Type>
    <NeoHooke>
        <a>FLOAT</a> <!-- shear modulus [Pa] -->
        <k>FLOAT</k> <!-- bulk modulus [Pa] -->
    </NeoHooke>
```

Listing 20: .xml settings for NeoHooke material

5.2.2 Tension Models

File provides tensions from a tension.list file. The actual data is outsourced and managed by a FileManager object that lives within the solver class. The tension between stored data gets interpolated from a CBDataFromFile* object. You have to provide one file at location `Filename` with the following content:

```
# TIME   # PATH to file containing the tension values for each element at
    TIME
```

Listing 21: File structure of a tension.list file

The individual files at `PATH` containing the element tension have to be binary with the following structure

```
int32    NrElements
double   tension[0]
double   tension[1]
double   tension[...]
double   tension[n]
```

Listing 22: File structure of individual tension files

```
    <TensionMax> any FLOAT </TensionMax>
    <TensionModel>File</TensionModel>
    <File>
        <StartTime> 0.0 </StartTime>
        <Period> 1.0 </Period>
        <Filename> PATH </Filename>
    </File>
```

Listing 23: .xml settings for the File tension model

Lumens is a phenomenological model based directly on measurements of length-dependent tension in myocardial cells, as detailed by Lumens et al. [5]. This model takes into account the current length of the muscle cell, making it noteworthy for its potential to reproduce the Starling effect. The initial formulation of the model is activated by a prescribed function $F_{\text{rise}}(t)$, which computes the activation parameter C. This parameter is described as "physiologically related to intracellular calcium concentration." In contrast to the original paper, certain constants within the formula underwent conversion to ensure consistent units representing time and space, using meters (m) and seconds (s) instead of micrometers (µm) and milliseconds (ms), respectively. Additionally, for 3D simulations, C_{rest} was set to 0 to initiate the system in an equilibrium state with zero force. The equations are

$$\begin{cases} L_{\text{s}} = L_{\text{s,ref}} \exp(\epsilon_{\text{f}})\,, & \text{with} \quad \epsilon_{\text{f}} = 1 - \lambda_{\text{f}} \\ F_{\text{rise}} = 0.02x^3(8-x)^2\exp(-x)\,, & \text{with} \quad x = \min(8.0, \max(t/\tau_{\text{R}})) \\ \frac{\mathrm{d}L_{\text{sc}}}{\mathrm{d}t} = (\frac{L_{\text{s}}-L_{\text{sc}}}{L_{\text{se,iso}}} - 1)v_{\max}\,, & \\ C_{\text{L}} = \tanh\big(4 \cdot 1\text{e}12 \cdot (L_{\text{sc}} - L_{\text{sc},0})^2\big)\,, & \\ T_{\text{fun}} = \tau_{\text{sc}}(0.29 + 0.3 \cdot 1\text{e}6 \cdot L_{\text{sc}})\,, & \\ \frac{\mathrm{d}C}{\mathrm{d}t} = \frac{1}{\tau_{\text{R}}} C_{\text{L}} F_{\text{rise}} + \frac{1}{\tau_{\text{D}}}(C_{\text{rest}} - C)/(1+\exp\Big(\frac{T_{\text{fun}}-t}{\tau_{\text{D}}}\Big))\,, & \\ t = t - t_{\text{cycle}}\,, \quad \text{while} \quad t > t_{\text{cycle}} & \\ \sigma_{\text{fact}} = C(L_{\text{sc}} - L_{\text{sc},0}) \cdot 1\text{e}6 \cdot \frac{L_{\text{s}}-L_{\text{sc}}}{L_{\text{se,iso}}}\,, & \text{if} \quad L_{\text{sc}} \geq L_{\text{sc},0}\,. \end{cases} \tag{55}$$

The generated tension is given by $T_\mathrm{a}(t) = T_\mathrm{max}\sigma_\mathrm{fact}$.

```
    <TensionMax> any FLOAT </TensionMax>
    <TensionModel>Lumens</TensionModel>
    <Lumens>
        <startTime> 0 </startTime> <!-- [s] -->
        <cycleLength> 0.8 </cycleLength> <!-- [s] -->
        <Filename> Path </Filename>
        <ExportIndices> ElementIndex </ExportIndices> <!-- export element
   data to file -->
        <Crest> 0.0 </Crest>
        <Lsc0> 1.51 * 1e-6 </Lsc0> <!-- [m] -->
        <Lsref> 2.0 * 1e-6 </Lsref> <!-- [m] -->
        <Lseiso> 0.04 * 1e-6 </Lseiso> <!-- [m] -->
        <vmax> 7.0 * 1e-6 </vmax> <!-- [1/s] -->
        <tauD> 32.0 * 1e-3 </tauD> <!-- [s] -->
        <tauR> 48.0 * 1e-3 </tauR> <!-- [s] -->
        <tausc> 425.0 * 1e-3 </tausc> <!-- [s] -->
    </Lumens>
```

Listing 24: .xml settings for the Lumens tension model

FromFunction is a collection of four simple functions to calculate an active tension response. The possibilities are `Linear`, `Sinus`, `SinusDriver`, and `DoubleHill`. Only choose one of the function types per material.

- `Linear` is a function increasing linearly between `StartTime` and `StopTime` from 0 to 1.
$$f(t) = \frac{t - t_\mathrm{start}}{t_\mathrm{stop} - t_\mathrm{start}} . \tag{56}$$
The generated tension is given by $T_\mathrm{a}(t) = T_\mathrm{max} f(t)$. This function can be used for constant tension at T_max by setting `KeepMaxValue` to `true`.

- `Sinus` is a function defining a sinus arc by using a frequency ω and phase ϕ.
$$f(t) = 0.5 - 0.5\cos(\omega(t - t_\mathrm{start}) + \phi) . \tag{57}$$
The generated tension is given by $T_\mathrm{a}(t) = T_\mathrm{max} f(t)$.

- `SinusDriver` is a sinus function for easier definition of only the positive sinus arc from `StartTime` to `StopTime`.
$$f(t) = \sin\left(\frac{\pi(t - t_\mathrm{start})}{t_\mathrm{stop} - t_\mathrm{start}}\right) . \tag{58}$$
The generated tension is given by $T_\mathrm{a}(t) = T_\mathrm{max} f(t)$.

- `DoubleHill` is a function based on the time-dependent compliance of the left ventricle. Active tension T_a is assumed to be equal to the time course of chamber elastance $E(t)$. Stergiopulos et al. [6] suggested that the elastance of the left ventricle can be approximated by two Hill functions:

$$
\begin{aligned}
e(t) = \frac{E(t)-E_{\min}}{E_{\max}-E_{\min}} &= \frac{1}{k}\left(\frac{g_{\mathrm{c}}}{1+g_{\mathrm{c}}}\right)\left(\frac{1}{1+g_{\mathrm{r}}}\right), \\
\text{with} \quad g_{\mathrm{c}} = \left(\frac{t'}{\tau_{\mathrm{c}}}\right)^{m_{\mathrm{c}}}, &\quad g_{\mathrm{r}} = \left(\frac{t'}{\tau_{\mathrm{r}}}\right)^{m_{\mathrm{r}}}, \\
t' = \quad \operatorname{mod}\,(t-t_0, T), &\quad k = \max(e(t))\,.
\end{aligned}
\tag{59}
$$

Equation (59) is used as an active stress driver function $T_{\mathrm{a}}(t) = T_{\max} e(t)$. Table 2 shows sample parameters for atrial and ventricular tissue.

Table 2: Parameters for the active stress driver function DoubleHill.

Description	Ventricles	Atria
Contraction rate const. m_{c}	1.32	1.99
Relaxation rate const. m_{r}	14.5	11.2
Contraction time offset τ_{c}	0.215 s	0.042 s
Relaxation time offset τ_{r}	0.362 s	0.138 s
Onset time t_0	0.15 s	0.0 s
Period T	0.8 s	0.8 s
Peak tension $T_{\max}$	80 kPa	25 kPa

```xml
    <TensionMax> any FLOAT </TensionMax>
    <TensionModel>FromFunction</TensionModel>
    <FromFunction>
        <!-- general settings valid for all functions -->
        <StartTime>0.0</StartTime>
        <StopTime>30.0</StopTime>
        <KeepMaxValue> false </KeepMaxValue>
        <!-- function specific options -->
        <Type> DoubleHill </Type>
        <DoubleHill>
            <Period> 0.8 </Period>
            <ContrTimeOffset> 0.215 </ContrTimeOffset>
            <RelaxTimeOffset> 0.362 </RelaxTimeOffset>
            <ContrRateConst> 1.32 </ContrRateConst>
            <RelaxRateConst> 14.5 </RelaxRateConst>
            <OnsetTime> 0.15 </OnsetTime>
        </DoubleHill>

        <Type> Sinus </Type>
        <Sinus>
            <Omega> 0.0 </Omega>
            <Phi> 0.0 </Phi>
        </Sinus>
    </FromFunction>
```

Listing 25: .xml settings for the FromFunction tension models

Land17 implements the model by Land et al. [7] with the default parameters from the original publication. You can run this tension model in three different modes depending on your choice of the option `CalciumTransientType`:

1. with a predefined calcium transient for the ventricles (`Coppini`) or the atria (`Brixius`).
2. a calcium transient read from an `External` file (requires path to file in `CalciumFile`) sampled at 1 ms intervals.
3. coupled to an electrophysiology solver using the option `Elphy`. Currently this is only supported with *acCELLerate*, which can be used within *CardioMechanics* as a plugin (refer to acCELLerate).

By setting the option `rateDependancy` to `OFF`, stretch rate is set to $\frac{\mathrm{d}\lambda}{\mathrm{d}t} = 0$ at all times. Internally, the model requires the same units as in the publication. If you are unsure, refer to [7].

```xml
    <TensionMax>1000</TensionMax>
    <TensionModel>Land17</TensionModel>
    <Land17>
        <startTime> 0 </startTime> <!-- [s] -->
        <cycleLength> 1.0 </cycleLength> <!-- [s] -->
        <kff> 1.0 </kff>
        <kss> 0 </kss>
        <knn> 0 </knn>
        <ksn> 0 </ksn>
        <Filename> Path </Filename>
        <ExportIndices> ElementIndex </ExportIndices> <!-- export element data to file -->
        <rateDependancy> ON </rateDependancy> <!-- ON or OFF -->
        <CalciumTransientType> STR </CalciumTransientType> <!-- Coppini, Brixius, External (requires CalciumFile), Elphy (coupled to electrophysiology) -->
        <CalciumFile> NONE </CalciumFile>
        <TRPNn> 2.0 </TRPNn>
        <TRPNk> 0.1 </TRPNk>
        <Ca50> 0.805 </Ca50>
        <ku> 1.0 </ku>
        <nTM> 5.0 </nTM>
        <TRPN50> 0.35 </TRPN50>
        <kuw> 0.182 </kuw>
        <kws> 0.012 </kws>
        <rs> 0.25 </rs>
        <rw> 0.5 </rw>
        <gammas> 0.0085 </gammas>
        <gammaw> 0.615 </gammaw>
        <phi> 2.23 </phi>
        <Aeff> 25.0 </Aeff>
        <beta0> 2.3 </beta0>
        <beta1> -2.4 </beta1>
        <Tref> 120.0 </Tref> <!-- [kPa] -->
        <a> 2.1 </a>
        <k> 7.0 </k>
        <eta_l> 200.0 </eta_l>
        <eta_s> 20.0 </eta_s>
        <xi> 1.0 </xi>
    </Land17>
```

Listing 26: .xml settings for the Land17 tension model

TanH employs the phenomenological model by Niederer et al. [8] to describe the temporal development of force generation:

$$\begin{cases} T_\mathrm{a}(t,\lambda_\mathrm{f}) = T_\mathrm{peak}\phi(\lambda)\tanh^2(\frac{t_\mathrm{s}}{\tau_c})\tanh^2(\frac{t_\mathrm{dur}-t_\mathrm{s}}{\tau_r}) \qquad \text{for} \quad 0 < t_\mathrm{s} < t_\mathrm{dur}\,, \\ \phi(\lambda_\mathrm{f}) = \max\{\tanh(\mathrm{ld}(\lambda_\mathrm{f}-\lambda_0)),0\}\,, \\ \tau_c = \tau_{c0} + \mathrm{ld}_\mathrm{up}(1-\phi(\lambda_\mathrm{f}))\,, \\ t_\mathrm{s} = t - t_\mathrm{a} - t_\mathrm{emd}\,, \end{cases} \tag{60}$$

where λ_f is the fiber stretch ratio, t_a is the time of activation, T_peak is the peak tension, t_emd is the electromechanical delay, τ_c is the contraction time, τ_r is the relaxation time, ld is the degree of length dependence, ld_up is the length dependence upstroke time, τ_{c0} is the contraction duration, and λ_0 is the minimal fiber stretch.

```
    <TensionMax>1000</TensionMax>
    <TensionModel>TanH</TensionModel>
    <TanH>
        <startTime> 0 </startTime> <!-- [s] -->
        <cycleLength> 1.0 </cycleLength> <!-- [s] -->
        <kff> 1.0 </kff>
        <kss> 0 </kss>
        <knn> 0 </knn>
        <ksn> 0 </ksn>
        <Filename> Path </Filename>
        <ExportIndices> ElementIndex </ExportIndices> <!-- export element
data to file -->
        <MinFiberStretch> 0.7 </MinFiberStretch>
        <EMD> 0.015 </EMD> <!-- [s] -->
        <PeakTension> 120 </PeakTension> <!-- [kPa] -->
        <Duration> 0.3 </Duration> <!-- [s] -->
        <ContractionTime> 0.1 </ContractionTime> <!-- [s] -->
        <DegreeOfLengthDependence> 5.0 </DegreeOfLengthDependence>
        <LengthDependenceUpstrokeTime> 0.5 </LengthDependenceUpstrokeTime>
<!-- [s] -->
        <RelaxationTime> 0.1 </RelaxationTime> <!-- [s] -->
    </TanH>
```

Listing 27: .xml settings for the TanH tension model

Bestel employs the phenomenological model by Bestel et al. [9], characterized through a time-dependent stress function T_a:

$$\begin{cases} \frac{\mathrm{d}T_\mathrm{a}(t)}{\mathrm{d}t} = -|a(t)|T_\mathrm{a}(t) + \sigma_0|a(t)|_+\,, \\ |a(t)|_+ = \max(a(t),0)\,, \\ a(t) = \alpha_\mathrm{max}f(t) + \alpha_\mathrm{min}(1-f(t))\,, \\ f(t) = S^+(t-t_\mathrm{sys})S^-(t-t_\mathrm{dias})\,, \\ S^\pm(\Delta t) = 0.5(1 \pm \tanh(\frac{\Delta t}{\gamma}))\,, \end{cases} \tag{61}$$

where $a(t)$ is the activation function, σ_0 is the contractility, α is the activation rate, γ is the steepness, t_sys and t_dias are the systolic and diastolic onset times, respectively.

