Mountian Lion CSS Documentation

Release 0.0.1

Christopher Macklen

Mar 21, 2020

CONTENTS:

CHAPTER

ONE

MOUNTIAN LION CSS

Note on the badges: The badges reflect the current developmnet state of the project. All releases must successfully pass the pipeline prior to release.

Badge	Service
PyPI	
Read the docs	
Pipeline	
Coverage	

Mountain Lion Continuum-Scale Lithium-Ion Cell Simulator uses FEniCS to solve partial differential equation models for lithium-ion cells.

- Free software: MIT license
- Documentation: https://mtnlion.readthedocs.io.

1.1 Features

- Fast and customizable using model-based design
- Easily attach external controllers to the cell model
- Built-in Rothe's method time stepping using first-order implicit Euler's method

1.2 Included Models

- Doyle-Fuller-Newman isothermal cell model
- Thermal model
- Metallic lithium plaing model
- Solid-electrolyte interphase (SEI) model
- Double-layer capacitance model

1.3 Getting Started

mtnlion can be installed via PyPI using `pip install mtnlion --user`. However, you'll have to ensure that the correct version of FEniCS is already installed on your machine. You can reference the installation guide. for help

preparing your own development environment. The contributing guide is also available for those who wish to add to this projec.

1.4 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER

TWO

INSTALLATION

Mountain Lion CSS is a wrapper around FEniCS, and inherits the installation difficulty. Mountain Lion provides two methods to install:

1. Docker image with provided dependencies
2. Manual install

It's recommended to use the provided docker images as a base for your environments, and customize as required. Unfortunately, docker does impose restrictions on how you can interact with your environment. Therefore the manual install option is provided for users who provide their own distribution of FEniCS.

2.1 Docker Install

There are two evolving docker images provided with this project: `latest` and `devel`. The `latest` tag is provides a docker image with the latest tested version of FEniCS, and tracks the latest releases from PyPI and Gitlab releases. The `devel` image tracks the development branch of the repository, and represents the nightly build of the platform. The `devel` tag provides a repository for mtnlion in the users home folder, which provides the installed distribution as a development package. This allows mtnlion source files to be modified without having to re-install the package after every change.

For a basic setup, simply run one of these tags directly from the registry:

```
$ docker run --rm -it registry.gitlab.com/macklenc/mtnlion:<tag>
```

This will launch a temporary container for developing in. However, you'll probably want to use it for development. In order to keep your changes (including your development code) between launches, you'll need to get a little fancier with the run command:

```
$ docker run --rm -ti --network host --name mtnlion -v mtnlion_dev:/home/fenics␣
↪registry.gitlab.com/macklenc/mtnlion:<tag>
```

This will create a docker volume called `mtnlion_dev` to save all of the data inside `/home/fenics`, which is the default home folder. Now when you leave and re-enter the container with the same command, you'll have the same files available to you as long as it's in the folder that you attached the volume to.

If you are using Linux, the script provided in the repository under `tools/launch_sde.sh` will automatically create an environment for you. After running the script for the first time, it creates a Dockerfile in `$HOME/mtnlion-docker/Dockerfile`. This Dockerfile can be customized without changing code in the repository. The script also automatically handles X11 forwarding, image building if the Dockerfile gets updated, and container launching. If you need to force the docker engine to build, pass the script `--build`. The script also provides a brief help command `--help`. This is the recommended option for Linux users.

For other OSes, it is recommended, however, that you customize the image to add development tools. An example that can be edited is provided in the Gitlab repo in the dockerfiles folder (`mtnlion-development.dockerfile`):

```
FROM registry.gitlab.com/macklenc/mtnlion:devel as sde

# Shell
RUN sudo apt-get install -y zsh

# Nice shell
RUN wget -O .zshrc https://git.grml.org/f/grml-etc-core/etc/zsh/zshrc &&\
    wget -O .zshrc.local  https://git.grml.org/f/grml-etc-core/etc/skel/.zshrc

# Install gvim
RUN sudo apt-get install -y vim-gtk3

# Install sublime
RUN wget -qO - https://download.sublimetext.com/sublimehq-pub.gpg | sudo apt-key add -
↪ &&\
    sudo apt-get install -y apt-transport-https &&\
    echo "deb https://download.sublimetext.com/ apt/stable/" | sudo tee /etc/apt/
↪sources.list.d/sublime-text.list &&\
    sudo apt-get update &&\
    sudo apt-get install -y sublime-text sublime-merge

# Install pycharm
RUN wget https://download.jetbrains.com/python/pycharm-community-2019.1.1.tar.gz -qO -
↪ | sudo tar xfz - -C /opt/ &&\
    cd /usr/bin &&\
    sudo ln -s /opt/pycharm-*/bin/pycharm.sh pycharm
```

As you can see, this adds some development tools that require a GUI. To display the GUI elements, you'll have to enable X11 forwarding. The process is different on each host OS:

2.1.1 Linux

In Linux, you can use the following script:

```
XSOCK=/tmp/.X11-unix
XAUTH=/tmp/.docker.xauth
xauth nlist $DISPLAY | sed -e 's/^..../ffff/' | xauth -f $XAUTH nmerge -
chmod 644 $XAUTH
docker run -ti --rm -v $XSOCK:$XSOCK -v $XAUTH:$XAUTH -e XAUTHORITY=$XAUTH -e DISPLAY=
↪$DISPLAY -v mtnlion_dev:/home/mtnlion registry.gitlab.com/macklenc/mtnlion:<tag>
```

This script will securely enable X11 forwarding to your host. Now you can run, for example, PyCharm.

2.1.2 MacOS

In MacOS, you'll need to install XQuartz to provide an X11 server. Once XQuartz is installed, enable the option Allow connections from network clients, then restart XQuartz. Then you'll need to run the following script every time you launch your development container:

```
xhost + 127.0.0.1
docker run -e DISPLAY=host.docker.internal:0 -v mtnlion_dev:/home/mtnlion registry.
↪gitlab.com/macklenc/mtnlion:<tag>
```

Now you should be able to run your graphical apps from inside the container.

2.2 Manual Install

You can FEnsCS by following the FEniCS installation instructions. Make sure that the FEniCS version is compatible with the version of mtnlion you are using by checking the setup.py requirements, or observing the changelog in the Gitlab release repository.

2.2.1 Stable release

To install Mountian Lion CSS, run this command in your terminal:

```
$ pip install mtnlion
```

This is the preferred method to install mtnlion outside of docker images, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2.2 From sources

The sources for Mountian Lion CSS can be downloaded from the Gitlab repo or Gitlab release repository.

You can either clone the public repository:

```
$ git clone git://gitlab.com/macklenc/mtnlion
```

Or download the release:

```
$ curl  -OL https://gitlab.com/macklenc/mtnlion/-/releases/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER

THREE

USAGE

To use Mountian Lion CSS in a project:

```
import mtnlion
```

Examples can be found in the examples directory of the project repository.

CHAPTER

FOUR

USING COMSOL DATA

To convert mesh from COMSOL to FEniCS is a multistep process. The method that has worked in the past is to use FEConv to convert from .mphtxt that comsol exports to gmsh .msh format. Then open the .msh in gmsh and expand Mesh to select the dimensionality of the mesh. Then save over the .msh file. Finally, use dolfin-convert to convert from .msh to .xml.

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://gitlab.com/macklenc/mtnlion/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitLab issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitLab issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

Mountian Lion CSS could always use more documentation, whether as part of the official Mountian Lion CSS docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://gitlab.com/macklenc/mtnlion/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up `mtnlion` for local development.

First, fork the mtnlion project into your personal account. This will give you a sandbox to play in that won't effect other users in any way. Then follow the instructions from the installation guide *Installation*, following the proceedures to build a docker image will provide the best results. If you are not using the provided docker image, it's highly recommended to use pipenv as your package manager. In order to enable the virtual environment that the python packages are installed in, simply run `pipenv shell` when you launch the container.

Every time you commit the pre-installed pre-commit hooks will evaluate your code for quality. If any of the formatters report a failure, this means that they applied changes to your code and unstaged the relevant files. You can then perform a git diff to view the changes the formatter made then re-add the files and try committing again.

When the feature or bug you've been working on in your forked repo is ready, you can submit a merge request to the upstream repo. To do so, in the forked repo, go to Merge Requests and select the branch that you want to merge and select the `devel` branch in the upstream repo as the target. Then follow the Merge Request Guidelines and fill out the Merge Request Template.

5.3 pipenv

This project uses pipenv as it's package manager since it uses the "modern" Pipfile. It's recommended that you read about pipenv, however here are the basics:

- Run `pipenv sync` to install the dependencies listed in the lock file. the lock file represents the collection of packages that are known to work.
- Run `pipenv install` to install new packages so they get added to the lock file.
- Run `pipenv shell` to activate the virtual environment in a new shell

5.4 Merge Request Guidelines

Before you submit a merge request, check that it meets these guidelines:

1. The merge request should include tests.
2. If the merge request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.6.

5.5 Tips

- To run a subset of tests:

```
$ pytest tests.test_mtnlion
```

- Use pycharm! To setup pycharm simply import mtnlion and go to settings `Ctrl+Alt+S` then go to `Project: mtnlion -> Project Interpreter`, click on the gear and select `add`. Select `existing interpreter`, and the system environment should be auto-discovered. Choose that and exit all menu's selecting "OK".

5.6 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed.

Then tag the release version providing a changelog in the annotation. Pushing the tag will trigger the pipeline to deploy to GitLab releases, GitLab registry (for docker images), and PyPI

CHAPTER

SIX

MTNLION

6.1 mtnlion package

Top-level package for Mountian Lion CSS.

6.1.1 Subpackages

mtnlion.formulas package

Submodules

mtnlion.formulas.approximation module

Tools for approximating functions

class mtnlion.formulas.approximation.**LagrangeMultiplier**(*domains: List[str], trial_name: str, lm_name: str = None*)

Bases: *mtnlion.formula.Formula*

This formula provides Lagrange Multiplier functionality for domain boundaries.

form(*arguments: mtnlion.formula.Arguments*, *domain: str*) → ufl.core.expr.Expr

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class mtnlion.formulas.approximation.**Legendre**(*num_functions: int*)

Bases: object

Generate Legendre matrices from Eqs. (3.30) and (3.31) in "Continuum-Scale Lithium-Ion Battery Cell Model in FEniCS"

K

static Kmn(*row: int*, *col: int*) → float

Calculate the Kmn matrix entries

Parameters

- **row** – row to calculate
- **col** – column to calculate

M

static Mmn(*row: int*, *col: int*) → float
Calculate the Mmn matrix entries

Parameters

- **row** – row to calculate
- **col** – column to calculate

mtnlion.formulas.dfn module

A collection of formulas useful for the Doyl-Fuller-Newman cell model

class mtnlion.formulas.dfn.**CapacityLoss**
Bases: *mtnlion.formula.Formula*

Capacity loss due to the side reactions.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class mtnlion.formulas.dfn.**FilmResistance**
Bases: *mtnlion.formula.Formula*

Resistance of the film that builds around the surface of the particles.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class mtnlion.formulas.dfn.**FilmThickness**
Bases: *mtnlion.formula.Formula*

Thickness of the film that builds around the surface of the particles.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class mtnlion.formulas.dfn.**KappaDEff**
Bases: *mtnlion.formula.Formula*

kappa_d effective

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class mtnlion.formulas.dfn.**KappaEff**
Bases: *mtnlion.formula.Formula*

Effective conductivity of the electrolyte.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class mtnlion.formulas.dfn.**KappaRef**(*kappa_ref_str*)
Bases: *mtnlion.formula.Formula*

Bulk conductivity of the homogeneous materials.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class mtnlion.formulas.dfn.**SOC**
Bases: *mtnlion.formula.Formula*

State of Charge (SOC) formula.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class mtnlion.formulas.dfn.**Uocp**(*uocp_str*)
Bases: *mtnlion.formula.Formula*

Open-circuit potential formula.

form(*arguments*, *domain*)
Evaluate the open-circuit potential equation.

mtnlion.formulas.dfn.**eval_form**(*formula: mtnlion.formula.Formula*, **parameters*) → mtnlion.domain.Domain[str, ufl.core.expr.Expr]
Evaluate a formula with the provided parameters. Note: The order of the parameters must match the definition of the formula.

Parameters

- **formula** – Formula to evaluate
- **parameters** – Parameters to use for evaluation

Returns FFL expression

mtnlion.models package

Available models for simulation with the mtnlion framework

class mtnlion.models.**DoubleLayer**(*Ns*)
Bases: *mtnlion.models.isothermal.Isothermal*

The double-layer model comes from surface science where a structure forms on the surface of a solid when exposed to a fluid. In this case, the structure that forms on the particles is an electrical charge which causes an opposing charge to build up in the electrolyte near the surface. The separation of electrical charge around the particle surface has the same behavior as a plate capacitor.

class DoubleLayer
Bases: *mtnlion.formula.Formula*

Double-layer flux.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class TotalFlux
Bases: *mtnlion.formula.Formula*

Replaces the standard intercalation flux from the isothermal model.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class mtnlion.models.**Isothermal**(*num_functions*)
Bases: *mtnlion.model.Model*

The basic DFN lithium-ion model with no thermal considerations and a 1D approximation of the solid concentration.

class ElectrolyteConcentration
Bases: *mtnlion.formula.Formula*

Concentration of lithium in the electrolyte.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class ElectrolytePotential
Bases: *mtnlion.formula.Formula*

Charge conservation in the electrolyte.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class ExchangeCurrentDensity
Bases: *mtnlion.formula.Formula*

The exchange current density is the current in the absence of net electrolysis and at zero overpotential.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class IntercalationFlux
Bases: *mtnlion.formula.Formula*

Describes how the electrical current on an electrode depends on the electrode potential.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class KappaDEff
Bases: *mtnlion.formula.Formula*

kappa_d effective

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class KappaEff
Bases: *mtnlion.formula.Formula*

Effective conductivity of the electrolyte.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class KappaRef
Bases: *mtnlion.formula.Formula*

Bulk conductivity of the homogeneous materials.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class OpenCircuitPotential
Bases: *mtnlion.formula.Formula*

Open-circuit potential formula.

form(*arguments*, *domain*)
Evaluate the open-circuit potential equation.

class Overpotential
Bases: *mtnlion.formula.Formula*

Voltage difference between a reduction potential and the potential of the redox event.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class SolidConcentration(*legendre*)
Bases: *mtnlion.formula.Formula*

Concentration of lithium in the solid, 1D approximation using Legendre polynomials.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class SolidConcentrationBoundary
Bases: *mtnlion.formula.Formula*

This *Formula* defines the value of the lithium concentration at the surface of the solid particle.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class SolidConcentrationNeumann
Bases: *mtnlion.formula.Formula*

Nuemann boundary for the solid concentration. This *Formula* doesn't use a boundary domain due to the 1D 1D approximation.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class SolidPotential
Bases: *mtnlion.formula.Formula*

Charge conservation in the solid.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class SolidPotentialNeumann
Bases: *mtnlion.formula.Formula*

Neumann boundary for the solid potential at the anode/cathode current collector boundaries.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class StateOfCharge
Bases: *mtnlion.formula.Formula*

State of Charge (SOC) formula.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class mtnlion.models.**LithiumPlating**(*Ns*)
Bases: *mtnlion.models.isothermal.Isothermal*

Lithium plating often occurs when the manufacturer-specified upper voltage on the cell is not observed, which can cause a cell to become inoperable within a few overcharge events.

class Overpotential
Bases: *mtnlion.formula.Formula*

Voltage difference between a reduction potential and the potential of the redox event.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*

- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class SideReactionExchangeCurrentDensity

Bases: *mtnlion.formula.Formula*

The current in the absence of net electrolysis and at zero overpotential in the side reaction.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class SideReactionFlux

Bases: *mtnlion.formula.Formula*

Describes how the electrical current on an electrode depends on the electrode potential due to the side reaction.

form(*arguments*, *domain*)

Flux through the boundary of the solid.

class SideReactionOverpotential

Bases: *mtnlion.formula.Formula*

Voltage difference between a reduction potential and the potential of the redox event in the side reaction.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class mtnlion.models.**SEI**(*Ns*)

Bases: *mtnlion.models.isothermal.Isothermal*

The SEI model is an extension to the isothermal model that attempts to quantify solid-electrolyte interphase formation and growth on the negative-electrode solid particles during chargning.

class LocalMolecularFlux

Bases: *mtnlion.formula.Formula*

The total flux of the system including intercalation flux and side reaction flux.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class Overpotential
Bases: *mtnlion.formula.Formula*

Voltage difference between a reduction potential and the potential of the redox event.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class SideReactionFlux
Bases: *mtnlion.formula.Formula*

Describes how the electrical current on an electrode depends on the electrode potential due to the side reaction.

form(*arguments*, *domain*)
Flux through the boundary of the solid.

class SideReactionOverpotential
Bases: *mtnlion.formula.Formula*

Voltage difference between a reduction potential and the potential of the redox event in the side reaction.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class mtnlion.models.**Thermal**(*Ns*)
Bases: *mtnlion.models.isothermal.Isothermal*

The thermal model extends the isothermal model to allow the modeling of internal heat generation which is used to determine the temperature of the cell at any location.

class AdaptT
Bases: *mtnlion.formula.Formula*

An adapter formula to allow existing formulas to use the temperature variable T as if it were still a parameter.

form(*arguments: mtnlion.formula.Arguments*, *domain: str*) → ufl.core.expr.Expr
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class DeEff
