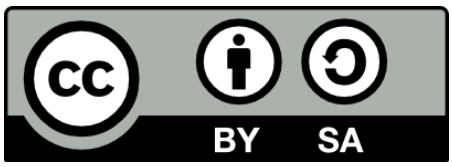
openCARP

- User's Manual -

Anton Prassl [c], Aurel Neic [f], Axel Loewe [a], Edward Vigmond [d,e], Gernot Plank [c], Gunnar Seemann [b], Jorge Sánchez [a], Martin Bishop [h], Mark Nothstein [a], Patrick Boyle [i], Philipp Zschumme [a], Rafael Sebastian [g], Yung-Lin Huang [b]

[a] Karlsruhe Institute of Technology (KIT), Germany, [b] Universitäts-Herzzentrum Freiburg, Germany, [c] Medical University of Graz, Austria, [d] University of Bordeaux, France, [e] LIRYC Electrophysiology and Heart Modeling Institute, France, [f] NumeriCor GmbH, Austria, [g] University of Valencia, Spain, [h] King's College London, UK, [i] University of Washington,USA

April 29, 2024



Contents

Introduction

openCARP is an open cardiac electrophysiology simulator for in-silico experiments. Its source code is public and the software is freely available for academic purposes. openCARP is easy to use and offers single cell as well as multiscale simulations from ion channel to organ level. Additionally, openCARP includes a wide variety of functions for pre- and post-processing of data as well as visualization. The Python-based carputils framework enables the user to develop and share simulation pipelines, i.e., automating in-silico experiments including all modeling, simulation, and evaluation steps.

The implementation of openCARP builds on two decades of experience gained from the proprietary predecessors, the Cardiac Arrhythmia Research Package (CARP) developed by Ed Vigmond and Gernot Plank and acCELLerate developed by Gunnar Seemann and Axel Loewe. Both simulators have been used in over 100 scientific studies. Gunnar Seemann (Freiburg, Germany) and Axel Loewe (Karlsruhe, Germany) received funding from the German Research Foundation (DFG) to develop a sustainable cardiac simulator. They asked Ed Vigmond (Bordeaux, France) and Gernot Plank (Graz, Austria) to join forces which they agreed to in March 2018. Since then, we are developing openCARP and will release the first version in March 2020. Beside the four group leaders, the core developer Aurel Neic joined the openCARP steering committee in November 2019.

1.1 Package description

The main benefit of openCARP is that it allows performing electrophysiological simulations of cardiac tissue with just a few steps. openCARP is backwards compatible with CARP/CARPentry allowing to reproduce a larger number of published studies. The program consists of three main components: a parabolic solver, an ionic current component, and an elliptic solver. Each of these components has a set of sockets in which to plug the components performing the bulk of the work. The parabolic solver is responsible for determining the propagation of the electrical activity by determining the change in transmembrane voltage from the extracellular electric field and the current state of the transmembrane voltage. The elliptic solver unit determines the extracellular potential from the transmembrane voltage at each time instant. The ionic model component describing ionic transmembrane currents is computed from a separate library linked in at compile time. openCARP has been written to run in either of two parallel computation modes, a shared memory model, or a distributed memory model. A large common code base is maintained between the two versions and only low-level wrapper functions implement the specific memory model. Parallelization is realized in the shared memory model based on OpenMP ® directives and native numerical libraries. For distributed memory parallelization, extensive use is made of the PETSc parallel library as well as MPI function calls.

1.2 Technical requirements

openCARP can be deployed on any Unix platform including all current Linux distributions as well as macOS. A Windows version is not available, although, technically, this would be achievable with some effort. Hardware platforms upon which openCARP was successfully used range from laptops to numerous large scale high performance computing (HPC) facilities around the world.

1.3 Distribution packages

The software is freely available for academic purposes under the Academic Public License and various distribution models were conceived to deal with technicalities of installation and support. The following openCARP packages are supported:

- **Docker container:** We provide an openCARP container based on Linux. This allows a fast execution of the software and all its command line tools. No support for meshalyzer visualization.
- **Source code:** The source code of openCARP is available under git.opencarp.org. We provide an easy way to compile using CMake which takes care of looking up the path to the dependencies. You can find a more detailed explanation in the Installation section.
- **Binary package:** Binaries for current Linux distributions and macOS are planned for the future. Currently we support the Ubuntu LTS versions.

Installation

The openCARP ecosystem comprises a number of different components:

- the openCARP simulator
- the carputils framework
- meshtool
- meshalyzer
- a number of examples.

There are several ways to install or build openCARP. Each of them is described in detail below.

2.1 Installation of Precompiled openCARP

Before installing `openCARP`, install Python3 and pip. Using your package manager would be preferable for most users.

Make sure they work by running the following commands.

```
python3 --version
python3 -m pip --version
```

For Ubuntu users: you might require to run these commands before installing your `python3-pip`.

```
add-apt-repository universe
apt-get update
```

2.1.1 Installation Packages

We provide openCARP installation packages on our release page. Currently, they will install openCARP, carputils, meshtool, and the examples. Meshalyzer is not yet included, please install it separately.

After installation, you can find the examples installed to `/usr/local/lib/opencarp/share/tutorials`.

2.1.1.0 Linux

For Ubuntu and Debian, please download the `.deb` package.

> **Note**: the packages of versions <= 9.0 are compatible with Ubuntu 18.04 and 20.04, and Debian 10. The packages of openCARP versions > 9.0 are compatible with Ubuntu 20.04 and 22.04, and Debian 11. If the latest `.deb` package is not compatible with your OS version, we invite you to use the AppImage package.

Installing the package can require to install the following packages as a prerequisite:

```
apt-get install git python3 python3-testresources python-is-python3
```

We recommand to install the package via `apt-get` like

```
cd <path-to-your-downloaded-deb-file>
apt-get update
apt-get install ./opencarp-v10.0.deb
```

For CentOS (8 or later) and Fedora (31 or later), please download the `.rpm` package. We recommand to install the package via `dnf` like

```
cd <path-to-your-downloaded-rpm-file>
dnf install ./opencarp-v10.0.rpm
```

> **Note:** If you are not using carputils, and want to run simulations with MPI using a command such as `mpiexec -n 4 openCARP ...`, you will in addition have to add the location of the MPI executables to your path: `export PATH=/usr/local/lib/opencarp/lib/petsc/bin:${PATH}`.

If the above packages don't work for you for some reason (no support of the operating system, no superuser permissions, etc.), please use the openCARP `AppImage`, which is a package for any recent enough Linux-based operating system. The documentation for using the `AppImage` is available inside the `README.md` file of the `AppImage` package or here.

The `AppImage` only supports openCARP binaries, meshtool and carputils installation, so please check other components (examples and Meshalyzer) according to your needs.

2.1.1.0 macOS

For macOS (10.14 or later), please download the `.pkg` installer. Before running the installer, call `xcode-select --install` on the command line (Programs -> Utilities -> Terminal).

> **Note:** For versions of openCARP >11.0, please use the `.pkg` installer corresponding to the architecture of your computer: use the `arm64` installer for an Apple Silicon computer, and the `x86_64` installer for systems with an Intel chip. If you don't know which one to choose, click on the Apple logo on the top left of the screen, then click on "About This Mac". If your processor's name contains "Intel", then use the `x86_64` installer ; if it contains "Apple", use the `arm64` installer.

You may need to open the installer using the context menu (right click on the item and then select `Open`) to confirm the security warning.

Note: If you are not using carputils, and want to run simulations with MPI using a command such as `mpiexec -n 4 openCARP ...`, you will in addition have to add the location of the MPI executables to your path: `export PATH=/usr/local/lib/opencarp/lib/openmpi/bin:${PATH}`

2.1.2 Confirm the Installation

To confirm if the package is successfully installed, open a terminal and try the following commands.

Run `openCARP` binary as follows. If it prints the building information (e.g. `GITtag: v10.0`, etc), your `openCARP` is successfully installed. Enjoy the simulator!

```
openCARP -buildinfo
```

Run `carputils` in Python as follows. If it prints the installation location, your `carputils` is successfully installed. Enjoy Python coding!

```
python3 -c "import carputils; print(carputils)"
```

2.1.3 Docker

If you want to keep your host system clean but still be able to run and change openCARP, our Docker container might be most suitable for you. Docker containers should also run on Windows. See docker/INSTALL.md for detailed instructions on how to install and use (docker/README.md) the openCARP docker container.

2.1.4 Installation from Source

If you expect to change the openCARP source code, we suggest you follow the build instructions (docs/BUILD.md) to install from source.

2.2 Building from Source

After making sure the prerequisites are installed, you can build openCARP using CMake (recommended) or the shipped Makefile.

2.2.1 Prerequisites

First install the following openCARP prerequisites using your package manager (e.g. apt or rpm on Linux, homebrew or macports on macOS) or from source. * binutils * C and C++ compilers (e.g. gcc/g++ or clang/clang++. On macOS, we recommend clang provided by the homebrew package llvm to support openMP) * CMake (Optional, minimum required version 3.13) * FFTW3 (Optional, for building `igbdft`) * gengetopt (mininum version for current compilers: 2.23) * gfortran * git * make * PETSc * PkgConfig * Python3 and the pip package installer * zlib development package

Building PETSc from source would be a better practice, so you could configure it depending on your needs. If you are not experienced with its various configurations, please refer to the following suggestions. Set `--prefix=` to the location you would install PETSc. After installation, set the environment variable PETSC_DIR and PETSC_ARCH accordingly.

```
./configure \
    --prefix=/opt/petsc \
    --download-mpich \
    --download-fblaslapack \
    --download-metis \
    --download-parmetis \
    --download-hypre \
    --with-debugging=0 \
    COPTFLAGS='-O2' \
    CXXOPTFLAGS='-O2' \
    FOPTFLAGS='-O2'
make all
make install
```

In this case, to make `mpirun` available on your machine, add `$PETSC_DIR/bin` to your `PATH`. That is, add the following line to your `.bashrc` (or equivalent if you don't use the `bash` shell).

```
export PATH=$PATH:$PETSC_DIR/bin
```

If you have all the listed dependencies, a quick way to get openCARP up and running is the following:

Clone the repository and enter the codebase folder `openCARP`

```
git clone https://git.opencarp.org/openCARP/openCARP.git
cd openCARP
```

To have a more stable environment, we recommend using the latest release version instead of the bleeding edge commit on the master branch:

```
git checkout latest
```

If you want to develop and push your changes back to the openCARP repository, then staying on a branch is the better option and you should skip the command above.

We provide CMake files and Makefiles for building the codebase, choose one fits your workflow.

2.2.2 Building using CMake

Use CMake to create the `_build` build folder, optionally add `-DDLOPEN=ON` to enable ionic shared library loading, and `-DCMAKE_BUILD_TYPE=Release` to optimize compilation. You can also optionally add `-DUSE_OPENMP=ON` in order to activate OpenMP parallelization (requires that OpenMP is installed on your system).

If you also want to build the additional tools (meshtool, carputils, and examples) locally, use in addition the option `-DBUILD_EXTERNAL=ON`:

```
cmake -S. -B_build -DDLOPEN=ON -DBUILD_EXTERNAL=ON -DCMAKE_BUILD_TYPE=Release
```

If you do not want to install the additional tools, you can use:

```
cmake -S. -B_build -DDLOPEN=ON -DCMAKE_BUILD_TYPE=Release
```

Compile the codebase, and all built executables will be located in the `_build/bin` folder.

```
cmake --build _build
```

If you built the additional tools, meshtool executable is also located in the `_build/bin` folder, carputils and experiments (contain examples) are cloned to the `external` folder.

Please note that in this case, carputils was downloaded, but you will still have to set it up in order to be able to run the examples. Check the installation instructions for carputils in order to do so.

2.2.3 Building using the Makefile

The codebase can be compiled using

```
make setup
```

This target includes updating the codebase (`make update`) and code compilation (`make all`).

Make sure to edit the `my_switches.def` file to match you requirements (e.g., to enable the use of shared libraries for ionic models).

Links to all compiled executables are located in the `bin` folder.

Please also refer to the carputils main page, the meshtool main page. Download the openCARP examples to the location of your choice using `git clone https://git.opencarp.org/openCAR`

2.3 Installing carputils

2.3.1 Linux / macOS

To locally run carputils, first install Python3, pip, and openCARP.

carputils has already been included to `openCARP/external/carputils` if you installed openCARP with additional tools. Otherwise, clone the carputils repository and enter the codebase folder `carputils`.

```
git clone https://git.opencarp.org/openCARP/carputils.git
cd carputils
```

2.3.1.0 Installation

This section explains how to build carputils and install it using `pip`. This is recommended if you do not expect to develop carputils.

In the carputils directory, to install carputils in the Python user-space, run

```
python3 -m pip install . --user
```

If the Python user-space bin folder is not in your `PATH`, you should add this line to your `.bashrc`.

```
export PATH="$PATH:‘python3 -m site --user-base‘/bin"
```

You can then generate the carputils settings file `settings.yaml` if it does not exist:

```
python3 ./bin/cusettings $HOME/.config/carputils/settings.yaml
```

This file contains the carputils settings, including the paths to openCARP binary files. It contains default values adapted to your local setup, but can be modified manually.

> **Note**: if the directory containing the openCARP binary files is not in your PATH (most likely if you compiled openCARP from sources), you will have to change the default value of `CARP_EXE_DIR` in the settings file. For example, if your `openCARP` executables are located at `/home/openCARP/_build/bin`, you will have to edit the file and set `CARP_EXE_DIR` to
>
> ```
> CARP_EXE_DIR:
>    CPU: /home/openCARP/_build/bin
> ```

On the other hand, if you plan to develop carputils, manually configuring it as follows is recommended.

2.3.1.0 Development Installation

Enter the `carputils` directory and follow the instructions to do a development installation of carputils.

Install the dependencies using `pip` - preferably into user-space:

```
python3 -m pip install --user -r requirements.txt
```

Add `carputils/bin` to your `PATH`, and `carputils` to your `PYTHONPATH`. For example, if carputils is located at `$HOME/openCARP/external`, add the following lines in your .bashrc:

```
export PATH=$PATH:$HOME/openCARP/external/carputils/bin
export PYTHONPATH=$PYTHONPATH:$HOME/openCARP/external/carputils
```

Create a `settings.yaml` file, which is a summary of paths and (default) parameters associated with your installation. To generate a default file, run the following command in the root folder of your carputils installation:

```
python3 ./bin/cusettings $HOME/.config/carputils/settings.yaml
```

Open the file and specify the proper directories for openCARP binary files. Notice using the *2 or 4 characters indentation* before the keyword `CPU`.

```
CARP_EXE_DIR:
    CPU: $HOME/install/openCARP/bin
```

Set directories for other openCARP tools if you use them. For example,

```
MESHALYZER_DIR:     $HOME/install/meshalyzer
```

2.3.1.0 Compiling C extensions and auto-generate code

To compile the C extensions for reading and writing meshes, just call

```
make
```

in the root folder of your carputils installation. If carputils is not installed under openCARP/external/carputils, you will also have to provide the path to the openCARP sources root folder:

```
make OPENCARP_PATH=/path/to/openCARPsources
```

If you need to use a non-default compiler (e.g. under macOS for OpenMP support), provide it as a variable

```
CC=clang make
```

A plain `make` will also auto-generate the model.ionic, model.activetension and model.mechanics python modules. They can be re-generated manually by calling

```
make models
```

2.4 Troubleshooting

The installation of some dependencies of carputils can fail if `pip` and `setuptools` are not up-to-date. They can be upgraded with the following commands:

```
python3 -m pip install --upgrade pip
python3 -m pip install --upgrade setuptools
```

2.5 Additional tools

To install additional tools from source, please refer to the instructions on the respective website:

- meshalzyer INSTALL
- meshtool README
- carputilsGUI README

Mathematical model formulation

openCARP is a generic solve for the cardiac bidomain equations and uses finite elements to discretize the cardiac domain. While the bidomain equations are considered to be an accurate description of cardiac bioelectric activity, for many scenarios of practical interest the computationally less expensive monodomain equation suffices.

This section gives an overview of the equations that are solved. The numerical schemes are detailed in section 29.

3.1 Ionic model equations

For electrophysiological simulations, the basic unit of the model is the cellular membrane. Cellular models of cardiac electrical activity were first constructed over 65 years ago, and today, these models have been refined and are routinely put into organ scale simulations. The modelling components are described below.

Ionic models refer to the electrical representation of the cell based on the movement of ions across the cell membrane, resulting in a change in the voltage across the membrane. Schematically, a capacitor is placed in parallel with a number of ionic transport mechanisms. In general, an ionic model is of the form:

$$I_{\mathrm{m}} = C_m \frac{dV_{\mathrm{m}}}{dt} + \Sigma_\chi I_\chi \tag{1}$$

where V_{m} is the voltage across the cell membrane, I_{m} is the net current across the membrane, C_{m} is the membrane capacitance, and I_χ is a particular transport mechanism, either a channel, pump or exchanger. Additional equations may track ionic concentrations and the processes which affect them, like calcium-induced release from the sarcoplasmic reticulum. By convention, outward current, i.e., positive ions leaving the cell, is defined as being positive, while inward currents are negative.

Channels are represented as resistors in series with a battery. The battery represents the Nernst Potential resulting from the electrical field developed by the ion concentration difference across the membrane. It is given by

$$E_S = \frac{RT}{zF} \ln \frac{[S]_i}{[S]_e}$$

where R is the gas constant, T is the temperature, F is Faraday's constant and z is the valence of the ion species S. Channels are dynamical systems which open and close in response to various conditions like transmembrane voltage, stretch, ligands, and other factors. The opening and closing rates of the channels vary by several orders of magnitudes, and are generally, nonlinear in nature. Thus, the equivalent electrical resistance of a channel is a time dependent quantity.

The first representation of a cardiac cell was that produced by D. Noble of a Purkinje cell, based on modification of the Hodgkin-Huxley nerve action potential. Since then, hundreds of models have been developed for many reasons. Ionic models need to be

developed to match experimental procedures if the models are to be predictive and offer insight into mechanistic workings. In mammals, action potential durations range from tens of milliseconds for mice to several hundreds of milliseconds for large animals. Atrial myocyte protein expression is quite different from that of ventricular myocytes, resulting in different action potential durations and shapes. Even nearby cells exhibit action potential differences due to slightly different levels of channel protein expression. The complexity of ionic models has been steadily growing as more knowledge is gained through better experimental techniques, equipment and specific blockers of transport mechanisms. As more mechanisms are identified as affecting electrophysiology, either directly or indirectly, they are incorporated into ionic models. Selection of the model to use is not always obvious as different models may be available for the same species and heart location, which may yield quite different behaviour. Regardless, identifying the strengths and weaknesses of an ionic model for a particular application is important.

Ionic models may be biophysically detailed or phenomenological. Biophysically detailed models attempt to discretely depict important currents and processes within the cell. Early models had approximately ten equations depicting only sodium and potassium channels, while present models have hundreds of equations taking into account not only membrane ion transporters, but intracellular calcium handling, mitochondrial function, and biochemical signalling pathways. Conversely, phenomenological ionic models use a set of currents which faithfully reproduce whole cell behaviour in terms of action potential shape and duration, and restitution properties. The currents in the phenomenological model are not physiological but can be considered as amalgamations of known currents. While these models are computationally much simpler and easier to tune, they lose the direct correspondence with the biophysics which makes implementing drug effects or cellular pathologies challenging. Finally, cellular automata models have also been used, and can be considered as a type of phenomenological model. These models do not use differential equations but have a set of rules which dictate transitions between discrete states of the model. As such, these models are computationally light, and can be as detailed as required. However, behaviour may not be as rich as differential equations. The choice of model, biophysical or phenomenological, depends on the nature of the problem being considered, and the availability of computational resources for the problem size.