```
    <TensionMax>1</TensionMax>
    <TensionModel>Bestel</TensionModel>
    <Bestel>
```

```
        <startTime> 0 </startTime> <!-- [s] -->
        <cycleLength> 1.0 </cycleLength> <!-- [s] -->
        <kff> 1.0 </kff>
        <kss> 0 </kss>
        <knn> 0 </knn>
        <ksn> 0 </ksn>
        <Filename> Path </Filename>
        <ExportIndices> ElementIndex </ExportIndices> <!-- export element
    data to file -->
        <Contractility> 1e5 </Contractility> <!-- [Pa] -->
        <MaxActivationRate> 5 </MaxActivationRate>
        <MinActivationRate> -30 </MinActivationRate>
        <Steepness> 5e-3 </Steepness>
        <OnsetSystole> 0.17 </OnsetSystole> <!-- [s] -->
        <OnsetDiastole> 0.484 </OnsetDiastole> <!-- [s] -->
    </Bestel>
```

Listing 28: .xml settings for the Bestel tension model

5.3 Solver

Two different solver classes are available. These are the `Static` solver and the `NewmarkBeta` solver. The `Static` solver does not require any further parameters to be set and is used to solve problems according to Equation 18 without momentum or problems where inertial forces can be neglected. In this case, time steps can be interpreted as loading steps instead. Dynamic problems should be solved with the `NewmarkBeta` solver, which uses the name-giving time integration method to solve the initial value problem. It provides two parameters to control the numerical stability of the solution. First order accuracy and unconditional stability can be achieved with $\beta = \frac{1}{4}(\gamma + \frac{1}{2})^2$ and $\gamma \geq \frac{1}{2}$. Using $\gamma > 0.5$ enables numerical dissipation of higher frequencies. However, it degrades the accuracy of the solver slightly. Additionally, you can choose to use a `ConsistentMassMatrix` (recommended) or a lumped mass matrix by setting the parameter to `false`. With `NewmarkBeta.Type` you can choose between the mumps and superlu solver packages to perform the factorization.

Adaptive Time Stepping Since the cardiac cycle consists of different phases that are more or less difficult to solve, CardioMechanic's solver is equipped with an adaptive time stepping scheme. The idea behind the time stepping is to decrease the time step by a factor of two each time a simulation step fails, and to increase it by the same factor if it succeeds. Increase should happen in such a way that the same points in time are hit that would be hit by the larger time steps to ensure a consistent export of data. To prevent too many of the computationally expensive failing solver steps, the relaxation can be delayed by the following two parameters:

- `MinStep` After reduction, do at least this number of steps before trying a larger time step. Should be dividable by two.
- `FastRelaxation` Allows the time step to increase by more than one level at the same time. Tries always the maximum possible time step length after each export, no matter how small the time step was before.

The solver will adapt the time step size based on solver and solver plugin return codes:

- `CBStatus::SUCCESS`: system solved, the counter increments by one and gets subsequently checked if an increase of the step size is possible for the next solver run.
- `CBStatus::REPEAT`: system solved, but coupling condition is not fulfilled yet.
- `CBStatus::FAILED`: no solution was found with the current time step. Reduce time step by a factor of two.

In most cases, a `TimeStep` of 1 ms is sufficient to solve the mechanical system. If you do not set `MaxTimeStep`, `TimeStep` is used instead. Experience shows that if you observe more than two or three time step reductions during a simulation, it is highly likely that the simulation will fail sooner or later. Typically this happens with an insufficient quality of the tetrahedral mesh. One really bad element is all it takes to give you a lot of trouble.

```xml
<Solver>
    <!-- General Settings -->
    <Precision> 1e-8 </Precision>
    <Epsilon> 1e-12 </Epsilon>
    <IgnoreMaxIt> false </IgnoreMaxIt>
    <MaxIterations> 50 </MaxIterations>
    <MaxFunEval> 1000 </MaxFunEval>
    <NonZeros> 2000 </NonZeros>
    <ExportSNESMatrix> false </ExportSNESMatrix>
    <StepUpIterations> 1000 </StepUpIterations>
    <Timing> true </Timing>
    <Formulation> TotalLagrangian </Formulation>

    <!-- Set Solver -->
    <Type> Static/NewmarkBeta </Type>
    <NewmarkBeta>
        <!-- Dissipation of higher frequencies for Gamma >= 0.5 -->
        <!-- Unconditional stability for 2*Beta >= Gamma >= 0.5 -->
        <Beta> 0.25 </Beta>
        <Gamma> 0.5 </Gamma>
        <ConsistentMassMatrix> true </ConsistentMassMatrix>
        <Type> mumps/superlu </Type>
    </NewmarkBeta>

    <!-- Timing -->
    <StartTime> 0 </StartTime>
    <StopTime> FLOAT </StopTime>
    <TimeStep> FLOAT </TimeStep>
    <MinTimeStep> 1e-9 </MinTimeStep>
    <MaxTimeStep> TimeStep </MaxTimeStep>
    <MaxInitTimeStep> MaxTimeStep </MaxInitTimeStep>
    <FastRelaxation> false </FastRelaxation>
    <MinSteps> 4 </MinSteps>
</Solver>
```

Listing 29: .xml settings for the solver class

5.4 Solver Plugins

The solver class supports the use of plugins which have to be activated in the `Solver` scope of the settings file. The individual plugin parameters are set in a separate `Plugins` scope, which are explained in further detail in the following sections. If you want to implement a new plugin, you should make sure and think about, which template functions defined in CBSolverPlugin.h you need to implement. The following is a list of the most important ones:

- `Init()`: called during solver initialization to initialize vectors, matrices, parameters etc.
- `Prepare()`: called before first `SolverStep()` to have access to the undeformed mesh.
- `Apply()`: called during `SolverStep()` before system is solved.
- `StepBack()`: controls plugin behavior when a solver step failed and is repeated with a smaller time step.
- `Export()`: primary export to VTK file called during solver export.
- `WriteToFile()`: secondary export to arbitrary files.
- `GetName()`: return the name of the plugin.
- `ApplyToNodalForces()`: to calculate and add external forces to the system, e.g. pressure.
- `AnalyzeResults()`: called during `SolverStep()` after system is solved. Can be used to check coupling conditions.
- `GetStatus()`: returns status code of the plugin at the end of `SolverStep()`.

```xml
<Solver>
.
.
.
    <Plugins>
        <Circulation>true/false</Circulation>
        <ContactHandling>true/false</ContactHandling>
        <LoadUnloadedState>true/false</LoadUnloadedState>
        <acCELLerate>true/false</acCELLerate>
        <ApplyPressure>true/false</ApplyPressure>
        <ApplyPressureFromFunction>true/false</ApplyPressureFromFunction>
        <ApplyPressureFromFunctionNodeExport>true/false</
    ApplyPressureFromFunctionNodeExport>
        <ReferenceRecovery>true/false</ReferenceRecovery>
        <RobinBoundary>true/false</RobinBoundary>
        <RobinBoundaryGeneral>true/false</RobinBoundaryGeneral>
    </Plugins>
</Solver>

<Plugins>
...
</Plugins>
```

Listing 30: .xml settings for solver plugins

5.4.1 LoadUnloadedState

The purpose of this plugin is to start a simulation using a stress and pressure free state of a geometry created from imaging. A .node file for the unloaded and inflated state is required as input for this plugin. They are typically acquired as a result of using the ReferenceRecovery plugin.

Algorithm 1 Algorithm of LoadUnloadedState plugin.

```
1: procedure LoadUnloadedState
2:     initialize $originalCoords\_ = nodes\_$; $unloadedCoords\_$; $inflatedCoords\_$
3:     set $nodes\_ = unloadedCoords\_$
4:     update shape functions
5:     set $velocity\_ = 0$; $acceleration\_ = 0$
6:     set $nodes\_ = inflatedCoords\_$
7:     solve system once during Solver::Init()
8: end procedure
```

```
<LoadUnloadedState>
    <unloadedNodes>PATH</unloadedNodes>
    <inflatedNodes>PATH</inflatedNodes>
</LoadUnloadedState>
```

Listing 31: .xml settings for LoadUnloadedState plugin

5.4.2 ContactHandling

This plugin models the effect of the tissue surrounding the heart according to Fritz et al. [10], which assumes a frictionless and permanent contact between the epicardium and the pericardium using a penalty formulation. For this purpose, the so called master (Γ^{M}) and slave (Γ^{S}) surfaces are defined as a contact interface. Γ^{M} is defined on the epicardium and Γ^{S} is the inner surface of a separate volume mesh surrounding the heart. The force acting on the master surface is given by $\mathbf{f}^{\mathrm{M}} = \alpha g(\mathbf{x})\mathbf{n}$ with the penalty parameter α, the gap function $g(\mathbf{x})$, and the current surface normal vector $\mathbf{n}$. For the slave surface, the same formulation is used except with negative sign for the contact forces $\mathbf{f}^{\mathrm{S}} = -\mathbf{f}^{\mathrm{M}}$. The gap function $g(\mathbf{x})$ is determined by projecting a point $\mathbf{x}^{\mathrm{m}} \in \Gamma^{\mathrm{M}}$ onto the surface Γ^{S} into the direction of the current normal direction $\mathbf{n}$. Depending on the distance between $\mathbf{x}^{\mathrm{m}}$ and the projected point $\mathbf{x}^{\mathrm{s}} \in \Gamma^{\mathrm{S}}$, the gap function reads

$$g(\mathbf{x}) = \begin{cases} \|\mathbf{x}^{\mathrm{m}} - \mathbf{x}^{\mathrm{s}}\|^2/2d & \text{for} \quad \|\mathbf{x}^{\mathrm{m}} - \mathbf{x}^{\mathrm{s}}\| < d\,, \\ \|\mathbf{x}^{\mathrm{m}} - \mathbf{x}^{\mathrm{s}}\| - \frac{d}{2} & \text{for} \quad \|\mathbf{x}^{\mathrm{m}} - \mathbf{x}^{\mathrm{s}}\| \geq d\,, \\ 0 & \text{for} \quad \|\mathbf{x}^{\mathrm{m}} - \mathbf{x}^{\mathrm{s}}\| > d_{\max}\,, \end{cases} \tag{62}$$

with the `TransitionDistance` d and the `MaxDistance` $d_{\max}$ below which contact is maintained. This adaptation to the original formulation was made to yield a smooth transition in case the surfaces Γ^{M} and Γ^{S} start to overlap, which results in a change of direction of **n**. Γ^{M} and Γ^{S} have to be defined under `Mesh.Surfaces` with the respective surface type of either `CONTACT_MASTER` or `CONTACT_SLAVE`. With the parameter `Alpha` you can set the penalty variable in Pa. The contact problem penalization can be ramped up during initialization to avoid large forces in the first time step. The ramp is either linear or exponential depending on the parameter `InitType` and will be carried out in n discrete `Steps`. `Beta` sets a multiplication factor for α during this phase. The option `MaxAngle` sets the maximal angle (in degree) between master and slave normals that still gets considered. The option `OneWayForce` determines if the contact force is applied in both directions (0) or if only positive (1) or negative (-1) forces are applied. If you set `SurfaceNormalDirection` as unidirectional, the forces are applied in the direction of the master surface normal. Otherwise, master surface normal and slave surface normal are added and normalized. The surface normal direction can be flipped during run time using `FlipSurfaceNormals`. If you find your slave surface is too far away from the heart's surface, you can use the option `useContactHandlingToFitPeri` to run a short simulation to slowly bring the surface normal closer to the heart. In such a simulation, it is recommended to fix the epicardial nodes using Dirichlet boundary conditions to avoid unwanted movement. If you want to apply a spatially varying penalty formulation, assign the third attribute to the master surface elements in the Surface Elements File (*.sur). With `Export` active, contact force, contact pressure, contact distance, and a binary state describing if a contact slave element was found are saved in arrays of the VTK Unstructured Grid Files (*.vtu) in each time step. Additionally, a text file with average contact distance and contact pressure is saved at `ExportPath`.

Remark: if you intent to use a combination of pericardial boundary conditions, only one plugin will export the data properly since they overwrite each others arrays. Hence, turn export only on for one of them!

```
<ContactHandling>
    <Export>true/false</Export>
    <ExportPath>PATH</ExportPath>
    <MaxDistance>FLOAT</MaxDistance>
    <TransitionDistance>FLOAT</TransitionDistance>
    <MaxAngle>FLOAT</MaxAngle>
    <Alpha>FLOAT</Alpha>
    <Beta>FLOAT</Beta>
    <StartTime>FLOAT</StartTime>
    <OneWayForce>0/1/-1</OneWayForce>
    <InitType>Linear/Exponential</InitType>
    <Steps>INT</Steps>
    <useContactHandlingToFitPeri>true/false</useContactHandlingToFitPeri>
    <FlipSlaveNormals>true/false</FlipSlaveNormals>
    <SurfaceNormalDirection>unidirectional/bidirectional</
    SurfaceNormalDirection>
</ContactHandling>
```

Listing 32: .xml settings for ContactHandling plugin

5.4.3 Circulation

The `Circulation` plugin implements various lumped parameter circulatory system models suitable for 1-,2-, and 4-chamber heart models. Figure 2 shows the basic class structure of this plugin. Some general settings are available for the base plugin. Chamber volumes and pressures in each time step can be exported to `IterationsFile` in CSV format. The parameter `Tolerance` decides how well 3D and 0D volumes have to match to be accepted by the solver. Surfaces to be excluded from applying pressure to can be added to `IgnoredSurfaceIndices` as a comma separated list. `PressureUnit` and `VolumeUnit` define the units used for the input parameters and model outputs. `MaxIterations` is the maximum number of iterations before the plugin return `CBStatus::FAILURE` to the solver. `Perturbation` defines, whether initial pressure perturbations shall be performed to determine the Jacobian at the beginning of each new time step. `SecantIterations` defines the number of iterations to be executed with the one-dimensional secant-method before perturbing the pressures (typically set to 0 in favor of the quasi Newton method described below).

