Battery Model Parameter Inference

Generated by Doxygen 1.8.13

Contents

1 Namespace Index

1.1 Packages

Here are the packages with brief descriptions (if available):

2 Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

3 Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

5 Namespace Documentation

5.1 models Namespace Reference

Namespaces

- DFN

    Contains a PyBaMM-compatible Doyle-Fuller-Newman (DFN) model.
- solversetup

    This file eases the setup and simulation of PyBaMM battery models.
- SPM

    Contains a PyBaMM-compatible Single-Particle Model (SPM).
- SPMe

    Contains a PyBaMM-compatible Single-Particle Model with electrolyte (SPMe).

5.2 models.DFN Namespace Reference

Contains a PyBaMM-compatible Doyle-Fuller-Newman (DFN) model.

5.2.1 Detailed Description

Contains a PyBaMM-compatible Doyle-Fuller-Newman (DFN) model.

5.3 models.solversetup Namespace Reference

This file eases the setup and simulation of PyBaMM battery models.

5.3.1 Detailed Description

This file eases the setup and simulation of PyBaMM battery models.

5.4 models.SPM Namespace Reference

Contains a PyBaMM-compatible Single-Particle Model (SPM).

5.4.1 Detailed Description

Contains a PyBaMM-compatible Single-Particle Model (SPM).

5.5 models.SPMe Namespace Reference

Contains a PyBaMM-compatible Single-Particle Model with electrolyte (SPMe).

5.5.1 Detailed Description

Contains a PyBaMM-compatible Single-Particle Model with electrolyte (SPMe).

5.6 optimization Namespace Reference

Namespaces

- run_estimation

  This file works with the parameter files in parameters.estimation to run the EP-BOLFI algorithm in optimization.EP←↩ _BOLFI.

5.7 optimization.run_estimation Namespace Reference

This file works with the parameter files in parameters.estimation to run the EP-BOLFI algorithm in optimization.EP_BOLFI.

5.7.1 Detailed Description

This file works with the parameter files in parameters.estimation to run the EP-BOLFI algorithm in optimization.EP_BOLFI.

Use it from the command line; read the user guide with ”python run_estimation.py -h” or –help.

5.8 particle_identification Namespace Reference

Namespaces

- particles
  Mask R-CNN Train on the NMC particle dataset for segmentation.

5.9 particle_identification.particles Namespace Reference

Mask R-CNN Train on the NMC particle dataset for segmentation.

5.9.1 Detailed Description

Mask R-CNN Train on the NMC particle dataset for segmentation.

If the file "mask_rcnn_particle.h5" is not present in this directory, please find it at "https://github.com/SSRL-↩
LiuGroup/LIBNet/releases".

Copyright (c) 2018 Matterport, Inc. Licensed under the MIT License (see LICENSE for details) Written by Waleed Abdulla Modified by Jizhou Li Modified by Yannick Kuhn (no modifications to function)

Usage: import the module (see Jupyter notebooks for examples), or run from the command line as such:

Train a new model starting from pre-trained COCO weights

python3 particles.py train −dataset=/path/to/particles/dataset −weights=coco

Resume training a model that you had trained earlier

python3 particles.py train −dataset=/path/to/particles/dataset −weights=last

Train a new model starting from ImageNet weights

python3 particles.py train −dataset=/path/to/particles/dataset −weights=imagenet

5.10 Python Namespace Reference

Namespaces

- analytic_impedance_visualization
- cc_cv_visualization
- dis_charge_visualisation
- estimate_ocv_from_cc_cv
- gitt_visualization
- identify_particles
- measurement_plot
- models
- ocv_visualization
- optimization
- parameters
- particle_identification
- utility
- watershed

5.11 Python.analytic_impedance_visualization Namespace Reference

Variables

- measurement = read_voltages_from_csv('../Datensätze/Simolka_EIS.csv')
- AI = AnalyticImpedance(parameters)
- int nop = 100
- ω = np.exp(np.linspace(np.log(0.001), np.log(0.31623), nop))
- int s = 1j ∗ ω
- fig
- ax
- figsize
- Z_SPM = AI.Z_SPM(s) - AI.Z_SPM_offset()
- Z_SPMe_1 = AI.Z_SPMe_1(s) - AI.Z_SPMe_1_offset()
- float Z_SPMe = 0.001 ∗ (Z_SPM + 20 ∗ Z_SPMe_1)
- legend_points
- int data_x = 80 ∗ np.array(measurement[0])
- float data_y = 0.5 ∗ np.array(measurement[1])
- lw
- ls
- label

5.11.1 Variable Documentation

5.11.1.1 AI

Python.analytic_impedance_visualization.AI = AnalyticImpedance(parameters)

5.11.1.2 ax

Python.analytic_impedance_visualization.ax

5.11.1.3 data_x

Python.analytic_impedance_visualization.data_x = 80 ∗ np.array(measurement[0])

5.11.1.4 data_y

Python.analytic_impedance_visualization.data_y = 0.5 ∗ np.array(measurement[1])

5.11.1.5 fig

Python.analytic_impedance_visualization.fig

5.11.1.6 figsize

Python.analytic_impedance_visualization.figsize

5.11.1.7 label

Python.analytic_impedance_visualization.label

5.11.1.8 legend_points

Python.analytic_impedance_visualization.legend_points

5.11.1.9 ls

Python.analytic_impedance_visualization.ls

5.11.1.10 lw

Python.analytic_impedance_visualization.lw

5.11.1.11 measurement

Python.analytic_impedance_visualization.measurement = read_voltages_from_csv('../Datensätze/Simolka_EIS.csv')

5.11.1.12 nop

int Python.analytic_impedance_visualization.nop = 100

5.11.1.13 s

int Python.analytic_impedance_visualization.s = 1j ∗ ω

5.11.1.14 Z_SPM

Python.analytic_impedance_visualization.Z_SPM = AI.Z_SPM(s) - AI.Z_SPM_offset()

5.11.1.15 Z_SPMe

Python.analytic_impedance_visualization.Z_SPMe = 0.001 ∗ (Z_SPM + 20 ∗ Z_SPMe_1)

5.11.1.16 Z_SPMe_1

Python.analytic_impedance_visualization.Z_SPMe_1 = AI.Z_SPMe_1(s) - AI.Z_SPMe_1_offset()

5.11.1.17 ω

Python.analytic_impedance_visualization.ω = np.exp(np.linspace(np.log(0.001), np.log(0.31623), nop))

5.12 Python.cc_cv_visualization Namespace Reference

Variables

- fig
- ax
- figsize
- check = cc_cv_visualization(fig, ax, dataset, max_number_of_clusters)

5.12.1 Variable Documentation

5.12.1.1 ax

Python.cc_cv_visualization.ax

5.12.1.2 check

Python.cc_cv_visualization.check = cc_cv_visualization(fig, ax, dataset, max_number_of_clusters)

5.12.1.3 fig

Python.cc_cv_visualization.fig

5.12.1.4 figsize

Python.cc_cv_visualization.figsize

5.13 Python.dis_charge_visualisation Namespace Reference

Variables

- float U_offset = 0.035
- float voltages_without_OCV = np.array(voltages_without_OCV) - U_offset
- tuple experiment = (capacities ∗ 3600 / current, voltages)
- dictionary parameters_to_try
- parameters_list
- combinations
- list solutions = [ ]
- list models = [DFN(halfcell=False, pybamm_control=False)]
- model = model.new_copy()
- fig
- ax
- figsize
- title
- ylabel
- xlabel
- feature_visualizer
- interactive_plot

5.13.1 Variable Documentation

5.13.1.1 ax

Python.dis_charge_visualisation.ax

5.13.1.2 combinations

Python.dis_charge_visualisation.combinations

5.13.1.3 experiment

tuple Python.dis_charge_visualisation.experiment = (capacities ∗ 3600 / current, voltages)

5.13.1.4 feature_visualizer

Python.dis_charge_visualisation.feature_visualizer

5.13.1.5 fig

Python.dis_charge_visualisation.fig

5.13.1.6 figsize

Python.dis_charge_visualisation.figsize

5.13.1.7 interactive_plot

Python.dis_charge_visualisation.interactive_plot

5.13.1.8 model

Python.dis_charge_visualisation.model = model.new_copy()

5.13.1.9 models

list Python.dis_charge_visualisation.models = [DFN(halfcell=False, pybamm_control=False)]

5.13.1.10 parameters_list

Python.dis_charge_visualisation.parameters_list

5.13.1.11 parameters_to_try

dictionary Python.dis_charge_visualisation.parameters_to_try

Initial value:

```
= {

}
```

5.13.1.12 solutions

list Python.dis_charge_visualisation.solutions = [ ]

5.13.1.13 title

Python.dis_charge_visualisation.title

5.13.1.14 U_offset

float Python.dis_charge_visualisation.U_offset = 0.035

5.13.1.15 voltages_without_OCV

float Python.dis_charge_visualisation.voltages_without_OCV = np.array(voltages_without_OCV) - U_offset

5.13.1.16 xlabel

Python.dis_charge_visualisation.xlabel

5.13.1.17 ylabel

Python.dis_charge_visualisation.ylabel

5.14 Python.estimate_ocv_from_cc_cv Namespace Reference

Variables

- figsize
- nrows
- ncols
- cathode_phases
- spline_smoothing

5.14.1 Variable Documentation

5.14.1.1 cathode_phases

Python.estimate_ocv_from_cc_cv.cathode_phases

5.14.1.2 figsize

Python.estimate_ocv_from_cc_cv.figsize

5.14.1.3 ncols

Python.estimate_ocv_from_cc_cv.ncols

5.14.1.4 nrows

Python.estimate_ocv_from_cc_cv.nrows

5.14.1.5 spline_smoothing

Python.estimate_ocv_from_cc_cv.spline_smoothing

5.15 Python.gitt_visualization Namespace Reference

Variables

- dictionary parameters_to_try
- parameters_list
- combinations
- list solutions = [ ]
- list models = [DFN(halfcell=True, pybamm_control=True)]
- model = model.new_copy()
- simulator
- fig
- ax
- figsize
- title
- xlabel
- ylabel

5.15.1 Variable Documentation

5.15.1.1 ax

Python.gitt_visualization.ax

5.15.1.2 combinations

Python.gitt_visualization.combinations

5.15.1.3 fig

Python.gitt_visualization.fig

5.15.1.4 figsize

Python.gitt_visualization.figsize

5.15.1.5 model

Python.gitt_visualization.model = model.new_copy()

5.15.1.6 models

list Python.gitt_visualization.models = [DFN(halfcell=True, pybamm_control=True)]

5.15.1.7 parameters_list

Python.gitt_visualization.parameters_list

5.15.1.8 parameters_to_try

dictionary Python.gitt_visualization.parameters_to_try

Initial value:

```
= {
#    "Positive electrode surface area density [m-1]":
#        (200000, 400000),
#    "Positive particle radius [m]":
#        (2e-6, 2e-5),
#    "Positive electrode diffusivity [m2.s-1]":
#        (0.8e-15, 3.085e-15, 10e-15),
#    "Positive electrode reaction rate":
#        (4e-7, 12e-7),
#    "1 + dlnf/dlnc":
#        (1, 3, 4, 5),
#    "Cation transference number":
#        (0.1, 0.25, 0.55, 0.7),
#    "Electrolyte diffusivity [m2.s-1]":
#        (1e-10, 5e-10, 5e-9),
#    "Electrolyte conductivity [S.m-1]":
#        (0.5, 2),
}
```

5.15.1.9 simulator

Python.gitt_visualization.simulator

Initial value:

```
= simulation_setup(
        model, input, parameters_entry,
        *spectral_mesh_pts_and_method(0, 20, 6, 0, 1, 2, halfcell=True),
        verbose=True
    )
```

5.15.1.10 solutions

list Python.gitt_visualization.solutions = [ ]

5.15.1.11 title

Python.gitt_visualization.title

5.15.1.12 xlabel

Python.gitt_visualization.xlabel

5.15.1.13 ylabel

Python.gitt_visualization.ylabel

5.16 Python.identify_particles Namespace Reference

Variables

- config = particles.ParticlesConfig()
- model
- by_name
- orig_image = skimage.io.imread("../Datensätze/RE_93_cathode_20mu.png")
- image = skimage.io.imread("../Datensätze/RE_93_cathode_20mu_edited2.png")
- result = model.detect([image], verbose=0)[0]
- bbox = utils.extract_bboxes(result['masks'])
- fig
- ax
- figsize
- title
- show_mask
- True
- show_bbox

5.16.1 Variable Documentation

5.16.1.1 ax

Python.identify_particles.ax

5.16.1.2 bbox

Python.identify_particles.bbox = utils.extract_bboxes(result['masks'])

5.16.1.3 by_name

Python.identify_particles.by_name

5.16.1.4 config

Python.identify_particles.config = particles.ParticlesConfig()

5.16.1.5 fig

Python.identify_particles.fig

5.16.1.6 figsize

Python.identify_particles.figsize

5.16.1.7 image

Python.identify_particles.image = skimage.io.imread("../Datensätze/RE_93_cathode_20mu_edited2.png")

5.16.1.8 model

Python.identify_particles.model

Initial value:

```
=  modellib.MaskRCNN(mode="inference", config=config,
                              model_dir="./log/")
```

5.16.1.9 orig_image

Python.identify_particles.orig_image = skimage.io.imread("../Datensätze/RE_93_cathode_20mu.png")

5.16.1.10 result

Python.identify_particles.result = model.detect([image], verbose=0)[0]

5.16.1.11 show_bbox

Python.identify_particles.show_bbox

5.16.1.12 show_mask

Python.identify_particles.show_mask

5.16.1.13 title

Python.identify_particles.title

5.16.1.14 True

Python.identify_particles.True

5.17 Python.measurement_plot Namespace Reference

Variables

- fig
- ax
- figsize
- constrained_layout

5.17.1 Variable Documentation

5.17.1.1 ax

Python.measurement_plot.ax

5.17.1.2 constrained_layout

Python.measurement_plot.constrained_layout

5.17.1.3 fig

Python.measurement_plot.fig

5.17.1.4 figsize

Python.measurement_plot.figsize

5.18 Python.models Namespace Reference

Namespaces

- analytic_impedance
- DFN
- solversetup
- SPM
- SPMe
- standard_parameters

5.19 Python.models.analytic_impedance Namespace Reference

Classes

- class AnalyticImpedance

  Analytic impedance of the SPMe.

Functions

- def complex_clip (x, lower, upper)

  Implements numpy.clip for complex values.

5.19.1 Function Documentation

5.19.1.1 complex_clip()

def Python.models.analytic_impedance.complex_clip (
x,
lower,
upper)

Implements numpy.clip for complex values.

Parameters

x	The complex values that are to be clipped.
lower	Lower bound for both real and imaginary part.
upper	Upper bound for both real and imaginary part.

Returns

Clipped complex values of x.

5.20 Python.models.DFN Namespace Reference

Classes

- class DFN

  Doyle-Fuller-Newman (DFN) model.
- class DFN_internal

  Defining equations for a Doyle-Fuller-Newman-Model.

5.21 Python.models.solversetup Namespace Reference

Functions

- def solver_setup (model, parameters, submesh_types, var_pts, spatial_methods, geometry=None, reltol=1e-6, abstol=1e-6, root_tol=1e-6, verbose=False)

  Processes the model and returns a runnable solver.
- def simulation_setup (model, input, parameters, submesh_types, var_pts, spatial_methods, geometry=None, reltol=1e-6, abstol=1e-6, root_tol=1e-6, verbose=False)

  Processes the model and returns a runnable solver.
- def auto_var_pts (x_n, x_s, x_p, r_n, r_p, y=1, z=1)

  Utility function for setting the discretization density.
- def spectral_mesh_pts_and_method (order_s_n, order_s_p, order_e, volumes_e_n=1, volumes_e_s=1, volumes_e_p=1, halfcell=False)

  Utility function for default mesh and spatial methods.

5.21.1 Function Documentation

5.21.1.1 auto_var_pts()

```
def Python.models.solversetup.auto_var_pts (
                x_n,
                x_s,
                x_p,
                r_n,
                r_p,
                y = 1,
                z = 1 )
```

Utility function for setting the discretization density.

x_n

The number of voxels for the electrolyte in the anode.

x_s

The number of voxels for the electrolyte in the separator.

x_p

The number of voxels for the electrolyte in the cathode.

r_n

The number of voxels for each anode representative particle.

r_p

The number of voxels for each cathode representative particle.

y

Used by PyBaMM for spatially resolved current collectors. Don't change the default (1) unless the model supports it.

z

Used by PyBaMM for spatially resolved current collectors. Don't change the default (1) unless the model supports it.

Returns

A discretization dictionary that can be used with PyBaMM models.

5.21.1.2 simulation_setup()

```
def Python.models.solversetup.simulation_setup (
                    model,
                    input,
                    parameters,
                    submesh_types,
                    var_pts,
                    spatial_methods,
                    geometry = None,
                    reltol = 1e-6,
                    abstol = 1e-6,
                    root_tol = 1e-6,
                    verbose = False )
```

Processes the model and returns a runnable solver.

In contrast to solversetup.solver_setup, this allows for a more verbose description of the operating conditions and should be preferred.

model

A PyBaMM model. Use one of the models in this folder.

input

A list of strings which describe the operating conditions. These exactly match the PyBaMM usage for pybamm.Experiment. Examples: "Hold at 4 V for 60 s", "Discharge at 1 A for 30 s", "Rest for 1800 s", "↩
Charge at 1 C for 30 s".

parameters

The parameters that the model requires as a dictionary. Please refer to models.standard_parameters for the names or adapt one of the examples in parameters.models.

submesh_types

A dictionary of the meshes to be used. The keys have to match the geometry names in the model. Use #spectral_mesh_and_method as reference or a shortcut.

var_pts

A dictionary giving the number of discretization volumes. Since the keys have to be special variables determined by PyBaMM, use auto_var_pts as a shortcut.

spatial_methods

A dictionary of the spatial methods to be used. The keys have to match the geometry names in the model. Use #spectral_mesh_and_method as reference or a shortcut.

geometry

The geometry of the model in dictionary form. Usually, model.default_geometry is sufficient, which is the default.

reltol The relative tolerance that the Casadi solver should

use. Default is 1e-6.

abstol The absolute tolerance that the Casadi solver should

use. Default is 1e-6.

root_tol The tolerance for rootfinding that the Casadi solver

should use. Default is 1e-6.

verbose The default (False) sets the PyBaMM flag to only

show warnings. True will show the details of preprocessing and the runtime of the solver. This applies globally, so don't set this to True if running simulations in parallel.

Returns

A pybamm.Simulation that runs the simulation when its solve() method is called. Please use solve(check↩ _model=False) as it won't work properly with redundant model checks in place.

5.21.1.3 solver_setup()

def Python.models.solversetup.solver_setup (
model,
parameters,
submesh_types,
var_pts,
spatial_methods,
geometry = None,
reltol = 1e-6,
abstol = 1e-6,
root_tol = 1e-6,
verbose = False)

Processes the model and returns a runnable solver.

model

A PyBaMM model. Use one of the models in this folder.

parameters

The parameters that the model requires as a dictionary. Please refer to models.standard_parameters for the names or adapt one of the examples in parameters.models.

submesh_types

A dictionary of the meshes to be used. The keys have to match the geometry names in the model. Use #spectral_mesh_and_method as reference or a shortcut.

var_pts

A dictionary giving the number of discretization volumes. Since the keys have to be special variables determined by PyBaMM, use auto_var_pts as a shortcut.

spatial_methods

A dictionary of the spatial methods to be used. The keys have to match the geometry names in the model. Use #spectral_mesh_and_method as reference or a shortcut.

geometry

The geometry of the model in dictionary form. Usually, model.default_geometry is sufficient, which is the default.

reltol The relative tolerance that the Casadi solver should

use. Default is 1e-6.

abstol The absolute tolerance that the Casadi solver should

use. Default is 1e-6.

root_tol The tolerance for rootfinding that the Casadi solver

should use. Default is 1e-6.

verbose The default (False) sets the PyBaMM flag to only

show warnings. True will show the details of preprocessing and the runtime of the solver. This applies globally, so don't set this to True if running simulations in parallel.

Returns

A lambda function that takes a numpy.array of timepoints to evaluate and runs the Casadi solver for those.

5.21.1.4 spectral_mesh_pts_and_method()

```
def Python.models.solversetup.spectral_mesh_pts_and_method (
                order_s_n,
                order_s_p,
                order_e,
                volumes_e_n = 1,
                volumes_e_s = 1,
                volumes_e_p = 1,
                halfcell = False )
```

Utility function for default mesh and spatial methods.

Only returns Spectral Volume mesh and spatial methods.

order_s_n

The order of the anode particles Spectral Volumes.

order_s_p

The order of the anode particles Spectral Volumes.

order_e

The order of the anode, separator and cathode electrolyte Spectral Volumes. These have to be the same, since the corresponding meshes get concatenated.

volumes_e_n

The # of Spectral Volumes to use for the anode electrolyte. This is useful to have different resolutions for each part.

volumes_e_s

The # of Spectral Volumes to use for the separator electrolyte.

volumes_e_p

The # of Spectral Volumes to use for the cathode electrolyte.

halfcell

Default is False, which sets up the mesh and spatial methods for a full-cell setup. Set it to True for a half-cell setup.

Returns

A (submesh_types, spatial_methods) tuple for PyBaMM usage.

5.22 Python.models.SPM Namespace Reference

Classes

- class SPM

  Single-Particle Model (SPM).
- class SPM_internal

  Defining equations for a Single-Particle Model (SPM).

5.23 Python.models.SPMe Namespace Reference

Classes

- class SPMe

  Single-Particle Model with electrolyte (SPMe).
- class SPMe_internal

  Defining equations for the SPMe.

5.24 Python.models.standard_parameters Namespace Reference

Functions

- def SOC$_n$_dim_init (x)

  Initial SOC of the anode.
- def SOC$_n$_init (x)

  Non-dimensionalized initial SOC of the anode.
- def SOC$_p$_dim_init (x)

  Initial SOC of the cathode.
- def SOC$_p$_init (x)

  Non-dimensionalized initial SOC of the cathode.
- def D$_e$_dim (c$_e$_dim, T_dim)

  Electrolyte diffusivity.
- def D$_e$ (c$_e$, T)

  Non-dimensionalized electrolyte diffusivity.
- def κ_e_dim (c$_e$_dim, T_dim)

  Electrolyte conductivity.
- def κ_e (c$_e$, T)

  Non-dimensionalized electrolyte conductivity.
- def t_plus_dim (c$_e$_dim)

  Transference number.
- def t_plus (c$_e$)

  Non-dimensionalized (referring to the input) transference number.
- def one_plus_dlnf_dlnc_dim (c$_e$_dim)

  Thermodynamic factor.
- def one_plus_dlnf_dlnc (c$_e$)

  Non-dimensionalized (referring to the input) thermodynamic factor.
- def D$_n$_dim (SOC$_n$, T_dim)

  Anode diffusivity.
- def D$_n$ (SOC$_n$, T)

  Non-dimensionalized anode diffusivity.
- def D$_p$_dim (SOC$_p$, T_dim)

  Cathode diffusivity.
- def D$_p$ (SOC$_p$, T)

  Non-dimensionalized cathode diffusivity.
- def i$_{sen}$_0_dim (c$_{en}$_dim, SOC$_n$_surf_dim, T_dim)

  Anode exchange current density.
- def i$_{sen}$_0 (c$_{en}$, SOC$_n$_surf, T)

  Non-dimensionalized anode exchange current density.
- def d_c$_{en}$_i$_{sen}$_0_dim (c$_{en}$_dim, SOC$_n$_surf_dim, T_dim)

  ∂ anode exchange current density / ∂ electrolyte concentration.
- def d_c$_{en}$_i$_{sen}$_0 (c$_{en}$, SOC$_n$_surf, T)

  The non-dimensionalized version of the prior variable.
- def i$_{sep}$_0_dim (c$_{ep}$_dim, SOC$_p$_surf_dim, T_dim)

  Cathode exchange current density.
- def i$_{sep}$_0 (c$_{ep}$, SOC$_p$_surf, T)

  Non-dimensionalized cathode exchange current density.
- def d_c$_{ep}$_i$_{sep}$_0_dim (c$_{ep}$_dim, SOC$_p$_surf_dim, T_dim)

  ∂ cathode exchange current density / ∂ electrolyte concentration.
- def d_c$_{ep}$_i$_{sep}$_0 (c$_{ep}$, SOC$_p$_surf, T)

The non-dimensionalized version of the prior variable.
- def $dOCV_n$_dT_dim (SOC_n)

  ∂ anode OCV / ∂ temperature.
- def $dOCV_n$_dT (SOC_n)

  Non-dimensionalized ∂ anode OCV / ∂ temperature.
- def $dOCV_n$_dT_$dSOC_n$_dim (SOC_n)

  (∂ anode OCV / ∂ temperature) / ∂ anode SOC.
- def $dOCV_n$_dT_$dSOC_n$ (SOC_n)

  Non-dimensionalized (∂ anode OCV / ∂ temperature) / ∂ anode SOC.
- def $dOCV_p$_dT_dim (SOC_p)

  ∂ cathode OCV / ∂ temperature.
- def $dOCV_p$_dT (SOC_p)

  Non-dimensionalized ∂ cathode OCV / ∂ temperature.
- def $dOCV_p$_dT_$dSOC_p$_dim (SOC_p)

  (∂ cathode OCV / ∂ temperature) / ∂ cathode SOC.
- def $dOCV_p$_dT_$dSOC_p$ (SOC_p)

  Non-dimensionalized (∂ cathode OCV / ∂ temperature) / ∂ cathode SOC.
- def OCV_n_dim (SOC_n, T_dim)

  Anode OCV.
- def OCV_n (SOC_n, T)

  Non-dimensionalized anode OCV.
- def $dOCV_n$_dim_$dSOC_n$ (SOC_n, T_dim)

  ∂ anode OCV / ∂ anode SOC.
- def $dOCV_n$_$dSOC_n$ (SOC_n, T)

  Non-dimensionalized ∂ anode OCV / ∂ anode SOC.
- def OCV_p_dim (SOC_p, T_dim)

  Cathode OCV.
- def OCV_p (SOC_p, T)

  Non-dimensionalized cathode OCV.
- def $dOCV_p$_dim_$dSOC_p$ (SOC_p, T_dim)

  ∂ cathode OCV / ∂ cathode SOC.
- def $dOCV_p$_$dSOC_p$ (SOC_p, T)

  Non-dimensionalized ∂ cathode OCV / ∂ cathode SOC.

Variables

- R = Scalar(constants.R)

  Gas constant.
- F = Scalar(constants.physical_constants["Faraday constant"][0])

  Faraday constant.
- k_B = constants.physical_constants["Boltzmann constant"][0]

  Boltzmann constant.
- q_e = constants.physical_constants["electron volt"][0]

  Electron volt.
- T_ref = Parameter("Reference temperature [K]")

  Reference temperature for non-dimensionalization.
- T_init = Parameter("Initial temperature [K]")

  Initial temperature.
- thermal_voltage = R ∗ T_ref / F
- ΔT = Scalar(1)

Typical temperature rise.

- L_n_dim = Parameter("Negative electrode thickness [m]")

  Anode thickness.
- L_s_dim = Parameter("Separator thickness [m]")

  Seperator thickness.
- L_p_dim = Parameter("Positive electrode thickness [m]")

  Cathode thickness.
- L_dim = L_n_dim + L_s_dim + L_p_dim

  Cell thickness.
- A = Parameter("Current collector perpendicular area [m2]")

  Cross-section area of the cell.
- V = Parameter("Cell volume [m3]")

  Volume of the cell.
- L_x = L_dim

  PyBaMM-compatible name for the cell thickness.
- L_y = Parameter("Electrode width [m]")

  Width of the electrode for PyBaMM compatibility.
- L_z = Parameter("Electrode height [m]")

  Height of the electrode for PyBaMM compatibility.
- C = Parameter("Typical current [A]")

  C-rate of the battery (in A).
- I_typ = C / A

  C-rate of the battery (in A/m²).
- capacity = Parameter("Cell capacity [A.h]")

  Capacity of the cell (in Ah).
- U_l = pybamm.Parameter("Lower voltage cut-off [V]")

  Lower threshold for the cell voltage.
- U_u = pybamm.Parameter("Upper voltage cut-off [V]")

  Upper threshold for the cell voltage.
- c_e_typ = pybamm.Parameter("Typical electrolyte concentration [mol.m-3]")

  Reference electrolyte concentration for non-dimensionalization.
- c_n = pybamm.Parameter("Maximum concentration in negative electrode [mol.m-3]")

  Maximum charge concentration in the anode active material.
- c_p = pybamm.Parameter("Maximum concentration in positive electrode [mol.m-3]")

  Maximum charge concentration in the cathode active material.
- σ_n_dim = pybamm.Parameter("Negative electrode conductivity [S.m-1]")

  Electronic conductivity of the anode.
- σ_p_dim = pybamm.Parameter("Positive electrode conductivity [S.m-1]")

  Electronic conductivity of the cathode.
- a_n_dim = Parameter("Negative electrode surface area to volume ratio [m-1]")

  Specific surface area of the anode.
- a_p_dim = Parameter("Positive electrode surface area to volume ratio [m-1]")

  Specific surface area of the cathode.
- R_n = Parameter("Negative particle radius [m]")

  Mean/Median radius of the anode particles.
- R_p = Parameter("Positive particle radius [m]")

  Mean/Median radius of the cathode particles.
- α_{nn} = Parameter("Negative electrode anodic charge-transfer coefficient")

  Anodic symmetry factor for the anode interface reaction.
- α_{pn} = Parameter("Negative electrode cathodic charge-transfer coefficient")

  Cathodic symmetry factor for the anode interface reaction.

- α_{np} = Parameter("Positive electrode anodic charge-transfer coefficient")

  Anodic symmetry factor for the cathode interface reaction.
- α_{pp} = Parameter("Positive electrode cathodic charge-transfer coefficient")

  Cathodic symmetry factor for the cathode interface reaction.
- β_{n}_scalar = Parameter("Negative electrode Bruggeman coefficient (electrolyte)")

  Bruggeman coefficient for the electrolyte in the anode (scalar).
- β_{es}_scalar = Parameter("Separator Bruggeman coefficient (electrolyte)")

  Bruggeman coefficient for the electrolyte in the separator (scalar).
- β_{ep}_scalar = Parameter("Positive electrode Bruggeman coefficient (electrolyte)")

  Bruggeman coefficient for the electrolyte in the cathode (scalar).
- β_{n} = pybamm.PrimaryBroadcast(β_{n}_scalar, "negative electrode")

  Bruggeman coefficient for the electrolyte in the anode (vector).
- β_{es} = pybamm.PrimaryBroadcast(β_{es}_scalar, "separator")

  Bruggeman coefficient for the electrolyte in the separator (vector).
- β_{ep} = pybamm.PrimaryBroadcast(β_{ep}_scalar, "positive electrode")

  Bruggeman coefficient for the electrolyte in the cathode (vector).
- β_{sn}_scalar = Parameter("Negative electrode Bruggeman coefficient (electrode)")

  Bruggeman coefficient for the solid part in the anode.
- β_{ss}_scalar = Parameter("Separator Bruggeman coefficient (electrode)")

  Bruggeman coefficient for the solid part in the seperator.
- β_{sp}_scalar = Parameter("Positive electrode Bruggeman coefficient (electrode)")

  Bruggeman coefficient for the solid part in the cathode.
- β_{e} = pybamm.Concatenation(β_{n}, β_{es}, β_{ep})

  Bruggeman coefficient for the electrolyte in the whole cell.
- ε_{n}_scalar = Parameter("Negative electrode porosity")

  Porosity of the anode (scalar).
- ε_{s}_scalar = Parameter("Separator porosity")

  Porosity of the separator (scalar).
- ε_{p}_scalar = Parameter("Positive electrode porosity")

  Porosity of the cathode (scalar).
- ε_{n} = pybamm.PrimaryBroadcast(ε_{n}_scalar, "negative electrode")

  Porosity of the anode (vector).
- ε_{s} = pybamm.PrimaryBroadcast(ε_{s}_scalar, "separator")

  Porosity of the separator (vector).
- ε_{p} = pybamm.PrimaryBroadcast(ε_{p}_scalar, "positive electrode")

  Porosity of the cathode (vector).
- ε = pybamm.Concatenation(ε_{n}, ε_{s}, ε_{p})

  Porosity of the whole cell.
- ε^{β} = ε**β_{e}

  Tortuosity of the whole cell.
- z_{n} = Parameter("Negative electrode electrons in reaction")

  Charge number of the anode interface reaction.
- z_{p} = Parameter("Positive electrode electrons in reaction")

  Charge number of the cathode interface reaction.
- c_{e}_dim_init = Parameter("Initial concentration in electrolyte [mol.m-3]")

  Initial electrolyte concentration.
- c_{e}_init = c_{e}_dim_init / c_{e}_typ

  Non-dimensionalized initial electrolyte concentration.
- def D_{e}_typ = D_{e}_dim(c_{e}_typ, T_ref)

  Reference electrolyte diffusivity for non-dimensionalization.
- def κ_{e}_typ = κ_{e}_dim(c_{e}_typ, T_ref)

Reference electrolyte conductivity for non-dimensionalization.

- tuple κ_e_hat = (R ∗ T_ref / F) / (I_typ ∗ L_dim / κ_e_typ)

  Thermal voltage divided by ionic resistance.
- def D_n_typ = D_n_dim(Scalar(0.5), T_ref)

  Reference anode diffusivity for non-dimensionalization.
- def D_p_typ = D_p_dim(Scalar(0.5), T_ref)

  Reference cathode diffusivity for non-dimensionalization.
- def i_{sen}_0_ref = i_{sen}_0_dim(c_e_typ, 0.5 ∗ c_n, T_ref)

  Reference anode exchange current density for non-dimensionalization.
- def i_{sep}_0_ref = i_{sep}_0_dim(c_e_typ, 0.5 ∗ c_p, T_ref)

  Reference cathode exchange current density for non-dimensionalization.
- def OCV_n_ref = OCV_n_dim(SOC_n_init(0), T_ref)

  Reference anode OCV for non-dimensionalization.
- def OCV_p_ref = OCV_p_dim(SOC_p_init(1), T_ref)

  Reference cathode OCV for non-dimensionalization.
- a_n = a_n_dim ∗ R_n

  Non-dimensionalized specific surface area of the anode.
- a_p = a_p_dim ∗ R_p

  Non-dimensionalized specific surface area of the cathode.
- tuple Q = (1 - ε_p_scalar) ∗ L_p_dim ∗ c_p ∗ z_p ∗ F ∗ A
- τ^d = F ∗ c_p ∗ L_dim / I_typ

  Discharge timescale.
- int τ_e = L_dim∗∗2 / D_e_typ

  Electrolyte diffusion timescale.
- int τ_n = R_n∗∗2 / D_n_typ

  Anode diffusion timescale.
- int τ_p = R_p∗∗2 / D_p_typ

  Cathode diffusion timescale.
- τ_{rn} = F ∗ c_n / (i_{sen}_0_ref ∗ a_n_dim)

  Anode interface reaction timescale.
- τ_{rp} = F ∗ c_p / (i_{sep}_0_ref ∗ a_p_dim)

  Cathode interface reaction timescale.
- timescale = τ^d

  Choose the discharge timescale for non-dimensionalization.
- int C_e = τ_e / τ^d

  Non-dimensionalized electrolyte diffusion timescale.
- int C_n = τ_n / τ^d

  Non-dimensionalized anode diffusion timescale.
- int C_p = τ_p / τ^d

  Non-dimensionalized cathode diffusion timescale.
- C_{rn} = τ_{rn} / τ^d

  Non-dimensionalized anode interface reaction timescale.
- C_{rp} = τ_{rp} / τ^d

  Non-dimensionalized cathode interface reaction timescale.
- γ_e = c_e_typ / c_p

  Non-dimensionalized reference electrolyte concentration.
- γ_n = c_n / c_p

  Non-dimensionalized maximum anode charge concentration.
- γ_p = c_p / c_p

  Non-dimensionalized cathode charge concentration.
- L_n = L_n_dim / L_dim

Non-dimensionalized anode thickness.

- L_s = L_s_dim / L_dim

  Non-dimensionlized separator thickness.
- L_p = L_p_dim / L_dim

  Non-dimensionalized cathode thickness.
- L_e

  Non-dimensionalized thicknesses for the whole cell.
- tuple σ_n = (thermal_voltage / (I_typ ∗ L_dim)) ∗ σ_n_dim

  Non-dimensionalized electronic conductivity of the anode.
- tuple σ_p = (thermal_voltage / (I_typ ∗ L_dim)) ∗ σ_p_dim

  Non-dimensionalized electronic conductivity of the cathode.
- I_extern_dim

  Externally applied current for galvanostatic operation (in A).
- I_extern = I_extern_dim / C

  Non-dimensionalized external current.
- n_electrodes_parallel

  Current divider.
- n_cells = Parameter("Number of cells connected in series to make a battery")

  Voltage multiplier.
- A_cc = A

  Copy of the cross-section area for PyBaMM compatibility.
- current_with_time = I_extern_dim

  Copy of the external current for PyBaMM compatibility.
- voltage_low_cut = U_l

  Copy of the lower voltage threshold for PyBaMM compatibility.
- voltage_high_cut = U_u

  Copy of the upper voltage threshold for PyBaMM compatibility.

5.24.1 Function Documentation

5.24.1.1 d_c_e_i_{sn}_0()

def Python.models.standard_parameters.d_c_e_i_{sn}_0 (

c_e,
SOC_n_surf,
T)

The non-dimensionalized version of the prior variable.

5.24.1.2 d_c_e_i_{sn}_0_dim()

def Python.models.standard_parameters.d_c_e_i_{sn}_0_dim (

c_e_dim,
SOC_n_surf_dim,
T_dim)

∂ anode exchange current density / ∂ electrolyte concentration.

5.24.1.3 d_c$_{ep}$_i$_{sep}$_0()

def Python.models.standard_parameters.d_c$_{ep}$_i$_{sep}$_0 (
c$_{ep}$,
SOC$_p$_surf,
T)

The non-dimensionalized version of the prior variable.