3.1.1 Hodgkin-Huxley dynamics

The Hodgkin-Huxley approach is named after the Nobel laureates who were the first to develop a mathematical model of the neural action potential. Single channel activity recordings show that channels have a small set of discrete conductance states, with the channel stochastically transitioning between closed states and open states. Short time analysis of a single channel is very difficult to interpret but ensemble averaging clearly reveals smooth kinetics in changes of channel conductance. In the Hodgkin-Huxley formulation, channel conductance is assumed to be controlled by gates which take on values between 0 and unity, representing the portion of the cells in one state. Since cells have hundreds of ion channels if not more, this approximation holds well. Current flow

produced by ion species X passing through a channel is then described by

$$I_{\mathrm{S}} = \overline{g}_S \prod_n \eta_n (V_{\mathrm{m}} - E_{\mathrm{S}})$$

where $\overline{g}_{\mathrm{S}}$ is the maximum conductance of the channel and η_{n} is a gating variable. Often the gating variable is assumed to follow first order dynamics so

$$\begin{aligned} \frac{d\eta}{dt} &= \alpha(V_{\mathrm{m}})(1-\eta) - \beta(V_{\mathrm{m}})\eta \\ &= \frac{\eta_\infty(V_{\mathrm{m}}) - \eta}{\tau_\eta(V_{\mathrm{m}})} \end{aligned}$$

where α and β are rates which can be cast into an equivalent form of a steady state value (η_∞) and a rate of change (τ_η). The advantage of this latter formulation is that mathematically, the update of the gating variables can be performed by a numerical integration method, the Rush-Larsen technique, which is guaranteed to unconditionally keep the gating variable bounded within the range [0,1] while allowing a large time step.

3.1.2 Markov models

Markov models describe the channel as a set off states, such as the conductance states seen in single channel recordings, as well as the non conducting states through which a channel must pass to reach them. Transitions between states are described by rate constants which can be functions of concentrations or voltages, or fixed. These is a variables for each state which represent the portion of channels in a cell which are in that state. As such, the sum of state variables is unity. A Hodgkin-Huxley made can be converted to a Markov model by considering a gating variable as a two state model. However, the advantage of the Markov representation is its ability to model drug interaction. Essentially, drug binding doubles the number of possible states in an ionic model, augmenting the original states with drug-bound versions. One may easily limit drug binding to a particular subset of channel states, and more finely control channel kinetics.

3.2 Bidomain equations

The bidomain equations relate intracellular potential, ϕ_i, to the extracellular potential, ϕ_e, through the transmembrane current density, I_m. They are written in the standard form as

$$\nabla \cdot \sigma_i \nabla \phi_i = \beta I_m - I_i \tag{2}$$

$$\nabla \cdot \sigma_e \nabla \phi_e = -\beta I_m - I_e \tag{3}$$

where σ_i and σ_e are the intracellular and extracellular (homogenized) bidomain conductivity tensors, respectively, β is the bidomain surface-to-volume ratio, I_e and I_i are an extracellular and intracellular current density stimuli, respectively, and I_m is the transmembrane current. Note that β in the bidomain equations is the ratio of cell surface to tissue volume, i.e. a tissue-level surface-to-volume ratio. As such, it is given by the product of the cellular-level surface-to-volume ratio (cell surface to cell volume, `IMPregion[].cellSurfVolRatio`) and the volume fraction occupied by cells (cell volume to tissue volume, `IMPregion[].volFrac`).

The transmembrane current is given by

$$V_m = \phi_i - \phi_e \tag{4}$$

$$I_m = C_m \frac{\partial V_m}{\partial t} + I_{ion} - I_{tr} \tag{5}$$

$$\frac{\partial \eta}{\partial t} = f(V_m, \eta, t) \tag{6}$$

$$I_{ion} = g(V_m, \eta, t), \tag{7}$$

where I_{tr} is an assumed transmembrane current density stimulus, as delivered by an intracellular electrode, C_m is the capacitance per unit area (fixed to $1\mu\text{F/cm}^2$), V_m is the transmembrane voltage, which is defined as $\phi_i - \phi_e$, and I_{ion} is the current density flowing through the ionic channels. The system is complemented by the following homogeneous Neumann boundary conditions:

$$\sigma_i \nabla \phi_i \cdot \vec{n} = 0 \sigma_e \nabla \phi_e \cdot \vec{n} = 0 \tag{8}$$

where $\vec{n}$ is a vector normal to the boundary. When the cardiac tissue is surrounded by a bath (extracellular medium only, for example torso, conductivity σ_b), ϕ_e and the current are supposed to be continuous across the tissue to bath interface:

$$\phi_e = \phi_b \sigma_e \nabla \phi_e \cdot \vec{n} = \sigma_b \nabla \phi_b \cdot \vec{n} \tag{9}$$

Ionic currents I_{ion} (equation 7) depend on the membrane state and various ion concentrations throughout the cell and in the interstitial domain which are given by the state equation 6. This form of the bidomain equations is referred to as the *parabolic-parabolic* form since both equations 2 and 3 are parabolic PDEs.

By adding Eq. 2 and Eq. 3 and using the definition of V_m, the equations can be cast in a slightly different form with V_m and ϕ_e as the independent variables [7].

$$\nabla \cdot (\sigma_i + \sigma_e) \nabla \phi_e = -\nabla \cdot \sigma_i \nabla V_m - I_e - I_i \tag{10}$$

$$\nabla \cdot \sigma_i \nabla V_m = -\nabla \cdot \sigma_i \nabla \phi_e + \beta I_m \tag{11}$$

Eq. 10 is an elliptic equation and Eq. 11 is a parabolic equation. Hence, this recast is referred to as the *elliptic-parabolic* which is the more popular cast for solving the equations since the two main variables of interest, V_m and ϕ_e, are retained. This is more convenient for experimental validation at the tissue scale.

3.3 Monodomain equation

Assuming that anisotropy ratios between intracellular and extracellular domains are equal, that is, the tensors can be related by a scalar, λ, like

$$\sigma_e = \lambda \sigma_i \tag{12}$$

we can recast the bidomain equation in a simpler form. Plugging Eq. 12 into 2 and using 4 yields

$$\nabla \cdot \sigma_i \nabla \phi_i = \beta I_m - I_i \tag{13}$$

$$\nabla \cdot \sigma_i \nabla \phi_e = \nabla \cdot \sigma_i \nabla \phi_i - \nabla \cdot \sigma_i \nabla V_m = -\frac{1}{\lambda}(\beta I_m + I_e). \tag{14}$$

Subtracting 14 from 13 and multiplying with $\lambda/(1+\lambda)$ results in

$$\frac{\lambda}{1+\lambda} \nabla \cdot \sigma_i \nabla V_m = \beta I_m + \frac{1}{1+\lambda} I_e - \frac{\lambda}{1+\lambda} I_i. \tag{15}$$

Now we subtract 13 from 15 to arrive at

$$\frac{\lambda}{1+\lambda} \nabla \cdot \sigma_i \nabla V_m = \beta I_m + \underbrace{\frac{1}{1+\lambda} I_e - \frac{\lambda}{1+\lambda} I_i}_{-\beta I_{tr}}. \tag{16}$$

$$(1+\lambda) \nabla \cdot \sigma_i \nabla \phi_e = -\nabla \cdot \sigma_i \nabla V_m - I_e - I_i \tag{17}$$

As indicated in Eq. 16, the combined effect of intracellularly and extracellularly injected stimulus currents can be interpreted as a depolarizing transmembrane stimulus if we define

$$I_{tr} = -\frac{1}{\beta} \left(\frac{1}{1+\lambda} I_e - \frac{\lambda}{1+\lambda} I_i \right). \tag{18}$$

The choice $I_e = -I_i$ at any given site is equivalent to a transmembrane current stimulus of strength I_i/β, that is,

$$\beta I_{tr} = -\left(\frac{1}{1+\lambda} I_e - \frac{\lambda}{1+\lambda} I_i \right) = -\left(-\frac{1}{1+\lambda} I_i - \frac{\lambda}{1+\lambda} I_i \right) = -\left(-\frac{1+\lambda}{1+\lambda} I_i \right) = I_i. \tag{19}$$

This is consistent with the notion that the injection of a positive current I_i into the intracellular space increases ϕ_i which exerts a depolarizing effect upon $V_m = \phi_i - \phi_e$.

Further, in this case the current terms in Eq. 17 cancel out. As expected, any current injected in one domain is withdrawn at the same spot in the other domain. Therefore no current flow occurs as a consequence of I_e or I_i and no extracellular potential field is set up. All changes in ϕ_e are caused indirectly then via changes in V_m.

Inspecting Eqs. 16-17 reveals that, unlike in the full bidomain case where $\sigma_e \neq \lambda\sigma_i$ holds, the temporal evolution of the transmembrane voltage in Eq. 16 is fully independent of ϕ_e. Hence, if only the evolution of V_m is of interest, only Eq. 16 needs to be solved, but not the elliptic PDE given by 17 which is a more expensive task. For more detailed considerations we refer to the report by Nielsen et al. [4].

It is worth noting that in 1D monodomain and bidomain equations are equivalent when using half the harmonic mean of the intracellular and extracellular bidomain conductivities as the monodomain conductivity. This holds true also for planar wave fronts in 3D propagating along any eigenaxes of the conductivity tensors. The monodomain is an approximation which can be used whenever the effect of extracellular fields upon tissue polarization can be ignored. As mentioned above, since the temporal evolution of the V_m in Eq. 16 is fully independent of ϕ_e, any changes in the extracellular potential fields cannot exert any influence upon V_m.

Therefore, under the assumption of equal anisotropy ratios one needs to solve only the parabolic PDE above with the conductivity set to $\sigma_i\sigma_e(\sigma_i + \sigma_e)^{-1}$, which is half the harmonic mean. This yields

$$\nabla \cdot (\sigma_m \nabla V_m) = \beta I_m + \beta I_{tr} \tag{20}$$

where the bidomain equivalent monodomain conductivity σ_m is given as

$$\sigma_m = \sigma_i\sigma_e(\sigma_i + \sigma_e)^{-1}. \tag{21}$$

The mondomain system is complemented by the following homogenoeus Neumann boundary condition:

$$\sigma_m \nabla V_m \cdot \vec{n} = 0, \tag{22}$$

where $\vec{n}$ is a vector normal to the boundary.

3.4 Units

Units are discussed for the monodomain equations for the sake of simplicity, but the same is valid for the bidomain. The monodomain equation can be written in the form

$$\nabla \cdot (\hat{\sigma}_m \nabla V_m) = \beta I_{ion} + \beta C_m \frac{\partial V_m}{dt}, \tag{23}$$

where transmebrane stimulus currents (in $\mu A/cm^2$) are included in I_{ion}. The units as defined in Table 1 are used. There are a few exceptions. Although t is input in ms in general, this is not the case for dt, where μs are used.

Rewriting the equation in a semi-discrete form reveals that the equation is not consistent in terms of units, neither with the input units nor the internal units:

Symbol	Input Unit	Internal Unit	Conversion
V_m	mV	mV	1
I_{ion}	$\mu A/cm^2$	$\mu A/cm^2$	1
t	ms	ms	1
x, y, z	μm	μm	1
β	μm^{-1}	μm^{-1}	1
C_m	fixed value (1)	$\mu F/cm^2$	-
$\hat{\sigma}_m$	S/m	$mS/\mu m$	10^3

Table 1: Units used in openCARP

$$\frac{1}{\Delta} \cdot \left(\hat{\sigma}_m \frac{1}{\Delta} V_m \right) = \beta I_{ion} + \frac{\beta C_m}{\Delta t} \Delta V_m \tag{24}$$

Inconsistencies arise due to the use of two different units for space coordinates, μm on the left hand side and for β, cm on the right hand side for C_m and I_{ion}. Nonetheless, the choice of internal units make sense. μm is an appropriate unit for resolving the tissue and cm is used in almost any model dealing with membrane kinetics. The inconsistency will be dealt with in the final form after spatial discretization using the finite element method (see section 3.7).

3.5 The Galerkin finite element formulation

The Galerkin method is a special case of the Method of Moments which minimizes the weighted residual. The solution is multiplied with a weighting function, ω, and integrated over the entire domain, Ω:

$$\int_\Omega \nabla \cdot (\hat{\sigma}_m \nabla V_m)\, \omega \, d\Omega = \beta \int_\Omega \left(I_{ion} + C_m \frac{\partial V_m}{dt} \right) \omega \, d\Omega \tag{25}$$

We assign a set of functions which will represent our solution. For FEM, these function are local in scope and are the shape functions of the element, α_i. For the Galerkin method, we choose the weighting function to be the shape functions. In operator notation, $\mathcal{L} = \nabla \cdot \sigma \nabla$, and $b = \beta I_m$

$$\mathcal{L}\phi = b \tag{26}$$

$$< \mathcal{L}\phi, \omega > = < b, \omega > \tag{27}$$

$$\underbrace{< \mathcal{L} \sum_i \alpha_i, \alpha_j >}_{\mathbf{K}} = \underbrace{< \sum_i b_i \alpha_i, \alpha_j >}_{\mathbf{M}} \tag{28}$$

3.6 The local stiffness matrix

$$K_{ij} =< \mathcal{L}\alpha_i, \alpha_j >= \int_{\Omega_e} \nabla\alpha_i \cdot \sigma_{m_{ij}} \nabla\alpha_j d\Omega_e \tag{29}$$

where K_{ij} is a single entry of the local stiffness matrix which we obtain by integrating over the volume Ω_e of the element e. In terms of units, the stiffness matrix is $[mS]$:

$$(\nabla\alpha_i)\left[\frac{1}{\mu m}\right](\sigma_{m_{ij}})\left[\frac{mS}{\mu m}\right](\nabla\alpha_j)\left[\frac{1}{\mu m}\right](\Omega_e)\left[\mu m^3\right] = (K_{ij})\,[mS] \tag{30}$$

3.7 The local mass matrix

$$M_{ij} =< \alpha_i, \alpha_j >= \int_{\Omega_e} \alpha_i \alpha_j d\Omega_e \tag{31}$$

where M_{ij} is a single entry of the local stiffness matrix which we obtain by integrating over the volume Ω_e of the element e. In terms of units, the stiffness matrix is $[\mu^3]$:

$$(\alpha_i)\,[-]\,(\alpha_j)\,[-]\,(\Omega_e)\left[\mu m^3\right] = (M_{ij})\left[\mu^3\right] \tag{32}$$

Rewriting 25 in a semi-discrete form yields

$$\mathbf{Kv} = \underbrace{\frac{\beta C_m}{\Delta t}}_{\kappa}\mathbf{M}\Delta\mathbf{v} + \beta\mathbf{MI_{ion}} \tag{33}$$

In terms of units, κ is evidently inconsistent and needs to be converted to ensure compatibility between left-hand and right-hand side:

$$\kappa = \left[\frac{1}{\mu m}\right]_{(\beta)}\left[\frac{\mu F}{cm^2}\right]_{(C_m)}\left[\frac{1}{ms}\right]_{(\Delta t)} \tag{34}$$

Rewriting μF as

$$\mu F = \frac{\mu A s}{V} = \frac{mA \cdot ms}{V} = mS \cdot ms \tag{35}$$

and inserting into 34 yields

$$\kappa = \left[\frac{1}{\mu m}\right]_{(\beta)}\left[\frac{mS \cdot ms}{cm^2}\right]_{(C_m)}\left[\frac{1}{ms}\right]_{(\Delta t)} = \left[\frac{mS}{cm^2 \mu m}\right]_{(\kappa)} \tag{36}$$

To arrive at μA, as required to match the left-hand side, when multiplying with $\mathbf{M}$, we convert κ to a per μm base. The required conversion factor is

$$\kappa = \left[\frac{mS}{10^8 \mu m^2 \mu m}\right]_{(\kappa)} = 10^{-8}\left[\frac{mS}{\mu m^3}\right]_{(\kappa)} = 10^{-8}\tilde{\kappa} \tag{37}$$

Using $\tilde{\kappa}$ results in a consistent equation now where all terms are given in μA, except for the term linked to I_{ion}. No unit conversion is required for I_{ion} due to the operator splitting technique used within openCARP:

$$\frac{\beta C_m}{\Delta t}\mathbf{M}\Delta\mathbf{v} = \mathbf{K}\mathbf{v} \tag{38}$$

$$C_m\frac{\Delta\mathbf{v}}{\Delta t} = -\mathbf{I_{ion}} \tag{39}$$

That is, the inconsistency is resolved since the terms given in μm drop out and only those terms given on a per cm base remain. Note that in a computing scheme where the ODE is not split of from the parabolic PDE, an additional unit conversion would be required to make the term $\beta\mathbf{M}I_{ion}$ consistent. For the sake of completeness, in this case I_{ion} has to be multiplied by

$$\begin{aligned}\beta\mathbf{M}I_{ion} = \left[\frac{1}{\mu m}\right]_{(\beta)}\left[\mu m^3\right]_{(\mathbf{M})}\left[\frac{\mu A}{cm^2}\right]_{(I_{ion})} = \\ \left[\frac{1}{\mu m}\right]_{(\beta)}\left[\mu m^3\right]_{(\mathbf{M})}\left[\frac{\mu A}{10^8\mu m^2}\right]_{(I_{ion})} = \beta\mathbf{M}\tilde{i}_{ion}\end{aligned} \tag{40}$$

where

$$\left[\frac{\mu A}{\mu m^2}\right]_{(\tilde{i}_{ion})} = 10^{-8}\left[\frac{\mu A}{cm^2}\right]_{(I_{ion})} \tag{41}$$

to arrive at μA like with all other terms of the equation.

The parabolic PDE is now consistent as can be seen here:

$$[mS]_{(\mathbf{K})}\cdot[mV]_{(\mathbf{v})} = \left[\frac{mS}{\mu m^3}\right]_{(\tilde{\kappa})}\cdot\left[\mu m^3\right]_{(\mathbf{M})}\cdot[mV]_{(\Delta\mathbf{v})} = [\mu A] \tag{42}$$

3.8 Conductivities

β is the ratio of the surface area of all cells within a volume to the volume. It is based on the cellular surface to volume ratio and the number of cells per unit volume. Conductivities are not material properties alone but also take into account the geometry. Let σ_ξ be the true isotropic conductivity of domain ξ. In any one direction, we homogenize over 1 domain to arrive at an equivalent conductivity in principal direction η, $\sigma^h_{\xi,\eta}$:

$$\frac{L}{\sigma^h_{\xi,\eta}\overline{\Gamma}} = \frac{1}{N}\sum_{k=1}^{N}\left(\int_0^{L_\eta}\frac{d\eta}{\sigma_\xi\mathcal{A}(\eta)} + \sum_j R_{k,j}\right) \tag{43}$$

where N is the number of parallel cells, $\overline{\Gamma}$ is the average domain dimension in direction η, L_η is the dimension of the cell in η, $\mathcal{A}_\eta$ is the cross-sectional area along η, and $R_{k,j}$ is the resistance of any gap junctions crossed. The principle directions are along the fibre (f), in the plane of the sheet (s), and the sheet normal (n).