Bases: *mtnlion.formula.Formula*

Effective diffusivity of the electrolyte.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class Ds
Bases: *mtnlion.formula.Formula*

Solid diffusivity.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class ExchangeCurrentDensity
Bases: mtnlion.models.isothermal.ExchangeCurrentDensity

The exchange current density is the current in the absence of net electrolysis and at zero overpotential.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class HeatGeneration
Bases: *mtnlion.formula.Formula*

Total heat generated in the cell.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class HeatGenerationChemical
Bases: *mtnlion.formula.Formula*

Irreversible heat generation due to chemical reactions for each chemical reaction at the interface.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.

Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class HeatGenerationEntropy

Bases: *mtnlion.formula.Formula*

Reversible heat generation due to a change in entropy for each chemical reaction at the interface

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class JouleHeatingElectrolyte1

Bases: *mtnlion.formula.Formula*

Joule heating due to electrical potential gradient in the electrolyte

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class JouleHeatingElectrolyte2

Bases: *mtnlion.formula.Formula*

Joule heating due to electrical potential gradient in the electrolyte

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class JouleHeatingSolid

Bases: *mtnlion.formula.Formula*

Joule heating due to electrical potential gradient in the solid.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*

- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class KappaDEff

Bases: *mtnlion.formula.Formula*

kappa_d effective.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class KappaEff

Bases: mtnlion.models.isothermal.KappaEff

Effective conductivity of the electrolyte.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class SigmaEff

Bases: *mtnlion.formula.Formula*

Effective conductivity (electrode-dependent parameter), represents a volume averaged conductivity of the solid matrix in a porous media in the vicinity of a given point.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class Temperature

Bases: *mtnlion.formula.Formula*

Temperature of the cell.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

Submodules

mtnlion.models.double_layer module

Isothermal model extended with double-layer capacitance

class mtnlion.models.double_layer.**DoubleLayer**(*Ns*)
Bases: *mtnlion.models.isothermal.Isothermal*

The double-layer model comes from surface science where a structure forms on the surface of a solid when exposed to a fluid. In this case, the structure that forms on the particles is an electrical charge which causes an opposing charge to build up in the electrolyte near the surface. The separation of electrical charge around the particle surface has the same behavior as a plate capacitor.

class DoubleLayer
Bases: *mtnlion.formula.Formula*

Double-layer flux.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class TotalFlux
Bases: *mtnlion.formula.Formula*

Replaces the standard intercalation flux from the isothermal model.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

mtnlion.models.isothermal module

Base Newman isothermal model

class mtnlion.models.isothermal.**Isothermal**(*num_functions*)
Bases: *mtnlion.model.Model*

The basic DFN lithium-ion model with no thermal considerations and a 1D approximation of the solid concentration.

class ElectrolyteConcentration
Bases: *mtnlion.formula.Formula*

Concentration of lithium in the electrolyte.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.

Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class ElectrolytePotential

Bases: *mtnlion.formula.Formula*

Charge conservation in the electrolyte.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class ExchangeCurrentDensity

Bases: *mtnlion.formula.Formula*

The exchange current density is the current in the absence of net electrolysis and at zero overpotential.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class IntercalationFlux

Bases: *mtnlion.formula.Formula*

Describes how the electrical current on an electrode depends on the electrode potential.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class KappaDEff

Bases: *mtnlion.formula.Formula*

kappa_d effective

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*

- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class KappaEff

Bases: *mtnlion.formula.Formula*

Effective conductivity of the electrolyte.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class KappaRef

Bases: *mtnlion.formula.Formula*

Bulk conductivity of the homogeneous materials.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class OpenCircuitPotential

Bases: *mtnlion.formula.Formula*

Open-circuit potential formula.

form(*arguments*, *domain*)

Evaluate the open-circuit potential equation.

class Overpotential

Bases: *mtnlion.formula.Formula*

Voltage difference between a reduction potential and the potential of the redox event.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class SolidConcentration(*legendre*)

Bases: *mtnlion.formula.Formula*

Concentration of lithium in the solid, 1D approximation using Legendre polynomials.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class SolidConcentrationBoundary

Bases: *mtnlion.formula.Formula*

This *Formula* defines the value of the lithium concentration at the surface of the solid particle.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class SolidConcentrationNeumann

Bases: *mtnlion.formula.Formula*

Nuemann boundary for the solid concentration. This *Formula* doesn't use a boundary domain due to the 1D 1D approximation.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class SolidPotential

Bases: *mtnlion.formula.Formula*

Charge conservation in the solid.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class SolidPotentialNeumann

Bases: *mtnlion.formula.Formula*

Neumann boundary for the solid potential at the anode/cathode current collector boundaries.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*

- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class StateOfCharge

Bases: *mtnlion.formula.Formula*

State of Charge (SOC) formula.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

mtnlion.models.lithium_plating module

Isothermal model extended with lithium plating

class mtnlion.models.lithium_plating.**LithiumPlating**(*Ns*)

Bases: *mtnlion.models.isothermal.Isothermal*

Lithium plating often occurs when the manufacturer-specified upper voltage on the cell is not observed, which can cause a cell to become inoperable within a few overcharge events.

class Overpotential

Bases: *mtnlion.formula.Formula*

Voltage difference between a reduction potential and the potential of the redox event.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class SideReactionExchangeCurrentDensity

Bases: *mtnlion.formula.Formula*

The current in the absence of net electrolysis and at zero overpotential in the side reaction.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class SideReactionFlux

Bases: *mtnlion.formula.Formula*

Describes how the electrical current on an electrode depends on the electrode potential due to the side reaction.

form(*arguments*, *domain*)
Flux through the boundary of the solid.

class SideReactionOverpotential
Bases: *mtnlion.formula.Formula*

Voltage difference between a reduction potential and the potential of the redox event in the side reaction.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

mtnlion.models.sei module

Isothermal model extended with SEI layer growth

class mtnlion.models.sei.**SEI**(*Ns*)
Bases: *mtnlion.models.isothermal.Isothermal*

The SEI model is an extension to the isothermal model that attempts to quantify solid-electrolyte interphase formation and growth on the negative-electrode solid particles during chargning.

class LocalMolecularFlux
Bases: *mtnlion.formula.Formula*

The total flux of the system including intercalation flux and side reaction flux.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class Overpotential
Bases: *mtnlion.formula.Formula*

Voltage difference between a reduction potential and the potential of the redox event.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class SideReactionFlux
Bases: *mtnlion.formula.Formula*

Describes how the electrical current on an electrode depends on the electrode potential due to the side reaction.

form(*arguments*, *domain*)
Flux through the boundary of the solid.

class SideReactionOverpotential
Bases: *mtnlion.formula.Formula*

Voltage difference between a reduction potential and the potential of the redox event in the side reaction.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

mtnlion.models.thermal module

Isothermal model extended with thermal modeling

class mtnlion.models.thermal.**Thermal**(*Ns*)
Bases: *mtnlion.models.isothermal.Isothermal*

The thermal model extends the isothermal model to allow the modeling of internal heat generation which is used to determine the temperature of the cell at any location.

class AdaptT
Bases: *mtnlion.formula.Formula*

An adapter formula to allow existing formulas to use the temperature variable T as if it were still a parameter.

form(*arguments: mtnlion.formula.Arguments*, *domain: str*) → ufl.core.expr.Expr
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class DeEff
Bases: *mtnlion.formula.Formula*

Effective diffusivity of the electrolyte.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*

- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class Ds

Bases: *mtnlion.formula.Formula*

Solid diffusivity.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class ExchangeCurrentDensity

Bases: mtnlion.models.isothermal.ExchangeCurrentDensity

The exchange current density is the current in the absence of net electrolysis and at zero overpotential.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class HeatGeneration

Bases: *mtnlion.formula.Formula*

Total heat generated in the cell.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class HeatGenerationChemical