```xml
<Circulation>
    <IgnoredSurfaceIndices> ... </IgnoredSurfaceIndices>

    <Tolerance> 1e-7 </Tolerance>
    <MaxIterations> 250 </MaxIterations>
    <MaxNonNewtonIterations> 10 </MaxNonNewtonIterations>
    <Perturbation> false </Perturbation>
    <SecantIterations> 0 </SecantIterations>
    <IterationsFile> PATH </IterationsFile>

    <PressureUnit> mmHg/hPa/Pa </PressureUnit>
    <VolumeUnit> ml/m3 </VolumeUnit>
    ...
</Circulation>
```

Listing 33: .xml settings for the circulation plugin

Coupling method The pressure values p_{C} with $\mathrm{C} \in \{\mathrm{LV}, \mathrm{RV}, \mathrm{LA}, \mathrm{RA}\}$ are used as a Neumann boundary condition of the form

$$\mathbf{F}\,\mathbf{S}\mathbf{N} = -p_{\mathrm{C}}(t) J \mathbf{F}^{-\mathsf{T}} \mathbf{N} \tag{63}$$

and are computed iteratively at each time step. In the first iteration $i = 0$ of each time step t_n of the mechanical system, the pressure p_{C} is extrapolated from the previous 5 time steps using a 4th order Adams-Bashforth scheme:

$$p_{\mathrm{C}}^{n,0} = p_{\mathrm{C}}^{n-1} + \Delta t_n \left(\frac{55}{24} \frac{\Delta p_{\mathrm{C}}^{n-1}}{\Delta t_{n-1}} - \frac{59}{24} \frac{\Delta p_{\mathrm{C}}^{n-2}}{\Delta t_{n-2}} + \frac{37}{24} \frac{\Delta p_{\mathrm{C}}^{n-3}}{\Delta t_{n-3}} - \frac{9}{24} \frac{\Delta p_{\mathrm{C}}^{n-4}}{\Delta t_{n-4}} \right), \tag{64}$$

with $\frac{\Delta p_{\mathrm{C}}^k}{\Delta t_k} = \frac{p_{\mathrm{C}}^k - p_{\mathrm{C}}^{k-1}}{t_k - t_{k-1}}$, $k \in \mathbb{N}$. If there are not enough pressure values available for extrapolation (i.e. $n < 5$), the pressure is simply increased $p_{\mathrm{C}}^{n,0} = p_{\mathrm{C}}^{n-1} + 1\,\mathrm{Pa}$. For the sake of brevity, the four chamber pressures are now denoted $\mathbf{p}_{\mathrm{C}} = (p_{\mathrm{LV}}, p_{\mathrm{RV}}, p_{\mathrm{LA}}, p_{\mathrm{RA}})^{\mathsf{T}}$. During consequent iterations $0 < i \leq 10$, the residual $\mathbf{r}_{\mathrm{C}} \colon \mathbb{R}^4 \to \mathbb{R}^4$ for all chambers

$$\mathbf{r}_{\mathrm{C}}(\mathbf{p}_{\mathrm{C}}^{n,i}) = (r_{\mathrm{LV}}, r_{\mathrm{RV}}, r_{\mathrm{LA}}, r_{\mathrm{RA}})^{\mathsf{T}},$$

with $r_{\mathrm{C}} = \left| V_{\mathrm{C}}^{3\mathrm{D}} - V_{\mathrm{C}}^{0\mathrm{D}} \right|$ is used to update $\mathbf{p}_{\mathrm{C}}$ by a quasi-Newton method:

$$\mathbf{p}_{\mathrm{C}}^{n,i} = \mathbf{p}_{\mathrm{C}}^{n,i-1} - \mathbf{C}_i^{-1} \mathbf{r}_{\mathrm{C}}(\mathbf{p}_{\mathrm{C}}^{n,i}), \tag{65}$$

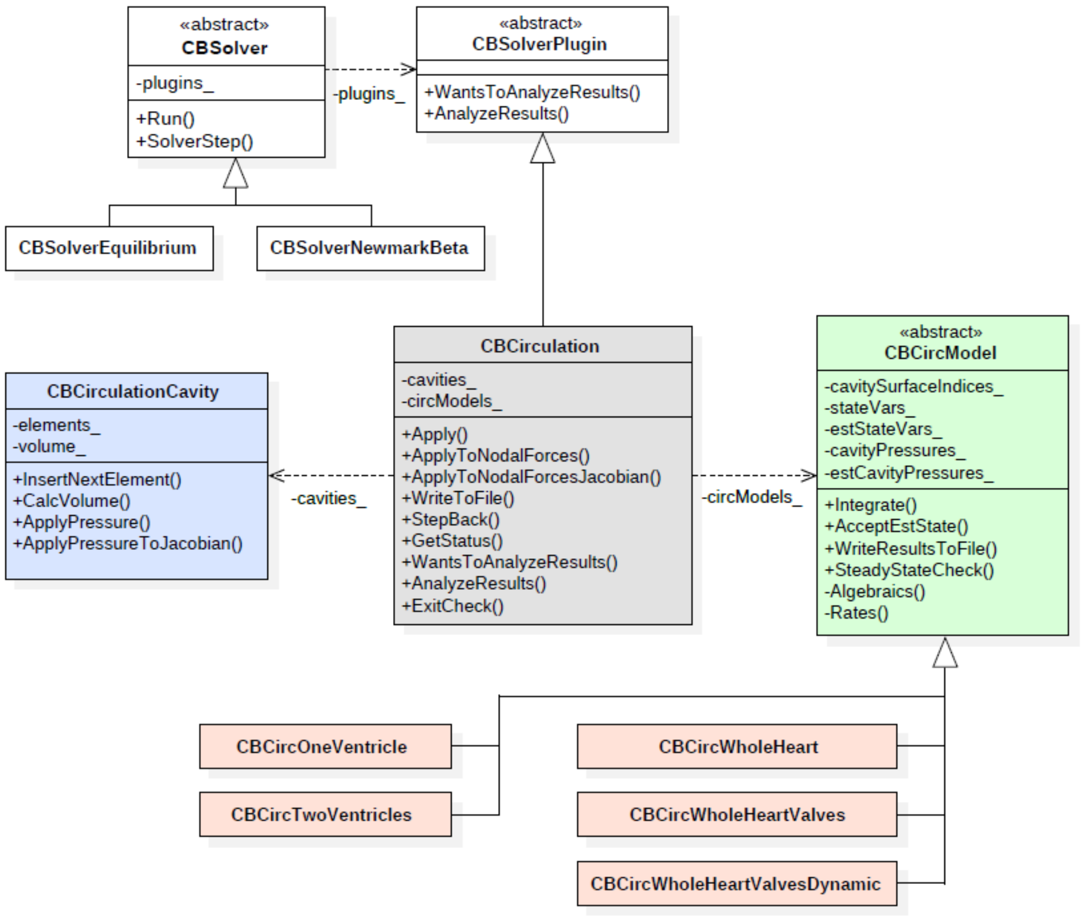

Figure 2: Simplified diagram of the circulation plugin class structure showing only the most important functions and variables. Created by Steffen Schuler.

where $\mathbf{C}_i$ is the compliance matrix determined by

$$\mathbf{C}_i^{-1} = \mathbf{C}_{i-1}^{-1} + \left(\triangle\mathbf{p}_\mathrm{C}^{n,i} - \mathbf{C}_{i-1}^{-1}\triangle\mathbf{r}_\mathrm{C}^{n,i}\right)\frac{\triangle(\mathbf{p}_\mathrm{C}^{n,i})^\mathsf{T}\mathbf{C}_{i-1}^{-1}}{\triangle(\mathbf{p}_\mathrm{C}^{n,i})^\mathsf{T}\mathbf{C}_{i-1}^{-1}\triangle\mathbf{r}_\mathrm{C}^{n,i}}\,, \qquad \mathbf{C}_0^{-1} = \mathbb{1}\,,$$

with $\triangle\mathbf{p}_\mathrm{C}^{n,i} = \mathbf{p}_\mathrm{C}^{n,i} - \mathbf{p}_\mathrm{C}^{n,i-1}$, $\triangle\mathbf{r}_\mathrm{C}^{n,i} = \mathbf{r}_\mathrm{C}(\mathbf{p}_\mathrm{C}^{n,i}) - \mathbf{r}_\mathrm{C}(\mathbf{p}_\mathrm{C}^{n,i-1})$. Typically, this method satisfies the volume-consistency constraint

$$\begin{cases} V_\mathrm{LV}^\mathrm{3D}(\mathbf{d}(t),\mathbf{p}_\mathrm{C}(t)) = V_\mathrm{LV}^\mathrm{0D}(\mathbf{c}(t),\mathbf{p}(t)) & \text{in} \quad (0,T]\,, \\ V_\mathrm{RV}^\mathrm{3D}(\mathbf{d}(t),\mathbf{p}_\mathrm{C}(t)) = V_\mathrm{RV}^\mathrm{0D}(\mathbf{c}(t),\mathbf{p}(t)) & \text{in} \quad (0,T]\,, \\ V_\mathrm{LA}^\mathrm{3D}(\mathbf{d}(t),\mathbf{p}_\mathrm{C}(t)) = V_\mathrm{LA}^\mathrm{0D}(\mathbf{c}(t),\mathbf{p}(t)) & \text{in} \quad (0,T]\,, \\ V_\mathrm{RA}^\mathrm{3D}(\mathbf{d}(t),\mathbf{p}_\mathrm{C}(t)) = V_\mathrm{RA}^\mathrm{0D}(\mathbf{c}(t),\mathbf{p}(t)) & \text{in} \quad (0,T]\,, \end{cases} \tag{66}$$

with a tolerance of $\|\mathbf{r}_\mathrm{C}\|_\infty < \varepsilon \leq 1\mathrm{e}{-7}\,\mathrm{mL}$ in 2 to 3 iterations. During the first time steps, the quasi-Newton update might not be sufficient in which case a standard Newton method is applied and the compliance matrix is calculated using the Jacobian $\mathbf{C} = \frac{\mathrm{d}\mathbf{r}_\mathrm{C}}{\mathrm{d}\mathbf{p}_\mathrm{C}}$ after successively perturbing one chamber at a time.

Coupling Damping When the valve model is included in the circulation model, the pressures start to oscillate with a frequency corresponding to the simulation time step. These oscillations do not decay on their own and can be explained by the effects of inertia on both sides to be coupled. This can be avoided by using the `CouplingDamping` option. First, the coupling algorithm iterates, until the coupling criterion is fulfilled, giving the pressure estimate $\mathbf{p}^n$. Then a pressure $\mathbf{p}_\mathrm{damped}^n$ between this pressure and the pressure $\mathbf{p}^{n-1}$ from the previous time step is applied in one more iteration:

$$\mathbf{p}_\mathrm{damped}^n = (1-d)\mathbf{p}^n + d\mathbf{p}^{n-1}\,, \tag{67}$$

with the damping factor

$$d = b^n d_\mathrm{init} \quad \text{with} \quad 0.5 \leq b < 1 \tag{68}$$

being the `DeclineFactor` and d_init is the `InitialFactor`. As soon as $d < 10^{-5}$, the coupling damping is automatically turned off. If you don't want this to happen, choose $b = 1$.

```
    <CouplingDamping>
        <InitialFactor> 0.5 </InitialFactor>
        <DeclineFactor> 0.9 </DeclineFactor>
    </CouplingDamping>
```

Listing 34: .xml settings for the coupling damping of the circulation plugin (subkeys of <Circulation>)

Preloading Alternatively to the ReferenceRecovery plugin, you can choose to preload the model using the `Circulation` plugin. This has the disadvantage of being less accurate and it has to be repeated with every single simulation, since no save state is produced. Nevertheless, this is how it works: (1) Gradually apply the negative diastolic pressures, which causes the chambers to shrink (set by `PreloadingTime1`). (2) Set the tension to zero again by resetting the node shape functions. Set the velocity and acceleration of nodes to zero. (3) Gradually apply the positive diastolic pressures, which inflates the chambers to approximately their original size (set by `PreloadingTime2`).

```
    <PreloadingTime1> 0.0 </PreloadingTime1>
    <PreloadingTime2> 0.0 </PreloadingTime2>
    ...
    <Circs>
        <Circ_X>
            ...
            <Cavities>
                ...
                <!-- CircOneVentricle -->
                <VentrPreloading> 0.0 </VentrPreloading>
                <!-- CircTwoVentricles -->
                <LvPreloading> 0.0 </LvPreloading>
                <RvPreloading> 0.0 </RvPreloading>
                <!-- 4-chamber models -->
                <RvPreloading>0.0</RvPreloading>
                <LvPreloading>0.0</LvPreloading>
                <RaPreloading>0.0</RaPreloading>
                <LaPreloading>0.0</LaPreloading>
            </Cavities>
        </Circ_X>
    </Circs>
```

Listing 35: .xml settings for the preloading procedure of the circulation plugin (subkeys of <Circulation>)

Limit Cycle Detection A meaningful comparison of model outputs necessitates the system to be in a limit cycle. In this context, reaching a limit cycle implies a consistent blood volume distribution that remains unchanged across cardiac cycles. Consequently, pressures and flows occurring at corresponding time points of successive cycles are constant. Given the challenge in accurately estimating the simulation time required to attain a limit cycle, a limit cycle check has been incorporated. This check automatically concludes the simulation once a predefined limit cycle criterion is satisfied.

Over an extended period of time, it is imperative that both ventricles eject an equivalent volume of blood within a closed-loop circulatory system. Consequently, the parity between left ventricular (LV) and right ventricular (RV) stroke volumes serves as a robust limit cycle criterion. To ascertain the accurate disparity in stroke volumes (SVD) amid non-ideally closing valves, an additional ODE is incorporated into the circulatory model and temporally integrated:

$$\frac{\mathrm{dSVD}}{\mathrm{d}t} = Q_{\mathrm{SysArt}} - Q_{\mathrm{PulArt}} \,. \tag{69}$$

Starting from a designated temporal offset, the absolute value of the stroke volume difference undergoes periodic assessments (occurring at integer multiples of the cycle length) against a pre-established threshold. After each assessment, the stroke volume difference is reset to zero. This manipulation of the stroke volume difference serves as the default limit cycle criterion. Alternatively, any other output variable within a circulation model may be selected as the limit cycle criterion. The designated parameter is subsequently checked for changes in its values between two consecutive cycles, thereby assessing its temporal difference. This makes it usable for the `CircOneVentricle` model as well.