To homogenize over both domains, the following relationship must be obeyed

$$\int_{\Omega} \nabla \cdot \overline{\sigma}_{\xi} \nabla \phi_{\xi} \, d\Omega = \pm \beta I_{ion} \tag{44}$$

which implies that

$$\overline{\sigma}_{\xi} = \begin{bmatrix} A_{\xi,f} & 0 & 0 \\ 0 & A_{\xi,s} & 0 \\ 0 & 0 & A_{\xi,n} \end{bmatrix} \overline{\sigma}_{\xi}^{h} \tag{45}$$

where $A_{\xi,\eta}$ is the portion of the volume cross-section in direction η occupied by domain ξ. If we assume the following mapping which scales the domains in the unit volume to occupy the entire volume

$$(f, s, n) \xrightarrow{T} (f', s', n') \tag{46}$$

where

$$T(f, s, n) = \left(f, \frac{s}{L_{\xi,s}}, \frac{n}{L_{\xi,n}} \right) \tag{47}$$

This leads to the relationship that the fraction of the volume occupied by domain ξ is the Jacobian of this scaling.

File Formats

The input and output files used in openCARP adhere to their own specific structure. This structure brings flexibility making different datasets easier to export or import. These file formats have been optimized to handle large amounts of data for fast I/O for pre- and post-processing. Using meshtool you can also import/export to different standard file formats. (e.g. VTK, obj, ensight).

4.1 Input Files

4.1.1 Parameter file

Extension: .par

The parameter file contains all the input options to define a simulation. These parameters can be specified either by using this file or by giving them as command line arguments. The last option definition, whether in a file or on the command line, overrides any previous definition. The command line is parsed from left to right. Command line options may be placed in the parameter file by removing the preceding `-` and using `=`. For example, the time step is specified on the command line by `-dt f` where `f` is a floating point number. This can be placed in a parameter file on a line by itself as `dt = f`.

openCARP uses the parameter file as an input file to read all information regarding input files, output files and parameters for the simulation as follows:

> `openCARP +F parameter.par`

4.1.2 Element file

Extension: .elem

In general, openCARP supports various types of elements which can be mixed in a single mesh. The file begins with a single header line containing the number of elements, followed by one element definition per line. The element definitions are composed of an element type specifier string, the nodes for that element, then optionally an integer specifying the region to which the element belongs.

In the table below is an example of the file format:

Format	Example
n	2000
T_0 $N_{0,0}$ $N_{0,1}$ $N_{0,_{mi-1}}$ $[\text{elemTag}]_0$	Tt 0 1 2 150 1
...	...
T_{n-1} $N_{n-1,0}$ $N_{n-1,1}$ $N_{n-1,_{mi-1}}$ $[\text{elemTag}]_{n-1}$	Tt 123 124 125 210 10

The node indicies are determined by their order in the points file. Note that the nodes

are 0-indexed, as indicated in Sec. Node file above. The element formats supported in openCARP, including their element type specifier strings are given in table below. Note that cH element type is for internal use only, not supported in mesh files. The ordering of nodes in each of the 3D element types is shown in Nodal ordering of 3D element types.

Type Specifier	Description	Interpolation	#Nodes
Ln	Line	linear	2
cH *	Line	cubic	2
Tr	Triangle	linear	3
Qd	Quadrilateral	Ansatz	4
Tt	Tetrahedron	linear	4
Py	Pyramid	Ansatz	5
Pr	Prism	Ansatzv	6
Hx	Hexahedron	Ansatz	8

4.1.3 Node file

Extension: .pts

The node (or points) file starts with a single header line with the number of nodes, followed by the coordinates (x,y,z) of the nodes, one per line, in μm.

Format	Example
n	2000
$x_0\ y_0\ z_0$	1000.00 1000.00 300.00
...	...
$x_{n-1}\ y_{n-1}\ z_{n-1}$	10000.00 10000.00 3000.00

4.1.4 Fiber orientation file

Extension: .lon

Fibres are defined on a per-element basis in openCARP. They may be defined by just the main fibre direction, in which case a transversely isotropic conductivity must be used, or additionally specifying the sheet (or transverse) direction to allow full orthotropy in the model. The file format starts with a single header line with the number of fibre vectors defined in the file (1 for fibre direction only, 2 for fibre and sheet directions), and then one line per element with the values of the fibre vector(s). Note that the number of fibres must be equal to the number of elements read from the element file.

Format	Example
n_f	2
$f_{0,x}\ f_{0,y}\ f_{0,z}\ s_{0,x}\ s_{0,y}\ s_{0,z}$	0.831 0.549 0.077 0.473 -0.775 0.417
...	...

Format	Example
$f_{n-1,x}$ $f_{n-1,y}$ $f_{n-1,z}$ $s_{n-1,x}$ $s_{n-1,y}$ $s_{n-1,z}$	0.0 0.0 0.0 0.0 0.0 0.0

4.1.5 Pulse definition file

Extension: .trc

The pulse definition file allows to introduce a signal with a predefined shape by the user. It starts with a single header line that is the total number of samples of the signal. Then, the signal is defined with one line per sample (time followed by the signal value to be applied between this time and the following time step). Note that the time must always increase from line to line.

Format	Example
n	1000
t_0 val_0	0 60
...	...
t_{n-1} val_{n-1}	1000 0

4.1.6 Vertex specification file

Extension: .vtx

Format	Example
n	1000
intra | extra	intra
node_0	0
...	...
node_{n-1}	123

4.1.7 Vertex adjustment file

Extension: .adj

Format	Example
n	1000
intra | extra	intra
node_0 val_0	0 1000
...	...
node_{n-1} val_{n-1}	123 65

4.2 Output Files

4.2.1 IGB file

Extension: .igb

openCARP simulations usually export data to a binary format called IGB. IGB is a format developed at the University of Montreal and originally used for visualizing regularly spaced data. An IGB file is composed of a 1024-byte long header followed by binary data which is terminated by the ASCII form feed character ($\hat{}L/character12$). For unstructured grids, the individual dimensions are meaningless but x * y * z should be equal to the number of vertices. The header is composed of strings of the following format, separated by white space: Keyword: value. Note that the header must be padded to 1024 bytes.

Keywords	Type	Keywords	Type
x	int	inc_x	float
y	int	inc_y	float
z	int	inc_z	float
t	int	dim_x	float
systeme	string	dim_y	float
type	string	dim_z	float
unites	string	unites_x	string
facteur	float	unites_y	string
zero	float	unites_z	string
org_x	float	comment	string
org_y	float	aut_name	string
org_z	float	transparent	hex
org_t	float		

Format	Example
x:int y:int z:int t:int type:string systeme:string org_t:float dim_t:float unites_x:string unites_y:string unites_z:string unites_t:string unites:string	x:333136 y:1 z:1 t:1452 type:float systeme:little_endian org_t:0 dim_t:1451 unites_x:um unites_y:um unites_z:um unites_t:ms unites:mV facteur:1 zero:0
float_{n0} float_{n1} ... float_{n-1}	-80 -80 ... -80
...	...
float_{n0} float_{n1} ... float_{n-1}	-75 -75 ... -75

4.2.2 Dynamic points file

Extension: .dynpts

Points may also move in space as functions of time. This can be used to represent displacement during contraction, for example. The number of points must remain constant for all time instants, as well as the elements defined by them. Dynamic point files use the IGB format (see IGB file) with data type vec3f.

4.2.3 Vector data file

Extension: .vec and .vpts To display vector data, an auxiliary set of points must be defined by a file with the .vpts suffix. It follows the same format as the Node file. A file with the same base name but having the extension .vec defines the vector and scalar data. The scalar datum, as indicated, is optional. The .vec format can also be used for visualization of fiber orientations stored in .lon files. For this sake, a .vpts file must be generated holding the coordinates of the centers of each element in the mesh. This is conveniently achieved with GlElemCenters and changing the fiber file extension to .vec. For example, for a mesh with elements and where corresponds to the components of the fiber vector :

Format	Example
data.x data.y data.z [scalar_datum]	8.12453e-01 -2.22623e-01 1.25001e00
data.x data.y data.z [scalar_datum]	5.68533e-01 -1.81807e-01 1.04956e00
data.x data.y data.z [scalar_datum]	5.70671e-01 -1.67572e-01 1.06615e00

Vector data can also be input as an IGB file using the types vec3f, vec4f, vec3d, vec4d where 3 or 4 refers to the number of elements in each datum, and d and f refer to float or double. The first 3 elements define the value of the vector field, and the optional 4-th element is the scalar component as above. This file has the suffix .vec.igb.

4.2.4 Auxiliary grid file

Extension: .pts_t, .elem_t, .dat_t

This format is used by Meshalyzer to display additional data on an auxiliary grid. The grid may contain any of the elements forming the main grid (points, lines, surface elements, volume elements), and the elements may change as a function of time. If scalar data were already read, the number of time instants in the auxiliary grid must either be one, or the same as the scalar data. The points file may have only one time instant, which is then assumed to be constant and applied for all time steps.

A vertex file is mandatory and has the extension .pts_t. An element file is optional. If present, it has the same base name as the vertex file but with the extension .elem_t. Scalar data may also be optionally defined on the auxiliary grid. The file has the same basename as the vertex file but with the extension .dat_t.

Single cell simulations

The single cell experiment tool bench is quite versatile and can be used to replicate a wider range of single cell experimental protocols. Bench handles all time units in ms; all voltages in mV and currents in pA/pF (or μA/cm^2 considering the fixed C_m of 1pF/cm^2). The following sections demonstrate how one explores the available bench functions and explains the command line arguments. To list all available bench options, use:

```
> bench --help
```

You can find a detailed explanation of the parameters available in bench in the Bench commands section.

5.1 Running a simulation

5.1.1 Running in plain mode

Running bench in plain mode is simple and fast. You can call bench in the terminal window by command line as shown in this example:

```
> bench --imp Courtemanche --imp-par "GKr*1.6" --stim-curr 20 --
    numstim 4 --bcl 1000
```

5.1.2 Running with carputils

Setting up an in-silico experiment with carputils and bench allows to combine several steps to create more complex experiments than in the plain mode. carputils also provides interfaces for visualization and can be used to expose only certain bench parameters to the user.

This is an example of an experiment defined using carputils:

```
import os
from datetime import date

from carputils import settings
from carputils import tools
from carputils import mesh
from carputils import testing

def parser():
    parser = tools.standard_parser()
    group  = parser.add_argument_group('experiment specific options')
    group.add_argument('--imp',  default='Courtemanche',
                       choices=['TT2','Courtemanche','HH'],
                       help='pick ionic model (default is TT2). For a
    full list look inside opencarp installation folder /physics/limpet/
    models')
```

```
    group.add_argument('--duration',
                       type=float, default=1000.,
                       help='Duration of simulation [ms] (default is
    1000.)')
    group.add_argument('--stim_strength',
                       type=float, default=60.,
                       help='pick transmembrane current stimulus
    strength in [uA/cm^2] (default is 60.)')
    group.add_argument('--stim_dur',
                       type=float, default=2.,
                       help='pick transmembrane current stimulus
    duration in [ms] (default is 2.)')
    #-----------------------------------------------------------------
    group.add_argument('--imp_par', default='',
                       help='parameter to modified in ionic model.')
    group.add_argument('--plug_in', default='',
                       help='plug_in to be added to base ionic model.')
    group.add_argument('--plug_par', default='',
                       help='parameter to modified in plug-in.')
    group.add_argument('--bcl',
                       type=float, default=1000.,
                       help='Basic Cycle Length for stimulation in [ms].
    ')
    group.add_argument('--overlay', default='',
                       choices=['True','False'],
                       help='Overlays all existing experiments if True')
    #-----------------------------------------------------------------
    return parser

def jobID(args):
    today = date.today()
    return '{}_basic_{}{}{}{}'.format(today.isoformat(),args.imp,args.
    imp_par,args.plug_in,args.plug_par)

@tools.carpexample(parser, jobID)
def run(args, job):
    # define & configure EP model
    cmd = [ settings.execs.BENCH,
            '--imp={}'.format(args.imp) ]

    if args.imp_par is not '':
        cmd += [ '--imp-par', args.imp_par ]

    # define & configure Ionic plug-in
    if args.plug_in is not '':
        cmd += [ '--plug-in', args.plug_in ]

    if args.plug_par is not '':
        cmd += [ '--plug-par', args.plug_par ]

    # setup stimulus
    cmd += [ '--stim-curr', args.stim_strength,
             '--numstim',  int(float(args.duration)/float(args.bcl)+1), #
     Number of stimulus based on the simulation duration and BCL
```

```
                '--bcl', args.bcl ]

    # run bench with available ionic models
    cmd  += ['--duration', args.duration ]

    # executing bench
    job.mpi(cmd,'Running {}'.format(args.imp))

if __name__ == '__main__':
    run()
```

Tissue simulations

6.1 Running a simulation

openCARP (similar to bench) offers two ways for defining in-silico experiments: plain mode and carputils. We recommend using carputils which gives the ability to create multi-step experiments and easily share them.

6.1.1 Running in plain mode

Running a simulation in plain mode needs a parameter file.

All available option can be displayed in the terminal using:

```
> openCARP +Help
```

More verbose explanation of command line parameters is obtained by using the *+Doc* parameter which prints the same output as *+Help*, but provides more details on each individual parameter:

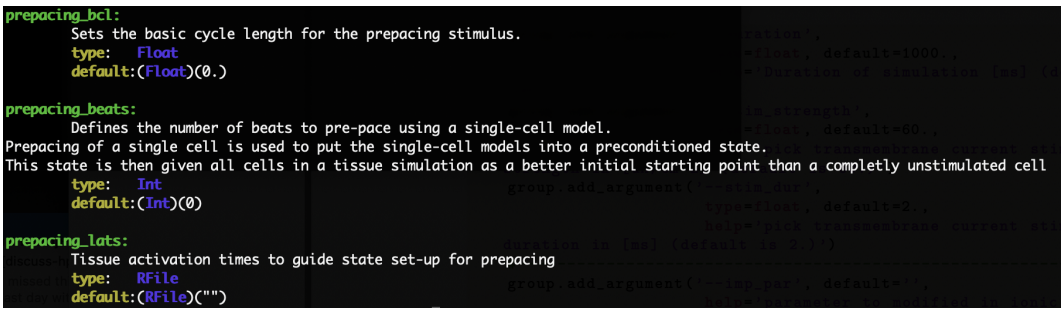
```
> openCARP +Doc
```

and for specific help on one parameter for example type:

```
> openCARP +Help phys_type[0].ptype
```

If you have carputils installed, you can also use the script `carphelp` to search for parameters related to given keywords and get their documentation:

```
> carphelp prepacing
```



For a complete detailed explanation of the parameters available in openCARP please see the openCARP commands section. Down below, we describe in detail a parameter file used to simulate a bidomain problem.

```
#------------------
#Output folder name
#------------------
simID = TESTOUTPUT #Name of the output folder created when a simulation
    starts
```

```
#--------------
#Mesh basename
#--------------
meshname = testmesh #Basename of the mesh in openCARP format

#-----------------------------------------------------------------
#Definition of physics regions for each unique tagged region in the mesh
#-----------------------------------------------------------------
num_phys_regions = 2 #Total number of physical regions. In a bidomain
    simulation we have intracelluar and extracellular domain.
phys_region[0].name = "Extracellular domain"
phys_region[0].ptype = 1 #Extracellular Electrics
phys_region[0].num_IDs = 1
phys_region[0].ID[0] = 100
phys_region[1].name = "Intracellular domain"
phys_region[1].ptype = 0 #Intracellular Electrics
phys_region[1].num_IDs = 1
phys_region[1].ID[0] = 1

#-----------------------------------------------------------------
#Definition of ionic model and plugins (IMP) for each unique tagged
    region in the mesh
#-----------------------------------------------------------------
num_imp_regions = 1
imp_region[0].im = Courtemanche # Ionic model proposed by Courtemanche et
     al.
imp_region[0].im_param = "g_CaL-55%,g_K1+100%,blf_i_Kur-50%,g_to-65%,g_Ks
    +100%,maxI_pCa+50%,maxI_NaCa+60%,g_Kr*1.6" #Parameters modification as
     suggested by Loewe et al. to simulate the effect of persistent atrial
     fibrillation remodel
imp_region[0].name = "AFRemodeledTissue"
imp_region[0].num_IDs = 1
imp_region[0].ID = 1

#-----------------------------------------------------------------
#Definition of gregion for each unique tagged region in the mesh
#-----------------------------------------------------------------
num_gregions = 2
gregion[0].g_il = 0.118
gregion[0].g_it = 0.118
gregion[0].g_in = 0.118
gregion[0].g_el = 0.4262
gregion[0].g_et = 0.4262
gregion[0].g_en = 0.4262
gregion[0].num_IDs = 1
gregion[0].ID = 1
gregion[1].g_bath = 0.7 #Conductivity value for an isotropic bath
gregion[1].num_IDs = 1
gregion[1].ID  = 100

#--------------------
#Definition of stimuli
#--------------------
```

```
num_stim = 1
stimulus[0].stimtype = 0
stimulus[0].strength = 120.0
stimulus[0].duration = 2.0
stimulus[0].start = 0
stimulus[0].x0 = 2100
stimulus[0].xd  = 32000
stimulus[0].y0 = 2100
stimulus[0].yd = 310
stimulus[0].z0 = 2100
stimulus[0].zd = 4200
floating_ground = 1

#-------------------------------
#Definition of general parameters
#-------------------------------
tend = 5.0 #Total time for simulation
spacedt = 1.0 #Frequency of
timedt = 1.0 #Frequency of
gridout_i = 2 #Output intracellular grid for visualizing the results of
    the intracellular domain

#-------------------------------------------------
#Definition of solving problem and numerical methods
#-------------------------------------------------
bidomain = 1
dt = 10 #Time step in microseconds
parab_options_file = parab_solve_opts #File describing the solver and its
     options
ellip_options_file = ellip_solv_opts #File describing the solver and its
    options
```

6.1.2 Running with carputils

Setting up an in-silico experiment with carputils simplifies the process and allows to create more complex experiments than in the plain mode.

