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 ep_bolfi Namespace Reference

Namespaces

- models
- optimization
- utility

5.2 ep_bolfi.EP_BOLFI.EP_BOLFI Namespace Reference

This file contains functions to perform Expectation Propagation on simulator models using BOLFI (Bayesian Optimization).

5.2.1 Detailed Description

This file contains functions to perform Expectation Propagation on simulator models using BOLFI (Bayesian Optimization).

5.3 ep_bolfi.models Namespace Reference

Namespaces

- assess_effective_parameters

  Evaluates a parameter set for, e.g., overpotential and capacity.
- electrolyte

  Contains a PyBaMM-compatible electrolyte model.
- solversetup

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

  Comprehensive list of parameters of every model.

5.4 ep_bolfi.models.assess_effective_parameters Namespace Reference

Evaluates a parameter set for, e.g., overpotential and capacity.

Classes

- class Effective_Parameters

  Calculates, stores and prints effective parameters.

5.4.1 Detailed Description

Evaluates a parameter set for, e.g., overpotential and capacity.

5.5 ep_bolfi.models.electrolyte Namespace Reference

Contains a PyBaMM-compatible electrolyte model.

Classes

- class Electrolyte_internal

  Defining equations for a symmetric Li cell with electrolyte.
- class Electrolyte

  Electrolyte model assuming a symmetric Li-metal cell.

5.5.1 Detailed Description

Contains a PyBaMM-compatible electrolyte model.

5.6 ep_bolfi.models.solversetup Namespace Reference

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

Functions

- def solver_setup (model, parameters, submesh_types, var_pts, spatial_methods, geometry=None, reltol=1e-6, abstol=1e-6, root_tol=1e-3, dt_max=None, free_parameters=[ ], verbose=False, logging_↩ file=None)

  Processes the model and returns a runnable solver.
- def simulation_setup (model, operation_input, parameters, submesh_types, var_pts, spatial_methods, geometry=None, reltol=1e-6, abstol=1e-6, root_tol=1e-3, dt_max=None, free_parameters=[ ], verbose=False, logging_file=None)

  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.

5.6.1 Detailed Description

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

5.6.2 Function Documentation

5.6.2.1 auto_var_pts()

def ep_bolfi.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.

Parameters

x↩ _n	The number of voxels for the electrolyte in the anode.
x↩ _s	The number of voxels for the electrolyte in the separator.

Parameters

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.

5.6.2.2 simulation_setup()

def ep_bolfi.models.solversetup.simulation_setup (
model,
operation_input,
parameters,
submesh_types,
var_pts,
spatial_methods,
geometry = None,
reltol = 1e-6,
abstol = 1e-6,
root_tol = 1e-3,
dt_max = None,
free_parameters = [],
verbose = False,
logging_file = None)

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.

Parameters

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

Parameters

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-3.
dt_max	The maximum timestep size for the Casadi solver in seconds. Default is chosen by PyBaMM.
free_parameters	A list of parameter names that shall be input later. They may be given to the solve() function of the returned Simulation object as a dictionary to the keyword parameter "inputs" with the names as keys and the values of the parameters as values. DO NOT USE GEOMETRICAL PARAMETERS, THEY WILL CRASH THE MESH. Instead, just use this function with a complete set of parameters where the relevant parameters are changed.
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.
logging_file	Optional name of a file to store the logs in.

Returns

A 2-tuple of a pybamm.Simulation.solve call that runs the simulation when called, and the proper callback for the logging file if specified (else None). Please use solve(check_model=False) if it doesn't work properly with redundant model checks in place.

5.6.2.3 solver_setup()

def ep_bolfi.models.solversetup.solver_setup (
model,
parameters,
submesh_types,
var_pts,
spatial_methods,
geometry = None,
reltol = 1e-6,
abstol = 1e-6,
root_tol = 1e-3,
dt_max = None,
free_parameters = [],
verbose = False,
logging_file = None)

Processes the model and returns a runnable solver.

Parameters

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.

Parameters

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 shall use. Default is 1e-6.
abstol	The absolute tolerance that the Casadi solver shall use. Default is 1e-6.
root_tol	The tolerance for rootfinding that the Casadi solver shall use. Default is 1e-3.
dt_max	The maximum timestep size for the Casadi solver in seconds. Default is chosen by PyBaMM.
free_parameters	A list of parameter names that shall be input later. They may be given to the returned lambda function as a dictionary with the names as keys and the values of the parameters as values. DO NOT USE GEOMETRICAL PARAMETERS, THEY WILL CRASH THE MESH. Instead, just use this function with a complete set of parameters where the relevant parameters are changed.
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.
logging_file	Optional name of a file to store the logs in.

Returns

A lambda function that takes a numpy.array of timepoints to evaluate and runs the Casadi solver for those. Optionally takes a dictionary of parameters as specified by ”free_parameters”.

5.6.2.4 spectral_mesh_pts_and_method() def ep_bolfi.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.

Parameters

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.

Parameters

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.

5.7 ep_bolfi.models.standard_parameters Namespace Reference

Comprehensive list of parameters of every model.

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, c_n_max, T_dim)

  Anode exchange current density.
- def i_{sen}_0 (c_{en}, SOC_n_surf, c_n_max, T)

  Non-dimensionalized anode exchange current density.
- def d_c_{en}_i_{sen}_0_dim (c_{en}_dim, SOC_n_surf_dim, c_n_max, T_dim)

  ∂ anode exchange current density / ∂ electrolyte concentration.
- def d_c_{en}_i_{sen}_0 (c_{en}, SOC_n_surf, c_n_max, T)

  The non-dimensionalized version of the prior variable.
- def i_{sep}_0_dim (c_{ep}_dim, SOC_p_surf_dim, c_p_max, T_dim)

  Cathode exchange current density.
- def i_{sep}_0 (c_{ep}, SOC_p_surf, c_p_max, T)

  Non-dimensionalized cathode exchange current density.
- def d_c_{ep}_i_{sep}_0_dim (c_{ep}_dim, SOC_p_surf_dim, c_p_max, T_dim)

  ∂ cathode exchange current density / ∂ electrolyte concentration.
- def d_c_{ep}_i_{sep}_0 (c_{ep}, SOC_p_surf, c_p_max, 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)
- F = Scalar(constants.physical_constants["Faraday constant"][0])
- k_B = constants.physical_constants["Boltzmann constant"][0]
- q_e = constants.physical_constants["electron volt"][0]
- T_ref = Parameter("Reference temperature [K]")
- T_init = Parameter("Initial temperature [K]")
- thermal_voltage = R ∗ T_ref / F
- ΔT = Scalar(1)
- L_n_dim = Parameter("Negative electrode thickness [m]")
- L_s_dim = Parameter("Separator thickness [m]")
- L_p_dim = Parameter("Positive electrode thickness [m]")
- L_dim = L_n_dim + L_s_dim + L_p_dim
- A = Parameter("Current collector perpendicular area [m2]")
- V = Parameter("Cell volume [m3]")
- L_x = L_dim
- L_y = Parameter("Electrode width [m]")
- L_z = Parameter("Electrode height [m]")
- C = Parameter("Typical current [A]")
- U_l = pybamm.Parameter("Lower voltage cut-off [V]")
- U_u = pybamm.Parameter("Upper voltage cut-off [V]")
- c_e_typ = pybamm.Parameter("Typical electrolyte concentration [mol.m-3]")
- c_n = pybamm.Parameter("Maximum concentration in negative electrode [mol.m-3]")
- c_p = pybamm.Parameter("Maximum concentration in positive electrode [mol.m-3]")
- σ_n_dim = pybamm.Parameter("Negative electrode conductivity [S.m-1]")
- σ_p_dim = pybamm.Parameter("Positive electrode conductivity [S.m-1]")
- a_n_dim = Parameter("Negative electrode surface area to volume ratio [m-1]")
- a_p_dim = Parameter("Positive electrode surface area to volume ratio [m-1]")
- R_n = Parameter("Negative particle radius [m]")
- R_p = Parameter("Positive particle radius [m]")
- α_{nn} = Parameter("Negative electrode anodic charge-transfer coefficient")
- α_{pn} = Parameter("Negative electrode cathodic charge-transfer coefficient")
- α_{np} = Parameter("Positive electrode anodic charge-transfer coefficient")
- α_{pp} = Parameter("Positive electrode cathodic charge-transfer coefficient")
- β_{en}_scalar
- β_{es}_scalar = Parameter("Separator Bruggeman coefficient (electrolyte)")
- β_{ep}_scalar
- β_{en} = pybamm.PrimaryBroadcast(β_{en}_scalar, "negative electrode")
- β_{es} = pybamm.PrimaryBroadcast(β_{es}_scalar, "separator")
- β_{ep} = pybamm.PrimaryBroadcast(β_{ep}_scalar, "positive electrode")
- β_{sn}_scalar = Parameter("Negative electrode Bruggeman coefficient (electrode)")
- β_{ss}_scalar = Parameter("Separator Bruggeman coefficient (electrode)")
- β_{sp}_scalar = Parameter("Positive electrode Bruggeman coefficient (electrode)")
- β_e = pybamm.Concatenation(β_{en}, β_{es}, β_{ep})
- ε_n_scalar = Parameter("Negative electrode porosity")
- ε_s_scalar = Parameter("Separator porosity")
- ε_p_scalar = Parameter("Positive electrode porosity")
- ε_n = pybamm.PrimaryBroadcast(ε_n_scalar, "negative electrode")
- ε_s = pybamm.PrimaryBroadcast(ε_s_scalar, "separator")
- ε_p = pybamm.PrimaryBroadcast(ε_p_scalar, "positive electrode")
- ε = pybamm.Concatenation(ε_n, ε_s, ε_p)
- ε^β = ε∗∗β_e
- z_n = Parameter("Negative electrode electrons in reaction")
- z_p = Parameter("Positive electrode electrons in reaction")