```
    <!-- default limit cycle detection -->
    <SteadyStateCheck>
        <Active> true </Active>
        <Mode> StrokeVolumeDifference </Mode>
        <StartTime> 0.0 </StartTime>
        <Period> 0.8 </Period>
        <Threshold> 1.0 </Threshold>
    </SteadyStateCheck>

    <!-- alternative limit cycle detection -->
    <SteadyStateCheck>
        <Active> true </Active>
        <Mode> TemporalDifference </Mode>
        <Parameter> VentrVolume </Parameter>
        <StartTime> 0.8 </StartTime>
        <Period> 0.8 </Period>
        <Threshold> 1.0 </Threshold>
    </SteadyStateCheck>
```

Listing 36: .xml settings for a limit cycle check for all circulation models (subkey of <Circulation><Circs><Circ_X>)

Cavity Volume and Closed Surface Check The circulation plugin regularly requires the calculation of chamber volumes in the finite element model. The chamber volumes are calculated by summing the signed volumes of all tetrahedrons formed by the vertices $\mathbf{a}_k$, $\mathbf{b}_k$, and $\mathbf{c}_k$ of the surface triangle k and the point of origin:

$$V = \sum_k \frac{1}{6}(\mathbf{a}_k - \mathbf{0}) \cdot ((\mathbf{b}_k - \mathbf{a}_k) \times (\mathbf{c}_k - \mathbf{a}_k))\,. \tag{70}$$

Equation 70 is only valid for closed surfaces. If the surface is not closed, the calculated volume generally depends on the reference point that is used to span the tetrahedrons. Therefore, each surface is checked during initialization with respect to two reference points $\mathbf{r}_1$ and $\mathbf{r}_2$ using the following criterion:

$$.\Delta V = \Big| \sum_k \frac{1}{6}(\mathbf{a}_k - \mathbf{r}_1) \cdot ((\mathbf{b}_k - \mathbf{a}_k) \times (\mathbf{c}_k - \mathbf{a}_k)) \tag{71}$$

$$- \sum_k \frac{1}{6}(\mathbf{a}_k - \mathbf{r}_2) \cdot ((\mathbf{b}_k - \mathbf{a}_k) \times (\mathbf{c}_k - \mathbf{a}_k))\Big| \overset{!}{<} 10^{-10}\mathrm{mL} \tag{72}$$

CircWholeHeart/CircWholeHeartValves/CircWholeHeartValvesDynamic The 4-chamber model uses a circulatory system model first published in [11]. The systemic and pulmonary circulation are each represented by a three-element Windkessel model. With `CircWholeHeartValvesDynamic`, a description of the pressure gradient across the atrioventricular and semilunar valves is included based on Garcia et al. [12] with dynamic opening and closing of the valves [13]. Valve dynamics are replaced by an "open on pressure, close on flow" valve in `CircWholeHeartValves`. In `CircWholeHeart`, valves are represented as diodes with a characteristic resistance. For the sake of simplicity, only the equations for `CircWholeHeartValvesDynamic` are presented here.

Briefly, the model is given by

$$\frac{\mathrm{d}\mathbf{c}(t)}{\mathrm{d}t} - \mathbf{G}_{\mathbf{c}}(t, \mathbf{c}(t), \mathbf{p}(t)) = \mathbf{0} \quad \text{in} \quad (0, T]\,, \tag{73}$$

where $\mathbf{c}\colon (0,T] \to \mathbb{R}^{16}$ are the state variables

$$\begin{aligned}\mathbf{c} =&(V_{\mathrm{LV}}, V_{\mathrm{SysArt}}, V_{\mathrm{SysVen}}, V_{\mathrm{RA}}, V_{\mathrm{RV}}, V_{\mathrm{PulArt}}, V_{\mathrm{PulVen}}, V_{\mathrm{LA}}, Q_{\mathrm{AV}}, Q_{\mathrm{TV}}, Q_{\mathrm{PV}}, Q_{\mathrm{MV}},\\ &\eta_{\mathrm{AV}}, \eta_{\mathrm{TV}}, \eta_{\mathrm{PV}}, \eta_{\mathrm{MV}})^{\mathsf{T}}\,,\end{aligned}$$

associated with the chamber and vessel volumes V, the blood flow Q through the atrioventricular and semilunar valves, and the current state of the valves η. $\mathbf{G_c}$ conveniently collects the r.h.s. of the ODEs and $\mathbf{p}\colon (0,T] \to \mathbb{R}^8$ contains the pressure values in all compartments of the circulatory system

$$\mathbf{p} = (p_{\mathrm{LV}}, p_{\mathrm{SysArt}}, p_{\mathrm{SysVen}}, p_{\mathrm{RA}}, p_{\mathrm{RV}}, p_{\mathrm{PulArt}}, p_{\mathrm{PulVen}}, p_{\mathrm{LA}})^{\mathsf{T}}\,.$$

The system of ODEs is given by

$$\begin{cases} \frac{\mathrm{d}V_{\mathrm{LV}}}{\mathrm{d}t} = Q_{\mathrm{MV}} - Q_{\mathrm{AV}}, & \frac{\mathrm{d}V_{\mathrm{SysArt}}}{\mathrm{d}t} = Q_{\mathrm{AV}} - Q_{\mathrm{SysPer}}\,,\\ \frac{\mathrm{d}V_{\mathrm{SysVen}}}{\mathrm{d}t} = Q_{\mathrm{SysPer}} - Q_{\mathrm{SysVen}}, & \frac{\mathrm{d}V_{\mathrm{RA}}}{\mathrm{d}t} = Q_{\mathrm{SysVen}} - Q_{\mathrm{TV}}\,,\\ \frac{\mathrm{d}V_{\mathrm{RV}}}{\mathrm{d}t} = Q_{\mathrm{TV}} - Q_{\mathrm{PV}}, & \frac{\mathrm{d}V_{\mathrm{PulArt}}}{\mathrm{d}t} = Q_{\mathrm{PV}} - Q_{\mathrm{PulPer}}\,,\\ \frac{\mathrm{d}V_{\mathrm{PulVen}}}{\mathrm{d}t} = Q_{\mathrm{PulPer}} - Q_{\mathrm{PulVen}}, & \frac{\mathrm{d}V_{\mathrm{LA}}}{\mathrm{d}t} = Q_{\mathrm{PulVen}} - Q_{\mathrm{MV}}\,,\\ L\frac{\mathrm{d}Q_{\mathrm{VT}}}{\mathrm{d}t} = \Delta p_{\mathrm{net}} - B|Q_{\mathrm{VT}}|Q_{\mathrm{VT}}\,, & \\ \frac{\mathrm{d}\eta_{\mathrm{VT}}}{\mathrm{d}t} = \begin{cases} K_o(1-\eta_{\mathrm{VT}})\Delta p_{\mathrm{net}} & \text{if} \quad \Delta p_{\mathrm{net}} > 0\,,\\ K_c\eta_{\mathrm{VT}}\Delta p_{\mathrm{net}} & \text{if} \quad \Delta p_{\mathrm{net}} \le 0\,, \end{cases} & \end{cases} \tag{74}$$

where $B = \frac{\rho_{\mathrm{Blood}}}{2A_{\mathrm{Eff}}^2}$ is the Borda-Carnot resistance, $L = \frac{6.28\rho_{\mathrm{Blood}}}{\sqrt{A_{\mathrm{Eff}}}}$ is the inertance, ρ_{Blood} is the density of blood, and Δp_{net} is the difference in pressure across the atrioventricular and semilunar valves $\mathrm{VT} \in \{\mathrm{AV}, \mathrm{TV}, \mathrm{PV}, \mathrm{MV}\}$. The rate of opening and closing of the valves is given by the coefficients K_o and K_c. The effective valve area A_{Eff} depends on the state of the valve and is given by

$$A_{\mathrm{Eff}}(t) = A_{\mathrm{Ref}}\frac{s(t)}{1-s(t)} \quad \text{using} \quad s(t) = (M_{\max} - M_{\min})\eta_{\mathrm{VT}}(t) + M_{\min}\,, \tag{75}$$

with the reference area A_{Ref} of the valve and the minimum and maximum area ratio, $M_{\min}$ and $M_{\max}$, respectively. For the pulmonary and systemic circulation, the blood flow is given by

$$Q_{\mathrm{SysPer}} = \frac{\frac{V_{\mathrm{SysArt}}}{C_{\mathrm{SysArt}}} - p_{\mathrm{SysVen}}}{R_{\mathrm{SysPer}}}\,, \qquad Q_{\mathrm{SysVen}} = \frac{p_{\mathrm{SysVen}} - p_{\mathrm{RA}}}{R_{\mathrm{SysVen}}}\,, \tag{76}$$

$$Q_{\mathrm{PulPer}} = \frac{\frac{V_{\mathrm{PulArt}}}{C_{\mathrm{PulArt}}} - p_{\mathrm{PulVen}}}{R_{\mathrm{PulPer}}}\,, \qquad Q_{\mathrm{PulVen}} = \frac{p_{\mathrm{PulVen}} - p_{\mathrm{LA}}}{R_{\mathrm{PulVen}}}\,, \tag{77}$$

with the resistances R, compliances C, and the pressures defined by

$$p_{\mathrm{SysArt}} = \frac{V_{\mathrm{SysArt}} - V^0_{\mathrm{SysArt}}}{C_{\mathrm{SysArt}}} + Q_{\mathrm{SysArt}}R_{\mathrm{SysArt}}\,, \qquad p_{\mathrm{SysVen}} = \frac{V_{\mathrm{SysVen}} - V^0_{\mathrm{SysVen}}}{C_{\mathrm{SysVen}}}\,, \tag{78}$$

$$p_{\mathrm{PulArt}} = \frac{V_{\mathrm{PulArt}} - V^0_{\mathrm{PulArt}}}{C_{\mathrm{PulArt}}} + Q_{\mathrm{PulArt}}R_{\mathrm{PulArt}}\,, \qquad p_{\mathrm{PulVen}} = \frac{V_{\mathrm{PulVen}} - V^0_{\mathrm{PulVen}}}{C_{\mathrm{PulVen}}}\,. \tag{79}$$

Unstressed systemic and peripheral volumes are denoted by a superscript 0.

```xml
<Circulation>
...
    <Circs>
        <Circ_1>
            <Active>true</Active>
            <Type>CircWholeHeartValvesDynamic</Type>
            <ExportFile> PATH </ExportFile>
            <MaxIntegrationTimeStep>1e-4</MaxIntegrationTimeStep>

            <CircParameters>
                <!-- CircWholeHeart -->
                <SysArtValveResist>   0.006</SysArtValveResist>
                <SysArtResist>        0.05 </SysArtResist>
                <SysArtCompli>        2.5  </SysArtCompli>
                <SysArtVolumeUnstr> 800.0  </SysArtVolumeUnstr>
                <SysPerResist>        0.60 </SysPerResist>
                <SysVenResist>        0.03 </SysVenResist>
                <SysVenCompli>      100.0  </SysVenCompli>
                <SysVenVolumeUnstr>2850.0  </SysVenVolumeUnstr>
                <RavValveResist>      0.003</RavValveResist>
                <PulArtValveResist>   0.003</PulArtValveResist>
                <PulArtResist>        0.02 </PulArtResist>
                <PulArtCompli>       10.0  </PulArtCompli>
                <PulArtVolumeUnstr> 150.0  </PulArtVolumeUnstr>
                <PulPerResist>        0.07 </PulPerResist>
                <PulVenResist>        0.03 </PulVenResist>
                <PulVenCompli>       15.0  </PulVenCompli>
                <PulVenVolumeUnstr> 200.0  </PulVenVolumeUnstr>
                <LavValveResist>      0.003</LavValveResist>
                <!-- CircWholeHeartValves -->
                <BloodDensity>7.95e-4</BloodDensity>
                <SysArtValveMax>         0.95 </SysArtValveMax>
                <SysArtValveMin>         0.001</SysArtValveMin>
                <SysArtValveAreaRef>     7.0  </SysArtValveAreaRef>
                <RavValveMax>            0.7  </RavValveMax>
                <RavValveMin>            0.001</RavValveMin>
                <RavValveAreaRef>       15.0  </RavValveAreaRef>
                <PulArtValveMax>         0.95 </PulArtValveMax>
                <PulArtValveMin>         0.001</PulArtValveMin>
                <PulArtValveAreaRef>     7.0  </PulArtValveAreaRef>
                <LavValveMax>            0.7  </LavValveMax>
                <LavValveMin>            0.001</LavValveMin>
                <LavValveAreaRef>       15.0  </LavValveAreaRef>
                <!-- CircWholeHeartValvesDynamic -->
                <SysArtValveRateOpening>10.0  </SysArtValveRateOpening>
                <SysArtValveRateClosing> 6.0  </SysArtValveRateClosing>
                <RavValveRateOpening>   20.0  </RavValveRateOpening>
                <RavValveRateClosing>    6.0  </RavValveRateClosing>
                <PulArtValveRateOpening>10.0  </PulArtValveRateOpening>
                <PulArtValveRateClosing> 6.0  </PulArtValveRateClosing>
                <LavValveRateOpening>   20.0  </LavValveRateOpening>
                <LavValveRateClosing>    6.0  </LavValveRateClosing>
            </CircParameters>
            <InitialConditions>
                <TotalVolume>    5500.0  </TotalVolume>
                <SysArtVolume>    985.0  </SysArtVolume>
                <PulArtVolume>    301.0  </PulArtVolume>
                <PulVenVolume>    310.0  </PulVenVolume>
                <SysArtFlow>        0.0  </SysArtFlow>
                <RavFlow>          50.0  </RavFlow>
                <PulArtFlow>        0.0  </PulArtFlow>
                <LavFlow>         100.0  </LavFlow>
                <SysArtValveState>  0.0  </SysArtValveState>
                <RavValveState>     0.1  </RavValveState>
                <PulArtValveState>  0.0  </PulArtValveState>
                <LavValveState>     0.1  </LavValveState>
                <RvPressure>        3.75 </RvPressure>
                <LvPressure>        7.50 </LvPressure>
                <RaPressure>        4.50 </RaPressure>
                <LaPressure>        8.25 </LaPressure>
            </InitialConditions>
            <Cavities>
                <!-- define these surfaces as CAVITY in Mesh.Surfaces -->
                <RvSurface>130</RvSurface>
                <LvSurface>131</LvSurface>
                <RaSurface>132</RaSurface>
                <LaSurface>133</LaSurface>
            </Cavities>
        </Circ_1>
    </Circs>
</Circulation>
```

Listing 37: .xml settings for the 4-chamber circulation model

CircOneVentricle was designed for simulations with only the left ventricle. The pressure within the left ventricle is determined by a simplified closed-loop lumped parameter model of the circulatory system of the same form as Eq. (73), with the state variables $\mathbf{c}\colon (0,T] \to \mathbb{R}^3$ that are associated with the chamber and vessel volumes

$$\mathbf{c} = (V_{\text{LV}}, V_{\text{Art}}, V_{\text{Ven}})^{\mathsf{T}}\,, \tag{80}$$

and the vector $\mathbf{p}\colon (0,T] \to \mathbb{R}^3$ collects the pressure values in all compartments of the system

$$\mathbf{p} = (p_{\text{LV}}, p_{\text{Art}}, p_{\text{Ven}})^{\mathsf{T}}\,. \tag{81}$$

The system of ODEs is given by

$$\frac{\mathrm{d}V_{\text{LV}}}{\mathrm{d}t} = Q_{\text{Ven}} - Q_{\text{Art}}, \quad \frac{\mathrm{d}V_{\text{Art}}}{\mathrm{d}t} = Q_{\text{Art}} - Q_{\text{Per}}, \quad \frac{\mathrm{d}V_{\text{Ven}}}{\mathrm{d}t} = Q_{\text{Per}} - Q_{\text{Ven}}\,, \tag{82}$$

with the blood flow

$$\begin{cases} Q_{\text{Art}} = \max\left\{\frac{p_{\text{LV}}-p_{\text{CArt}}}{R_{\text{AV}}+R_{\text{Art}}}, 0\right\} & Q_{\text{Per}} = \frac{p_{\text{CArt}}-p_{\text{Ven}}}{R_{\text{Per}}}\,, \\ Q_{\text{Ven}} = \max\left\{\frac{p_{\text{Ven}}-p_{\text{LV}}}{R_{\text{Ven}}}, 0\right\}, & \end{cases} \tag{83}$$

and the pressure

$$p_{\text{CArt}} = \frac{V_{\text{Art}} - V^0_{\text{Art}}}{C_{\text{Art}}}, \quad p_{\text{Art}} = p_{\text{CArt}} - Q_{\text{Art}} R_{\text{Art}}, \quad p_{\text{Ven}} = \frac{V_{\text{Ven}} - V^0_{\text{Ven}}}{C_{\text{Ven}}}\,, \tag{84}$$

in the system.