This is an example of a basic structure for an experiment defined using carputils:

```
#-------------------------------------------------
#Import basic Python modules
#-------------------------------------------------
import os
from datetime import date

#-------------------------------------------------
#Import carputils Python modules
#-------------------------------------------------
from carputils import settings
from carputils import tools
from carputils import mesh
from carputils import testing
```

```
#------------------------------------------------
#carputils parser function
#------------------------------------------------
def parser():
    parser = tools.standard_parser()
    group = parser.add_argument_group('experiment specific options')
    group.add_argument('--ionic_model',  default='tenTusscherPanfilov',
                       choices=['tenTusscherPanfilov','Courtemanche','
    HodgkinHuxley'],
                       help='pick ionic model (default is TT2). For a
    full list look inside opencarp installation folder /physics/limpet/
    models')
    group.add_argument('--duration',
                       type=float, default=1000.,
                       help='Duration of simulation (ms) (default is
    1000.)')
    group.add_argument('--stim-strength',
                       type=float, default=60.,
                       help='pick transmembrane current stimulus
    strength in [uA/cm^2] (default is 60.)')
    group.add_argument('--stim-dur',
                       type=float, default=2.,
                       help='pick transmembrane current stimulus
    duration in [ms] (default is 2.)')
    return parser

#------------------------------------------------
#carputils jobID name for output folder
#------------------------------------------------
def jobID(args):
    today = date.today()
    return '{}_basic_{}'.format(today.isoformat(), args.duration)

#------------------------------------------------
#carputils main function
#------------------------------------------------
@tools.carpexample(parser, jobID)
def run(args, job):
#------------------------------------------------
#Here goes your experiment
#------------------------------------------------
if __name__ == '__main__':
    run()
```

Pre and post processing

openCARP provides several standalone tools for pre- and post-processing of simulations.

7.1 mesher

mesher is a program for generating simple regular FE meshes. It can also produce element tag regions and fiber definitions during element generation.

- **size:** Set the size of the mesh in each axis direction.
- **resolution:** Set the resolution of the mesh in each axis direction. This is the node-to-node distance within planes. For tetrahedral meshes, the edge length along the diagonals will be longer ($\sqrt{2(res)^2}$), thus also the average edge length of the mesh will be higher than the resolution.
- **bath:** Set the bath size in each axis direction. Negative sizes denote a bath on both sides.
- **fibers:** Set fiber rotation.
- **mesh:** Set output mesh name.
- **regions:** Defines tags for regions in the mesh.

NOTE: The unit of the size and resolution command line parameters are in centimeters, while the unit of the mesh resolution is in micrometers.

You can query mesher by using:

```
> mesher +Help
```

7.2 igbutils

You can find igbutils inside the tools folder in the openCARP root folder. For all tools, help is output with the -h option.

7.2.1 igbhead

Perform operations on the headers of IGB files. Most commonly, it it used to print the header information but can also be usd to modify it. This utility can also be used to change the storage type of the file, or put an IGB header on to ASCII and binary data file.

7.2.2 igbapd

A simple utility to compute action potential durations.

7.2.3 igbops

This utility can perform basic math operations on one or two IGB files. There are preset functions as well as a parser for arbitrary functions. For example, one can compute the difference between two IGB files, find maxima over time, or apply a box filter.

7.2.4 igbextract

This is a tool to extract a hyperslab of data from an IGB file. The format of the output can be chosen between ASCII, binary or IGB.

Bench commands

In the section below you will find a detailed description of the commands available in bench.

Use the IMP library for single cell experiments. All times in ms; all voltages in mV; currents in uA/cm^2

--help	-h	Print help and exit
--detailed-help		Print help, including all details and hidden options, and exit
--full-help		Print help, including hidden options, and exit
--version	-V	Print version and exit

8.1 Mode: regstim

--numstim=INT		number of stimuli (default='1')
--stim-start=DOUBLE	-i	start of stimulation [ms] (default='1.')
--bcl=DOUBLE	-b	basic cycle length [ms] (default='1000.0')

8.2 Mode: neqstim

--stim-times=STRING	comma separated list of stim times [ms] (default='')
--DIA	interpret stim times as distolic intervals (default=off)

8.3 Mode: restitute

--restitute=STRING	restitution experiment (possible values="S1S2", "dyn", "S1S2f")
--res-file=STRING	definition file for restitution parameters (default='')
--res-trace	output ionic model trace (default=off)
--res-state-vector	save state vector (default=off)

8.4 Mode: info

--list-imps	list all available IMPs (default=off)
--plugin-outputs	list the outputs of available plugins (default=off)
--imp-info	print tunable parameters and state variables for particular IMPs (default=off)
--buildinfo	(deprecated now printed by default) print build information about this executable (default=off)

8.5 Mode: vclamp

--clamp-ini=DOUBLE		outside of clamp pulse, clamp Vm to this value (default='-80.0')
--clamp-file=STRING		clamp voltage to a signal read from a file (default='')

8.6 Group: stimtype

--stim-curr=FLOAT	-c	stimulation current [pA/pF]=[uA/cm^2] (default='60.0')
--stim-volt=FLOAT		stimulation voltage
--stim-file=STRING		use signal from a .trc file for current stimulus

8.7 Simulation control:

--duration=DOUBLE	-a	duration of simulation [ms]
--past-stim=DOUBLE		duration after last stim [ms] (default='1000.')
--dt=DOUBLE	-D	time step [ms] (default='.01')
--num=INT	-n	number of cells (default='1')
--ext-vm-update		update Vm externally (default=off)
--rseed=INT		random number seed (default='1')
--target=STRING		target to run the simulation on (default='auto')

8.8 Stimuli:

--stim-dur=DOUBLE	-T	duration of stimulus pulse [ms] (default='1.')
--stim-assign	-A	stimulus current assignment (default=off)
--stim-species=STRING		concentrations that should be affected by stimuli, e.g. 'Ki:Cli' (default='Ki')
--stim-ratios=STRING		proportions of stimlus current carried by each species, e.g. '0.7:0.3' (default='1.0')
--resistance=DOUBLE		coupling resistance [kOhm] (default='100.0')

8.9 Illumination (light stimulus ON/OFF):

--light-irrad=DOUBLE	unattenuated irradiance of illumination pulse (e.g. 0.24) (default='0.0')
--light-dur=DOUBLE	duration of illumination pulses (default='10.')
--light-numstim=INT	number of illumination pulses (0: no limit) (default='1')
--light-bcl=DOUBLE	basic cycle length of illumination (default='1000.')
--light-start=DOUBLE	start time for illumination pulse (default='10.')
--light-times=STRING	comma-separated list of stim times (overrides –light-vars for timing) (default='')
--light-file=STRING	overrides ALL –light- vars with a signal from a file (default='')

8.10 Ionic Models:

--imp=STRING	-I	IMP to use (default='DrouhardRoberge')
--imp-par=STRING	-p	params to modify IMP (default='')
--plug-in=STRING	-P	plugins to use, separate with ':' (default='')
--plug-par=STRING	-m	params to modify plug-ins, separate params with ',' and plugin params with ':' (default='')
--load-module=STRING		load a module for use with bench (implies –imp) (default='')

8.11 Voltage clamp pulse:

--clamp=DOUBLE	-l	clamp Vm to this value [mV] (default='0.0')
--clamp-dur=DOUBLE	-L	duration of Vm clamp pulse [ms] (default='0.0')
--clamp-start=DOUBLE		start time of Vm clamp pulse [ms] (default='10.0')

8.12 Clamps:

--clamp-SVs=STRING	colon separated list of state variable to clamp (default='')
--SV-clamp-files=STRING	: separated list of files from which to read state variables (default='')
--SV-I-trigger	apply SV clamps at each current stim (default=on)
--AP-clamp-file=STRING	action potential trace applied at each stimulus (default='')

8.13 Strain:

--strain=DOUBLE	-N	amount of strain to apply (default='0')
--strain-time=DOUBLE	-t	time to strain [ms] (default='200')
--strain-dur=FLOAT	-y	duration of strain (default=tonic) [ms]
--strain-rate=DOUBLE		time to apply/remove strain [ms] (default='2')

8.14 Output:

--dt-out=DOUBLE	-o	temporal output granularity [ms] (default='1.0')
--start-out=DOUBLE		start time of output [ms] (default='0.0')
--fout[=STRING]	-O	output to files (default='BENCH_REG')
--no-trace		do not output trace (default=off)
--trace-no=INT		number for trace file name (default='0')
--bin	-B	write binary files (default=off)
--imp-sv-dump=STRING	-u	sv dump list (default='')
--plug-sv-dump=STRING	-g	: separated sv dump list for plug-ins, lists separated by semicolon (default='')
--dump-lut	-d	dump lookup tables (default=off)
--APstatistics		compute AP statistics (default=off)
--validate	-v	output all SVs (default=off)

8.15 State saving/restoring:

--save-time=DOUBLE	-s	time at which to save binary state [ms] (default='0')
--save-file=STRING	-f	file in which to save binary state (default='a.sv')
--restore=STRING	-r	restore saved binary state file (default='a.sv')
--save-ini-file=STRING	-F	text file in which to save state of a single cell (default='singlecell.sv')
--save-ini-time=DOUBLE	-S	time at which to save single cell state [ms] (default='0.0')
--read-ini-file=STRING	-R	text file from which to read state of a single cell (default='singlecell.sv')
--SV-init=STRING		colon separated list of comma separated SV=initial_values

openCARP commands

openCARP provides a set of commands that allows you to configure your in-silico experiment. Here you can find a detailed explanation of all available parameters.

If you have carputils installed, you can use the script `carphelp` to find the openCARP parameters related to given keywords and their documentation.

runtime_models

openCARP offers the possibility to load custom ionic models during run-time without recompiling the simulator.

10.1 num_external_imp

Parameters

num_external_imp <Int>

Defines the number of models to read in from external libraries
Default: (Int)(0)

Required parameter:
external_imp

10.2 external_imp

Parameters

external_imp <RFile>

Defines external imp modules to read into openCARP. Multiple imps can be grouped using a comma seperated list which can be used here.
Default: (RFile)(“”)

runtime_fcn

In openCARP, you can create and use custom functions that can be loaded during run-time without recompiling the simulator.

11.1 rt_lib

Parameters

rt_lib <String>

Gives a colon separated list of runtime shared object libraries
Default: (String)(“”)

11.2 rt_lib_args

Parameters

rt_lib_args <String>

Gives colon separated list of parameters for objects
Default: (String)(“”)

(null)

12.1 num_trace

Parameters

num_trace <Int>

Number of nodes at which to gather trace info.
Default: (Int)(0)

Required parameter:
trace_node

12.2 trace_node

Parameters

trace_node <Int>

Nodes at which to gather trace information.
Default: (Int)(0)

12.3 tracedt

Parameters

tracedt <Double>

Time resolution to print out trace info.
Unit: ms
Default: (Double)(timedt)

Value must be greater than (Double)(dt/1000.)

12.4 dump_protocol

Parameters

dump_protocol <Short>

If set to 1, the overall stimulation protocol will be outputted as a trace file.
Default: (Short)(0)

Value must be between (Short)(0) and (Short)(1)

phie_recovery

In openCARP, you can quickly recover the extracellular potential from monodomain simulations using the parameters explained below.

13.1 phie_rec_ptf

Parameters

phie_rec_ptf <WFile>

Defines the basename for the phie recovery file. This file specifies the phie recovery points. Use the same convention and format as for the mesh points file (see the first chapters of the openCARP manual file format section). Provide the filename without extension.
Default: (WFile)(“”)

13.2 phie_recovery_file

Parameters

phie_recovery_file <WFile>

Defines the file name used to output recovered extracellular potentials. Output granularity is defined by 'spacedt'.
Default: (WFile)(“phie_recovery“)

13.3 phie_rec_meth

Parameters

phie_rec_meth <Int>

Defines the method for recovering phie with monodomain runs.
Default: (Int)(1)

Possible values are:

(Int)(3): Infrequent bidomain solve (recover phie only on phie grid, not on 'phie_rec_ptf') NOT IMPLEMENTED YET!
(Int)(2): Integrate over element-centered Im using integral solution of Poisson PDE
(Int)(1): Use FE mass and stiffness matrices
(Int)(0): Don't recover phie

13.4 dump_ecg_leads

Parameters

dump_ecg_leads <Int>

Dump recovered phie data from nodes stored in 'phie_rec_ptf'.
The stated node file may contain arbitrary nodes of interest.
In the case of an ecg recording, the spatial coordinates of LA, RA, LF, V1-V6 need to be provided.
Default: (Int)(strlen(phie_rec_ptf)>0?0:1)

Value must be between (Int)(0) and (Int)(1)

gregions

In this section, you can find all the information to define the conductivity of your model.

14.1 num_gregions

Parameters

num_gregions <Int>

Defines the total number of regions with respectively different conductivity settings.
Default: (Int)(1)

Value must be greater than (Int)(1)

Required parameters:
gregion
gregion[PrMelem1].num_IDs
gregion[PrMelem1].name
gregion[PrMelem1].g_bath
gregion[PrMelem1].g_en
gregion[PrMelem1].g_in
gregion[PrMelem1].g_et
gregion[PrMelem1].g_it
gregion[PrMelem1].g_el
gregion[PrMelem1].g_il
gregion[PrMelem1].g_mult
gregion[PrMelem1].ID

14.2 gRegion

Description Sets the conductivity for different regions. The array index allows enumeration of all gregions.

Parameters

ID <Int>

Specify a list of model regions (equivalent to mesh element tags) using this conductivity setting.
Default: (Int)(0)

Parameters (continued)

Required parameters:
gregion.num_IDs
gregion

num_IDs <Int>

Shows how many model regions use this conductivity setting
Default: (Int)(0)

Required parameter:
gregion

name <String>

Symbolic description of this conductive region (e.g. bath, cavity etc.)
Default: (String)(“”)

Required parameter:
gregion

g_bath <Double>

Defines the isotropic conductivity for non-myocardium domain, e.g blood.
The absence of anisotropy is assumed for this region. If anisotropy is required use a negative tag for the mesh elements in the .elem file and set fibre direction in the .lon file.
Unit: S/m
Default: (Double)(1.)

Value must be greater than (Double)(0.)

Required parameter:
gregion

Parameters (continued)

g_en <Double>

Defines the extracellular conductivity in the local sheet direction, i.e. perpendicular to the longitudinal fiber 'g_el' and transversal fiber directions 'g_et'.
Unit: S/m
Default: (Double)(0.236)

Value must be greater than (Double)(0.)

Required parameter:
gregion

g_in <Double>

Defines the intracellular conductivity in the local sheet direction, i.e. perpendicular to the longitudinal fiber 'g_il' and transversal fiber 'g_it' directions.
Unit: S/m
Default: (Double)(0.019)

Value must be greater than (Double)(0.)

Required parameter:
gregion

g_et <Double>

Defines the extracellular conductivity transverse to the fiber direction i.e. perpendicular to the longitudinal fiber 'g_il' and local sheet 'g_it' direction.
Unit: S/m
Default: (Double)(0.236)

Value must be greater than (Double)(0.)

Required parameter:
gregion

g_it <Double>

Defines the intracellular conductivity transverse to the fiber direction i.e. perpendicular to the longitudinal fiber 'g_il' and local sheet 'g_it' direction.

Parameters (continued)

Unit: S/m
Default: (Double)(0.019)

Value must be greater than (Double)(0.)

Required parameter:
gregion

g_el <Double>

Defines the extracellular conductivity along the fiber direction (longitudinal).
Unit: S/m
Default: (Double)(0.625)

Value must be greater than (Double)(0.)

Required parameter:
gregion

g_il <Double>

Defines the intracellular conductivity along the fiber direction (longitudinal).
Unit: S/m
Default: (Double)(0.174)

Value must be greater than (Double)(0.)

Required parameter:
gregion

g_mult <Float>

Defines the factor by which all conductivities in the region should be multiplied (after all other modifications are performed).
Default: (Float)(1.0)

Value must be greater than (Float)(0.)

Required parameter:
gregion

14.3 gi_scale_vec

Parameters

gi_scale_vec <RFile>

Path to element-wise vector with intra-cellular conductivity scaling
Default: (RFile)(“”)

14.4 ge_scale_vec

Parameters

ge_scale_vec <RFile>

Path to element-wise vector with extra-cellular conductivity scaling
Default: (RFile)(“”)

gvecs

In this section, you can find all the information to define global variables in your simulation.

15.1 num_gvecs

Parameters

num_gvecs <Int>

Specify the number of global state variable vector definitions.
Default: (Int)(0)

Required parameters:
gvec
gvec[PrMelem1].bogus
gvec[PrMelem1].imp
gvec[PrMelem1].units
gvec[PrMelem1].name
gvec[PrMelem1].ID

15.2 GVecs

Description Gives the global state variable vectors of each model region

Parameters

bogus <Float>

Set to 1 if the state variable is not in a region.
Default: (Float)(0.)

Required parameter:
gvec

imp <String>

If the state variable is defined in a plugin, specify the plugin name here.

Parameters (continued)

Default: (String)(“”)

Required parameter:
gvec

units <String>

Units of the state variables to be written in the header of the ouput igb file.
Default: (String)(“”)

Required parameter:
gvec

ID <String>

Defines the list of state variable names in a region, which are combined into a global vector. For example gvec[0].ID[0] = ’Cai’
Default: (String)([[Father]].name)

Required parameters:
gvec.name
num_imp_regions
gvec

name <WFile>

Defines the name of the output file for the global state variable.
Default: (WFile)(“sv“)

Required parameter:
gvec

ionic_parameter_regions

In this section, you can find all the information to define more or more ionic models for your simulation.

16.1 num_imp_regions

Parameters

num_imp_regions <Int>

Number of different region definitions.
Default: (Int)(1)

Value must be greater than (Int)(1)

Required parameters:
imp_region
imp_region[PrMelem1].cellSurfVolRatio
imp_region[PrMelem1].volFrac
imp_region[PrMelem1].plug_param
imp_region[PrMelem1].plug_sv_dumps
imp_region[PrMelem1].im_sv_dumps
imp_region[PrMelem1].num_IDs
imp_region[PrMelem1].name
imp_region[PrMelem1].plugins
imp_region[PrMelem1].im_sv_init
imp_region[PrMelem1].im
imp_region[PrMelem1].im_param
imp_region[PrMelem1].ID
gvec[PrMelem1].ID

16.2 IMPregion

Description Sets the ionic model for different regions. The array index allows enumeration of all imp_regions.

Parameters

cellSurfVolRatio <Float>

Defines the default single cell surface-to-volume-ratio used if it is not specified by the ionic model plugin (IMP).
Unit: umˆ-1
Default: (Float)(0.14)

Value must be greater than (Float)(0.)

Required parameter:
imp_region

volFrac <Float>

Defines the portion of the volume occupied by cells.
Default: (Float)(1.)

Value must be between (Float)(0.) and (Float)(1.)