- c_e_dim_init = Parameter("Initial concentration in electrolyte [mol.m-3]")
- c_e_init = c_e_dim_init / c_e_typ
- def D_e_typ = D_e_dim(c_e_typ, T_ref)
- def κ_e_typ = κ_e_dim(c_e_typ, T_ref)
- tuple κ_e_hat = (R $*$ T_ref / F) / (C / A $*$ L_dim / κ_e_typ)
- def D_n_typ = D_n_dim(Scalar(0.5), T_ref)
- def D_p_typ = D_p_dim(Scalar(0.5), T_ref)
- def i_{sn}_0_ref = i_{sn}_0_dim(c_e_typ, 0.5 $*$ c_n, c_n, T_ref)
- def i_{sep}_0_ref = i_{sep}_0_dim(c_e_typ, 0.5 $*$ c_p, c_p, T_ref)
- def OCV_n_ref = OCV_n_dim(SOC_n_init(0), T_ref)
- def OCV_p_ref = OCV_p_dim(SOC_p_init(1), T_ref)
- a_n = a_n_dim $*$ R_n
- a_p = a_p_dim $*$ R_p
- tuple Q = (1 - ε_p_scalar) $*$ L_p_dim $*$ c_p $*$ z_p $*$ F $*$ A
- τ^d = F $*$ c_p $*$ L_dim / (C / A)
- int τ_e = L_dim$**$2 / D_e_typ
- int τ_n = $R_n$$**$2 / D_n_typ
- int τ_p = $R_p$$**$2 / D_p_typ
- τ_{rn} = F $*$ c_n / (i_{sn}_0_ref $*$ a_n_dim)
- τ_{rp} = F $*$ c_p / (i_{sep}_0_ref $*$ a_p_dim)
- timescale = τ^d
- int C_e = τ_e / τ^d
- int C_n = τ_n / τ^d
- int C_p = τ_p / τ^d
- C_{rn} = τ_{rn} / τ^d
- C_{rp} = τ_{rp} / τ^d
- γ_e = c_e_typ / c_p
- γ_n = c_n / c_p
- γ_p = c_p / c_p
- L_n = L_n_dim / L_dim
- L_s = L_s_dim / L_dim
- L_p = L_p_dim / L_dim
- L_e
- tuple σ_n = (thermal_voltage / (C / A $*$ L_dim)) $*$ σ_n_dim
- tuple σ_p = (thermal_voltage / (C / A $*$ L_dim)) $*$ σ_p_dim
- I_extern_dim
- I_extern = I_extern_dim / C
- n_electrodes_parallel
- n_cells = Parameter("Number of cells connected in series to make a battery")
- I_typ = C
- A_cc = A
- current_with_time = I_extern
- dimensional_current_with_time = I_extern_dim
- dimensional_current_density_with_time = I_extern_dim / A
- voltage_low_cut = U_l
- voltage_high_cut = U_u
- capacity = Parameter("Nominal cell capacity [A.h]")

5.7.1 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".

5.7.2 Function Documentation

5.7.2.1 d_c$_{en}$_i$_{sen}$_0()

def ep_bolfi.models.standard_parameters.d_c$_{en}$_i$_{sen}$_0 (
c$_{en}$,
SOC$_n$_surf,
c$_n$_max,
T)

The non-dimensionalized version of the prior variable.

5.7.2.2 d_c$_{en}$_i$_{sen}$_0_dim()

def ep_bolfi.models.standard_parameters.d_c$_{en}$_i$_{sen}$_0_dim (
c$_{en}$_dim,
SOC$_n$_surf_dim,
c$_n$_max,
T_dim)

∂ anode exchange current density / ∂ electrolyte concentration.

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

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

The non-dimensionalized version of the prior variable.

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

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

∂ cathode exchange current density / ∂ electrolyte concentration.

5.7.2.5 dOCV$_n$_dim_dSOC$_n$()

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

∂ anode OCV / ∂ anode SOC.

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

Non-dimensionalized ∂ anode OCV / ∂ anode SOC.

5.7.2.7 dOCV$_n$_dT() def ep_bolfi.models.standard_parameters.dOCV$_n$_dT (
SOC$_n$)

Non-dimensionalized ∂ anode OCV / ∂ temperature.

5.7.2.8 dOCV$_n$_dT_dim() def ep_bolfi.models.standard_parameters.dOCV$_n$_dT_dim (
SOC$_n$)

∂ anode OCV / ∂ temperature.

5.7.2.9 dOCV$_n$_dT_dSOC$_n$() def ep_bolfi.models.standard_parameters.dOCV$_n$_dT_dSOC$_n$ (
SOC$_n$)

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

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

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

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

∂ cathode OCV / ∂ cathode SOC.

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

Non-dimensionalized ∂ cathode OCV / ∂ cathode SOC.

5.7.2.13 $\mathbf{dOCV_p\_dT()}$ def ep_bolfi.models.standard_parameters.dOCV$_p$_dT (
SOC_p)

Non-dimensionalized ∂ cathode OCV / ∂ temperature.

5.7.2.14 $\mathbf{dOCV_p\_dT\_dim()}$ def ep_bolfi.models.standard_parameters.dOCV$_p$_dT_dim (
SOC_p)

∂ cathode OCV / ∂ temperature.

5.7.2.15 $\mathbf{dOCV_p\_dT\_dSOC_p()}$ def ep_bolfi.models.standard_parameters.dOCV$_p$_dT_dSOC$_p$ (
SOC_p)

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

5.7.2.16 $\mathbf{dOCV_p\_dT\_dSOC_p\_dim()}$ def ep_bolfi.models.standard_parameters.dOCV$_p$_dT_dSOC$_p$_dim (
SOC_p)

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

5.7.2.17 $\mathbf{D_e()}$ def ep_bolfi.models.standard_parameters.D$_e$ (
c_e,
T)

Non-dimensionalized electrolyte diffusivity.

5.7.2.18 $\mathbf{D_e\_dim()}$ def ep_bolfi.models.standard_parameters.D$_e$_dim (
c_e_dim,
T_dim)

Electrolyte diffusivity.

5.7.2.19 $\mathbf{D_n()}$ def ep_bolfi.models.standard_parameters.D$_n$ (
SOC_n,
T)

Non-dimensionalized anode diffusivity.

5.7.2.20 D_n_dim() def ep_bolfi.models.standard_parameters.D_n_dim (
SOC_n,
T_dim)

Anode diffusivity.

5.7.2.21 D_p() def ep_bolfi.models.standard_parameters.D_p (
SOC_p,
T)

Non-dimensionalized cathode diffusivity.

5.7.2.22 D_p_dim() def ep_bolfi.models.standard_parameters.D_p_dim (
SOC_p,
T_dim)

Cathode diffusivity.

5.7.2.23 i_{sen}_0() def ep_bolfi.models.standard_parameters.i_{sen}_0 (
c_{en},
SOC_n_surf,
c_n_max,
T)

Non-dimensionalized anode exchange current density.

5.7.2.24 i_{sen}_0_dim() def ep_bolfi.models.standard_parameters.i_{sen}_0_dim (
c_{en}_dim,
SOC_n_surf_dim,
c_n_max,
T_dim)

Anode exchange current density.

5.7.2.25 i_{sep}_0() def ep_bolfi.models.standard_parameters.i_{sep}_0 (
c_{ep},
SOC_p_surf,
c_p_max,
T)

Non-dimensionalized cathode exchange current density.

5.7.2.26 i_{sep}_0_dim() def ep_bolfi.models.standard_parameters.i_{sep}_0_dim (
c_{ep}_dim,
SOC_p_surf_dim,
c_p_max,
T_dim)

Cathode exchange current density.

5.7.2.27 OCV_n() def ep_bolfi.models.standard_parameters.OCV_n (
SOC_n,
T)

Non-dimensionalized anode OCV.

5.7.2.28 OCV_n_dim() def ep_bolfi.models.standard_parameters.OCV_n_dim (
SOC_n,
T_dim)

Anode OCV.

5.7.2.29 OCV_p() def ep_bolfi.models.standard_parameters.OCV_p (
SOC_p,
T)

Non-dimensionalized cathode OCV.

5.7.2.30 OCV_p_dim() def ep_bolfi.models.standard_parameters.OCV_p_dim (
SOC_p,
T_dim)

Cathode OCV.

5.7.2.31 one_plus_dlnf_dlnc() def ep_bolfi.models.standard_parameters.one_plus_dlnf_dlnc (
c_e)

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

5.7.2.32 one_plus_dlnf_dlnc_dim() def ep_bolfi.models.standard_parameters.one_plus_dlnf_dlnc_dim (*c_e_dim*)

Thermodynamic factor.

5.7.2.33 SOC$_n$_dim_init() def ep_bolfi.models.standard_parameters.SOC$_n$_dim_init (*x*)

Initial SOC of the anode.

5.7.2.34 SOC$_n$_init() def ep_bolfi.models.standard_parameters.SOC$_n$_init (*x*)

Non-dimensionalized initial SOC of the anode.

5.7.2.35 SOC$_p$_dim_init() def ep_bolfi.models.standard_parameters.SOC$_p$_dim_init (*x*)

Initial SOC of the cathode.

5.7.2.36 SOC$_p$_init() def ep_bolfi.models.standard_parameters.SOC$_p$_init (*x*)

Non-dimensionalized initial SOC of the cathode.

5.7.2.37 t_plus() def ep_bolfi.models.standard_parameters.t_plus (*c_e*)

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

5.7.2.38 t_plus_dim() def ep_bolfi.models.standard_parameters.t_plus_dim (*c_e_dim*)

Transference number.

5.7.2.39 κ_e() def ep_bolfi.models.standard_parameters.κ_e (
c_e,
T)

Non-dimensionalized electrolyte conductivity.

5.7.2.40 κ_e_dim() def ep_bolfi.models.standard_parameters.κ_e_dim (
c_e_dim,
T_dim)

Electrolyte conductivity.

5.7.3 Variable Documentation

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

5.7.3.2 A_cc ep_bolfi.models.standard_parameters.A_cc = A

5.7.3.3 a_n ep_bolfi.models.standard_parameters.a_n = a_n_dim ∗ R_n

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

5.7.3.5 a_p ep_bolfi.models.standard_parameters.a_p = a_p_dim ∗ R_p

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

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

5.7.3.8 capacity

ep_bolfi.models.standard_parameters.capacity = Parameter("Nominal cell capacity [A.h]")

5.7.3.9 current_with_time

ep_bolfi.models.standard_parameters.current_with_time = I_extern

5.7.3.10 C_{rn}

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

5.7.3.11 C_{rp}

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

5.7.3.12 C_e

int ep_bolfi.models.standard_parameters.C_e = τ_e / τ^{d}

5.7.3.13 c_e_dim_init

ep_bolfi.models.standard_parameters.c_e_dim_init = Parameter("Initial concentration in electrolyte [mol.↩
m-3]")

5.7.3.14 c_e_init

ep_bolfi.models.standard_parameters.c_e_init = c_e_dim_init / c_e_typ

5.7.3.15 c_e_typ

ep_bolfi.models.standard_parameters.c_e_typ = pybamm.Parameter("Typical electrolyte concentration [mol.m-3]")

5.7.3.16 c_n

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

5.7.3.17 C_n

int ep_bolfi.models.standard_parameters.C_n = τ_n / τ^{d}

5.7.3.18 c_p

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

5.7.3.19 C_p

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

5.7.3.20 dimensional_current_density_with_time

ep_bolfi.models.standard_parameters.dimensional_current_←
density_with_time = I_extern_dim / A

5.7.3.21 dimensional_current_with_time

ep_bolfi.models.standard_parameters.dimensional_current_with_time =
I_extern_dim

5.7.3.22 D_e_typ

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

5.7.3.23 D_n_typ

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

5.7.3.24 D_p_typ

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

5.7.3.25 F

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

5.7.3.26 I_extern

ep_bolfi.models.standard_parameters.I_extern = I_extern_dim / C

5.7.3.27 I_extern_dim

ep_bolfi.models.standard_parameters.I_extern_dim

Initial value:

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

5.7.3.28 I_typ

ep_bolfi.models.standard_parameters.I_typ = C

5.7.3.29 i_{sen}_0_ref def ep_bolfi.models.standard_parameters.i_{sen}_0_ref = i_{sen}_0_dim(c_e_typ, 0.5 ∗ c_n, c_n, T_ref)

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

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

5.7.3.32 L_dim ep_bolfi.models.standard_parameters.L_dim = L_n_dim + L_s_dim + L_p_dim

5.7.3.33 L_x ep_bolfi.models.standard_parameters.L_x = L_dim

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

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

5.7.3.36 L_e ep_bolfi.models.standard_parameters.L_e

Initial value:

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

5.7.3.37 L_n ep_bolfi.models.standard_parameters.L_n = L_n_dim / L_dim

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

5.7.3.39 L_p

ep_bolfi.models.standard_parameters.L_p = L_p_dim / L_dim

5.7.3.40 L_p_dim

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

5.7.3.41 L_s

ep_bolfi.models.standard_parameters.L_s = L_s_dim / L_dim

5.7.3.42 L_s_dim

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

5.7.3.43 n_cells

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

5.7.3.44 n_electrodes_parallel

ep_bolfi.models.standard_parameters.n_electrodes_parallel

Initial value:

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

5.7.3.45 OCV_n_ref

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

5.7.3.46 OCV_p_ref

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

5.7.3.47 Q

tuple ep_bolfi.models.standard_parameters.Q = (1 - ε_p_scalar) ∗ L_p_dim ∗ c_p ∗ z_p ∗ F ∗ A

5.7.3.48 q_e

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

5.7.3.49 R

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

5.7.3.50 R_n

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

5.7.3.51 R_p

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

5.7.3.52 T_init

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

5.7.3.53 T_ref

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

5.7.3.54 thermal_voltage

ep_bolfi.models.standard_parameters.thermal_voltage = R $*$ T_ref / F

5.7.3.55 timescale

ep_bolfi.models.standard_parameters.timescale = τ^d

5.7.3.56 U_u

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

5.7.3.57 U_l

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

5.7.3.58 V

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

5.7.3.59 voltage_high_cut

ep_bolfi.models.standard_parameters.voltage_high_cut = U_u

5.7.3.60 voltage_low_cut

ep_bolfi.models.standard_parameters.voltage_low_cut = U_l

5.7.3.61 z_n

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

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

5.7.3.63 ΔT ep_bolfi.models.standard_parameters.ΔT = Scalar(1)

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

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

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

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

5.7.3.68 β_e ep_bolfi.models.standard_parameters.β_e = pybamm.Concatenation(β_{en}, β_{es}, β_{ep})

5.7.3.69 β_{en} ep_bolfi.models.standard_parameters.β_{en} = pybamm.PrimaryBroadcast(β_{en}_scalar, "negative electrode")

5.7.3.70 β_{en}_scalar ep_bolfi.models.standard_parameters.β_{en}_scalar

Initial value:

```
= Parameter(
    "Negative electrode Bruggeman coefficient (electrolyte)"
)
```

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

5.7.3.72 β_{ep}_scalar ep_bolfi.models.standard_parameters.β_{ep}_scalar

Initial value:

```
= Parameter(
    "Positive electrode Bruggeman coefficient (electrolyte)"
)
```

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

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

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

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

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

5.7.3.78 γ_e ep_bolfi.models.standard_parameters.γ_e = c_e_typ / c_p

5.7.3.79 γ_n ep_bolfi.models.standard_parameters.γ_n = c_n / c_p

5.7.3.80 γ_p ep_bolfi.models.standard_parameters.γ_p = c_p / c_p

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

5.7.3.82 ε^{β}

ep_bolfi.models.standard_parameters.ε^{β} = ε**β_e

5.7.3.83 ε_n

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

5.7.3.84 ε_n_scalar

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

5.7.3.85 ε_p

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

5.7.3.86 ε_p_scalar

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

5.7.3.87 ε_s

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

5.7.3.88 ε_s_scalar

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

5.7.3.89 κ_e_hat

tuple ep_bolfi.models.standard_parameters.κ_e_hat = (R * T_ref / F) / (C / A * L_dim / κ_e_typ)

5.7.3.90 κ_e_typ

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

5.7.3.91 σ_n

tuple ep_bolfi.models.standard_parameters.σ_n = (thermal_voltage / (C / A * L_dim)) * σ_n_dim

5.7.3.92 σ_n_dim

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

5.7.3.93 σ_p

tuple ep_bolfi.models.standard_parameters.σ_p = (thermal_voltage / (C / A * L_dim)) * σ_p_dim

5.7.3.94 σ_p_dim

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

5.7.3.95 τ^d

ep_bolfi.models.standard_parameters.τ^d = F ∗ c_p ∗ L_dim / (C / A)

5.7.3.96 τ_{rn}

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

5.7.3.97 τ_{rp}

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

5.7.3.98 τ_e

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

5.7.3.99 τ_n

int ep_bolfi.models.standard_parameters.τ_n = R_n∗∗2 / D_n_typ

5.7.3.100 τ_p

int ep_bolfi.models.standard_parameters.τ_p = R_p∗∗2 / D_p_typ

5.8 ep_bolfi.optimization Namespace Reference

Namespaces

- EP_BOLFI

5.9 ep_bolfi.optimization.EP_BOLFI Namespace Reference

Classes

- class NDArrayEncoder
- class Preprocessed_Simulator

  Normalizes sampling to a standard normal distribution.
- class Optimizer_State

  Handles the heuristics for the EP-BOLFI operation modes.
- class EP_BOLFI

  Expectation Propagation and Bayesian Optimization.

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.

5.9.1 Function Documentation

5.9.1.1 combine_parameters_to_try() def ep_bolfi.optimization.EP_BOLFI.combine_parameters_to_try (*parameters*, *parameters_to_try_dict*)

Give every combination as full parameter sets.

Parameters

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.

5.9.1.2 fix_parameters() def ep_bolfi.optimization.EP_BOLFI.fix_parameters (*parameters_to_be_fixed*)

Returns a function which sets some parameters in advance.

Parameters

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.

5.10 ep_bolfi.utility Namespace Reference

Namespaces

- dataset_formatting

Defines datatypes for further processing.
- fitting_functions

  Various helper and fitting functions for processing measurement curves.
- preprocessing

  Contains frequently used workflows in dataset preprocessing.
- visualization

  Various helper and plotting functions for common data visualizations.

5.11 ep_bolfi.utility.dataset_formatting Namespace Reference

Defines datatypes for further processing.

Classes

- class Measurement

  Defines common methods for measurement objects.
- class Cycling_Information

  Contains basic cycling informations.
- class Static_Information

  Contains additional informations, e.g.
- class Impedance_Measurement

  Contains basic impedance data.

Functions

- def read_csv_from_measurement_system (path, encoding, number_of_comment_lines, headers, delimiter='\t', decimal='.', datatype="cycling", segment_column=-1, segments_to_process=None, current_sign↩ _correction={}, correction_column=-1, flip_voltage_sign=False, flip_imaginary_impedance_sign=False, max_number_of_lines=-1)

  Read the measurements as returned by common instruments.
- def print_hdf5_structure (h5py_object, depth=1, table_limit=4, verbose_limit=64)

  brief Simple HDF5 structure viewer.
- def convert_none_notation_to_slicing (h5py_object, index)

  Access a slice of an HDF5 object by transferable notation.
- def get_hdf5_dataset_by_path (h5py_object, path)

  Follow the structure of a HDF object to get a certain part.
- def read_hdf5_table (path, data_location, headers, datatype="cycling", segment_location=None, segments_to_process=None, current_sign_correction={}, correction_location=None, flip_voltage_↩ sign=False, flip_imaginary_impedance_sign=False)

  Read the measurements as stored in a HDF5 file.

5.11.1 Detailed Description

Defines datatypes for further processing.

5.11.2 Function Documentation

5.11.2.1 convert_none_notation_to_slicing()

def ep_bolfi.utility.dataset_formatting.convert_none_notation_to_slicing (
h5py_object,
index)

Access a slice of an HDF5 object by transferable notation.

Parameters

h5py_object	The HDF5 object wrapped by H5Py, for example h5py.File(filename, 'r').
index	A 2-tuple or a 2-list. (None, x) denotes slicing [:, x] and (x, None) denotes slicing [x, :].

5.11.2.2 get_hdf5_dataset_by_path() def ep_bolfi.utility.dataset_formatting.get_hdf5_dataset_by_path (

h5py_object,
path)

Follow the structure of a HDF object to get a certain part.

Parameters

h5py_object	The HDF5 object wrapped by H5Py, for example h5py.File(filename, 'r').
path	A list. Each entry goes one level deeper into the HDF structure. Each entry can either be the index to go into next itself, or a 2-tuple or a 2-list. In the latter case, (None, x) denotes slicing [:, x] and (x, None) denotes slicing [x, :].

Returns

Returns the HDF5 object found at the end of ”path”.

5.11.2.3 print_hdf5_structure() def ep_bolfi.utility.dataset_formatting.print_hdf5_structure (

h5py_object,
depth = 1,
table_limit = 4,
verbose_limit = 64)

brief Simple HDF5 structure viewer.

Parameters

h5py_object	The HDF5 object wrapped by H5Py, for example h5py.File(filename, 'r').
depth	For pretty printing the recursive depth. Do not change.
table_limit	h5py.Dataset object tables will be truncated prior to printing up to this number in the higher dimension.
verbose_limit	h5py.Dataset objects will be truncated to at most this number in any dimension.

5.11.2.4 read_csv_from_measurement_system() def ep_bolfi.utility.dataset_formatting.read_csv_from_measurement↩ _system (

path,
encoding,

number_of_comment_lines,
headers,
delimiter = '\t',
decimal = '.',
datatype = "cycling",
segment_column = -1,
segments_to_process = None,
current_sign_correction = {},
correction_column = -1,
flip_voltage_sign = False,
flip_imaginary_impedance_sign = False,
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.

Parameters

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: ".".
datatype	Default is "cycling", where cycling information is assumed in the file. "static" will trigger the additional extraction of exponential decays that are relevant to e.g. GITT. "impedance" will treat the file as an impedance measurement with frequencies and impedances instead of time and voltage.
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.
segments_to_process	A list of indices which give the segments that shall be processed. Default is None, i.e., the whole file gets processed.
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).

Parameters

flip_voltage_sign	Defaults to False, where measured voltage remains unaltered. Change to True if the voltage shall be multiplied by -1. Also applies to impedances; real and imaginary parts.
flip_imaginary_impedance_sign	Defaults to False, where measured impedance remains unaltered. Change to True if the imaginary part of the impedance shall be multiplied by -1. Cancels out with 'flip-voltage-sign'.