5.24.1.4 d_c$_{ep}$_i$_{sep}$_0_dim()

def Python.models.standard_parameters.d_c$_{ep}$_i$_{sep}$_0_dim (
c$_{ep}$_dim,
SOC$_p$_surf_dim,
T_dim)

∂ cathode exchange current density / ∂ electrolyte concentration.

5.24.1.5 dOCV$_n$_dim_dSOC$_n$()

def Python.models.standard_parameters.dOCV$_n$_dim_dSOC$_n$ (
SOC$_n$,
T_dim)

∂ anode OCV / ∂ anode SOC.

5.24.1.6 dOCV$_n$_dSOC$_n$()

def Python.models.standard_parameters.dOCV$_n$_dSOC$_n$ (
SOC$_n$,
T)

Non-dimensionalized ∂ anode OCV / ∂ anode SOC.

5.24.1.7 dOCV$_n$_dT()

def Python.models.standard_parameters.dOCV$_n$_dT (
SOC$_n$)

Non-dimensionalized ∂ anode OCV / ∂ temperature.

5.24.1.8 dOCV$_n$_dT_dim()

def Python.models.standard_parameters.dOCV$_n$_dT_dim (
SOC$_n$)

∂ anode OCV / ∂ temperature.

5.24.1.9 dOCV$_n$_dT_dSOC$_n$()

def Python.models.standard_parameters.dOCV$_n$_dT_dSOC$_n$ (
SOC$_n$)

Non-dimensionalized (∂ anode OCV / ∂ temperature) / ∂ anode SOC.

5.24.1.10 dOCV$_n$_dT_dSOC$_n$_dim()

def Python.models.standard_parameters.dOCV$_n$_dT_dSOC$_n$_dim (
SOC$_n$)

(∂ anode OCV / ∂ temperature) / ∂ anode SOC.

5.24.1.11 dOCV$_p$_dim_dSOC$_p$()

def Python.models.standard_parameters.dOCV$_p$_dim_dSOC$_p$ (
SOC$_p$,
T_dim)

∂ cathode OCV / ∂ cathode SOC.

5.24.1.12 dOCV$_p$_dSOC$_p$()

def Python.models.standard_parameters.dOCV$_p$_dSOC$_p$ (
SOC$_p$,
T)

Non-dimensionalized ∂ cathode OCV / ∂ cathode SOC.

5.24.1.13 dOCV$_p$_dT()

def Python.models.standard_parameters.dOCV$_p$_dT (
SOC$_p$)

Non-dimensionalized ∂ cathode OCV / ∂ temperature.

5.24.1.14 dOCV$_p$_dT_dim()

def Python.models.standard_parameters.dOCV$_p$_dT_dim (
SOC$_p$)

∂ cathode OCV / ∂ temperature.

5.24.1.15 dOCV$_p$_dT_dSOC$_p$()

def Python.models.standard_parameters.dOCV$_p$_dT_dSOC$_p$ (
SOC$_p$)

Non-dimensionalized (∂ cathode OCV / ∂ temperature) / ∂ cathode SOC.

5.24.1.16 dOCV$_p$_dT_dSOC$_p$_dim()

def Python.models.standard_parameters.dOCV$_p$_dT_dSOC$_p$_dim (
SOC$_p$)

(∂ cathode OCV / ∂ temperature) / ∂ cathode SOC.

5.24.1.17 D$_e$()

def Python.models.standard_parameters.D$_e$ (
c$_e$,
T)

Non-dimensionalized electrolyte diffusivity.

5.24.1.18 D$_e$_dim()

def Python.models.standard_parameters.D$_e$_dim (
c$_e$_dim,
T_dim)

Electrolyte diffusivity.

5.24.1.19 D$_n$()

def Python.models.standard_parameters.D$_n$ (
SOC$_n$,
T)

Non-dimensionalized anode diffusivity.

5.24.1.20 D_n_dim()

def Python.models.standard_parameters.D_n_dim (
SOC_n,
T_dim)

Anode diffusivity.

5.24.1.21 D_p()

def Python.models.standard_parameters.D_p (
SOC_p,
T)

Non-dimensionalized cathode diffusivity.

5.24.1.22 D_p_dim()

def Python.models.standard_parameters.D_p_dim (
SOC_p,
T_dim)

Cathode diffusivity.

5.24.1.23 i_{sn}_0()

def Python.models.standard_parameters.i_{sn}_0 (
c_e,
SOC_n_surf,
T)

Non-dimensionalized anode exchange current density.

5.24.1.24 i_{sn}_0_dim()

def Python.models.standard_parameters.i_{sn}_0_dim (
c_e_dim,
SOC_n_surf_dim,
T_dim)

Anode exchange current density.

5.24.1.25 $\mathrm{i}_{\mathrm{sep}}$_0()

def Python.models.standard_parameters.$\mathrm{i}_{\mathrm{sep}}$_0 (
c_{ep},
SOC_p_surf,
T)

Non-dimensionalized cathode exchange current density.

5.24.1.26 $\mathrm{i}_{\mathrm{sep}}$_0_dim()

def Python.models.standard_parameters.$\mathrm{i}_{\mathrm{sep}}$_0_dim (
c_{ep}_dim,
SOC_p_surf_dim,
T_dim)

Cathode exchange current density.

5.24.1.27 $\mathrm{OCV}_{\mathrm{n}}$()

def Python.models.standard_parameters.$\mathrm{OCV}_{\mathrm{n}}$ (
SOC_n,
T)

Non-dimensionalized anode OCV.

5.24.1.28 $\mathrm{OCV}_{\mathrm{n}}$_dim()

def Python.models.standard_parameters.$\mathrm{OCV}_{\mathrm{n}}$_dim (
SOC_n,
T_dim)

Anode OCV.

5.24.1.29 $\mathrm{OCV}_{\mathrm{p}}$()

def Python.models.standard_parameters.$\mathrm{OCV}_{\mathrm{p}}$ (
SOC_p,
T)

Non-dimensionalized cathode OCV.

5.24.1.30 OCV_p_dim()

def Python.models.standard_parameters.OCV_p_dim (
SOC_p,
T_dim)

Cathode OCV.

5.24.1.31 one_plus_dlnf_dlnc()

def Python.models.standard_parameters.one_plus_dlnf_dlnc (
c_e)

Non-dimensionalized (referring to the input) thermodynamic factor.

5.24.1.32 one_plus_dlnf_dlnc_dim()

def Python.models.standard_parameters.one_plus_dlnf_dlnc_dim (
c_e_dim)

Thermodynamic factor.

5.24.1.33 SOC_n_dim_init()

def Python.models.standard_parameters.SOC_n_dim_init (
x)

Initial SOC of the anode.

5.24.1.34 SOC_n_init()

def Python.models.standard_parameters.SOC_n_init (
x)

Non-dimensionalized initial SOC of the anode.

5.24.1.35 SOC_p_dim_init()

def Python.models.standard_parameters.SOC_p_dim_init (
x)

Initial SOC of the cathode.

5.24.1.36 SOC_p_init()

def Python.models.standard_parameters.SOC_p_init (
x)

Non-dimensionalized initial SOC of the cathode.

5.24.1.37 t_plus()

def Python.models.standard_parameters.t_plus (
c_e)

Non-dimensionalized (referring to the input) transference number.

5.24.1.38 t_plus_dim()

def Python.models.standard_parameters.t_plus_dim (
c_e*_dim*)

Transference number.

5.24.1.39 κ_e()

def Python.models.standard_parameters.κ_e (
c_e,
T)

Non-dimensionalized electrolyte conductivity.

5.24.1.40 κ_e_dim()

def Python.models.standard_parameters.κ_e_dim (
c_e*_dim,*
T_dim)

Electrolyte conductivity.

5.24.2 Variable Documentation

5.24.2.1 A

Python.models.standard_parameters.A = Parameter("Current collector perpendicular area [m2]")

Cross-section area of the cell.

5.24.2.2 A_cc

Python.models.standard_parameters.A_cc = A

Copy of the cross-section area for PyBaMM compatibility.

5.24.2.3 a_n

Python.models.standard_parameters.a_n = a_n_dim ∗ R_n

Non-dimensionalized specific surface area of the anode.

5.24.2.4 a_n_dim

Python.models.standard_parameters.a_n_dim = Parameter("Negative electrode surface area to volume ratio [m-1]")

Specific surface area of the anode.

5.24.2.5 a_p

Python.models.standard_parameters.a_p = a_p_dim ∗ R_p

Non-dimensionalized specific surface area of the cathode.

5.24.2.6 a_p_dim

Python.models.standard_parameters.a_p_dim = Parameter("Positive electrode surface area to volume ratio [m-1]")

Specific surface area of the cathode.

5.24.2.7 C

Python.models.standard_parameters.C = Parameter("Typical current [A]")

C-rate of the battery (in A).

5.24.2.8 capacity

Python.models.standard_parameters.capacity = Parameter("Cell capacity [A.h]")

Capacity of the cell (in Ah).

Cell capacity for calculating amperage from C-rates (in Ah).

5.24.2.9 current_with_time

Python.models.standard_parameters.current_with_time = I_extern_dim

Copy of the external current for PyBaMM compatibility.

5.24.2.10 C_{rn}

Python.models.standard_parameters.C_{rn} = τ_{rn} / τ^{d}

Non-dimensionalized anode interface reaction timescale.

5.24.2.11 C_{rp}

Python.models.standard_parameters.C_{rp} = τ_{rp} / τ^{d}

Non-dimensionalized cathode interface reaction timescale.

5.24.2.12 C_{e}

int Python.models.standard_parameters.C_{e} = τ_{e} / τ^{d}

Non-dimensionalized electrolyte diffusion timescale.

5.24.2.13 c_{e}_dim_init

Python.models.standard_parameters.c_{e}_dim_init = Parameter("Initial concentration in electrolyte [mol.m-3]")

Initial electrolyte concentration.

5.24.2.14 c_{e}_init

Python.models.standard_parameters.c_{e}_init = c_{e}_dim_init / c_{e}_typ

Non-dimensionalized initial electrolyte concentration.

5.24.2.15 c_{e}_typ

Python.models.standard_parameters.c_{e}_typ = pybamm.Parameter("Typical electrolyte concentration [mol.m-3]")

Reference electrolyte concentration for non-dimensionalization.

5.24.2.16 c_n

Python.models.standard_parameters.c_n = pybamm.Parameter("Maximum concentration in negative electrode [mol.m-3]")

Maximum charge concentration in the anode active material.

5.24.2.17 C_n

int Python.models.standard_parameters.C_n = τ_n / τ^d

Non-dimensionalized anode diffusion timescale.

5.24.2.18 c_p

Python.models.standard_parameters.c_p = pybamm.Parameter("Maximum concentration in positive electrode [mol.m-3]")

Maximum charge concentration in the cathode active material.

5.24.2.19 C_p

int Python.models.standard_parameters.C_p = τ_p / τ^d

Non-dimensionalized cathode diffusion timescale.

5.24.2.20 D_e_typ

def Python.models.standard_parameters.D_e_typ = D_e_dim(c_e_typ, T_ref)

Reference electrolyte diffusivity for non-dimensionalization.

5.24.2.21 D_n_typ

def Python.models.standard_parameters.D_n_typ = D_n_dim(Scalar(0.5), T_ref)

Reference anode diffusivity for non-dimensionalization.

5.24.2.22 D_p_typ

def Python.models.standard_parameters.D_p_typ = D_p_dim(Scalar(0.5), T_ref)

Reference cathode diffusivity for non-dimensionalization.

5.24.2.23 F

Python.models.standard_parameters.F = Scalar(constants.physical_constants["Faraday constant"][0])

Faraday constant.

5.24.2.24 I_extern

Python.models.standard_parameters.I_extern = I_extern_dim / C

Non-dimensionalized external current.

5.24.2.25 I_extern_dim

Python.models.standard_parameters.I_extern_dim

Initial value:

```
= pybamm.FunctionParameter(
  "Current function [A]",
  {"Time [s]": pybamm.t * timescale}
)
```

Externally applied current for galvanostatic operation (in A).

Please note that the variable name is important for PyBaMM comp..

5.24.2.26 I_typ

Python.models.standard_parameters.I_typ = C / A

C-rate of the battery (in A/m²).

5.24.2.27 i_{sn}_0_ref

def Python.models.standard_parameters.i_{sn}_0_ref = i_{sn}_0_dim(c_e_typ, 0.5 ∗ c_n, T_ref)

Reference anode exchange current density for non-dimensionalization.

5.24.2.28 i_{sep}_0_ref

def Python.models.standard_parameters.i_{sep}_0_ref = i_{sep}_0_dim(c_e_typ, 0.5 ∗ c_p, T_ref)

Reference cathode exchange current density for non-dimensionalization.

5.24.2.29 k_B

Python.models.standard_parameters.k_B = constants.physical_constants["Boltzmann constant"][0]

Boltzmann constant.

5.24.2.30 L_dim

Python.models.standard_parameters.L_dim = L_n_dim + L_s_dim + L_p_dim

Cell thickness.

5.24.2.31 L_x

Python.models.standard_parameters.L_x = L_dim

PyBaMM-compatible name for the cell thickness.

5.24.2.32 L_y

Python.models.standard_parameters.L_y = Parameter("Electrode width [m]")

Width of the electrode for PyBaMM compatibility.

5.24.2.33 L_z

Python.models.standard_parameters.L_z = Parameter("Electrode height [m]")

Height of the electrode for PyBaMM compatibility.

5.24.2.34 L_e

Python.models.standard_parameters.L_e

Initial value:

```
=  pybamm.Concatenation(
   pybamm.PrimaryBroadcast(Lₙ, "negative electrode"),
   pybamm.PrimaryBroadcast(Lₛ, "separator"),
   pybamm.PrimaryBroadcast(Lₚ, "positive electrode")
)
```

Non-dimensionalized thicknesses for the whole cell.

5.24.2.35 L_n

Python.models.standard_parameters.L_n = L_n_dim / L_dim

Non-dimensionalized anode thickness.

5.24.2.36 L_n_dim

Python.models.standard_parameters.L_n_dim = Parameter("Negative electrode thickness [m]")

Anode thickness.

5.24.2.37 L_p

Python.models.standard_parameters.L_p = L_p_dim / L_dim

Non-dimensionalized cathode thickness.

5.24.2.38 L_p_dim

Python.models.standard_parameters.L_p_dim = Parameter("Positive electrode thickness [m]")

Cathode thickness.

5.24.2.39 L_s

Python.models.standard_parameters.L_s = L_s_dim / L_dim

Non-dimensionlized separator thickness.

5.24.2.40 L_s_dim

Python.models.standard_parameters.L_s_dim = Parameter("Separator thickness [m]")

Seperator thickness.

5.24.2.41 n_cells

Python.models.standard_parameters.n_cells = Parameter("Number of cells connected in series to make a battery")

Voltage multiplier.

5.24.2.42 n_electrodes_parallel

Python.models.standard_parameters.n_electrodes_parallel

Initial value:

```
= Parameter("Number of electrodes connected in parallel"
                        " to make a cell")
```

Current divider.

5.24.2.43 OCV_n_ref

def Python.models.standard_parameters.OCV_n_ref = OCV_n_dim(SOC_n_init(0), T_ref)

Reference anode OCV for non-dimensionalization.

5.24.2.44 OCV_p_ref

def Python.models.standard_parameters.OCV_p_ref = OCV_p_dim(SOC_p_init(1), T_ref)

Reference cathode OCV for non-dimensionalization.

5.24.2.45 Q

tuple Python.models.standard_parameters.Q = (1 - ε_p_scalar) $*$ L_p_dim $*$ c_p $*$ z_p $*$ F $*$ A

5.24.2.46 q_e

Python.models.standard_parameters.q_e = constants.physical_constants["electron volt"][0]

Electron volt.

5.24.2.47 R

Python.models.standard_parameters.R = Scalar(constants.R)

Gas constant.

5.24.2.48 R_n

Python.models.standard_parameters.R_n = Parameter("Negative particle radius [m]")

Mean/Median radius of the anode particles.

5.24.2.49 R_p

Python.models.standard_parameters.R_p = Parameter("Positive particle radius [m]")

Mean/Median radius of the cathode particles.

5.24.2.50 T_init

Python.models.standard_parameters.T_init = Parameter("Initial temperature [K]")

Initial temperature.

5.24.2.51 T_ref

Python.models.standard_parameters.T_ref = Parameter("Reference temperature [K]")

Reference temperature for non-dimensionalization.

5.24.2.52 thermal_voltage

Python.models.standard_parameters.thermal_voltage = R * T_ref / F

5.24.2.53 timescale

Python.models.standard_parameters.timescale = τ^d

Choose the discharge timescale for non-dimensionalization.

5.24.2.54 U_u

Python.models.standard_parameters.U_u = pybamm.Parameter("Upper voltage cut-off [V]")

Upper threshold for the cell voltage.

5.24.2.55 U_l

Python.models.standard_parameters.U_l = pybamm.Parameter("Lower voltage cut-off [V]")

Lower threshold for the cell voltage.

5.24.2.56 V

Python.models.standard_parameters.V = Parameter("Cell volume [m3]")

Volume of the cell.

Left as a separate Parameter for compatibility.

5.24.2.57 voltage_high_cut

Python.models.standard_parameters.voltage_high_cut = U_u

Copy of the upper voltage threshold for PyBaMM compatibility.

5.24.2.58 voltage_low_cut

Python.models.standard_parameters.voltage_low_cut = U_l

Copy of the lower voltage threshold for PyBaMM compatibility.

5.24.2.59 z_n

Python.models.standard_parameters.z_n = Parameter("Negative electrode electrons in reaction")

Charge number of the anode interface reaction.

5.24.2.60 z_p

Python.models.standard_parameters.z_p = Parameter("Positive electrode electrons in reaction")

Charge number of the cathode interface reaction.

5.24.2.61 ΔT

Python.models.standard_parameters.ΔT = Scalar(1)

Typical temperature rise.

5.24.2.62 α_{nn}

Python.models.standard_parameters.α_{nn} = Parameter("Negative electrode anodic charge-transfer coefficient")

Anodic symmetry factor for the anode interface reaction.

5.24.2.63 α_{np}

Python.models.standard_parameters.α_{np} = Parameter("Positive electrode anodic charge-transfer coefficient")

Anodic symmetry factor for the cathode interface reaction.

5.24.2.64 α_{pn}

Python.models.standard_parameters.α_{pn} = Parameter("Negative electrode cathodic charge-transfer coefficient")

Cathodic symmetry factor for the anode interface reaction.

5.24.2.65 α_{pp}

Python.models.standard_parameters.α_{pp} = Parameter("Positive electrode cathodic charge-transfer coefficient")

Cathodic symmetry factor for the cathode interface reaction.

5.24.2.66 β_e

Python.models.standard_parameters.β_e = pybamm.Concatenation(β_n, β_{es}, β_{ep})

Bruggeman coefficient for the electrolyte in the whole cell.

5.24.2.67 β_n

Python.models.standard_parameters.β_n = pybamm.PrimaryBroadcast(β_n_scalar, "negative electrode")

Bruggeman coefficient for the electrolyte in the anode (vector).

5.24.2.68 β_n_scalar

Python.models.standard_parameters.β_n_scalar = Parameter("Negative electrode Bruggeman coefficient (electrolyte)")

Bruggeman coefficient for the electrolyte in the anode (scalar).

5.24.2.69 β_{ep}

Python.models.standard_parameters.β_{ep} = pybamm.PrimaryBroadcast(β_{ep}_scalar, "positive electrode")

Bruggeman coefficient for the electrolyte in the cathode (vector).

5.24.2.70 β_{ep}_scalar

Python.models.standard_parameters.β_{ep}_scalar = Parameter("Positive electrode Bruggeman coefficient (electrolyte)")

Bruggeman coefficient for the electrolyte in the cathode (scalar).

5.24.2.71 β_{es}

Python.models.standard_parameters.β_{es} = pybamm.PrimaryBroadcast(β_{es}_scalar, "separator")

Bruggeman coefficient for the electrolyte in the separator (vector).

5.24.2.72 β_{es}_scalar

Python.models.standard_parameters.β_{es}_scalar = Parameter("Separator Bruggeman coefficient (electrolyte)")

Bruggeman coefficient for the electrolyte in the separator (scalar).

5.24.2.73 β_{sn}_scalar

Python.models.standard_parameters.β_{sn}_scalar = Parameter("Negative electrode Bruggeman coefficient (electrode)")

Bruggeman coefficient for the solid part in the anode.

5.24.2.74 β_{sp}_scalar

Python.models.standard_parameters.β_{sp}_scalar = Parameter("Positive electrode Bruggeman coefficient (electrode)")

Bruggeman coefficient for the solid part in the cathode.

5.24.2.75 β_{ss}_scalar

Python.models.standard_parameters.β_{ss}_scalar = Parameter("Separator Bruggeman coefficient (electrode)")

Bruggeman coefficient for the solid part in the seperator.

5.24.2.76 γ_e

Python.models.standard_parameters.γ_e = c_e_typ / c_p

Non-dimensionalized reference electrolyte concentration.

5.24.2.77 γ_n

Python.models.standard_parameters.γ_n = c_n / c_p

Non-dimensionalized maximum anode charge concentration.

5.24.2.78 γ_p

Python.models.standard_parameters.γ_p = c_p / c_p

Non-dimensionalized cathode charge concentration.

5.24.2.79 ε

Python.models.standard_parameters.ε = pybamm.Concatenation(ε_n, ε_s, ε_p)

Porosity of the whole cell.

5.24.2.80 ε^β

Python.models.standard_parameters.ε^β = ε**β_e

Tortuosity of the whole cell.

5.24.2.81 ε_n

Python.models.standard_parameters.ε_n = pybamm.PrimaryBroadcast(ε_n_scalar, "negative electrode")

Porosity of the anode (vector).

5.24.2.82 ε_n_scalar

Python.models.standard_parameters.ε_n_scalar = Parameter("Negative electrode porosity")

Porosity of the anode (scalar).

5.24.2.83 ε_p

Python.models.standard_parameters.ε_p = pybamm.PrimaryBroadcast(ε_p_scalar, "positive electrode")

Porosity of the cathode (vector).

5.24.2.84 ε_p_scalar

Python.models.standard_parameters.ε_p_scalar = Parameter("Positive electrode porosity")

Porosity of the cathode (scalar).

5.24.2.85 ε_s

Python.models.standard_parameters.ε_s = pybamm.PrimaryBroadcast(ε_s_scalar, "separator")

Porosity of the separator (vector).

5.24.2.86 ε_s_scalar

Python.models.standard_parameters.ε_s_scalar = Parameter("Separator porosity")

Porosity of the separator (scalar).

5.24.2.87 κ_e_hat

tuple Python.models.standard_parameters.κ_e_hat = (R ∗ T_ref / F) / (I_typ ∗ L_dim / κ_e_typ)

Thermal voltage divided by ionic resistance.

5.24.2.88 κ_e_typ

def Python.models.standard_parameters.κ_e_typ = κ_e_dim(c_e_typ, T_ref)

Reference electrolyte conductivity for non-dimensionalization.

5.24.2.89 σ_n

tuple Python.models.standard_parameters.σ_n = (thermal_voltage / (I_typ ∗ L_dim)) ∗ σ_n_dim

Non-dimensionalized electronic conductivity of the anode.

5.24.2.90 σ_n_dim

Python.models.standard_parameters.σ_n_dim = pybamm.Parameter("Negative electrode conductivity [S.m-1]")

Electronic conductivity of the anode.

5.24.2.91 σ_p

tuple Python.models.standard_parameters.σ_p = (thermal_voltage / (I_typ ∗ L_dim)) ∗ σ_p_dim

Non-dimensionalized electronic conductivity of the cathode.

5.24.2.92 σ_p_dim

Python.models.standard_parameters.σ_p_dim = pybamm.Parameter("Positive electrode conductivity [S.m-1]")

Electronic conductivity of the cathode.

5.24.2.93 τ^d

Python.models.standard_parameters.τ^d = F ∗ c_p ∗ L_dim / I_typ

Discharge timescale.

5.24.2.94 τ_{rn}

Python.models.standard_parameters.τ_{rn} = F ∗ c_n / (i_{sn}_0_ref ∗ a_n_dim)

Anode interface reaction timescale.

5.24.2.95 τ_{rp}

Python.models.standard_parameters.τ_{rp} = F ∗ c_p / (i_{sep}_0_ref ∗ a_p_dim)

Cathode interface reaction timescale.

5.24.2.96 τ_e

int Python.models.standard_parameters.τ_e = L_dim∗∗2 / D_e_typ

Electrolyte diffusion timescale.

5.24.2.97 τ_n

int Python.models.standard_parameters.τ_n = R_n**2 / D_n_typ

Anode diffusion timescale.

5.24.2.98 τ_p

int Python.models.standard_parameters.τ_p = R_p**2 / D_p_typ

Cathode diffusion timescale.

5.25 Python.ocv_visualization Namespace Reference

Variables

- ax0
- ax1
- figsize
- ncols
- phases
- spline_smoothing
- inverted
- spline_order
- sign

5.25.1 Variable Documentation

5.25.1.1 ax0

Python.ocv_visualization.ax0

5.25.1.2 ax1

Python.ocv_visualization.ax1

5.25.1.3 figsize

Python.ocv_visualization.figsize

5.25.1.4 inverted

Python.ocv_visualization.inverted

5.25.1.5 ncols

Python.ocv_visualization.ncols

5.25.1.6 phases

Python.ocv_visualization.phases

5.25.1.7 sign

Python.ocv_visualization.sign

5.25.1.8 spline_order

Python.ocv_visualization.spline_order

5.25.1.9 spline_smoothing

Python.ocv_visualization.spline_smoothing

5.26 Python.optimization Namespace Reference

Namespaces

- EP_BOLFI
- run_estimation

5.27 Python.optimization.EP_BOLFI Namespace Reference

Functions

- def return_simulator (simulator, fixed_parameters, free_parameters, r, Q, experimental_data, feature_↩ extractor, transform_parameters={}, log_of_tried_parameters={})

  Transforms simulator output into covariance eigenvectors.
- def expectation_propagation (fixed_parameters, free_parameters, simulators, experimental_datasets, feature_extractors, covariance=None, bolfi_initial_evidence=10, bolfi_total_evidence=100, bolfi_↩ posterior_samples=20, ep_iterations=2, ep_dampener=0.5, ep_dampener_reduction_steps=1, free↩ _parameters_boundaries=None, transform_parameters={}, display_current_feature=None, show_↩ trials=False, verbose=False, seed=None)

  Estimates parameters and their uncertainties.

5.27.1 Function Documentation

5.27.1.1 expectation_propagation()

def Python.optimization.EP_BOLFI.expectation_propagation (
fixed_parameters,
free_parameters,
simulators,
experimental_datasets,
feature_extractors,
covariance = None,
bolfi_initial_evidence = 10,
bolfi_total_evidence = 100,
bolfi_posterior_samples = 20,
ep_iterations = 2,
ep_dampener = 0.5,
ep_dampener_reduction_steps = 1,
free_parameters_boundaries = None,
transform_parameters = {},
display_current_feature = None,
show_trials = False,
verbose = False,
seed = None)

Estimates parameters and their uncertainties.

fixed_parameters

Dictionary of parameters that stay fixed. Their values are the values to which the parameters are fixed.

free_parameters

Dictionary of parameters which shall be inferred. The keys are those variables which are recognized by the 'simulators' when being passed as keys in a dictionary. The values are their initial guesses / expected values. Optionally, the values may be a 2-tuple where the second entry would be the variance of that parameter. For finer tuning and covariances, use the argument 'covariance' (will take precedence).

simulators

A list of functions that take as only argument a dictionary of combined 'fixed_parameters' and 'free_↩ parameters'. Most of the time, one function will be sufficient. Additional functions simply allow for simpler, thus more efficient simulators which each give a subset of the total experimental methods.

experimental_datasets

A list where each entry has the same structure as the corresponding 'simulators' entry's output. They represent the real experimental data. The simulators' output will be compared against this data.

feature_extractors

A list of functions which each take the output of the corresponding 'simulators' entry and return a list of numbers, which typically are reduced features of a simulation.

covariance

Initial covariance of the initial guesses. Has to be a symmetric matrix (list of list or numpy matrix). A reasonable simple choice is to have a diagonal matrix and set the standard deviation σ_i of each parameter to half of the distance between initial guess and biggest/smallest value that shall be tried. The diagonal entries then are σ_i^2 => 95% coverage.

bolfi_initial_evidence

Number of evidence samples BOLFI will take for each feature before using BO sampling. Default: 10.

bolfi_total_evidence

Number of evidence samples BOLFI will take for each feature in total (including initial). Default: 100.

bolfi_posterior_samples

Number of samples BOLFI will take from the posterior distribution. These are then used to fit a Gaussian to the posterior. Typically less expensive than the evidence, but with ill-conditioned posteriors, may take longer. Convergence of fit scales with $1/\sqrt{n}$. Default: 20.

ep_iterations

The number of iterations of the expectation propagation algorithm, i.e. the number of passes over each feature. Default is 2.

ep_dampener

The linear combination factor of the posterior calculated by BOLFI and the pseudo-prior. 0 means no dampening, i.e. the pseudo-prior gets replaced by the posterior, and for values up to one that fraction of the pseudo-prior remains. Default is 0.

ep_dampener_reduction_steps

Number of iterations after which the 'ep_dampener' got reduced to 0. In each iteration, an equal fraction of it gets subtracted. Default is 1.

free_parameters_boundaries

Optional hard boundaries of the space in which optimal parameters are searched for. They are given as a dictionary where the keys are the parameter names and the values are a 2-tuple with the left and right boundaries. This 'search space' is the model parameter space after applying the transforms defined in 'transform↵
_parameters'. If None are given, only the soft boundaries that result from the spread of the (co-)variances around the guesses apply. If a hard boundary for one parameter is given, it must be given for all parameters in 'free_parameters'.

transform_parameters

Optional transformations between the parameter space that is used for searching for optimal parameters and the battery model parameters. The keys are the names of the free parameters; if one is missing, no transformation takes place. The values are a 2-tuple where the first entry is a function taking the search space parameter and returning the battery model parameter. The second entry does this in reverse. Guesses and (co-)variances refer to the resulting search space! For convenience, the value may also be one of the following:

- 'none' => (identity, identity)
- 'log' => (exp, log)

display_current_feature

A list of functions. Each corresponds to a feature extractor with the same index. Given an index of the array of its features, this returns a short description of it. If None is given, only the index will be shown in the console.

show_trials

True shows the log of tried parameters live. Default is False.

verbose

True shows verbose error messages and logs of the estimation process. Default is False.

seed

Optional seed that is used in the RNG. If None is given, the results will be slightly different each time, "actually" random.

5.27.1.2 return_simulator()

```
def Python.optimization.EP_BOLFI.return_simulator (
                    simulator,
                    fixed_parameters,
                    free_parameters,
                    r,
                    Q,
                    experimental_data,
                    feature_extractor,
                    transform_parameters = {},
                    log_of_tried_parameters = {} )
```

Transforms simulator output into covariance eigenvectors.

simulator

A function that takes a dictionary of parameters and returns the simulation output in any format. 'feature↩ _extractor' will have to convert it into a list.

fixed_parameters

Dictionary of parameters that stay fixed. Their values are the values to which the parameters are fixed.

free_parameters

List of parameters which shall be inferred. They are given as those variables which are recognized by the 'simulators' when being passed as keys in a dictionary.

r

'Q' times the mean of the free parameters.

Q

Inverse covariance matrix of free parameters, i.e. precision. It is used to transform the free parameters given to 'simulator' into the ones used in the battery model. Most notably, these univariate standard normal distributions get transformed into a multivariate normal distribution corresponding to Q and r.

experimental_data

The experimental data to be fitted against. It has to have the same structure as the 'simulator' output.

feature_extractor

A function that takes the output of 'simulator' or the 'experimental_data' and returns a list of features. These have to be orderable, i.e. implement '<'.

transform_parameters

Optional transformations between the parameter space that is used for searching for optimal parameters and the battery model parameters. 'Q' and 'r' define a normal distribution in that search space. The keys are the names of the free parameters. The values are a 2-tuple where the first entry is a function taking the search space parameter and returning the battery model parameter. The second entry does this in reverse.

log_of_tried_parameters

If you pass a dictionary to this argument, it will be appended by the log of all parameters that were passed to the simulator. The keys are/will be the parameter names and the values are lists of the tried parameters. If you use multiple simulators, use the same dictionary each time to get the full log. In parallel processing, combine them at the end.

Returns

A 6-tuple with the following contents (counting from 0): 0: a simulator that can be used with ELFI. 1↩
: features of the experimental data. 2: function that transforms the raw samples (which reside in the search space) into the battery model parameter∗ samples. ∗If 'transform_parameters' contains transformations, those have been applied to this output. 3: function that reverses this transformation. 4: function that applies the transformations of 'transform_parameters' to a dictionary of parameters. 5: function that reverses those transformations.

5.28 Python.optimization.run_estimation Namespace Reference

Classes

- class SmartFormatter

  Initialize the parser for the command-line parameters.

Functions

- def neg_int (value)

  Define a method to reject negative integers.

Variables

- parser = argparse.ArgumentParser(formatter_class=SmartFormatter)

  The command-line input parser.
- args = parser.parse_args()

  Parses the command-line arguments.
- opt = run_path(args.estimator_parameters)

  Contains the global variables of the run file.

5.28.1 Function Documentation

5.28.1.1 neg_int()

def Python.optimization.run_estimation.neg_int (
    value)

Define a method to reject negative integers.

value

Any number type that is castable to int.

5.28.2 Variable Documentation

5.28.2.1 args

Python.optimization.run_estimation.args = parser.parse_args()

Parses the command-line arguments.

5.28.2.2 opt

Python.optimization.run_estimation.opt = run_path(args.estimator_parameters)

Contains the global variables of the run file.

5.28.2.3 parser

Python.optimization.run_estimation.parser = argparse.ArgumentParser(formatter_class=SmartFormatter)

The command-line input parser.

5.29 Python.parameters Namespace Reference

Namespaces

- estimation
- models

5.30 Python.parameters.estimation Namespace Reference

Namespaces

- cccv_inhouse_pouch_cell
- cccv_samsung
- discharge_thick_electrodes
- gitt_timo

5.31 Python.parameters.estimation.cccv_inhouse_pouch_cell Namespace Reference

Variables

- E_0_g = np.array([0.35973, 0.17454, 0.12454, 0.081957])

  Fitted plateau voltages of a graphite anode.
- γUeminus1_g = np.array([-0.33144, 8.9434e-3, 7.2404e-2, 6.7789e-2])

  Fitted plateau inverse widths of a graphite anode.
- a_g = a_fit(γUeminus1_g)

  Fitted plateau inverse widths (transformed) of a graphite anode.
- Δx_g = np.array([8.041e-2, 0.23299, 0.29691, 0.39381])

  Fitted plateau SOC fractions of a graphite anode.
- list anode = [p[i] for i in range(4) for p in [E_0_g, a_g, Δx_g]]

  Fit parameters of a graphite anode OCV curve.
- int cathode_phases = 4

  Number of plateaus that are to be fitted to the OCV curve.
- int smoothing_factor = 1e-7

  Lever for adjusting the smoothing spline for the OCV curves.
- complete_dataset

  The experiment dataset, stored as a Cycling_Information object.
- int states_per_cycle = 3

  Number of dataset segments per "cycle" for plotting.
- indices
- dataset = complete_dataset.subslice(81, -2)

  Subset of interest of the complete experiment dataset.
- int max_number_of_clusters = 4

  Maximum number of clusters for the automated labelling.
- charge = dataset.subslice(0, None, states_per_cycle)

  CC charge curves.
- cv = dataset.subslice(1, None, states_per_cycle)

  CV segments between charge and discharge curves.
- discharge = dataset.subslice(states_per_cycle - 1, None, states_per_cycle)

  CC discharge curves.

5.31.1 Variable Documentation

5.31.1.1 a_g

Python.parameters.estimation.cccv_inhouse_pouch_cell.a_g = a_fit(γUeminus1_g)

Fitted plateau inverse widths (transformed) of a graphite anode.

5.31.1.2 anode

list Python.parameters.estimation.cccv_inhouse_pouch_cell.anode = [p[i] for i in range(4) for p in [E_0_g, a_g, Δx_g]]

Fit parameters of a graphite anode OCV curve.

5.31.1.3 cathode_phases

int Python.parameters.estimation.cccv_inhouse_pouch_cell.cathode_phases = 4

Number of plateaus that are to be fitted to the OCV curve.

5.31.1.4 charge

Python.parameters.estimation.cccv_inhouse_pouch_cell.charge = dataset.subslice(0, None, states_per_cycle)

CC charge curves.

5.31.1.5 complete_dataset

Python.parameters.estimation.cccv_inhouse_pouch_cell.complete_dataset

Initial value:

```
= read_channels_from_measurement_system(
  '../Datensätze/L_ABE078_BP_1.094', 'iso-8859-1', 2,
  headers={
      3: "t [s]",
      7: "I [A]",
      8: "U [V]"
  }, delimiter='\t', decimal='.', type="", segment_column=2,
  current_sign_correction={
      "S": 1.0, "P": 1.0, "R": 1.0, "C": 1.0, "D": -1.0, "O": 1.0,
    }, correction_column=9
)
```

The experiment dataset, stored as a Cycling_Information object.

5.31.1.6 cv

Python.parameters.estimation.cccv_inhouse_pouch_cell.cv = dataset.subslice(1, None, states_per_cycle)

CV segments between charge and discharge curves.

5.31.1.7 dataset

Python.parameters.estimation.cccv_inhouse_pouch_cell.dataset = complete_dataset.subslice(81, -2)

Subset of interest of the complete experiment dataset.

dataset = complete_dataset.subarray(#[5, 6 [13, 14, 15, 19, 20, 21, 73, 74, 75])

5.31.1.8 discharge

Python.parameters.estimation.cccv_inhouse_pouch_cell.discharge = dataset.subslice(states_per_cycle - 1, None, states_per_cycle)

CC discharge curves.

5.31.1.9 E_0_g

Python.parameters.estimation.cccv_inhouse_pouch_cell.E_0_g = np.array([0.35973, 0.17454, 0.12454, 0.081957])

Fitted plateau voltages of a graphite anode.