Required parameter:
imp_region

plug_param <String>

User can modify the values for various ionic model parameters.
Every PARAMETER=VALUE modification needs to be separated with a comma.
Default: (String)(“”)

Required parameter:
imp_region

plug_sv_dumps <String>

This produces a colon separated lists of state variables of a plugin to dump (use bench –imp=XXX –plug-in=YYY –imp-info to get a list of SVs for imp XXX and plug-in YYY)
Default: (String)(“”)

Parameters (continued)

Required parameter:
imp_region

im_sv_dumps <String>

Specify a comma separated list of state variables of an ionic model to be dumped into the simulation folder.
Use bench –imp=XXX –imp-info to get a list of SVs for imp XXX.
Default: (String)(“”)

Required parameter:
imp_region

im_param <String>

Specify a comma separated list of changes from default values, e.g. 'APDshorten·4,Gks·1.29,...'.
Default: (String)(strcmp([[Father]].im,TO_STRING(DiFranN
TO_STRING():TO_STRING(G_Na·3))

Required parameters:
imp_region.im
imp_region

ID <Int>

Array of element tags associated with this model region.
Default: (Int)(0)

Required parameters:
imp_region.num_IDs
imp_region

Parameters (continued)

num_IDs <Int>

Number of elements tags listed in 'imp_region[].ID[]'.
Default: (Int)(0)

Required parameter:
imp_region

name <String>

Defines the symbolic name of region. This helps with structuring extensive parameter lists and makes them more human readable.
Default: (String)(strncmp(TO_STRING([[Father]]),TO_STR
TO_STRING(Myocardium):TO_STRING(Purkinje))

Required parameter:
imp_region

plugins <String>

Specify a colon separated list of plug-ins to use with the ionic model.
Query for available plug-ins by calling 'bench –plugin-outputs'.
Default: (String)(“”)

Required parameter:
imp_region

im_sv_init <RFile>

Name of the file containing the initial of state variable values.
Default: (RFile)(“”)

Required parameter:
imp_region

Parameters (continued)

im <String>

Defines the ionic model to be used in the simulation.
Query for available ionic models by calling 'bench' with the command list-imps.
Default: (String)(TO_STRING(LuoRudy91))

Required parameter:
imp_region

16.3 dump_imp_region

Parameters

dump_imp_region <Int>

flag controling whether to dump nodal ionic region indices. output file name is 'imp_region.dat'
Default: (Int)(0)

Possible values are:

(Int)(1): dump nodal ionic region indices
(Int)(0): dont dump

random

openCARP allows modifying of the conductivity based on the element index.

17.1 rseed

Parameters

rseed <Int>

Defines the seed for the random generator
Default: (Int)(1)

17.2 fluct

Parameters

fluct <Float>

Defines the conductivity fluctuation in percent.
Unit: %
Default: (Float)(0.)

Value must be between (Float)(0.0) and (Float)(100.0)

adjust_state_variables

openCARP offers the possibility to modify the ionic models on a nodal basis.

18.1 num_adjustments

Parameters

num_adjustments <Int>

Size of the adjustment array.
Default: (Int)(0)

Value must be greater than (Int)(0)

Required parameters:
adjustment
adjustment[PrMelem1].dump
adjustment[PrMelem1].file
adjustment[PrMelem1].variable

18.2 IMPVariableAdjustment

Description file of adjustments

Parameters

dump <Short>

Dump nodal adjustments for state variables for display on intra grid
Default: (Short)(0)

Value must be between (Short)(0) and (Short)(1)

Required parameter:
adjustment

file <RFile>

Defines the filename filled with nodal adjustments for the state variable
Default: (RFile)(“”)

Parameters (continued)

Required parameter:
adjustment

variable <String>

Defines the name of the variable that should be adjusted on initialization.
The variable name should be defined as an external variable like 'Lambda', or an ionic model variable like 'LR1.tau_f_factor' in the cellmodel file.
Default: (String)(“”)

Required parameter:
adjustment

meshing

openCARP offers the possibility to pre-process your mesh. In this section, you can find the different parameters that can help you to, for example, retag your mesh on-the-fly.

19.1 mesh_statistics

Parameters

mesh_statistics <Flag>

Compute mesh statistic parameters (only edge lengths at this point).
Default: (Flag)(PrMFALSE)

19.2 meshname

Parameters

meshname <String>

Defines the basename for mesh files
Default: (String)(“project“)

19.3 orthoname

Parameters

orthoname <String>

Basename for lon file holding orthotropy data

!!!! This parameter is unsupported in openCARP, but kept for compatibility with CARPentry !!!!
Default: (String)(““)

Parameters (continued)

19.4 orthogonalize

Parameters

orthogonalize <Short>

Toggle automatic fiber orthogonalization
Default: (Short)(0)

Value must be between (Short)(0) and (Short)(1)

19.5 orthoname_output

Parameters

orthoname_output <WFile>

Basename for .lon file holding orthotropy data that is used for output purposed, e.g. strain in this fiber direction.

!!!! This parameter is unsupported in openCARP, but kept for compatibility with CARPentry !!!!
Default: (WFile)(“”)

19.6 meshformat

Parameters

meshformat <Short>

Mesh format ID
Default: (Short)(2)

Parameters (continued)

Possible values are:

(Short)(2): Auto: Use binary if possible, else use text.
(Short)(1): openCARP binary format
(Short)(0): openCARP text format

19.7 numtagreg

Parameters

numtagreg <Int>

Defines the number of regions to retag
Default: (Int)(0)

Value must be greater than (Int)(0)

Required parameters:
tagreg
tagreg[PrMelem1].radius
tagreg[PrMelem1].p1
tagreg[PrMelem1].p0
tagreg[PrMelem1].elemfile
tagreg[PrMelem1].no_elem_split
tagreg[PrMelem1].type
tagreg[PrMelem1].name
tagreg[PrMelem1].tag

19.8 TagRegion

Description Defines the regions to which to assign new tags

Parameters

radius <Float>

Used to describe the profile chosen in tagregion.type in more detail.
- If sphere was chosen, this value is the radius of the sphere.
- If box was chosen, this value is not necessary/ignored. Use tagregion.p0 & tagregion.p1 instead.
- If cylinder was chosen, this value is the radius of the cylinder.
Unit: micrometers
Default: (Float)(100.)

Required parameter:
tagreg

p1 <Float>

Used to describe the profile chosen in tagregion.type in more detail.
- If sphere was chosen, this value is not necessary/ignored. Use tagregion.radius instead.
- If box was chosen, this value describes the upper right corner of the box.
- If cylinder was chosen, this value describes the center of the cylinders' top.
- For sphere & cylinder the additional parameter tagregion.radius is needed.
- For box & cylinder the additional parameter tagregion.p0 is needed.
Unit: micrometers
Default: (Float)(0.)

Required parameter:
tagreg

Parameters (continued)

p0 <Float>

Used to describe the profile chosen in tagregion.type in more detail.
- If sphere was chosen, this value describes the center of the sphere.
- If box was chosen, this value describes the lower left corner of the box.
- If cylinder was chosen, this value describes the center of the cylinders' base.
- For sphere & cylinder the additional parameter tagregion.radius is needed.
- For box & cylinder the additional parameter tagregion.p1 is needed.
Unit: micrometers
Default: (Float)(0.)(Float)(0.)(Float)(0.)

Required parameter:
tagreg

elemfile <RFile>

Name of file with list of elements to be assigned to this region. The file needs the extension .regele with the format being the number of elements followed by one element number per line
Default: (RFile)(“”)

Required parameter:
tagreg

no_elem_split <Short>

Defines whether a mesh element needs to be fully enclosed by the tagreg.type definition to become part of this region.
1 = all nodes of each element need to be fully contained inside tagreg.type
0 = a single node of an element within tagreg.type is sufficient to become a member of this region

Parameters (continued)

Default: (Short)(0)

Value must be between (Short)(0) and (Short)(1)

Required parameter:
tagreg

type <Int>

Defines profile of region. 1-3 are predefined shapes while 4 can be used to create own region profile.
Default: (Int)(1)

Possible values are:

(Int)(4): element list
(Int)(3): cylinder
(Int)(2): block
(Int)(1): sphere

Required parameter:
tagreg

name <String>

Label for a region in the mesh. This helps with structuring extensive parameter lists and makes them more human readable.
Default: (String)(“”)

Required parameter:
tagreg

tag <Int>

New tag for a region in the mesh. Each element in this region is re-assigned this tag. In general, each element in a mesh is designated as part of a region. Electrical or mechanical properties are assigned based on region definitions.
Default: (Int)(321)

Parameters (continued)

Required parameter:
tagreg

19.9 retagfile

Parameters

retagfile <WFile>

Defines an output file storing the element labels after applying all 'dynamic' tagreg choices to the input mesh.
Default: (WFile)(“”)

postprocessing

In this section, you can find the parameters that define the different available experiments in openCARP.

20.1 ppID

Parameters

ppID <String>

Defines the name of the output directory in post processing mode.
- It comes into play when choosing 'experiment 4'.
- If it is specified, it must not already exist.
- A folder with the default name will be created, if it is not specified.
Default: (String)("POSTPROC_DIR")

20.2 experiment

Parameters

experiment <Short>

Defines how the simulation will be solved and what will be outputted
Default: (Short)(0)

Possible values are:

(Short)(4): Post process only
(Short)(3): Build model only
(Short)(2): Laplace solve
(Short)(1): Output FEM matrices
(Short)(0): NORMAL RUN

20.3 post_processing_opts

Parameters

post_processing_opts <Short>

Post-processing Options, add up option numbers to use multiple options
Default: (Short)(0)

Possible values are:

(Short)(1): Recover phie
(Short)(0): No post-processing done.

solution_methods

openCARP offers the possibility to solve monodomain, pseudo-bidomain, and bidomain. In this section, you can find all the parameters related to the solving method and the solver parameters.

21.1 bidomain

Parameters

bidomain <Short>

Defines if the simulation is solved using the monodomain, bidomain or pseudo-bidomain approach.
- Monodomain model (less costly). Dervied from bidomain under the assumption that intracellular and extracellular tensors are related.
- Bidomain model solves for both the intra- & extracellular space.
- Pseudo-bidomain model (monodomain model with adjustments to account for bath loading effects).
Default: (Short)(0)

Possible values are:

(Short)(2): Pseudo-bidomain
(Short)(1): Bidomain
(Short)(0): Monodomain

21.2 pstrat

Parameters

pstrat <Short>

Defines the partitioning strategy.
Default: (Short)(2)

Possible values are:

Parameters (continued)

(Short)(2): KDtree partitioning
(Short)(1): Parmetis partitioning
(Short)(0): Linear partitioning

21.3 pstrat_i

Parameters

pstrat_i <Short>

Defines the partitioning strategy for the intracellular grid

!!!! This parameter is unsupported in openCARP, but kept for compatibility with CARPentry !!!!
Default: (Short)(1)

Possible values are:

(Short)(2): KDtree partitioning
(Short)(1): Parmetis partitioning
(Short)(0): Linear partitioning

21.4 pstrat_imbalance

Parameters

pstrat_imbalance <Float>

Amount of imbalance to tolerate in parmetis partitioning
Default: (Float)(1.0001)

21.5 ellip_solve

Parameters

ellip_solve <Short>

Defines the solving method for elliptic problems.
- Direct is typically more accurate but memory consuming.
- Iterative methods are more suitable for large problems.
Default: (Short)(0)

Possible values are:

(Short)(1): Iterative
(Short)(0): Direct if available

21.6 flavor

Parameters

flavor <String>

String defining the backend to be used
Default: (String)(“petsc“)

21.7 ginkgo_exec

Parameters

ginkgo_exec <String>

Defines the hardware backend used by Ginkgo.

Default: (String)(“ref“)

Possible values are:

Parameters (continued)

(String)(“dpcpp“): Dpcpp executor for execution on Intel GPUs
(String)(“hip“): Hip executor for execution on AMD GPUs
(String)(“cuda“): Cuda executor for execution on NVIDIA GPUs
(String)(“omp“): OpenMP parallelized CPU execution
(String)(“ref“): Reference Executor for sequential CPU execution

21.8 device_id

Parameters

device_id <Int>

Device ID used for the Ginkgo backend.
Default: (Int)(0)

Value must be greater than (Int)(0)

21.9 ellip_options_file

Parameters

ellip_options_file <RFile>

File containing PETSc or Ginkgo options for elliptic solver.
For all available PETSc options refer to the PETSc documentation.
Default: (RFile)(““)

21.10 floating_ground

Parameters

floating_ground <Short>

Enforces average extracellular potential to be zero.

!!!! This parameter is unsupported in openCARP, but kept for compatibility with CARPentry !!!!
Default: (Short)(0)

Value must be between (Short)(0) and (Short)(1)

21.11 floating_ground_refnode

Parameters

floating_ground_refnode <Int>

Reference node which is clamped to zero for elliptic solve prior to shifting phie average to zero.

!!!! This parameter is unsupported in openCARP, but kept for compatibility with CARPentry !!!!
Default: (Int)(0)

Value must be greater than (Int)(0)

21.12 parab_solve

Parameters

parab_solve <Short>

Defines the solution method for the parabolic problem
Default: (Short)(1)

Possible values are:

(Short)(2): 2nd order dt

Parameters (continued)

(Short)(1): Crank-Nicolson
(Short)(0): Explicit

21.13 theta

Parameters

theta <Float>

Defines the weight given to solution at t(n+1) when using Crank-Nicolson (theta) method
Default: (Float)(0.5)

Value must be between (Float)(0.1) and (Float)(0.99)

21.14 parab_options_file

Parameters

parab_options_file <RFile>

File containing PETSc or Ginkgo options for parabolic solver
Default: (RFile)(“”)

21.15 bidm_eqv_mono

Parameters

bidm_eqv_mono <Short>

Use monodomain conductivities that are equivalent to the bidomain.
Default: (Short)(1)

Parameters (continued)

Possible values are:

(Short)(1): Use harmonic mean tensor
(Short)(0): use intracellular tensor

21.16 stimactivedelay

Parameters

stimactivedelay <Float>

Time after stimulus ends to continue FEM solve.
Unit: ms
Default: (Float)(0.)

21.17 vm_per_phie

Parameters

vm_per_phie <Short>

Defines the number of Vm solves per phi_e solve
Default: (Short)(1)

Value must be between (Short)(1) and (Short)(1000)

21.18 par_fac

Parameters

par_fac <Short>

Defines the number of parabolic solves per global dt

Parameters (continued)

Default: (Short)(1)

Value must be between (Short)(1) and (Short)(100)

21.19 ode_fac

Parameters

ode_fac <Short>

Defines the number of ode solves per global dt
Default: (Short)(1)

Value must be between (Short)(1) and (Short)(10)

21.20 extracell_monodomain_stim

Parameters

extracell_monodomain_stim <Flag>

When bidomain mode is turned off, i.e. '-bidomain=0', set phi_e to be utterly determined by the extracellular stimuli.
Default: (Flag)(0)

21.21 cg_tol_ellip

Parameters

cg_tol_ellip <Double>

conjugate gradient solver tolerance for elliptic problem
Default: (Double)(1.0e-8)

Parameters (continued)

21.22 cg_norm_ellip

Parameters

cg_norm_ellip <Short>

Pick a norm for checking convergence of elliptic solve for PETSc solvers
Default: (Short)(0)

Possible values are:

(Short)(3): combined tolerance, using both 0 and 2, iteration stops if either 0 or 2 are met
(Short)(2): relative tolerance
(Short)(1): absolute tolerance using L2 of unpreconditioned residual (not always possible, defaults to 0 then)
(Short)(0): absolute tolerance using L2 of preconditioned residual (energy norm)

21.23 cg_maxit_ellip

Parameters

cg_maxit_ellip <Int>

Defines the maximum number of iterations for iterative solver of elliptic PDE.
Default: (Int)(500)

Value must be between (Int)(0) and (Int)(10000)

21.24 ellip_use_pt

Parameters

ellip_use_pt <Short>

Choose solvers for the elliptic PDE from:
- PETSc, then set to 0, or
- PT, then set to 1.
Default: (Short)(0)

Value must be between (Short)(0) and (Short)(1)

21.25 cg_tol_parab

Parameters

cg_tol_parab <Double>

conjugate gradient solver tolerance for parabolic problem
Default: (Double)(1.0e-8)

21.26 cg_norm_parab

Parameters

cg_norm_parab <Short>

Pick a norm for checking convergence of parabolic solve (PETSc solvers only, ignored with PT)
Default: (Short)(0)

Possible values are:

(Short)(3): combined tolerance, using both 0 and 2, iteration stops if either 0 or 2 are met
(Short)(2): relative tolerance

Parameters (continued)

(Short)(1): absolute tolerance using L2 of unpreconditioned residual (not always possible, defaults to 0 then)
(Short)(0): absolute tolerance using L2 of preconditioned residual (energy norm)

21.27 cg_maxit_parab

Parameters

cg_maxit_parab <Int>

maximum number of iterations for iterative solver of parabolic PDE.

Default: (Int)(100)

Value must be between (Int)(0) and (Int)(1000)

21.28 parab_use_pt

Parameters

parab_use_pt <Short>

Choose solvers for the parabolic PDE from:
- PETSc, then set to 0, or
- PT, then set to 1.
Default: (Short)(0)

Value must be between (Short)(0) and (Short)(1)

21.29 cg_precond

Parameters

cg_precond <Int>

Defines the preconditioning method to ensure fast convergence of the conjugate gradient method.
Default: (Int)(2)

Possible values are:

(Int)(3): System reduction
(Int)(2): Incomplete Cholesky
(Int)(1): diagonal
(Int)(0): none

fe_methods

You can select different methods to solve the finite element method.

22.1 mat_entries_per_row

Parameters

mat_entries_per_row <Short>

Assumed average number of entries per PETSc matrix row timportant for memory preallocation.

!!!! This parameter is unsupported in openCARP, but kept for compatibility with CARPentry !!!!
Default: (Short)(27)

22.2 mass_lumping

Parameters

mass_lumping <Short>

toggles mass matrix lumping
Default: (Short)(1)

Possible values are:

(Short)(1): Lump mass matrix
(Short)(0): Use full mass matrix

22.3 operator_splitting

Parameters

operator_splitting <Short>

toggles operator splitting
Default: (Short)(1)

Possible values are:

(Short)(1): Use operator splitting
(Short)(0): Don't use operator splitting

stimulation

openCARP offers you the possibility to define one or more stimuli. Different stimuli parameters can specify in the simulation (for example, strength, duration, BCL, and more).