Returns

The measurement, packaged in a Measurement subclass. It depends on ”datatype” which one it is:

- ”cycling”: Cycling_Information
- ”static”: Static_Information
- ”impedance”: Impedance_Measurement

5.11.2.5 read_hdf5_table()

def ep_bolfi.utility.dataset_formatting.read_hdf5_table (
path,
data_location,
headers,
datatype = "cycling",
segment_location = None,
segments_to_process = None,
current_sign_correction = {},
correction_location = None,
flip_voltage_sign = False,
flip_imaginary_impedance_sign = False)

Read the measurements as stored in a HDF5 file.

Parameters

path	The full or relative path to the measurement file.
data_location	A list. Gives the location in the HDF5 file where the data table is stored. Set to None if everything is stored at the top level. Each entry goes one level deeper into the HDF structure. Each entry can either be the index to go into next itself, or a 2-tuple or a 2-list. In the latter case, (None, x) denotes slicing [:, x] and (x, None) denotes slicing [x, :].
headers	A dictionary. Its keys are 2-tuples, slicing the data which are to be read in. Use the format ”(x, None)” or ”(None, x)” to slice the dimension with ”None”. The corresponding values are there to tell this function which kind of data is in which column. If necessary, the keys may also be tuples like 'data_location'. 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”.

Parameters

datatype	Default is ”cycling”, where cycling information is assumed in the file. ”static” will trigger the additional extraction of exponential decays that are relevant to e.g. GITT. ”impedance” will treat the file as an impedance measurement with frequencies and impedances instead of time and voltage. @segment_location A list, with the same format as ”data_location”. It points to the part of the data 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 None, which returns the dataset in one segment.
segments_to_process	A list of indices which give the segments that shall be processed. Default is None, i.e., the whole file gets processed.
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_location	A list, with the same format as ”data_location”. For its use, see ”current_sign_correction”. Default: None.
flip_voltage_sign	Defaults to False, where measured voltage remains unaltered. Change to True if the voltage shall be multiplied by -1. Also applies to impedances; real and imaginary parts.
flip_imaginary_impedance_sign	Defaults to False, where measured impedance remains unaltered. Change to True if the imaginary part of the impedance shall be multiplied by -1. Cancels out with 'flip-voltage-sign'.

Returns

The measurement, packaged in a Measurement subclass. It depends on ”datatype” which one it is:

- ”cycling”: Cycling_Information
- ”static”: Static_Information
- ”impedance”: Impedance_Measurement

5.12 ep_bolfi.utility.fitting_functions Namespace Reference

Various helper and fitting functions for processing measurement curves.

Classes

- class NDArrayEncoder
- class OCV_fit_result

  Contains OCV fit parameters and related information.

Functions

- def find_occurrences (sequence, value)

  Gives indices in sequence where it is closest to value.
- def smooth_fit (x, y, order=3, splits=None, w=None, s=None, display=False, derivatives=0)

  Calculates a smoothed spline with derivatives.
- def fit_exponential_decay (timepoints, voltages, recursive_depth=1, threshold=0.95)

  Extracts a set amount of exponential decay curves.
- def fit_sqrt (timepoints, voltages, threshold=0.95)

  Extracts a square root at the beginning of the data.
- def fit_drt (frequencies, impedances, lambda_value=-2.0)
- 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, fit_SOC_range=False, rescale=False)

  The OCV model from "A parametric OCV model".
- def d_dE_OCV_fit_function (E_OCV, ∗args, z=1.0, T=298.15, individual=False, fit_SOC_range=False, rescale=False)

  The derivative of fitting_functions.OCV_fit_function.
- def d2_dE2_OCV_fit_function (E_OCV, ∗args, z=1.0, T=298.15, individual=False, fit_SOC_range=False, rescale=False)

  The 2^{n} derivative of fitting_functions.OCV_fit_function.
- def inverse_OCV_fit_function (SOC, ∗args, z=1.0, T=298.15, inverted=True)

  The inverse of fitting_functions.OCV_fit_function.
- def inverse_d_dSOC_OCV_fit_function (SOC, ∗args, z=1.0, T=298.15, inverted=True)

  The derivative of the inverse of fitting_functions.OCV_fit_function.
- def inverse_d2_dSOC2_OCV_fit_function (SOC, ∗args, z=1.0, T=298.15, inverted=True)

  The 2nd derivative of the inverse of .OCV_fit_function.
- def fit_OCV (SOC, OCV, N=4, SOC_range_bounds=(0.2, 0.8), SOC_range_limits=(0.0, 1.0), z=1.0, T=298.↩ 15, inverted=True, fit_SOC_range=True, distance_order=2, weights=None, initial_parameters=None, minimize_options=None)

  Fits data to fitting_functions.OCV_fit_function.
- def verbose_spline_parameterization (coeffs, knots, order, format='python', function_name="OCV", function_args="SOC", derivatives=0, spline_transformation='', verbose=False)

  Gives the monomic representation of a B-spline.

5.12.1 Detailed Description

Various helper and fitting functions for processing measurement curves.

5.12.2 Function Documentation

5.12.2.1 a_fit() def ep_bolfi.utility.fitting_functions.a_fit (*γUeminus1*)

Calculates the conversion from "A parametric OCV model".

Parameters

γUeminus1	$\gamma * U_i$ / e from "A parametric OCV model".

Returns

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

5.12.2.2 d2_dE2_OCV_fit_function()

def ep_bolfi.utility.fitting_functions.d2_dE2_OCV_fit_function (
E_OCV,
∗ args,
z = 1.0,
T = 298.15,
individual = False,
fit_SOC_range = False,
rescale = False)

The 2nd derivative of fitting_functions.OCV_fit_function.

Parameters

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...). The last Δx_j may be omitted to force Σ Δx_j = 1.
z	The charge number of the electrode interface reaction.
T	The temperature of the electrode.
individual	If True, the model function summands are not summed up.
fit_SOC_range	If True, this function takes two additional arguments at the start of "args" that may be used to adjust the data SOC range.
rescale	If True, the expected "args" now contain the slopes of the summands at their respective origins instead of a. Formula: a / slope = -4 ∗ (k_B / e) ∗ T / (Δx ∗ z).

Returns

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

5.12.2.3 d_dE_OCV_fit_function()

def ep_bolfi.utility.fitting_functions.d_dE_OCV_fit_function (
E_OCV,
∗ args,
z = 1.0,
T = 298.15,
individual = False,
fit_SOC_range = False,
rescale = False)

The derivative of fitting_functions.OCV_fit_function.

Parameters

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...). The last Δx_j may be omitted to force Σ Δx_j = 1.
z	The charge number of the electrode interface reaction.
T	The temperature of the electrode.
individual	If True, the model function summands are not summed up.
fit_SOC_range	If True, this function takes two additional arguments at the start of "args" that may be used to adjust the data SOC range.
rescale	If True, the expected "args" now contain the slopes of the summands at their respective origins instead of a. Formula: a / slope = -4 ∗ (k_B / e) ∗ T / (Δx ∗ z).

Returns

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

5.12.2.4 find_occurrences() def ep_bolfi.utility.fitting_functions.find_occurrences (
sequence,
value)

Gives indices in sequence where it is closest to value.

Parameters

sequence	A list that represents a differentiable function.
value	The value that is searched for in "sequence". Also, crossings of consecutive values in "sequence" with "value" are searched for.

Returns

A list of indices in "sequence" in ascending order where "value" or a close match for "value" was found.

5.12.2.5 fit_drt() def ep_bolfi.utility.fitting_functions.fit_drt (
frequencies,
impedances,
lambda_value = -2.0)

```
Distribution of Relaxation Times.
@param frequencies
    Array of the measured frequencies.
@param impedances
    Array of the measured complex impedances.
@param lambda_value
    Takes the place of the R² value as a tuning parameter.
    Consult the pyimpspec documentation for the precise usage.
    Default is -2, which "uses the L-curve approach to estimate lambda".
    -1 uses a different heuristic, and values > 0 set lambda directly.
@return
    A 3-tuple of characteristic DRT time constants, their corresponding
    resistances, and the whole TRNNLSResult object returned by pyimpspec.
    Note that the .pseudo_chisqr attribute is not useful for the same
    reasons that went into the R² calculations in the other fit functions.
```

5.12.2.6 fit_exponential_decay()

def ep_bolfi.utility.fitting_functions.fit_exponential_decay (
timepoints,
voltages,
recursive_depth = 1,
threshold = 0.95)

Extracts a set amount of exponential decay curves.

Parameters

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 lower threshold value for the R^2 coefficient of determination. If "threshold" is smaller than 1, the subset of the exponential decay data is searched that just fulfills it. Defaults to 0.95. Values above 1 are set to 1.

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, (U_0, ΔU, τ_r^{-1}) $\mapsto$ U_0 + ΔU $*$ np.exp(-τ_r^{-1} $*$ (t - timepoints[0])).

5.12.2.7 fit_OCV()

def ep_bolfi.utility.fitting_functions.fit_OCV (
SOC,
OCV,
N = 4,
SOC_range_bounds = (0.2, 0.8),
SOC_range_limits = (0.0, 1.0),
z = 1.0,
T = 298.15,
inverted = True,
fit_SOC_range = True,
distance_order = 2,
weights = None,
initial_parameters = None,
minimize_options = None)

Fits data to fitting_functions.OCV_fit_function.

In addition to the fit itself, 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. If "other_electrode" is not None, the stoichiometric offset and the scaling that gives the (adjusted) SOC of this electrode given the SOC of the "other_electrode" are put before SOC_start and SOC_end.

Parameters

SOC	The SOCs at which measurements were made.
OCV	The corresponding open-circuit voltages.
N	The number of phases of the OCV model.
SOC_range_bounds	Optional hard upper and lower bounds for the SOC correction from the left and the right side, respectively, as a 2-tuple. Use it as a limiting guess for the actual SOC range represented in the measurement. Has to be inside (0.0, 1.0). Set to (0.0, 1.0) to effectively disable SOC range estimation.
SOC_range_limits	Optional hard lower and upper bounds for the SOC correction from the left and the right side, respectively, as a 2-tuple. Use it if you know that your OCV data is incomplete and by how much. Has to be inside (0.0, 1.0). Set to (0.0, 1.0) to allow the SOC range estimation to assign datapoints to the asymptotes.
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.
fit_SOC_range	If True (default), a model-based correction to the provided SOC-OCV-data is made.
distance_order	The order of the norm of the vector of the distances between OCV data and OCV model. Default is 2, i.e., the Euclidean norm. 1 sets it to absolute distance, and float('inf') sets it to maximum distance. Note that 1 will lead to worse performance.
weights	Optional weights to apply to the vector of the distances between OCV data and OCV model. Defaults to equal weights.
initial_parameters	Optional initial guess for the model parameters. If left as-is, this will be automatically gleaned from the data. Use only if you have another fit to data of the same electrode material.
minimize_options	Dictionary that gets passed to scipy.optimize.minimize with the method 'trust-constr'. See scipy.optimize.show_options with the arguments 'minimize' and 'trust-constr' for details.