```
<Circulation>
...
    <Circs>
        <Circ_1>
            <Active>true</Active>
            <Type>CircOneVentricle</Type>
            <ExportFile> PATH </ExportFile>
            <MaxIntegrationTimeStep>1e-4</MaxIntegrationTimeStep>

            <CircParameters>
                <ArtValveResist>   0.006</ArtValveResist>
                <ArtResist>        0.07 </ArtResist>
                <ArtCompli>        2.0  </ArtCompli>
                <ArtVolumeUnstr> 800.0  </ArtVolumeUnstr>
                <PerResist>        0.9  </PerResist>
                <VenResist>        0.03 </VenResist>
                <VenCompli>      100.0  </VenCompli>
                <VenVolumeUnstr>2850.0  </VenVolumeUnstr>
            </CircParameters>
            <InitialConditions>
                <TotalVolume>   5500.0 </TotalVolume>
                <ArtVolume>      985.0 </ArtVolume>
                <VentrPressure>   7.50 </VentrPressure>
            </InitialConditions>
            <Cavities>
                <!-- define these surfaces as CAVITY in Mesh.Surfaces -->
                <VentrSurface> 130 </VentrSurface>
            </Cavities>
        </Circ_1>
    </Circs>
</Circulation>
```

Listing 38: .xml settings for the 1-chamber circulation model

CircTwoVentricles combines both ventricles in a closed circulation model by using the structure of the CircOneVentricle model twice. The state variables $\mathbf{c}\colon (0,T] \to \mathbb{R}^6$ that are associated with the chamber and vessel volumes

$$\mathbf{c} = \left(V_{\text{LV}}, V_{\text{SysArt}}, V_{\text{SysVen}}, V_{\text{RV}}, V_{\text{PulArt}}, V_{\text{PulVen}}\right)^{\mathsf{T}}, \tag{85}$$

and the vector $\mathbf{p}\colon (0,T] \to \mathbb{R}^6$ collects the pressure values in all compartments of the system

$$\mathbf{p} = \left(p_{\text{LV}}, p_{\text{SysArt}}, p_{\text{SysVen}}, p_{\text{RV}}, p_{\text{PulArt}}, p_{\text{PulVen}}\right)^{\mathsf{T}}. \tag{86}$$

The system of ODEs is given by

$$\begin{cases} \frac{\mathrm{d}V_{\text{LV}}}{\mathrm{d}t} = Q_{\text{PulVen}} - Q_{\text{SysArt}}, & \frac{\mathrm{d}V_{\text{RV}}}{\mathrm{d}t} = Q_{\text{SysVen}} - Q_{\text{PulArt}}, \\ \frac{\mathrm{d}V_{\text{SysArt}}}{\mathrm{d}t} = Q_{\text{SysArt}} - Q_{\text{SysPer}}, & \frac{\mathrm{d}V_{\text{PulArt}}}{\mathrm{d}t} = Q_{\text{PulArt}} - Q_{\text{PulPer}}, \\ \frac{\mathrm{d}V_{\text{SysVen}}}{\mathrm{d}t} = Q_{\text{SysPer}} - Q_{\text{SysVen}}, & \frac{\mathrm{d}V_{\text{PulVen}}}{\mathrm{d}t} = Q_{\text{PulPer}} - Q_{\text{PulVen}}, \end{cases} \tag{87}$$

with the blood flow

$$\begin{cases} Q_{\text{SysArt}} = \max\left\{\frac{p_{\text{LV}} - p_{\text{CSysArt}}}{R_{\text{AV}} - R_{\text{SysArt}}}, 0\right\}, & Q_{\text{PulArt}} = \max\left\{\frac{p_{\text{RV}} - p_{\text{CPulArt}}}{R_{\text{PV}} - R_{\text{PulArt}}}, 0\right\}, \\ Q_{\text{SysVen}} = \max\left\{\frac{p_{\text{SysVen}} - p_{\text{RV}}}{R_{\text{SysVen}}}, 0\right\}, & Q_{\text{PulVen}} = \max\left\{\frac{p_{\text{PulVen}} - p_{\text{LV}}}{R_{\text{PulVen}}}, 0\right\}, \\ Q_{\text{SysPer}} = \frac{p_{\text{CSysArt}} - p_{\text{SysVen}}}{R_{\text{SysPer}}}, & Q_{\text{PulPer}} = \frac{p_{\text{CPulArt}} - p_{\text{PulVen}}}{R_{\text{PulPer}}}, \end{cases} \tag{88}$$

and the pressure

$$
\begin{cases}
p_{\text{CSysArt}} = \frac{V_{\text{SysArt}} - V^0_{\text{SysArt}}}{C_{\text{SysArt}}}, & p_{\text{CPulArt}} = \frac{V_{\text{PulArt}} - V^0_{\text{PulArt}}}{C_{\text{PulArt}}} \, , \\
p_{\text{SysVen}} = \frac{V_{\text{SysVen}} - V^0_{\text{SysVen}}}{C_{\text{SysVen}}}, & p_{\text{PulVen}} = \frac{V_{\text{PulVen}} - V^0_{\text{PulVen}}}{C_{\text{PulVen}}} \, , \\
p_{\text{SysArt}} = p_{\text{CSysArt}} + Q_{\text{SysArt}} R_{\text{SysArt}}, & p_{\text{PulArt}} = p_{\text{CPulArt}} + Q_{\text{PulArt}} R_{\text{PulArt}} \, ,
\end{cases}
\tag{89}
$$

in the system.

```xml
<Circulation>
...
    <Circs>
        <Circ_1>
            <Active>true</Active>
            <Type>CircTwoVentricles</Type>
            <ExportFile> PATH </ExportFile>
            <MaxIntegrationTimeStep>1e-4</MaxIntegrationTimeStep>

            <CircParameters>
                <SysArtValveResist>   0.006</SysArtValveResist>
                <SysArtResist>        0.07 </SysArtResist>
                <SysArtCompli>        2.0  </SysArtCompli>
                <SysArtVolumeUnstr> 800.0  </SysArtVolumeUnstr>
                <SysPerResist>        0.9  </SysPerResist>
                <SysVenResist>        0.03 </SysVenResist>
                <SysVenCompli>      100.0  </SysVenCompli>
                <SysVenVolumeUnstr>2850.0  </SysVenVolumeUnstr>

                <PulArtValveResist>   0.003</PulArtValveResist>
                <PulArtResist>        0.02 </PulArtResist>
                <PulArtCompli>       10.0  </PulArtCompli>
                <PulArtVolumeUnstr> 150.0  </PulArtVolumeUnstr>
                <PulPerResist>        0.07 </PulPerResist>
                <PulVenResist>        0.03 </PulVenResist>
                <PulVenCompli>       15.0  </PulVenCompli>
                <PulVenVolumeUnstr> 200.0  </PulVenVolumeUnstr>
            </CircParameters>
            <InitialConditions>
                <TotalVolume>   5500.0 </TotalVolume>
                <SysArtVolume>   985.0 </SysArtVolume>
                <PulArtVolume>   300.0 </PulArtVolume>
                <PulVenVolume>   300.0 </PulVenVolume>
                <LvPressure>       7.50 </LvPressure>
                <RvPressure>       3.50 </RvPressure>
            </InitialConditions>
            <Cavities>
                <!-- define these surfaces as CAVITY in Mesh.Surfaces -->
                <LvSurface> 130 </LvSurface>
                <RvSurface> 131 </RvSurface>
            </Cavities>
        </Circ_1>
    </Circs>
</Circulation>
```

Listing 39: .xml settings for the 2-chamber circulation model

5.4.4 ApplyPressure

This plugin does exactly what its name suggests. It applies a pressure as a Neumann boundary condition of the form

$$\mathbf{F}\,\mathbf{S}\mathbf{N} = -p_{\mathrm{C}}(t) J \mathbf{F}^{-\mathsf{T}} \mathbf{N} \tag{90}$$

to a surface Γ. The pressure is calculated with

$$p(t) = p_{\max} \frac{t}{t_{\mathrm{stop}} - t_{\mathrm{start}}} \quad \text{if} \quad t < t_{\mathrm{stop}} \,. \tag{91}$$

If `KeepMaxPressure` is set to true, $p(t) = p_{\max} \,\forall\, t$. Pressure and volume data is exported to a file called `ApplyPressure.dat` or `Filename`.

```xml
    <ApplyPressure>
        <StartTime> 0.0 </StartTime> <!-- [s] -->
        <StopTime> any FLOAT </StopTime> <!-- [s] -->
        <SurfaceIndex> index </SurfaceIndex>
        <MaxPressure> any FLOAT </MaxPressure> <!-- [Pa] -->
        <KeepMaxPressure> true </KeepMaxPressure>
        <Filename> STR </Filename>
    </ApplyPressure>
```

Listing 40: .xml settings for the ApplyPressure plugin

5.4.5 ApplyPressureFromFunction

This plugin is a more sophisticated version of ApplyPressure. It adds the possibilities to use different functions for the pressure calculation and the definition of multiple groups and intervals. Pressure is calculated using

$$p(t) = p_{\mathrm{Offset}} + p_{\mathrm{Amp}} \cdot f(t) \,, \tag{92}$$

where the function $f(t)$ can be defined as one of the functions given in paragraph 5.2.2 by specifying one of the functions using the key `Type` in each interval. The default function is `Linear`. If the chosen function requires additional parameters, supply them with the appropriate keys. Setting the option `Invert` to true, will give the pressure

$$p(t) = p_{\mathrm{Offset}} + p_{\mathrm{Amp}} \cdot (1 - f(t)) \,, \tag{93}$$

instead. Each group you define can have multiple intervals, but the intervals of each group can be different. If you define multiple intervals in a group, the `StartTime` of interval $I + 1$ has to be bigger or equal to the `StopTime` of interval I. With the option `RelaxElementsAtStart` you can recalculate the shape function derivatives and set velocity and acceleration to zero at the start of each interval. This will only affect the material given in `MaterialsToRelax`. The exported file contains pressure and volume data of each specified surface in mmHg and mL, respectively.

```xml
<ApplyPressureFromFunction>
    <ExportFile> STR </ExportFile> <!-- default: ApplyPressureFromFunction.
   dat -->

```

```xml
    <Groups>
        <Group_X>
            <Surfaces> INT, INT, ... </Surfaces>
            <Intervals>
                <Interval_1>
                    <Type>Linear</Type>
                    <StartTime> FLOAT </StartTime> <!-- [s] -->
                    <StopTime> FLOAT </StopTime> <!-- [s] -->
                    <Offset> FLOAT </Offset> <!-- [Pa] -->
                    <Amplitude> FLOAT </Amplitude> <!-- [Pa] -->
                    <Invert> false </Invert>
                    <RelaxElementsAtStart> false </RelaxElementsAtStart>
                    <MaterialsToRelax> INT, INT, ... </MaterialsToRelax>
                </Interval_1>
                <Interval_2>
                    <Type>Sinus</Type>
                    <StartTime> FLOAT </StartTime> <!-- [s] -->
                    <StopTime> FLOAT </StopTime> <!-- [s] -->
                    <Offset> FLOAT </Offset> <!-- [Pa] -->
                    <Amplitude> FLOAT </Amplitude> <!-- [Pa] -->
                    <Invert> false </Invert>
                    <RelaxElementsAtStart> false </RelaxElementsAtStart>
                    <MaterialsToRelax> INT, INT, ... </MaterialsToRelax>

                    <Sinus>
                        <Omega> FLOAT </Omega>
                        <Phi> FLOAT </Phi>
                    </Sinus>
                </Interval_2>
            </Intervals>
        </Group_X>
        <Group_Y>
            <Surfaces> ... </Surfaces>
            <Intervals>
                <Interval_1>
                    ...
                </Interval_1>
            </Intervals>
        </Group_Y>
    </Groups>
</ApplyPressureFromFunction>
```

Listing 41: .xml settings for the ApplyPressureFromFunction plugin

5.4.6 ApplyPressureFromFunctionNodeExport

Conceptually the same plugin as ApplyPressureFromFunction but with added functionality. You can either choose to terminate the simulation at a certain time (`NodeExportTime`) or when a designated closed surface (`StopSurface`) reaches the desired volume (`StopVolume`).

```xml
<ApplyPressureFromFunctionNodeExport>
    <ExportFile> STR </ExportFile> <!-- default:
    ApplyPressureFromFunctionNodeExport.dat -->
    <StopSurface> INT </StopSurface> <!-- default: -1 -->
    <StopVolume> FLOAT </StopVolume> <!-- [mL] -->
    <NodeExportTime> FLOAT </NodeExportTime> <!-- [s] -->
    <NodeExportFile> STR </NodeExportFile>

    <Groups>
        ...
    </Groups>
</ApplyPressureFromFunctionNodeExport>
```

Listing 42: .xml settings for the ApplyPressureFromFunctionNodeExport plugin

5.4.7 RobinBoundary

This plugin adds a boundary condition in the form of a traction $\mathbf{t}$ acting on the surface Γ

$$\mathbf{t} = \mathbf{N}(\alpha\mathbf{d} \cdot \mathbf{N} + \beta\dot{\mathbf{d}} \cdot \mathbf{N}) \, . \tag{94}$$

This kind of boundary condition is typically used on the epicardium as a more simple model compared to the ContactHandling plugin. It essentially models the pericardium as a linear spring (with stiffness α) in parallel with a dashpot (with viscosity β). A more thorough explanation can be found in [14]. Compared to the ContactHandling plugin, these boundary conditions do not require the user to set up a dedicated mesh for the pericardium, thus making it more suitable for (bi-)ventricular simulations. By default, the spring stiffness and dashpot viscosity are constant on the entire surface. However, regional changes can be introduced by adding an element wise surface traction scaling to the Surface Elements File (*.sur). Export of this plugin adds the arrays `ContactForce`, `ContactPressure`, and `ContactDistance` to the VTK Unstructured Grid Files (*.vtu). The surface Γ defined by `SurfaceIndex` needs to be declared as type `CONTACT_ROBIN` using the key `Mesh.Surfaces.Surface_INT` (refer to Mesh).

Remark: if you intent to use a combination of pericardial boundary conditions, only one plugin will export the data properly since they overwrite each others arrays. Hence, turn export only on for one of them!