5.31.1.10 indices

Python.parameters.estimation.cccv_inhouse_pouch_cell.indices

5.31.1.11 max_number_of_clusters

int Python.parameters.estimation.cccv_inhouse_pouch_cell.max_number_of_clusters = 4

Maximum number of clusters for the automated labelling.

5.31.1.12 smoothing_factor

int Python.parameters.estimation.cccv_inhouse_pouch_cell.smoothing_factor = 1e-7

Lever for adjusting the smoothing spline for the OCV curves.

5.31.1.13 states_per_cycle

int Python.parameters.estimation.cccv_inhouse_pouch_cell.states_per_cycle = 3

Number of dataset segments per ”cycle” for plotting.

5.31.1.14 Δx_g

Python.parameters.estimation.cccv_inhouse_pouch_cell.Δx_g = np.array([8.041e-2, 0.23299, 0.29691, 0.39381])

Fitted plateau SOC fractions of a graphite anode.

5.31.1.15 γUeminus1_g

Python.parameters.estimation.cccv_inhouse_pouch_cell.γUeminus1_g = np.array([-0.33144, 8.9434e-3, 7.2404e-2, 6.7789e-2])

Fitted plateau inverse widths of a graphite anode.

5.32 Python.parameters.estimation.cccv_samsung Namespace Reference

Variables

- E_0_g = np.array([0.35973, 0.17454, 0.12454, 0.081957])
  Fitted plateau voltages of a graphite anode.
- γUeminus1_g = np.array([-0.33144, 8.9434e-3, 7.2404e-2, 6.7789e-2])
  Fitted plateau inverse widths of a graphite anode.
- a_g = a_fit(γUeminus1_g)
  Fitted plateau inverse widths (transformed) of a graphite anode.
- Δx_g = np.array([8.041e-2, 0.23299, 0.29691, 0.39381])
  Fitted plateau SOC fractions of a graphite anode.
- list anode = [p[i] for i in range(4) for p in [E_0_g, a_g, Δx_g]]
  Fit parameters of a graphite anode OCV curve.
- int cathode_phases = 6
  Number of plateaus that are to be fitted to the OCV curve.
- int smoothing_factor = 1e-7
  Lever for adjusting the smoothing spline for the OCV curves.
- complete_dataset
  The experiment dataset, stored as a Cycling_Information object.
- int states_per_cycle = 4
  Number of dataset segments per ”cycle” for plotting.
- indices
- dataset = complete_dataset
  Subset of interest of the complete experiment dataset.
- int max_number_of_clusters = 12
  Maximum number of clusters for the automated labelling.
- charge = dataset.subslice(0, None, states_per_cycle)
  CC charge curves.
- cv = dataset.subslice(1, None, states_per_cycle // 2)
  CV segments between charge and discharge curves.
- discharge = dataset.subslice(states_per_cycle - 1, None, states_per_cycle)
  CC discharge curves.

5.32.1 Variable Documentation

5.32.1.1 a_g

Python.parameters.estimation.cccv_samsung.a_g = a_fit(γUeminus1_g)

Fitted plateau inverse widths (transformed) of a graphite anode.

5.32.1.2 anode

list Python.parameters.estimation.cccv_samsung.anode = [p[i] for i in range(4) for p in [E_0_g, a_g, Δx_g]]

Fit parameters of a graphite anode OCV curve.

5.32.1.3 cathode_phases

int Python.parameters.estimation.cccv_samsung.cathode_phases = 6

Number of plateaus that are to be fitted to the OCV curve.

5.32.1.4 charge

Python.parameters.estimation.cccv_samsung.charge = dataset.subslice(0, None, states_per_cycle)

CC charge curves.

5.32.1.5 complete_dataset

Python.parameters.estimation.cccv_samsung.complete_dataset

Initial value:

```
= read_channels_from_measurement_system(
  '../Datensätze/E_RE_93cyc.094', 'iso-8859-1', 2,
  headers={
      3: "t [s]",
      7: "I [A]",
      8: "U [V]"
  }, delimiter='\t', decimal='.', type="", segment_column=2,
  current_sign_correction={
      "S": 1.0, "P": 1.0, "R": 1.0, "C": 1.0, "D": -1.0
   }, correction_column=9#, max_number_of_lines=10000
)
```

The experiment dataset, stored as a Cycling_Information object.

5.32.1.6 cv

Python.parameters.estimation.cccv_samsung.cv = dataset.subslice(1, None, states_per_cycle // 2)

CV segments between charge and discharge curves.

5.32.1.7 dataset

Python.parameters.estimation.cccv_samsung.dataset = complete_dataset

Subset of interest of the complete experiment dataset.

5.32.1.8 discharge

Python.parameters.estimation.cccv_samsung.discharge = dataset.subslice(states_per_cycle - 1, None, states_per_cycle)

CC discharge curves.

5.32.1.9 E_0_g

Python.parameters.estimation.cccv_samsung.E_0_g = np.array([0.35973, 0.17454, 0.12454, 0.081957])

Fitted plateau voltages of a graphite anode.

5.32.1.10 indices

Python.parameters.estimation.cccv_samsung.indices

5.32.1.11 max_number_of_clusters

int Python.parameters.estimation.cccv_samsung.max_number_of_clusters = 12

Maximum number of clusters for the automated labelling.

5.32.1.12 smoothing_factor

int Python.parameters.estimation.cccv_samsung.smoothing_factor = 1e-7

Lever for adjusting the smoothing spline for the OCV curves.

5.32.1.13 states_per_cycle

int Python.parameters.estimation.cccv_samsung.states_per_cycle = 4

Number of dataset segments per ”cycle” for plotting.

5.32.1.14 Δx_g

Python.parameters.estimation.cccv_samsung.Δx_g = np.array([8.041e-2, 0.23299, 0.29691, 0.39381])

Fitted plateau SOC fractions of a graphite anode.

5.32.1.15 γUeminus1_g

Python.parameters.estimation.cccv_samsung.γUeminus1_g = np.array([-0.33144, 8.9434e-3, 7.2404e-2, 6.7789e-2])

Fitted plateau inverse widths of a graphite anode.

5.33 Python.parameters.estimation.discharge_thick_electrodes Namespace Reference

Functions

- def simulator (trial_parameters)

    A simulator compatible with EP-BOLFI.
- def get_features (dataset)

    Defines the features.
- def name_of_dis_charge_features (index)

    Gives the descriptions of the features by index.
- def feature_visualizer (args)

    May be used for plotting purposes.

Variables

- l = np.log(10)

    Use the natural logarithm of 10 for scalings in orders of magnitude.
- dictionary free_parameters

    Parameters that are to be estimated.
- dictionary transform_parameters

    Transformations/Scalings of the parameter ranges.
- dictionary free_parameters_boundaries

    Bounds in which a parameter set is searched for.
- list E_0 = [3.9074103413415218, 3.7586234939671175, 3.7517405949585427]
- list a = [-0.11927628084606558, -1.8526410797182222, -0.4921205974305653]
- list Δx = [0.6837809843229503, 0.021811690524785213, 0.29440732515226453]
- list cathode = [par[i] for i in range(len(E_0)) for par in [E_0, a, Δx]]

    OCV fit parameters for the cathode.
- complete_dataset

The experimental data to be used for inference.

- float current = 0.00231

  The current that is applied during the experiment.
- list input = ["Discharge at " + str(current) + " A for 360 s",]

  The experiment as a simulation input for use with simulation_setup.
- capacities = np.array(complete_dataset[0])

  The SOC of the cell represented as total discharged capacity (in Ah).
- voltages = np.array(complete_dataset[1])

  The voltages at those SOCs.
- int starting_SOC = 1 - OCV_fit_function(voltages[0], ∗cathode)

  The initial SOC of the cathode as guessed by its OCV fit function.
- voltages_without_OCV

  The isolated overpotential of the experiment.
- smoothed_voltages = smooth_fit(capacities, voltages)

  Limit the detrimental effect of random fluctuations by smoothing.
- list indices = [int(i) for i in len(capacities) ∗ np.array([0.0, 0.5, 0.9])]

  Chooses the points in the discharge curve used for comparison.
- list plateau_capacities = [capacities[i] for i in indices]

  The SOC points for comparison (in Ah).
- list plateau_voltages = [smoothed_voltages(capacities[i]) for i in indices]

  The voltages for comparison.
- list experimental_datasets

  Packs the comparison points for EP-BOLFI.
- fixed_parameters = parameters.copy()

  The free parameters will have been subtracted from this deep copy.
- float t_eval = 0.9 ∗ np.array(capacities) ∗ 3600 / current

  Evaluation timepoints for the simulator.
- list simulators = [simulator]

  Pack the one simulator for EP-BOLFI.
- list feature_extractors = [get_features]

  Pack the defined features for EP-BOLFI.
- list display_current_feature = [name_of_dis_charge_features]

  Pack the feature names for EP-BOLFI.
- covariance = None

  (Don't) choose the initial covariance matrix.
- int bolfi_initial_evidence = 17

  Number of random evidence samples taken before BO, per feature.
- int bolfi_total_evidence = 51

  Number of initial and BO samples, per feature.
- int bolfi_posterior_samples = 34

  Number of random samples from the posterior distribution.
- int ep_iterations = 2

  Number of EP iterations, i.e.
- float ep_dampener = 0.5

  Think of this as the dampener in Newton iterations.
- int ep_dampener_reduction_steps = 1

  Number of steps after which the dampening is reduced to 0.
- bool show_trials = True

  Show the log of tried parameters live.
- int seed = 42

  Seed for consistent pseudo-randomness.

5.33.1 Function Documentation

5.33.1.1 feature_visualizer()

def Python.parameters.estimation.discharge_thick_electrodes.feature_visualizer (
args)

May be used for plotting purposes.

Fur further information, please refer to utility.visualization.plot_comparison.

5.33.1.2 get_features()

def Python.parameters.estimation.discharge_thick_electrodes.get_features (
dataset)

Defines the features.

dataset

The dataset in the same form as the experimental_datasets and the output of the simulators.

Returns

The unchanged dataset.

5.33.1.3 name_of_dis_charge_features()

def Python.parameters.estimation.discharge_thick_electrodes.name_of_dis_charge_features (
index)

Gives the descriptions of the features by index.

index

The index number of the desired feature.

Returns

The name of the desired feature.

5.33.1.4 simulator()

def Python.parameters.estimation.discharge_thick_electrodes.simulator (
trial_parameters)

A simulator compatible with EP-BOLFI.

Bruggeman coefficients are set to be equal for electrode and electrolyte so that they can be inferred as one parameter.

trial_parameters

A dictionary of parameters for the DFN battery model.

Returns

The simulated counterpart to the experimental features.

5.33.2 Variable Documentation

5.33.2.1 a

list Python.parameters.estimation.discharge_thick_electrodes.a = [-0.11927628084606558, -1.8526410797182222, -0.4921205974305653]

5.33.2.2 bolfi_initial_evidence

int Python.parameters.estimation.discharge_thick_electrodes.bolfi_initial_evidence = 17

Number of random evidence samples taken before BO, per feature.

5.33.2.3 bolfi_posterior_samples

int Python.parameters.estimation.discharge_thick_electrodes.bolfi_posterior_samples = 34

Number of random samples from the posterior distribution.

5.33.2.4 bolfi_total_evidence

int Python.parameters.estimation.discharge_thick_electrodes.bolfi_total_evidence = 51

Number of initial and BO samples, per feature.

5.33.2.5 capacities

Python.parameters.estimation.discharge_thick_electrodes.capacities = np.array(complete_dataset[0])

The SOC of the cell represented as total discharged capacity (in Ah).

5.33.2.6 cathode

list Python.parameters.estimation.discharge_thick_electrodes.cathode = [par[i] for i in range(len(E_0)) for par in [E_0, a, Δx]]

OCV fit parameters for the cathode.

5.33.2.7 complete_dataset

Python.parameters.estimation.discharge_thick_electrodes.complete_dataset

Initial value:

```
= read_voltages_from_csv(
    "../Datensätze/C_tenth_exp_thin_discharge_cathode.csv"
)
```

The experimental data to be used for inference.

5.33.2.8 covariance

Python.parameters.estimation.discharge_thick_electrodes.covariance = None

(Don't) choose the initial covariance matrix.

5.33.2.9 current

float Python.parameters.estimation.discharge_thick_electrodes.current = 0.00231

The current that is applied during the experiment.

5.33.2.10 display_current_feature

list Python.parameters.estimation.discharge_thick_electrodes.display_current_feature = [name_of_dis_charge_features]

Pack the feature names for EP-BOLFI.

5.33.2.11 E_0

list Python.parameters.estimation.discharge_thick_electrodes.E_0 = [3.9074103413415218, 3.7586234939671175, 3.7517405949585427]

5.33.2.12 ep_dampener

float Python.parameters.estimation.discharge_thick_electrodes.ep_dampener = 0.5

Think of this as the dampener in Newton iterations.

0: no dampening.

5.33.2.13 ep_dampener_reduction_steps

int Python.parameters.estimation.discharge_thick_electrodes.ep_dampener_reduction_steps = 1

Number of steps after which the dampening is reduced to 0.

5.33.2.14 ep_iterations

int Python.parameters.estimation.discharge_thick_electrodes.ep_iterations = 2

Number of EP iterations, i.e.

the number of passes over each feature.

5.33.2.15 experimental_datasets

list Python.parameters.estimation.discharge_thick_electrodes.experimental_datasets

Initial value:

```
=  [[plateau_voltages[0],
              plateau_voltages[1],
              plateau_voltages[2],
              #plateau_voltages[3],
              capacities[-1]]]
```

Packs the comparison points for EP-BOLFI.

5.33.2.16 feature_extractors

list Python.parameters.estimation.discharge_thick_electrodes.feature_extractors = [get_features]

Pack the defined features for EP-BOLFI.

5.33.2.17 fixed_parameters

Python.parameters.estimation.discharge_thick_electrodes.fixed_parameters = parameters.copy()

The free parameters will have been subtracted from this deep copy.

5.33.2.18 free_parameters

dictionary Python.parameters.estimation.discharge_thick_electrodes.free_parameters

Initial value:

```
= {
#    "Negative electrode surface area density [m-1]": (
#        np.log(116458), (0.5*0.5*l)**2),
#    "Positive electrode surface area density [m-1]": (
#        np.log(238065), (0.5*0.5*l)**2),
#    "Negative particle radius [m]": (
#        np.log(12e-6), (0.5*0.25*l)**2),
#    "Positive particle radius [m]": (
#        np.log(5.5e-6), (0.5*0.25*l)**2),
#    "Negative electrode diffusivity [m2.s-1]": (
#        np.log(3.9e-14), (0.5*l)**2),
     "Positive electrode diffusivity [m2.s-1]": (
         np.log(2e-15), (0.5*l)**2),
     "Positive electrode exchange-current density [A.m-2]": (
         np.log(6e-7), (0.5*l)**2),
#    "Electrolyte diffusivity [m2.s-1]": (
#        np.log(2.72e-10), (0.5*0.5*l)**2),
#    "Positive electrode active material volume fraction": (
#        0.49, (0.5*0.4)**2),
     "Initial concentration in positive electrode [mol.m-3]": (
         5000, (0.5*2000)**2),
     "Positive electrode thickness [m]": (
         70e-6, (0.5*30e-6)**2),
}
```

Parameters that are to be estimated.

Note: the guesses and variances refer to the transformed values of the guesses as given by 'transform_parameters'.

5.33.2.19 free_parameters_boundaries

dictionary Python.parameters.estimation.discharge_thick_electrodes.free_parameters_boundaries

Initial value:

```
= {
#    "Negative electrode surface area density [m-1]":
#        (11645.8, 1164580),
#    "Positive electrode surface area density [m-1]":
#        (23806.5, 2380650),
#    "Negative particle radius [m]":
#        (1e-6, 2e-5),
#    "Positive particle radius [m]":
#        (1e-6, 1e-5),
     "Positive electrode diffusivity [m2.s-1]":
         (2e-16, 2e-14),
     "Positive electrode exchange-current density [A.m-2]":
         (6e-8, 6e-6),
#    "Electrolyte diffusivity [m2.s-1]":
#        (2.72e-11, 2.72e-9),
#    "Positive electrode active material volume fraction":
#        (0.1, 0.9),
     "Initial concentration in positive electrode [mol.m-3]":
         (2000, 10000),
     "Positive electrode thickness [m]":
         (40e-6, 100e-6),
}
```

Bounds in which a parameter set is searched for.

5.33.2.20 indices

list Python.parameters.estimation.discharge_thick_electrodes.indices = [int(i) for i in len(capacities) ∗ np.array([0.0, 0.5, 0.9])]

Chooses the points in the discharge curve used for comparison.

5.33.2.21 input

list Python.parameters.estimation.discharge_thick_electrodes.input = ["Discharge at " + str(current) + " A for 360 s",]

The experiment as a simulation input for use with simulation_setup.

Alternatively, the current may be set for use with solver_setup by ”parameters[”Current function [A]”]” (a function dependent on time).

5.33.2.22 l

Python.parameters.estimation.discharge_thick_electrodes.l = np.log(10)

Use the natural logarithm of 10 for scalings in orders of magnitude.

5.33.2.23 plateau_capacities

list Python.parameters.estimation.discharge_thick_electrodes.plateau_capacities = [capacities[i] for i in indices]

The SOC points for comparison (in Ah).

5.33.2.24 plateau_voltages

list Python.parameters.estimation.discharge_thick_electrodes.plateau_voltages = [smoothed_voltages(capacities[i]) for i in indices]

The voltages for comparison.

5.33.2.25 seed

int Python.parameters.estimation.discharge_thick_electrodes.seed = 42

Seed for consistent pseudo-randomness.

Use None for randomness.

5.33.2.26 show_trials

bool Python.parameters.estimation.discharge_thick_electrodes.show_trials = True

Show the log of tried parameters live.

5.33.2.27 simulators

list Python.parameters.estimation.discharge_thick_electrodes.simulators = [simulator]

Pack the one simulator for EP-BOLFI.

5.33.2.28 smoothed_voltages

Python.parameters.estimation.discharge_thick_electrodes.smoothed_voltages = smooth_fit(capacities, voltages)

Limit the detrimental effect of random fluctuations by smoothing.

5.33.2.29 starting_SOC

int Python.parameters.estimation.discharge_thick_electrodes.starting_SOC = 1 - OCV_fit_function(voltages[0], ∗cathode)

The initial SOC of the cathode as guessed by its OCV fit function.

5.33.2.30 t_eval

float Python.parameters.estimation.discharge_thick_electrodes.t_eval = 0.9 ∗ np.array(capacities) ∗ 3600 / current

Evaluation timepoints for the simulator.

5.33.2.31 transform_parameters

dictionary Python.parameters.estimation.discharge_thick_electrodes.transform_parameters

Initial value:

```
= {
#   "Negative electrode surface area density [m-1]": "log",
#   "Positive electrode surface area density [m-1]": "log",
#   "Electrolyte diffusivity [m2.s-1]": "log",
#   "Negative particle radius [m]": "log",
    "Positive electrode diffusivity [m2.s-1]": "log",
#   "Positive particle radius [m]": "log",
    "Positive electrode exchange-current density [A.m-2]": "log",
}
```

Transformations/Scalings of the parameter ranges.

5.33.2.32 voltages

Python.parameters.estimation.discharge_thick_electrodes.voltages = np.array(complete_dataset[1])

The voltages at those SOCs.

5.33.2.33 voltages_without_OCV

Python.parameters.estimation.discharge_thick_electrodes.voltages_without_OCV

Initial value:

```
=  subtract_OCV_curve(
    voltages, capacities * 3600 / current, current, parameters, cathode
)
```

The isolated overpotential of the experiment.

5.33.2.34 Δx

list Python.parameters.estimation.discharge_thick_electrodes.Δx = [0.6837809843229503, 0.021811690524785213, 0.29440732515226453]

5.34 Python.parameters.estimation.gitt_timo Namespace Reference

Functions

- def current_function (t)
  The current that is applied during the GITT experiment.
- def simulator (trial_parameters)
  A simulator compatible with EP-BOLFI.
- def get_list_of_dataset (dataset)
  Defines the features.
- def name_of_gitt_features (index)
  Gives the descriptions of the features by index.
- def feature_visualizer (args)
  May be used for plotting purposes.

Variables

- l = np.log(10)
  Use the natural logarithm of 10 for scalings in orders of magnitude.
- dictionary free_parameters
  Parameters that are to be estimated.
- dictionary transform_parameters
  Transformations/Scalings of the parameter ranges.
- dictionary free_parameters_boundaries
  Bounds in which a parameter set is searched for.
- list E_0
  Fitted plateau voltages of the cathode.
- list a
  Fitted plateau inverse widths of the cathode.
- list Δx
  Fitted plateau inverse widths (transformed) of the cathode.
- list cathode = [par[i] for i in range(len(E_0)) for par in [E_0, a, Δx]]

OCV fit parameters for the cathode.

- complete_dataset

  The experimental data to be used for inference.
- starting_OCV = complete_dataset.voltages[6][-1]

  Define the starting OCV before each used pulse.
- dataset = complete_dataset.subslice(7, 10)

  Define the pulses that are to be used.
- list input = [ ]

  The current as a pybamm.Simulation input.
- int starting_SOC = 1 - OCV_fit_function(starting_OCV, ∗cathode)

  The initial SOC of the cathode as guessed by its OCV fit function.
- list OCV_fit = [par[i] for i in range(len(E_0)) for par in [E_0, a, Δx]]

  Fit parameters of the cathode OCV curve.
- voltages_without_OCV

  The overpotential curve during the GITT experiment.
- tuple experiment

  The representation of the measurement that gets plotted.
- complete_OCV = np.array(complete_dataset.asymptotic_voltages[3:-1:2])

  Complete measured OCV curve.
- list complete_SOC

  The SOCs for which that OCV was measured.
- list OCV = [starting_OCV] + dataset.asymptotic_voltages[1::2]

  The OCV voltages before each charge pulse.
- list ΔE_s = [U1 - U0 for U0, U1 in zip(OCV[:-1], OCV[1:])]

  The OCV rises between charge pulses.
- ΔE_t = dataset.exp_U_decays[::2][1]

  The transient voltage rises during charge pulses.
- float τ_r = 1.0 / np.array(dataset.exp_U_decays[2])

  The exponential decay rates during each segment.
- IR = dataset.ir_steps

  The IR steps between segments, i.e.
- list experimental_datasets = [[∗ΔE_t, ∗ΔE_s, ∗τ_r]]

  Packs the comparison points for EP-BOLFI.
- fixed_parameters = parameters.copy()

  The free parameters will have been subtracted from this deep copy.
- list simulators = [simulator]

  Pack the one simulator for EP-BOLFI.
- list feature_extractors = [get_list_of_dataset]

  Pack the defined features for EP-BOLFI.
- dictionary feature_by_index

  Gives the descriptions of the features by index.
- list display_current_feature = [name_of_gitt_features]

  Pack the feature names for EP-BOLFI.
- covariance = None

  (Don't) choose the initial covariance matrix.
- int bolfi_initial_evidence = 17

  Number of random evidence samples taken before BO, per feature.
- int bolfi_total_evidence = 51

  Number of initial and BO samples, per feature.
- int bolfi_posterior_samples = 34

  Number of random samples from the posterior distribution.

- int ep_iterations = 2

  Number of EP iterations, i.e.
- float ep_dampener = 0.5

  Think of this as the dampener in Newton iterations.
- int ep_dampener_reduction_steps = 1

  Number of steps after which the dampening is reduced to 0.
- bool show_trials = True

  Show the log of tried parameters live.
- int seed = 42

  Seed for consistent pseudo-randomness.

5.34.1 Function Documentation

5.34.1.1 current_function()

def Python.parameters.estimation.gitt_timo.current_function (
t)

The current that is applied during the GITT experiment.

t

The time since start of the experiment in seconds.

Returns

The current at that time in A.

5.34.1.2 feature_visualizer()

def Python.parameters.estimation.gitt_timo.feature_visualizer (
args)

May be used for plotting purposes.

Fur further information, please refer to utility.visualization.plot_comparison.

5.34.1.3 get_list_of_dataset()

def Python.parameters.estimation.gitt_timo.get_list_of_dataset (
dataset)

Defines the features.

dataset

The dataset in the same form as the experimental_datasets and the output of the simulators.

Returns

The unchanged dataset.

5.34.1.4 name_of_gitt_features()

def Python.parameters.estimation.gitt_timo.name_of_gitt_features (
index)

Gives the descriptions of the features by index.

index

The index number of the desired feature.

Returns

The name of the desired feature.

5.34.1.5 simulator()

def Python.parameters.estimation.gitt_timo.simulator (
trial_parameters)

A simulator compatible with EP-BOLFI.

Bruggeman coefficients are set to be equal for electrode and electrolyte so that they can be inferred as one parameter. The starting SOC is set to match the fitted cathode OCV curve.

trial_parameters

A dictionary of parameters for the DFN battery model.

Returns

The simulated counterpart to the experimental features.

5.34.2 Variable Documentation

5.34.2.1 a

list Python.parameters.estimation.gitt_timo.a

Initial value:

```
=  [-0.16206853844191263, -0.25502021816052556,
    -0.7291652571718426,  -1.3090028878917788]
```

Fitted plateau inverse widths of the cathode.

5.34.2.2 bolfi_initial_evidence

int Python.parameters.estimation.gitt_timo.bolfi_initial_evidence = 17

Number of random evidence samples taken before BO, per feature.

5.34.2.3 bolfi_posterior_samples

int Python.parameters.estimation.gitt_timo.bolfi_posterior_samples = 34

Number of random samples from the posterior distribution.

5.34.2.4 bolfi_total_evidence

int Python.parameters.estimation.gitt_timo.bolfi_total_evidence = 51

Number of initial and BO samples, per feature.

5.34.2.5 cathode

list Python.parameters.estimation.gitt_timo.cathode = [par[i] for i in range(len(E_0)) for par in [E_0, a, Δx]]

OCV fit parameters for the cathode.

5.34.2.6 complete_dataset

Python.parameters.estimation.gitt_timo.complete_dataset

Initial value:

```
= read_channels_from_measurement_system(
    '../Datensätze/KN19097_001_02.txt', 'iso-8859-1', 13,
    headers={0: "t [h]", 10: "I [A]", 9: "U [V]"},
    delimiter='\t', decimal=',', type="static",
    segment_column=7, current_sign_correction={
        "Pause": 1.0, "Charge": 1.0, "Discharge": 1.0,
    },
    correction_column=8
)
```

The experimental data to be used for inference.

5.34.2.7 complete_OCV

Python.parameters.estimation.gitt_timo.complete_OCV = np.array(complete_dataset.asymptotic_voltages[3:-1:2])

Complete measured OCV curve.

5.34.2.8 complete_SOC

list Python.parameters.estimation.gitt_timo.complete_SOC

Initial value:

```
= [0.0] + [C[-1] for C in calculate_SOC(
    complete_dataset.timepoints[4:], complete_dataset.currents[4:]
)][:-1:2]
```

The SOCs for which that OCV was measured.

5.34.2.9 covariance

Python.parameters.estimation.gitt_timo.covariance = None

(Don't) choose the initial covariance matrix.

5.34.2.10 dataset

Python.parameters.estimation.gitt_timo.dataset = complete_dataset.subslice(7, 10)

Define the pulses that are to be used.

5.34.2.11 display_current_feature

list Python.parameters.estimation.gitt_timo.display_current_feature = [name_of_gitt_features]

Pack the feature names for EP-BOLFI.

5.34.2.12 E_0

list Python.parameters.estimation.gitt_timo.E_0

Initial value:

```
= [4.233752896396318,   3.90112483467987,
      3.735878045479363,   3.6321014766418487]
```

Fitted plateau voltages of the cathode.

5.34.2.13 ep_dampener

float Python.parameters.estimation.gitt_timo.ep_dampener = 0.5

Think of this as the dampener in Newton iterations.

0: no dampening.

5.34.2.14 ep_dampener_reduction_steps

int Python.parameters.estimation.gitt_timo.ep_dampener_reduction_steps = 1

Number of steps after which the dampening is reduced to 0.

5.34.2.15 ep_iterations

int Python.parameters.estimation.gitt_timo.ep_iterations = 2

Number of EP iterations, i.e.

the number of passes over each feature.

5.34.2.16 experiment

tuple Python.parameters.estimation.gitt_timo.experiment

Initial value:

```
=  ([t - dataset.timepoints[0][0]
         for timepoints in dataset.timepoints for t in timepoints],
        [U for voltages in dataset.voltages for U in voltages])
```

The representation of the measurement that gets plotted.

5.34.2.17 experimental_datasets

list Python.parameters.estimation.gitt_timo.experimental_datasets = [[$*\Delta E_t$, $*\Delta E_s$, $*\tau_r$]]

Packs the comparison points for EP-BOLFI.

Alternatives: ΔE_s / (ΔE_t ∗ np.sqrt(Δt)), ΔE_s / ΔE_t∗∗2.

5.34.2.18 feature_by_index

dictionary Python.parameters.estimation.gitt_timo.feature_by_index

Initial value:

```
= {
#    0: "IR_rise",
#    1: "IR_drop",
#    2: "ΔEs, i.e. OCV rise",
    0: "Charge voltage peak",
    1: "Charge voltage step",
    3: "Charge relaxation time",
    2: "Pause voltage step",
    4: "Pause relaxation time",
}
```

Gives the descriptions of the features by index.

5.34.2.19 feature_extractors

list Python.parameters.estimation.gitt_timo.feature_extractors = [get_list_of_dataset]

Pack the defined features for EP-BOLFI.

5.34.2.20 fixed_parameters

Python.parameters.estimation.gitt_timo.fixed_parameters = parameters.copy()

The free parameters will have been subtracted from this deep copy.

5.34.2.21 free_parameters

dictionary Python.parameters.estimation.gitt_timo.free_parameters

Initial value:

```
= {
#    "Electrolyte diffusivity [m2.s-1]": (
#        np.log(2.72e-10), (0.5*0.5*l)**2),
#    "Negative electrode conductivity [S.m-1]": (
#        np.log(0.00949), (0.5*1*l)**2),
#    "Positive electrode conductivity [S.m-1]": (
#        np.log(0.107), (0.5*1*l)**2),
#    "Positive electrode double-layer capacity [F.m-2]": (
#        np.log(2e-1), (0.5*1*l)**2),
#    "Positive electrode surface area density [m-1]": (
#        300000, (0.5*100000)**2),
    "Positive electrode exchange-current density [A.m-2]": (
        np.log(1e-6), (0.5*0.5*l)**2),
    "Positive electrode diffusivity [m2.s-1]": (
        np.log(3e-15), (0.5*0.5*l)**2),
#    "Positive particle radius [m]": (
#        np.log(5.75e-6), (0.5*0.5*l)**2),
    #"Positive electrode Bruggeman coefficient (electrolyte)": (
    #    2, 0.5**2),
#    "Electrolyte diffusivity [m2.s-1]": (
#        np.log(2.72e-10), (0.5*1*l)**2),
    "Electrolyte conductivity [S.m-1]": (
        np.log(7), (0.5*0.25*l)**2),
#    "Electrode width [m]": (
#        0.015, (0.5*0.01)**2),
#    "1 + dlnf/dlnc": (
#        2, (0.5*1)**2),
    "Cation transference number": (
        0.5, (0.5*0.2)**2),
}
```

Parameters that are to be estimated.

Note: the guesses and variances refer to the transformed values of the guesses as given by 'transform_parameters'.

5.34.2.22 free_parameters_boundaries

dictionary Python.parameters.estimation.gitt_timo.free_parameters_boundaries

Initial value:

```
= {
#    "Electrolyte diffusivity [m2.s-1]":
#        (2.72e-11, 2.72e-9),
#    "Negative electrode conductivity [S.m-1]":
#        (0.000949, 0.0949),
#    "Positive electrode conductivity [S.m-1]":
#        (0.00107, 10.7),
#    "Positive electrode double-layer capacity [F.m-2]":
#        (2e-2, 2e-0),
#    "Positive electrode surface area density [m-1]":
#        (200000, 400000),
     "Positive electrode exchange-current density [A.m-2]":
         (1e-7, 1e-5),
     "Positive electrode diffusivity [m2.s-1]":
         (0.8e-15, 15e-15),
#    "Positive particle radius [m]":
#        (5.75e-7, 5.75e-5),
    #"Positive electrode Bruggeman coefficient (electrolyte)":
    #    (1, 3),
#    "Electrolyte diffusivity [m2.s-1]":
#        (2.72e-11, 2.72e-9),
     "Electrolyte conductivity [S.m-1]":
         (1, 20),
#    "Electrode width [m]":
#        (0.005, 0.03),
#    "1 + dlnf/dlnc":
#        (1, 3),
     "Cation transference number":
         (0.3, 0.7),
}
```

Bounds in which a parameter set is searched for.

5.34.2.23 input

list Python.parameters.estimation.gitt_timo.input = [ ]

The current as a pybamm.Simulation input.

Other example inputs: "Hold at 3.95 V for 180 s", "Discharge at 1 C for 1 s".

5.34.2.24 IR

Python.parameters.estimation.gitt_timo.IR = dataset.ir_steps

The IR steps between segments, i.e.

the ohmic voltage drops.

5.34.2.25 l

Python.parameters.estimation.gitt_timo.l = np.log(10)

Use the natural logarithm of 10 for scalings in orders of magnitude.

5.34.2.26 OCV

list Python.parameters.estimation.gitt_timo.OCV = [starting_OCV] + dataset.asymptotic_voltages[1::2]

The OCV voltages before each charge pulse.

5.34.2.27 OCV_fit

list Python.parameters.estimation.gitt_timo.OCV_fit = [par[i] for i in range(len(E_0)) for par in [E_0, a, Δx]]

Fit parameters of the cathode OCV curve.

5.34.2.28 seed

int Python.parameters.estimation.gitt_timo.seed = 42

Seed for consistent pseudo-randomness.

Use None for randomness.

5.34.2.29 show_trials

bool Python.parameters.estimation.gitt_timo.show_trials = True

Show the log of tried parameters live.

5.34.2.30 simulators

list Python.parameters.estimation.gitt_timo.simulators = [simulator]

Pack the one simulator for EP-BOLFI.

5.34.2.31 starting_OCV

Python.parameters.estimation.gitt_timo.starting_OCV = complete_dataset.voltages[6][-1]

Define the starting OCV before each used pulse.

5.34.2.32 starting_SOC

int Python.parameters.estimation.gitt_timo.starting_SOC = 1 - OCV_fit_function(starting_OCV, ∗cathode)

The initial SOC of the cathode as guessed by its OCV fit function.

5.34.2.33 transform_parameters

dictionary Python.parameters.estimation.gitt_timo.transform_parameters

Initial value:

```
= {
#   "Electrolyte diffusivity [m2.s-1]": "log",
#   "Negative electrode conductivity [S.m-1]": "log",
#   "Positive electrode conductivity [S.m-1]": "log",
#   "Positive electrode double-layer capacity [F.m-2]": "log",
    "Positive electrode exchange-current density [A.m-2]": "log",
    "Positive electrode diffusivity [m2.s-1]": "log",
#   "Positive particle radius [m]": "log",
    "Electrolyte conductivity [S.m-1]": "log",
}
```

Transformations/Scalings of the parameter ranges.

5.34.2.34 voltages_without_OCV

Python.parameters.estimation.gitt_timo.voltages_without_OCV

Initial value:

```
=  subtract_OCV_curve_from_cycles(
       dataset, parameters, cathode, starting_OCV=starting_OCV, electrode="anode"
)
```

The overpotential curve during the GITT experiment.

5.34.2.35 ΔE_s

list Python.parameters.estimation.gitt_timo.ΔE_s = [U1 - U0 for U0, U1 in zip(OCV[:-1], OCV[1:])]

The OCV rises between charge pulses.

5.34.2.36 ΔE_t

Python.parameters.estimation.gitt_timo.ΔE_t = dataset.exp_U_decays[::2][1]

The transient voltage rises during charge pulses.

5.34.2.37 Δx

list Python.parameters.estimation.gitt_timo.Δx

Initial value:

```
= [0.3528575991130032,  0.22284809800294844,
   0.32070177746632755, 0.10359252541772077]
```

Fitted plateau inverse widths (transformed) of the cathode.

5.34.2.38 τ_r

float Python.parameters.estimation.gitt_timo.τ_r = 1.0 / np.array(dataset.exp_U_decays[2])

The exponential decay rates during each segment.

5.35 Python.parameters.models Namespace Reference

Namespaces

- basf_samsung
- basytec
- diffusivity_curves
- ocv_curves
- thick_electrodes

5.36 Python.parameters.models.basf_samsung Namespace Reference

Variables

- float π = 3.141592653589793

  Pi is needed because the cell is cylindrical.
- float α_{nn} = 0.5

  Assumptions made without justification by measurements.
- float α_{pn} = 0.5

  Cathodic symmetry factor at the anode.
- float α_{np} = 0.5

  Anodic symmetry factor at the cathode.
- float α_{pp} = 0.5

  Cathodic symmetry factor at the cathode.

5.36.1 Variable Documentation

5.36.1.1 α_{nn}

float Python.parameters.models.basf_samsung.α_{nn} = 0.5

Assumptions made without justification by measurements.

# Note: some of them are at the bottom of this file. # Anodic symmetry factor at the anode.

5.36.1.2 α_{np}

float Python.parameters.models.basf_samsung.α_{np} = 0.5

Anodic symmetry factor at the cathode.

5.36.1.3 α_{pn}

float Python.parameters.models.basf_samsung.α_{pn} = 0.5

Cathodic symmetry factor at the anode.

5.36.1.4 α_{pp}

float Python.parameters.models.basf_samsung.α_{pp} = 0.5

Cathodic symmetry factor at the cathode.

5.36.1.5 π

float Python.parameters.models.basf_samsung.π = 3.141592653589793

Pi is needed because the cell is cylindrical.

5.37 Python.parameters.models.basytec Namespace Reference

Functions

- def D_int_positive (c, T, T_ref, E_D_s_p, R)
  Define the diffusivity function in a form PyBaMM can use.