23.1 num_stim

Parameters

num_stim <Int>

Defines the number of stimuli.
Default: (Int)(2)

Value must be greater than (Int)(0)

Required parameters:
stim
stimulus
stim[PrMelem1].crct
stim[PrMelem1].elec
stim[PrMelem1].ptcl
stim[PrMelem1].pulse
stim[PrMelem1].name
stimulus[PrMelem1].vtx_fcn
stimulus[PrMelem1].data_file
stimulus[PrMelem1].pulse_file
stimulus[PrMelem1].total_current
stimulus[PrMelem1].balance
stimulus[PrMelem1].stimtype
stimulus[PrMelem1].bias
stimulus[PrMelem1].tau_plateau
stimulus[PrMelem1].tau_edge
stimulus[PrMelem1].s2
stimulus[PrMelem1].strength
stimulus[PrMelem1].d1
stimulus[PrMelem1].start
stimulus[PrMelem1].geometry
stimulus[PrMelem1].ctr_def
stimulus[PrMelem1].zd
stimulus[PrMelem1].yd
stimulus[PrMelem1].xd
stimulus[PrMelem1].z0

Parameters (continued)

stimulus[PrMelem1].y0
stimulus[PrMelem1].x0
stimulus[PrMelem1].dump_vtx_file
stimulus[PrMelem1].vtx_file
stimulus[PrMelem1].name
stim[PrMelem1].crct.total_current
stim[PrMelem1].crct.balance
stim[PrMelem1].crct.type
stim[PrMelem1].elec.dump_vtx_file
stim[PrMelem1].elec.geom_type
stim[PrMelem1].elec.radius
stim[PrMelem1].elec.p1
stim[PrMelem1].elec.p0
stim[PrMelem1].elec.vtx_fcn
stim[PrMelem1].elec.vtx_file
stim[PrMelem1].elec.domain
stim[PrMelem1].elec.geomID
stim[PrMelem1].ptcl.stimlist
stim[PrMelem1].ptcl.npls
stim[PrMelem1].ptcl.start
stim[PrMelem1].ptcl.name
stim[PrMelem1].pulse.bias
stim[PrMelem1].pulse.tau_plateau
stim[PrMelem1].pulse.tau_edge
stim[PrMelem1].pulse.s2
stim[PrMelem1].pulse.tilt_ampl
stim[PrMelem1].pulse.tilt_time
stim[PrMelem1].pulse.strength
stim[PrMelem1].pulse.file
stim[PrMelem1].pulse.shape
stim[PrMelem1].pulse.name
stimulus[PrMelem1].duration
stim[PrMelem1].ptcl.duration
stimulus[PrMelem1].bcl
stim[PrMelem1].ptcl.bcl
stimulus[PrMelem1].npls

23.2 Stimulus

Description Array of stimuli

Parameters

vtx_fcn <Short>

Set true to specify stimulation strengths on a nodal basis. The specific values need to be provided in the stimulus.vtx_file. For file format specifications check the first chapters of the openCARP manual.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Default: (Short)(0)

Required parameter:
stimulus

data_file <RFile>

Stimulus dependent auxiliary data. Used for presribed takeoff and prescribed phie.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Default: (RFile)(“”)

Required parameter:
stimulus

pulse_file <RFile>

Reads in pulse definition from an external file.
For file format specifications check the first chapters of the openCARP manual.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Default: (RFile)(“”)

Required parameter:
stimulus

Parameters (continued)

total_current <Short>

Treat strengths as total current (uA) and not density

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Default: (Short)(0)

Value must be between (Short)(0) and (Short)(1)

Required parameter:
stimulus

balance <Int>

Defines whether the electrode is balancing another electrode. The balance value is interpreted as the electrode index to balance. The waveform is mirrored, but with opposite polarity.
If the balance value is -1, no balancing is applied.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Default: (Int)(-1)

Value must be between (Int)(-1) and (Int)(num_stim-1)
Required parameters:
num_stim
stimulus

stimtype <Short>

Defines the stimulus type. Closed loop stimuli return to ground (0 mV) when expired.
Open loop stimuli are removed entirely when expired.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Default: (Short)(0)

Parameters (continued)

Possible values are:

(Short)(9): Vm clamp (use mV)
(Short)(6): illumination (use mW/mmˆ2)
(Short)(5): extracellular voltage (open loop, use mV)
(Short)(4): intracellular current
(Short)(3): extracellular ground (enforce 0 mV)
(Short)(2): extracellular voltage (closed loop, use mV)
(Short)(1): extracellular current (use uA/cmˆ3)
(Short)(0): transmembrane current (use uA/cmˆ2)

Required parameter:
stimulus

bias <Float>

Defines a constant term which is added to the stimuluspulse.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Default: (Float)(0.0)

Required parameter:
stimulus

Parameters (continued)

tau_plateau <Float>

Defines a time constant governing plateau of pulse (>10^5 is infinite)
A larger tau_plateau will result in a longer plateauphase.
Other important parameters for stimulus form definition are: stimulus.tau_edge, stimulus.tau_plateau, stimulus.strength, stimulus.duration

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Unit: ms
Default: (Float)(1000.)

Value must be between (Float)(0.1) and (Float)(1000000.)
Required parameter:
stimulus

tau_edge <Float>

Defines the time constant governing leading and trailing edge of pulse. The formulation used resembles the equation: Amp · (1 - exp(-t/tau_edge)) · exp(-t/tau_plateau).
A larger tau_edge will result in a fast drop, while a smaller tau_edge will have a longer exponentialy decreasing flank.
Other important parameters for stimulus form definition are: stimulus.tau_edge, stimulus.tau_plateau, stimulus.strength, stimulus.duration

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Unit: ms
Default: (Float)(0.01)

Value must be between (Float)(0.) and (Float)(1000.)
Required parameter:
stimulus

Parameters (continued)

s2 <Float>

This value is only used for defining a biphasic pulse.
This value is a relative value and defines the stimulus strength of the trailing pulse (after zero crossing) relative to leading pulse (from start to zero crossing).
Giving the value 0 will result in a flip of polarity
Other important parameters for stimulus form definition are: stimulus.tau_edge, stimulus.tau_plateau, stimulus.strength, stimulus.duration

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Default: (Float)(0.)

Value must be between (Float)(0.) and (Float)(10.)
Required parameter:
stimulus

strength <Float>

Defines strength of prescribed stimulation. Strength is defined as amplitude of signal. If dealing with currents it is defined as uA/volume. If dealing with voltages it is defined as mV.
To create a stimulation the minimum requirement is a given stimulus.duration and stimulus.strength. This creates a monophasic stimulus. For biphasic stimuli check stimulus.s2 and stimulus.d1.
For more advanced pulsesignals introduce them using a pulse file with stimulus.pulse_file

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Unit: uA/cm^2(2D current), uA/cm^3(3D current), or mV
Default: (Float)(0.)

Parameters (continued)

Required parameter:
stimulus

d1 <Float>

This parameter is used to introduce biphasic stimulation pulses. It specifies the duration of the first part of the pulse relative to the duration of the entire pulse (1.0 = monophasic).
Further parameters to define biphasic stimuli are: stimulus.strength, stimulus.s2, stimulus.tau_edge and stimulus.tau_plateau

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Default: (Float)(1.0)

Value must be between (Float)(0.) and (Float)(1.0)
Required parameter:
stimulus

duration <Float>

Defines the duration of one a single stimulation event.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Unit: ms
Default: (Float)(tend-[[Father]].start)

Value must be between (Float)(0.) and (Float)(tend-stimulus.start)
Required parameters:
stimulus.start
tend
stimulus

Parameters (continued)

npls <Int>

Defines the number of pulses in the stimulation protocol. The period of pulses can be set with bcl in ms.
If set to 0, the stimulus will have a single instance.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Default: (Int)(tend > 0 ? 1 : 0)

Value must be between (Int)(0) and (Int)(tend > 0 ? 1+tend/stimulus.bcl : 0)
Required parameters:
tend
stimulus.bcl
stimulus

bcl <Float>

Defines the basic cycle length for repetitive stimulation.
Duration must be smaller than bcl, as it specifies the duration of a single stimulation event.
The full protocol duration is npls · bcl. This is derived automatically from the user input.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Unit: ms
Default: (Float)(tend-[[Father]].start)

Value must be between (Float)(stimulus.duration) and (Float)(tend-stimulus.start)
Required parameters:
stimulus.start
tend
stimulus.duration
stimulus

Parameters (continued)

start <Double>

Defines the start time when the stimulationpulse is introduced

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Unit: ms
Default: (Double)(0.)

Value must be between (Double)(0.) and (Double)(tend)
Required parameters:
tend
stimulus

geometry <Int>

Refers to a region ID to be used as geometry definition for the stimulus electrode

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Default: (Int)(-1)

Required parameter:
stimulus

ctr_def <Flag>

setting this to 1 yields limits in the form of [x0-xd/2,x0+xd/2] for x,y and z. Setting to 0 yields [x0,x0+xd] for x,y and z.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Default: (Flag)(0)

Required parameter:
stimulus

Parameters (continued)

zd <Float>

z dimension of electrode volume. This will give the limits [z0,z0+zd]. To change this to [z0-zd/2,z0+zd/2] use stimulus.ctr_def

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Unit: um
Default: (Float)(cell_length)

Value must be greater than (Float)(0.)

Required parameters:
cell_length
stimulus

yd <Float>

y dimension of electrode volume. This will give the limits [y0,y0+yd]. To change this to [y0-yd/2,y0+yd/2] use stimulus.ctr_def

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Unit: um
Default: (Float)(cell_length)

Value must be greater than (Float)(0.)

Required parameters:
cell_length
stimulus

xd <Float>

x dimension of electrode volume. This will give the limits [x0,x0+xd]. To change this to [x0-xd/2,x0+xd/2] use stimulus.ctr_def

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Unit: um

Parameters (continued)

Default: (Float)(cell_length)

Value must be greater than (Float)(0.)

Required parameters:
cell_length
stimulus

z0 <Float>

Lower z ordinate of electrode volume (zd defines the spatial electrode extension in z)

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Unit: um
Default: (Float)(0.)

Required parameter:
stimulus

y0 <Float>

Lower y ordinate of electrode volume (yd defines the spatial electrode extension in y)

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Unit: um
Default: (Float)(0.)

Required parameter:
stimulus

x0 <Float>

Lower x ordinate of electrode volume (xd defines the spatial electrode extension in x)

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!

Parameters (continued)

Unit: um
Default: (Float)(0.)

Required parameter:
stimulus

dump_vtx_file <Short>

For volume based electrode definitions, vertices of this electrode are dumped to a file. The output file name is electrode_num.stim

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Default: (Short)(0)

Value must be between (Short)(0) and (Short)(1)

Required parameter:
stimulus

vtx_file <RFile>

File name allowing a vertex based electrode definition. Using a non-empty string switches to vertex-based definition and ignores other settings. For file format specifications check the first chapters of the openCARP manual.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Default: (RFile)(“”)

Required parameter:
stimulus

Parameters (continued)

name <String>

Definition of the label for a single electrode.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!!
Default: (String)(“”)

Required parameter:
stimulus

23.3 Stim

Description Definition of stimuli

Parameters

name <String>

“Definition of the label for a single electrode“

Required parameter:
stim

23.3.1 crct

Description “Definition of setup and wiring of stimulation circuit“

Parameters

total_current <Short>

Treat strengths as total current (uA)
Default: (Short)(0)

Value must be between (Short)(0) and (Short)(1)

Required parameter:
stim

Parameters (continued)

balance <Int>

Defines whether the electrode is balancing another electrode. The balance value is interpreted as the electrode index to balance. The waveform is mirrored, but with opposite polarity.
If the balance value is -1, no balancing is applied.
Default: (Int)(-1)

Value must be between (Int)(-1) and (Int)(num_stim-1)
Required parameters:
num_stim
stim

type <Short>

Defines the physics of the actual source used for stimulation and in some cases also the wiring of the electric stimulation circuit.

Default: (Short)(0)

Possible values are:

(Short)(9): Vm clamp (use mV)
(Short)(6): illumination (use mW/mmˆ2)
(Short)(5): extracellular voltage (open loop, use mV)
(Short)(4): intracellular current
(Short)(3): extracellular ground (enforce 0 mV)
(Short)(2): extracellular voltage (closed loop, use mV)
(Short)(1): extracellular current (use uA/cmˆ3)
(Short)(0): transmembrane current (use uA/cmˆ2)

Required parameter:
stim

23.3.2 elec

Description "Definition of electrode geometry"

Parameters

dump_vtx_file <Short>

For volume based electrode definitions, vertices of this electrode are dumped to a file.
Default: (Short)(0)

Value must be between (Short)(0) and (Short)(1)

Required parameter:
stim

geom_type <Int>

This value defines the geometry used for the electrode. For now only predefined geometries are supported.
Default: (Int)(2)

Possible values are:

(Int)(5): vertex list, not implemented
(Int)(4): element list (volume or surface), not implemented
(Int)(3): cylinder
(Int)(2): block
(Int)(1): sphere

Required parameter:
stim

Parameters (continued)

radius <Float>

Used to describe the profile chosen in electrode.geom_type.
- If sphere was chosen, this value is sets the radius of the sphere.
- If box was chosen, this value is not necessary/ignored. Use TagRegion.p1 & TagRegion.p0 instead.
- If cylinder was chosen, this value is sets the radius of the cylinder.
Unit: micrometers
Default: (Float)(100.)

Required parameter:
stim

p1 <Float>

Used to describe the profile chosen in electrode.geom_type.
- If sphere was chosen, this value is not necessary/ignored. Use TagRegion.radius instead.
- If box was chosen, this value describes the upper right corner of box.
- If cylinder was chosen, this value describes the end of the cylinders main axis.
- For sphere & cylinder the additional parameter electrode.radius is needed.
- For box & cylinder the additional parameter electrode.p0 is needed.
Unit: micrometers
Default: (Float)(0.)

Required parameter:
stim

Parameters (continued)

p0 <Float>

Used to describe the profile chosen in electrode.geom_type.
- If sphere was chosen, this value describes the center of the sphere.
- If box was chosen, this value describes the lower left corner of box.
- If cylinder was chosen, this value describes the start of the cylinders main axis.
- For sphere & cylinder the additional parameter electrode.radius is needed.
- For box & cylinder the additional parameter electrode.p1 is needed.
Unit: micrometers
Default: (Float)(0.)(Float)(0.)(Float)(0.)

Required parameter:
stim

vtx_fcn <Short>

Whether the electrode is spatially constant (homogenous) or varying (inhomogenous).Spatially inhomogenous stimulations need to be defined via a .vtx file that includes nodes and nodal scaling.
Default: (Short)(0)

Possible values are:

(Short)(1): inhomogenous
(Short)(0): homogenous

Required parameter:
stim

Parameters (continued)

vtx_file <RFile>

File name allowing a vertex based electrode definition. Using a non-empty stringswitches to vertex-based definition and ignores other settings.
For file format specifications check the first chapters of the openCARP manual.
Default: (RFile)(“”)

Required parameter:
stim

domain <Int>

tissue domains affected
Default: (Int)(1)

Possible values are:

(Int)(3): all
(Int)(2): Purkinje only
(Int)(1): myocardium only

Required parameter:
stim

geomID <Int>

region ID defining an electrode's geometry, standard block defs are used if set to -1.
Default: (Int)(-1)

Required parameter:
stim

23.3.3 ptcl

Description “Definition of stimulation protocol“

Parameters

stimlist <String>

List of activation times (float), defining the stim protocol. Entries are separated by a comma (,). The list should be surrounded by quotes. An activation time can also be expressed as an increment to the previous
activation, by starting with a '+', e.g.: '0,+240'
Unit: ms
Default: (String)(“”)

Required parameter:
stim

bcl <Float>

Defines the basic cycle length for repetitive stimulation.
protocol.duration must be smaller than protocol.bcl, as it specifies the duration of a single stimulation event.The full protocol duration is protocol.npls · protocol.bcl. This is derived automatically from the user input.
Unit: ms
Default: (Float)(tend-[[Father]].start)

Value must be between (Float)(stim.duration) and (Float)(tend-stim.start)
Required parameters:
stim.start
tend
stim.duration
stim

npls <Int>

Defines the number of pulses in the stimulation protocol. The period of pulses can be set with bcl in ms.If set to 0, the stimulus will have a single instance.
Default: (Int)(tend > 0 ? 1 : 0)

Parameters (continued)

Value must be greater than (Int)(0)

Required parameters:
tend
stim

duration <Float>

Defines the duration of one a single stimulation event.
Unit: ms
Default: (Float)(tend-[[Father]].start)

Value must be between (Float)(0.) and (Float)(tend-stim.start)
Required parameters:
stim.start
tend
stim

start <Double>

Defines the start time of the stimulation protocol
Unit: ms
Default: (Double)(0.)

Value must be between (Double)(0.) and (Double)(tend)
Required parameters:
tend
stim

name <String>

Label of protocol used (S1-S2, restitution, etc). This helps with structuring extensive parameter lists and makes them more human readable.
Default: (String)(“”)

Required parameter:
stim

23.3.4 pulse

Description “Definition of stimulation pulse“

Parameters

bias <Float>

Defines a constant term which is added to the stimulus pulse.
Default: (Float)(0.)

Required parameter:
stim

tau_plateau <Float>

Defines a time constant governing plateau of pulse (>10ˆ5 is infinite).
A larger tau_plateau will result in a longer plateauphase.
Other important parameters for stimulus form definition are: pulse.tau_edge, pulse.tau_plateau, pulse.strength, pulse.duration
Unit: ms
Default: (Float)(1000.)

Value must be between (Float)(0.1) and (Float)(1000000.)
Required parameter:
stim

tau_edge <Float>

Defines the time constant governing leading and trailing edge of pulse. The formulation used resembles the equation: Amp · (1 - exp(-t/tau_edge)) · exp(-t/tau_plateau).
A larger tau_edge will result in a fast drop, while a smaller tau_edge will have a longer exponentialy decreasing flank.
Other important parameters for stimulus form definition are: pulse.tau_edge, pulse.tau_plateau, pulse.strength, pulse.duration

Parameters (continued)

Unit: ms
Default: (Float)(0.01)

Value must be between (Float)(0.) and (Float)(1000.)
Required parameter:
stim

s2 <Float>

This value is only used for defining a biphasic pulse.
This value is a relative value and defines the stimulus strength of the trailing pulse (after 0 crossing) relative to leading pulse (from start to 0 crossing). Giving the value 0 will result in a flip of polarity. Other important parameters for stimulus form definition are: pulse.tau_edge, pulse.tau_plateau, pulse.strength, pulse.duration
Default: (Float)(0.)

Value must be between (Float)(0.) and (Float)(10.)
Required parameter:
stim

tilt_ampl <Float>

prescribe fixed tilt amplitude (as in real shock delivery devices by capacitive discharge)
Default: (Float)(1.0)

Value must be between (Float)(0.) and (Float)(1.0)
Required parameter:
stim

tilt_time <Float>

prescribe fixed tilt time (not value) relative to pulse duration
Default: (Float)(1.0)

Parameters (continued)

Value must be between (Float)(0.) and (Float)(1.0)
Required parameter:
stim

strength <Float>

Defines strength of prescribed stimulation. Strength is defined as amplitude of signal. If dealing with currents it is defined as uA/volume. If dealing with voltages it is defined as mV.
To create a stimulation the minimum requirement is a given pulse.duration and pulse.strength
Unit: uA/cmˆ2(2D current), uA/cmˆ3(3D current), or mV
Default: (Float)(0.)

Required parameter:
stim

file <RFile>

Reads in pulse definition from external file.
For file format specifications check the first chapters of the openCARP manual.
Default: (RFile)(“”)

Required parameter:
stim

Parameters (continued)

shape <Short>

Defines the shape of the prescribed pulse if an implemented pulseform is chosen (see choices below). To impose a manually created stimulus use the pulse.file functionality.
By default, the pulse shape is defined as a monophasic pulse of square-like shape of a certain duration and strength. There are two time constants:
- tau_edge, governs the leading and trailing edges of the pulse.
- tau_plateau, governs the plateau phase.
By default, these time constants are set to yield essentially a square-like pulse shape, but can be easily adjusted to generate truncated exponential-like pulses.
Biphasic shapes are specified by two additional parameters, the duration of the first part of the pulse relative to the duration of the entire pulse (specified by pulse.d1 in range of [0,1]), and the strength of the second part of the pulse relative to the strength of the first part (specified by pulse.s2 in range of [0,1]).
To create a stimulation the minimum requirement is a given pulse.duration and pulse.strength
Default: (Short)(0)

Possible values are:

(Short)(2): sine wave
(Short)(1): truncated exponential wave
(Short)(0): square wave

Required parameter:
stim

name <String>

Definition of the Label for the prescribed pulse waveform

Parameters (continued)

Default: (String)(“”)

Required parameter:
stim

output

In this section, you can find the parameters related to all the output files of a simulation. Moreover, you can also find parameters that will modify the output in the terminal.