```
<RobinBoundary>
    <Export> false </Export>
    <StartTime> FLOAT </StartTime> <!-- [s] -->
    <FlipSurfaceNormals> false </FlipSurfaceNormals>
    <SurfaceIndex> INT </SurfaceIndex>
    <alpha> FLOAT </alpha> <!-- [Pa/m] -->
    <beta> FLOAT </beta> <!-- [(Pa*s) / m] -->
</RobinBoundary>
```

Listing 43: .xml settings for the RobinBoundary plugin

5.4.8 RobinBoundaryGeneral

This plugin adds a boundary condition in the form of a traction $\mathbf{t}$ acting on the surface Γ

$$\mathbf{t} = \alpha\mathbf{d} + \beta\dot{\mathbf{d}} \, , \tag{95}$$

i.e. it behaves like omnidirectional spring-dashpots. Otherwise, it offers the same functionality as the RobinBoundary plugin. Useful for truncated vessels, e.g. aorta and pulmonary artery.

Remark: if you intent to use a combination of pericardial boundary conditions, only one plugin will export the data properly since they overwrite each others arrays. Hence, turn export only on for one of them!

```
<RobinBoundaryGeneral>
    <Export> false </Export>
    <StartTime> FLOAT </StartTime> <!-- [s] -->
    <FlipSurfaceNormals> false </FlipSurfaceNormals>
    <SurfaceIndex> INT </SurfaceIndex>
    <alpha> FLOAT </alpha> <!-- [Pa/m] -->
    <beta> FLOAT </beta> <!-- [(Pa*s) / m] -->
</RobinBoundaryGeneral>
```

Listing 44: .xml settings for the RobinBoundaryGeneral plugin

5.4.9 ReferenceRecovery

A pressure and stress-free reference configuration of the heart has to be estimated to accurately model the biomechanical diastolic function. Methods applied in the context of cardiac mechanics try to find a stress-free reference configuration iteratively by using fixed-point iterations, as originally proposed by Sellier et al. [15]. To increase the convergence rate and accelerate the fixed-point method by Sellier et al., the following augmentation approach proposed by Rausch et al. [16] is used: For each iteration k, solve the forward problem $\phi(\mathbf{X}^k, p^{\mathrm{ED}})$

$$\mathbf{f}^{\mathrm{int}}(\mathbf{d}, T_{\mathrm{a}} = 0) - \mathbf{f}^{\mathrm{ext}}(\mathbf{d}, p^{\mathrm{ED}}) = \mathbf{0} \quad \text{in} \quad \Omega_k\,, \tag{96}$$

by applying an end-diastolic pressure p^{ED} to an intermediate configuration of the heart with coordinates $\mathbf{X}^k$ with no active stress (technically it is possible to add active stress, we just never did it). The resulting node coordinates $\mathbf{x}^k$ can be used to calculate the nodal error $\mathbf{R}^k = \mathbf{x}^k - \mathbf{x}^{\mathrm{dat}}$ w. r. t. the target coordinates $\mathbf{x}^{\mathrm{dat}}$. Finally, the reference coordinates are updated $\mathbf{X}^{k+1} = \mathbf{X}^k - \beta\mathbf{R}^k$ with the augmentation parameter β as shown in Algorithm 2. After solving the forward problem $\phi(\mathbf{X}, p^{\mathrm{ED}})$ once again with the end-diastolic pressure values p^{ED}, the target configuration is recovered and the solution $\mathbf{d}$ together with the pressure can be used as initial conditions for further simulations:

$$\mathbf{d}^0 = \mathbf{d}, \quad p^0 = p^{\mathrm{ED}}\,.$$

This method is not restricted to whole heart geometries and is applicable to (bi-)ventricular simulations as well. Sometimes the algorithm is more stable and leads to better results, if a pressure ramp is used, i.e. we divide the end-diastolic pressure into even increments and find an intermediate stress free reference configuration before we move to the next increment. Using the option `InflationDuration` combined with the solver's `TimeStep` option controls the amount of loading steps taken to from $p = 0\,\mathrm{Pa}$ to $\tilde{p}$. This plugin exports four distinct files. The first file contains pressure and volume data for all surfaces. The second file contains a summary of the following data: Cycle, Increment, Time, ResidualNorm, ResidualNormL2 and UnloadedVolume. Furthermore, a Node File (*.node) and a Fiber Orientation File (*.bases) of the stress free reference state after each iteration of the algorithm is saved.

```
<ReferenceRecovery>
    <ExportDir> STR </ExportDir>
    <Surfaces> INT, INT, ... </Surfaces>
    <Pressures> FLOAT, FLOAT, ...</Pressures> <!-- [Pa] -->
    <NumberOfIncrements> INT </NumberOfIncrements>
    <InflationDuration> FLOAT </InflationDuration>
```

Algorithm 2 Reference configuration recovery

```
1: procedure ReferenceRecovery($\Omega_0, p^{\mathrm{ED}}$)
2:     initialize $\mathbf{X}^1 = \mathbf{x}^{\mathrm{dat}}$; $k = 0$; $\beta = displacementFactor\_$; $n^p_{incr} = numIncrements\_$
3:     for $j \leq n^p_{incr}$ do
4:         $\tilde{p} = \frac{p^{\mathrm{ED}}}{n^p_{incr}} \cdot j$
5:         while $\left\|\mathbf{R}^k\right\| \geq tolerance\_$ or $k < cycleMax\_$ do
6:             update counter, $k = k + 1$
7:             solve forward problem, $\mathbf{x}^k = \phi(\mathbf{X}^k, \tilde{p})$
8:             calculate residual vector, $\mathbf{R}^k = \mathbf{x}^k - \mathbf{x}^{\mathrm{dat}}$
9:             if $k > 1$ then
10:                update augmentation parameter, $\beta = -\beta \frac{\mathbf{R}^{k-1} : [\mathbf{R}^k - \mathbf{R}^{k-1}]}{[\mathbf{R}^k - \mathbf{R}^{k-1}] : [\mathbf{R}^k - \mathbf{R}^{k-1}]}$
11:            end if
12:            update reference vector, $\mathbf{X}^{k+1} = \mathbf{X}^k - \beta \mathbf{R}^k$
13:            update $\frac{\partial N}{\partial X^{k+1}}$
14:        end while
15:        $k = 0$
16:        $j + +$
17:    end for
18:    return stress-free reference configuration, $\mathbf{X} = \mathbf{X}^k$
19: end procedure
```

```
    <DisplacementFactor> FLOAT </DisplacementFactor>
    <Precision> FLOAT </Precision>
    <CycleMax> INT </CycleMax>
    <Augmentation> true </Augmentation>
</ReferenceRecovery>
```

Listing 45: .xml settings for the ReferenceRecovery plugin

5.4.10 acCELLerate

This plugin makes electromechanically coupled simulations possible by solving both systems using a staggered algorithm. The class structure of this plugin and all relevant functions inside *acCELLerate* itself are shown in Figure 3. First, we shortly recapitulate how *acCELLerate* solves the monodomain equation. Using finite elements for space discretization, the monodomain equation in its semidiscretized form is given by

$$\mathbf{K}\mathbf{u}(t) = \beta C_{\mathrm{m}} \mathbf{M} \dot{\mathbf{u}}(t) + \beta \mathbf{I}_{\mathrm{ion}}(\mathbf{u}(t)) + \beta \mathbf{I}_{\mathrm{app}}(t)\,, \tag{97}$$

with

$$
\begin{aligned}
(\mathbf{K})_{ij} &= \int_{\Omega_0} (\mathbf{D}\boldsymbol{\nabla}\phi_j) \cdot \boldsymbol{\nabla}\phi_i \, \mathrm{d}\Omega_0 \ , \\
(\mathbf{M})_{ij} &= \int_{\Omega_0} \phi_j \phi_i \, \mathrm{d}\Omega_0 \ , \\
(\mathbf{I}_{\text{ion}}(\mathbf{u}(t))_i &= \int_{\Omega_0} I_{\text{ion}}(v(t))\phi_i \, \mathrm{d}\Omega_0 \ , \\
(\mathbf{I}_{\text{app}}(t))_i &= \int_{\Omega_0} I_{\text{app}}(t)\phi_i \, \mathrm{d}\Omega_0 \ ,
\end{aligned}
$$

where $\mathbf{u}$ is the transmembrane potential, β is the cell surface to volume ratio, and C_m is the membrane capacitance. To solve Equation 97 in time, *acCELLerate* uses first order Godunov splitting to decouple the parabolic diffusion problem from the ODEs of the reaction system. First, the ODEs of the reaction system are solved using explicit time integration methods such that

$$
\tilde{\mathbf{u}}^{n+1} = \mathbf{u}^n + \frac{\Delta t}{C_\text{m}} \left[\mathbf{I}_{\text{ion}}(\mathbf{u}^n) + \mathbf{I}_{\text{app}}(n)\right] \ . \tag{98}
$$

Next, we use the intermediate solution $\tilde{\mathbf{u}}^{n+1}$ to integrate the diffusion part in time by means of the Crank-Nicolson method:

$$
\left(\frac{\beta C_\text{m}}{\Delta t}\mathbf{M} + \theta \mathbf{K}\right) \mathbf{u}^{n+1} = \left(\frac{\beta C_\text{m}}{\Delta t}\mathbf{M} - (1-\theta)\mathbf{K}\right) \tilde{\mathbf{u}}^{n+1} \ , \tag{99}
$$

using $\theta = 0.5$. The linear system (99) is then solved using the generalized minimum residual method (GMRES) with a Jacobi preconditioning to obtain $\mathbf{u}^{n+1}$.

In the case of an electromechanically coupled simulation, the stiffness matrix $\mathbf{K}$ is rebuilt after every successful mechanical time step to incorporate the deformation:

$$
(\mathbf{K})_{ij} = \int_{\Omega_0} (J\mathbf{F}^{-1}\mathbf{D}\mathbf{F}^{-\mathsf{T}}\boldsymbol{\nabla}\phi_j) \cdot \boldsymbol{\nabla}\phi_i \, \mathrm{d}\Omega_0 \ , \tag{100}
$$

using the diffusion tensor

$$
\mathbf{D} = \sigma_f \frac{\mathbf{F}\mathbf{f}_0 \otimes \mathbf{F}\mathbf{f}_0}{\|\mathbf{F}\mathbf{f}_0\|^2} + \sigma_s \frac{\mathbf{F}\mathbf{s}_0 \otimes \mathbf{F}\mathbf{s}_0}{\|\mathbf{F}\mathbf{s}_0\|^2} + \sigma_n \frac{\mathbf{F}\mathbf{n}_0 \otimes \mathbf{F}\mathbf{n}_0}{\|\mathbf{F}\mathbf{n}_0\|^2} \ , \tag{101}
$$

with the conductivities σ. Additionally, the stretch ratio λ and the stretch rate $\frac{\mathrm{d}\lambda}{\mathrm{d}t}$ are available to the ionic models, which in certain models (mainly Courtemanche and O'Hara-Rudy) can be used for stretch activated currents I_sac and a Troponin C feedback. $[\text{Ca}^{2+}]_\text{i}$ is handed to *CardioMechanics* via the function `SetActiveTensionAtQuadraturePoint()`[1], which is currently only implemented in the Land17 model. Algorithm 3 shows the principle behavior of the plugin. Due to the different requirements towards mesh resolution and time integration step size, two different meshes and two different time steps can (and should) be used for electrophysiology and mechanics problems (typically $\Delta t^\text{EP} < \Delta t^\text{M}$).

[1]The name of the function can be misleading. In an earlier version of the coupling algorithm, tension was calculated in *acCELLerate* and interpolated back to *CardioMechanics*(hence the name of the function). However, this resulted in instabilities when using stretch rate feedback and was therefore changed.

Algorithm 3 Staggered coupling of *CardioMechanics* and *acCELLerate*