Returns

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

5.12.2.8 fit_sqrt() def ep_bolfi.utility.fitting_functions.fit_sqrt (

timepoints,
voltages,
threshold = 0.95)

Extracts a square root at the beginning of the data.

Parameters

timepoints	The timepoints of the measurements.
voltages	The corresponding voltages.
threshold	The lower threshold value for the R^2 coefficient of determination. If ”threshold” is smaller than 1, the subset of the experimental data is searched that just fulfills it. Defaults to 0.95. Values above 1 are set to 1.

Returns

A 3-tuple with the timepoints, the fitted voltage evaluations and a 2-tuple of the parameters of the following sqrt function: t, (U_0, dU_d√t) ↦ U_0 + dU_d√t ∗ √(t - timepoints[0]).

5.12.2.9 inverse_d2_dSOC2_OCV_fit_function()

def ep_bolfi.utility.fitting_functions.inverse_d2_dSOC2_OCV_fit_↩
function (

SOC,
∗ *args,*
z = 1.0,
T = 298.15,
inverted = True)

The 2nd derivative of the inverse of .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.

Parameters

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 derivative of the inverse model function.

5.12.2.10 inverse_d_dSOC_OCV_fit_function()

def ep_bolfi.utility.fitting_functions.inverse_d_dSOC_OCV_fit_function
(

SOC,
∗ *args,*
z = 1.0,
T = 298.15,
inverted = True)

The derivative of 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.

Parameters

E_OCV	The SOCs for which the voltages shall be evaluated.

Parameters

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 derivative of the inverse model function.

5.12.2.11 inverse_OCV_fit_function()

def ep_bolfi.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.

Parameters

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.

5.12.2.12 laplace_transform()

def ep_bolfi.utility.fitting_functions.laplace_transform (
x,
y,
s)

Performs a basic laplace transformation.

Parameters

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.

5.12.2.13 OCV_fit_function()

def ep_bolfi.utility.fitting_functions.OCV_fit_function (
E_OCV,
∗ args,
z = 1.0,
T = 298.15,
individual = False,
fit_SOC_range = False,
rescale = False)

The OCV model from ”A parametric OCV model”.

5.12.2.14 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

Parameters

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 the paper referenced above in the order (E_0_0, a_0, Δx_0, E_0_1, a_1, Δx_1...). If ”fit_SOC_range” is True, two additional arguments are at the front (see there). The last Δx_j may be omitted to force Σ Δx_j = 1.
z	The charge number of the electrode interface reaction.
T	The temperature of the electrode.
individual	If True, the model function summands are not summed up.
fit_SOC_range	If True, this function takes two additional arguments at the start of ”args” that may be used to adjust the data SOC range.
rescale	If True, the expected ”args” now contain the slopes of the summands at their respective origins instead of a. Formula: a / slope = -4 ∗ (k_B / e) ∗ T / (Δx ∗ z).

Returns

The evaluation of the model function. This is the referenced fit function on a linearly transformed SOC range if ”fit_SOC_range” is True. If ”individual” is True, the individual summands in the model function get returned as a list.

5.12.2.15 smooth_fit()

def ep_bolfi.utility.fitting_functions.smooth_fit (
x,
y,
order = 3,
splits = None,
w = None,
s = None,
display = False,
derivatives = 0)

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.

Parameters

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.
w	Optional list of weights. Works best if 1 / w approximates the standard deviation of the noise in y at each point. Defaults to SciPy default behaviour of scipy.interpolate.UnivariateSpline.
s	Optional tuning parameter. Higher values lead to coarser, but more smooth interpolations and vice versa. Defaults to SciPy default behaviour of scipy.interpolate.UnivariateSpline.
display	If set to True, the fit parameters of the spline will be printed to console. If possible, a monomial representation is printed.
derivatives	The derivatives of the spline to also include in the return. Default is 0, which gives the spline. 1 would give the spline, followed by its derivative. Can not be higher than spline order. Derivatives are only continuous when derivatives $<$ order.

Returns

A smoothing spline in the form of scipy.UnivariateSpline.

5.12.2.16 verbose_spline_parameterization()

def ep_bolfi.utility.fitting_functions.verbose_spline_parameterization (
coeffs,
knots,
order,
format = 'python',
function_name = "OCV",
function_args = "SOC",
derivatives = 0,
spline_transformation = '',
verbose = False)

Gives the monomic representation of a B-spline.

Parameters

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.
format	Gives the file/language format for the function representation. Default is 'python'. The other choice is 'matlab'.
function_name	An optional name for the printed Python function.
function_args	An optional string for the arguments of the function.
derivatives	The derivatives of the spline to also include in the return. Default is 0, which gives the spline. 1 would give the spline, followed by its derivative. Can not be higher than spline order. Derivatives are only continuous when derivatives $<$ order.
spline_transformation	Give a string if you want to include a function that gets applied to the whole spline, e.g. 'exp'.
verbose	Print information about the progress of the conversion.

Returns

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

5.13 ep_bolfi.utility.preprocessing Namespace Reference

Contains frequently used workflows in dataset preprocessing.

Classes

- class SubstitutionDict

  A dictionary with some automatic substitutions.

Functions

- def fix_parameters (parameters_to_be_fixed)

  Returns a function which sets some parameters in advance.
- def combine_parameters_to_try (parameters, parameters_to_try_dict)

  Give every combination as full parameter sets.
- def calculate_means_and_standard_deviations (mean, covariance, free_parameters_names, transform_↩ parameters={}, bounds_in_standard_deviations=1, ∗∗kwargs)

  Calculate means and standard deviations.
- def approximate_confidence_ellipsoid (parameters, free_parameters_names, covariance, mean=None, transform_parameters={}, refinement=True, confidence=0.95)

  Approximate a confidence ellipsoid.
- def capacity (parameters, electrode=”positive”)

  Convenience function for calculating the capacity.
- def calculate_SOC (timepoints, currents, initial_SOC=0, sign=1, capacity=1)

  Transforms applied current over time into SOC.
- def calculate_both_SOC_from_OCV (parameters, negative_SOC_from_cell_SOC, positive_SOC_from_↩ cell_SOC, OCV)

  Calculates the SOC of both electrodes from their OCV.

- def subtract_OCV_curve_from_cycles (dataset, parameters, starting_SOC=None, starting_OCV=None, electrode="positive", current_sign=0, voltage_sign=0)

  Removes the OCV curve from a cycling measurement.
- def subtract_both_OCV_curves_from_cycles (dataset, parameters, negative_SOC_from_cell_SOC, positive_SOC_from_cell_SOC, starting_SOC=None, starting_OCV=None)

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

  Performs a basic laplace transformation.
- def find_occurrences (sequence, value)

  Gives indices in sequence where it is closest to value.
- 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=None, parameters_print=False)

  Tries to extract the OCV curve from CC-CV cycling data.
- def calculate_desired_voltage (solution, t_eval, voltage_scale, overpotential, three_electrode=None, dimensionless_reference_electrode_location=0.5, parameters={})
- def solve_all_parameter_combinations (model, t_eval, parameters, parameters_to_try, submesh_types, var_pts, spatial_methods, full_factorial=True, ∗∗kwargs)
- def prepare_parameter_combinations (parameters, parameters_to_try, covariance, order_of_parameter↩ _names, transform_parameters, confidence)

  Calculates all permutations of the parameter boundaries.
- def parallel_simulator_with_setup (model, current_input, parameters, submesh_types, var_pts, spatial↩ _methods, calc_esoh, inputs, t_eval, voltage_scale, overpotential, three_electrode, dimensionless_↩ reference_electrode_location, kwargs)
- def simulate_all_parameter_combinations (model, current_input, submesh_types, var_pts, spatial_↩ methods, parameters, parameters_to_try=None, covariance=None, order_of_parameter_names=None, additional_input_parameters=[ ], transform_parameters={}, confidence=0.95, full_factorial=True, calc↩ _esoh=False, voltage_scale=1.0, overpotential=False, three_electrode=None, dimensionless_reference_↩ electrode_location=0.5, ∗∗kwargs)

5.13.1 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.

5.13.2 Function Documentation

5.13.2.1 approximate_confidence_ellipsoid() def ep_bolfi.utility.preprocessing.approximate_confidence_ellipsoid (

parameters,
free_parameters_names,
covariance,
mean = None,
transform_parameters = {},
refinement = True,
confidence = 0.95)

Approximate a confidence ellipsoid.

Compatible with SubstitutionDict, if "parameters" is one. The geometric approximation is a refinement of the polytope with nodes on the semiaxes of the confidence ellipsoid. The refinement step adds a node for each face, i.e., each sub-polytope with dimension smaller by 1. This node is centered on that face and projected onto the confidence ellipsoid.

Parameters

parameters	The base full parameter set as a dictionary.
free_parameters_names	The names of the parameters that are uncertain as a list. This parameter has to match the order of parameters in "covariance".
covariance	The covariance of the uncertain parameters as a two-dimensional numpy array.
mean	The mean of the uncertain parameters as a dictionary. If not set, the values from 'parameters' will be used.
transform_parameters	Optional transformations between the parameter space that is used for searching for optimal parameters and the model parameters. Any missing free parameter is not transformed. The values are 2-tuples. The first entry is a function taking the search space parameter and returning the model parameter. The second entry is the inverse function. For convenience, any value may also be one of the following: • 'none' => (identity, identity) • 'log' => (exp, log)
confidence	The confidence within the ellipsoid. Defaults to 0.95, i.e., the 95% confidence ellipsoid.
refinement	If False, only the nodes on the semiaxes get returned. If True, the nodes centered on the faces get returned as well.

Returns

A 2-tuple where the first item is the list of all parameter set combinations and the second the ellipsoid nodes only as a two-dimensional numpy array with each node in on row.

5.13.2.2 calculate_both_SOC_from_OCV()

def ep_bolfi.utility.preprocessing.calculate_both_SOC_from_OCV (
parameters,
negative_SOC_from_cell_SOC,
positive_SOC_from_cell_SOC,
OCV)

Calculates the SOC of both electrodes from their OCV.

The SOCs are substitued in the given "parameters". The SOC of the cell as a whole gets returned in case it is needed.

Parameters

parameters	The parameters of the battery as used for the PyBaMM simulations (see models.standard_parameters).
negative_SOC_from_cell_SOC	A function that takes the SOC of the cell and returns the SOC of the negative electrode.
positive_SOC_from_cell_SOC	A function that takes the SOC of the cell and returns the SOC of the positive electrode.
OCV	The OCV for which the SOCs shall be calculated.

Returns

The SOC of the cell as a whole.