Variables

- float α_{nn} = 0.5

  Anodic symmetry factor at the anode.
- float α_{pn} = 0.5

  Cathodic symmetry factor at the anode.
- float α_{np} = 0.5

  Anodic symmetry factor at the cathode.
- float α_{pp} = 0.5

  Cathodic symmetry factor at the cathode.
- dictionary parameters

  Parameter dictionary.
- dictionary c_n_max = parameters["Maximum concentration in negative electrode [mol.m-3]"]

  Maximum charge concentration in the anode active material.
- dictionary c_p_max = parameters["Maximum concentration in positive electrode [mol.m-3]"]

  Maximum charge concentration in the cathode active material.

5.37.1 Function Documentation

5.37.1.1 D_int_positive()

```
def Python.parameters.models.basytec.D_int_positive (
                    c,
                    T,
                    T_ref,
                    E_D_s_p,
                    R )
```

Define the diffusivity function in a form PyBaMM can use.

5.37.2 Variable Documentation

5.37.2.1 c_n_max

dictionary Python.parameters.models.basytec.c_n_max = parameters["Maximum concentration in negative electrode [mol.m-3]"]

Maximum charge concentration in the anode active material.

5.37.2.2 c_p_max

dictionary Python.parameters.models.basytec.c_p_max = parameters["Maximum concentration in positive electrode [mol.m-3]"]

Maximum charge concentration in the cathode active material.

5.37.2.3 parameters

dictionary Python.parameters.models.basytec.parameters

Parameter dictionary.

Initial conditions, exchange current densities, diffusivity and OCV curves will have been added.

5.37.2.4 α_{nn}

float Python.parameters.models.basytec.α_{nn} = 0.5

Anodic symmetry factor at the anode.

5.37.2.5 α_{np}

float Python.parameters.models.basytec.α_{np} = 0.5

Anodic symmetry factor at the cathode.

5.37.2.6 α_{pn}

float Python.parameters.models.basytec.α_{pn} = 0.5

Cathodic symmetry factor at the anode.

5.37.2.7 α_{pp}

float Python.parameters.models.basytec.α_{pp} = 0.5

Cathodic symmetry factor at the cathode.

5.38 Python.parameters.models.diffusivity_curves Namespace Reference

Functions

- def D_int_graphite (SOC)
  Standard graphite diffusivity.
- def D_int_NMC_111 (SOC)
  NMC diffusivity from "Thick electrodes".
- def D_int_NMC_622 (SOC)
  NMC diffusivity as used for the half-cell with GITT from Timo.

5.38.1 Function Documentation

5.38.1.1 D_int_graphite()

def Python.parameters.models.diffusivity_curves.D_int_graphite (
SOC)

Standard graphite diffusivity.

5.38.1.2 D_int_NMC_111()

def Python.parameters.models.diffusivity_curves.D_int_NMC_111 (
SOC)

NMC diffusivity from "Thick electrodes".

5.38.1.3 D_int_NMC_622()

def Python.parameters.models.diffusivity_curves.D_int_NMC_622 (
SOC)

NMC diffusivity as used for the half-cell with GITT from Timo.

5.39 Python.parameters.models.ocv_curves Namespace Reference

Functions

- def OCV_graphite (SOC)
  OCV curve of graphite as used in "Thick electrodes".
- def d_OCV_graphite_d_SOC (SOC)
  Derivative of the prior OCV curve.
- def OCV_NMC_111 (SOC)
  OCV curve of NMC as used in "Thick electrodes".
- def d_OCV_NMC_111_d_SOC (SOC)
  Derivative of the prior OCV curve.
- def OCV_NMC_622 (SOC)
  OCV curve of NMC as used for the half-cell with GITT from Timo.
- def d_OCV_NMC_622_d_SOC (SOC)
  Derivative of the prior OCV curve.
- def OCV_NMC_thick_electrodes (SOC)
  OCV curve fitted to the GITT measurement in "Thick electrodes".
- def d_OCV_NMC_thick_electrodes_d_SOC (SOC)
  Derivative of the prior OCV curve.
- def OCV_graphite_precise (SOC)
  OCV curve for graphite as seen in "A Parametric OCV Model".
- def d_OCV_graphite_precise_d_SOC (SOC)
  Derivative of the prior OCV curve.
- def OCV_NMC_442 (SOC)
  OCV curve of NMC fitted to the CC-CV measurements of a Samsung cell.
- def OCV_NMC_622_precise (SOC)
  OCV curve of NMC fitted to the GITT measurement from Timo.
- def d_OCV_NMC_622_precise_d_SOC (SOC)
  Derivative of the prior OCV curve.
- def OCV_BASF (SOC)
  OCV curve of NMC fitted to the in-house CC-CV measurements from BASF.

5.39.1 Function Documentation

5.39.1.1 d_OCV_graphite_d_SOC()

def Python.parameters.models.ocv_curves.d_OCV_graphite_d_SOC (
SOC)

Derivative of the prior OCV curve.

5.39.1.2 d_OCV_graphite_precise_d_SOC()

def Python.parameters.models.ocv_curves.d_OCV_graphite_precise_d_SOC (
SOC)

Derivative of the prior OCV curve.

5.39.1.3 d_OCV_NMC_111_d_SOC()

def Python.parameters.models.ocv_curves.d_OCV_NMC_111_d_SOC (
SOC)

Derivative of the prior OCV curve.

5.39.1.4 d_OCV_NMC_622_d_SOC()

def Python.parameters.models.ocv_curves.d_OCV_NMC_622_d_SOC (
SOC)

Derivative of the prior OCV curve.

5.39.1.5 d_OCV_NMC_622_precise_d_SOC()

def Python.parameters.models.ocv_curves.d_OCV_NMC_622_precise_d_SOC (
SOC)

Derivative of the prior OCV curve.

5.39.1.6 d_OCV_NMC_thick_electrodes_d_SOC()

def Python.parameters.models.ocv_curves.d_OCV_NMC_thick_electrodes_d_SOC (
SOC)

Derivative of the prior OCV curve.

5.39.1.7 OCV_BASF()

def Python.parameters.models.ocv_curves.OCV_BASF (
SOC)

OCV curve of NMC fitted to the in-house CC-CV measurements from BASF.

5.39.1.8 OCV_graphite()

def Python.parameters.models.ocv_curves.OCV_graphite (
SOC)

OCV curve of graphite as used in ”Thick electrodes”.

5.39.1.9 OCV_graphite_precise()

def Python.parameters.models.ocv_curves.OCV_graphite_precise (
SOC)

OCV curve for graphite as seen in ”A Parametric OCV Model”.

5.39.1.10 OCV_NMC_111()

def Python.parameters.models.ocv_curves.OCV_NMC_111 (
SOC)

OCV curve of NMC as used in ”Thick electrodes”.

5.39.1.11 OCV_NMC_442()

def Python.parameters.models.ocv_curves.OCV_NMC_442 (
SOC)

OCV curve of NMC fitted to the CC-CV measurements of a Samsung cell.

5.39.1.12 OCV_NMC_622()

def Python.parameters.models.ocv_curves.OCV_NMC_622 (
SOC)

OCV curve of NMC as used for the half-cell with GITT from Timo.

5.39.1.13 OCV_NMC_622_precise()

def Python.parameters.models.ocv_curves.OCV_NMC_622_precise (
SOC)

OCV curve of NMC fitted to the GITT measurement from Timo.

5.39.1.14 OCV_NMC_thick_electrodes()

def Python.parameters.models.ocv_curves.OCV_NMC_thick_electrodes (
SOC)

OCV curve fitted to the GITT measurement in ”Thick electrodes”.

5.40 Python.parameters.models.thick_electrodes Namespace Reference

Variables

- float α_{nn} = 0.5

  Anodic symmetry factor at the anode.
- float α_{pn} = 0.5

  Cathodic symmetry factor at the anode.
- float α_{np} = 0.5

  Anodic symmetry factor at the cathode.
- float α_{pp} = 0.5

  Cathodic symmetry factor at the cathode.
- dictionary parameters

  Parameter dictionary.
- dictionary c_n_max = parameters[”Maximum concentration in negative electrode [mol.m-3]”]

  Maximum charge concentration in the anode active material.
- dictionary c_p_max = parameters[”Maximum concentration in positive electrode [mol.m-3]”]

  Maximum charge concentration in the cathode active material.

5.40.1 Variable Documentation

5.40.1.1 c_n_max

dictionary Python.parameters.models.thick_electrodes.c_n_max = parameters["Maximum concentration in negative electrode [mol.m-3]"]

Maximum charge concentration in the anode active material.

5.40.1.2 c_p_max

dictionary Python.parameters.models.thick_electrodes.c_p_max = parameters["Maximum concentration in positive electrode [mol.m-3]"]

Maximum charge concentration in the cathode active material.

5.40.1.3 parameters

dictionary Python.parameters.models.thick_electrodes.parameters

Parameter dictionary.

Initial conditions and the exchange current densities will have been added after this file was loaded.

5.40.1.4 α_{nn}

float Python.parameters.models.thick_electrodes.α_{nn} = 0.5

Anodic symmetry factor at the anode.

5.40.1.5 α_{np}

float Python.parameters.models.thick_electrodes.α_{np} = 0.5

Anodic symmetry factor at the cathode.

5.40.1.6 α_{pn}

float Python.parameters.models.thick_electrodes.α_{pn} = 0.5

Cathodic symmetry factor at the anode.

5.40.1.7 α_{pp}

float Python.parameters.models.thick_electrodes.α_{pp} = 0.5

Cathodic symmetry factor at the cathode.

5.41 Python.particle_identification Namespace Reference

Namespaces

- particles

5.42 Python.particle_identification.particles Namespace Reference

Classes

- class InferenceConfig
- class ParticlesConfig

  Configurations.
- class ParticlesDataset

  Provides access to the toy dataset.

Functions

- def train (model)

  Train the model.

Variables

- ROOT_DIR = os.path.abspath("../../")

  Root directory of the project.
- COCO_WEIGHTS_PATH = os.path.join(ROOT_DIR, "mask_rcnn_coco.h5")

  Path to trained weights file (coco).
- BALLOON_WEIGHTS_PATH = os.path.join(ROOT_DIR, "mask_rcnn_balloon.h5")

  Path to trained weights file (balloon).
- DEFAULT_LOGS_DIR = os.path.join(ROOT_DIR, "logs")

  Directory to save logs and model checkpoints, if not provided through the command line argument –logs.
- parser
- metavar
- help
- required
- default
- args = parser.parse_args()
- config = ParticlesConfig()
- model
- weights_path = COCO_WEIGHTS_PATH
- by_name
- True
- exclude
- image_path
- video_path

5.42.1 Function Documentation

5.42.1.1 train()

```
def Python.particle_identification.particles.train (
                    model )
```

Train the model.

5.42.2 Variable Documentation

5.42.2.1 args

Python.particle_identification.particles.args = parser.parse_args()

5.42.2.2 BALLOON_WEIGHTS_PATH

Python.particle_identification.particles.BALLOON_WEIGHTS_PATH = os.path.join(ROOT_DIR, "mask_rcnn_balloon.h5")

Path to trained weights file (balloon).

5.42.2.3 by_name

Python.particle_identification.particles.by_name

5.42.2.4 COCO_WEIGHTS_PATH

Python.particle_identification.particles.COCO_WEIGHTS_PATH = os.path.join(ROOT_DIR, "mask_rcnn_coco.h5")

Path to trained weights file (coco).

5.42.2.5 config

Python.particle_identification.particles.config = ParticlesConfig()

5.42.2.6 default

Python.particle_identification.particles.default

5.42.2.7 DEFAULT_LOGS_DIR

Python.particle_identification.particles.DEFAULT_LOGS_DIR = os.path.join(ROOT_DIR, "logs")

Directory to save logs and model checkpoints, if not provided through the command line argument −logs.

5.42.2.8 exclude

Python.particle_identification.particles.exclude

5.42.2.9 help

Python.particle_identification.particles.help

5.42.2.10 image_path

Python.particle_identification.particles.image_path

5.42.2.11 metavar

Python.particle_identification.particles.metavar

5.42.2.12 model

Python.particle_identification.particles.model

Initial value:

```
=  modellib.MaskRCNN(mode="training", config=config,
                              model_dir=args.logs)
```

5.42.2.13 parser

Python.particle_identification.particles.parser

Initial value:

```
=  argparse.ArgumentParser(
       description='Train Mask R-CNN to detect particles.')
```

5.42.2.14 required

Python.particle_identification.particles.required

5.42.2.15 ROOT_DIR

Python.particle_identification.particles.ROOT_DIR = os.path.abspath("../../")

Root directory of the project.

5.42.2.16 True

Python.particle_identification.particles.True

5.42.2.17 video_path

Python.particle_identification.particles.video_path

5.42.2.18 weights_path

Python.particle_identification.particles.weights_path = COCO_WEIGHTS_PATH

5.43 Python.utility Namespace Reference

Namespaces

- calculate_characteristics
- fitting_functions
- preprocessing
- read_csv_datasets
- visualization

5.44 Python.utility.calculate_characteristics Namespace Reference

Functions

- def print (parameters_path)

  Prints the characteristics for a 1+1D model.

5.44.1 Function Documentation

5.44.1.1 print()

def Python.utility.calculate_characteristics.print (
parameters_path)

Prints the characteristics for a 1+1D model.

parameters_path

The path to the parameters file.

5.45 Python.utility.fitting_functions Namespace Reference

Functions

- def smooth_fit (x, y, order=5, splits=None, smoothing_factor=2e-3, display=False)

  Calculates a smoothed spline with derivatives.
- def fit_exponential_decay (timepoints, voltages, recursive_depth=1, threshold=0.75e-4)

  Extracts a set amount of exponential decay curves.
- def laplace_transform (x, y, s)

  Performs a basic laplace transformation.
- def a_fit (γUeminus1)

  Calculates the conversion from ”A parametric OCV model”.
- def OCV_fit_function (E_OCV, args, z=1.0, T=298.15, individual=False)

  The OCV model from ”A parametric OCV model”.
- def d_dE_OCV_fit_function (E_OCV, args, z=1.0, T=298.15)

  The derivative of fitting_functions.OCV_fit_function.
- def d2_dE2_OCV_fit_function (E_OCV, args, z=1.0, T=298.15)

  The 2^n derivative of fitting_functions.OCV_fit_function.
- def OCV_and_SOC_range_fit_function (E_OCV, args, z=1.0, T=298.15)

  The OCV model from ”A parametric OCV model”.
- def inverse_OCV_fit_function (SOC, args, z=1.0, T=298.15, inverted=True)

  The inverse of fitting_functions.OCV_fit_function.
- def fit_OCV (SOC, OCV, N=4, initial=None, z=1.0, T=298.15, inverted=True)

  Fits data to fitting_functions.OCV_fit_function.
- def fit_OCV_and_SOC_range (SOC, OCV, N=4, initial=None, z=1.0, T=298.15, inverted=True)

  Fits data to fitting_functions.OCV_fit_function.
- def verbose_spline_parameterization (coeffs, knots, order, function_name=”OCV”, function_args=”SOC”, verbose=False)

  Gives the monomic representation of a B-spline.

5.45.1 Function Documentation

5.45.1.1 a_fit()

```
def Python.utility.fitting_functions.a_fit (
                    γUeminus1 )
```

Calculates the conversion from "A parametric OCV model".

γUeminus1

$\gamma * U_i$ / e from "A parametric OCV model".

Returns

The approximation factor a_i from "A parametric OCV model".

5.45.1.2 d2_dE2_OCV_fit_function()

```
def Python.utility.fitting_functions.d2_dE2_OCV_fit_function (
                    E_OCV,
                    args,
                    z = 1.0,
                    T = 298.15 )
```

The 2nd derivative of fitting_functions.OCV_fit_function.

E_OCV

The voltages for which the 2nd derivative shall be evaluated.

args

A list which length is dividable by three. These are the parameters E_0, a and Δx from "A parametric OCV model" in the order (E_0_0, a_0, Δx_0, E_0_1, a_1, Δx_1...).

z

The charge number of the electrode interface reaction.

T

The temperature of the electrode.

Returns

The evaluation of ∂^2OCV_fit_function(OCV) / ∂^2OCV.

5.45.1.3 d_dE_OCV_fit_function()

def Python.utility.fitting_functions.d_dE_OCV_fit_function (
E_OCV,
args,
z = 1.0,
T = 298.15)

The derivative of fitting_functions.OCV_fit_function.

E_OCV

The voltages for which the derivative shall be evaluated.

args

A list which length is dividable by three. These are the parameters E_0, a and Δx from "A parametric OCV model" in the order (E_0_0, a_0, Δx_0, E_0_1, a_1, Δx_1...).

z

The charge number of the electrode interface reaction.

T

The temperature of the electrode.

Returns

The evaluation of ∂OCV_fit_function(OCV) / ∂OCV.

5.45.1.4 fit_exponential_decay()

def Python.utility.fitting_functions.fit_exponential_decay (
timepoints,
voltages,
recursive_depth = 1,
threshold = 0.75e-4)

Extracts a set amount of exponential decay curves.

timepoints

The timepoints of the measurements.

voltages

The corresponding voltages.

recursive_depth

The default (1) fits one exponential curve to the data. For higher values that fit is repeated with the data minus the preceding fit(s) for this amount of times minus one.

threshold

The upper threshold value for the normalized sum of absolute errors / the integral of absolute errors.

Returns

A list of length "recursive_depth" where each element is a 3-tuple with the timepoints, the fitted voltage evaluations and a 3-tuple of the parameters of the following decay function: t, (b,c,d) $\mapsto$ b + c $*$ np.exp(-d $*$ (t - timepoints[0])).

5.45.1.5 fit_OCV()

```
def Python.utility.fitting_functions.fit_OCV (
                SOC,
                OCV,
                N = 4,
                initial = None,
                z = 1.0,
                T = 298.15,
                inverted = True )
```

Fits data to fitting_functions.OCV_fit_function.

SOC

The SOCs at which measurements were made.

OCV

The corresponding open-circuit voltages.

initial

An optional initial guess for the parameters of the model.

z

The charge number of the electrode interface reaction.

T

The temperature of the electrode.

inverted

If True (default), the widely adopted SOC convention is assumed. If False, the formulation of ”A parametric OCV model” is used.

Returns

The fitted parameters of fitting_functions.OCV_fit_function.

5.45.1.6 fit_OCV_and_SOC_range()

```
def Python.utility.fitting_functions.fit_OCV_and_SOC_range (
                    SOC,
                    OCV,
                    N = 4,
                    initial = None,
                    z = 1.0,
                    T = 298.15,
                    inverted = True )
```

Fits data to fitting_functions.OCV_fit_function.

In addition to fitting_functions.fit_OCV, a model-based correction to the provided SOC-OCV-data is made. If "↩
SOC" lives in a (0,1)-range, the correction is given as its transformation to the (SOC_start, SOC_end)-range given as the first two returned numbers.

SOC

The SOCs at which measurements were made.

OCV

The corresponding open-circuit voltages.

initial

An optional initial guess for the parameters of the model.

z

The charge number of the electrode interface reaction.

T

The temperature of the electrode.

inverted

If True (default), the widely adopted SOC convention is assumed. If False, the formulation of "A parametric OCV model" is used.

Returns

The fitted parameters of fitting_functions.OCV_fit_function plus the fitted SOC range prepended.

5.45.1.7 inverse_OCV_fit_function()

```
def Python.utility.fitting_functions.inverse_OCV_fit_function (
        SOC,
        args,
        z = 1.0,
        T = 298.15,
        inverted = True )
```

The inverse of fitting_functions.OCV_fit_function.

Basically OCV(SOC). Requires that the Δx entries are sorted by x. This corresponds to the parameters being sorted by decreasing E_0.

E_OCV

The SOCs for which the voltages shall be evaluated.

args

A list which length is dividable by three. These are the parameters E_0, a and Δx from "A parametric OCV model" in the order (E_0_0, a_0, Δx_0, E_0_1, a_1, Δx_1...).

z

The charge number of the electrode interface reaction.

T

The temperature of the electrode.

inverted

The default (False) uses the formulation from "A parametric OCV model". If True, the SOC argument gets flipped internally to give the more widely adopted convention for the SOC direction.

Returns

The evaluation of the inverse model function.

5.45.1.8 laplace_transform()

```
def Python.utility.fitting_functions.laplace_transform (
        x,
        y,
        s )
```

Performs a basic laplace transformation.

x

The independent variable.

y

The dependent variable.

s

The (possibly complex) frequencies for which to perform the transform.

Returns

The evaluation of the laplace transform at s.

5.45.1.9 OCV_and_SOC_range_fit_function()

def Python.utility.fitting_functions.OCV_and_SOC_range_fit_function (
E_OCV,
args,
z = 1.0,
T = 298.15)

The OCV model from "A parametric OCV model".

E_OCV

The voltages for which the SOCs shall be evaluated.

args

A list which length plus 2 is dividable by three. These are the SOC range and the parameters E_0, a and Δx from "A parametric OCV model" in the order (SOC_start, SOC_end, E_0_0, a_0, Δx_0, E_0_1, a_1, Δx_1...).

z

The charge number of the electrode interface reaction.

T

The temperature of the electrode.

Returns

The evaluation of the model function. This is the output of fitting_functions.OCV_fit_function, linearly transformed from the range (0, 1) to the range (SOC_start, SOC_end).

5.45.1.10 OCV_fit_function()

def Python.utility.fitting_functions.OCV_fit_function (
E_OCV,
args,
z = 1.0,
T = 298.15,
individual = False)

The OCV model from "A parametric OCV model".

Reference

C. R. Birkl, E. McTurk, M. R. Roberts, P. G. Bruce and D. A. Howey. “A Parametric Open Circuit Voltage Model for Lithium Ion Batteries”. Journal of The Electrochemical Society, 162(12):A2271-A2280, 2015

E_OCV

The voltages for which the SOCs shall be evaluated.

args

A list which length is dividable by three. These are the parameters E_0, a and Δx from ”A parametric OCV model” in the order (E_0_0, a_0, Δx_0, E_0_1, a_1, Δx_1...).

z

The charge number of the electrode interface reaction.

T

The temperature of the electrode.

individual

Alters the returned value.

Returns

If ”individual” is False, the evaluation of the model function gets returned. When it is True, the individual summands in the model functions get returned.

5.45.1.11 smooth_fit()

```
def Python.utility.fitting_functions.smooth_fit (
        x,
        y,
        order = 5,
        splits = None,
        smoothing_factor = 2e-3,
        display = False )
```

Calculates a smoothed spline with derivatives.

Note: the ”roots” function of a spline only works if it is cubic, i.e. of third order. Each ”derivative” reduces the order by one.

x

The independent variable.

y

The dependent variable (”plotted over x”).

order

Interpolation order of the spline.

splits

Optional tuning parameter. A list of points between which splines will be fitted first. The returned spline then is a fit of these individual splines.

smoothing_factor

Optional tuning parameter. Higher values lead to coarser, but more smooth interpolations and vice versa.

display

If set to True, the fit parameters of the spline will be printed to console. If possible, a monomial representation is printed.

Returns

A smoothing spline in the form of scipy.UnivariateSpline.

5.45.1.12 verbose_spline_parameterization()

def Python.utility.fitting_functions.verbose_spline_parameterization (
coeffs,
knots,
order,
function_name = "OCV",
function_args = "SOC",
verbose = False)

Gives the monomic representation of a B-spline.

coeffs

The B-spline coefficients as structured by scipy.interpolate.

knots

The B-spline knots as structured by scipy.interpolate.

order

The order of the B-spline.

function_name

An optional name for the printed Python function.

function_args

An optional string for the arguments of the function.

verbose

Print information about the progress of the conversion.

Returns

A string that gives a Python function when ”exec”-uted.

5.46 Python.utility.preprocessing Namespace Reference

Functions

- def combine_parameters_to_try (parameters, parameters_to_try_dict)

  Give every combination as full parameter sets.
- def fix_parameters (parameters_to_be_fixed)

  Returns a function which sets some parameters in advance.
- def capacity (parameters)

  Convenience function for calculating the capacity.
- def calculate_SOC (timepoints, currents)

  Transforms applied current over time into SOC.
- def subtract_OCV_curve (voltages, timepoints, currents, parameters, OCV_fit, starting_OCV=-1, electrode="cathode")

  Removes the OCV curve from a single cycle.
- def subtract_OCV_curve_from_cycles (dataset, parameters, OCV_fit, starting_OCV=-1, electrode="cathode")

  Removes the OCV curve from a cycling measurement.
- def laplace_transform (x, y, s)

  Performs a basic laplace transformation.
- def OCV_from_CC_CV (charge, cv, discharge, name, phases, eval_points=200, spline_SOC_range=(0.01, 0.99), spline_order=2, spline_smoothing=2e-3, spline_print=False, parameters_print=False)

  Tries to extract the OCV curve from CC-CV cycling data.

5.46.1 Function Documentation

5.46.1.1 calculate_SOC()

def Python.utility.preprocessing.calculate_SOC (
  timepoints,
  currents)

Transforms applied current over time into SOC.

timepoints

  Array of the timepoint segments.

currents

  Array of the current segments.

Returns

  An array of the same shape describing SOC in C.

5.46.1.2 capacity()

def Python.utility.preprocessing.capacity (
parameters)

Convenience function for calculating the capacity.

parameters

A parameter file as defined by models.standard_parameters.

Returns

The capacity of the parameterized battery in C.

5.46.1.3 combine_parameters_to_try()

def Python.utility.preprocessing.combine_parameters_to_try (
parameters,
parameters_to_try_dict)

Give every combination as full parameter sets.

parameters

The base full parameter set as a dictionary.

parameters_to_try_dict

The keys of this dictionary correspond to the "parameters"' keys where different values are to be inserted. These are given by the tuples which are the values of this dictionary.

Returns

A 2-tuple where the first item is the list of all parameter set combinations and the second the list of the combinations only.

5.46.1.4 fix_parameters()

def Python.utility.preprocessing.fix_parameters (
parameters_to_be_fixed)

Returns a function which sets some parameters in advance.

parameters_to_be_fixed

These parameters will at least be a part of the dictionary that the returned function returns.

Returns

The function which adds additional parameters to a dictionary or replaces existing parameters with the new ones.

5.46.1.5 laplace_transform()

def Python.utility.preprocessing.laplace_transform (
x,
y,
s)

Performs a basic laplace transformation.

x

The independent variable.

y

The dependent variable.

s

The (possibly complex) frequencies for which to perform the transform.

Returns

The evaluation of the laplace transform at s.

5.46.1.6 OCV_from_CC_CV()

def Python.utility.preprocessing.OCV_from_CC_CV (
charge,
cv,
discharge,
name,
phases,
eval_points = 200,
spline_SOC_range = (0.01, 0.99),
spline_order = 2,
spline_smoothing = 2e-3,
spline_print = False,
parameters_print = False)

Tries to extract the OCV curve from CC-CV cycling data.

charge

A Cycling_Information object containing the constant charge cycle(s). If more than one CC-CV-cycle shall be analyzed, please make sure that the order of this, cv and discharge align.

cv

A Cycling_Information object containing the constant voltage part between charge and discharge cycle(s).

discharge

A Cycling_Information object containing the constant discharge cycle(s). These occur after each cv cycle.

name

Name of the material for which the CC-CV-cycling was measured.

phases

Number of phases in the fitting_functions.OCV_fit_function as an int. The higher it is, the more (over-)fitted the model becomes.

eval_points

The number of points for plotting of the OCV curves.

spline_SOC_range

2-tuple giving the SOC range in which the inverted fitting_functions.OCV_fit_function will be interpolated by a smoothing spline. Outside of this range the spline is used for extrapolation. Use this to fit the SOC range of interest more precisely, since a fit of the whole range usually fails due to the singularities at SOC 0 and 1. Please note that this range considers the 0-1-range in which the given SOC lies and not the linear transformation of it from the fitting process.

spline_order

Order of this smoothing spline. If it is set to 0, only the fitting_functions.OCV_fit_function is calculated and plotted.

spline_smoothing

Smoothing factor for this smoothing spline. Default: 2e-3. Lower numbers give more precision, while higher numbers give a simpler spline that smoothes over steep steps in the fitted OCV curve.

spline_print

Set to True if the smoothing spline should be simplified and printed.

parameters_print

Set to True if the fit parameters should be printed to console.

Returns

A 8-tuple consisting of the following: 0: OCV_fits The fitted OCV curve parameters for each CC-CV cycle as returned by fitting_functions.fit_OCV_and_SOC_range. 1: I_mean The currents assigned to each CC-CV cycle (without CV). 2: C_charge The moved capacities during the charge segment(s). This is a list of the same length as charge, cv or discharge. 3: U_charge The voltages during the charge segment(s). Length: same. 4: C_discharge The moved capacities during the discharge segment(s). Length: same. 5: U_discharge The voltages during the discharge segment(s). Length: same. 6: C_evals Structurally the same as C_charge or C_discharge, this contains the moved capacities that were assigned to the mean voltages of charge and discharge cycle(s). 7: U_means The mean voltages of each charge and discharge cycle.

5.46.1.7 subtract_OCV_curve()

```
def Python.utility.preprocessing.subtract_OCV_curve (
                    voltages,
                    timepoints,
                    currents,
                    parameters,
                    OCV_fit,
                    starting_OCV = -1,
                    electrode = "cathode" )
```

Removes the OCV curve from a single cycle.

voltages

An array of the voltage measurement points.

timepoints

An array of the timepoints at which voltages were measured.

currents

An array of the corresponding currents. A number works as well.

parameters

The parameters of the battery as used for the PyBaMM simulations (see models.standard_parameters).

OCV_fit

The fit parameters of an OCV curve as returned by fitting_functions.fit_OCV(). If fit_OCV_and_SOC_range() was used, crop the first two entries for SOC start and end.

starting_OCV

The OCV at the beginning of the measurement. Put in a negative number to use the first entry of voltages for this. Default: -1.

electrode

”cathode” (default) or ”anode” for sign correction.

Returns

voltages minus the OCV as estimated for each data point.

5.46.1.8 subtract_OCV_curve_from_cycles()

def Python.utility.preprocessing.subtract_OCV_curve_from_cycles (
dataset,
parameters,
OCV_fit,
starting_OCV = -1,
electrode = "cathode")

Removes the OCV curve from a cycling measurement.

dataset

A Cycling_Information object of the measurement.

parameters

The parameters of the battery as used for the PyBaMM simulations (see models.standard_parameters).

OCV_fit

The fit parameters of an OCV curve as returned by fitting_functions.fit_OCV(). If fit_OCV_and_SOC_range() was used, crop the first two entries for SOC start and end.

starting_OCV

The OCV at the beginning of the measurement. Put in a negative number to use the first entry of voltages for this. Default: -1.

electrode

"cathode" (default) or "anode" for sign correction.

Returns

voltages minus the OCV as estimated for each data point. These are structured in exactly the same way as in the "dataset".

5.47 Python.utility.read_csv_datasets Namespace Reference

Classes

- class Cycling_Information
  Contains basic cycling informations.
- class Static_Information
  Contains additional informations, e.g.

Functions

- def read_voltages_from_csv (path)
  Read a (dis-)charge curve from a csv file.
- def read_channels_from_measurement_system (path, encoding, number_of_comment_lines, headers, delimiter='\t', decimal='.', type="", segment_column=-1, current_sign_correction={}, correction_column=-1, max_number_of_lines=-1)
  Read the measurements as returned by common instruments.

5.47.1 Function Documentation

5.47.1.1 read_channels_from_measurement_system()

```
def Python.utility.read_csv_datasets.read_channels_from_measurement_system (
                    path,
                    encoding,
                    number_of_comment_lines,
                    headers,
                    delimiter = '\t',
                    decimal = '.',
                    type = "",
                    segment_column = -1,
                    current_sign_correction = {},
                    correction_column = -1,
                    max_number_of_lines = -1 )
```

Read the measurements as returned by common instruments.

Example: cycling measurements from Basytec devices. Their format resembles a csv file with one title and one header comment line. So the first line will be ignored and the second used for headers.

path

The full or relative path to the measurement file.

encoding

The encoding of that file, e.g. "iso-8859-1".

number_of_comment_lines

The number of lines that have to be skipped over in order to arrive at the first dataset line.

headers

A dictionary. Its keys are the indices of the columns which are to be read in. The corresponding values are there to tell this function which kind of data is in which column. The following format has to be used: "<name> [<unit>]" where "name" is "U" (voltage), "I" (current) or "t" (time) and "unit" is "V", "A", "h", "m" or "s" with the optional prefixes "k", "m", "μ" or "n". This converts the data to prefix-less SI units. Additional columns may be read in with keys not in this format. The columns for segments and sign correction are only given by #segment_column and #correction_column.

delimiter

The delimiter string between datapoints. The default is "\t".

decimal

The string used for the decimal point. Default: ".".

type

Default is "", where only basic information is extracted from the file. "static" will trigger the additional extraction of exponential decays that are relevant to e.g. GITT.

segment_column

The index of the column that stores the index of the current segment. If it changes from one data point to the next, that is used as the dividing line between two segments. Default is -1, which returns the dataset in one segment.

current_sign_correction

A dictionary. Its keys are the strings used in the file to indicate a state. The column from which this state is retrieved is given by #correction_column. The dictionaries' values are used to correct/normalize the current value in the file. For example, if discharge currents have the same positive sign as charge currents in the file, use -1 to correct that, or if the values are to be scaled by weight, use the scaling factor. The default is the empty dictionary.

correction_column

See #current_sign_correction. Default: -1.

max_number_of_lines

The maximum number of dataset lines that are to be read in. Default: -1 (no limit).

Returns

A Cycling_Information or Static_Information object, depending on the value of "type".

5.47.1.2 read_voltages_from_csv()

def Python.utility.read_csv_datasets.read_voltages_from_csv (
path)

Read a (dis-)charge curve from a csv file.

Example: the csv files generated by WebPlotDigitizer.

path

The full or relative path to the csv measurement file.

Returns

A 2-tuple containing the capacities (sorted in ascending order) and the corresponding voltages as arrays.

5.48 Python.utility.visualization Namespace Reference

Classes

- class HandlerColorLineCollection

  Automates multicolored lines in legends.

Functions

- def update_limits (ax, xmin=float('inf'), xmax=-float('inf'), ymin=float('inf'), ymax=-float('inf'))

  Convenience function for adjusting the view.
- def set_fontsize (ax, title=12, xaxis=12, yaxis=12, xticks=12, yticks=12, legend=12)

  Convenience function for fontsize changes.
- def update_legend (ax, additional_handles=[ ], additional_labels=[ ], additional_handler_map={})

  Makes sure that all items remain and all items show up.
- def make_segments (x, y)

  Create a list of line segments from x and y coordinates.
- def colorline (x, y, z=None, cmap=plt.get_cmap('viridis'), norm=matplotlib.colors.Normalize(0, 1), linewidth=1, linestyle='-', alpha=1.0)

  Generates a colored line using LineCollection.
- def impedance_visualization (fig, ax, ω, Z, cmap=plt.get_cmap('tab20b'), ls='-', lw=3, legend_points=4, legend_text="impedance", colorbar_label="Frequency / Hz")
- def plot_comparison (ax, solutions, experiment, t_eval=None, title="", xlabel="", ylabel="", feature_↩ visualizer=lambda ∗args:[ ], interactive_plot=False, output_variables=None)

  Tool for comparing simulation<->experiment with features.
- def cc_cv_visualization (fig, ax, dataset, max_number_of_clusters=4, cmap=plt.get_cmap('tab20c'), check↩ _location=[0.1)

  Automatically labels and displays a CC-CV dataset.
- def plot_OCV_from_CC_CV (ax_ICA_meas, ax_ICA_mean, ax_OCV_meas, ax_OCV_mean, charge, cv, discharge, name, phases, eval_points=200, spline_SOC_range=(0.01, 0.99), spline_order=2, spline_↩ smoothing=2e-3, spline_print=False, parameters_print=False)

  Visualizes the OCV_fitting.OCV_from_CC_CV output.
- def plot_ICA (ax, SOC, OCV, name, spline_order=2, spline_smoothing=2e-3, sign=1)

  Show the derivative of charge by voltage.
- def plot_measurement (fig, ax, dataset, title, cmap=plt.get_cmap('tab20c'))

  Plots current and voltage curves in one diagram.
- def fit_and_plot_OCV (ax, SOC, OCV, name, phases, eval_points=200, spline_SOC_range=(0.01, 0.99), spline_order=2, spline_smoothing=2e-3, spline_print=False, parameters_print=False, inverted=True)

  Fits an SOC(OCV)-model and an OCV(SOC)-evaluable spline.

5.48.1 Function Documentation

5.48.1.1 cc_cv_visualization()

def Python.utility.visualization.cc_cv_visualization (
fig,
ax,
dataset,
max_number_of_clusters = 4,
cmap = plt.get_cmap('tab20c'),
check_location = [0.1)

Automatically labels and displays a CC-CV dataset.

A checkbutton list gets added for browsing through the labels.

fig

The Figure where the check boxes shall be drawn.

ax

The Axes where the measurements shall be drawn.

dataset

An instance of Cycling_Information. Please refer to utility.read_csv_datasets for further information.

max_number_of_clusters

The maximum number of different labels that shall be tried in the automatic labelling of the dataset.

cmap

The Colormap that is used for colorcoding the cycles.

check_location

The (x,y)-coordinates (first two entries) and the (width,height) (last two entries) of the checkbutton list canvas.

Returns

The CheckButtons instance. The only thing that must be done with this is to keep it in memory. Otherwise, it gets garbage collected (”weak reference”) and the checkbuttons don't work.

5.48.1.2 colorline()

def Python.utility.visualization.colorline (
x,
y,
z = None,
cmap = plt.get_cmap('viridis'),
norm = matplotlib.colors.Normalize(0, 1),
linewidth = 1,
linestyle = '-',
alpha = 1.0)

Generates a colored line using LineCollection.

http://nbviewer.ipython.org/github/dpsanders/matplotlib-examples/ blob/master/colorline.ipynb http://matplotlib.↩
org/examples/pylab_examples/multicolored_line.html

x

The independent variable.

y

The dependent variable.

z

Specify colors.

cmap

Specify a colormap for colors.

norm

Specify a normalization for mapping z to the colormap. Example: matplotlib.colors.LogNorm(10∗∗(-2), 10∗∗4).

linewidth

The linewidth of the generated LineCollection.

linestyle

The linestyle of the generated LineCollection. If the individual lines in there are too short, its effect might not be visible.

alpha

The transparency of the generated LineCollection.