Bases: *mtnlion.formula.Formula*

Irreversible heat generation due to chemical reactions for each chemical reaction at the interface.

form(*arguments*, *domain*)

This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class HeatGenerationEntropy
Bases: *mtnlion.formula.Formula*

Reversible heat generation due to a change in entropy for each chemical reaction at the interface

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class JouleHeatingElectrolyte1
Bases: *mtnlion.formula.Formula*

Joule heating due to electrical potential gradient in the electrolyte

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class JouleHeatingElectrolyte2
Bases: *mtnlion.formula.Formula*

Joule heating due to electrical potential gradient in the electrolyte

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class JouleHeatingSolid
Bases: *mtnlion.formula.Formula*

Joule heating due to electrical potential gradient in the solid.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class KappaDEff
Bases: *mtnlion.formula.Formula*

kappa_d effective.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class KappaEff
Bases: mtnlion.models.isothermal.KappaEff

Effective conductivity of the electrolyte.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class SigmaEff
Bases: *mtnlion.formula.Formula*

Effective conductivity (electrode-dependent parameter), represents a volume averaged conductivity of the solid matrix in a porous media in the vicinity of a given point.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

class Temperature
Bases: *mtnlion.formula.Formula*

Temperature of the cell.

form(*arguments*, *domain*)
This method must be overloaded to define the form of the *Formula*.
Parameters
- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

mtnlion.newman package

Provides Newmann cell model equations.

Submodules

mtnlion.newman.equations module

Provides isothermal Newman cell model.

mtnlion.newman.equations.**U_ocp**(*cse*, *csmax*, *uocp_str*, **_)
Evaluate the open-circuit potential equation.

mtnlion.newman.equations.**Uocp_interp**(*Uocp_neg_interp*, *Uocp_pos_interp*, *cse*, *csmax*, *utilities*)
Create an interpolator expression for the open circuit potential.

mtnlion.newman.equations.**c_e**(*jbar*, *ce*, *v*, *a_s*, *De_eff*, *t_plus*, *L*, *eps_e*, **_)
Concentration of lithium in the electrolyte.

mtnlion.newman.equations.**c_s**(*cs*, *v*, *Rs*, *Ds_ref*, **_)
Concentration of lithium in the solid.

mtnlion.newman.equations.**euler**(*y*, *y_1*, *dt*)
Create FFL expression for euler explicit/implicit time stepping.

mtnlion.newman.equations.**j**(*ce*, *cse*, *phie*, *phis*, *Uocp*, *csmax*, *ce0*, *alpha*, *k_norm_ref*, *F*, *R*, *Tref*, *degree=1*, **_)
Flux through the boundary of the solid.

mtnlion.newman.equations.**phi_e**(*jbar*, *ce*, *phie*, *v*, *kappa_eff*, *kappa_Deff*, *L*, *a_s*, *F*, **_)
Charge conservation in the electrolyte.

mtnlion.newman.equations.**phi_s**(*jbar*, *phis*, *v*, *a_s*, *F*, *sigma_eff*, *L*, **_)
Charge conservation in the solid.

mtnlion.structures package

Submodules

mtnlion.structures.mountain module

Data structure for handling mathematical operations on dictionaries.

class mtnlion.structures.mountain.**Mountain**
Bases: dict

A dictionary wrapper that allows math to be performed between like objects, or basic types. Also allows slicing into every value of the dictionary.

mtnlion.tools package

Submodules

mtnlion.tools.cache module

File caching tools.

mtnlion.tools.cache.**persist_to_npy_file**(*file_name: str, do_cache: Callable[[numpy.ndarray, Any], bool] = <function <lambda>>*) → Callable

Decorator to cache numpy arrays to a file.

Parameters

- **file_name** – name of the cached file
- **do_cache** – function to determine if caching should be performed

mtnlion.tools.comsol module

COMSOL Data Handling.

This module is designed to load 1D solution data from a Gu & Wang reference model from COMSOL as CSV files. The idea is that CSV files take a long time to load, so it is more efficient to convert the data to a binary (npz) format before processing.

COMSOL now saves it's data as a 2D matrix, however it still only uses repeated x values when the boundary solves for different values on either side. In order to normalize the repeated bounds, all bounds are check to ensure they've got repeated x values, such that the y values are duplicated.

class mtnlion.tools.comsol.**FenicsFunctions**(*data: Mapping[str, mtnlion.domain.Domain[str, numpy.ndarray]], function_space: dolfin.FunctionSpace*)

Bases: object

Handle the assignment of raw COMSOL data to FEniCS functions.

update(*time*)

Update the FEniCS function values for the current time. Interpolates the data as required.

Parameters **time** – Time at which to assign the function

mtnlion.tools.comsol.**adimensionalize_comsol_data**(*comsol_data: mtnlion.deprecated_domain.ReferenceCell, mesh: numpy.ndarray*) → Mapping[str, Mapping[str, numpy.ndarray]]

Separate a one dimensional (in time) set of cell data into three [0-1] domains.

Parameters

- **comsol_data** – data to adimensionalize
- **mesh** – destination mesh

mtnlion.tools.comsol.**collect_parameters**(*params: Mapping[str, Mapping[str, float]]*) → Tuple[Mapping[str, Mapping[str, float]], Mapping[str, float]]

Translate deprecated domains into current domain.

Parameters **params** – parameters

mtnlion.tools.comsol.**comsol_preprocessor**(*comsol_data: mtnlion.deprecated_domain.ReferenceCell, mesh: numpy.ndarray*) → Dict[str, Dict[str, scipy.interpolate.interpolate.interp1d]]

Pre-process COMSOL data to make it more useful in simulations.

Parameters

- **comsol_data** – Data to preprocess

- **mesh** – New solution mesh (adimensionalized)

mtnlion.tools.comsol.**fix_boundaries**(*mesh: numpy.ndarray, data: numpy.ndarray, boundaries: Union[float, List[int], numpy.ndarray]*) → numpy.ndarray

Adjust COMSOL's interpretation of two-sided boundaries.

When COMSOL outputs data from the reference model there are two solutions at every internal boundary, which causes COMSOL to have repeated domain values; one for the right and one for the left of the boundary. If there is only one internal boundary on the variable mesh at a given time, then a duplicate is added.

Parameters

- **mesh** – x data to use to correct the y data
- **data** – in 2D, this would be the y data
- **boundaries** – internal boundaries

Returns normalized boundaries to be consistent

mtnlion.tools.comsol.**format_data**(*raw_data: Mapping[str, numpy.ndarray], boundaries: Union[float, List[int]]*) → Optional[Mapping[str, numpy.ndarray]]

Format COMSOL stacked 1D data into a 2D matrix.

Collect single-column 2D data from COMSOL CSV format and convert into 2D matrix for easy access, where the first dimension is time and the second is the solution in space. Each solution has it's own entry in a dictionary where the key is the name of the variable. The time step size (time_integration) and mesh have their own keys.

Parameters

- **raw_data** – COMSOL formatted CSV files
- **boundaries** – internal boundary locations

Returns convenient dictionary of non-stationary solutions

mtnlion.tools.comsol.**format_name**(*name: str*) → str

Determine variable name from filename to be used in loader.collect_files.

Parameters **name** – filename

Returns variable name

mtnlion.tools.comsol.**get_standardized**(*cell: mtnlion.deprecated_domain.ReferenceCell*) → Optional[mtnlion.deprecated_engine.Mountain]

Convert COMSOL solutions to something more easily fed into FEniCS (remove repeated coordinates at boundaries).

Parameters **cell** – reference cell to remove double boundary values from

Returns Simplified solution cell

mtnlion.tools.comsol.**interp_time**(*time: numpy.ndarray, adim_data: Mapping[str, Mapping[str, numpy.ndarray]]*) → Dict[str, Dict[str, scipy.interpolate.interpolate.interp1d]]

Return one dimensional interpolation functions corresponding to a dictionary of dictionaries.

Parameters

- **time** – Times at which the data is sampled
- **adim_data** – Data to interpolate

mtnlion.tools.comsol.**load**(*filename: str*) → mtnlion.deprecated_engine.Mountain

Load COMSOL reference cell from formatted npz file.

Parameters **filename** – name of the npz file

Returns ReferenceCell

mtnlion.tools.comsol.**remove_dup_boundary**(*data: mtnlion.deprecated_domain.ReferenceCell*, *item: numpy.ndarray*) → Optional[numpy.ndarray]

Remove points at boundaries where two values exist at the same coordinate, favor electrodes over separator.

Parameters

- **data** – data in which to reference the mesh and separator indices from
- **item** – item to apply change to

Returns Array of points with interior boundaries removed

mtnlion.tools.helpers module

Assorted useful helper functions

class mtnlion.tools.helpers.**Timer**

Bases: object

Convenient class for measuring time with *with* operators.

mtnlion.tools.helpers.**build_expression_class**(*class_name: str*, *eval_expr: str*, ***kwargs*)

Create a FEniCS C++ expression from a template

Parameters

- **class_name** – Name of the expression
- **eval_expr** – Expression to evaluate
- **kwargs** – Required arguments

mtnlion.tools.helpers.**create_solution_matrices**(*num_rows: int*, *num_cols: int*, *num_solutions: int*) → Tuple[numpy.ndarray, ...]

Create numpy arrays for storing data.

Parameters

- **num_rows** –
- **num_cols** –
- **num_solutions** –

Returns

mtnlion.tools.helpers.**gather_expressions**() → Mapping[str, str]

Collect C++ based expressions :return: Dictionary of C++ strings

mtnlion.tools.helpers.**get_1d**(*func: dolfin.Function*) → numpy.ndarray

Fetch the one-dimensional solution from a FEniCS function

Parameters

- **func** – FEniCS function
- **V** – Function space

mtnlion.tools.helpers.**norm_rmse**(*estimated: numpy.ndarray*, *true: numpy.ndarray*)

Calculate the normalized RMSE

Parameters

- **estimated** – Estimated quantity
- **true** – True quantity

mtnlion.tools.helpers.**overlay_plt**(*xdata: numpy.ndarray*, *sample_time: numpy.ndarray*, *title: str*, **ydata*, *figsize: Tuple[int, int] = (15, 9)*, *linestyles: Tuple[str, str] = ('-', '–')*)

Plot solution data at multiple time slices against a comparison data set.

Parameters

- **xdata** – Common x axis