24.1 simID

Parameters

simID <String>

Defines the Simulation ID to generate output directory.
A good practice is to include date and time of the simulation in the 'simID'to avoid overwriting simulations. If 'simID' already exists, e.g the directory was created in a previous run, the user needs to decide on how to proceed (overwrite, append, abort) the simulation.
Default: (String)("OUTPUT_DIR")

24.2 dt

Parameters

dt <Double>

Defines the time step size to solve the numeric equations for.
Check the first chapters of the openCARP manual for a comprehensive explanation on how to choose 'dt'.
Unit: microseconds
Default: (Double)(5.)

Value must be greater than (Double)(0.)

24.3 tend

Parameters

tend <Double>

Defines the point in time when the simulation stops.
To get a rough estimation for 'tend', multiply the number of stimulation pulses 'stimulation.npls' with the basic cycle length 'stimulation.bcl'. 'tend' then needs to be larger than 'stimulation.npls · stimulation.bcl' to cover your entire stimulation protocol.
Unit: ms
Default: (Double)(100.)

Value must be greater than (Double)(dt/1000.)

24.4 num_io_nodes

Parameters

num_io_nodes <Int>

The number of nodes to dedicate to doing IO
Default: (Int)(0)

24.5 buildinfo

Parameters

buildinfo <Flag>

(deprecated) the build info is now shown by default.
hence this flag has no effect.
Default: (Flag)(PrMFALSE)

Parameters (continued)

24.6 output_level

Parameters

output_level <Int>

Defines the level of verbosity [0, 10] to the terminal output.
0 is considered minimal feedback to the user on the terminal.
Default: (Int)(1)

Value must be between (Int)(0) and (Int)(10)

24.7 dump2MatLab

Parameters

dump2MatLab <Flag>

Dumps stiffness and mass matrices, mappings and stimulation vectors
to be used with MatLab to the simulation folder
Default: (Flag)(0)

24.8 dump_basename

Parameters

dump_basename <String>

Defines the basename for dumped files. Different endings will be attached to specify different variables:
- _Ki & _Kie ... for the intra- & extracellular stiffness matrices
- _Mi & _Me ... for the intra- & extracellular mass matrices
- _i2e & _e2i ... for the intracellular-to-extracellular and vice-versa mappings
- _Itr & _Ie ... for the transmembrane- & extracellular currents
Default: (String)("MatLabDump")

24.9 vofile

Parameters

vofile <WFile>

IGB formatted file of transmembrane voltages.
Default: (WFile)("vm")

24.10 phiefile

Parameters

phiefile <WFile>

IGB formatted file of extracellular potential (phi).
Default: (WFile)("phie")

24.11 phieifile

Parameters

phieifile <WFile>

IGB formatted file of extracellular potential (phie) on intracellular grid (only in presence of bath required).
Default: (WFile)("phie_i")

24.12 gridout_i

Parameters

gridout_i <Int>

Defines if intracellular grid is outputted in simulation directory.
Settings: 0=none, 1=surface, 2=volumetric mesh, 3=surface & volumetric mesh.

Default: (Int)(0)

Value must be greater than (Int)(0)

24.13 gridout_e

Parameters

gridout_e <Int>

Defines if extracellular grid is outputted in simulation directory.
Settings: 0=none, 1=surface, 2=volumetric mesh, 3=surface & volumetric mesh.

Default: (Int)(0)

Parameters (continued)

Value must be greater than (Int)(0)

24.14 gridout_p

Parameters

gridout_p <Int>

If not 0, writes partition index of each element as .dat file in simulation directory.
Default: (Int)(0)

24.15 dataout_i

Parameters

dataout_i <Short>

Defines how intracellular grid data is outputted.
Default: (Short)(2)

Possible values are:

(Short)(3): output defined in .vtx file, see dataout_i_vtx
(Short)(2): volume output
(Short)(1): surface output
(Short)(0): turn off output

24.16 dataout_i_vtx

Parameters

dataout_i_vtx <RFile>

if option dataout_i == 3, intracellular output is restricted to the provided vertices.
Default: (RFile)(“”)

24.17 dataout_e

Parameters

dataout_e <Short>

Defines how extracellular grid data is outputted.
Default: (Short)(2)

Possible values are:

(Short)(3): output defined in .vtx file, see dataout_e_vtx
(Short)(2): volume output
(Short)(1): surface output
(Short)(0): turn off output

24.18 dataout_e_vtx

Parameters

dataout_e_vtx <RFile>

if option dataout_e == 3, extracellular output is restricted to the provided vertices.
Default: (RFile)(“”)

24.19 spacedt

Parameters

spacedt <Double>

Defines the temporal interval to output data to files.
It can only be as small as 'dt/1000'.
For long simulations outputting every single calculated value, would yield terrabytes of data. So here you can reduce the outputted values.
Unit: ms
Default: (Double)(3.)

Value must be between (Double)(dt/1000.) and (Double)(tend)

24.20 timedt

Parameters

timedt <Double>

Defines the temporal interval between progress updates made to the terminal. (For informational purposes only).
Unit: ms
Default: (Double)(1.)

Value must be between (Double)(dt/1000.) and (Double)(tend)

24.21 spacetol

Parameters

spacetol <Float>

Defines the spatial tolerance when searching for a node.
Unit: um
Default: (Float)(100.)

Parameters (continued)

Value must be greater than (Float)(0.)

24.22 display_meminfo

Parameters

display_meminfo <Short>

Turns on/off displaying memory malloc info.
Default: (Short)(0)

Possible values are:

(Short)(1): Turn on meminfo output
(Short)(0): Turn off meminfo output

cell_size

Defines the length of the cell used in the model.

25.1 cell_length

Parameters

cell_length <Float>

Defines the default cell length.
Unit: um
Default: (Float)(100.)

Value must be greater than (Float)(0.)

savestate

Defines all variables to save and read the simulated system states and therefore the simulation progress. Additionally allows for interval saving of simulation and reserving queue time in batch processing.

26.1 num_tsav

Parameters

num_tsav <Int>

Defines the number of states to be saved. Each time instant of the simulation progression needs to be specified in 'tsav'.
Default: (Int)(0)

Value must be between (Int)(0) and (Int)(50)

Required parameters:
tsav
tsav_ext

26.2 tsav

Parameters

tsav <Double>

Defines the times at which to save the simulation state.
Multiple saves for different times can be defined by adding the times to this array.
Unit: ms
Default: (Double)(tend-dt/1000.)

Value must be between (Double)(0.) and (Double)(num_tsav>1?tend:1.0e50)

26.3 tsav_ext

Parameters

tsav_ext <WFile>

Defines the filename for each saved state file. For multiple save states of the simulation, add the names for all savetimes in respective order to this array.
Default: (WFile)(opencarp::stringify(tsav[PrMelem1]))

26.4 write_statef

Parameters

write_statef <WFile>

Defines the basename of the output file in which to write a single saved state. Each saved state will have a timestamp appended to the basename.
Default: (WFile)(“state“)

26.5 start_statef

Parameters

start_statef <RFile>

Loads a statefile and continues the simulation from there.
Default: (RFile)(““)

26.6 chkpt_start

Parameters

chkpt_start <Float>

Defines the start time to trigger interval-based checkpointing (saving of the simulation progress).
Unit: ms
Default: (Float)(0.)

Value must be between (Float)(0.) and (Float)(tend)

26.7 chkpt_intv

Parameters

chkpt_intv <Float>

Defines the interval for checkpointing (saving of the simulation progress) in ms. 0 means no checkpointing.
Unit: ms
Default: (Float)(0.)

Value must be between (Float)(0.) and (Float)(tend)

26.8 chkpt_stop

Parameters

chkpt_stop <Float>

Defines the time to stop interval-based checkpointing (saving of the simulation progress).
Unit: ms
Default: (Float)(tend)

Value must be between (Float)(0.) and (Float)(tend)

26.9 queue_time

Parameters

queue_time <Float>

Defines the reserved queue time when running in batch mode
Unit: hours
Default: (Float)(168.)

Value must be between (Float)(0.) and (Float)(168.)

26.10 shrimp_pipe

Parameters

shrimp_pipe <String>

Defines the path to file where info should be dumped when openCARP receives SIGIO (in general, this should be a named pipe)
Default: (String)(“”)

26.11 state_dump_buffer

Parameters

state_dump_buffer <Long>

Defines the dump buffer size
Default: (Long)(100000000)

Value must be greater than (Long)(1000)

activation

In openCARP, the user can define several methods to detect an activation threshold in tissue simulations. It also offers several functions to start a simulation based on LATs, stop or restart a simulation. Additionally, you can obtain APD statics as an output file.

27.1 LAT_ID

Parameters

LAT_ID <String>

a variable
Default: default_value[PrMelem1]

27.2 num_LATs

Parameters

num_LATs <Int>

Defines the number of local activation measurements.
Default: (Int)(0)

Value must be greater than (Int)(0)

Required parameters:
lats
lats[PrMelem1].measurand
lats[PrMelem1].all
lats[PrMelem1].mode
lats[PrMelem1].threshold
lats[PrMelem1].method
lats[PrMelem1].ID

27.3 prepacing_lats

Parameters

prepacing_lats <RFile>

Tissue activation times to guide state set-up for prepacing
Default: (RFile)(“”)

27.4 prepacing_beats

Parameters

prepacing_beats <Int>

Defines the number of beats to pre-pace using a single-cell model.
Prepacing of a single cell is used to put the single-cell models into a preconditioned state.
This state is then given all cells in a tissue simulation as a better initial starting point than a completly unstimulated cell
Default: (Int)(0)

27.5 prepacing_bcl

Parameters

prepacing_bcl <Float>

Sets the basic cycle length for the prepacing stimulus.
Default: (Float)(0.)

27.6 LAT

Description Array containing LATs. Index of array corresponds to LAT measurement.

Parameters

ID <WFile>

Defines the output filename.
Default: (WFile)([[Father]].measurand ? LAT_ID[1] : LAT_ID[0])

Required parameters:
lats.method
lats.measurand
lats

measurand <Int>

Defines the quantity to measure, e.g. the signal (transmembrane voltage or extracellular potential) to use the thresholds on.
Default: (Int)(0)

Possible values are:

(Int)(1): Phie
(Int)(0): Vm

Required parameter:
lats

all <Int>

Defines that all activations should be detected (1) or only the first one (0). Detecting 'all' is the default.
Default: (Int)(1)

Value must be between (Int)(0) and (Int)(1)

Required parameter:
lats

Parameters (continued)

mode <Short>

Toggles between detecting max derivative or positive(+) slope threshold crossing and detecting minimum derivative and negative(-) slope threshold crossing.
Default: (Short)(0)

Possible values are:

(Short)(1): detect min derivative or -slope threshold crossing
(Short)(0): detect max derivative or +slope threshold crossing

Required parameter:
lats

threshold <Float>

Defines the crossing threshold (for method 1) or maximum derivative threshold (for method 2)
Default: (Float)(-10.)

Required parameter:
lats

method <Int>

Describes the method used to determine the instant of local activation. Define the threshold using structure.threshold. Choose if you want to evaluate during the rising or falling slope of the signal using structure.mode.
Default: (Int)(1)

Possible values are:

(Int)(2): instant of maximum derivative (do not use, not implemented yet)
(Int)(1): instant of threshold crossing

Parameters (continued)

Required parameter:
lats

27.7 t_sentinel

Parameters

t_sentinel <Float>

Sentinel checks for activations, based on LATs. If none are found, exits simulation, else continues.
t_sentinel should always be >0 .
t_sentinel describes the time sentinel is run for from the t_sentinel_start time.
If during this time period no lats[] are detected, savequit() cleanly.
Default: (Float)(-1.)

27.8 t_sentinel_start

Parameters

t_sentinel_start <Float>

Defines the time instant when checking for quiescence is started
Default: (Float)(0.)

27.9 sentinel_ID

Parameters

sentinel_ID <Int>

Sentinel will check the LAT ID specified here as a reference to quit or continue the simulation.
Default: (Int)(-1)

Value must be smaller than (Int)(num_LATs-1)

27.10 compute_APD

Parameters

compute_APD <Flag>

Defines if the actionpotential duration should be computed
- If is set to 1 = computes action potential durations
- If is set to 0 = action potential durations are not calculated
Default: (Flag)(PrMFALSE)

27.11 actthresh

Parameters

actthresh <Float>

Defines the threshold to determine if element was activated, e.g the threshold where an action potential was triggered.
The magnitude is used from the signal to be thresholded.
Unit: mV
Default: (Float)(30.)

Parameters (continued)

27.12 recovery_thresh

Parameters

recovery_thresh <Float>

Defines the threshold to determine if element wich was activated recovered back to its steadystate. The magnitude is used from the signal to be thresholded.
Unit: mV
Default: (Float)(-60.)

phys_regions

openCARP is designed to support multi-physics from the ground up. Therefore, the mesh regions are assigned to different physics. For simple EP experiments on the whole mesh without bath, no additional input w.r.t. CARPentry is required, as all regions are assigned by default to the Intracellular and Extracellular domains. With a bath present, the user needs to inform the simulator which regions form which simulation domains.

28.1 num_phys_regions

Parameters

num_phys_regions <Int>

The number of physics regions
Default: (Int)(0)

Value must be greater than (Int)(0)

Required parameters:
phys_region
phys_region[PrMelem1].num_IDs
phys_region[PrMelem1].name
phys_region[PrMelem1].ptype
phys_region[PrMelem1].ID

28.2 p_region

Description Array containing the defined physics regions.

Parameters

ID <Int>

Define a list of tags forming a mesh region.
Default: (Int)(0)

Required parameters:
phys_region.num_IDs
phys_region

Parameters (continued)

num_IDs <Int>

Defines the number of IDs (equivalent to element tags) listed under 'ID'.
Default: (Int)(0)

Required parameter:
phys_region

name <String>

Symbolic name for the physics region
Default: (String)(“”)

Required parameter:
phys_region

ptype <Int>

Defines the type of physics.
Default: (Int)(0)

Possible values are:

(Int)(1): Extracellular Electrics
(Int)(0): Intracellular Electrics

Required parameter:
phys_region

29.1 Spatial discretization using the Galerkin FEM

For the spatial discretization of PDEs (10), (11) we use the classical finite element method (FEM) (for the introduction's details see [12], [2], [13], [5], [1]). An application of the FEM to the cardiac electro physiology (bidomain model) is in details described in [10].

Briefly, the bidomain equations are multiplied with a weighting function, α, and integrated over the entire domain, Ω:

$$\int_\Omega \nabla \cdot (\boldsymbol{\sigma_i} + \boldsymbol{\sigma_e}) \nabla \phi_e \alpha d\Omega = -\int_\Omega \nabla \cdot \boldsymbol{\sigma_i} \nabla V_m \alpha d\Omega - \int_\Omega \alpha I_e d\Omega \tag{48}$$

$$\int_\Omega \nabla \cdot \boldsymbol{\sigma_i} \nabla V_m \alpha d\Omega = -\int_\Omega \nabla \cdot \boldsymbol{\sigma_i} \nabla \phi_e \alpha d\Omega + \beta \int_\Omega \alpha I_m d\Omega \tag{49}$$

Arbitrary weighting functions can be chosen for α, however, α has to fulfill any prescribed boundary conditions. Using the identity

$$\nabla \cdot (\alpha \mathbf{j}) = \nabla \alpha \cdot \mathbf{j} + \alpha \nabla \cdot \mathbf{j} \tag{50}$$

or

$$\alpha \nabla \cdot \mathbf{j} = \nabla \cdot (\alpha \mathbf{j}) - \nabla \alpha \cdot \mathbf{j} \tag{51}$$

where the term $\alpha \nabla \cdot \mathbf{j}$ with $\mathbf{j} = \boldsymbol{\sigma} \nabla \phi$ appears in the integral of Eqs. (48)-(49), allows to rewrite these integrals as

$$\int_\Omega \alpha \nabla \cdot \mathbf{j} d\Omega = \int_\Omega \nabla \cdot (\alpha \mathbf{j}) d\Omega - \int_\Omega \nabla \alpha \cdot \mathbf{j} d\Omega \tag{52}$$

or

$$\int_\Omega \alpha \nabla \cdot \mathbf{j} d\Omega = \int_\Gamma \alpha \mathbf{j} d\Gamma - \int_\Omega \nabla \alpha \cdot \mathbf{j} d\Omega \tag{53}$$

where the surface integral over the domain boundary $\partial\Omega$ is zero, unless non-zero Neumann flux boundary conditions are enforced. Substituting Eq. (53) into Eqs. (48)-(49) yields

$$-\int_\Omega \nabla \alpha \cdot (\boldsymbol{\sigma_i} + \boldsymbol{\sigma_e}) \nabla \phi_e d\Omega = \int_\Omega \nabla \alpha \cdot \boldsymbol{\sigma_i} \nabla V_m d\Omega - \int_\Omega \alpha I_e d\Omega \tag{54}$$

$$-\int_\Omega \nabla \alpha \cdot \boldsymbol{\sigma_i} \nabla V_m d\Omega = \int_\Omega \nabla \alpha \cdot \boldsymbol{\sigma_i} \nabla \phi_e d\Omega + \beta \int_\Omega \alpha I_m d\Omega \tag{55}$$

If we further assume that $\alpha(x) = 1$ holds everywhere over the domain, except maybe along Dirichlet boundaries, we may write all sought after functions in terms of products

such as

$$V_m(x) = \alpha(x) V_m(x) \tag{56}$$
$$I_m(x) = \alpha(x) I_m(x) \tag{57}$$
$$I_{ioin}(x) = \alpha(x) I_{ion}(x) \tag{58}$$
$$\phi_e(x) = \alpha(x) \phi_e(x) \tag{59}$$

and substituting into Eqs. (54)-(55) yields the final form

$$-\int_\Omega \nabla\alpha \cdot (\boldsymbol{\sigma_i} + \boldsymbol{\sigma_e}) \nabla\phi_e d\Omega = \int_\Omega \nabla\alpha \cdot \boldsymbol{\sigma_i} \nabla V_m d\Omega - \int_\Omega \alpha I_e d\Omega \tag{60}$$
$$-\int_\Omega \nabla\alpha \cdot \boldsymbol{\sigma_i} \nabla V_m d\Omega = \int_\Omega \nabla\alpha \cdot \boldsymbol{\sigma_i} \nabla\phi_e d\Omega + \beta \int_\Omega \alpha I_m d\Omega \tag{61}$$

Stiffness matrices, $\boldsymbol{\sigma}$ are discretized to have positive main diagonals, that is

$$\mathbf{K}_\zeta \phi \approx -\nabla \cdot \boldsymbol{\sigma_\zeta} \nabla\phi \tag{62}$$

We write the elliptic parabolic cast of the bidomain equations, as given by Eq. (10)-(11), in their spatially discrete form as follows:

$$\mathbf{K_{i+e}} \boldsymbol{\phi_e} = -\mathbf{K_i} \mathbf{v_m} + \mathbf{M_e} \mathbf{I_e} \tag{63}$$
$$\beta C_m \mathbf{M_i} \frac{\partial \mathbf{v_m}}{\partial t} = -\mathbf{K_i} (\mathbf{v_m} + \boldsymbol{\phi}_e) - \beta \mathbf{M_i} (I_{ion} - I_{tr}) \tag{64}$$

By default the implementation of the bidomain equations in CARP relies on elliptic-parabolic decoupling and operator splitting of the parabolic equations which results in

$$\mathbf{K_{i+e}} \boldsymbol{\phi_e} = -\mathbf{K_i} \mathbf{v_m} + \mathbf{M_e} \mathbf{I_e} \tag{65}$$
$$\beta C_m \mathbf{M_i} \frac{\partial \mathbf{v_m}}{\partial t} = -\beta \mathbf{M_i} (\mathbf{I_{ion}} - \mathbf{I_{tr}}) \tag{66}$$
$$\beta C_m \mathbf{M_i} \frac{\partial \mathbf{v_m}}{\partial t} = -\mathbf{K_i} (\mathbf{v_m} + \boldsymbol{\phi}_e) \tag{67}$$

where in Eq. (66) $\beta \mathbf{M_i}$ cancels out. The implementation of Eq. (66) is then

$$\frac{\partial \mathbf{v_m}}{\partial t} = -\frac{1}{C_m} (\mathbf{I_{ion}} - \mathbf{I_{tr}}) \tag{68}$$

29.2 Domain mapping

In the presence of a bath intracellular and extracellular domain differ in size. Extracellular and intracellular domain overlap over the entire domain Ω_i, however, in the bath domain $\Omega_b = \Omega_e \setminus \Omega_i$ the intracellular space does not exist. Therefore, vectors defined on the two discrete spaces have to mapped as follows

$$\mathbf{K_{i+e}} \boldsymbol{\phi_e} = -\mathbf{K_i} \mathbf{P} \mathbf{v_m} + \mathbf{M_e} \mathbf{I_e} \tag{69}$$
$$\beta C_m \mathbf{M_i} \frac{\partial \mathbf{v_m}}{\partial t} = -\mathbf{K_i} \left(\mathbf{v_m} + \mathbf{P^{-T}} \boldsymbol{\phi}_e\right) \tag{70}$$
$$\tag{71}$$

where $\mathbf{P}$ is the prolongation $\Omega_i \mapsto \Omega_e$ and $\mathbf{P^{-T}}$ is the restriction $\Omega_e \mapsto \Omega_i$. It is worth noting that in our implementation the same spatial discretization is shared in the overlapping domain, thus only index-based mapping is performed for transferring data between the grids.