Returns

A matplotlib.collections.LineCollection object ”lc”. It can be plotted by a Matplotlib axis ax with ”ax.add↩ _collection(lc)”.

5.48.1.3 fit_and_plot_OCV()

def Python.utility.visualization.fit_and_plot_OCV (
ax,
SOC,
OCV,
name,
phases,
eval_points = 200,
spline_SOC_range = (0.01, 0.99),
spline_order = 2,
spline_smoothing = 2e-3,
spline_print = False,
parameters_print = False,
inverted = True)

Fits an SOC(OCV)-model and an OCV(SOC)-evaluable spline.

Exemplary fit parameters:

Fit parameters of a graphite anode.

E_0_g = np.array([0.35973, 0.17454, 0.12454, 0.081957]) γUeminus1_g = np.array([-0.33144, 8.9434e-3, 7.2404e-2, 6.7789e-2]) a_g = a_fit(γUeminus1_g) Δx_g = np.array([8.041e-2, 0.23299, 0.29691, 0.39381])#0.22887 graphite = [p[i] for i in range(4) for p in [E_0_g, a_g, Δx_g]]

Fit parameters of a NMC-622 cathode.

E_0_NMC = np.array([4.2818, 3.9632, 3.9118, 3.6788]) γUeminus1_NMC = np.array([-0.22022, -0.083146, 0.070787, -0.11461]) a_NMC = a_fit(γUeminus1_NMC) Δx_NMC = np.array([0.38646, 0.28229, 0.15104, 0.26562])#0.30105 NMC = [p[i] for i in range(4) for p in [E_0_NMC, a_NMC, Δx_NMC]]

ax

The matplotlib.Axes instance for plotting.

SOC

Presumed SOC points of the OCV measurement. They only need to be precise in respect to relative capacity between measurements. The SOC endpoints of the measurement will be fitted using the fitting_functions.↩ OCV_fit_function. Type: list or np.array.

OCV

OCV measurements as a list or np.array.

name

Name of the material for which the OCV curve was measured.

phases

Number of phases in the fitting_functions.OCV_fit_function as an int. The higher it is, the more (over-)fitted the model becomes.

eval_points

The number of points for plotting of the OCV curves.

spline_SOC_range

2-tuple giving the SOC range in which the inverted fitting_functions.OCV_fit_function will be interpolated by a smoothing spline. Outside of this range the spline is used for extrapolation. Use this to fit the SOC range of interest more precisely, since a fit of the whole range usually fails due to the singularities at SOC 0 and 1. Please note that this range considers the 0-1-range in which the given SOC lies and not the linear transformation of it from the fitting process.

spline_order

Order of this smoothing spline. If it is set to 0, only the fitting_functions.OCV_fit_function is calculated and plotted.

spline_smoothing

Smoothing factor for this smoothing spline. Default: 2e-3. Lower numbers give more precision, while higher numbers give a simpler spline that smoothes over steep steps in the fitted OCV curve.

spline_print

Set to True if the smoothing spline should be simplified and printed.

parameters_print

Set to True if the fit parameters should be printed to console.

inverted

If True (default), the widely adopted SOC convention is assumed. If False, the formulation of "A parametric OCV model" is used.

5.48.1.4 impedance_visualization()

def Python.utility.visualization.impedance_visualization (
fig,
ax,
ω,
Z,
cmap = plt.get_cmap('tab20b'),
ls = '-',
lw = 3,
legend_points = 4,
legend_text = "impedance",
colorbar_label = "Frequency / Hz")

5.48.1.5 make_segments()

def Python.utility.visualization.make_segments (
x,
y)

Create a list of line segments from x and y coordinates.

x

The independent variable.

y

The dependent variable.

Returns

An array of the form numlines x (points per line) times 2 (x and y) array. This is the correct format for LineCollection.

5.48.1.6 plot_comparison()

def Python.utility.visualization.plot_comparison (
ax,
solutions,
experiment,
t_eval = None,
title = "",
xlabel = "",
ylabel = "",
*feature_visualizer = lambda *args: [],*
interactive_plot = False,
output_variables = None)

Tool for comparing simulation<->experiment with features.

First, a pybamm.QuickPlot shows the contents of ”solutions”. Then, a plot for feature visualization is generated.

ax

The Axes onto which the comparison shall be plotted.

solutions

A list of pybamm.Solution objects.

experiment

A list/tuple of at least length 2. The first two entries are the data timepoints and voltages. The entries after that are only relevant as additional arguments to ”feature_visualizer”.

t_eval

The timepoints at which the ”solutions” shall be evaluated. If None are given, the timepoints of the solutions will be chosen.

title

The optional title of the feature visualization plot.

xlabel

The optional label of the x-axis there.

ylabel

The optional label of the y-axis there.

feature_visualizer

This is an optional function that takes ”experiment” and returns a list of 2- or 3-tuples. The first two entries in the tuples are x- and y-data to be plotted alongside the other curves. The third entry is a string that is plotted at the respective (x[0], y[0])-coordinates.

interactive_plot

Choose whether or not a browsable overview of the solution components shall be shown. Please note that this disrupts the execution of this function until that plot is closed, since it is plotted in a new figure rather than in ax.

output_variables

The variables of ”solutions” that are to be plotted. When None are specified, some default variables get plotted. The full list of possible variables to plot are returned by PyBaMM models from their get_fundamental↩
_variables and get_coupled_variables functions. Enter the keys from that as strings in a list here.

5.48.1.7 plot_ICA()

```
def Python.utility.visualization.plot_ICA (
                    ax,
                    SOC,
                    OCV,
                    name,
                    spline_order = 2,
                    spline_smoothing = 2e-3,
                    sign = 1 )
```

Show the derivative of charge by voltage.

ax

The matplotlib.Axes instance for plotting.

SOC

Presumed SOC points of the OCV measurement. They only need to be precise in respect to relative capacity between measurements.

OCV

OCV measurements as a list or np.array, matching SOC.

name

Name of the material for which the OCV curve was measured.

spline_order

Order of the smoothing spline used for derivation. Default: 2.

spline_smoothing

Smoothing factor for this smoothing spline. Default: 2e-3. Lower numbers give more precision, while higher numbers give a simpler spline that smoothes over steep steps in the fitted OCV curve.

sign

Put -1 if the ICA comes out negative. Default: 1.

5.48.1.8 plot_measurement()

```
def Python.utility.visualization.plot_measurement (
                    fig,
                    ax,
                    dataset,
                    title,
                    cmap = plt.get_cmap('tab20c') )
```

Plots current and voltage curves in one diagram.

Please don't use "fig.tight_layout()" with this, as it very well might mess up the placement of the colorbar and the second y-axis. Rather, use "plt.subplots(..., constrained_layout=True)".

5.48.1.9 plot_OCV_from_CC_CV()

def Python.utility.visualization.plot_OCV_from_CC_CV (
ax_ICA_meas,
ax_ICA_mean,
ax_OCV_meas,
ax_OCV_mean,
charge,
cv,
discharge,
name,
phases,
eval_points = 200,
spline_SOC_range = (0.01, 0.99),
spline_order = 2,
spline_smoothing = 2e-3,
spline_print = False,
parameters_print = False)

Visualizes the OCV_fitting.OCV_from_CC_CV output.

ax_ICA_meas

The Axes where the Incremental Capacity Analysis of the measured charge and discharge cycle(s) shall be plotted.

ax_ICA_mean

The Axes where the Incremental Capacity Analysis of the mean voltages of charge and discharge cycle(s) shall be plotted.

ax_OCV_meas

The Axes where the measured voltage curves shall be plotted.

ax_OCV_mean

The Axes where the mean voltage curves shall be plotted.

charge

A Cycling_Information object containing the constant charge cycle(s). If more than one CC-CV-cycle shall be analyzed, please make sure that the order of this, cv and discharge align.

cv

A Cycling_Information object containing the constant voltage part between charge and discharge cycle(s).

discharge

A Cycling_Information object containing the constant discharge cycle(s). These occur after each cv cycle.

name

Name of the material for which the CC-CV-cycling was measured.

phases

Number of phases in the fitting_functions.OCV_fit_function as an int. The higher it is, the more (over-)fitted the model becomes.

eval_points

The number of points for plotting of the OCV curves.

spline_SOC_range

2-tuple giving the SOC range in which the inverted fitting_functions.OCV_fit_function will be interpolated by a smoothing spline. Outside of this range the spline is used for extrapolation. Use this to fit the SOC range of interest more precisely, since a fit of the whole range usually fails due to the singularities at SOC 0 and 1. Please note that this range considers the 0-1-range in which the given SOC lies and not the linear transformation of it from the fitting process.

spline_order

Order of this smoothing spline. If it is set to 0, only the fitting_functions.OCV_fit_function is calculated and plotted.

spline_smoothing

Smoothing factor for this smoothing spline. Default: 2e-3. Lower numbers give more precision, while higher numbers give a simpler spline that smoothes over steep steps in the fitted OCV curve.

spline_print

Set to True if the smoothing spline should be simplified and printed.

parameters_print

Set to True if the fit parameters should be printed to console.

5.48.1.10 set_fontsize()

```
def Python.utility.visualization.set_fontsize (
        ax,
        title = 12,
        xaxis = 12,
        yaxis = 12,
        xticks = 12,
        yticks = 12,
        legend = 12 )
```

Convenience function for fontsize changes.

ax

The axis which texts shall be adjusted.

title

The new fontsize for the title.

xaxis

The new fontsize for the x-axis label.

yaxis

The new fontsize for the y-axis label.

xticks

The new fontsize for the ticks/numbers at the x-axis.

yticks

The new fontsize for the ticks/numbers at the y-axis.

legend

The new fontsize for the legend entries.

5.48.1.11 update_legend()

```
def Python.utility.visualization.update_legend (
                    ax,
                    additional_handles = [],
                    additional_labels = [],
                    additional_handler_map = {} )
```

Makes sure that all items remain and all items show up.

This basically replaces "ax.legend()" in a way that makes sure that new items can be added to the legend without losing old ones. Please note that only "handler_map"s with class keys work correctly.

ax

The axis which legend shall be updated.

additional_handles

The same input "ax.legend(...)" would expect. A list of artists.

additional_labels

The same input "ax.legend(...)" would expect. A list of strings.

additonal_handler_map

The same input "ax.legend(...)" would expect for "handler_map". Please note that, due to the internal structure of the Legend class, only entries with keys that represent classes work right. Entries that have *instances* of classes (i.e., objects) for keys work exactly once, since the original handle of them is lost in the initialization of a Legend.

5.48.1.12 update_limits()

def Python.utility.visualization.update_limits (
ax,
xmin = float('inf'),
xmax = -float('inf'),
ymin = float('inf'),
ymax = -float('inf'))

Convenience function for adjusting the view.

ax

The axis which viewport shall be adjusted.

xmin

The highest lower bound for the x-axis.

xmax

The lowest upper bound for the x-axis.

ymin

The highest lower bound for the y-axis.

ymax

The lowest upper bound for the y-axis.

5.49 Python.watershed Namespace Reference

Functions

- def gaussian_kernel (x, μ, σ, w)

Variables

- π = np.pi
- image = skimage.io.imread(”../Datensätze/RE_93_cathode_20mu_edited.png”)
- orig_image = skimage.io.imread(”../Datensätze/RE_93_cathode_20mu.png”)
- foreground
- background
- markers = np.zeros_like(image)
- elevation_map = sobel(image)
- segmentation = watershed(elevation_map, markers)
- fig
- ax
- figsize
- int n = 4
- prop_cycle = plt.rcParams['axes.prop_cycle']
- colors = prop_cycle.by_key()['color']
- pixels = image.flatten()

- hist
- clust = KMeans(n_clusters=n)
- labels = clust.fit_predict(pixels.reshape(-1, 1))
- centers = clust.cluster_centers_.flatten()
- list indices = [(labels == i).nonzero()[0] for i in range(n)]
- ranges
- list thresholds
- processed_image = np.zeros_like(image)
- float last_t = 0.0
- nrows
- ncols
- grayscale = np.linspace(0, 1, 255)[:, np.newaxis]
- plot_range = np.linspace(r[0], r[1], 100)
- color
- alpha

5.49.1 Function Documentation

5.49.1.1 gaussian_kernel()

def Python.watershed.gaussian_kernel (
    x,
    μ,
    σ,
    w)

5.49.2 Variable Documentation

5.49.2.1 alpha

Python.watershed.alpha

5.49.2.2 ax

Python.watershed.ax

5.49.2.3 background

Python.watershed.background

5.49.2.4 centers

Python.watershed.centers = clust.cluster_centers_.flatten()

5.49.2.5 clust

Python.watershed.clust = KMeans(n_clusters=n)

5.49.2.6 color

Python.watershed.color

5.49.2.7 colors

Python.watershed.colors = prop_cycle.by_key()['color']

5.49.2.8 elevation_map

Python.watershed.elevation_map = sobel(image)

5.49.2.9 fig

Python.watershed.fig

5.49.2.10 figsize

Python.watershed.figsize

5.49.2.11 foreground

Python.watershed.foreground

5.49.2.12 grayscale

Python.watershed.grayscale = np.linspace(0, 1, 255)[:, np.newaxis]

5.49.2.13 hist

Python.watershed.hist

Initial value:

```
= KernelDensity(kernel='gaussian', bandwidth=1.0 / 255).fit(
    pixels.reshape(-1, 1)
)
```

5.49.2.14 image

Python.watershed.image = skimage.io.imread("../Datensätze/RE_93_cathode_20mu_edited.png")

5.49.2.15 indices

list Python.watershed.indices = [(labels == i).nonzero()[0] for i in range(n)]

5.49.2.16 labels

Python.watershed.labels = clust.fit_predict(pixels.reshape(-1, 1))

5.49.2.17 last_t

Python.watershed.last_t = 0.0

5.49.2.18 markers

Python.watershed.markers = np.zeros_like(image)

5.49.2.19 n

int Python.watershed.n = 4

5.49.2.20 ncols

Python.watershed.ncols

5.49.2.21 nrows

Python.watershed.nrows

5.49.2.22 orig_image

Python.watershed.orig_image = skimage.io.imread("../Datensätze/RE_93_cathode_20mu.png")

5.49.2.23 pixels

Python.watershed.pixels = image.flatten()

5.49.2.24 plot_range

Python.watershed.plot_range = np.linspace(r[0], r[1], 100)

5.49.2.25 processed_image

Python.watershed.processed_image = np.zeros_like(image)

5.49.2.26 prop_cycle

Python.watershed.prop_cycle = plt.rcParams['axes.prop_cycle']

5.49.2.27 ranges

Python.watershed.ranges

Initial value:

```
=  sorted([(np.min(pixels[array]), np.max(pixels[array]))
              for array in indices])
```

5.49.2.28 segmentation

Python.watershed.segmentation = watershed(elevation_map, markers)

5.49.2.29 thresholds

list Python.watershed.thresholds

Initial value:

```
= [0.5 * (r1[0] + r0[1])
        for (r0, r1) in zip(ranges[:-1], ranges[1:])]
```

5.49.2.30 π

Python.watershed.π = np.pi

5.50 utility Namespace Reference

Namespaces

- characteristics

  This file calculates and displays battery characteristics and timescales from a 1+1D-DFN parameter set as it used in PyBaMM.
- fitting_functions

  Various helper and fitting functions for processing measurement curves.
- preprocessing

  Contains frequently used workflows in dataset preprocessing.
- read_csv_datasets

  Defines datatypes for further processing and containtes the means to convert commonly encountered measurement files into them.
- visualization

  Various helper and plotting functions for common data visualizations.

5.51 utility.characteristics Namespace Reference

This file calculates and displays battery characteristics and timescales from a 1+1D-DFN parameter set as it used in PyBaMM.

5.51.1 Detailed Description

This file calculates and displays battery characteristics and timescales from a 1+1D-DFN parameter set as it used in PyBaMM.

5.52 utility.fitting_functions Namespace Reference

Various helper and fitting functions for processing measurement curves.

5.52.1 Detailed Description

Various helper and fitting functions for processing measurement curves.

5.53 utility.preprocessing Namespace Reference

Contains frequently used workflows in dataset preprocessing.

5.53.1 Detailed Description

Contains frequently used workflows in dataset preprocessing.

The functions herein are a collection of simple, but frequent, transformations of arrays of raw measurement data.

5.54 utility.read_csv_datasets Namespace Reference

Defines datatypes for further processing and containtes the means to convert commonly encountered measurement files into them.

5.54.1 Detailed Description

Defines datatypes for further processing and containtes the means to convert commonly encountered measurement files into them.

5.55 utility.visualization Namespace Reference

Various helper and plotting functions for common data visualizations.

5.55.1 Detailed Description

Various helper and plotting functions for common data visualizations.

6 Class Documentation

6.1 Python.models.analytic_impedance.AnalyticImpedance Class Reference

Analytic impedance of the SPMe.

Inherits object.

Public Member Functions

- def __init__ (self, parameters)

    Preprocesses the model parameters.
- def Z_SPM (self, s_dim)

    Transfer function U(s) / I(s) of the SPM.
- def Z_SPM_offset (self)

    Static part of the transfer function of the SPM.
- def A_s (self, s)

    Integration constant; please refer to the PDF.
- def B_s (self, s)

    Integration constant; please refer to the PDF.
- def A_n (self, s)

    Integration constant; please refer to the PDF.
- def A_p (self, s)

    Integration constant; please refer to the PDF.
- def stable_coefficients (self, s)

    Integration constants; please refer to the PDF.
- def Z_SPMe_1 (self, s_dim)

    Additional correction to the SPM's transfer function.
- def Z_SPMe_1_offset (self)

    Static part of the correction to the SPM's impedance.
- def Z_SPMe (self, s_dim)

    Transfer function U(s) / I(s) of the SPMe.
- def Z_SPMe_offset (self)

    Static part of the transfer function of the SPMe.
- def Z_SPM_halfcell (self, s_dim)

    Transfer function U(s) / I(s) of the half-cell SPM.
- def Z_SPM_offset_halfcell (self)

    Static part of the half-cell SPM's transfer function.
- def A_p_halfcell (self, s)

    Integration constant; please refer to the PDF.
- def A_s_halfcell (self, s)

    Integration constant; please refer to the PDF.
- def B_s_halfcell (self, s)

    Integration constant; please refer to the PDF.
- def bar_c_{ep}_1_halfcell (self, s)

    Integration constant; please refer to the PDF.
- def left_c_{es}_1_halfcell (self, s)

    Integration constant; please refer to the PDF.
- def stable_coefficients_halfcell (self, s)

    Integration constants; please refer to the PDF.
- def Z_SPMe_1_halfcell (self, s_dim)

    Correction to the half-cell SPM's transfer function.
- def Z_SPMe_1_offset_halfcell (self)

    Static part of the half-cell SPMe(1)'s impedance.
- def Z_SPMe_halfcell (self, s_dim)

    Transfer function U(s) / I(s) of the half-cell SPMe.
- def Z_SPMe_offset_halfcell (self)

    Static part of the half-cell SPMe's transfer function.

Public Attributes

- parameters

  The parameter dictionary converted to PyBaMM form.
- symbolic_constants

  All relevant constants in a dictionary.
- constants
- R_p_int

  Positive electrode intercalation resistance.
- Z_p_int

  Positive electrode intercalation impedance scaling.
- R_n_int

  Negative electrode intercalation resistance.
- Z_n_int

  Negative electrode intercalation impedance scaling.
- bar_c_{ep}_0

  Positive electrolyte concentration offset.
- Z_c_{ep}

  Positive electrolyte concentration impedance scaling.
- left_c_{es}_0

  Separator electrolyte concentration offset.
- Z_c_{es}

  Separator electrolyte concentration impedance scaling.
- bar_c_{en}_0

  Negative electrolyte concentration offset.
- Z_c_{en}

  Negative electrolyte concentration impedance scaling.
- R_{se}

  Charge transfer resistance.
- R_e

  Electrolyte conductivity resistance.
- R_e_halfcell

  Electrolyte conductivity resistance for half-cell.
- R_s

  Electrode conductivity resistance.
- R_s_halfcell

  Electrode conductivity resistance for half-cell.
- timescale

  Timescale used for non-dimensionalization in s.
- thermal_voltage

  Voltage used for non-dimensionalization in V.
- C

  C-rate of the battery in A.
- C_e

  Non-dimensionalized timescale of the electrolyte.
- D_e

  Non-dimensionalized diffusivity of the electrolyte.
- β_p

  Bruggeman coefficient of the cathode.
- β_s

  Bruggeman coefficient of the separator.

- β_n

  Bruggeman coefficient of the anode.
- ε_p

  Porosity of the cathode.
- ε_s

  Porosity of the separator.
- ε_n

  Porosity of the anode.
- L_p

  Fraction of the cathode length of the battery length.
- L_s

  Fraction of the separator length of the battery length.
- L_n

  Fraction of the anode length of the battery length.
- t_plus

  Transference number.
- γ_e

  Non-dimensionalized electrolyte concentration.
- one_plus_dlnf_dlnc

  Thermodynamic factor.
- z_p

  Charge number for the cathode.
- z_n

  Charge number for the anode.

6.1.1 Detailed Description

Analytic impedance of the SPMe.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 __init__()

```
def Python.models.analytic_impedance.AnalyticImpedance.__init__ (
                self,
                parameters )
```

Preprocesses the model parameters.

Parameters

parameters	A dictionary of parameter values with the parameter names as keys. For these names, please refer to models.standard_parameters.

6.1.3 Member Function Documentation

6.1.3.1 A_n()

def Python.models.analytic_impedance.AnalyticImpedance.A_n (
self,
s)

Integration constant; please refer to the PDF.

Integration constant of the solution of c_e^1 in the SPMe. Note: compared to its formulation in the appended PDF, they have already been scaled with I(s).

Parameters

s	The non-dimensionalized inverse frequencies as an array.

Returns

Integration constant of the solution of c_e^1 in the SPMe.

6.1.3.2 A_p()

def Python.models.analytic_impedance.AnalyticImpedance.A_p (
self,
s)

Integration constant; please refer to the PDF.

Integration constant of the solution of c_e^1 in the SPMe. Note: compared to its formulation in the appended PDF, they have already been scaled with I(s).

Parameters

s	The non-dimensionalized inverse frequencies as an array.

Returns

Integration constant of the solution of c_e^1 in the SPMe.

6.1.3.3 A_p_halfcell()

def Python.models.analytic_impedance.AnalyticImpedance.A_p_halfcell (
self,
s)

Integration constant; please refer to the PDF.

Integration constant of the solution of c_e^1 in the half-cell SPMe. Note: compared to its formulation in the appended PDF, they have already been scaled with I(s).

Parameters

s	The non-dimensionalized inverse frequencies as an array.

Returns

Integration constant of the solution of c_e^1 in the half-cell SPMe.

6.1.3.4 A_s()

```
def Python.models.analytic_impedance.AnalyticImpedance.As (
                self,
                s )
```

Integration constant; please refer to the PDF.

Integration constant of the solution of c_e^1 in the SPMe. Note: compared to its formulation in the appended PDF, they have already been scaled with I(s).

Parameters

s	The non-dimensionalized inverse frequencies as an array.

Returns

Integration constant of the solution of c_e^1 in the SPMe.

6.1.3.5 A_s_halfcell()

```
def Python.models.analytic_impedance.AnalyticImpedance.As_halfcell (
                self,
                s )
```

Integration constant; please refer to the PDF.

Integration constant of the solution of c_e^1 in the half-cell SPMe. Note: compared to its formulation in the appended PDF, they have already been scaled with I(s).

Parameters

s	The non-dimensionalized inverse frequencies as an array.

Returns

Integration constant of the solution of c_e^1 in the half-cell SPMe.

6.1.3.6 bar_c_{ep}_1_halfcell()

def Python.models.analytic_impedance.AnalyticImpedance.bar_c_{ep}_1_halfcell (
self,
s)

Integration constant; please refer to the PDF.

Integration constant of the solution of c_e^1 in the half-cell SPMe. Note: compared to its formulation in the appended PDF, they have already been scaled with I(s).

Parameters

s	The non-dimensionalized inverse frequencies as an array.

Returns

Integration constant of the solution of c_e^1 in the half-cell SPMe.

6.1.3.7 B_s()

def Python.models.analytic_impedance.AnalyticImpedance.B_s (
self,
s)

Integration constant; please refer to the PDF.

Integration constant of the solution of c_e^1 in the SPMe. Note: compared to its formulation in the appended PDF, they have already been scaled with I(s).

Parameters

s	The non-dimensionalized inverse frequencies as an array.

Returns

Integration constant of the solution of c_e^1 in the SPMe.

6.1.3.8 B_s_halfcell()

def Python.models.analytic_impedance.AnalyticImpedance.B_s_halfcell (
self,
s)

Integration constant; please refer to the PDF.

Integration constant of the solution of c_e^1 in the half-cell SPMe. Note: compared to its formulation in the appended PDF, they have already been scaled with I(s).

Parameters

s	The non-dimensionalized inverse frequencies as an array.

Returns

Integration constant of the solution of c_e^1 in the half-cell SPMe.

6.1.3.9 left_c_{es}_1_halfcell()

def Python.models.analytic_impedance.AnalyticImpedance.left_c_{es}_1_halfcell (
self,
s)

Integration constant; please refer to the PDF.

Integration constant of the solution of c_e^1 in the half-cell SPMe. Note: compared to its formulation in the appended PDF, they have already been scaled with I(s).

Parameters

s	The non-dimensionalized inverse frequencies as an array.

Returns

Integration constant of the solution of c_e^1 in the half-cell SPMe.

6.1.3.10 stable_coefficients()

def Python.models.analytic_impedance.AnalyticImpedance.stable_coefficients (
self,
s)

Integration constants; please refer to the PDF.

Integration constants of the solution of c_e^1 in the SPMe. Note: compared to their formulation in the appended PDF, they have already been scaled with I(s).

Compute $A_s(s)$, $B_s(s)$, $A_n(s)$ and $A_p(s)$ numerically, since the analytic formulation above is numerically unstable for high ω.

Parameters

s	The non-dimensionalized inverse frequencies as an array.

Returns

Integration constants of the solution of c_e^1 in the SPMe as a tuple: (A_n, A_s, B_s, A_p).

6.1.3.11 stable_coefficients_halfcell()

def Python.models.analytic_impedance.AnalyticImpedance.stable_coefficients_halfcell (
self,
s)

Integration constants; please refer to the PDF.

Integration constants of the solution of c_e^1 in the half-cell SPMe. Note: compared to their formulation in the appended PDF, they have already been scaled with I(s).

Compute A_p(s), A_s(s) and B_s(s) numerically, since the analytic formulation above is numerically unstable for high ω.

Parameters

s	The non-dimensionalized inverse frequencies as an array.

Returns

Integration constants of the solution of c_e^1 in the SPMe as a tuple: (A_p, A_s, B_s).

6.1.3.12 Z_SPM()

def Python.models.analytic_impedance.AnalyticImpedance.Z_SPM (
self,
s_dim)

Transfer function U(s) / I(s) of the SPM.

Parameters

s_dim	An array of the inverse frequencies to evaluate.

Returns

The evaluated impedances as an array.

6.1.3.13 Z_SPM_halfcell()

def Python.models.analytic_impedance.AnalyticImpedance.Z_SPM_halfcell (
self,
s_dim)

Transfer function U(s) / I(s) of the half-cell SPM.

Parameters

s_dim	An array of the inverse frequencies to evaluate.

Returns

The evaluated impedances as an array.

6.1.3.14 Z_SPM_offset()

def Python.models.analytic_impedance.AnalyticImpedance.Z_SPM_offset (
self)

Static part of the transfer function of the SPM.

Returns

The part of the SPM's impedance that doesn't depend on frequency.

6.1.3.15 Z_SPM_offset_halfcell()

def Python.models.analytic_impedance.AnalyticImpedance.Z_SPM_offset_halfcell (
self)

Static part of the half-cell SPM's transfer function.

Returns

The part of the half-cell SPM's impedance that doesn't depend on frequency.

6.1.3.16 Z_SPMe()

def Python.models.analytic_impedance.AnalyticImpedance.Z_SPMe (
self,
s_dim)

Transfer function U(s) / I(s) of the SPMe.

Parameters

s_dim	An array of the inverse frequencies to evaluate.

Returns

The evaluated impedances as an array.

6.1.3.17 Z_SPMe_1()

def Python.models.analytic_impedance.AnalyticImpedance.Z_SPMe_1 (
self,
s_dim)

Additional correction to the SPM's transfer function.

Parameters

s_dim	An array of the inverse frequencies to evaluate.

Returns

The evaluated impedances as an array.

6.1.3.18 Z_SPMe_1_halfcell()

def Python.models.analytic_impedance.AnalyticImpedance.Z_SPMe_1_halfcell (
self,
s_dim)

Correction to the half-cell SPM's transfer function.

Parameters

s_dim	An array of the inverse frequencies to evaluate.

Returns

The evaluated impedances as an array.

6.1.3.19 Z_SPMe_1_offset()

def Python.models.analytic_impedance.AnalyticImpedance.Z_SPMe_1_offset (
self)

Static part of the correction to the SPM's impedance.

Returns

The part of the difference between the SPM's and SPMe's impedance that doesn't depend on frequency.

6.1.3.20 Z_SPMe_1_offset_halfcell()

def Python.models.analytic_impedance.AnalyticImpedance.Z_SPMe_1_offset_halfcell (
self)

Static part of the half-cell SPMe(1)'s impedance.

Returns

The part of the difference between the SPM's and SPMe's impedance for half-cells that doesn't depend on frequency.

6.1.3.21 Z_SPMe_halfcell()

def Python.models.analytic_impedance.AnalyticImpedance.Z_SPMe_halfcell (
self,
s_dim)

Transfer function U(s) / I(s) of the half-cell SPMe.

Parameters

s_dim	An array of the inverse frequencies to evaluate.

Returns

The evaluated impedances as an array.

6.1.3.22 Z_SPMe_offset()

def Python.models.analytic_impedance.AnalyticImpedance.Z_SPMe_offset (
self)

Static part of the transfer function of the SPMe.

Returns

The part of the SPMe's impedance that doesn't depend on frequency.

6.1.3.23 Z_SPMe_offset_halfcell()

def Python.models.analytic_impedance.AnalyticImpedance.Z_SPMe_offset_halfcell (
    self)

Static part of the half-cell SPMe's transfer function.

Returns

The part of the half-cell SPMe's impedance that doesn't depend on frequency.

6.1.4 Member Data Documentation

6.1.4.1 bar_c_{en}_0

Python.models.analytic_impedance.AnalyticImpedance.bar_c_{en}_0

Negative electrolyte concentration offset.

6.1.4.2 bar_c_{ep}_0

Python.models.analytic_impedance.AnalyticImpedance.bar_c_{ep}_0

Positive electrolyte concentration offset.

6.1.4.3 C

Python.models.analytic_impedance.AnalyticImpedance.C

C-rate of the battery in A.

6.1.4.4 constants

Python.models.analytic_impedance.AnalyticImpedance.constants

6.1.4.5 C_e

Python.models.analytic_impedance.AnalyticImpedance.C_e

Non-dimensionalized timescale of the electrolyte.

6.1.4.6 D_e

Python.models.analytic_impedance.AnalyticImpedance.D_e

Non-dimensionalized diffusivity of the electrolyte.

6.1.4.7 left_c_{es}_0

Python.models.analytic_impedance.AnalyticImpedance.left_c_{es}_0

Separator electrolyte concentration offset.

6.1.4.8 L_n

Python.models.analytic_impedance.AnalyticImpedance.L_n

Fraction of the anode length of the battery length.

6.1.4.9 L_p

Python.models.analytic_impedance.AnalyticImpedance.L_p

Fraction of the cathode length of the battery length.

6.1.4.10 L_s

Python.models.analytic_impedance.AnalyticImpedance.L_s

Fraction of the separator length of the battery length.

6.1.4.11 one_plus_dlnf_dlnc

Python.models.analytic_impedance.AnalyticImpedance.one_plus_dlnf_dlnc

Thermodynamic factor.

6.1.4.12 parameters

Python.models.analytic_impedance.AnalyticImpedance.parameters

The parameter dictionary converted to PyBaMM form.

6.1.4.13 R_e

Python.models.analytic_impedance.AnalyticImpedance.R_e

Electrolyte conductivity resistance.

6.1.4.14 R_e_halfcell

Python.models.analytic_impedance.AnalyticImpedance.R_e_halfcell

Electrolyte conductivity resistance for half-cell.

6.1.4.15 R_n_int

Python.models.analytic_impedance.AnalyticImpedance.R_n_int

Negative electrode intercalation resistance.

6.1.4.16 R_p_int

Python.models.analytic_impedance.AnalyticImpedance.R_p_int

Positive electrode intercalation resistance.

6.1.4.17 R_s

Python.models.analytic_impedance.AnalyticImpedance.R_s

Electrode conductivity resistance.

6.1.4.18 R_s_halfcell

Python.models.analytic_impedance.AnalyticImpedance.R_s_halfcell

Electrode conductivity resistance for half-cell.

6.1.4.19 R_{se}

Python.models.analytic_impedance.AnalyticImpedance.R_{se}

Charge transfer resistance.

6.1.4.20 symbolic_constants

Python.models.analytic_impedance.AnalyticImpedance.symbolic_constants

All relevant constants in a dictionary.

Note: any term here that corresponds to U^1 - $\eta_p{}^1$ + $\eta_n{}^1$ is already scaled by C_e / I. All constants herein are duplicated as members after this. Their descriptions match their names (the keys of this dictionary).

6.1.4.21 t_plus

Python.models.analytic_impedance.AnalyticImpedance.t_plus

Transference number.

6.1.4.22 thermal_voltage

Python.models.analytic_impedance.AnalyticImpedance.thermal_voltage

Voltage used for non-dimensionalization in V.

6.1.4.23 timescale

Python.models.analytic_impedance.AnalyticImpedance.timescale

Timescale used for non-dimensionalization in s.

6.1.4.24 Z_c$_{en}$

Python.models.analytic_impedance.AnalyticImpedance.Z_c$_{en}$

Negative electrolyte concentration impedance scaling.

6.1.4.25 Z_c$_{ep}$

Python.models.analytic_impedance.AnalyticImpedance.Z_c$_{ep}$

Positive electrolyte concentration impedance scaling.

6.1.4.26 Z_c$_{es}$

Python.models.analytic_impedance.AnalyticImpedance.Z_c$_{es}$

Separator electrolyte concentration impedance scaling.

6.1.4.27 z_n

Python.models.analytic_impedance.AnalyticImpedance.z_n

Charge number for the anode.

6.1.4.28 Z_n_int

Python.models.analytic_impedance.AnalyticImpedance.Z_n_int

Negative electrode intercalation impedance scaling.

6.1.4.29 z_p

Python.models.analytic_impedance.AnalyticImpedance.z_p

Charge number for the cathode.

6.1.4.30 Z_p_int

Python.models.analytic_impedance.AnalyticImpedance.Z_p_int

Positive electrode intercalation impedance scaling.

6.1.4.31 β_n

Python.models.analytic_impedance.AnalyticImpedance.β_n

Bruggeman coefficient of the anode.

6.1.4.32 β_p

Python.models.analytic_impedance.AnalyticImpedance.β_p

Bruggeman coefficient of the cathode.

6.1.4.33 β_s

Python.models.analytic_impedance.AnalyticImpedance.β_s

Bruggeman coefficient of the separator.

6.1.4.34 γ_e

Python.models.analytic_impedance.AnalyticImpedance.γ_e

Non-dimensionalized electrolyte concentration.

6.1.4.35 ε_n

Python.models.analytic_impedance.AnalyticImpedance.ε_n

Porosity of the anode.

6.1.4.36 ε_p

Python.models.analytic_impedance.AnalyticImpedance.ε_p

Porosity of the cathode.

6.1.4.37 ε_s

Python.models.analytic_impedance.AnalyticImpedance.ε_s

Porosity of the separator.

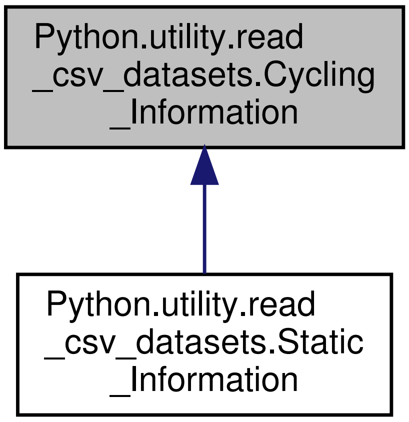
The documentation for this class was generated from the following file:

- models/analytic_impedance.py

6.2 Python.utility.read_csv_datasets.Cycling_Information Class Reference

Contains basic cycling informations.

Inheritance diagram for Python.utility.read_csv_datasets.Cycling_Information:


Python.utility.read_csv_datasets.Cycling_Information
Python.utility.read_csv_datasets.Static_Information

Public Member Functions

- def __init__ (self, timepoints, currents, voltages, other_columns={}, indices=None)
- def subslice (self, start, stop, step=1)

  Selects a subslice of the data segments.
- def subarray (self, array)

  Selects the data segments with the given indices.

Public Attributes

- timepoints

  The times at which measurements were taken.
- currents

  The measured current at those times.
- voltages

  The measured voltage at those times.
- other_columns

  The contents of any other columns.
- indices

  The indices of the individual segments.

6.2.1 Detailed Description

Contains basic cycling informations.

Each member variable is a list and has the same length as the other ones.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 __init__()

```
def Python.utility.read_csv_datasets.Cycling_Information.__init__ (
                    self,
                    timepoints,
                    currents,
                    voltages,
                    other_columns = {},
                    indices = None )
```

6.2.3 Member Function Documentation

6.2.3.1 subarray()

```
def Python.utility.read_csv_datasets.Cycling_Information.subarray (
        self,
        array )
```

Selects the data segments with the given indices.

array

The indices of the segments that are to be returned.

Returns

A new Cycling_Information object containing the subset.