- **sample_time** – Sample times
- **title** – Title of the plot
- **ydata** – One or more sets of data in both space and time
- **figsize** – Size of the figure
- **linestyles** – style of the lines

mtnlion.tools.helpers.**save_fig**(*fig: matplotlib.figure.Figure*, *local_module_path: str*, *name: str*)

Save a figure to the given path using the given name

Parameters

- **fig** – figure to save
- **local_module_path** – path at which to save
- **name** – name of the file

mtnlion.tools.helpers.**set_domain_data**(*anode=None*, *cathode=None*, *separator=None*)

Convenience function for unpacking data into a dictionary

Parameters

- **anode** –
- **cathode** –
- **separator** –

mtnlion.tools.ldp module

Required modules.

class mtnlion.tools.ldp.**Spreadsheet**(*assumption=None*)

Bases: object

Hold spreadsheet data.

cell(*xpos*, *ypos*)

Retrieve cell information.

Parameters

- **xpos** (*integer*) – cell row
- **ypos** (*integer*) – cell column

Returns cell values and info

Return type xlrd.sheet.Cell

set_ctypes(*ctype*)
Set spreadsheet cell types. I.e. NUMBER, TEXT, etc.

Parameters **ctype** – cell types to set

set_data(*data_in*)
Set spreadsheet data using cell generators.

set_values(*values*)
Set spreadsheet cell values.

Parameters **values** (*container, e.g. list*) – values to set

size()
Retrieve the dimensions of the spreadsheet.

Returns spreadsheed dimensions

Return type tuple

mtnlion.tools.ldp.**load**(*file*, *mmap_mode=None*, *allow_pickle=True*, *fix_imports=True*, *encoding='ASCII'*)
Load numpy .npy and .npz files to an array or map of arrays respectively using np.load.

mtnlion.tools.ldp.**load_mat**(*filename*, *variable*)
Read the variable from filename.

Example

sheet = read_mat("parameter.mat", "cse")

Parameters

- **filename** (*string*) – name of the .mat file to read
- **variable** (*string*) – variable to load

Returns variable data

Return type array

mtnlion.tools.ldp.**load_params**(*sheet*, *rows=None*, *ncols=None*, *pcols=None*, *cols=None*, *nrows=None*, *prows=None*)
Read designated parameters from the sheet.

Example

sheet=read_excel('parameter_list.xlsx', 0, 'index') params["pos"] = load_params(sheet, range(55, 75), ncols=2, pcols=3)

Parameters

- **sheet** (ldp.Spreadsheet) – spreadsheet data
- **rows** (*range*) – same as nrows=prows
- **cols** (*range*) – same as ncols=pcols
- **nrows** (*int*) – cell rows to read for parameter names
- **ncols** (*int*) – cell columns to read for parameter names
- **prows** (*int*) – cell rows to read for parameter data
- **pcols** (*int*) – cell columns to read for parameter data

Returns mapping of parameter names to values

Return type dict

mtnlion.tools.ldp.**load_section**(*sheet*, *row_range=None*, *col_range=None*)
Read a 'chunk' of data from a spreadsheet.

Given a selection of rows and columns, this function will return the intersection of the two ranges. Note that the minimum value for each range is 1.

Example

spreadsheet = read_excel('parameters.xlsx', 'Parameters') cell_data = load_section(spreadsheet, [1, 3, 5], range(7, 42))

Parameters

- **sheet** (*xlrd.sheet*) – spreadsheet data
- **row_range** (*list of integers or integer*) – selected rows
- **col_range** (*list of integers or integer*) – selected columns

Returns section of sheet data

Return type array if assume=NUMBER else list

mtnlion.tools.ldp.**loadtxt**(**args*, ***kwargs*)
Load ascii files into a numpy ndarray using numpy.loadtxt.

mtnlion.tools.ldp.**read_csv**(*filename*, *start=1*, *stop=None*, *assume=1*)
Read a csv file into a *Spreadsheet*.

Example

sheet = read_csv('parameters.csv', start=9, assume=NUMBER)

Parameters

- **filename** (*string*) – name of the file to read
- **start** (*integer*) – row to start reading
- **stop** (*integer*) – row to stop reading
- **assume** (*integer*) – type of data to assume

Returns spreadsheet data

Return type *Spreadsheet*

mtnlion.tools.ldp.**read_excel**(*filename*, *sheet=None*)
Read sheet data or sheet names from an Excel workbook into a *Spreadsheet*.

Example

sheet_names = read_excel('parameter.xlsx') # returns a list of sheet names

Example

spreadsheet = read_excel('parameter.xlsx', 0) # read the first sheet

Example

spreadsheet = read_excel(parameter.xls', 'sheet_2') # load 'sheet_2'

Parameters

- **filename** (*string*) – name of the excel woorkbook to import
- **sheet** (*string or integer or None*) – spreadsheet name or index to import

Returns sheet names if sheet is None, otherwise sheet data

Return type list of strings if sheet is None, otherwise *Spreadsheet*

mtnlion.tools.loader module

This module provides utilities for loading and saving data in various file formats.

mtnlion.tools.loader.**collect_files**(*file_list: List[str], format_key: Callable = <function format_name>, loader: Callable = <function load_numpy_file>, **kwargs*) → Dict[str, numpy.ndarray]

Collect files using the provided loader.

Collect files given as a list of filenames using the function loader to load the file and the function format_key to format the variable name. :param file_list: list of filenames :param format_key: function to format variable names :param loader: function to load files :param kwargs: extra arguments to the loader :return: data dictionary

mtnlion.tools.loader.**format_name**(*name: str*) → str

Do nothing for formatting names and log the event.

Parameters **name** – filename

Returns variable name

mtnlion.tools.loader.**load_csv_file**(*filename: str, comments: str = '%', delimiter: str = ', ', d_type: type = <class 'numpy.float64'>, **kwargs*) → numpy.ndarray

Load data from a csv file. See numpy.load for additional argument options.

Parameters

- **filename** – name of the csv file
- **comments** – lines starting with a comment will be ignored
- **delimiter** – delimiting character(s)
- **d_type** – data type
- **kwargs** – additional numpy.loadtxt arguments

Returns file data

mtnlion.tools.loader.**load_numpy_file**(*filename: str, **kwargs*) → Dict[str, numpy.ndarray]

Load data from an npz file. See numpy.load for additional argument options.

Parameters

- **filename** – name of the npz file
- **kwargs** – additional numpy.load arguments

Returns data dictionary

mtnlion.tools.loader.**save_npz_file**(*filename: str, data_dict: Dict[str, numpy.ndarray], **kwargs*) → None

Save data to an npz file. See numpy.savez for additional argument options.

Parameters

- **data_dict** – data to be saved to an npz file
- **filename** – name of the npz file
- **kwargs** – additional numpy.savez arguments

6.1.2 Submodules

mtnlion.cell module

Cell domain descriptions

class mtnlion.cell.**DomainData**
Bases: tuple

Data required to minimally describe the problem domain.

boundary_measure
Alias for field number 2

domain_measure
Alias for field number 1

mesh
Alias for field number 0

class mtnlion.cell.**DomainInterface**(*mesh_grid: numpy.ndarray*)
Bases: abc.ABC

Minimal interface for describing problem domains.

class BoundaryMap(*primary_domain*, *marker*)
Bases: tuple

marker
Alias for field number 1

primary_domain
Alias for field number 0

add_domain_measure(*domain: str*, *measure: ufl.measure.Measure*, *multiple: int = 1*) → None
Set the domain measure and its multiplicity for a given domain.

Parameters

- **domain** – The domain to assign
- **measure** – The measure to apply to the domain
- **multiple** – The multiplicity to apply to the domain measure

Returns None

boundary_of(*boundary_domain: str, primary_domains: Iterable[str], measures: Iterable[ufl.measure.Measure], multiples: Iterable[int]*) → None
Create a boundary domain that is the boundary of one or more primary domains and assign a boundary measure with its corresponding multiplicity.

Parameters

- **boundary_domain** – name of the boundary domain
- **primary_domains** – names of the domains this domain is a boundary of
- **measures** – assign a boundary measure to the boundary domain
- **multiples** – assign a multiplicity to the boundary measure

Returns None

create_element(*element_type: str*, *degree: int*) → dolfin.FiniteElement
Create a FEniCS finite element. See documentation for *fem.FiniteElement*.

Parameters

- **element_type** – Type of the finite element.
- **degree** – Degree of the element

Returns *fem.FiniteElement*

create_function_space(*element: dolfin.FiniteElement*)
Create a FEniCS function space given an element. See documentation for *fem.FunctionSpace*.

Parameters **element** – Element to use to create a function space

Returns *fem.FunctionSpace*

is_boundary(*domain: str*) → bool
Deduce if a given domain is a boundary domain by whether or not is has parent domains defined.

Parameters **domain** – name of the domain to check

Returns true if domain is a boundary

class mtnlion.cell.**MeasureMap**
Bases: tuple

A mapping between measures, multiplicities, and primary (parent) domains

measure
Alias for field number 0

multiple
Alias for field number 1

primary_domain
Alias for field number 2

class mtnlion.cell.**P2D**(*mesh_grid: numpy.ndarray*)
Bases: *mtnlion.cell.DomainInterface*

Definition for a pseudo two-dimensional domain.

mtnlion.cli module

Console script for mtnlion.

mtnlion.deprecated_domain module

Domain creation/manipulation.

class mtnlion.deprecated_domain.**ReferenceCell**(*mesh: numpy.ndarray, time_mesh: numpy.ndarray, boundaries: Union[numpy.ndarray, List[float]], **kwargs*)
Bases: *mtnlion.deprecated_engine.Mountain*

Reference lithium-ion cell geometry, where the dimensions are normalized.

The x dimension is defined such that the negative electrode exists between [0, 1], the separator exists between [1, 2], and the positive electrode exists between [2, 3]. For convenience the subdomains are added onto engine.Mountain.

get_solution_in(*subspace: str*) → Union[None, mtnlion.deprecated_engine.Mountain]
Return the solution for only the given subspace.

Returns reduced solution set to only contain the given space

mtnlion.deprecated_domain.**subdomain**(*comparison: numpy.ndarray*) → slice
Find the indices of the requested subdomain, correcting for internal boundaries.

I.e. if the mesh is defined by `numpy.arange(0, 3, 0.1)` and you wish to find the subdomain `0 <= x <= 1` then you would call:

```
subdomain(mesh, x < 1)
```

Subdomain returns `x <= 1`, the reason for the exclusive less-than is to work around having repeated internal domain problems. I.e. if `x <= 1` had been used on a mesh with repeated boundaries at 1, then the subdomain would exist over both boundaries at 1.

Parameters **comparison** – list of boolean values

Returns indices of subdomain

mtnlion.deprecated_domain.**subdomains**(*mesh: numpy.ndarray, regions: List[Tuple[float, float]]*)
Find indices of given subdomains.

For example separating a domain from [0, 3] into [0, 1], [1, 2], and [2, 3] would be:

```
subdomains(np.arange(0, 3, 0.1), [(0, 1), (1, 2), (2, 3)])
```

Parameters

- **mesh** – one-dimensional list of domain values
- **regions** – two dimensional list containing multiple ranges for subdomains

Returns tuple of each subdomain indices

mtnlion.deprecated_engine module

Equation solver.

class mtnlion.deprecated_engine.**Mountain**(*mesh: numpy.ndarray, time_mesh: numpy.ndarray, boundaries: Union[numpy.ndarray, List[float]], **kwargs*)
Bases: `object`

Container for holding n-variable n-dimensional data in space and time.

filter(*index: List*, *func: Callable = <function Mountain.<lambda>>*) → Dict[str, numpy.ndarray]
Filter through dictionary to collect sections of the contained ndarrays.

Parameters

- **index** – subset of arrays to collect
- **func** – function to call on every variable in data

Returns dictionary of reduced arrays

filter_space(*index: List, func: Callable = <function Mountain.<lambda>>*) → mtnlion.deprecated_engine.Mountain
Filter the Mountain for a subset of space indices.

For example:: solution.filter_time([slice(0,5), 4]) # for 2D space solution.filter_time([0, 3, 5]) solution.filter_time(slice(step=-1))

will return the solutions from in space where x=[0, 5] and y=5, and x=[0, 3, 5], even reverse the first dimension respectively. :param index: indices or slices of space to retrieve :param func: function to call on every variable in data :return: space filtered Mountain

filter_time(*index: List*, *func: Callable = <function Mountain.<lambda>>*) → mtnlion.deprecated_engine.Mountain

Filter the Mountain for a subset of time indices.

For example:: solution.filter_time(slice(0,5)) solution.filter_time([0, 3, 5]) solution.filter_time(slice(step=-1)) solution.filter_time(numpy.where(solution.time_mesh == time) # time could be [1, 2, 3] seconds

will return the solutions from time index [0, 4], [0, 3, 5], reverse time, and fetch specific times respectively. :param index: indices or slices of time to retrieve :param func: function to call on every variable in data :return: time filtered Mountain

classmethod from_dict(*data: Dict[str, numpy.ndarray]*) → mtnlion.deprecated_engine.Mountain

Convert dictionary to SolutionData.

Parameters **data** – dictionary of formatted data

Returns consolidated simulation data

to_dict() → Dict[str, numpy.ndarray]

Retrieve dictionary of Mountain to serialize.

Returns data dictionary

mtnlion.deprecated_engine.**fetch_params**(*filename: str*) → Union[Dict[str, Dict[str, float]], None, munch.DefaultMunch]

TODO: read template from config file.

mtnlion.deprecated_engine.**find_ind**(*data: numpy.ndarray, value: Union[List[int], List[float]]*) → numpy.ndarray

Find the indices of the values given in the data.

Parameters

- **data** – data to find indices in
- **value** – values to find indices with

Returns indices of value in data

mtnlion.deprecated_engine.**find_ind_near**(*data: numpy.ndarray, value: Union[List[int], List[float]]*) → numpy.ndarray

Find the indices of the closest values given in the data.

Parameters

- **data** – data to find indices in
- **value** – values to find indices with

Returns indices of value in data

mtnlion.deprecated_engine.**rmse**(*estimated: numpy.ndarray*, *true: numpy.ndarray*) → Optional[numpy.ndarray]

Calculate the root-mean-squared error between two arrays.

Parameters

- **estimated** – estimated solution
- **true** – 'true' solution

Returns root-mean-squared error

mtnlion.domain module

Tools for defining data on multiple domains.

class mtnlion.domain.**Domain**(**args*, ***kwargs*)
Bases: *mtnlion.structures.mountain.Mountain*, typing.Mapping

Implementation of Mountain that verifies that the domains fall within VALID_DOMAINS.

domains
Return the domains contained in the object.

Returns tuple of domains

update(*[E]*, ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

class mtnlion.domain.**DomainFunction**(*func: Callable, domain_names: Union[List[str], Tuple[str]], pass_domain: Optional[bool] = False*)
Bases: object

Decorate a given function to run in each of the domains given. The function arguments will be parsed to determine if they are domain aware, and the correct domain will be selected. Otherwise, non domain aware values are passed to the function call exactly the same in every domain.

mtnlion.domain.**eval_domain**(**domain_names*, *pass_domain: Optional[bool] = False*) → Callable
Easy wrapper to decorate a function to run on given domains.

Parameters

- **domain_names** – Domains to run on
- **pass_domain** – Add the "domain" argument to the evaluation

mtnlion.element module

Element mapping utilities

class mtnlion.element.**Element**
Bases: object

Abstract base class to define the interface for "elements"

elements
A list of elements defining the solution vector.

:raises NotImplementedError if not implemented.

mapping
A dictionary defining the mapping between trial functions and elements. Elements are stored as a list in self.elements.

:raises NotImplementedError if not implemented.

class mtnlion.element.**ElementSpace**(*equation_spec: Mapping[str, mtnlion.domain.Domain[str, List[Any]]]*)
Bases: *mtnlion.element.Element*

ElementSpace is used to define the mapping between functions and their domains to a list of elements that FEniCS can recognize.

elements
A list of elements defining the solution vector.

:raises NotImplementedError if not implemented.

mapping
A dictionary defining the mapping between trial functions and elements. Elements are stored as a list in self.elements.

:raises NotImplementedError if not implemented.

shift(*offset: int*) → None
Shift the indices of the map by an offset. Usually used to merge element spaces.

Parameters **offset** – Amount to offset the map

class mtnlion.element.**MixedElementSpace**(**spaces*)
Bases: *mtnlion.element.Element*

Handles the combination of multiple element spaces into one vector.

elements
A list of elements defining the solution vector.

:raises NotImplementedError if not implemented.

mapping
A dictionary defining the mapping between trial functions and elements. Elements are stored as a list in self.elements.

:raises NotImplementedError if not implemented.

mtnlion.formula module

The *Formula* class is the basis for defining models. The formulas define the name, applicable domains, and arguments required to be able to fully assemble the formula for FFL.

exception mtnlion.formula.**ArgumentError**
Bases: Exception

Raised when an argument doesn't match the class specified

class mtnlion.formula.**Arguments**(*variables: Iterable[mtnlion.variable.Variable] = ()*, *parameters: Iterable = ()*, *lambdas: Iterable[Callable] = ()*, *time_discretization: Iterable = ()*)
Bases: object

This class is designed to serve as a data class to pass variables, parameters, formulas, lambda functions, and time discretization schemes to any formula. This class also handles the assignment of named tuples from generic tuples and performs basic error checking.

assign_argument_classes(*variables: Type[NamedTuple]*, *parameters: Type[NamedTuple]*, *lambdas: Type[NamedTuple]*, *time_discretization: Type[NamedTuple]*) → None
Assign the classes that the Iterable's should be converted to.

Parameters

- **variables** – Named tuple for variables
- **parameters** – Named tuple for parameters

- **lambdas** – Named tuple for lambdas
- **time_discretization** – Named tuple for time_discretizations

Returns None

convert() → None
Convert the internal properties to the classes defined in _classes. :return: None

pop_parameters(*parameters: Iterable[str]*) → NamedTuple
Return a tuple containing the values of the requested arguments. The requested arguments are then removed from the object.

Parameters **parameters** – List of arguments to pop

Returns Tuple

pop_time_derivatives(*time_derivatives: Iterable[str]*) → NamedTuple