```
Require: solution of (16) at time $t$ and $\frac{\partial N}{\partial X}$ on $\Omega_0^{\mathrm{EP}}$
1: repeat
2:     update $\mathbf{F}$, $\lambda$ and $\frac{\mathrm{d}\lambda}{\mathrm{d}t}$ on $\Omega^{\mathrm{EP}}$
3:     do $\mathbf{X} + \Delta\mathbf{d}^t$ on $\Omega^{\mathrm{EP}}$
4:     assemble $\mathbf{K}$ using (100)
5:     solve Monodomain equation following (98) to (99)
6:     update $[\mathrm{Ca}^{2+}]_{\mathrm{i}}$ on $\Omega^{\mathrm{M}}$
7: until $t + \Delta t^{\mathrm{EP}} = t + \Delta t^{\mathrm{M}}$
```

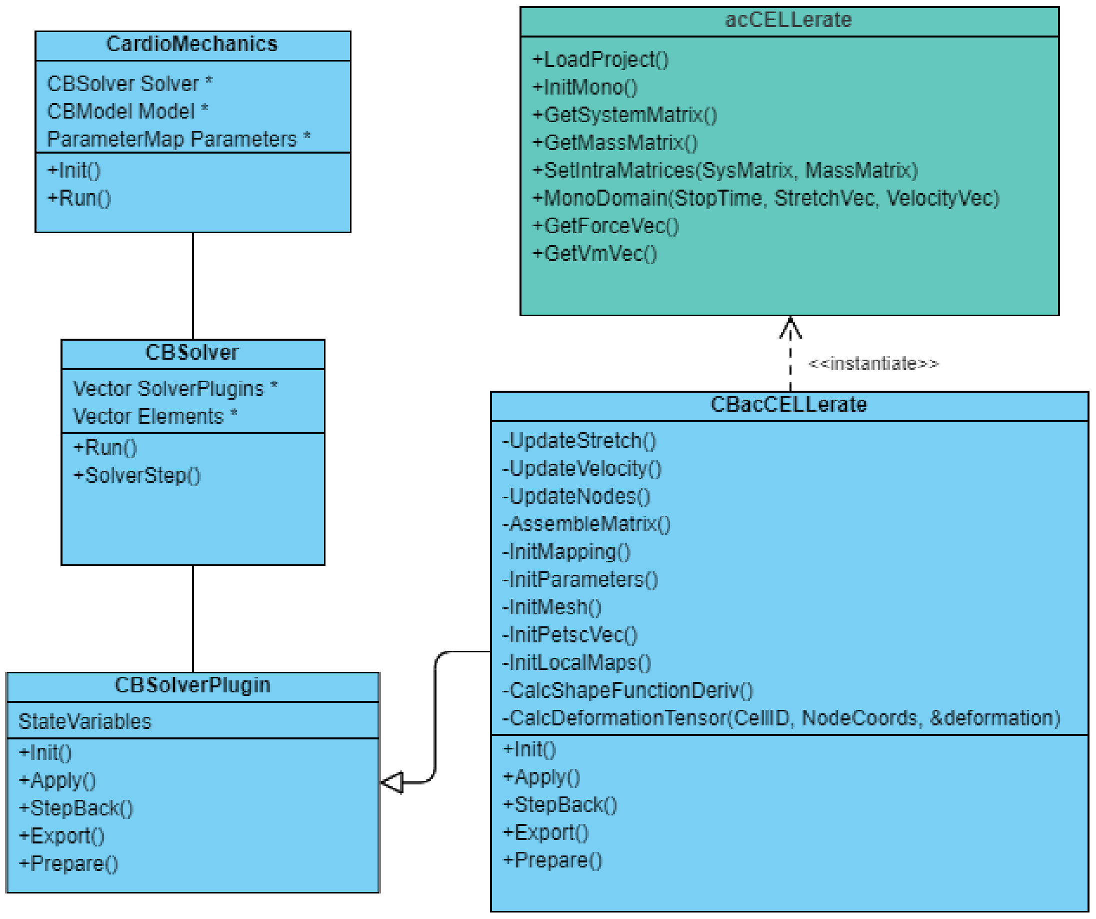

Figure 3: Class diagram of the plugin CBacCELLerate, which acts as an interface to the electrophysiology solver acCELLerate.

acCELLerate Specific Input Files Since the plugin is only an interface to an entirely different software, we need to provide an extra set of input files that are specific to the electrophysiology simulator *acCELLerate*. An overview can be found in acCELLerate Specific Input Files. The files that are required to run coupled simulations are the Config File (*.aclt), Condition File (*.acnd), Material File (*.def), Cell Model File (*.cmf), CellModel

Parameter File (*.ev), precalculated mass and stiffness matrices in PETSc format, and the mesh as a VTK Unstructured Grid Files (*.vtu). Providing a mass and stiffness matrix is necessary for *acCELLerate* to be initialized properly. *CardioMechanics* takes advantage of this by directly copying the parallel allocation of the matrices. Both of these matrices can be set up using the tool BidomainMatrixGenerator.

For efficiency reasons, the `Export` option will interpolate $[\mathrm{Ca}^{2+}]_\mathrm{i}$ and transmembrane voltage $\mathbf{u}$ onto the mechanical mesh.[2] If high resolution data on the electrophysiology mesh is required, activate export in the Config File (*.aclt) or use a Sensor File (*.txt) for specific point data. With `MEF` set to `NONE`, deformation is ignored in the monodomain equation, `FULL` will follow Algorithm 3. Using the option `Material` defines the **mechanical** materials to be used for coupling. It is important that these materials exist in both the mechanical and electrophysiological mesh (even though the material number does not need to be the same). Ideally, the points of the EP mesh lie within the mechanical mesh elements or on the edges of them to ensure that the mapping algorithm works properly. In general, mesh features that exist in the EP mesh should exist in the mechanics mesh as well. This is not necessarily the case the other way around. The `INT` array specified using the option `TissuePriority` defines a custom material priority list. Materials with higher precedence should be given first. This is especially important for smaller structures such as pectinate muscles or a thin endocardial layer that might only have a one element thickness. In such a case, if the surrounding tissue would have a higher precedence, the smaller structure is not properly considered when building the matrices. By default, lower numbers are prioritized, i.e. 1, 2, 3, 4, ..., 255. Make sure to use the same tissue priority list when you allocate the matrices with BidomainMatrixGenerator. Supply the Material File (*.def) using the option `MaterialFile` and the Config File (*.aclt) using the option `ProjectFile`. The mesh for electrophysiology should be given as a VTK Unstructured Grid Files (*.vtu) using the option `acCELLerateMesh` and needs to contain the arrays `Material`, `Fiber`, `Sheet`, and `Sheetnormal`.

```xml
<acCELLerate>
    <ProjectFile> STR </ProjectFile>
    <acCELLerateMesh> STR </acCELLerateMesh>
    <Permute> false </Permute>
    <MaterialFile> STR </MaterialFile>
    <TissuePriority> INT, INT, ... </TissuePriority>
    <ResultPreFix> STR </ResultPreFix>
    <ResultFolder> STR </ResultFolder>
    <PvdFileName> STR </PvdFileName>
    <Export> true </Export>
    <MEF> NONE/FULL </MEF>
    <Material> INT, INT, ... </Material>
    <OffsetTime> FLOAT </OffsetTime> <!-- [s] -->
    <constStretchRate> false </constStretchRate>
</acCELLerate>
```

Listing 46: .xml settings for the acCELLerate plugin

[2]Exporting on the electrophysiology mesh slowed down the simulation significantly and increased storage requirements dramatically due to the high mesh resolution. If you use the native export of *acCELLerate*, it will still slow down the simulation and increase storage requirements but not as significantly, since it will export binary PETSc vectors instead of VTK.

5.5 Export

Data export is defined in the CardioMechanics Config File (*.xml) inside the Export tag. Currently, only export as VTK files is available. Data is written to nodes and elements respectively. Some data is written to individual text files. Refer to Table 3 for information about what is exported. Try to keep exported data to a minimum, since it can significantly impact simulation performance and increase file size. Plugins call their own `Export()` function which is individually controlled in the respective plugin settings.

Table 3: Export option for CardioMechanics. k indexes elements; n indexes time.

Option	File type	Location	Data exported
Active Stress	VTK	CellData	T_a
Fiber	VTK	CellData	$\mathbf{Ff}_0$
Sheet	VTK	CellData	$\mathbf{Fs}_0$
SheetNormal	VTK	CellData	$\mathbf{Fn}_0$
Jacobian	VTK	CellData	$J = \det(\mathbf{F})$
TetgenBases	VTK, .bases	CellData	$\|\mathbf{Ff}_0\|,\|\mathbf{Fs}_0\|,\|\mathbf{Fn}_0\|$ at each quadr. point
Deformation	VTK	CellData	$\mathbf{C} = \mathbf{F}^\mathrm{T}\mathbf{F}$
DeformationEnergy	Textfile	Scalar	$E_\mathrm{def} = \sum_k V_0^k \Psi^k$
ModelVolume	Textfile	Scalar	$V = \sum_k J^k/6$
Lambda	VTK	CellData	$\sqrt{\mathbf{Ff}_0 \cdot \mathbf{Ff}_0}$, $\sqrt{\mathbf{Fs}_0 \cdot \mathbf{Fs}_0}$, $\sqrt{\mathbf{Fn}_0 \cdot \mathbf{Fn}_0}$
Cauchy	VTK	CellData	$J^{-1}\mathbf{FSF}^\mathrm{T}$
PK2Stress	VTK	CellData	$\mathbf{S}$
GreenLagrangeStrain	VTK	CellData	$E = 0.5(\mathbf{C} - \mathbb{1})$
LocalActivationTime	VTK	CellData	t_a (only when using an LAT file)
KineticEnergy	Textfile	Scalar	$E_\mathrm{kin} = 0.5\mathbf{M}\mathbf{v}^2$
DampingEnergyDissipation	Textfile	Scalar	$E_\mathrm{damp} = \sum_n \mathbf{C}^n_\mathrm{Rayleigh}\mathbf{v}^n\mathbf{d}^n$
TotalEnergy	Textfile	Scalar	$E_\mathrm{tot} = E_\mathrm{damp} + E_\mathrm{def} + E_\mathrm{kin}$
Displacement	VTK	PointData	$\Delta\mathbf{d}^n$
Velocity	VTK	PointData	$\dot{\mathbf{d}}$
Acceleration	VTK	PointData	$\ddot{\mathbf{d}}$
AbsDisplacement	VTK	PointData	$\sum_0^n \mathbf{d}^n$
NodalForces	VTK	PointData	$f_\mathrm{int} + f_\mathrm{ext}$

```
<Export>
    <Format>  VTK   </Format>
    <Prefix>  prefix  </Prefix>
    <TimeStep>  FLOAT </TimeStep>
    <Options>
      <!-- Solver export options default settings -->
      <ActiveStress>      true   </ActiveStress>
      <Fiber>           true   </Fiber>
      <Sheet>           true   </Sheet>
      <SheetNormal>      false </SheetNormal>
      <Jacobian>         false </Jacobian>
      <TetgenBases>      false </TetgenBases>
        <Deformation>      false </Deformation>
        <DeformationEnergy>   true   </DeformationEnergy>
        <ModelVolume>           true     </ModelVolume>
      <Lambda>          true   </Lambda>
      <Cauchy>          false </Cauchy>
      <PK2Stress>         true   </PK2Stress>
      <GreenLagrangeStrain>   true   </GreenLagrangeStrain>
      <LocalActivationTime> false </LocalActivationTime>
      <!-- Newmark export options default settings -->
      <KineticEnergy>       false </KineticEnergy>
      <DampingEnergyDissipation>  false   </DampingEnergyDissipation>
      <TotalEnergy>       false </TotalEnergy>
      <Displacement>        true   </Displacement>
      <Velocity>          true   </Velocity>
      <Acceleration>        true   </Acceleration>
      <AbsDisplacement>     true   </AbsDisplacement>
      <NodalForces>       false </NodalForces>
    </Options>
</Export>
```

Listing 47: .xml settings for data export options.

6 Tools

In the following, you will find short descriptions of some regularly used tools that are supplied with *CardioMechanics*. Each of them can support you in either setting up your simulations or dealing with geometry related tasks. Additionally, it is recommended to take a look into the IBT GitHub for other useful tools, e.g. rule based fiber generation algorithms (RESILIENT, LDRB Fibers), ventricular coordinate system (Cobiveco), and the powerful vtkToolbox for MATLAB.

6.1 BidomainMatrixGenerator

This tool generates matrices and vectors needed for mono- and bidomain FEM simulations with acCELLerate. That includes stiffness and mass matrices, materials vector, and index set. Output files are in PETSc format. When using the `-gzip` option, all output file will have the additional suffix .gz. You can use these files with acCELLerate without decompressing them! The only mandatory argument is the geometry as a VTK Unstructured Grid Files (*.vtu). It should contain the data arrays `Material` and one of the following arrays containing information about the fiber orientation:

- `Phi`, `Theta` are a specific angle measurement to define fiber orientation with a binary quantization of the angle using 8 bits or 1 Byte such that $\phi = \tilde{\phi} \cdot \frac{254}{\pi}$ and $\theta = \tilde{\theta} \cdot \frac{254}{\pi}$, where $\tilde{\phi}$ and $\tilde{\theta}$ are spherical coordinates given in radians. An orientation vector is calculated using

$$\mathbf{f}_0 = \begin{pmatrix} -\cos\tilde{\theta}\cos\tilde{\phi} \\ -\cos\tilde{\theta}\sin\tilde{\phi} \\ \sin\tilde{\theta} \end{pmatrix} \tag{102}$$

  $\phi = \theta = 255$ represents the isotropic case. Aligning the fibers with the x-axis can be done with $\phi = 0$ and $\theta = 127$.

- If you already have an orientation vector, you can supply it with an array called either `Fiber` or `Orientation`.

```
BidomainMatrixGenerator <.vt?> [-o <prefix>] [-v]
    [-bath <mask> [mat.def]]
    [-mono <mat.def> [mat2.def]]
    [-bi <intra.def> <extra.def>]
    [-tissue <list>] [-freq <freq>]
    [-combined] [-coupled <betaCm_dt>] [-gauss <n>]
    [-fullLumping] [-massLumping <theta>]
    [-iso] [-gzip] [-ignoreMissing]

-v --- Enable additional informational (verbose) output.

-bath <mask> [mat.def] --- All elements matching <mask> will only be part
    of the extracellular domain. Requires [-bi ...]. If [mat.def] is not
    given, <extra.def> is used.

-mono <mat.def> [mat2.def] --- Creates monodomain matrix. If [mat2.def] is
    given, the conductivity will be the harmonic mean of the two given
    conductivities for each cell (parallel circuit).

-bi <intra.def> <extra.def> --- Creates bidomain matrices with the intra-
    and extracellular conductivities from the given material files.

-combined --- Creates bidomain matrices where intracellular matrix is
    already added to extracellular matrix.

-coupled <betaCm_dt> [theta] --- Create fully coupled bidomain matrices (
    LHS and RHS) <betaCm_dt> is (beta*C_m)/dt. theta is the time
    discretization weight (0=explicit, 1=implicit, 0.5=Crank-Nicolson,
    default = 0).

-gauss <n> --- Create Gauss interpolation and integration matrices. <n> is
    the number of Gauss points. Default integration rules are provided.

-fullLumping --- Use full mass lumping and multiply intra/mono matrix with
    inverse of lumped mass matrix. Will not save mass matrix separately.
    Does not apply with -coupled. Use -massLumping 0 instead.

-massLumping <theta> --- Use mass lumping. Saved mass matrix will be
    weighted sum: M* = (theta)M + (1-theta)*M_lumped Usually, you would
    choose theta in [0,1]. Different choices may work, however.

-tissue <list> --- Comma-separated list of tissue classes in order of their
    , precedence when creating the tissue vector. Highest precedence first.
     Default: smaller numbers have higher precedence.

-freq <freq> --- Evaluate conductivities at frequency <freq> if supported.

-iso --- Ignore fiber orientation (isotropic case).

-ignoreMissing --- Ignore missing intra conductivities, i.e., set them to
    0. Only recommended to use with -bi.

-bds <value> --- CV stabilization factor beta*Delta_s.

-gzip --- Compress output files.

-o <prefix> --- Prefix for saving the resulting files. Defaults to the name
     of the input geometry sans extension. Depending on the given options,
    the following files may be created:
            <prefix>.vec - Material class vector.
            <prefix>.mat - Monodomain matrix.
            <prefix>.mass.mat - Mass matrix.
            <prefix>.intra.mat - Bidomain intra matrix.
            <prefix>.extra.mat - Bidomain extra matrix.
            <prefix>.i+e.mat - Bidomain combined matrix.
            <prefix>.n2e.mat - Node-to-element Gauss point (interpolation)
    matrix.
            <prefix>.e2n.mat - Element-to-node Gauss point (integration)
    matrix.
            <prefix>.lhs.mat - Coupled LHS matrix.
            <prefix>.rhs.mat - Coupled RHS matrix.
            <prefix>.is - Index set of intracellular domain.
```

Listing 48: Runtime options for BidomainMatrixGenerator

6.2 PETScVec2VTK

Converts a PETSc vector as e.g. provided by acCELLerate to a VTK unstructured grid (.vtu) or VTK poly data (.vtp) file for postprocessing or visualization. The input VTK file should be the same that was used to generate the matrix using BidomainMatrixGenerator. The python script ConvertPETScDir2VTK can be used to convert an entire directory at once.