6.2.3.2 subslice()

```
def Python.utility.read_csv_datasets.Cycling_Information.subslice (
        self,
        start,
        stop,
        step = 1 )
```

Selects a subslice of the data segments.

The arguments exactly match the slice(...) notation.

start

The index of the first segment to be included.

stop

The index of the first segment to not be included.

step

Steps between selected segments. Default: 1.

Returns

A new Cycling_Information object containing the slice.

6.2.4 Member Data Documentation

6.2.4.1 currents

Python.utility.read_csv_datasets.Cycling_Information.currents

The measured current at those times.

A list which usually contains lists for segments with variable current and floats for segments with constant current.

6.2.4.2 indices

Python.utility.read_csv_datasets.Cycling_Information.indices

The indices of the individual segments.

Defaults to a simple numbering of the segments present. May be used for plotting purposes, e.g. for colorcoding the segments by cycle.

6.2.4.3 other_columns

Python.utility.read_csv_datasets.Cycling_Information.other_columns

The contents of any other columns.

A dictionary ("columns") which values are lists which contain lists for segments. The keys should match user input for the columns.

6.2.4.4 timepoints

Python.utility.read_csv_datasets.Cycling_Information.timepoints

The times at which measurements were taken.

Usually a list of lists where each list corresponds to a segment of the measurement.

6.2.4.5 voltages

Python.utility.read_csv_datasets.Cycling_Information.voltages

The measured voltage at those times.

A list which usually contains lists for segments with variable voltage and floats for segments with constant voltage.

The documentation for this class was generated from the following file:

- utility/read_csv_datasets.py

6.3 Python.models.DFN.DFN Class Reference

Doyle-Fuller-Newman (DFN) model.

Inherits BaseBatteryModel.

Public Member Functions

- def __init__ (self, name="DFN", halfcell=False, pybamm_control=False, build=True)

  Sets up a DFN model usable by PyBaMM.
- def set_standard_output_variables (self)

  Adds "the horizontal axis" to self.variables.
- def new_copy (self, build=True)

  Create an empty copy with identical options.
- def set_voltage_variables (self)

  Adds voltage-specific variables to self.variables.
- def default_geometry (self)

  Override: corrects the geometry for half-cells.

Public Attributes

- halfcell

  Equations build a full-cell if False and a half-cell if True.
- pybamm_control

  Current is fixed if False and a variable if True.
- param

  Contains all the relevant parameters for this model.
- timescale

  Non-dimensionalization timescale.
- length_scales

  Non-dimensionalization length scales.

6.3.1 Detailed Description

Doyle-Fuller-Newman (DFN) model.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 __init__()

```
def Python.models.DFN.DFN.__init__ (
            self,
            name = "DFN",
            halfcell = False,
            pybamm_control = False,
            build = True )
```

Sets up a DFN model usable by PyBaMM.

name

The optional name of the model. Default is "DFN".

halfcell

Per default False, which indicates a full-cell setup. If set to True, the equations for a half-cell will be used instead.

pybamm_control

Per default False, which indicates that the current is given as a function. If set to True, this model is compatible with PyBaMM experiments, e.g. CC-CV simulations. The current is then a variable and it or voltage can be fixed functions.

build

Per default True, which builds the model equations right away. If set to False, they remain as symbols for later.

6.3.3 Member Function Documentation

6.3.3.1 default_geometry()

def Python.models.DFN.DFN.default_geometry (
self)

Override: corrects the geometry for half-cells.

6.3.3.2 new_copy()

def Python.models.DFN.DFN.new_copy (
self,
build = True)

Create an empty copy with identical options.

build

If True, the new model gets built right away. This is the default behavior. If set to False, it remains as symbols.

Returns

The copy of this model.

6.3.3.3 set_standard_output_variables()

def Python.models.DFN.DFN.set_standard_output_variables (
self)

Adds ”the horizontal axis” to self.variables.

Don't use the super() version of this function, as it introduces keys with integer values in self.variables.

6.3.3.4 set_voltage_variables()

def Python.models.DFN.DFN.set_voltage_variables (
self)

Adds voltage-specific variables to self.variables.

Override this inherited function, since it adds superfluous variables.

6.3.4 Member Data Documentation

6.3.4.1 halfcell

Python.models.DFN.DFN.halfcell

Equations build a full-cell if False and a half-cell if True.

6.3.4.2 length_scales

Python.models.DFN.DFN.length_scales

Non-dimensionalization length scales.

6.3.4.3 param

Python.models.DFN.DFN.param

Contains all the relevant parameters for this model.

6.3.4.4 pybamm_control

Python.models.DFN.DFN.pybamm_control

Current is fixed if False and a variable if True.

6.3.4.5 timescale

Python.models.DFN.DFN.timescale

Non-dimensionalization timescale.

The documentation for this class was generated from the following file:

- models/DFN.py

6.4 Python.models.DFN.DFN_internal Class Reference

Defining equations for a Doyle-Fuller-Newman-Model.

Inherits BaseSubModel.

Public Member Functions

- def __init__ (self, param, halfcell=False, pybamm_control=False)

  Sets the model properties.
- def get_fundamental_variables (self)

  Builds all relevant model variables' symbols.
- def get_coupled_variables (self, variables)

  Builds all model symbols that rely on other models.
- def set_rhs (self, variables)

  Sets up the right-hand-side equations in self.rhs.
- def set_algebraic (self, variables)

  Sets up the algebraic equations in self.algebraic.
- def set_boundary_conditions (self, variables)

  Sets the (self.)boundary(_)conditions.
- def set_initial_conditions (self, variables)

  Sets the (self.)initial(_)conditions.
- def set_events (self, variables)

  Sets up the termination switches in self.events.

Public Attributes

- halfcell

  Equations build a full-cell if False and a half-cell if True.
- pybamm_control

  Current is fixed if False and a variable if True.

6.4.1 Detailed Description

Defining equations for a Doyle-Fuller-Newman-Model.

Reference

SG Marquis, V Sulzer, R Timms, CP Please and SJ Chapman. “An asymptotic derivation of a single particle model with electrolyte”. Journal of The Electrochemical Society, 166(15):A3693–A3706, 2019

6.4.2 Constructor & Destructor Documentation

6.4.2.1 __init__()

def Python.models.DFN.DFN_internal.__init__ (
self,
param,
halfcell = False,
pybamm_control = False)

Sets the model properties.

param

A class containing all the relevant parameters for this model. For example, models.standard_parameters represents a valid choice for this parameter.

halfcell

Per default False, which indicates a full-cell setup. If set to True, the equations for a half-cell will be used instead.

pybamm_control

Per default False, which indicates that the current is given as a function. If set to True, this model is compatible with PyBaMM experiments, e.g. CC-CV simulations. The current is then a variable and it or voltage can be fixed functions.

6.4.3 Member Function Documentation

6.4.3.1 get_coupled_variables()

def Python.models.DFN.DFN_internal.get_coupled_variables (
self,
variables)

Builds all model symbols that rely on other models.

variables

A dictionary containing at least all variable symbols that are required for the variable symbols built here.

Returns

A dictionary with the new variables' names as keys and their symbols (of type pybamm.Symbol) as values.

6.4.3.2 get_fundamental_variables()

def Python.models.DFN.DFN_internal.get_fundamental_variables (
self)

Builds all relevant model variables' symbols.

Returns

A dictionary with the variables' names as keys and their symbols (of type pybamm.Symbol) as values.

6.4.3.3 set_algebraic()

def Python.models.DFN.DFN_internal.set_algebraic (
self,
variables)

Sets up the algebraic equations in self.algebraic.

6.4.3.4 set_boundary_conditions()

def Python.models.DFN.DFN_internal.set_boundary_conditions (
self,
variables)

Sets the (self.)boundary(_)conditions.

6.4.3.5 set_events()

def Python.models.DFN.DFN_internal.set_events (
self,
variables)

Sets up the termination switches in self.events.

6.4.3.6 set_initial_conditions()

def Python.models.DFN.DFN_internal.set_initial_conditions (
self,
variables)

Sets the (self.)initial(_)conditions.

6.4.3.7 set_rhs()

```
def Python.models.DFN.DFN_internal.set_rhs (
                        self,
                        variables )
```

Sets up the right-hand-side equations in self.rhs.

6.4.4 Member Data Documentation

6.4.4.1 halfcell

Python.models.DFN.DFN_internal.halfcell

Equations build a full-cell if False and a half-cell if True.

6.4.4.2 pybamm_control

Python.models.DFN.DFN_internal.pybamm_control

Current is fixed if False and a variable if True.

The documentation for this class was generated from the following file:

- models/DFN.py

6.5 Python.utility.visualization.HandlerColorLineCollection Class Reference

Automates multicolored lines in legends.

Inherits HandlerLineCollection.

Public Member Functions

- def create_artists (self, legend, artist, xdescent, ydescent, width, height, fontsize, trans)
  Please refer to the Matplotlib documentation for details.

6.5.1 Detailed Description

Automates multicolored lines in legends.

Taken from https://stackoverflow.com/questions/49223702/ adding-a-legend-to-a-matplotlib-plot-with-a-multicolored-line

6.5.2 Member Function Documentation

6.5.2.1 create_artists()

def Python.utility.visualization.HandlerColorLineCollection.create_artists (
self,
legend,
artist,
xdescent,
ydescent,
width,
height,
fontsize,
trans)

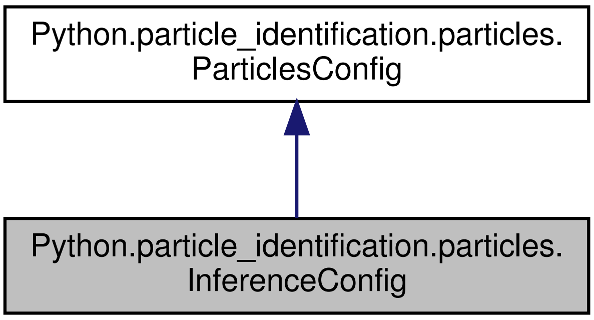
Please refer to the Matplotlib documentation for details.

The documentation for this class was generated from the following file:

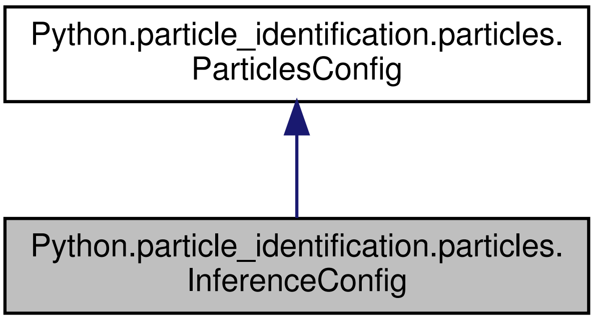
- utility/visualization.py

6.6 Python.particle_identification.particles.InferenceConfig Class Reference

Inheritance diagram for Python.particle_identification.particles.InferenceConfig:


Python.particle_identification.particles.
ParticlesConfig
Python.particle_identification.particles.
InferenceConfig

Collaboration diagram for Python.particle_identification.particles.InferenceConfig:


Python.particle_identification.particles.
ParticlesConfig
Python.particle_identification.particles.
InferenceConfig

Static Public Attributes

- int GPU_COUNT = 3
- int IMAGES_PER_GPU = 1

6.6.1 Member Data Documentation

6.6.1.1 GPU_COUNT

int Python.particle_identification.particles.InferenceConfig.GPU_COUNT = 3 [static]

6.6.1.2 IMAGES_PER_GPU

int Python.particle_identification.particles.InferenceConfig.IMAGES_PER_GPU = 1 [static]

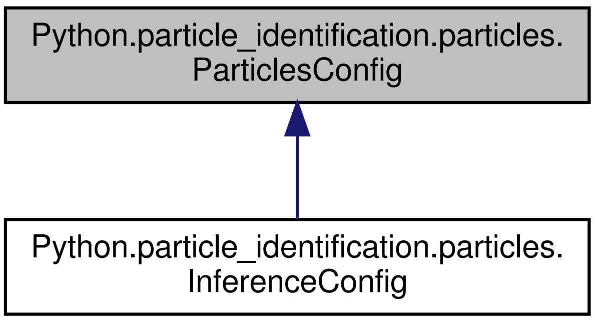
The documentation for this class was generated from the following file:

- particle_identification/particles.py

6.7 Python.particle_identification.particles.ParticlesConfig Class Reference

Configurations.

Inheritance diagram for Python.particle_identification.particles.ParticlesConfig:


Python.particle_identification.particles.
ParticlesConfig
Python.particle_identification.particles.
InferenceConfig

Static Public Attributes

- string NAME = ”particles”

    Give the configuration a recognizable name.
- int IMAGES_PER_GPU = 1

    We use a GPU with 12GB memory, which can fit two images.
- int NUM_CLASSES = 1 + 1

    Number of classes (including background)
- int STEPS_PER_EPOCH = 100

    Number of training steps per epoch.
- float DETECTION_MIN_CONFIDENCE = 0.9

    Skip detections with $<$ 90% confidence.
- int GPU_COUNT = 1

    Number of GPUs to run on.
- string IMAGE_RESIZE_MODE = ”square”

    The normalization procedure.
- int IMAGE_MIN_DIM = 800

    The minimum size of the normalized image.
- int IMAGE_MAX_DIM = 1024

    The maximum size of the normalized image.
- int BATCH_SIZE = 1

    The batch size (must match the images passed at once).

6.7.1 Detailed Description

Configurations.

Configuration for training on the toy dataset.

Derives from the base Config class and overrides some values.

6.7.2 Member Data Documentation

6.7.2.1 BATCH_SIZE

int Python.particle_identification.particles.ParticlesConfig.BATCH_SIZE = 1 [static]

The batch size (must match the images passed at once).

6.7.2.2 DETECTION_MIN_CONFIDENCE

float Python.particle_identification.particles.ParticlesConfig.DETECTION_MIN_CONFIDENCE = 0.9 [static]

Skip detections with $<$ 90% confidence.

6.7.2.3 GPU_COUNT

int Python.particle_identification.particles.ParticlesConfig.GPU_COUNT = 1 [static]

Number of GPUs to run on.

6.7.2.4 IMAGE_MAX_DIM

int Python.particle_identification.particles.ParticlesConfig.IMAGE_MAX_DIM = 1024 [static]

The maximum size of the normalized image.

6.7.2.5 IMAGE_MIN_DIM

int Python.particle_identification.particles.ParticlesConfig.IMAGE_MIN_DIM = 800 [static]

The minimum size of the normalized image.

6.7.2.6 IMAGE_RESIZE_MODE

string Python.particle_identification.particles.ParticlesConfig.IMAGE_RESIZE_MODE = "square" [static]

The normalization procedure.

6.7.2.7 IMAGES_PER_GPU

int Python.particle_identification.particles.ParticlesConfig.IMAGES_PER_GPU = 1 [static]

We use a GPU with 12GB memory, which can fit two images.

Adjust down if you use a smaller GPU.

6.7.2.8 NAME

string Python.particle_identification.particles.ParticlesConfig.NAME = "particles" [static]

Give the configuration a recognizable name.

6.7.2.9 NUM_CLASSES

int Python.particle_identification.particles.ParticlesConfig.NUM_CLASSES = 1 + 1 [static]

Number of classes (including background)

6.7.2.10 STEPS_PER_EPOCH

int Python.particle_identification.particles.ParticlesConfig.STEPS_PER_EPOCH = 100 [static]

Number of training steps per epoch.

The documentation for this class was generated from the following file:

- particle_identification/particles.py

6.8 Python.particle_identification.particles.ParticlesDataset Class Reference

Provides access to the toy dataset.

Inherits Dataset.

Public Member Functions

- def load_particles (self, dataset_dir, subset)

  Load a subset of the Balloon dataset.
- def load_mask (self, image_id)

  Generate instance masks for an image.
- def image_reference (self, image_id)

  Return the path of the image.
- def watershed (self, image_id)

  Apply a simple watershed transform.

6.8.1 Detailed Description

Provides access to the toy dataset.

Derives from the base Dataset class and overrides some values.

6.8.2 Member Function Documentation

6.8.2.1 image_reference()

def Python.particle_identification.particles.ParticlesDataset.image_reference (
self,
image_id)

Return the path of the image.

6.8.2.2 load_mask()

def Python.particle_identification.particles.ParticlesDataset.load_mask (
self,
image_id)

Generate instance masks for an image.

Returns

masks: A bool array of shape [height, width, instance count] with one mask per instance. class_ids: a 1D array of class IDs of the instance masks.

6.8.2.3 load_particles()

def Python.particle_identification.particles.ParticlesDataset.load_particles (
self,
dataset_dir,
subset)

Load a subset of the Balloon dataset.

dataset_dir

Root directory of the dataset.

subset

Subset to load: train or val

6.8.2.4 watershed()

def Python.particle_identification.particles.ParticlesDataset.watershed (
self,
image_id)

Apply a simple watershed transform.

The documentation for this class was generated from the following file:

- particle_identification/particles.py

6.9 Python.optimization.run_estimation.SmartFormatter Class Reference

Initialize the parser for the command-line parameters.

Inherits HelpFormatter.

6.9.1 Detailed Description

Initialize the parser for the command-line parameters.

This specialization enables the use of
to define line breaks if the description string begins with 'R|'.

The documentation for this class was generated from the following file:

- optimization/run_estimation.py

6.10 Python.models.SPM.SPM Class Reference

Single-Particle Model (SPM).

Inherits BaseBatteryModel.

Public Member Functions

- def __init__ (self, name="SPM", halfcell=False, pybamm_control=False, build=True)

  Sets up a SPM model usable by PyBaMM.
- def set_standard_output_variables (self)

  Adds "the horizontal axis" to self.variables.
- def new_copy (self, build=True)

  Create an empty copy with identical options.
- def set_voltage_variables (self)

  Adds voltage-specific variables to self.variables.
- def default_geometry (self)

  Override: corrects the geometry for half-cells.

Public Attributes

- halfcell

  Equations build a full-cell if False and a half-cell if True.
- pybamm_control

  Current is fixed if False and a variable if True.
- param

  Contains all the relevant parameters for this model.
- timescale

  Non-dimensionalization timescale.
- length_scales

  Non-dimensionalization length scales.

6.10.1 Detailed Description

Single-Particle Model (SPM).

6.10.2 Constructor & Destructor Documentation

6.10.2.1 __init__()

def Python.models.SPM.SPM.__init__ (
self,
name = "SPM",
halfcell = False,
pybamm_control = False,
build = True)

Sets up a SPM model usable by PyBaMM.

name

The optional name of the model. Default is "SPM".

halfcell

Per default False, which indicates a full-cell setup. If set to True, the equations for a half-cell will be used instead.

pybamm_control

Per default False, which indicates that the current is given as a function. If set to True, this model is compatible with PyBaMM experiments, e.g. CC-CV simulations. The current is then a variable and it or voltage can be fixed functions.

build

Per default True, which builds the model equations right away. If set to False, they remain as symbols for later.

6.10.3 Member Function Documentation

6.10.3.1 default_geometry()

def Python.models.SPM.SPM.default_geometry (
self)

Override: corrects the geometry for half-cells.

6.10.3.2 new_copy()

def Python.models.SPM.SPM.new_copy (
self,
build = True)

Create an empty copy with identical options.

build

If True, the new model gets built right away. This is the default behavior. If set to False, it remains as symbols.

Returns

The copy of this model.

6.10.3.3 set_standard_output_variables()

def Python.models.SPM.SPM.set_standard_output_variables (
self)

Adds ”the horizontal axis” to self.variables.

Don't use the super() version of this function, as it introduces keys with integer values in self.variables.

6.10.3.4 set_voltage_variables()

def Python.models.SPM.SPM.set_voltage_variables (
self)

Adds voltage-specific variables to self.variables.

Override this inherited function, since it adds superfluous variables.

6.10.4 Member Data Documentation

6.10.4.1 halfcell

Python.models.SPM.SPM.halfcell

Equations build a full-cell if False and a half-cell if True.

6.10.4.2 length_scales

Python.models.SPM.SPM.length_scales

Non-dimensionalization length scales.

6.10.4.3 param

Python.models.SPM.SPM.param

Contains all the relevant parameters for this model.

6.10.4.4 pybamm_control

Python.models.SPM.SPM.pybamm_control

Current is fixed if False and a variable if True.

6.10.4.5 timescale

Python.models.SPM.SPM.timescale

Non-dimensionalization timescale.

The documentation for this class was generated from the following file:

- models/SPM.py

6.11 Python.models.SPM.SPM_internal Class Reference

Defining equations for a Single-Particle Model (SPM).

Inherits BaseSubModel.

Public Member Functions

- def __init__ (self, param, halfcell=False, pybamm_control=False)

  Sets the model properties.
- def get_fundamental_variables (self)

  Builds all relevant model variables' symbols.
- def get_coupled_variables (self, variables)

  Builds all model symbols that rely on other models.
- def set_rhs (self, variables)

  Sets up the right-hand-side equations in self.rhs.
- def set_algebraic (self, variables)

  Sets up the algebraic equations in self.algebraic.
- def set_boundary_conditions (self, variables)

  Sets the (self.)boundary(_)conditions.
- def set_initial_conditions (self, variables)

  Sets the (self.)initial(_)conditions.
- def set_events (self, variables)

  Sets up the termination switches in self.events.

Public Attributes

- halfcell

  Equations build a full-cell if False and a half-cell if True.
- pybamm_control

  Current is fixed if False and a variable if True.

6.11.1 Detailed Description

Defining equations for a Single-Particle Model (SPM).

Reference

SG Marquis, V Sulzer, R Timms, CP Please and SJ Chapman. "An asymptotic derivation of a single particle model with electrolyte". Journal of The Electrochemical Society, 166(15):A3693–A3706, 2019

6.11.2 Constructor & Destructor Documentation

6.11.2.1 __init__()

def Python.models.SPM.SPM_internal.__init__ (
self,
param,
halfcell = False,
pybamm_control = False)

Sets the model properties.

param

A class containing all the relevant parameters for this model. For example, models.standard_parameters represents a valid choice for this parameter.

halfcell

Per default False, which indicates a full-cell setup. If set to True, the equations for a half-cell will be used instead.

pybamm_control

Per default False, which indicates that the current is given as a function. If set to True, this model is compatible with PyBaMM experiments, e.g. CC-CV simulations. The current is then a variable and it or voltage can be fixed functions.

6.11.3 Member Function Documentation

6.11.3.1 get_coupled_variables()

def Python.models.SPM.SPM_internal.get_coupled_variables (
self,
variables)

Builds all model symbols that rely on other models.

variables

A dictionary containing at least all variable symbols that are required for the variable symbols built here.

Returns

A dictionary with the new variables' names as keys and their symbols (of type pybamm.Symbol) as values.

6.11.3.2 get_fundamental_variables()

def Python.models.SPM.SPM_internal.get_fundamental_variables (
self)

Builds all relevant model variables' symbols.

Returns

A dictionary with the variables' names as keys and their symbols (of type pybamm.Symbol) as values.

6.11.3.3 set_algebraic()

def Python.models.SPM.SPM_internal.set_algebraic (
self,
variables)

Sets up the algebraic equations in self.algebraic.

6.11.3.4 set_boundary_conditions()

def Python.models.SPM.SPM_internal.set_boundary_conditions (
self,
variables)

Sets the (self.)boundary(_)conditions.

6.11.3.5 set_events()

def Python.models.SPM.SPM_internal.set_events (
self,
variables)

Sets up the termination switches in self.events.

6.11.3.6 set_initial_conditions()

def Python.models.SPM.SPM_internal.set_initial_conditions (
self,
variables)

Sets the (self.)initial(_)conditions.

6.11.3.7 set_rhs()

def Python.models.SPM.SPM_internal.set_rhs (
self,
variables)

Sets up the right-hand-side equations in self.rhs.

6.11.4 Member Data Documentation

6.11.4.1 halfcell

Python.models.SPM.SPM_internal.halfcell

Equations build a full-cell if False and a half-cell if True.

6.11.4.2 pybamm_control

Python.models.SPM.SPM_internal.pybamm_control

Current is fixed if False and a variable if True.

The documentation for this class was generated from the following file:

- models/SPM.py

6.12 Python.models.SPMe.SPMe Class Reference

Single-Particle Model with electrolyte (SPMe).

Inherits BaseBatteryModel.

Public Member Functions

- def __init__ (self, name="SPMe", halfcell=False, pybamm_control=False, build=True)

  Sets up a SPMe model usable by PyBaMM.
- def set_standard_output_variables (self)

  Adds "the horizontal axis" to self.variables.
- def new_copy (self, build=True)

  Create an empty copy with identical options.
- def set_voltage_variables (self)

  Adds voltage-specific variables to self.variables.
- def default_geometry (self)

  Override: corrects the geometry for half-cells.

Public Attributes

- halfcell

  Equations build a full-cell if False and a half-cell if True.
- pybamm_control

  Current is fixed if False and a variable if True.
- param

  Contains all the relevant parameters for this model.
- timescale

  Non-dimensionalization timescale.
- length_scales

  Non-dimensionalization length scales.

6.12.1 Detailed Description

Single-Particle Model with electrolyte (SPMe).

6.12.2 Constructor & Destructor Documentation

6.12.2.1 __init__()

```
def Python.models.SPMe.SPMe.__init__ (
                    self,
                    name = "SPMe",
                    halfcell = False,
                    pybamm_control = False,
                    build = True )
```

Sets up a SPMe model usable by PyBaMM.

name

The optional name of the model. Default is ”SPMe”.

halfcell

Per default False, which indicates a full-cell setup. If set to True, the equations for a half-cell will be used instead.

pybamm_control

Per default False, which indicates that the current is given as a function. If set to True, this model is compatible with PyBaMM experiments, e.g. CC-CV simulations. The current is then a variable and it or voltage can be fixed functions.

build

Per default True, which builds the model equations right away. If set to False, they remain as symbols for later.

6.12.3 Member Function Documentation

6.12.3.1 default_geometry()

```
def Python.models.SPMe.SPMe.default_geometry (
                    self )
```

Override: corrects the geometry for half-cells.

6.12.3.2 new_copy()

```
def Python.models.SPMe.SPMe.new_copy (
                    self,
                    build = True )
```

Create an empty copy with identical options.

build

If True, the new model gets built right away. This is the default behavior. If set to False, it remains as symbols.

Returns

The copy of this model.

6.12.3.3 set_standard_output_variables()

def Python.models.SPMe.SPMe.set_standard_output_variables (
self)

Adds ”the horizontal axis” to self.variables.

Don't use the super() version of this function, as it introduces keys with integer values in self.variables.

6.12.3.4 set_voltage_variables()

def Python.models.SPMe.SPMe.set_voltage_variables (
self)

Adds voltage-specific variables to self.variables.

Override this inherited function, since it adds superfluous variables.

6.12.4 Member Data Documentation

6.12.4.1 halfcell

Python.models.SPMe.SPMe.halfcell

Equations build a full-cell if False and a half-cell if True.

6.12.4.2 length_scales

Python.models.SPMe.SPMe.length_scales

Non-dimensionalization length scales.

6.12.4.3 param

Python.models.SPMe.SPMe.param

Contains all the relevant parameters for this model.

6.12.4.4 pybamm_control

Python.models.SPMe.SPMe.pybamm_control

Current is fixed if False and a variable if True.

6.12.4.5 timescale

Python.models.SPMe.SPMe.timescale

Non-dimensionalization timescale.

The documentation for this class was generated from the following file:

- models/SPMe.py

6.13 Python.models.SPMe.SPMe_internal Class Reference

Defining equations for the SPMe.

Inherits BaseSubModel.

Public Member Functions

- def __init__ (self, param, halfcell=False, pybamm_control=False)
  Sets the model properties.
- def get_fundamental_variables (self)
  Builds all relevant model variables' symbols.
- def get_coupled_variables (self, variables)
  Builds all model symbols that rely on other models.
- def set_rhs (self, variables)
  Sets up the right-hand-side equations in self.rhs.
- def set_algebraic (self, variables)
  Sets up the algebraic equations in self.algebraic.
- def set_boundary_conditions (self, variables)
  Sets the (self.)boundary(_)conditions.
- def set_initial_conditions (self, variables)
  Sets the (self.)initial(_)conditions.
- def set_events (self, variables)
  Sets up the termination switches in self.events.

Public Attributes

- halfcell
  Equations build a full-cell if False and a half-cell if True.
- pybamm_control
  Current is fixed if False and a variable if True.

6.13.1 Detailed Description

Defining equations for the SPMe.

Reference

SG Marquis, V Sulzer, R Timms, CP Please and SJ Chapman. “An asymptotic derivation of a single particle model with electrolyte”. Journal of The Electrochemical Society, 166(15):A3693–A3706, 2019

6.13.2 Constructor & Destructor Documentation

6.13.2.1 __init__()

```
def Python.models.SPMe.SPMe_internal.__init__ (
                    self,
                    param,
                    halfcell = False,
                    pybamm_control = False )
```

Sets the model properties.

param

A class containing all the relevant parameters for this model. For example, models.standard_parameters represents a valid choice for this parameter.

halfcell

Per default False, which indicates a full-cell setup. If set to True, the equations for a half-cell will be used instead.

pybamm_control

Per default False, which indicates that the current is given as a function. If set to True, this model is compatible with PyBaMM experiments, e.g. CC-CV simulations. The current is then a variable and it or voltage can be fixed functions.

6.13.3 Member Function Documentation

6.13.3.1 get_coupled_variables()

```
def Python.models.SPMe.SPMe_internal.get_coupled_variables (
                    self,
                    variables )
```

Builds all model symbols that rely on other models.

variables

A dictionary containing at least all variable symbols that are required for the variable symbols built here.

Returns

A dictionary with the new variables' names as keys and their symbols (of type pybamm.Symbol) as values.

6.13.3.2 get_fundamental_variables()

def Python.models.SPMe.SPMe_internal.get_fundamental_variables (
    self)

Builds all relevant model variables' symbols.

Returns
A dictionary with the variables' names as keys and their symbols (of type pybamm.Symbol) as values.

6.13.3.3 set_algebraic()

def Python.models.SPMe.SPMe_internal.set_algebraic (
    self,
    variables)

Sets up the algebraic equations in self.algebraic.

6.13.3.4 set_boundary_conditions()

def Python.models.SPMe.SPMe_internal.set_boundary_conditions (
    self,
    variables)

Sets the (self.)boundary(_)conditions.

6.13.3.5 set_events()

def Python.models.SPMe.SPMe_internal.set_events (
    self,
    variables)

Sets up the termination switches in self.events.

6.13.3.6 set_initial_conditions()

def Python.models.SPMe.SPMe_internal.set_initial_conditions (
    self,
    variables)

Sets the (self.)initial(_)conditions.

6.13.3.7 set_rhs()

def Python.models.SPMe.SPMe_internal.set_rhs (
self,
variables)

Sets up the right-hand-side equations in self.rhs.

6.13.4 Member Data Documentation

6.13.4.1 halfcell

Python.models.SPMe.SPMe_internal.halfcell

Equations build a full-cell if False and a half-cell if True.

6.13.4.2 pybamm_control

Python.models.SPMe.SPMe_internal.pybamm_control

Current is fixed if False and a variable if True.

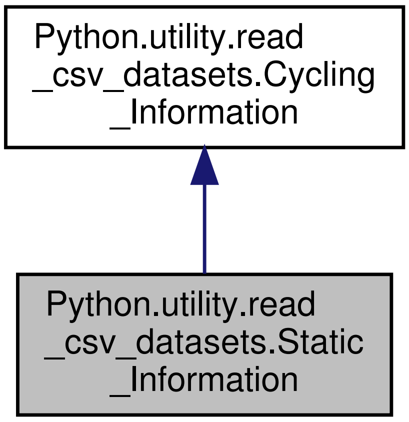
The documentation for this class was generated from the following file:

- models/SPMe.py

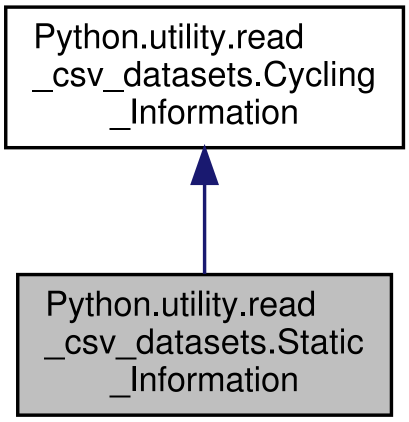
6.14 Python.utility.read_csv_datasets.Static_Information Class Reference

Contains additional informations, e.g.

Inheritance diagram for Python.utility.read_csv_datasets.Static_Information:


Python.utility.read _csv_datasets.Cycling _Information
Python.utility.read _csv_datasets.Static _Information

Collaboration diagram for Python.utility.read_csv_datasets.Static_Information:


Python.utility.read_csv_datasets.Cycling_Information
Python.utility.read_csv_datasets.Static_Information

Public Member Functions

- def __init__ (self, timepoints, currents, voltages, asymptotic_voltages, ir_steps, exp_I_decays, exp_U_↩ decays, other_columns={}, indices=None)
- def subslice (self, start, stop, step=1)

  Selects a subslice of the data segments.
- def subarray (self, array)

  Selects the data segments with the given indices.

Public Attributes

- asymptotic_voltages

  The voltages that the voltage curve seems to converge to in a segment.
- ir_steps

  The instantaneous IR drops before each segment.
- exp_I_decays

  Same as exp_U_decays for current decays (PITT).
- exp_U_decays

  The fit parameters of the exponential voltage decays in each segment.

6.14.1 Detailed Description

Contains additional informations, e.g.

for GITT.

Each member variable is a list and has the same length as the other ones.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 __init__()

def Python.utility.read_csv_datasets.Static_Information.__init__ (
self,
timepoints,
currents,
voltages,
asymptotic_voltages,
ir_steps,
exp_I_decays,
exp_U_decays,
other_columns = {},
indices = None)

6.14.3 Member Function Documentation

6.14.3.1 subarray()

def Python.utility.read_csv_datasets.Static_Information.subarray (
self,
array)

Selects the data segments with the given indices.

array

The indices of the segments that are to be returned.

Returns

A new Static_Information object containing the subset.

6.14.3.2 subslice()

def Python.utility.read_csv_datasets.Static_Information.subslice (
self,
start,
stop,
step = 1)

Selects a subslice of the data segments.

The arguments exactly match the slice(...) notation.

start

The index of the first segment to be included.

stop

The index of the first segment to not be included.

step

Steps between selected segments. Default: 1.

Returns

A new Static_Information object containing the slice.

6.14.4 Member Data Documentation

6.14.4.1 asymptotic_voltages

Python.utility.read_csv_datasets.Static_Information.asymptotic_voltages

The voltages that the voltage curve seems to converge to in a segment.

Only makes sense for those segments that are rest periods or when the OCV was subtracted.

6.14.4.2 exp_I_decays

Python.utility.read_csv_datasets.Static_Information.exp_I_decays

Same as exp_U_decays for current decays (PITT).

6.14.4.3 exp_U_decays

Python.utility.read_csv_datasets.Static_Information.exp_U_decays

The fit parameters of the exponential voltage decays in each segment.

Each set of fit parameters is a 3-tuple (a,b,c) where the fit function has the following form: a + b ∗ exp(-c ∗ (t - t_end_of_segment)). Failed or missing fits are best indicated by (NaN, NaN, NaN).

6.14.4.4 ir_steps

Python.utility.read_csv_datasets.Static_Information.ir_steps

The instantaneous IR drops before each segment.

Positive values are voltage rises and negative values voltage drops.

The documentation for this class was generated from the following file:

- utility/read_csv_datasets.py

7 File Documentation

7.1 __init__.py File Reference

Namespaces

- Python

7.2 models/__init__.py File Reference

Namespaces

- Python.models

7.3 optimization/__init__.py File Reference

Namespaces

- Python.optimization

7.4 parameters/__init__.py File Reference

Namespaces

- Python.parameters

7.5 parameters/estimation/__init__.py File Reference

Namespaces

- Python.parameters.estimation

7.6 parameters/models/__init__.py File Reference

Namespaces

- Python.parameters.models

7.7 particle_identification/__init__.py File Reference

Namespaces

- Python.particle_identification

7.8 utility/__init__.py File Reference

Namespaces

- Python.utility

7.9 analytic_impedance_visualization.py File Reference

Example usage of the analytic impedance models.

Namespaces

- Python.analytic_impedance_visualization

Variables

- Python.analytic_impedance_visualization.measurement = read_voltages_from_csv('../Datensätze/Simolka←↩_EIS.csv')
- Python.analytic_impedance_visualization.AI = AnalyticImpedance(parameters)
- int Python.analytic_impedance_visualization.nop = 100
- Python.analytic_impedance_visualization.ω = np.exp(np.linspace(np.log(0.001), np.log(0.31623), nop))
- int Python.analytic_impedance_visualization.s = 1j ∗ ω
- Python.analytic_impedance_visualization.fig
- Python.analytic_impedance_visualization.ax
- Python.analytic_impedance_visualization.figsize
- Python.analytic_impedance_visualization.Z_SPM = AI.Z_SPM(s) - AI.Z_SPM_offset()
- Python.analytic_impedance_visualization.Z_SPMe_1 = AI.Z_SPMe_1(s) - AI.Z_SPMe_1_offset()
- float Python.analytic_impedance_visualization.Z_SPMe = 0.001 ∗ (Z_SPM + 20 ∗ Z_SPMe_1)
- Python.analytic_impedance_visualization.legend_points
- int Python.analytic_impedance_visualization.data_x = 80 ∗ np.array(measurement[0])
- float Python.analytic_impedance_visualization.data_y = 0.5 ∗ np.array(measurement[1])
- Python.analytic_impedance_visualization.lw
- Python.analytic_impedance_visualization.ls
- Python.analytic_impedance_visualization.label

7.9.1 Detailed Description

Example usage of the analytic impedance models.

7.10 cc_cv_visualization.py File Reference

Example usage of the CC-CV-cycling visualization tools.

Namespaces

- Python.cc_cv_visualization

Variables

- Python.cc_cv_visualization.fig
- Python.cc_cv_visualization.ax
- Python.cc_cv_visualization.figsize
- Python.cc_cv_visualization.check = cc_cv_visualization(fig, ax, dataset, max_number_of_clusters)

7.10.1 Detailed Description

Example usage of the CC-CV-cycling visualization tools.