Return a tuple containing the values of the requested arguments. The requested arguments are then removed from the object.

Parameters **time_derivatives** – List of arguments to pop

Returns Tuple

pop_variables(*variables: Iterable[str]*) → NamedTuple
Return a tuple containing the values of the requested arguments. The requested arguments are then removed from the object.

Parameters **variables** – List of arguments to pop

Returns Tuple

class mtnlion.formula.**FormMap**(*formula: mtnlion.formula.Formula*, *measure_map: mtnlion.cell.MeasureMap*, *eval_domain: str*, *formulation: Optional[mtnlion.domain.Domain[str, List]] = None*, *name: Optional[str] = None*)
Bases: object

A mapping between formula specifications (unformulated), the formulated FFL representation, the domain/boundary measure map, and evaluation domain.

domains
Domains that the formula is defined in

name
Name of the formula

primary_domain
Return the value of the measure's primary (parent) domain. This is usually set if the current measure is a boundary.

Returns Primary domain

variables
List of test functions in the formula

class mtnlion.formula.**Formula**(*name: Optional[str] = None*, *domains: Iterable[str] = ()*)
Bases: abc.ABC

Formula defines the name, domains of evaluation, and relevant dependencies in order to define the FFL formulation.

The method *Formula.form* must be overridden to return the fully specified form given the arguments. The arguments are defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*.

Formula.Variables: Must contain the names of variables defined in a given model *Formula.Formulas*: Must contain the names of *Formulas* defined in a given model *Formula.Parameters*: Contains the names of parameters that will be provided later *Formula.Lambdas*: Contains the names of lambda functions that will be provided later *Formula.TimeDiscretization*: Contains the names of time derivatives that will be defined later. This can be either a time stepping scheme for Rothes, or a member of the solution vector for MOL.

append_arguments(*type_: Type[NamedTuple], arguments: Iterable*) → Type[NamedTuple]
Append argument names to the given type definition.

Parameters

- **type** – Existing type
- **arguments** – New arguments to add

Returns NamedTuple

form(*arguments: mtnlion.formula.Arguments*, *domain: str*) → ufl.core.expr.Expr
This method must be overloaded to define the form of the *Formula*.

Parameters

- **arguments** – All arguments defined by overriding one or more of *Formula.Variables*, *Formula.Formulas*, *Formula.Parameters*, *Formula.Lambdas*, and *Formula.TimeDiscretization*
- **domain** – The current domain in which the function is being evaluated

Returns FFL form

formulate(*arguments: mtnlion.formula.Arguments*, *domain: str*) → ufl.core.expr.Expr
Validate arguments and perform type conversions necessary for evaluating *Formula.form*.

Parameters

- **arguments** – Arguments required to evaluate
- **domain** – Domain in which the form is evaluated

Returns FFL form

parameters
Return a tuple of argument names.

Returns Names of the required parameters

time_discretizations
Return a tuple of argument names.

Returns Names of the required time_discretizations

static typedef(*name: str*, *parameter_string: Iterable*) → Type[NamedTuple]
Creates a *namedtuple* class with a given name and parameter string.

Parameters

- **name** – Name of the new class
- **parameter_string** – string of parameter names for each tuple element. Comma or space delimited.

Returns *namedtuple*

variables
Return a tuple of argument names.

Returns Names of the required variables

mtnlion.model module

Provides a class interface for defining FEM models.

class mtnlion.model.**Model**(*minimum_rothes_order: int = 1*)
Bases: object

Abstract class for defining FEM models.

add_formula(**forms*) → mtnlion.model.Model
Add a formulae to the model :param forms: formulas to add

check() → None
Perform high-level sanity checks to ensure the model is correctly defined.

Returns None

domains
Retrieve a list of domains used in this model.

Returns Tuple of sorted domain strings

exclude_formulas(*formula_names: Iterable[str] = ()*, *formula_types: Iterable[Type] = ()*)
Exclude a formula definition from assembly.

This function will remove any object instances from the assembly list that match the given name or class definition.

Parameters

- **formula_names** – Name of the formula(s) to remove
- **formula_types** – Class of the formula(s) to remove

Returns None

formula_by_name(*name: str*) → mtnlion.formula.Formula
Retrieve a *Formula* object by name from the list of formulas.

Parameters **name** – name of the formula

Returns formula object

rename_parameter(*old: str*, *new: str*) → None
Rename a given parameter defined in the model to a new name.

Parameters

- **old** – Parameter to rename
- **new** – New name

Returns None

rename_variable(*old: str*, *new: str*) → None
Rename a given variable defined in the model to a new name.

Parameters

- **old** – Parameter to rename

- **new** – New name

Returns None

replace_formulas(**formulas*) → mtnlion.model.Model
Replace an existing formula with a new formula. The formulas are replaced by name.

Parameters **formulas** – New formula

Returns None

variable_by_name(*name: str*) → mtnlion.variable.Variable
Retrieve a *Variable* object by name from the list of variables.

Parameters **name** – name of the variable

Returns variable object

mtnlion.mtnlion module

Main module.

mtnlion.problem_space module

Tools for assigning a problem space to a given model.

class mtnlion.problem_space.**ElementAssigner**(*model: mtnlion.model.Model*)
Bases: object

Provides a set of tools to generate and assign elements to variables in a given model.

assign_elements(*element: dolfin.FiniteElement*, *domains: Iterable = ()*, *names: Iterable = ()*) → mtnlion.problem_space.ElementAssigner
Assign a given element to given domains or variable names.

Parameters

- **element** – FEniCS element to assign
- **domains** – An iterable of domain names to assign
- **names** – An iterable of variable names to assign

Returns *ElementAssigner*

check_assigned_elements() → None
Check to make sure all variables have assigned elements.

Returns None

generate_elements() → mtnlion.element.ElementSpace
Create the element space defined by the model and assigned elements.

Returns *ElementSpace*

static update_element_map(*variable: mtnlion.variable.Variable*, *element: dolfin.FiniteElement*, *domains: Iterable[str]*) → None
Update the element map for a given variable to use the given element on the specified domains.

Parameters

- **variable** – *Variable* to update
- **element** – Element to assign

- **domains** – Domains of the variable to assign

Returns None

class mtnlion.problem_space.**FunctionManager**(*mesh: dolfin.mesh*, *element_space: mtnlion.element.ElementSpace*)

Bases: object

Manages the mesh, function space, trial function vector, and test function vector.

assign(*to_fenics: dolfin.Function, from_domain: Mapping[Any, mtnlion.domain.Domain]*) → None
Assign a mixed element function from a Domain based variable.

Parameters

- **to_fenics** – Function to assign
- **from_domain** – Domain variable to assign from

function() → dolfin.Function
Create a FEniCS function defined in the mixed element function space

get_subspace(*variable_name: str*, *domain: str*) → List[dolfin.FunctionSpace]
Retrieve the subspace definition for a given variable, domain, and optionally index if the variable is an approximation.

Parameters

- **variable_name** – name of the variable
- **domain** – domain to retrieve

Returns Function Space

initialize_from_constant(*variable_name: str, constants: Iterable[dolfin.Constant], domain: str, time_index: int = 0*) → None
Initialize a variable from a constant value.

Parameters

- **variable_name** – Name of the variable
- **constants** – List of constants, lenght should be 0 if the function is not an approximation
- **domain** – Domain to apply the constant to
- **time_index** – Time index to apply the constant to

Returns None

initialize_from_expression(*variable_name: str, expressions: Iterable[dolfin.Constant], domain: str, time_index: int = 0*) → None
Initialize a variable from an expression.

Parameters

- **variable_name** – Name of the variable
- **expressions** – List of constants, lenght should be 0 if the function is not an approximation
- **domain** – Domain to apply the constant to
- **time_index** – Time index to apply the constant to