```
PETScVec2VTK <vector filename> <VTK filename (.vtp/.vtu) in>  <VTK filename
     (.vtp/.vtu) out>
  [--arrayName] Name of the VTK array to be appended (default: Results)
  [--delete]  Delte PETScVec after conversion
```

Listing 49: Syntax for PETScVec2VTK

6.3 ConvertT4toT10

Use to convert linear elements to quadratic elements based on Node File (*.node) and Element file (*.ele).

```
ConvertT4toT10 <Nodes_File.node> <Elements_File.ele> <Output>
```

Listing 50: Syntax for ConvertT4toT10

6.4 VTPExtractSurfaceFromMesh

Given a .node and .ele file of a geometry as well as a surface in .vtp/.stl format, you can use this tool to generate an appropriate .sur file with matching node indexing for use in *CardioMechanics*. The `-append` option can be used to fill a single .sur file with multiple surfaces in case of repeated execution of `VTPExtractSurfaceFromMesh`.

```
VTPExtractSurfaceFromMesh <nodes_file.node> <elements_file.ele> <vtp.file>
    <element_index> <surface_index> <output_prefix>
Options:
  -free_surface_element_index <val>
  -unit <val>
  -append
  -first_index <val>
  -export_nodes_indices <exp_nodes_file.node>
  -export_stand_alone <standalone_prefix>
```

Listing 51: Syntax for VTPExtractSurfaceFromMesh

6.5 FixNodes

Can be used to apply Dirichlet boundary conditions to nodes specified in fixation.list.

```
FixNodes <input.node> <output.node> <fixation.list>
  -val <val> (optional)
  -keep_original (optional) Modify only nodes from list, if not set, all
    other nodes are set to 0 (unfixed)
```

Listing 52: Syntax for FixNodes

7 Scripts

7.1 VTK2tetgen

Convert a vtk file to .node and .ele files for CardioMechanics. If fiber orientation is given with the appropriate `Fiber`, `Sheet`, and `Sheetnormal` data arrays, a .bases file is generated as well.

```
usage: VTK2tetgen.py [-h] [-outfile OUTFILE] [-scale SCALE]
                     [-fixMaterial FIXMATERIAL [FIXMATERIAL ...]]
                     mesh

positional arguments:
  mesh                  VTK input file.

optional arguments:
  -h, --help            show this help message and exit
  -outfile OUTFILE      Set an alternative output filename PREFIX if
    desired.
  -scale SCALE          Scale the VTK input file by a factor SCALE.
  -fixMaterial FIXMATERIAL [FIXMATERIAL ...]
                        Fix nodes if they are connected to cells that match
                        the given mask in "Material". Example: "-
                        fixMaterial 32 162"
```

Listing 53: Syntax for VTK2tetgen.py

7.2 ConvertPETScDir2VTK

ConvertPETScDir2VTK.py automatically converts all *.vec files in the provided directory to *.vtu format and creates a *.pvd file containing all results ready to be viewed in ParaView.

```
Usage: ConvertPETScDir2VTK.py [options]

Options:
  --version              show programs version number and exit
  -h, --help             show this help message and exit
  -d PATH, --directory=PATH
                         Directory containing simulation results
  -g GEOFILE, --geometry=GEOFILE
                         Geometry file (*.vtu) used for the simulation
  -o FILENAME, --outfile=FILENAME
                         Filename of the resulting *.pvd file
  -c, --clean            Choose wether or not to delete converted *.vec
    files
```

Listing 54: Syntax for ConvertPETScDir2VTK.py

7.3 tuneCV

This script can be used to tune conductivity values to achieve a desired conduction velocity on tetrahedral meshes based on the method described in [17]. It requires acCELLerate, Gmsh, and VTK to be installed. If your Gmsh root directory differs from the default path used on MacOS, please provide it using one of the parameters of the script. Conduction velocity is only evaluated in one direction using a planar wavefront. For anisotropic cases, you have to do multiple runs with the desired conduction velocity. Keep in mind that this evaluation is done on a very simple geometry, meaning the actual CV in your heart geometries may differ depending on a plethora of circumstances. Nevertheless, this tool should still leave you with a good estimate.

```
usage: tuneCV.py [-h] [-resolution RESOLUTION] [-length LENGTH]
                 [-velocity VELOCITY]
                 [-gi GI] [-ge GE] [-gm GM] [-beta BETA] [-cm CM] [-
                 sourceModel {mono}]
                 [-tol TOL] [-converge] [-maxit MAXIT] [-dt DT] [-theta
                 {0,0.5,1}]
                 [-stol STOL] [-lumping {True,False}]
                 [-activationThreshold ACTIVATIONTHRESHOLD] [-log LOG]
                 [-gmshexec GMSHEXEC]
                 {BeelerReuter,TenTusscher2,CourtemancheEtAl,OHaraRudy,
    HimenoEtAl,KoivumaekiEtAl,GrandiEtAlVentricle,GrandiEtAlAtrium}
                 amplitude duration

positional arguments:
  {BeelerReuter,TenTusscher2,CourtemancheEtAl,OHaraRudy,HimenoEtAl,
    KoivumaekiEtAl,GrandiEtAlVentricle,GrandiEtAlAtrium}
                         Ionic model
  amplitude              Stimulus amplitude
  duration               Stimulus duration in s

optional arguments:
  -h, --help             show this help message and exit
  -resolution RESOLUTION mesh resolution in mm (default is 0.5 mm)
  -length LENGTH         length of strand in mm (default is 20.0 mm)
  -velocity VELOCITY     desired conduction velocity in m/s
                         (default: 0.67 m/s)
  -gi GI                 initial value intracellular conductivity in S/m
                         (default: 0.174 S/m)
  -ge GE                 initial value extracellular conductivity in S/m
                         (default: 0.625 S/m)
  -gm GM                 initial value monodomain conductivity (harmonic
                        mean of gi/ge) in S/m (default: 0.136 S/m)
  -beta BETA            Membrane surface-to-volume ratio (default: 140000
                        m^-1)
  -cm CM                Membrane capacitance per unit area
                        (default: 0.01 Fm^-2)
  -sourceModel {mono}   pick type of electrical source model
                        (default is monodomain)
  -tol TOL              Percentage of error in conduction velocity
                        (default: 0.01)
  -converge             Iterate until converged velocity (default: False)
  -maxit MAXIT          Maximum number of conduction velocity iterations
                        (default: 20)
  -dt DT                Integration time step on seconds (default: 1e-5 s)
  -theta {0,0.5,1}      Choose time stepping method 0: explicit, 0.5: CN,
                        1: implicit (default: 0.5)
  -stol STOL            Solver tolerance (default: 10^-6)
  -lumping {True,False} Use mass lumping (default: False)
  -activationThreshold ACTIVATIONTHRESHOLD
                        Threshold to record LAT (default: -0.02 V)
  -log LOG              Generate/append to a log file.
  -gmshexec GMSHEXEC    Use this option to define an alternative Gmsh path.
```

Listing 55: Syntax for tuneCV.py

7.4 CircWholeHeart

This script is an implementation of the basic 4-chamber circulatory system model described in Section 5.4.3. The script has not much use by itself, but is ready to be used with the Python library cardioemulator by Francesco Regazzoni to construct zero-dimensional emulators of the cardiac electromechanical function.

7.5 createSimDir

This script helps you to set up a directory for electromechanical simulations with *CardioMechanics*. All arguments are optional. However, if you already know which material numbers, conductivities, and ionic models you want to use, you can provide lists in the associated arguments. Other parameters for the monodomain equation such as membrane surface-to-volume ratio, membrane capacitance and integration time step can be given as well. This will automatically create the correct entries in the respective files. If no arguments are given, the files will be populated with some default arguments. Files that require the geometry need to be set up by the user. The location of files that are not generated by the script are indicated by the command line output.

```
usage: createSimDir [-h] [-material [MATERIAL ...]]
                    [-model [{BeelerReuter,TenTusscher2,CourtemancheEtAl,
   OHaraRudy,HimenoEtAl,KoivumaekiEtAl,GrandiEtAlVentricle,
   GrandiEtAlAtrium} ...]]
                    [-sigma [SIGMA ...]] [-beta BETA] [-cm CM] [-time TIME]
                    [-dt DT] [-theta {0,0.5,1}] [-sourceModel {mono}]
                    [-activationThreshold ACTIVATIONTHRESHOLD] [-dir DIR]

options:
  -h, --help            show this help message and exit
  -material [MATERIAL ...]
```

```
                        List of material numbers (default: 0)
  -model [{BeelerReuter,TenTusscher2,CourtemancheEtAl,OHaraRudy,HimenoEtAl,
    KoivumaekiEtAl,GrandiEtAlVentricle,GrandiEtAlAtrium} ...]
                        List of ionic models (default: ElphyDummy)
  -sigma [SIGMA ...]    List of monodomain conductivities in S/m (default:
                        0.136 S/m)
  -beta BETA            Membrane surface-to-volume ratio (default: 140000
                        m^-1)
  -cm CM                Membrane capacitance per unit area (default: 0.01
                        Fm^-2)
  -time TIME            Simulation time in s (default: 0.8 s)
  -dt DT                Integration time step in seconds (default: 1e-5 s)
  -theta {0,0.5,1}      Choose time stepping method 0: explicit, 0.5: CN,
    1:
                        implicit (default: 0.5)
  -sourceModel {mono}   pick type of electrical source model (default is
                        monodomain)
  -activationThreshold ACTIVATIONTHRESHOLD
                        Threshold to record LAT (default: -0.02 V)
  -dir DIR              Name of top level directory (default: SimDir)
```

Listing 56: Syntax for createSimDir.py

7.6 ClipVTUwithVTU

This script is a short post-processing tool for vtu files. Using a clipped version of the mesh created in e.g. ParaView, you can clip an entire time series while preserving all elements even when they move out of the clipped plane.

```
usage: ClipVTUwithVTU.py [-h] [-outfile OUTFILE] [-start START] [-stop STOP
    ] [-step STEP] mesh clip

positional arguments:
  mesh                  vtk/vtu/vtp data you want to clip.
  clip                  clipped version of <mesh> created in e.g. ParaView
    invtu/vtk/vtp.

optional arguments:
  -h, --help            show this help message and exit
  -outfile OUTFILE      prefix of the results file.
  -start START          start value for the time series.
  -stop STOP            stop value for the time series.
  -step STEP            increment of the time series.
```

Listing 57: Syntax for ClipVTUwithVTU.py

References

[1] T. Gerach, “Personalized electromechanical modeling of the human heart: challenges and opportunities for the simulation of pathophysiological scenarios,” 2022.

[2] J. M. Guccione, K. D. Costa, and A. D. McCulloch, “Finite element stress analysis of left ventricular mechanics in the beating dog heart,” *J. Biomechanics*, vol. 28, pp. 1167–1177, 1995.

[3] G. A. Holzapfel and R. W. Ogden, "Constitutive modelling of passive myocardium: a structurally based framework for material characterization." *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, vol. 367, pp. 3445–75, 2009.

[4] T. Usyk, R. Mazhari, and A. McCulloch, "Effect of laminar orthotropic myofiber architecture on regional stress and strain in the canine left ventricle," *Journal of Elasticity*, vol. 61, pp. 143–164, 2000.

[5] J. Lumens, T. Delhaas, B. Kirn, and T. Arts, "Three-wall segment (triseg) model describing mechanics and hemodynamics of ventricular interaction," *Annals of biomedical engineering*, vol. 37, pp. 2234–2255, 2009.

[6] N. Stergiopulos et al., "Determinants of stroke volume and systolic and diastolic aortic pressure," *American Journal of Physiology-Heart and Circulatory Physiology*, vol. 270, pp. H2050–H2059, 1996.

[7] S. Land, S.-J. Park-Holohan, N. P. Smith, C. G. Dos Remedios, J. C. Kentish, and S. A. Niederer, "A model of cardiac contraction based on novel measurements of tension development in human cardiomyocytes," *Journal of Molecular and Cellular Cardiology*, vol. 106, pp. 68–83, 2017.

[8] S. A. Niederer, G. Plank, P. Chinchapatnam, et al., "Length-dependent tension in the failing heart and the efficacy of cardiac resynchronization therapy." *Cardiovascular research*, vol. 89, pp. 336–43, 2011.

[9] J. Bestel, F. Clément, and M. Sorine, "A biomechanical model of muscle contraction," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2001*, W. J. Niessen and M. A. Viergever, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 1159–1161.

[10] T. Fritz, C. Wieners, G. Seemann, H. Steen, and O. Dössel, "Simulation of the contraction of the ventricles in a human heart model including atria and pericardium : Finite element analysis of a frictionless contact problem," *Biomechanics and Modeling in Mechanobiology*, vol. 13, pp. 627–641, 2014.

[11] T. Gerach et al., "Electro-mechanical whole-heart digital twins: A fully coupled multi-physics approach," *Mathematics*, vol. 9, p. 1247, 2021.

[12] D. Garcia et al., "Analytical modeling of the instantaneous pressure gradient across the aortic valve." *J Biomech*, vol. 38, pp. 1303–11, 2005.

[13] J. P. Mynard et al., "A simple, versatile valve model for use in lumped parameter and one-dimensional cardiovascular models," *International Journal for Numerical Methods in Biomedical Engineering*, vol. 28, pp. 626–641, 2012.

[14] M. R. Pfaller, J. M. Hörmann, M. Weigl, et al., "The importance of the pericardium for cardiac biomechanics: from physiology to computational modeling," *Biomechanics and modeling in mechanobiology*, vol. 18, pp. 503–529, 2019.

[15] M. Sellier, "An iterative method for the inverse elasto-static problem," *Journal of Fluids and Structures*, vol. 27, pp. 1461–1470, 2011.

[16] M. K. Rausch, M. Genet, and J. D. Humphrey, “An augmented iterative method for identifying a stress-free reference configuration in image-based biomechanical modeling,” *Journal of Biomechanics*, vol. 58, pp. 227–231, 2017.

[17] C. M. Costa, E. Hoetzl, B. M. Rocha, A. J. Prassl, and G. Plank, “Automatic parameterization strategy for cardiac electrophysiology simulations,” in *Computing in Cardiology 2013*. IEEE, 2013, pp. 373–376.