7.11 dis_charge_visualisation.py File Reference

Example usage of the visualization tools for discharge.

Namespaces

- Python.dis_charge_visualisation

Variables

- float Python.dis_charge_visualisation.U_offset = 0.035
- float Python.dis_charge_visualisation.voltages_without_OCV = np.array(voltages_without_OCV) - U_↩ offset
- tuple Python.dis_charge_visualisation.experiment = (capacities ∗ 3600 / current, voltages)
- dictionary Python.dis_charge_visualisation.parameters_to_try
- Python.dis_charge_visualisation.parameters_list
- Python.dis_charge_visualisation.combinations
- list Python.dis_charge_visualisation.solutions = [ ]
- list Python.dis_charge_visualisation.models = [DFN(halfcell=False, pybamm_control=False)]
- Python.dis_charge_visualisation.model = model.new_copy()
- Python.dis_charge_visualisation.fig
- Python.dis_charge_visualisation.ax
- Python.dis_charge_visualisation.figsize
- Python.dis_charge_visualisation.title
- Python.dis_charge_visualisation.ylabel
- Python.dis_charge_visualisation.xlabel
- Python.dis_charge_visualisation.feature_visualizer
- Python.dis_charge_visualisation.interactive_plot

7.11.1 Detailed Description

Example usage of the visualization tools for discharge.

7.12 estimate_ocv_from_cc_cv.py File Reference

Example usage of CC-CV analysis tools for OCV fitting.

Namespaces

- Python.estimate_ocv_from_cc_cv

Variables

- Python.estimate_ocv_from_cc_cv.figsize
- Python.estimate_ocv_from_cc_cv.nrows
- Python.estimate_ocv_from_cc_cv.ncols
- Python.estimate_ocv_from_cc_cv.cathode_phases
- Python.estimate_ocv_from_cc_cv.spline_smoothing

7.12.1 Detailed Description

Example usage of CC-CV analysis tools for OCV fitting.

7.13 gitt_visualization.py File Reference

Example usage of the visualization tools for GITT.

Namespaces

- Python.gitt_visualization

Variables

- dictionary Python.gitt_visualization.parameters_to_try
- Python.gitt_visualization.parameters_list
- Python.gitt_visualization.combinations
- list Python.gitt_visualization.solutions = [ ]
- list Python.gitt_visualization.models = [DFN(halfcell=True, pybamm_control=True)]
- Python.gitt_visualization.model = model.new_copy()
- Python.gitt_visualization.simulator
- Python.gitt_visualization.fig
- Python.gitt_visualization.ax
- Python.gitt_visualization.figsize
- Python.gitt_visualization.title
- Python.gitt_visualization.xlabel
- Python.gitt_visualization.ylabel

7.13.1 Detailed Description

Example usage of the visualization tools for GITT.

7.14 identify_particles.py File Reference

Example usage of the classification tools for particle identification.

Namespaces

- Python.identify_particles

Variables

- Python.identify_particles.config = particles.ParticlesConfig()
- Python.identify_particles.model
- Python.identify_particles.by_name
- Python.identify_particles.orig_image = skimage.io.imread("../Datensätze/RE_93_cathode_20mu.png")
- Python.identify_particles.image = skimage.io.imread("../Datensätze/RE_93_cathode_20mu_edited2.png")
- Python.identify_particles.result = model.detect([image], verbose=0)[0]
- Python.identify_particles.bbox = utils.extract_bboxes(result['masks'])
- Python.identify_particles.fig
- Python.identify_particles.ax
- Python.identify_particles.figsize
- Python.identify_particles.title
- Python.identify_particles.show_mask
- Python.identify_particles.True
- Python.identify_particles.show_bbox

7.14.1 Detailed Description

Example usage of the classification tools for particle identification.

7.15 measurement_plot.py File Reference

Example usage of the general visualization tool.

Namespaces

- Python.measurement_plot

Variables

- Python.measurement_plot.fig
- Python.measurement_plot.ax
- Python.measurement_plot.figsize
- Python.measurement_plot.constrained_layout

7.15.1 Detailed Description

Example usage of the general visualization tool.

7.16 models/analytic_impedance.py File Reference

Contains equations to evaluate the analytic impedance of the SPMe model.

Classes

- class Python.models.analytic_impedance.AnalyticImpedance

    Analytic impedance of the SPMe.

Namespaces

- Python.models.analytic_impedance

Functions

- def Python.models.analytic_impedance.complex_clip (x, lower, upper)

  Implements numpy.clip for complex values.

7.16.1 Detailed Description

Contains equations to evaluate the analytic impedance of the SPMe model.

7.17 models/DFN.py File Reference

Classes

- class Python.models.DFN.DFN_internal

  Defining equations for a Doyle-Fuller-Newman-Model.
- class Python.models.DFN.DFN

  Doyle-Fuller-Newman (DFN) model.

Namespaces

- Python.models.DFN
- models.DFN

  Contains a PyBaMM-compatible Doyle-Fuller-Newman (DFN) model.

7.18 models/solversetup.py File Reference

Namespaces

- Python.models.solversetup
- models.solversetup

  This file eases the setup and simulation of PyBaMM battery models.

Functions

- def Python.models.solversetup.solver_setup (model, parameters, submesh_types, var_pts, spatial_↩ methods, geometry=None, reltol=1e-6, abstol=1e-6, root_tol=1e-6, verbose=False)

  Processes the model and returns a runnable solver.
- def Python.models.solversetup.simulation_setup (model, input, parameters, submesh_types, var_pts, spatial_methods, geometry=None, reltol=1e-6, abstol=1e-6, root_tol=1e-6, verbose=False)

  Processes the model and returns a runnable solver.
- def Python.models.solversetup.auto_var_pts (x_n, x_s, x_p, r_n, r_p, y=1, z=1)

  Utility function for setting the discretization density.
- def Python.models.solversetup.spectral_mesh_pts_and_method (order_s_n, order_s_p, order_e, volumes↩ _e_n=1, volumes_e_s=1, volumes_e_p=1, halfcell=False)

  Utility function for default mesh and spatial methods.

7.19 models/SPM.py File Reference

Classes

- class Python.models.SPM.SPM_internal

  Defining equations for a Single-Particle Model (SPM).
- class Python.models.SPM.SPM

  Single-Particle Model (SPM).

Namespaces

- Python.models.SPM
- models.SPM

  Contains a PyBaMM-compatible Single-Particle Model (SPM).

7.20 models/SPMe.py File Reference

Classes

- class Python.models.SPMe.SPMe_internal

  Defining equations for the SPMe.
- class Python.models.SPMe.SPMe

  Single-Particle Model with electrolyte (SPMe).

Namespaces

- Python.models.SPMe
- models.SPMe

  Contains a PyBaMM-compatible Single-Particle Model with electrolyte (SPMe).

7.21 models/standard_parameters.py File Reference

Comprehensive list of parameters of every model.

Namespaces

- Python.models.standard_parameters

Functions

- def Python.models.standard_parameters.SOC$_n$_dim_init (x)

  Initial SOC of the anode.
- def Python.models.standard_parameters.SOC$_n$_init (x)

  Non-dimensionalized initial SOC of the anode.
- def Python.models.standard_parameters.SOC$_p$_dim_init (x)

  Initial SOC of the cathode.
- def Python.models.standard_parameters.SOC$_p$_init (x)

  Non-dimensionalized initial SOC of the cathode.
- def Python.models.standard_parameters.D$_e$_dim (c$_e$_dim, T_dim)

  Electrolyte diffusivity.
- def Python.models.standard_parameters.D$_e$ (c$_e$, T)

  Non-dimensionalized electrolyte diffusivity.
- def Python.models.standard_parameters.κ_e_dim (c$_e$_dim, T_dim)

  Electrolyte conductivity.
- def Python.models.standard_parameters.κ_e (c$_e$, T)

  Non-dimensionalized electrolyte conductivity.
- def Python.models.standard_parameters.t_plus_dim (c$_e$_dim)

  Transference number.
- def Python.models.standard_parameters.t_plus (c$_e$)

  Non-dimensionalized (referring to the input) transference number.
- def Python.models.standard_parameters.one_plus_dlnf_dlnc_dim (c$_e$_dim)

  Thermodynamic factor.
- def Python.models.standard_parameters.one_plus_dlnf_dlnc (c$_e$)

  Non-dimensionalized (referring to the input) thermodynamic factor.
- def Python.models.standard_parameters.D$_n$_dim (SOC$_n$, T_dim)

  Anode diffusivity.
- def Python.models.standard_parameters.D$_n$ (SOC$_n$, T)

  Non-dimensionalized anode diffusivity.
- def Python.models.standard_parameters.D$_p$_dim (SOC$_p$, T_dim)

  Cathode diffusivity.
- def Python.models.standard_parameters.D$_p$ (SOC$_p$, T)

  Non-dimensionalized cathode diffusivity.
- def Python.models.standard_parameters.i$_{sen}$_0_dim (c$_{en}$_dim, SOC$_n$_surf_dim, T_dim)

  Anode exchange current density.
- def Python.models.standard_parameters.i$_{sen}$_0 (c$_{en}$, SOC$_n$_surf, T)

  Non-dimensionalized anode exchange current density.
- def Python.models.standard_parameters.d_c$_{en}$_i$_{sen}$_0_dim (c$_{en}$_dim, SOC$_n$_surf_dim, T_dim)

  ∂ anode exchange current density / ∂ electrolyte concentration.
- def Python.models.standard_parameters.d_c$_{en}$_i$_{sen}$_0 (c$_{en}$, SOC$_n$_surf, T)

  The non-dimensionalized version of the prior variable.
- def Python.models.standard_parameters.i$_{sep}$_0_dim (c$_{ep}$_dim, SOC$_p$_surf_dim, T_dim)

  Cathode exchange current density.
- def Python.models.standard_parameters.i$_{sep}$_0 (c$_{ep}$, SOC$_p$_surf, T)

  Non-dimensionalized cathode exchange current density.
- def Python.models.standard_parameters.d_c$_{ep}$_i$_{sep}$_0_dim (c$_{ep}$_dim, SOC$_p$_surf_dim, T_dim)

  ∂ cathode exchange current density / ∂ electrolyte concentration.
- def Python.models.standard_parameters.d_c$_{ep}$_i$_{sep}$_0 (c$_{ep}$, SOC$_p$_surf, T)

  The non-dimensionalized version of the prior variable.
- def Python.models.standard_parameters.dOCV$_n$_dT_dim (SOC$_n$)

∂ anode OCV / ∂ temperature.

- def Python.models.standard_parameters.dOCV$_n$_dT (SOC$_n$)

  Non-dimensionalized ∂ anode OCV / ∂ temperature.
- def Python.models.standard_parameters.dOCV$_n$_dT_dSOC$_n$_dim (SOC$_n$)

  (∂ anode OCV / ∂ temperature) / ∂ anode SOC.
- def Python.models.standard_parameters.dOCV$_n$_dT_dSOC$_n$ (SOC$_n$)

  Non-dimensionalized (∂ anode OCV / ∂ temperature) / ∂ anode SOC.
- def Python.models.standard_parameters.dOCV$_p$_dT_dim (SOC$_p$)

  ∂ cathode OCV / ∂ temperature.
- def Python.models.standard_parameters.dOCV$_p$_dT (SOC$_p$)

  Non-dimensionalized ∂ cathode OCV / ∂ temperature.
- def Python.models.standard_parameters.dOCV$_p$_dT_dSOC$_p$_dim (SOC$_p$)

  (∂ cathode OCV / ∂ temperature) / ∂ cathode SOC.
- def Python.models.standard_parameters.dOCV$_p$_dT_dSOC$_p$ (SOC$_p$)

  Non-dimensionalized (∂ cathode OCV / ∂ temperature) / ∂ cathode SOC.
- def Python.models.standard_parameters.OCV$_n$_dim (SOC$_n$, T_dim)

  Anode OCV.
- def Python.models.standard_parameters.OCV$_n$ (SOC$_n$, T)

  Non-dimensionalized anode OCV.
- def Python.models.standard_parameters.dOCV$_n$_dim_dSOC$_n$ (SOC$_n$, T_dim)

  ∂ anode OCV / ∂ anode SOC.
- def Python.models.standard_parameters.dOCV$_n$_dSOC$_n$ (SOC$_n$, T)

  Non-dimensionalized ∂ anode OCV / ∂ anode SOC.
- def Python.models.standard_parameters.OCV$_p$_dim (SOC$_p$, T_dim)

  Cathode OCV.
- def Python.models.standard_parameters.OCV$_p$ (SOC$_p$, T)

  Non-dimensionalized cathode OCV.
- def Python.models.standard_parameters.dOCV$_p$_dim_dSOC$_p$ (SOC$_p$, T_dim)

  ∂ cathode OCV / ∂ cathode SOC.
- def Python.models.standard_parameters.dOCV$_p$_dSOC$_p$ (SOC$_p$, T)

  Non-dimensionalized ∂ cathode OCV / ∂ cathode SOC.

Variables

- Python.models.standard_parameters.R = Scalar(constants.R)

  Gas constant.
- Python.models.standard_parameters.F = Scalar(constants.physical_constants["Faraday constant"][0])

  Faraday constant.
- Python.models.standard_parameters.k_B = constants.physical_constants["Boltzmann constant"][0]

  Boltzmann constant.
- Python.models.standard_parameters.q$_e$ = constants.physical_constants["electron volt"][0]

  Electron volt.
- Python.models.standard_parameters.T_ref = Parameter("Reference temperature [K]")

  Reference temperature for non-dimensionalization.
- Python.models.standard_parameters.T_init = Parameter("Initial temperature [K]")

  Initial temperature.
- Python.models.standard_parameters.thermal_voltage = R ∗ T_ref / F
- Python.models.standard_parameters.ΔT = Scalar(1)

  Typical temperature rise.
- Python.models.standard_parameters.L$_n$_dim = Parameter("Negative electrode thickness [m]")

Anode thickness.

- Python.models.standard_parameters.L_s_dim = Parameter("Separator thickness [m]")

  Seperator thickness.

- Python.models.standard_parameters.L_p_dim = Parameter("Positive electrode thickness [m]")

  Cathode thickness.

- Python.models.standard_parameters.L_dim = L_n_dim + L_s_dim + L_p_dim

  Cell thickness.

- Python.models.standard_parameters.A = Parameter("Current collector perpendicular area [m2]")

  Cross-section area of the cell.

- Python.models.standard_parameters.V = Parameter("Cell volume [m3]")

  Volume of the cell.

- Python.models.standard_parameters.L_x = L_dim

  PyBaMM-compatible name for the cell thickness.

- Python.models.standard_parameters.L_y = Parameter("Electrode width [m]")

  Width of the electrode for PyBaMM compatibility.

- Python.models.standard_parameters.L_z = Parameter("Electrode height [m]")

  Height of the electrode for PyBaMM compatibility.

- Python.models.standard_parameters.C = Parameter("Typical current [A]")

  C-rate of the battery (in A).

- Python.models.standard_parameters.I_typ = C / A

  C-rate of the battery (in A/m²).

- Python.models.standard_parameters.capacity = Parameter("Cell capacity [A.h]")

  Capacity of the cell (in Ah).

- Python.models.standard_parameters.U_l = pybamm.Parameter("Lower voltage cut-off [V]")

  Lower threshold for the cell voltage.

- Python.models.standard_parameters.U_u = pybamm.Parameter("Upper voltage cut-off [V]")

  Upper threshold for the cell voltage.

- Python.models.standard_parameters.c_e_typ = pybamm.Parameter("Typical electrolyte concentration [mol.m-3]")

  Reference electrolyte concentration for non-dimensionalization.

- Python.models.standard_parameters.c_n = pybamm.Parameter("Maximum concentration in negative electrode [mol.m-3]")

  Maximum charge concentration in the anode active material.

- Python.models.standard_parameters.c_p = pybamm.Parameter("Maximum concentration in positive electrode [mol.m-3]")

  Maximum charge concentration in the cathode active material.

- Python.models.standard_parameters.σ_n_dim = pybamm.Parameter("Negative electrode conductivity [S.↩ m-1]")

  Electronic conductivity of the anode.

- Python.models.standard_parameters.σ_p_dim = pybamm.Parameter("Positive electrode conductivity [S.↩ m-1]")

  Electronic conductivity of the cathode.

- Python.models.standard_parameters.a_n_dim = Parameter("Negative electrode surface area to volume ratio [m-1]")

  Specific surface area of the anode.

- Python.models.standard_parameters.a_p_dim = Parameter("Positive electrode surface area to volume ratio [m-1]")

  Specific surface area of the cathode.

- Python.models.standard_parameters.R_n = Parameter("Negative particle radius [m]")

  Mean/Median radius of the anode particles.

- Python.models.standard_parameters.R_p = Parameter("Positive particle radius [m]")

  Mean/Median radius of the cathode particles.

- Python.models.standard_parameters.α_{nn} = Parameter("Negative electrode anodic charge-transfer coefficient")

  Anodic symmetry factor for the anode interface reaction.
- Python.models.standard_parameters.α_{pn} = Parameter("Negative electrode cathodic charge-transfer coefficient")

  Cathodic symmetry factor for the anode interface reaction.
- Python.models.standard_parameters.α_{np} = Parameter("Positive electrode anodic charge-transfer coefficient")

  Anodic symmetry factor for the cathode interface reaction.
- Python.models.standard_parameters.α_{pp} = Parameter("Positive electrode cathodic charge-transfer coefficient")

  Cathodic symmetry factor for the cathode interface reaction.
- Python.models.standard_parameters.β_{en}_scalar = Parameter("Negative electrode Bruggeman coefficient (electrolyte)")

  Bruggeman coefficient for the electrolyte in the anode (scalar).
- Python.models.standard_parameters.β_{es}_scalar = Parameter("Separator Bruggeman coefficient (electrolyte)")

  Bruggeman coefficient for the electrolyte in the separator (scalar).
- Python.models.standard_parameters.β_{ep}_scalar = Parameter("Positive electrode Bruggeman coefficient (electrolyte)")

  Bruggeman coefficient for the electrolyte in the cathode (scalar).
- Python.models.standard_parameters.β_{en} = pybamm.PrimaryBroadcast(β_{en}_scalar, "negative electrode")

  Bruggeman coefficient for the electrolyte in the anode (vector).
- Python.models.standard_parameters.β_{es} = pybamm.PrimaryBroadcast(β_{es}_scalar, "separator")

  Bruggeman coefficient for the electrolyte in the separator (vector).
- Python.models.standard_parameters.β_{ep} = pybamm.PrimaryBroadcast(β_{ep}_scalar, "positive electrode")

  Bruggeman coefficient for the electrolyte in the cathode (vector).
- Python.models.standard_parameters.β_{sn}_scalar = Parameter("Negative electrode Bruggeman coefficient (electrode)")

  Bruggeman coefficient for the solid part in the anode.
- Python.models.standard_parameters.β_{ss}_scalar = Parameter("Separator Bruggeman coefficient (electrode)")

  Bruggeman coefficient for the solid part in the seperator.
- Python.models.standard_parameters.β_{sp}_scalar = Parameter("Positive electrode Bruggeman coefficient (electrode)")

  Bruggeman coefficient for the solid part in the cathode.
- Python.models.standard_parameters.β_{e} = pybamm.Concatenation(β_{en}, β_{es}, β_{ep})

  Bruggeman coefficient for the electrolyte in the whole cell.
- Python.models.standard_parameters.ε_{n}_scalar = Parameter("Negative electrode porosity")

  Porosity of the anode (scalar).
- Python.models.standard_parameters.ε_{s}_scalar = Parameter("Separator porosity")

  Porosity of the separator (scalar).
- Python.models.standard_parameters.ε_{p}_scalar = Parameter("Positive electrode porosity")

  Porosity of the cathode (scalar).
- Python.models.standard_parameters.ε_{n} = pybamm.PrimaryBroadcast(ε_{n}_scalar, "negative electrode")

  Porosity of the anode (vector).
- Python.models.standard_parameters.ε_{s} = pybamm.PrimaryBroadcast(ε_{s}_scalar, "separator")

  Porosity of the separator (vector).
- Python.models.standard_parameters.ε_{p} = pybamm.PrimaryBroadcast(ε_{p}_scalar, "positive electrode")

  Porosity of the cathode (vector).
- Python.models.standard_parameters.ε = pybamm.Concatenation(ε_{n}, ε_{s}, ε_{p})

  Porosity of the whole cell.

- Python.models.standard_parameters.ε^{β} = ε∗∗β_e

  Tortuosity of the whole cell.
- Python.models.standard_parameters.z_n = Parameter("Negative electrode electrons in reaction")

  Charge number of the anode interface reaction.
- Python.models.standard_parameters.z_p = Parameter("Positive electrode electrons in reaction")

  Charge number of the cathode interface reaction.
- Python.models.standard_parameters.c_e_dim_init = Parameter("Initial concentration in electrolyte [mol.↩ m-3]")

  Initial electrolyte concentration.
- Python.models.standard_parameters.c_e_init = c_e_dim_init / c_e_typ

  Non-dimensionalized initial electrolyte concentration.
- def Python.models.standard_parameters.D_e_typ = D_e_dim(c_e_typ, T_ref)

  Reference electrolyte diffusivity for non-dimensionalization.
- def Python.models.standard_parameters.κ_e_typ = κ_e_dim(c_e_typ, T_ref)

  Reference electrolyte conductivity for non-dimensionalization.
- tuple Python.models.standard_parameters.κ_e_hat = (R ∗ T_ref / F) / (I_typ ∗ L_dim / κ_e_typ)

  Thermal voltage divided by ionic resistance.
- def Python.models.standard_parameters.D_n_typ = D_n_dim(Scalar(0.5), T_ref)

  Reference anode diffusivity for non-dimensionalization.
- def Python.models.standard_parameters.D_p_typ = D_p_dim(Scalar(0.5), T_ref)

  Reference cathode diffusivity for non-dimensionalization.
- def Python.models.standard_parameters.i_{sn}_0_ref = i_{sn}_0_dim(c_e_typ, 0.5 ∗ c_n, T_ref)

  Reference anode exchange current density for non-dimensionalization.
- def Python.models.standard_parameters.i_{sep}_0_ref = i_{sep}_0_dim(c_e_typ, 0.5 ∗ c_p, T_ref)

  Reference cathode exchange current density for non-dimensionalization.
- def Python.models.standard_parameters.OCV_n_ref = OCV_n_dim(SOC_n_init(0), T_ref)

  Reference anode OCV for non-dimensionalization.
- def Python.models.standard_parameters.OCV_p_ref = OCV_p_dim(SOC_p_init(1), T_ref)

  Reference cathode OCV for non-dimensionalization.
- Python.models.standard_parameters.a_n = a_n_dim ∗ R_n

  Non-dimensionalized specific surface area of the anode.
- Python.models.standard_parameters.a_p = a_p_dim ∗ R_p

  Non-dimensionalized specific surface area of the cathode.
- tuple Python.models.standard_parameters.Q = (1 - ε_p_scalar) ∗ L_p_dim ∗ c_p ∗ z_p ∗ F ∗ A
- Python.models.standard_parameters.τ^d = F ∗ c_p ∗ L_dim / I_typ

  Discharge timescale.
- int Python.models.standard_parameters.τ_e = L_dim∗∗2 / D_e_typ

  Electrolyte diffusion timescale.
- int Python.models.standard_parameters.τ_n = R_n∗∗2 / D_n_typ

  Anode diffusion timescale.
- int Python.models.standard_parameters.τ_p = R_p∗∗2 / D_p_typ

  Cathode diffusion timescale.
- Python.models.standard_parameters.τ_{rn} = F ∗ c_n / (i_{sn}_0_ref ∗ a_n_dim)

  Anode interface reaction timescale.
- Python.models.standard_parameters.τ_{rp} = F ∗ c_p / (i_{sep}_0_ref ∗ a_p_dim)

  Cathode interface reaction timescale.
- Python.models.standard_parameters.timescale = τ^d

  Choose the discharge timescale for non-dimensionalization.
- int Python.models.standard_parameters.C_e = τ_e / τ^d

  Non-dimensionalized electrolyte diffusion timescale.
- int Python.models.standard_parameters.C_n = τ_n / τ^d

Non-dimensionalized anode diffusion timescale.

- int Python.models.standard_parameters.C_p = τ_p / τ^d

  Non-dimensionalized cathode diffusion timescale.
- Python.models.standard_parameters.C_{rn} = τ_{rn} / τ^d

  Non-dimensionalized anode interface reaction timescale.
- Python.models.standard_parameters.C_{rp} = τ_{rp} / τ^d

  Non-dimensionalized cathode interface reaction timescale.
- Python.models.standard_parameters.γ_e = c_e_typ / c_p

  Non-dimensionalized reference electrolyte concentration.
- Python.models.standard_parameters.γ_n = c_n / c_p

  Non-dimensionalized maximum anode charge concentration.
- Python.models.standard_parameters.γ_p = c_p / c_p

  Non-dimensionalized cathode charge concentration.
- Python.models.standard_parameters.L_n = L_n_dim / L_dim

  Non-dimensionalized anode thickness.
- Python.models.standard_parameters.L_s = L_s_dim / L_dim

  Non-dimensionlized separator thickness.
- Python.models.standard_parameters.L_p = L_p_dim / L_dim

  Non-dimensionalized cathode thickness.
- Python.models.standard_parameters.L_e

  Non-dimensionalized thicknesses for the whole cell.
- tuple Python.models.standard_parameters.σ_n = (thermal_voltage / (I_typ ∗ L_dim)) ∗ σ_n_dim

  Non-dimensionalized electronic conductivity of the anode.
- tuple Python.models.standard_parameters.σ_p = (thermal_voltage / (I_typ ∗ L_dim)) ∗ σ_p_dim

  Non-dimensionalized electronic conductivity of the cathode.
- Python.models.standard_parameters.I_extern_dim

  Externally applied current for galvanostatic operation (in A).
- Python.models.standard_parameters.I_extern = I_extern_dim / C

  Non-dimensionalized external current.
- Python.models.standard_parameters.n_electrodes_parallel

  Current divider.
- Python.models.standard_parameters.n_cells = Parameter("Number of cells connected in series to make a battery")

  Voltage multiplier.
- Python.models.standard_parameters.A_cc = A

  Copy of the cross-section area for PyBaMM compatibility.
- Python.models.standard_parameters.current_with_time = I_extern_dim

  Copy of the external current for PyBaMM compatibility.
- Python.models.standard_parameters.voltage_low_cut = U_l

  Copy of the lower voltage threshold for PyBaMM compatibility.
- Python.models.standard_parameters.voltage_high_cut = U_u

  Copy of the upper voltage threshold for PyBaMM compatibility.

7.21.1 Detailed Description

Comprehensive list of parameters of every model.

SI units are assumed unless stated otherwise. When imported, this package turns into the list of variables contained in here. The "syntactic sugar" with the lower and upper indexed letters is treated by Python to be the same as their normal counterparts. E.g., "a_n" and "an" refer to the exact same variable. Greek letters aren't converted, unless they are also indexed, in which case the same as before applies: "ε^β" is the same as "εβ".

7.22 ocv_visualization.py File Reference

Example usage of the OCV fitting tools.

Namespaces

- Python.ocv_visualization

Variables

- Python.ocv_visualization.ax0
- Python.ocv_visualization.ax1
- Python.ocv_visualization.figsize
- Python.ocv_visualization.ncols
- Python.ocv_visualization.phases
- Python.ocv_visualization.spline_smoothing
- Python.ocv_visualization.inverted
- Python.ocv_visualization.spline_order
- Python.ocv_visualization.sign

7.22.1 Detailed Description

Example usage of the OCV fitting tools.

7.23 optimization/EP_BOLFI.py File Reference

This file contains functions to perform Expectation Propagation on battery models using BOLFI (Bayesian Optimization).

Namespaces

- Python.optimization.EP_BOLFI

Functions

- def Python.optimization.EP_BOLFI.return_simulator (simulator, fixed_parameters, free_parameters, r, Q, experimental_data, feature_extractor, transform_parameters={}, log_of_tried_parameters={})

  Transforms simulator output into covariance eigenvectors.
- def Python.optimization.EP_BOLFI.expectation_propagation (fixed_parameters, free_parameters, simulators, experimental_datasets, feature_extractors, covariance=None, bolfi_initial_evidence=10, bolfi←_total_evidence=100, bolfi_posterior_samples=20, ep_iterations=2, ep_dampener=0.5, ep_dampener←_reduction_steps=1, free_parameters_boundaries=None, transform_parameters={}, display_current_←feature=None, show_trials=False, verbose=False, seed=None)

  Estimates parameters and their uncertainties.

7.23.1 Detailed Description

This file contains functions to perform Expectation Propagation on battery models using BOLFI (Bayesian Optimization).

7.24 optimization/run_estimation.py File Reference

Classes

- class Python.optimization.run_estimation.SmartFormatter

  Initialize the parser for the command-line parameters.

Namespaces

- Python.optimization.run_estimation
- optimization.run_estimation

  This file works with the parameter files in parameters.estimation to run the EP-BOLFI algorithm in optimization.EP↩ _BOLFI.

Functions

- def Python.optimization.run_estimation.neg_int (value)

  Define a method to reject negative integers.

Variables

- Python.optimization.run_estimation.parser = argparse.ArgumentParser(formatter_class=SmartFormatter)

  The command-line input parser.
- Python.optimization.run_estimation.args = parser.parse_args()

  Parses the command-line arguments.
- Python.optimization.run_estimation.opt = run_path(args.estimator_parameters)

  Contains the global variables of the run file.

7.25 parameters/estimation/cccv_inhouse_pouch_cell.py File Reference

Parameter file for the estimation of battery model parameters from the cycling data provided by BASF for one of their in-house pouch cells.

Namespaces

- Python.parameters.estimation.cccv_inhouse_pouch_cell

Variables

- Python.parameters.estimation.cccv_inhouse_pouch_cell.E_0_g = np.array([0.35973, 0.17454, 0.12454, 0.↩ 081957])

    Fitted plateau voltages of a graphite anode.
- Python.parameters.estimation.cccv_inhouse_pouch_cell.γUeminus1_g = np.array([-0.33144, 8.9434e-3, 7.2404e-2, 6.7789e-2])

    Fitted plateau inverse widths of a graphite anode.
- Python.parameters.estimation.cccv_inhouse_pouch_cell.a_g = a_fit(γUeminus1_g)

    Fitted plateau inverse widths (transformed) of a graphite anode.
- Python.parameters.estimation.cccv_inhouse_pouch_cell.Δx_g = np.array([8.041e-2, 0.23299, 0.29691, 0.↩ 39381])

    Fitted plateau SOC fractions of a graphite anode.
- list Python.parameters.estimation.cccv_inhouse_pouch_cell.anode = [p[i] for i in range(4) for p in [E_0_g, a_g, Δx_g]]

    Fit parameters of a graphite anode OCV curve.
- int Python.parameters.estimation.cccv_inhouse_pouch_cell.cathode_phases = 4

    Number of plateaus that are to be fitted to the OCV curve.
- int Python.parameters.estimation.cccv_inhouse_pouch_cell.smoothing_factor = 1e-7

    Lever for adjusting the smoothing spline for the OCV curves.
- Python.parameters.estimation.cccv_inhouse_pouch_cell.complete_dataset

    The experiment dataset, stored as a Cycling_Information object.
- int Python.parameters.estimation.cccv_inhouse_pouch_cell.states_per_cycle = 3

    Number of dataset segments per "cycle" for plotting.
- Python.parameters.estimation.cccv_inhouse_pouch_cell.indices
- Python.parameters.estimation.cccv_inhouse_pouch_cell.dataset = complete_dataset.subslice(81, -2)

    Subset of interest of the complete experiment dataset.
- int Python.parameters.estimation.cccv_inhouse_pouch_cell.max_number_of_clusters = 4

    Maximum number of clusters for the automated labelling.
- Python.parameters.estimation.cccv_inhouse_pouch_cell.charge = dataset.subslice(0, None, states_per_↩ cycle)

    CC charge curves.
- Python.parameters.estimation.cccv_inhouse_pouch_cell.cv = dataset.subslice(1, None, states_per_cycle)

    CV segments between charge and discharge curves.
- Python.parameters.estimation.cccv_inhouse_pouch_cell.discharge = dataset.subslice(states_per_cycle - 1, None, states_per_cycle)

    CC discharge curves.

7.25.1 Detailed Description

Parameter file for the estimation of battery model parameters from the cycling data provided by BASF for one of their in-house pouch cells.

7.26 parameters/estimation/cccv_samsung.py File Reference

Parameter file for the estimation of battery model parameters from the cycling data provided by BASF for a commercial Samsung 18650 cell.

Namespaces

- Python.parameters.estimation.cccv_samsung

Variables

- Python.parameters.estimation.cccv_samsung.E_0_g = np.array([0.35973, 0.17454, 0.12454, 0.081957])

  Fitted plateau voltages of a graphite anode.
- Python.parameters.estimation.cccv_samsung.γUeminus1_g = np.array([-0.33144, 8.9434e-3, 7.2404e-2, 6.↩ 7789e-2])

  Fitted plateau inverse widths of a graphite anode.
- Python.parameters.estimation.cccv_samsung.a_g = a_fit(γUeminus1_g)

  Fitted plateau inverse widths (transformed) of a graphite anode.
- Python.parameters.estimation.cccv_samsung.Δx_g = np.array([8.041e-2, 0.23299, 0.29691, 0.39381])

  Fitted plateau SOC fractions of a graphite anode.
- list Python.parameters.estimation.cccv_samsung.anode = [p[i] for i in range(4) for p in [E_0_g, a_g, Δx_g]]

  Fit parameters of a graphite anode OCV curve.
- int Python.parameters.estimation.cccv_samsung.cathode_phases = 6

  Number of plateaus that are to be fitted to the OCV curve.
- int Python.parameters.estimation.cccv_samsung.smoothing_factor = 1e-7

  Lever for adjusting the smoothing spline for the OCV curves.
- Python.parameters.estimation.cccv_samsung.complete_dataset

  The experiment dataset, stored as a Cycling_Information object.
- int Python.parameters.estimation.cccv_samsung.states_per_cycle = 4

  Number of dataset segments per "cycle" for plotting.
- Python.parameters.estimation.cccv_samsung.indices
- Python.parameters.estimation.cccv_samsung.dataset = complete_dataset

  Subset of interest of the complete experiment dataset.
- int Python.parameters.estimation.cccv_samsung.max_number_of_clusters = 12

  Maximum number of clusters for the automated labelling.
- Python.parameters.estimation.cccv_samsung.charge = dataset.subslice(0, None, states_per_cycle)

  CC charge curves.
- Python.parameters.estimation.cccv_samsung.cv = dataset.subslice(1, None, states_per_cycle // 2)

  CV segments between charge and discharge curves.
- Python.parameters.estimation.cccv_samsung.discharge = dataset.subslice(states_per_cycle - 1, None, states_per_cycle)

  CC discharge curves.

7.26.1 Detailed Description

Parameter file for the estimation of battery model parameters from the cycling data provided by BASF for a commercial Samsung 18650 cell.

7.27 parameters/estimation/discharge_thick_electrodes.py File Reference

Parameter file for the estimation of battery model parameters from the discharge data taken from the "Thick electrodes" paper (Latz et al.).

Namespaces

- Python.parameters.estimation.discharge_thick_electrodes

Functions

- def Python.parameters.estimation.discharge_thick_electrodes.simulator (trial_parameters)

  A simulator compatible with EP-BOLFI.
- def Python.parameters.estimation.discharge_thick_electrodes.get_features (dataset)

  Defines the features.
- def Python.parameters.estimation.discharge_thick_electrodes.name_of_dis_charge_features (index)

  Gives the descriptions of the features by index.
- def Python.parameters.estimation.discharge_thick_electrodes.feature_visualizer (args)

  May be used for plotting purposes.

Variables

- Python.parameters.estimation.discharge_thick_electrodes.l = np.log(10)

  Use the natural logarithm of 10 for scalings in orders of magnitude.
- dictionary Python.parameters.estimation.discharge_thick_electrodes.free_parameters

  Parameters that are to be estimated.
- dictionary Python.parameters.estimation.discharge_thick_electrodes.transform_parameters

  Transformations/Scalings of the parameter ranges.
- dictionary Python.parameters.estimation.discharge_thick_electrodes.free_parameters_boundaries

  Bounds in which a parameter set is searched for.
- list Python.parameters.estimation.discharge_thick_electrodes.E_0 = [3.9074103413415218, 3.7586234939671175, 3.7517405949585427]
- list Python.parameters.estimation.discharge_thick_electrodes.a = [-0.11927628084606558, -1.8526410797182222, -0.4921205974305653]
- list Python.parameters.estimation.discharge_thick_electrodes.Δx = [0.6837809843229503, 0.021811690524785213, 0.29440732515226453]
- list Python.parameters.estimation.discharge_thick_electrodes.cathode = [par[i] for i in range(len(E_0)) for par in [E_0, a, Δx]]

  OCV fit parameters for the cathode.
- Python.parameters.estimation.discharge_thick_electrodes.complete_dataset

  The experimental data to be used for inference.
- float Python.parameters.estimation.discharge_thick_electrodes.current = 0.00231

  The current that is applied during the experiment.
- list Python.parameters.estimation.discharge_thick_electrodes.input = ["Discharge at " + str(current) + " A for 360 s",]

  The experiment as a simulation input for use with simulation_setup.
- Python.parameters.estimation.discharge_thick_electrodes.capacities = np.array(complete_dataset[0])

  The SOC of the cell represented as total discharged capacity (in Ah).
- Python.parameters.estimation.discharge_thick_electrodes.voltages = np.array(complete_dataset[1])

  The voltages at those SOCs.
- int Python.parameters.estimation.discharge_thick_electrodes.starting_SOC = 1 - OCV_fit_function(voltages[0], ∗cathode)

  The initial SOC of the cathode as guessed by its OCV fit function.
- Python.parameters.estimation.discharge_thick_electrodes.voltages_without_OCV

  The isolated overpotential of the experiment.
- Python.parameters.estimation.discharge_thick_electrodes.smoothed_voltages = smooth_fit(capacities, voltages)

  Limit the detrimental effect of random fluctuations by smoothing.
- list Python.parameters.estimation.discharge_thick_electrodes.indices = [int(i) for i in len(capacities) ∗ np.array([0.0, 0.5, 0.9])]

  Chooses the points in the discharge curve used for comparison.