5.13.2.3 calculate_desired_voltage()

def ep_bolfi.utility.preprocessing.calculate_desired_voltage (
solution,
t_eval,
voltage_scale,
overpotential,
three_electrode = None,
dimensionless_reference_electrode_location = 0.5,
parameters = {})

Parameters

solution	The pybamm.Solution object from which to calculate the voltage.
t_eval	The times at which to evaluate the "solution".
voltage_scale	The returned voltage gets divided by this value. For example, 1e-3 would produce a plot in [mV].
overpotential	If True, only the overpotential of "solutions" gets plotted. Otherwise, the cell voltage (OCV + overpotential) is plotted.
three_electrode	With None, does nothing (i.e., cell potentials are used). If set to either 'positive' or 'negative', instead of cell potentials, the base for the displayed voltage will be the potential of the 'positive' or 'negative' electrode against a reference electrode. For placement of said reference electrode, please refer to "dimensionless_reference_electrode_location".
dimensionless_reference_electrode_location	The location of the reference electrode, given as a scalar between 0 (placed at the point where negative electrode and separator meet) and 1 (placed at the point where positive electrode and separator meet). Defaults to 0.5 (in the middle).
parameters	The parameter dictionary that was used for the simulation. Only needed for a three-electrode output.

Returns

The array of the specified voltages over time.

5.13.2.4 calculate_means_and_standard_deviations()

def ep_bolfi.utility.preprocessing.calculate_means_and_↩
standard_deviations (
mean,
covariance,
free_parameters_names,
transform_parameters = {},
bounds_in_standard_deviations = 1,
∗∗ *kwargs*)

Calculate means and standard deviations.

Please note that standard deviations translate differently into confidence regions in different dimensions. For the confidence region, use ”approximate_confidence_ellipsoid”.

Parameters

mean	The mean of the uncertain parameters as a dictionary.
covariance	The covariance of the uncertain parameters as a two-dimensional numpy array.
free_parameters_names	The names of the parameters that are uncertain as a list. This parameter maps the order of parameters in "covariance".
transform_parameters	Optional transformations between the parameter space that is used for searching for optimal parameters and the model parameters. Any missing free parameter is not transformed. The values are 2-tuples. The first entry is a function taking the search space parameter and returning the model parameter. The second entry is the inverse function. For convenience, any value may also be one of the following: • 'none' => (identity, identity) • 'log' => (exp, log)
bounds_in_standard_deviations	Sets how many standard deviations in each direction the returned error bounds are. These are first applied and then transformed.
∗∗kwargs	Keyword arguments for scipy.integrate.quad, which is used to numerically calculate mean and variance.

Returns

A 3-tuple with three dictionaries. Their keys are the free parameters' names as keys and their values are those parameters' means, standard deviations, and error bounds.

5.13.2.5 calculate_SOC()

def ep_bolfi.utility.preprocessing.calculate_SOC (
timepoints,
currents,
initial_SOC = 0,
sign = 1,
capacity = 1)

Transforms applied current over time into SOC.

Parameters

timepoints	Array of the timepoint segments.
currents	Array of the current segments.
initial_SOC	The SOC value to start accumulating from.
sign	The value by which to multiply the current.
capacity	A scaling by which to convert from C to dimensionless SOC.

Returns

An array of the same shape describing SOC in C.

5.13.2.6 capacity() def ep_bolfi.utility.preprocessing.capacity (

parameters,
electrode = "positive")

Convenience function for calculating the capacity.

Parameters

parameters	A parameter file as defined by models.standard_parameters.
electrode	The prefix of the electrode to use for capacity calculation. Change to "negative" to use the one with the lower OCP.

Returns

The capacity of the parameterized battery in C.

5.13.2.7 combine_parameters_to_try() def ep_bolfi.utility.preprocessing.combine_parameters_to_try (

parameters,
parameters_to_try_dict)

Give every combination as full parameter sets.

Compatible with SubstitutionDict, if "parameters" is one.

Parameters

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.

5.13.2.8 find_occurrences() def ep_bolfi.utility.preprocessing.find_occurrences (

sequence,
value)

Gives indices in sequence where it is closest to value.

Parameters

sequence	A list that represents a differentiable function.
value	The value that is searched for in "sequence". Also, crossings of consecutive values in "sequence" with "value" are searched for.

Returns

A list of indices in "sequence" in ascending order where "value" or a close match for "value" was found.

5.13.2.9 fix_parameters()

def ep_bolfi.utility.preprocessing.fix_parameters (
parameters_to_be_fixed)

Returns a function which sets some parameters in advance.

Parameters

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.

5.13.2.10 laplace_transform()

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

Performs a basic laplace transformation.

Parameters

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.

5.13.2.11 OCV_from_CC_CV()

def ep_bolfi.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 = None,
parameters_print = False)

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

Parameters

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	If set to either 'python' or 'matlab', a string representation of the smoothing spline is printed in the respective format.
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. 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.

5.13.2.12 parallel_simulator_with_setup() def ep_bolfi.utility.preprocessing.parallel_simulator_with_setup (

model,
current_input,
parameters,

submesh_types,
var_pts,
spatial_methods,
calc_esoh,
inputs,
t_eval,
voltage_scale,
overpotential,
three_electrode,
dimensionless_reference_electrode_location,
kwargs)

5.13.2.13 prepare_parameter_combinations()

def ep_bolfi.utility.preprocessing.prepare_parameter_combinations (
parameters,
parameters_to_try,
covariance,
order_of_parameter_names,
transform_parameters,
confidence)

Calculates all permutations of the parameter boundaries.

Parameters

parameters	The model parameters as a dictionary.
parameters_to_try	A dictionary with the names of the model parameters as keys and lists of the values that are to be tried out for them as values. Mutually exclusive to "covariance".
covariance	A covariance matrix describing an estimation result of model parameters. Will be used to calculate parameters to try that together approximate the confidence ellipsoid. This confidence ellipsoid will be centered on "parameters". Mutually exclusive to "parameters_to_try".
order_of_parameter_names	A list of names from "parameters" that correspond to the order these parameters appear in the rows and columns of "covariance". Only needed when "covariance" is set.
transform_parameters	Optional transformations between the parameter space that is used for searching for optimal parameters and the model parameters. Any missing free parameter is not transformed. The values are 2-tuples. The first entry is a function taking the search space parameter and returning the model parameter. The second entry is the inverse function. For convenience, any value may also be one of the following: • 'none' => (identity, identity) • 'log' => (exp, log)
confidence	The confidence within the ellipsoid. Defaults to 0.95, i.e., the 95% confidence ellipsoid.

Returns

A 2-tuple with the individual parameter variations and then all permutations of them.

5.13.2.14 simulate_all_parameter_combinations()

def ep_bolfi.utility.preprocessing.simulate_all_parameter_↩
combinations (

model,
current_input,
submesh_types,
var_pts,
spatial_methods,
parameters,
parameters_to_try = None,
covariance = None,
order_of_parameter_names = None,
additional_input_parameters = [],
transform_parameters = {},
confidence = 0.95,
full_factorial = True,
calc_esoh = False,
voltage_scale = 1.0,
overpotential = False,
three_electrode = None,
dimensionless_reference_electrode_location = 0.5,
∗∗ *kwargs*)

Parameters

model	The PyBaMM battery model that is to be solved.
current_input	The list of battery operation conditions. See pybamm.Simulation.
submesh_types	The submeshes for discretization. See solversetup.spectral_mesh_pts_and_method.
var_pts	The number of discretization points. See solversetup.spectral_mesh_pts_and_method.
spatial_methods	The spatial methods for discretization. See solversetup.spectral_mesh_pts_and_method.
parameters	The model parameters as a dictionary.
parameters_to_try	A dictionary with the names of the model parameters as keys and lists of the values that are to be tried out for them as values. Mutually exclusive to ”covariance”.
covariance	A covariance matrix describing an estimation result of model parameters. Will be used to calculate parameters to try that together approximate the confidence ellipsoid. This confidence ellipsoid will be centered on ”parameters”. Mutually exclusive to ”parameters_to_try”.
order_of_parameter_names	A list of names from ”parameters” that correspond to the order these parameters appear in the rows and columns of ”covariance”. Only needed when ”covariance” is set.
additional_input_parameters	A list of the parameter names that are changed by any of the variable parameters, if ”parameters” is a SubstitutionDict.
transform_parameters	Optional transformations between the parameter space that is used for searching for optimal parameters and the model parameters. Any missing free parameter is not transformed. The values are 2-tuples. The first entry is a function taking the search space parameter and returning the model parameter. The second entry is the inverse function. For convenience, any value may also be one of the following: • 'none' => (identity, identity) • 'log' => (exp, log)

Parameters

confidence	The confidence within the ellipsoid. Defaults to 0.95, i.e., the 95% confidence ellipsoid.
full_factorial	When "parameters_to_try" is set: • If True, all parameter combinations are tried out. If False, only each parameter is varied with the others staying fixed. When "covariance" is set: • If False, only the points on the semiaxes of the confidence ellipsoid constitute the parameters to try. If True, the centres of the faces of the polytope of these points get added to the parameters to try, projected onto the surface of the confidence ellipsoid.
calc_esoh	Passed on to pybamm.Simulator, see there.
kwargs	The optional parameters for solversetup.solver_setup. See there.

Returns

A 2-tuple with the model solution for parameters as first entry. The second entry mimics parameters_to_try with each entry in their lists replaced by the model solution for the corresponding parameter substitution. The second entry has one additional key "all parameters", where all parameters_to_try combinations are the value.

5.13.2.15 solve_all_parameter_combinations()

def ep_bolfi.utility.preprocessing.solve_all_parameter_combinations (
model,
t_eval,
parameters,
parameters_to_try,
submesh_types,
var_pts,
spatial_methods,
full_factorial = True,
∗∗ *kwargs*)

Parameters

model	The PyBaMM battery model that is to be solved.
t_eval	The timepoints in s at which this model is to be solved.
parameters	The model parameters as a dictionary.
parameters_to_try	A dictionary with the names of the model parameters as keys and lists of the values that are to be tried out for them as values.
submesh_types	The submeshes for discretization. See solversetup.spectral_mesh_pts_and_method.
var_pts	The number of discretization points. See solversetup.spectral_mesh_pts_and_method.
spatial_methods	The spatial methods for discretization. See solversetup.spectral_mesh_pts_and_method.
full_factorial	If True, all parameter combinations are tried out. If False, only each parameter is varied with the others staying fixed.
kwargs	The optional parameters for solversetup.solver_setup. See there.

Returns

A 2-tuple with the model solution for parameters as first entry. The second entry mimics parameters_to_try with each entry in their lists replaced by the model solution for the corresponding parameter substitution. The second entry has one additional key "all parameters", where all parameters_to_try combinations are the value.