Returns None

split(*function: dolfin.Function*) → Mapping[Any, mtnlion.domain.Domain]
Split a FEniCS function into a domain structure.

Parameters **function** – FEniCS function

class mtnlion.problem_space.**ProblemSpace**(*model: mtnlion.model.Model*, *domain*, *time_discretization*)

Bases: *mtnlion.problem_space.ElementAssigner*

Defines the relationship between the model, domain, and temporal discretization schemes.

generate_variables() → mtnlion.problem_space.ProblemSpace

Given the element mapping defined by the variables in the model, generate the element space for the model and use that element space to form the test and trial function vectors. The vectors are then split into individual functions and assigned to variables.

Returns *ProblemSpace*

class mtnlion.problem_space.**ProblemSpaceAssembler**(*problem_space: mtnlion.problem_space.ProblemSpace*, *parameters*, *lambdas=()*)

Bases: object

Handles the assembly of the FFL form from the definition of the model as it relates to the problem space.

form_dependents(*form_name: str*) → Iterator

Search the list of forms to see if form_name occurs in any formula parameters.

Parameters **form_name** – Name of the form

Returns filter object

formulate() → dolfin.Form

Formulate the model equations on the problem space.

static iter_forms(*mapping: Mapping*) → Generator

Generator to iterate through a mapping of mappings.

Parameters **mapping** – Top level mapping

Returns generator

primary_forms

Fetch the formulas that are not dependent on other forms.

Returns Mapping of formulas

secondary_forms

Fetch the formulas that are dependent on other forms.

Returns Mapping of formulas

class mtnlion.problem_space.**UndefinedFormula**(*formula: str*, *domain: str*)

Bases: object

This class is a placeholder for formulas that do not exist (yet).

mtnlion.report module

Tools for better reporting of the solution

class mtnlion.report.**Report**(*solution*, *sample_times*, *split=False*, *comsol_data=None*)

Bases: object

Simplify the reporting of the gathered solutions.

plot(*local_path: Optional[str] = None*, *save: Optional[str] = None*)

Plot the stored solutions.

Parameters

- **local_path** – Path of the calling module
- **save** – true to save the plot to disk

report_rmse()
Report the normalized RMSE

mtnlion.rothes module

Rothe's method based time-stepping schemes.

class mtnlion.rothes.**Euler**(*dt*)
Bases: *mtnlion.rothes.Rothes*

First order time-stepping method.

class mtnlion.rothes.**Rothes**(*order: int*, *dt: float*)
Bases: object

Base class for defining Rothe's based time-stepping schemes.

mtnlion.solution module

Tools for access and storage of simulation solutions.

class mtnlion.solution.**Solution**(*problem_space: mtnlion.problem_space.ProblemSpaceAssembler, save_list: List[str], dae_space: dolfin.FunctionSpace*)
Bases: object

This class provides an interface for converting FEniCS functions into numpy-based solutions

get_1d(*function: dolfin.Function*, *all_funcs=False*) → Dict[str, mtnlion.domain.Domain[str, numpy.ndarray]]
Retrieve the one dimensional values from the given mixed element space function.

Parameters

- **function** –
- **all_funcs** –

Returns

static interp_time(*time: numpy.ndarray, domain: Mapping[str, mtnlion.domain.Domain[str, numpy.ndarray]]*)
Create an interpolation function for each of the solutions in storage :param time: The times at which to :param domain: Data to interpolate :return: Interpolation functions for the solution

project(*function: Mapping[str, mtnlion.domain.Domain[str, ufl.core.expr.Expr]]*)
Project an expression onto the DAE space

Parameters **function** – Non mixed-element function

save_solution(*iteration: int*, *time: float*)
Save the state of the solution vector given the iteration and current time.

Parameters

- **iteration** – current simulation iteration
- **time** – time at the same iteration

set_solution_time_steps(*num_time_steps: int*) → None
Initialize the solution storage for the specified number of iterations

Parameters **num_time_steps** – number of iterations in the simulation

mtnlion.variable module

Variables are used to define the relationship between trial functions, in both time and approximations, to test functions and the mapping between both functions and their FEniCS subdomain mapping.

exception mtnlion.variable.**SingleDomainError**
Bases: Exception

This exception is raised when the number of domains should have been reduced to one for the operation.

class mtnlion.variable.**UVMap**
Bases: tuple

Mapping between the trial functions (u), test functions (v), and the mapping of elements to the trial functions.

Trial functions are represented as a list of functions in time. Each element of the list is a dictionary of the applicable domains containing a list of related functions. Test functions are not effected by time stepping schemes, so there is no need to provide a list in time. Similarly, the element types are not allowed to change with each step in time.

element_map
Alias for field number 2

test_function
Alias for field number 1

trial_function
Alias for field number 0

class mtnlion.variable.**UVMapSingleDomain**
Bases: tuple

Mapping between the trial functions (u), test functions (v), and the mapping of elements to the trial functions restricted to a single domain.

Trial functions are represented as a list of functions in time. Each element of the list is a list of related functions. Test functions are not effected by time stepping schemes, so there is no need to provide a list in time. Similarly, the element types are not allowed to change with each step in time.

element_map
Alias for field number 2

test_function
Alias for field number 1

trial_function
Alias for field number 0

class mtnlion.variable.**UndefinedDomain**(*domain: str*)
Bases: object

Used to represent objects with an undefined domain.

class mtnlion.variable.**Variable**(*name: str, domains: Iterable[str], num_functions: int = 1*)
Bases: object

This class allows the partial definition of trial functions and test functions allowing them to be defined and used before the relationships in time, function approximations, and element mappings are defined.

***Variable*'s are able to then define the relationships in formulations without worrying about the specifics of** time discretization schemes or function approximations.

get_domain(*domain: str*) → mtnlion.variable.Variable
Retrieve a *Variable* object defined only on the given domain using *UVMapSingleDomain*.

Parameters **domain** – Domain to restrict the variable to

Returns *Variable* defined on a single domain

is_defined
Indicates whether or not the variable has been defined by checking the uv map element map.

Returns true if the elements have been defined

test(*subfunction: int = 0*, *all_funcs: bool = False*) → Union[List[Optional[ufl.indexed.Indexed]], ufl.indexed.Indexed, None]
Retrieve the test function for this *Variable*.

If no constraints are applied, the entire test function will be returned.

Parameters

- **subfunction** – Index of the desired subfunction
- **all_funcs** – return all subfunctions

Returns Desired test function

trial(*history: Optional[int] = None*, *subfunction: int = 0*, *all_funcs: bool = False*) → Union[ufl.indexed.Indexed, None, List[Union[ufl.indexed.Indexed, mtnlion.variable.UndefinedDomain]], List[List[Union[ufl.indexed.Indexed, mtnlion.variable.UndefinedDomain]]]]
Retrieve the trial function for this *Variable*.

History and subfunction constraints may be applied simultaneously. If no constraints are applied, the entire trial function will be returned.

Parameters

- **history** – Index in time to retrieve the trial function, 0 represents the current time, one is the previous time step, etc.
- **subfunction** – Index of the desired subfunction
- **all_funcs** – return all subfunctions

Returns Desired trial function

mtnlion.variable.**check_single**(*func: Callable*) → Callable
Used to wrap operations of *Variable* in order to allow variables to be used as parameters in formulas.

Parameters **func** – operation to wrap

Returns wrapped function

CHAPTER

SEVEN

CREDITS

7.1 Development Lead

- Christopher Macklen <cmacklen@uccs.edu>

7.2 Contributors

None yet. Why not be the first?

CHAPTER

EIGHT

HISTORY

To see the history of releases please refer to the GitLab releases page.

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

F

G

I

J

K

R

S

T

U

V