- list Python.parameters.estimation.discharge_thick_electrodes.plateau_capacities = [capacities[i] for i in indices]

  The SOC points for comparison (in Ah).
- list Python.parameters.estimation.discharge_thick_electrodes.plateau_voltages = [smoothed_voltages(capacities[i]) for i in indices]

  The voltages for comparison.
- list Python.parameters.estimation.discharge_thick_electrodes.experimental_datasets

  Packs the comparison points for EP-BOLFI.
- Python.parameters.estimation.discharge_thick_electrodes.fixed_parameters = parameters.copy()

  The free parameters will have been subtracted from this deep copy.
- float Python.parameters.estimation.discharge_thick_electrodes.t_eval = 0.9 $*$ np.array(capacities) $*$ 3600 / current

  Evaluation timepoints for the simulator.
- list Python.parameters.estimation.discharge_thick_electrodes.simulators = [simulator]

  Pack the one simulator for EP-BOLFI.
- list Python.parameters.estimation.discharge_thick_electrodes.feature_extractors = [get_features]

  Pack the defined features for EP-BOLFI.
- list Python.parameters.estimation.discharge_thick_electrodes.display_current_feature = [name_of_dis_↩ charge_features]

  Pack the feature names for EP-BOLFI.
- Python.parameters.estimation.discharge_thick_electrodes.covariance = None

  (Don't) choose the initial covariance matrix.
- int Python.parameters.estimation.discharge_thick_electrodes.bolfi_initial_evidence = 17

  Number of random evidence samples taken before BO, per feature.
- int Python.parameters.estimation.discharge_thick_electrodes.bolfi_total_evidence = 51

  Number of initial and BO samples, per feature.
- int Python.parameters.estimation.discharge_thick_electrodes.bolfi_posterior_samples = 34

  Number of random samples from the posterior distribution.
- int Python.parameters.estimation.discharge_thick_electrodes.ep_iterations = 2

  Number of EP iterations, i.e.
- float Python.parameters.estimation.discharge_thick_electrodes.ep_dampener = 0.5

  Think of this as the dampener in Newton iterations.
- int Python.parameters.estimation.discharge_thick_electrodes.ep_dampener_reduction_steps = 1

  Number of steps after which the dampening is reduced to 0.
- bool Python.parameters.estimation.discharge_thick_electrodes.show_trials = True

  Show the log of tried parameters live.
- int Python.parameters.estimation.discharge_thick_electrodes.seed = 42

  Seed for consistent pseudo-randomness.

7.27.1 Detailed Description

Parameter file for the estimation of battery model parameters from the discharge data taken from the ”Thick electrodes” paper (Latz et al.).

Example result:

inferred parameters: { 'Negative particle radius [m]': 1.1156208227150757e-05, 'Positive particle radius [m]': 5.0336931882823365e-06, 'Positive electrode reaction rate': 6.8491853329305844e-06, 'Initial concentration in positive electrode [mol.m-3]': 4245.305679321547 } covariance of inferred parameters: [[2.07523519e-03 7.↩ 98576741e-04 2.59706419e-03 2.96127926e+00] [7.98576741e-04 4.31707044e-03 5.77436587e-03 3.24502225e+00] [2.59706419e-03 5.77436587e-03 9.09910580e-02 1.97109186e+01] [2.96127926e+00 3.24502225e+00 1.↩ 97109186e+01 2.20409355e+04]] correlation of inferred parameters: [[1. 0.2668016 0.18899472 0.43785538] [0.2668016 1. 0.29134688 0.33266576] [0.18899472 0.29134688 1. 0.44014168] [0.43785538 0.33266576 0.44014168 1.]]

resulting approximate error bounds: Negative particle radius m Positive particle radius m Positive electrode reaction rate (3.746549315336851e-06, 1.2521212394774964e-05) Initial concentration in positive electrode mol.↩
m-3

7.28 parameters/estimation/gitt_timo.py File Reference

Parameter file for the estimation of battery model parameters from the GITT data provided by Timo Danner from a TT-Basytec device.

Namespaces

- Python.parameters.estimation.gitt_timo

Functions

- def Python.parameters.estimation.gitt_timo.current_function (t)

  The current that is applied during the GITT experiment.
- def Python.parameters.estimation.gitt_timo.simulator (trial_parameters)

  A simulator compatible with EP-BOLFI.
- def Python.parameters.estimation.gitt_timo.get_list_of_dataset (dataset)

  Defines the features.
- def Python.parameters.estimation.gitt_timo.name_of_gitt_features (index)

  Gives the descriptions of the features by index.
- def Python.parameters.estimation.gitt_timo.feature_visualizer (args)

  May be used for plotting purposes.

Variables

- Python.parameters.estimation.gitt_timo.l = np.log(10)

  Use the natural logarithm of 10 for scalings in orders of magnitude.
- dictionary Python.parameters.estimation.gitt_timo.free_parameters

  Parameters that are to be estimated.
- dictionary Python.parameters.estimation.gitt_timo.transform_parameters

  Transformations/Scalings of the parameter ranges.
- dictionary Python.parameters.estimation.gitt_timo.free_parameters_boundaries

  Bounds in which a parameter set is searched for.
- list Python.parameters.estimation.gitt_timo.E_0

  Fitted plateau voltages of the cathode.
- list Python.parameters.estimation.gitt_timo.a

  Fitted plateau inverse widths of the cathode.
- list Python.parameters.estimation.gitt_timo.Δx

  Fitted plateau inverse widths (transformed) of the cathode.
- list Python.parameters.estimation.gitt_timo.cathode = [par[i] for i in range(len(E_0)) for par in [E_0, a, Δx]]

  OCV fit parameters for the cathode.
- Python.parameters.estimation.gitt_timo.complete_dataset

The experimental data to be used for inference.

- Python.parameters.estimation.gitt_timo.starting_OCV = complete_dataset.voltages[6][-1]

  Define the starting OCV before each used pulse.
- Python.parameters.estimation.gitt_timo.dataset = complete_dataset.subslice(7, 10)

  Define the pulses that are to be used.
- list Python.parameters.estimation.gitt_timo.input = [ ]

  The current as a pybamm.Simulation input.
- int Python.parameters.estimation.gitt_timo.starting_SOC = 1 - OCV_fit_function(starting_OCV, ∗cathode)

  The initial SOC of the cathode as guessed by its OCV fit function.
- list Python.parameters.estimation.gitt_timo.OCV_fit = [par[i] for i in range(len(E_0)) for par in [E_0, a, Δx]]

  Fit parameters of the cathode OCV curve.
- Python.parameters.estimation.gitt_timo.voltages_without_OCV

  The overpotential curve during the GITT experiment.
- tuple Python.parameters.estimation.gitt_timo.experiment

  The representation of the measurement that gets plotted.
- Python.parameters.estimation.gitt_timo.complete_OCV = np.array(complete_dataset.asymptotic_↩ voltages[3:-1:2])

  Complete measured OCV curve.
- list Python.parameters.estimation.gitt_timo.complete_SOC

  The SOCs for which that OCV was measured.
- list Python.parameters.estimation.gitt_timo.OCV = [starting_OCV] + dataset.asymptotic_voltages[1::2]

  The OCV voltages before each charge pulse.
- list Python.parameters.estimation.gitt_timo.ΔE_s = [U1 - U0 for U0, U1 in zip(OCV[:-1], OCV[1:])]

  The OCV rises between charge pulses.
- Python.parameters.estimation.gitt_timo.ΔE_t = dataset.exp_U_decays[::2][1]

  The transient voltage rises during charge pulses.
- float Python.parameters.estimation.gitt_timo.τ_r = 1.0 / np.array(dataset.exp_U_decays[2])

  The exponential decay rates during each segment.
- Python.parameters.estimation.gitt_timo.IR = dataset.ir_steps

  The IR steps between segments, i.e.
- list Python.parameters.estimation.gitt_timo.experimental_datasets = [[$*\Delta E_t$, $*\Delta E_s$, $*\tau_r$]]

  Packs the comparison points for EP-BOLFI.
- Python.parameters.estimation.gitt_timo.fixed_parameters = parameters.copy()

  The free parameters will have been subtracted from this deep copy.
- list Python.parameters.estimation.gitt_timo.simulators = [simulator]

  Pack the one simulator for EP-BOLFI.
- list Python.parameters.estimation.gitt_timo.feature_extractors = [get_list_of_dataset]

  Pack the defined features for EP-BOLFI.
- dictionary Python.parameters.estimation.gitt_timo.feature_by_index

  Gives the descriptions of the features by index.
- list Python.parameters.estimation.gitt_timo.display_current_feature = [name_of_gitt_features]

  Pack the feature names for EP-BOLFI.
- Python.parameters.estimation.gitt_timo.covariance = None

  (Don't) choose the initial covariance matrix.
- int Python.parameters.estimation.gitt_timo.bolfi_initial_evidence = 17

  Number of random evidence samples taken before BO, per feature.
- int Python.parameters.estimation.gitt_timo.bolfi_total_evidence = 51

  Number of initial and BO samples, per feature.
- int Python.parameters.estimation.gitt_timo.bolfi_posterior_samples = 34

  Number of random samples from the posterior distribution.

- int Python.parameters.estimation.gitt_timo.ep_iterations = 2

  Number of EP iterations, i.e.
- float Python.parameters.estimation.gitt_timo.ep_dampener = 0.5

  Think of this as the dampener in Newton iterations.
- int Python.parameters.estimation.gitt_timo.ep_dampener_reduction_steps = 1

  Number of steps after which the dampening is reduced to 0.
- bool Python.parameters.estimation.gitt_timo.show_trials = True

  Show the log of tried parameters live.
- int Python.parameters.estimation.gitt_timo.seed = 42

  Seed for consistent pseudo-randomness.

7.28.1 Detailed Description

Parameter file for the estimation of battery model parameters from the GITT data provided by Timo Danner from a TT-Basytec device.

Example result:

Sampling the pseudo-posterior... 4 chains of 34 iterations acquired. Effective sample size and Rhat for each parameter: 0 24.473896557536154 1.1613461103017668 1 23.156293949375286 1.1794510323899081 2 34.25761431270452 1.0657349792139257 3 26.84535817769179 1.1483181622131176 4 22.018989257242797 1.1569796195612065 inferred parameters: { 'Electrolyte diffusivity [m2.s-1]': 2.7138690398578775e-10, 'Positive electrode surface area density [m-1]': 283768.33176267147, 'Positive electrode diffusivity [m2.s-1]': 1.3650324331359737e-15, 'Positive particle radius [m]': 5.79066606133241e-06, 'Electrolyte conductivity [S.m-1]': 1.59178976825668 } covariance of inferred parameters: [[1.15669800e-03 -2.03518200e+02 -8.23631698e-04 -6.91489781e-04 1.36791850e-03] [-2.03518200e+02 6.45934019e+07 1.18010377e+02 1.05247802e+02 -1.90930581e+02] [-8.23631698e-04 1.18010377e+02 1.42219984e-03 1.28107433e-03 -1.48937621e-03] [-6.91489781e-04 1.05247802e+02 1.28107433e-03 1.18293190e-03 -1.35351132e-03] [1.36791850e-03 -1.90930581e+02 -1.48937621e-03 -1.35351132e-03 2.17851726e-03]] correlation of inferred parameters: [[1. -0.74455911 -0.64215926 -0.59114756 0.86172706] [-0.74455911 1. 0.38935505 0.38074927 -0.50898023] [-0.64215926 0.38935505 1. 0.98767518 -0.84614267] [-0.59114756 0.38074927 0.98767518 1. -0.84314354] [0.86172706 -0.50898023 -0.84614267 -0.84314354 1.]] resulting approximate error bounds: Electrolyte diffusivity m2.s-1 Positive electrode surface area density m-1 Positive electrode diffusivity m2.s-1 Positive particle radius m Electrolyte conductivity S.m-1 posterior without prior: inferred parameters: { 'Electrolyte diffusivity [m2.s-1]': 2.7334740529796257e-10, 'Positive electrode surface area density [m-1]': 282495.89898455143, 'Positive electrode diffusivity [m2.s-1]': 1.3521658694834716e-15, 'Positive particle radius [m]': 5.741420628930796e-06, 'Electrolyte conductivity [S.m-1]': 1.6099175313642218 } covariance of inferred parameters: [[1.18742433e-03 -2.11039031e+02 -8.49266482e-04 -7.14424375e-04 1.40490939e-03] [-2.11039031e+02 6.66385488e+07 1.23532033e+02 1.10178189e+02 -1.99206008e+02] [-8.49266482e-04 1.23532033e+02 1.44825161e-03 1.30452871e-03 -1.52417469e-03] [-7.14424375e-04 1.10178189e+02 1.30452871e-03 1.20405897e-03 -1.38476039e-03] [1.40490939e-03 -1.99206008e+02 -1.52417469e-03 -1.38476039e-03 2.22667629e-03]] correlation of inferred parameters: [[1. -0.75023452 -0.64761814 -0.59748831 0.86400608] [-0.75023452 1. 0.39764435 0.38896362 -0.51714357] [-0.64761814 0.39764435 1. 0.98788809 -0.84875861] [-0.59748831 0.38896362 0.98788809 1. -0.84571145] [0.86400608 -0.51714357 -0.84875861 -0.84571145 1.]] resulting approximate error bounds: Electrolyte diffusivity m2.s-1 Positive electrode surface area density m-1 Positive electrode diffusivity m2.s-1 Positive particle radius m Electrolyte conductivity S.m-1 itemized covariances per feature: ΔE_s, i.e. OCV rise: covariance: [[1.38436514e-01 1.73191343e+04 -1.51841735e-01 -4.35198426e-02 -1.88236681e-02] [1.73191343e+04 2.77835003e+09 -1.96512738e+04 -5.06336124e+03 -4.14616170e+03] [-1.51841735e-01 -1.96512738e+04 1.69427277e-01 4.97964011e-02 1.96936884e-02] [-4.35198426e-02 -5.06336124e+03 4.97964011e-02 1.71747860e-02 8.06716913e-05] [-1.88236681e-02 -4.14616170e+03 1.96936884e-02 8.06716913e-05 1.66814767e-02]] correlation: [[1. -0.75023452 -0.64761814 -0.59748831 0.86400608] [-0.75023452 1. 0.39764435 0.38896362 -0.51714357] [-0.64761814 0.39764435 1. 0.98788809 -0.84875861] [-0.59748831 0.38896362 0.98788809 1. -0.84571145] [0.86400608 -0.51714357 -0.84875861 -0.84571145 1.]] guess: Electrolyte diffusivity [m2.s-1] 3.7252762546832316e-10 Positive electrode surface area density [m-1] 314596.0662560463 Positive electrode diffusivity [m2.s-1] 9.48045172086094e-16 Positive particle radius [m] 5.114262861842014e-06 Electrolyte conductivity [S.m-1] 1.5973046004562264

Long-term charge relaxation: covariance: [[1.98821179e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00] [0.00000000e+00 1.50000000e+10 0.00000000e+00 0.00000000e+00 0.00000000e+00] [0.↩00000000e+00 0.00000000e+00 1.98821179e+00 0.00000000e+00 0.00000000e+00] [0.00000000e+00 0.00000000e+00 0.00000000e+00 1.98821179e+00 0.00000000e+00] [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.↩00000000e+00 1.98821179e+00]] correlation: [[1. -0.75023452 -0.64761814 -0.59748831 0.86400608] [-0.75023452 1. 0.39764435 0.38896362 -0.51714357] [-0.64761814 0.39764435 1. 0.98788809 -0.84875861] [-0.59748831 0.↩38896362 0.98788809 1. -0.84571145] [0.86400608 -0.51714357 -0.84875861 -0.84571145 1.]] guess: Electrolyte diffusivity [m2.s-1] 2.7200000000000015e-10 Positive electrode surface area density [m-1] 300000.0 Positive electrode diffusivity [m2.s-1] 6.000000000000047e-15 Positive particle radius [m] 5.750000000000012e-06 Electrolyte conductivity [S.m-1] 0.98 Short-term charge relaxation: covariance: [[2.09329509e+00 5.60509902e+04 1.↩84835000e+00 1.97535743e+00 1.43716608e+00] [5.60509902e+04 1.79980958e+09 5.06245182e+04 5.40293780e+04 3.83262875e+04] [1.84835000e+00 5.06245182e+04 1.64262853e+00 1.75437903e+00 1.26525112e+00] [1.↩97535743e+00 5.40293780e+04 1.75437903e+00 1.87391939e+00 1.35252480e+00] [1.43716608e+00 3.83262875e+04 1.26525112e+00 1.35252480e+00 9.88794656e-01]] correlation: [[1. -0.75023452 -0.64761814 -0.59748831 0.↩86400608] [-0.75023452 1. 0.39764435 0.38896362 -0.51714357] [-0.64761814 0.39764435 1. 0.98788809 -0.↩84875861] [-0.59748831 0.38896362 0.98788809 1. -0.84571145] [0.86400608 -0.51714357 -0.84875861 -0.84571145 1.]] guess: Electrolyte diffusivity [m2.s-1] 7.009209202979146e-11 Positive electrode surface area density [m-1] 239346.21948051453 Positive electrode diffusivity [m2.s-1] 4.0649258238183613e-16 Positive particle radius [m] 1.5851252207050106e-06 Electrolyte conductivity [S.m-1] 0.6247854489969904 Long-term pause relaxation: covariance: [[8.80361514e-02 -2.54215147e+04 4.94785725e-03 9.23346323e-03 -1.95893430e-02] [-2.54215147e+04 8.32199155e+09 -1.59184090e+04 -1.71683557e+04 2.40405157e+04] [4.94785725e-03 -1.↩59184090e+04 8.66454882e-01 8.75158776e-01 -9.71813007e-01] [9.23346323e-03 -1.71683557e+04 8.75158776e-01 8.87130631e-01 -9.84740264e-01] [-1.95893430e-02 2.40405157e+04 -9.71813007e-01 -9.84740264e-01 1.↩12714073e+00]] correlation: [[1. -0.75023452 -0.64761814 -0.59748831 0.86400608] [-0.75023452 1. 0.39764435 0.38896362 -0.51714357] [-0.64761814 0.39764435 1. 0.98788809 -0.84875861] [-0.59748831 0.38896362 0.98788809 1. -0.84571145] [0.86400608 -0.51714357 -0.84875861 -0.84571145 1.]] guess: Electrolyte diffusivity [m2.s-1] 3.↩1103027068816663e-10 Positive electrode surface area density [m-1] 275164.06605134904 Positive electrode diffusivity [m2.s-1] 2.3067780266178093e-15 Positive particle radius [m] 8.08418863444697e-06 Electrolyte conductivity [S.m-1] 0.9000953715137837 Short-term pause relaxation: covariance: [[2.86732774e-03 -5.00981424e+02 -2.↩31003035e-03 -2.00590502e-03 3.61314495e-03] [-5.00981424e+02 1.38221148e+08 3.63229920e+02 3.25199171e+02 -5.35300880e+02] [-2.31003035e-03 3.63229920e+02 4.14424171e-03 3.80387637e-03 -4.30995184e-03] [-2.↩00590502e-03 3.25199171e+02 3.80387637e-03 3.54830170e-03 -3.97567834e-03] [3.61314495e-03 -5.35300880e+02 -4.30995184e-03 -3.97567834e-03 5.94026835e-03]] correlation: [[1. -0.75023452 -0.64761814 -0.59748831 0.↩86400608] [-0.75023452 1. 0.39764435 0.38896362 -0.51714357] [-0.64761814 0.39764435 1. 0.98788809 -0.↩84875861] [-0.59748831 0.38896362 0.98788809 1. -0.84571145] [0.86400608 -0.51714357 -0.84875861 -0.84571145 1.]] guess: Electrolyte diffusivity [m2.s-1] 2.699470283649517e-10 Positive electrode surface area density [m-1] 284662.135117352 Positive electrode diffusivity [m2.s-1] 1.3756597290534754e-15 Positive particle radius [m] 5.832287928370287e-06 Electrolyte conductivity [S.m-1] 1.576331578091006

7.29 parameters/models/basf_samsung.py File Reference

Parameter file for the Samsung battery that was measured by BASF.

Namespaces

- Python.parameters.models.basf_samsung

Variables

- float Python.parameters.models.basf_samsung.π = 3.141592653589793

  Pi is needed because the cell is cylindrical.
- float Python.parameters.models.basf_samsung.α_{nn} = 0.5

  Assumptions made without justification by measurements.
- float Python.parameters.models.basf_samsung.α_{pn} = 0.5

Cathodic symmetry factor at the anode.

- float Python.parameters.models.basf_samsung.α_{np} = 0.5

  Anodic symmetry factor at the cathode.
- float Python.parameters.models.basf_samsung.α_{pp} = 0.5

  Cathodic symmetry factor at the cathode.

7.29.1 Detailed Description

Parameter file for the Samsung battery that was measured by BASF.

Measurement files: E_RE_93cyc.094, PCA_SEM_17Y36944_SEM.pptx, PCA_TEM_17Y38544_TEM.pptx, PCA_↩
XRD_17Y36945_XRD.docx.

7.30 parameters/models/basytec.py File Reference

Parameter file for the battery that was measured in the "KN19097_001_02" Basytec GITT measurement.

Namespaces

- Python.parameters.models.basytec

Functions

- def Python.parameters.models.basytec.D_int_positive (c, T, T_ref, E_D_s_p, R)

  Define the diffusivity function in a form PyBaMM can use.

Variables

- float Python.parameters.models.basytec.α_{nn} = 0.5

  Anodic symmetry factor at the anode.
- float Python.parameters.models.basytec.α_{pn} = 0.5

  Cathodic symmetry factor at the anode.
- float Python.parameters.models.basytec.α_{np} = 0.5

  Anodic symmetry factor at the cathode.
- float Python.parameters.models.basytec.α_{pp} = 0.5

  Cathodic symmetry factor at the cathode.
- dictionary Python.parameters.models.basytec.parameters

  Parameter dictionary.
- dictionary Python.parameters.models.basytec.c_n_max = parameters["Maximum concentration in negative electrode [mol.m-3]"]

  Maximum charge concentration in the anode active material.
- dictionary Python.parameters.models.basytec.c_p_max = parameters["Maximum concentration in positive electrode [mol.m-3]"]

  Maximum charge concentration in the cathode active material.

7.30.1 Detailed Description

Parameter file for the battery that was measured in the "KN19097_001_02" Basytec GITT measurement.

7.31 parameters/models/diffusivity_curves.py File Reference

This file contains several diffusivity curves.

Namespaces

- Python.parameters.models.diffusivity_curves

Functions

- def Python.parameters.models.diffusivity_curves.D_int_graphite (SOC)
  Standard graphite diffusivity.
- def Python.parameters.models.diffusivity_curves.D_int_NMC_111 (SOC)
  NMC diffusivity from "Thick electrodes".
- def Python.parameters.models.diffusivity_curves.D_int_NMC_622 (SOC)
  NMC diffusivity as used for the half-cell with GITT from Timo.

7.31.1 Detailed Description

This file contains several diffusivity curves.

7.32 parameters/models/ocv_curves.py File Reference

This file contains several OCV curves.

Namespaces

- Python.parameters.models.ocv_curves

Functions

- def Python.parameters.models.ocv_curves.OCV_graphite (SOC)
  OCV curve of graphite as used in "Thick electrodes".
- def Python.parameters.models.ocv_curves.d_OCV_graphite_d_SOC (SOC)
  Derivative of the prior OCV curve.
- def Python.parameters.models.ocv_curves.OCV_NMC_111 (SOC)
  OCV curve of NMC as used in "Thick electrodes".
- def Python.parameters.models.ocv_curves.d_OCV_NMC_111_d_SOC (SOC)
  Derivative of the prior OCV curve.
- def Python.parameters.models.ocv_curves.OCV_NMC_622 (SOC)
  OCV curve of NMC as used for the half-cell with GITT from Timo.
- def Python.parameters.models.ocv_curves.d_OCV_NMC_622_d_SOC (SOC)
  Derivative of the prior OCV curve.
- def Python.parameters.models.ocv_curves.OCV_NMC_thick_electrodes (SOC)
  OCV curve fitted to the GITT measurement in "Thick electrodes".
- def Python.parameters.models.ocv_curves.d_OCV_NMC_thick_electrodes_d_SOC (SOC)
  Derivative of the prior OCV curve.

- def Python.parameters.models.ocv_curves.OCV_graphite_precise (SOC)

  OCV curve for graphite as seen in "A Parametric OCV Model".
- def Python.parameters.models.ocv_curves.d_OCV_graphite_precise_d_SOC (SOC)

  Derivative of the prior OCV curve.
- def Python.parameters.models.ocv_curves.OCV_NMC_442 (SOC)

  OCV curve of NMC fitted to the CC-CV measurements of a Samsung cell.
- def Python.parameters.models.ocv_curves.OCV_NMC_622_precise (SOC)

  OCV curve of NMC fitted to the GITT measurement from Timo.
- def Python.parameters.models.ocv_curves.d_OCV_NMC_622_precise_d_SOC (SOC)

  Derivative of the prior OCV curve.
- def Python.parameters.models.ocv_curves.OCV_BASF (SOC)

  OCV curve of NMC fitted to the in-house CC-CV measurements from BASF.

7.32.1 Detailed Description

This file contains several OCV curves.

7.33 parameters/models/thick_electrodes.py File Reference

Parameter file for the thin full battery cell in "Thick electrodes for Li-ion batteries".

Namespaces

- Python.parameters.models.thick_electrodes

Variables

- float Python.parameters.models.thick_electrodes.α_{nn} = 0.5

  Anodic symmetry factor at the anode.
- float Python.parameters.models.thick_electrodes.α_{pn} = 0.5

  Cathodic symmetry factor at the anode.
- float Python.parameters.models.thick_electrodes.α_{np} = 0.5

  Anodic symmetry factor at the cathode.
- float Python.parameters.models.thick_electrodes.α_{pp} = 0.5

  Cathodic symmetry factor at the cathode.
- dictionary Python.parameters.models.thick_electrodes.parameters

  Parameter dictionary.
- dictionary Python.parameters.models.thick_electrodes.c_n_max = parameters["Maximum concentration in negative electrode [mol.m-3]"]

  Maximum charge concentration in the anode active material.
- dictionary Python.parameters.models.thick_electrodes.c_p_max = parameters["Maximum concentration in positive electrode [mol.m-3]"]

  Maximum charge concentration in the cathode active material.

7.33.1 Detailed Description

Parameter file for the thin full battery cell in "Thick electrodes for Li-ion batteries".

7.34 particle_identification/particles.py File Reference

Classes

- class Python.particle_identification.particles.ParticlesConfig

  Configurations.
- class Python.particle_identification.particles.ParticlesDataset

  Provides access to the toy dataset.
- class Python.particle_identification.particles.InferenceConfig

Namespaces

- Python.particle_identification.particles
- particle_identification.particles

  Mask R-CNN Train on the NMC particle dataset for segmentation.

Functions

- def Python.particle_identification.particles.train (model)

  Train the model.

Variables

- Python.particle_identification.particles.ROOT_DIR = os.path.abspath("../../")

  Root directory of the project.
- Python.particle_identification.particles.COCO_WEIGHTS_PATH = os.path.join(ROOT_DIR, "mask_↩ rcnn_coco.h5")

  Path to trained weights file (coco).
- Python.particle_identification.particles.BALLOON_WEIGHTS_PATH = os.path.join(ROOT_DIR, "mask↩ _rcnn_balloon.h5")

  Path to trained weights file (balloon).
- Python.particle_identification.particles.DEFAULT_LOGS_DIR = os.path.join(ROOT_DIR, "logs")

  Directory to save logs and model checkpoints, if not provided through the command line argument –logs.
- Python.particle_identification.particles.parser
- Python.particle_identification.particles.metavar
- Python.particle_identification.particles.help
- Python.particle_identification.particles.required
- Python.particle_identification.particles.default
- Python.particle_identification.particles.args = parser.parse_args()
- Python.particle_identification.particles.config = ParticlesConfig()
- Python.particle_identification.particles.model
- Python.particle_identification.particles.weights_path = COCO_WEIGHTS_PATH
- Python.particle_identification.particles.by_name
- Python.particle_identification.particles.True
- Python.particle_identification.particles.exclude
- Python.particle_identification.particles.image_path
- Python.particle_identification.particles.video_path

7.35 utility/calculate_characteristics.py File Reference

Namespaces

- Python.utility.calculate_characteristics
- utility.characteristics

    This file calculates and displays battery characteristics and timescales from a 1+1D-DFN parameter set as it used in PyBaMM.

Functions

- def Python.utility.calculate_characteristics.print (parameters_path)

    Prints the characteristics for a 1+1D model.

7.36 utility/fitting_functions.py File Reference

Namespaces

- Python.utility.fitting_functions
- utility.fitting_functions

    Various helper and fitting functions for processing measurement curves.

Functions

- def Python.utility.fitting_functions.smooth_fit (x, y, order=5, splits=None, smoothing_factor=2e-3, display=False)

    Calculates a smoothed spline with derivatives.
- def Python.utility.fitting_functions.fit_exponential_decay (timepoints, voltages, recursive_depth=1, threshold=0.75e-4)

    Extracts a set amount of exponential decay curves.
- def Python.utility.fitting_functions.laplace_transform (x, y, s)

    Performs a basic laplace transformation.
- def Python.utility.fitting_functions.a_fit (γUeminus1)

    Calculates the conversion from "A parametric OCV model".
- def Python.utility.fitting_functions.OCV_fit_function (E_OCV, args, z=1.0, T=298.15, individual=False)

    The OCV model from "A parametric OCV model".
- def Python.utility.fitting_functions.d_dE_OCV_fit_function (E_OCV, args, z=1.0, T=298.15)

    The derivative of fitting_functions.OCV_fit_function.
- def Python.utility.fitting_functions.d2_dE2_OCV_fit_function (E_OCV, args, z=1.0, T=298.15)

    The 2^n derivative of fitting_functions.OCV_fit_function.
- def Python.utility.fitting_functions.OCV_and_SOC_range_fit_function (E_OCV, args, z=1.0, T=298.15)

    The OCV model from "A parametric OCV model".
- def Python.utility.fitting_functions.inverse_OCV_fit_function (SOC, args, z=1.0, T=298.15, inverted=True)

    The inverse of fitting_functions.OCV_fit_function.
- def Python.utility.fitting_functions.fit_OCV (SOC, OCV, N=4, initial=None, z=1.0, T=298.15, inverted=True)

    Fits data to fitting_functions.OCV_fit_function.
- def Python.utility.fitting_functions.fit_OCV_and_SOC_range (SOC, OCV, N=4, initial=None, z=1.↩ 0, T=298.15, inverted=True)

    Fits data to fitting_functions.OCV_fit_function.
- def Python.utility.fitting_functions.verbose_spline_parameterization (coeffs, knots, order, function_↩ name="OCV", function_args="SOC", verbose=False)

    Gives the monomic representation of a B-spline.

7.37 utility/preprocessing.py File Reference

Namespaces

- Python.utility.preprocessing
- utility.preprocessing

  Contains frequently used workflows in dataset preprocessing.

Functions

- def Python.utility.preprocessing.combine_parameters_to_try (parameters, parameters_to_try_dict)

  Give every combination as full parameter sets.
- def Python.utility.preprocessing.fix_parameters (parameters_to_be_fixed)

  Returns a function which sets some parameters in advance.
- def Python.utility.preprocessing.capacity (parameters)

  Convenience function for calculating the capacity.
- def Python.utility.preprocessing.calculate_SOC (timepoints, currents)

  Transforms applied current over time into SOC.
- def Python.utility.preprocessing.subtract_OCV_curve (voltages, timepoints, currents, parameters, OCV↩_fit, starting_OCV=-1, electrode="cathode")

  Removes the OCV curve from a single cycle.
- def Python.utility.preprocessing.subtract_OCV_curve_from_cycles (dataset, parameters, OCV_fit, starting_OCV=-1, electrode="cathode")

  Removes the OCV curve from a cycling measurement.
- def Python.utility.preprocessing.laplace_transform (x, y, s)

  Performs a basic laplace transformation.
- def Python.utility.preprocessing.OCV_from_CC_CV (charge, cv, discharge, name, phases, eval_points=200, spline_SOC_range=(0.01, 0.99), spline_order=2, spline_smoothing=2e-3, spline_print=False, parameters↩_print=False)

  Tries to extract the OCV curve from CC-CV cycling data.

7.38 utility/read_csv_datasets.py File Reference

Classes

- class Python.utility.read_csv_datasets.Cycling_Information

  Contains basic cycling informations.
- class Python.utility.read_csv_datasets.Static_Information

  Contains additional informations, e.g.

Namespaces

- Python.utility.read_csv_datasets
- utility.read_csv_datasets

  Defines datatypes for further processing and containtes the means to convert commonly encountered measurement files into them.

Functions

- def Python.utility.read_csv_datasets.read_voltages_from_csv (path)

  Read a (dis-)charge curve from a csv file.
- def Python.utility.read_csv_datasets.read_channels_from_measurement_system (path, encoding, number_of_comment_lines, headers, delimiter='\t', decimal='.', type="", segment_column=-1, current_↩ sign_correction={}, correction_column=-1, max_number_of_lines=-1)

  Read the measurements as returned by common instruments.

7.39 utility/visualization.py File Reference

Classes

- class Python.utility.visualization.HandlerColorLineCollection

  Automates multicolored lines in legends.

Namespaces

- Python.utility.visualization
- utility.visualization

  Various helper and plotting functions for common data visualizations.

Functions

- def Python.utility.visualization.update_limits (ax, xmin=float('inf'), xmax=-float('inf'), ymin=float('inf'), ymax=-float('inf'))

  Convenience function for adjusting the view.
- def Python.utility.visualization.set_fontsize (ax, title=12, xaxis=12, yaxis=12, xticks=12, yticks=12, legend=12)

  Convenience function for fontsize changes.
- def Python.utility.visualization.update_legend (ax, additional_handles=[ ], additional_labels=[ ], additional↩ _handler_map={})

  Makes sure that all items remain and all items show up.
- def Python.utility.visualization.make_segments (x, y)

  Create a list of line segments from x and y coordinates.
- def Python.utility.visualization.colorline (x, y, z=None, cmap=plt.get_cmap('viridis'), norm=matplotlib.↩ colors.Normalize(0, 1), linewidth=1, linestyle='-', alpha=1.0)

  Generates a colored line using LineCollection.
- def Python.utility.visualization.impedance_visualization (fig, ax, ω, Z, cmap=plt.get_cmap('tab20b'), ls='-', lw=3, legend_points=4, legend_text="impedance", colorbar_label="Frequency / Hz")
- def Python.utility.visualization.plot_comparison (ax, solutions, experiment, t_eval=None, title="", xlabel="", ylabel="", feature_visualizer=lambda ∗args:[ ], interactive_plot=False, output_variables=None)

  Tool for comparing simulation<->experiment with features.
- def Python.utility.visualization.cc_cv_visualization (fig, ax, dataset, max_number_of_clusters=4, cmap=plt.get_cmap('tab20c'), check_location=[0.1)

  Automatically labels and displays a CC-CV dataset.
- def Python.utility.visualization.plot_OCV_from_CC_CV (ax_ICA_meas, ax_ICA_mean, ax_OCV_meas, ax_OCV_mean, charge, cv, discharge, name, phases, eval_points=200, spline_SOC_range=(0.01, 0.99), spline_order=2, spline_smoothing=2e-3, spline_print=False, parameters_print=False)

  Visualizes the OCV_fitting.OCV_from_CC_CV output.

- def Python.utility.visualization.plot_ICA (ax, SOC, OCV, name, spline_order=2, spline_smoothing=2e-3, sign=1)

  Show the derivative of charge by voltage.
- def Python.utility.visualization.plot_measurement (fig, ax, dataset, title, cmap=plt.get_cmap('tab20c'))

  Plots current and voltage curves in one diagram.
- def Python.utility.visualization.fit_and_plot_OCV (ax, SOC, OCV, name, phases, eval_points=200, spline←
_SOC_range=(0.01, 0.99), spline_order=2, spline_smoothing=2e-3, spline_print=False, parameters_←
print=False, inverted=True)

  Fits an SOC(OCV)-model and an OCV(SOC)-evaluable spline.

7.40 watershed.py File Reference

Namespaces

- Python.watershed

Functions

- def Python.watershed.gaussian_kernel (x, μ, σ, w)

Variables

- Python.watershed.π = np.pi
- Python.watershed.image = skimage.io.imread("../Datensätze/RE_93_cathode_20mu_edited.png")
- Python.watershed.orig_image = skimage.io.imread("../Datensätze/RE_93_cathode_20mu.png")
- Python.watershed.foreground
- Python.watershed.background
- Python.watershed.markers = np.zeros_like(image)
- Python.watershed.elevation_map = sobel(image)
- Python.watershed.segmentation = watershed(elevation_map, markers)
- Python.watershed.fig
- Python.watershed.ax
- Python.watershed.figsize
- int Python.watershed.n = 4
- Python.watershed.prop_cycle = plt.rcParams['axes.prop_cycle']
- Python.watershed.colors = prop_cycle.by_key()['color']
- Python.watershed.pixels = image.flatten()
- Python.watershed.hist
- Python.watershed.clust = KMeans(n_clusters=n)
- Python.watershed.labels = clust.fit_predict(pixels.reshape(-1, 1))
- Python.watershed.centers = clust.cluster_centers_.flatten()
- list Python.watershed.indices = [(labels == i).nonzero()[0] for i in range(n)]
- Python.watershed.ranges
- list Python.watershed.thresholds
- Python.watershed.processed_image = np.zeros_like(image)
- float Python.watershed.last_t = 0.0
- Python.watershed.nrows
- Python.watershed.ncols
- Python.watershed.grayscale = np.linspace(0, 1, 255)[:, np.newaxis]
- Python.watershed.plot_range = np.linspace(r[0], r[1], 100)
- Python.watershed.color
- Python.watershed.alpha