5.13.2.16 subtract_both_OCV_curves_from_cycles()

def ep_bolfi.utility.preprocessing.subtract_both_OCV_curves↩_from_cycles (
dataset,
parameters,
negative_SOC_from_cell_SOC,
positive_SOC_from_cell_SOC,
starting_SOC = None,
starting_OCV = None)

Removes the OCV curve from a single cycle.

Parameters

dataset	A Cycling_Information object of the measurement.
parameters	The parameters of the battery as used for the PyBaMM simulations (see models.standard_parameters).
negative_SOC_from_cell_SOC	A function that takes the SOC of the cell and returns the SOC of the negative electrode.
positive_SOC_from_cell_SOC	A function that takes the SOC of the cell and returns the SOC of the positive electrode.
starting_SOC	The SOC at the beginning of the measurement. If not given, the OCV curves will be inverted to determine the initial SOC.
starting_OCV	The OCV at the beginning of the measurement. If not given, the first entry of voltages is used for this.

Returns

2-tuple. First entry are the voltages minus the OCV as estimated for each data point. These are structured in exactly the same way as in the "dataset". Second entry are the electrode SOCs as counted in the data.

5.13.2.17 subtract_OCV_curve_from_cycles()

def ep_bolfi.utility.preprocessing.subtract_OCV_curve_from_cycles (
dataset,
parameters,
starting_SOC = None,
starting_OCV = None,
electrode = "positive",
current_sign = 0,
voltage_sign = 0)

Removes the OCV curve from a cycling measurement.

Parameters

dataset	A Cycling_Information object of the measurement.
parameters	The parameters of the battery as used for the PyBaMM simulations (see models.standard_parameters).
starting_SOC	The SOC at the beginning of the measurement. If not given, the OCV curve will be inverted to determine the initial SOC.
starting_OCV	The OCV at the beginning of the measurement. If not given and starting_SOC is also not given, the first entry of voltages is used for this. If not given, but starting_SOC is, the OCP function will be evaluated at starting_SOC to get the OCV.
electrode	"positive" (default) or "negative" for current sign correction and capacity calculation. "positive" adds SOC with positive current and vice versa. The sign corrections can be overwritten with '∗_sign'.
current_sign	1 adds SOC, -1 subtracts it, 0 follows the default behaviour above.
voltage_sign	1 subtracts the OCP, -1 adds it, 0 follows the default behaviour above.

Returns

2-tuple. First entry are the voltages minus the OCV as estimated for each data point. These are structured in exactly the same way as in the "dataset". Second entry are the electrode SOCs as counted in the data.

5.14 ep_bolfi.utility.visualization Namespace Reference

Various helper and plotting functions for common data visualizations.

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 push_apart_text (fig, ax, text_objects, lock_xaxis=False, temp_path="./temp_render.png")

  Push apart overlapping texts until no overlaps remain.
- 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 nyquist_plot (fig, ax, ω, Z, cmap=plt.get_cmap('tab20b'), ls='-', lw=3, title_text="Impedance Measurement", legend_text="impedance", colorbar_label="Frequency / Hz", add_frequency_colorbar=True, equal_aspect=True)

  Plot an impedance measurement.
- def bode_plot (fig, ax_real, ax_imag, ω, Z, cmap=plt.get_cmap('tab20b'), ls_real='-', ls_imag='-.', lw=3, title_text="Impedance Measurement", legend_text="impedance")

  Plot an impedance measurement.
- def plot_comparison (ax, solutions, errorbars, experiment, solution_visualization=[ ], t_eval=None, title="", xlabel="", ylabel="", feature_visualizer=lambda ∗args:[ ], feature_fontsize=12, interactive_plot=False, output_variables=None, voltage_scale=1.0, use_cycles=False, overpotential=False, three_electrode=None, dimensionless_reference_electrode_location=0.5, parameters=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, 0.7, 0.2, 0.225])

  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=None, 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'), plot_current=True)

  Plots current and voltage curves in one diagram.
- def fit_and_plot_OCV (ax, SOC, OCV, name, phases, SOC_range_bounds=(0.2, 0.8), SOC_range_↩limits=(0.0, 1.0), z=1.0, T=298.15, fit=None, eval_SOC=[0, 1], eval_points=200, spline_SOC_range=(0.01, 0.99), spline_order=2, spline_print=None, parameters_print=False, inverted=True, info_accuracy=True, normalized_xaxis=False, distance_order=2, weights=None, initial_parameters=None, minimize_↩options=None)

  Fits an SOC(OCV)-model and an OCV(SOC)-evaluable spline.
- def visualize_correlation (fig, ax, correlation, names=None, title=None, cmap=plt.get_cmap('BrBG'), entry_color='w')

  Produces a heatmap of a correlation matrix.

5.14.1 Detailed Description

Various helper and plotting functions for common data visualizations.

5.14.2 Function Documentation

5.14.2.1 bode_plot()

def ep_bolfi.utility.visualization.bode_plot (
fig,
ax_real,
ax_imag,
ω,
Z,
cmap = plt.get_cmap('tab20b'),
ls_real = '-',
ls_imag = '-.',
lw = 3,
title_text = "Impedance Measurement",
legend_text = "impedance")

Plot an impedance measurement.

Parameters

fig	The matplotlib.Figure for plotting
ax	The matplotlib.Axes for plotting.
ω	The frequencies at which the impedeance was measured. May be a list of lists for multiple measurements.

Parameters

Z	The impedances that were measured at those frequencies. May be a list of lists for multiple measurements.
cmap	The colormap that is used to differentiate multiple impedances.
ls_real	The linestyle of the plot of the real part of the impedance.
ls_imag	The linestyle of the plot of the imaginary part of the impedance.
lw	The linewidth of the plot.
title_text	The text for the title.
legend_text	The text for the legend. May be a list for multiple measurements.

5.14.2.2 cc_cv_visualization() def ep_bolfi.utility.visualization.cc_cv_visualization (

fig,
ax,
dataset,
max_number_of_clusters = 4,
cmap = plt.get_cmap('tab20c'),
check_location = [0.1, 0.7, 0.2, 0.225])

Automatically labels and displays a CC-CV dataset.

A checkbutton list gets added for browsing through the labels.

Parameters

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.dataset_formatting 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.

5.14.2.3 colorline() def ep_bolfi.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

Parameters

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)".

5.14.2.4 fit_and_plot_OCV() def ep_bolfi.utility.visualization.fit_and_plot_OCV (

ax,
SOC,
OCV,
name,
phases,
SOC_range_bounds = (0.2, 0.8),
SOC_range_limits = (0.0, 1.0),
z = 1.0,
T = 298.15,
fit = None,
eval_SOC = [0, 1],
eval_points = 200,
spline_SOC_range = (0.01, 0.99),
spline_order = 2,
spline_print = None,
parameters_print = False,
inverted = True,
info_accuracy = True,
normalized_xaxis = False,
distance_order = 2,
weights = None,
initial_parameters = None,
minimize_options = None)

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

5.14.3 Exemplary fit parameters:

5.14.4 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]]

5.14.5 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]]

Parameters

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.
SOC_range_bounds	Optional hard upper and lower bounds for the SOC correction from the left and the right side, respectively, as a 2-tuple. Use it as a limiting guess for the actual SOC range represented in the measurement. Has to be inside (0.0, 1.0). Set to (0.0, 1.0) to effectively disable SOC range estimation.
SOC_range_limits	Optional hard lower and upper bounds for the SOC correction from the left and the right side, respectively, as a 2-tuple. Use it if you know that your OCV data is incomplete and by how much. Has to be inside (0.0, 1.0). Set to (0.0, 1.0) to allow the SOC range estimation to assign datapoints to the asymptotes.
z	The charge number of the electrode interface reaction.
T	The temperature of the electrode.
fit	May provide the fit parameters if they are already known.
eval_SOC	Denotes the minimum and maximum SOC to plot the OCV curves at.
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_print	If set to either 'python' or 'matlab', a string representation of the smoothing spline is printed in the respective format.
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.

Parameters

info_accuracy	If True, some measures of fit accuracy are displayed in the figure legend: RMSE (root mean square error), MAE (mean absolute error) and ME (maximum error).
normalized_xaxis	If True, the x-axis gets rescaled to [0,1], where {0,1} matches the asymptotes of the OCV fit function.
distance_order	The argument passed to the numpy.linalg.norm of the vector of distances between OCV data and OCV model. Default is 2, i.e., the Euclidean norm. 1 sets it to absolute distance.
weights	Optional weights to apply to the vector of the distances between OCV data and OCV model. Defaults to equal weights.
initial_parameters	Optional initial guess for the model parameters. If left as-is, this will be automatically gleaned from the data. Use only if you have another fit to data of the same electrode material.
minimize_options	Dictionary that gets passed to scipy.optimize.minimize with the method 'trust-constr'. See scipy.optimize.show_options with the arguments 'minimize' and 'trust-constr' for details.

5.14.5.1 make_segments()

def ep_bolfi.utility.visualization.make_segments (
x,
y)

Create a list of line segments from x and y coordinates.

Parameters

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.

5.14.5.2 nyquist_plot()

def ep_bolfi.utility.visualization.nyquist_plot (
fig,
ax,
ω,
Z,
cmap = plt.get_cmap('tab20b'),
ls = '-',
lw = 3,
title_text = "Impedance Measurement",
legend_text = "impedance",
colorbar_label = "Frequency / Hz",
add_frequency_colorbar = True,
equal_aspect = True)

Plot an impedance measurement.

Parameters

fig	The matplotlib.Figure for plotting
ax	The matplotlib.Axes for plotting.
ω	The frequencies at which the impedeance was measured. May be a list of lists for multiple measurements.
Z	The impedances that were measured at those frequencies. May be a list of lists for multiple measurements.
cmap	The colormap that is used to visualize the frequencies.
ls	The linestyle of the plot.
lw	The linewidth of the plot.
title_text	The text for the title.
legend_text	The text for the legend. May be a list for multiple measurements.
colorbar_label	The label that is displayed next to the colorbar.
add_frequency_colorbar	Set to False if 'fig' was already decorated with a colorbar.
equal_aspect	Set to False in case the impedance is too vertical or horizontal.

Returns

A list of the LineCollection objects of the impedance plots.

5.14.5.3 plot_comparison() def ep_bolfi.utility.visualization.plot_comparison (

ax,
solutions,
errorbars,
experiment,
solution_visualization = [],
t_eval = None,
title = "",
xlabel = "",
ylabel = "",
*feature_visualizer = lambda *args: [],*
feature_fontsize = 12,
interactive_plot = False,
output_variables = None,
voltage_scale = 1.0,
use_cycles = False,
overpotential = False,
three_electrode = None,
dimensionless_reference_electrode_location = 0.5,
parameters = 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.

Parameters

ax	The Axes onto which the comparison shall be plotted.