29.3 Temporal discretization schemes

Temporal discretization is based on the assumption of parabolic-elliptic decoupling of the bidomain equations. That is, unlike in [8], elliptic and parabolic portions are not solved as a single system. First, the elliptic portion is solved for to obtain $\phi_e(t)$ as a function of the current distribution of the transmembrane voltage, V_m, enforced Dirichlet potentials ϕ_{eD} or extracellular stimulus currents, I_e. The extracellular potential field ϕ_e is used then in the parabolic equation to solve for V_m^{t+dt}. There are two basic approaches implemented for this solve where the standard method relies upon operator splitting [9]. In theory, a Strang splitting should be used to achieve second order accuracy, however, in practice only a first order accurate Godunov splitting scheme is used. See [11] for details. In practice, we do not expect any difference between the two schemes. Only the first time step is different between the two schemes. That is, if the solution during the first shift by half a time step to achieve a $dt/2$ delay between ODE and parabolic PDE solution does not change, there cannot be any differences between a Godunov and a Strang splitting. The main advantage of operator splitting is that it renders the parabolic PDE linear, which can be beneficial in terms of iteration numbers when using an iterative method. In the alternative scenario we refrain from operator splitting and solve the parabolic PDE which is non-linear now since the non-linear term $I_{ion}(V_m, \boldsymbol{\eta}, t)$ shows up on the right hand side.

Using the spatially discrete representations of V_m, $\mathbf{v_m}$, and of ϕ_e, $\boldsymbol{\phi_e}$, and discretizing in time with

$$t = kdt \tag{72}$$

we write spatio-temporally discretized representations as

$$V_m(x,t) \approx \mathbf{v_m^k} \tag{73}$$

$$\phi_e(x,t) \approx \boldsymbol{\phi_e}^{\mathbf{k}} \tag{74}$$

The basics of the two schemes are given as follows:

29.3.0.0 Temporal discretization with operator splitting

$$\mathbf{K_{i+e}}\boldsymbol{\phi}_\mathbf{e}^\mathbf{k} = -\mathbf{P}\mathbf{K_i}\mathbf{v}_\mathbf{m}^\mathbf{k} + \mathbf{M_e}\mathbf{I_e} \tag{75}$$

$$\frac{\partial \boldsymbol{\eta}}{\partial t} = g(\mathbf{v}_\mathbf{m}^\mathbf{k}, \boldsymbol{\eta}^\mathbf{k}) \Longrightarrow \boldsymbol{\eta}^{\boldsymbol{k+1}} \tag{76}$$

$$\mathbf{I}_\mathbf{ion}^\chi = f(\mathbf{v}_\mathbf{m}^\mathbf{k}, \boldsymbol{\eta}^{\mathbf{k+1}}) \tag{77}$$

$$\mathbf{v}_\mathbf{m}^\chi = \mathbf{v}_\mathbf{m}^\mathbf{k} - \frac{\Delta t_o}{C_m}\left(\mathbf{I}_\mathbf{ion}^\chi - \mathbf{I_{tr}}\right) \tag{78}$$

$$\beta C_m \mathbf{M_i}\frac{\partial \mathbf{v}_\mathbf{m}^\chi}{\partial t} = -\mathbf{K_i}\left(\mathbf{v}_\mathbf{m}^\chi + \mathbf{P}^{-\mathbf{T}}\boldsymbol{\phi}_e{}^\mathbf{k}\right) \tag{79}$$

Here $\mathbf{M}$ is the mass matrix, $\mathbf{K}$ is the stiffness matrix with the subscript specifying which conductivity to use, intracellular (i), extracellular (e) or their sum $(i+e)$, $\boldsymbol{\eta}$ is the set of state variables of the ionic model, $g()$ is the ionic model dependent set of state equations and $f()$ is a function describing the total ionic current across the membrane as a function of the current state, $\boldsymbol{\eta}$. Note that χ indicates the result of an intermediate compute step where $\mathbf{I}_\mathbf{ion}^\chi$ is computed with an updated state $\boldsymbol{\eta}^{\mathbf{k+1}}$ and the current transmembrane voltage $\mathbf{v}_\mathbf{m}^\mathbf{k}$. The symbol $\Longrightarrow$ indicates that the set of ODEs is solved to compute $\boldsymbol{\eta}^{\boldsymbol{k+1}}$. Unlike in Eq. (78) where we always use a simple forward Euler update where we advance the solution by the ODE time step, Δt_o. Numerous options are available for updating $\boldsymbol{\eta}^{\boldsymbol{k+1}}$, thus only a symbol is used to indicate that an implementation-dependent solver step is executed. By default our implementation relies upon an accelerated Rush-Larsen technique which has been described in detail elsewhere [6]. In the following derivations it is convenient to scale the intracellular mass matrix, $\mathbf{M_i}$ with κ such that

$$\kappa = \frac{\beta C_m}{\Delta t_p} \tag{80}$$

$$\bar{\mathbf{M}}_\mathbf{i} = \kappa \mathbf{M_i} \tag{81}$$

$$\Delta t_p = \frac{\Delta t}{n_s} \tag{82}$$

where $\mathbf{M_i}$ is the unscaled mass matrix and Δt_p is the time step used for advancing the parabolic solution which can be a fraction (but not a multiple) of the global electrical time step, Δt. That is, $n_s \geq 1$ is the integer number which specifies the number of parabolic sub-timesteps which are used to diffuse the change in V_m due to the ODE reaction terms. In general, we use $\Delta t = \Delta t_o = \Delta t_p$ since $n_s = 1$ is the default value. Using a value $n_s > 1$ is beneficial when using an explicit method with an unstructured grid where the CFL condition may impose severe limits upon the choice of Δt. In such cases, one may chose $\Delta t_o = \Delta t$ and n_s sufficiently large. Thus the ODE load is kept constant, the parabolic compute lead increases linearly with n_s, since parabolic solver steps are repeated n_s times. However, significant improvements in strong scalaling characteristics may allow to achieve shorter execution times. Details are found in [3].

29.3.0.0 Temporal discretization without operator splitting

$$\mathbf{K_{i+e}}\boldsymbol{\phi}_{\mathbf{e}}^{\mathbf{k}} = -\mathbf{P}\mathbf{K_i}\mathbf{v}_{\mathbf{m}}^{\mathbf{k}} + \mathbf{M_e}\mathbf{I_e} \tag{83}$$

$$\frac{\partial \boldsymbol{\eta}}{\partial t} = g(\mathbf{v}_{\mathbf{m}}^{\mathbf{k}}, \boldsymbol{\eta}^{\mathbf{k}}) \Longrightarrow \boldsymbol{\eta}^{\boldsymbol{k+1}} \tag{84}$$

$$\mathbf{I}_{\mathbf{ion}}^{\chi} = f(\mathbf{v}_{\mathbf{m}}^{\mathbf{k}}, \boldsymbol{\eta}^{\mathbf{k+1}}) \tag{85}$$

$$\beta C_m \mathbf{M_i}\frac{\partial \mathbf{v}_{\mathbf{m}}^{\chi}}{\partial t} = -\mathbf{K_i}\left(\mathbf{v}_{\mathbf{m}}^{\mathbf{k}} + \mathbf{P}^{-\mathbf{T}}{\boldsymbol{\phi}_e}^{\mathbf{k}}\right) - \beta \mathbf{M_i}\left(\mathbf{I}_{\mathbf{ion}}^{\chi} - \mathbf{I_{tr}}\right) \tag{86}$$

For the time discretization we consider one of the simple, for construction and implementation, explicit technique — Forward Euler scheme, as well as two fully Implicit methods — Crank-Nicolson and Second order time stepping schemes.

29.3.1 Forward Euler scheme (FE)

Forward Euler scheme is well known as Explicit method and has a following form:

$$\mathbf{v}_{\mathbf{m}}^{\chi} = \mathbf{v}_{\mathbf{m}}^{\mathbf{k}} - \frac{\Delta t_o}{C_m}\left(\mathbf{I}_{\mathbf{ion}}^{\chi} - \mathbf{I_{tr}}\right) \tag{87}$$

$$\mathbf{v}_{\mathbf{m}}^{\mathbf{k+1}} = \mathbf{v}_{\mathbf{m}}^{\mathbf{k}} + \bar{\mathbf{M}}_{\mathbf{i}}^{-\mathbf{1}}\mathbf{K_i}\left(\mathbf{v}_{\mathbf{m}}^{\chi} + \mathbf{P}^{-\mathbf{T}}{\boldsymbol{\phi}_e}^{\mathbf{k}}\right) \tag{88}$$

or, in the case without operator splitting,

$$\mathbf{v}_{\mathbf{m}}^{\mathbf{k+1}} = \mathbf{v}_{\mathbf{m}}^{\mathbf{k}} + \bar{\mathbf{M}}_{\mathbf{i}}^{-\mathbf{1}}\mathbf{K_i}\left(\mathbf{v}_{\mathbf{m}}^{\mathbf{k}} + \mathbf{P}^{-\mathbf{T}}{\boldsymbol{\phi}_e}^{\mathbf{k}}\right) - \beta \mathbf{M_i}\bar{\mathbf{M}}_{\mathbf{i}}^{-\mathbf{1}}(\mathbf{I_{ion}} - \mathbf{I_{tr}}) \tag{89}$$

where

$$\beta \mathbf{M_i}\bar{\mathbf{M}}_{\mathbf{i}}^{-\mathbf{1}} = \frac{\Delta t_p}{C_m}. \tag{90}$$

It has advantages in sense of less requirements of memory and computational time, but is not as stable and has some stability restrictions on the time step which are governed by the Courant-Friedrichs-Levy (CFL) condition. This restrictions upon Δt_p may become prohibitive when using fine spatial discretizations. This is of particular relevance when using unstructured grids for spatial discretization since a single short element edge in the grid may enforce a very small time step. In this case, FE mesh quality is of importance. One way of alleviating this problem is to use parabolic sub-timestepping. In this scenario a global Δt_p is used for solving the system of ODEs, but diffusion is computed in time steps of $\Delta t_p/n_s$ where n_s is an integer for subdividing the interval. Since a single Forward Euler step is so cheap, the method can still be beneficial, see for instance, in Niederer et al [3]. A further disadvantage is due to the requirement of mass lumping which is necessitated for simple inversion of the mass matrix since $\mathbf{M}_{\mathbf{i}}^{-\mathbf{1}}$ is used at the right hand side of Eq. (89).

29.3.2 Crank-Nicolson scheme (CN)

The Crank-Nicolson method gives possibility to avoids the strict stability restrictions on the time step and to improve accuracy and stability. In the case of operator splitting we solve

$$\mathbf{v_m^\chi} = \mathbf{v_m^k} - \frac{\Delta t_o}{C_m}\left(\mathbf{I_{ion}^\chi} - \mathbf{I_{tr}}\right) \tag{91}$$

$$\left(\bar{\mathbf{M}}_\mathbf{i} + \frac{\mathbf{K_i}}{2}\right)\mathbf{v_m^{k+1}} = -\mathbf{K_i}\left(\frac{\mathbf{v_m^\chi}}{2} + \mathbf{P^{-T}}{\boldsymbol{\phi}_e}^\mathbf{k}\right) + \bar{\mathbf{M}}_\mathbf{i}\mathbf{v_m^\chi} \tag{92}$$

or, without operator splitting, we have

$$\left(\bar{\mathbf{M}}_\mathbf{i} + \frac{\mathbf{K_i}}{2}\right)\mathbf{v_m^{k+1}} = -\mathbf{K_i}\left(\frac{\mathbf{v_m^\chi}}{2} + \mathbf{P^{-T}}{\boldsymbol{\phi}_e}^\mathbf{k}\right) + \bar{\mathbf{M}}_\mathbf{i}\mathbf{v_m^\chi} - \beta\mathbf{M_i}(\mathbf{I_{ion}} - \mathbf{I_{tr}}). \tag{93}$$

Note that in the case where we refrain from operator splitting both versions of the intracellular mass matrix, the unscaled matrix $\mathbf{M_i}$ and the scaled matrix $\bar{\mathbf{M}}_\mathbf{i}$, are used. Since only the scaled matrix $\bar{\mathbf{M}}_\mathbf{i}$ is stored we compute the right hand side contribution of ionic current and transmembrane stimulus current differently, using

$$\beta\mathbf{M_i}(\mathbf{I_{ion}} - \mathbf{I_{tr}}) = \frac{\Delta t_p}{C_m}\bar{\mathbf{M}}_\mathbf{i}(\mathbf{I_{ion}} - \mathbf{I_{tr}}). \tag{94}$$

In our current implementation we allow the bidomain surface-to-volume ratio β to vary, however, the membrane capacitance C_m is considered to be constant, i.e. $C_m = 1\mu F/cm^2$. As opposed to the explicit method, the solution of large systems of non-linear equations is required for each time step. However, due to the diagonal dominance of the problem the systems is solved quite efficiently with iterative methods, requiring only a few iterations. A particular advantage of the operator splitting approach is that the number of iterations tends to be even lower since the parabolic problem is linear.

29.3.3 Θ-schemes

Crank-Nicolson, forward as well as backward Euler can be seen as special cases of a general Θ-scheme where the individual methods correspond to the choices $\Theta = 0.5$ (CN), $\Theta = 0.$ (FE) and $\Theta = 1.0$ (BE). CN is second order accurate, but may be more prone to oscillations than BE. The general Θscheme for the bidomain is given as

$$\mathbf{v_m^\chi} = \mathbf{v_m^k} - \frac{\Delta t_o}{C_m}\left(\mathbf{I_{ion}^\chi} - \mathbf{I_{tr}}\right) \tag{95}$$

$$\left(\bar{\mathbf{M}}_\mathbf{i} + \theta\mathbf{K_i}\right)\mathbf{v_m^{k+1}} = -\mathbf{K_i}\left((1-\Theta)\mathbf{v_m^\chi} + \mathbf{P^{-T}}{\boldsymbol{\phi}_e}^\mathbf{k}\right) + \bar{\mathbf{M}}_\mathbf{i}\mathbf{v_m^\chi} \tag{96}$$

or, without operator splitting, we have

$$\left(\bar{\mathbf{M}}_\mathbf{i} + \frac{\mathbf{K_i}}{2}\right)\mathbf{v_m^{k+1}} = -\mathbf{K_i}\left(\frac{\mathbf{v_m^\chi}}{2} + \mathbf{P^{-T}}{\boldsymbol{\phi}_e}^\mathbf{k}\right) + \bar{\mathbf{M}}_\mathbf{i}\mathbf{v_m^\chi} - \beta\mathbf{M_i}(\mathbf{I_{ion}} - \mathbf{I_{tr}}). \tag{97}$$

Index

References

[1] K. J. Bathe. *Finite Element Procedures*. Prentice-Hall, 1995.

[2] P. G. Ciarlet. *The Finite Element Method for Elliptic Problems*, volume 4. North-Holland Publishing Company, 1978.

[3] Steven Niederer, Lawrence Mitchell, Nicolas Smith, and Gernot Plank. Simulating human cardiac electrophysiology on clinical time-scales. *Front Physiol*, 2:14, 2011.

[4] Bjørn Fredrik Nielsen, Tomas Syrstad Ruud, Glenn Terje Lines, and Aslak Tveito. Optimal monodomain approximations of the bidomain equations. *Applied Mathematics and Computation*, 184(2):276–290, 2007.

[5] J. Tinsley Oden and Graham F. Carey. *Finite Elements: Mathematical Aspects*, volume IV. Prentice Hall, 1983.

[6] Gernot Plank, Lufang Zhou, Joseph L. Greenstein, Sonia Cortassa, Raimond L. Winslow, Brian O'Rourke, and Natalia A. Trayanova. From mitochondrial ion channels to arrhythmias in the heart: computational techniques to bridge the spatio-temporal scales. *Philos Trans A Math Phys Eng Sci*, 366(1879):3381–3409, 2008.

[7] Andrew E. Pollard, Natalia Trayanova, and Craig S. Henriquez. A comparison of iterative methods for the determination of the interstitial potential distribution with the bidomain model. In *1992 14th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, volume 2, pages 602–603, 1992.

[8] James A. Southern, Gernot Plank, Edward J. Vigmond, and Jonathan P. Whiteley. Solving the coupled system improves computational efficiency of the bidomain equations. *IEEE Trans Biomed Eng*, 56(10):2404–2412, 2009.

[9] G. Strang. On the construction and comparision of difference scheme. *SIAM Journal on Numerical Analysis*, 5:506–517, 1968.

[10] J. Sundnes, G. T. Lines, X. Cai, B. F. Nielsen, K.-A. Mardal, and A. Tveito. *Computing the Electrical Activity in the Heart*. Springer-Verlag, Berlin, 2006.

[11] Joakim Sundnes, Glenn Terje Lines, and Aslak Tveito. An operator splitting method for solving the bidomain equations coupled to a volume conductor model for the torso. *Math Biosci*, 194(2):233–248, 2005.

[12] B. Szabo and I. Babuska. *Finite element analysis*. John Wiley & Sons, Inc., New York, 1991.

[13] M. Trew, I. Le Grice, B. Smaill, and A. Pullan. A finite volume method for modeling discontinuous electrical activation in cardiac tissue. *Ann. Biomed. Eng.*, 33:590–602, 2005.