Parameters

solutions	A dictionary of pybamm.Solution objects. The key goes into the figure legend and the value gets plotted as a line.
errorbars	A dictionary of lists of either pybamm.Solution objects or lists of the desired variable at 't_eval' timepoints. The key goes into the figure legend and the values get plotted as a shaded area between the minimum and maximum.
experiment	A list/tuple of at least length 2. The first two entries are the data timepoints in s and voltages in V. The entries after that are only relevant as optional arguments to ”feature_visualizer”.
solution_visualization	This list/tuple is passed on to ”feature_visualizer” in place of the additional entries of ”experiment” for the visualization of the simulated features.
t_eval	The timepoints at which the ”solutions” and ”errorbars” shall be evaluated in s. 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. Please note that the time will be given in h.
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.
voltage_scale	The plotted voltage gets divided by this value. For example, 1e-3 would produce a plot in [mV]. The voltage given to the ”feature_visualizer” is not affected.
use_cycles	If True, the .cycles property of the ”solutions” is used for the ”feature_visualizer”. Plotting is not affected.
overpotential	If True, only the overpotential of ”solutions” gets plotted. Otherwise, the cell voltage (OCV + overpotential) is plotted.
three_electrode	By default, does nothing (i.e., cell potentials are used). If set to either 'positive' or 'negative', instead of cell potentials, the base for the displayed voltage will be the potential of the 'positive' or 'negative' electrode against a reference electrode. For placement of said reference electrode, please refer to ”dimensionless_reference_electrode_location”.
dimensionless_reference_electrode_location	The location of the reference electrode, given as a scalar between 0 (placed at the point where negative electrode and separator meet) and 1 (placed at the point where positive electrode and separator meet). Defaults to 0.5 (in the middle).
parameters	The parameter dictionary that was used for the simulation. Only needed for a three-electrode output.

Returns

The text objects that were generated according to "feature_visualizer".

5.14.5.4 plot_ICA() def ep_bolfi.utility.visualization.plot_ICA (

ax,
SOC,
OCV,
name,
spline_order = 2,
spline_smoothing = 2e-3,
sign = 1)

Show the derivative of charge by voltage.

Parameters

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.

5.14.5.5 plot_measurement() def ep_bolfi.utility.visualization.plot_measurement (

fig,
ax,
dataset,
title,
cmap = plt.get_cmap('tab20c'),
plot_current = True)

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)".

Returns

The list of text objects for the numbers.

5.14.5.6 plot_OCV_from_CC_CV() def ep_bolfi.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 = None,
parameters_print = False)

Visualizes the OCV_fitting.OCV_from_CC_CV output.

Parameters

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	If set to either 'python' or 'matlab', a string representation of the smoothing spline is printed in the respective format.
parameters_print	Set to True if the fit parameters should be printed to console.

5.14.5.7 push_apart_text() def ep_bolfi.utility.visualization.push_apart_text (

fig,
ax,
text_objects,
lock_xaxis = False,
temp_path = "./temp_render.png")

Push apart overlapping texts until no overlaps remain.

Parameters

fig	The figure which contains the text.
ax	The axis which contains the text.
text_objects	A list of the text objects that shall be pushed apart.
lock_xaxis	If True, texts will only be moved in the y-direction.
temp_path	The path to which a temporary image of the figure ”fig” gets saved. This is necessary to establish the text bbox sizes.

5.14.5.8 set_fontsize() def ep_bolfi.utility.visualization.set_fontsize (

ax,
title = 12,
xaxis = 12,
yaxis = 12,
xticks = 12,
yticks = 12,
legend = 12)

Convenience function for fontsize changes.

Parameters

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.

5.14.5.9 update_legend() def ep_bolfi.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.

Parameters

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.

5.14.5.10 update_limits() def ep_bolfi.utility.visualization.update_limits (

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

Convenience function for adjusting the view.

Parameters

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.

5.14.5.11 visualize_correlation() def ep_bolfi.utility.visualization.visualize_correlation (

fig,
ax,
correlation,
names = None,
title = None,
cmap = plt.get_cmap('BrBG'),
entry_color = 'w')

Produces a heatmap of a correlation matrix.

Parameters

fig	The matplotlib.Figure object for plotting.
ax	The matplotlib.Axes object for plotting.
correlation	A two-dimensional (numpy) array that is the correlation matrix.
names	A list of strings that are names of the variables corresponding to each row or column in the correlation matrix.

Parameters

title	The title of the heatmap.
cmap	The matplotlib colormap for the heatmap.
entry_color	The colour of the correlation matrix entries.

6 Class Documentation

6.1 ep_bolfi.utility.dataset_formatting.Cycling_Information Class Reference

Contains basic cycling informations.

Inheritance diagram for ep_bolfi.utility.dataset_formatting.Cycling_Information:

Collaboration diagram for ep_bolfi.utility.dataset_formatting.Cycling_Information:

Public Member Functions

- def __init__ (self, timepoints, currents, voltages, other_columns={}, indices=None)
- def __len__ (self)
- def to_json (self)
- def from_json (cls, json_string)
- def table_descriptors (self)
- def table_mapping (cls)
- def segment_tables (self, start=0, stop=None, step=1)
- def example_table_row (self)
- 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.
- def extend (self, other)

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.

6.1.1 Detailed Description

Contains basic cycling informations.

Each member variable is a list and has the same length as the other ones.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 __init__()

def ep_bolfi.utility.dataset_formatting.Cycling_Information.__init__ (
self,
timepoints,
currents,
voltages,
other_columns = {},
indices = None)

Reimplemented in ep_bolfi.utility.dataset_formatting.Static_Information.

6.1.3 Member Function Documentation

6.1.3.1 __len__()

def ep_bolfi.utility.dataset_formatting.Cycling_Information.__len__ (
self)

6.1.3.2 example_table_row()

def ep_bolfi.utility.dataset_formatting.Cycling_Information.example_table_row (
self)

Reimplemented from ep_bolfi.utility.dataset_formatting.Measurement.

6.1.3.3 extend()

def ep_bolfi.utility.dataset_formatting.Cycling_Information.extend (
self,
other)

Reimplemented from ep_bolfi.utility.dataset_formatting.Measurement.

Reimplemented in ep_bolfi.utility.dataset_formatting.Static_Information.

6.1.3.4 from_json() def ep_bolfi.utility.dataset_formatting.Cycling_Information.from_json (
cls,
json_string)

Reimplemented from ep_bolfi.utility.dataset_formatting.Measurement.

6.1.3.5 segment_tables() def ep_bolfi.utility.dataset_formatting.Cycling_Information.segment_tables (
self,
start = 0,
stop = None,
step = 1)

Reimplemented from ep_bolfi.utility.dataset_formatting.Measurement.

6.1.3.6 subarray() def ep_bolfi.utility.dataset_formatting.Cycling_Information.subarray (
self,
array)

Selects the data segments with the given indices.

Parameters

array	The indices of the segments that are to be returned.

Returns

A new Cycling_Information object containing the subset.

Reimplemented from ep_bolfi.utility.dataset_formatting.Measurement.

Reimplemented in ep_bolfi.utility.dataset_formatting.Static_Information.

6.1.3.7 subslice() def ep_bolfi.utility.dataset_formatting.Cycling_Information.subslice (
self,
start,
stop,
step = 1)

Selects a subslice of the data segments.

The arguments exactly match the slice(...) notation.

Parameters

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.

Reimplemented from ep_bolfi.utility.dataset_formatting.Measurement.

Reimplemented in ep_bolfi.utility.dataset_formatting.Static_Information.

6.1.3.8 table_descriptors() def ep_bolfi.utility.dataset_formatting.Cycling_Information.table_descriptors (*self*)

Reimplemented from ep_bolfi.utility.dataset_formatting.Measurement.

6.1.3.9 table_mapping() def ep_bolfi.utility.dataset_formatting.Cycling_Information.table_mapping (*cls*)

Reimplemented from ep_bolfi.utility.dataset_formatting.Measurement.

6.1.3.10 to_json() def ep_bolfi.utility.dataset_formatting.Cycling_Information.to_json (*self*)

Reimplemented from ep_bolfi.utility.dataset_formatting.Measurement.

Reimplemented in ep_bolfi.utility.dataset_formatting.Static_Information.

6.1.4 Member Data Documentation

6.1.4.1 currents ep_bolfi.utility.dataset_formatting.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.

6.1.4.2 indices ep_bolfi.utility.dataset_formatting.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 colourcoding the segments by cycle.

6.1.4.3 other_columns ep_bolfi.utility.dataset_formatting.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.

6.1.4.4 timepoints ep_bolfi.utility.dataset_formatting.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.

6.1.4.5 voltages ep_bolfi.utility.dataset_formatting.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/dataset_formatting.py

6.2 ep_bolfi.models.assess_effective_parameters.Effective_Parameters Class Reference

Calculates, stores and prints effective parameters.

Public Member Functions

- def __init__ (self, parameters, working_electrode='both')

    Preprocesses the model parameters.
- def eval (self, expression)

    Short-hand for PyBaMM symbol evaluation.
- def print (self, c_rates=[0.1, 0.2, 0.5, 1.0])

    Prints the voltage losses for the given C-rates.

Public Attributes

- parameters

    The parameter dictionary converted to PyBaMM form.
- working_electrode

    Sets whether full- or half-cell parameters are calculated.
- Q_n

    Negative electrode theoretical capacity.
- Q_p

    Positive electrode theoretical capacity.
- R_c_e_0_1

    Integration constant for the electrolyte concentration / I.
- R_bar_c_{en}_1

    Electrolyte concentration / I at the negative electrode.
- R_bar_c_{ep}_1

    Electrolyte concentration / I at the positive electrode.
- R_bar_φ_{sn}_1

    Effective resistance of the negative electrode.
- R_bar_φ_{sp}_1

    Effective resistance of the positive electrode.
- R_{ns}

    Negative electrode resistance (as calculated by the SPMe(S)).
- R_{ps}

    Positive electrode resistance (as calculated by the SPMe(S)).
- R_e

    Electrolyte resistance (as calculated by the SPMe(S)).
- i_{sen}

    SPM(e) negative electrode exchange-current.
- i_{sep}

    SPM(e) positive electrode exchange-current.

6.2.1 Detailed Description

Calculates, stores and prints effective parameters.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 __init__()

def ep_bolfi.models.assess_effective_parameters.Effective_Parameters.__init__ (
self,
parameters,
working_electrode = 'both')

Preprocesses the model parameters.

Parameters

parameters	A dictionary of parameter values with the parameter names as keys. For these names, please refer to ep_bolfi.models.standard_parameters.
working_electrode	When set to either 'negative' or 'positive', the parameters will be treated as a half-cell setup with said electrode.

6.2.3 Member Function Documentation

6.2.3.1 eval() def ep_bolfi.models.assess_effective_parameters.Effective_Parameters.eval (
self,
expression)

Short-hand for PyBaMM symbol evaluation.

Parameters

expression	A pybamm.Symbol.

Returns

The numeric value of ”expression”.

6.2.3.2 print() def ep_bolfi.models.assess_effective_parameters.Effective_Parameters.print (
self,
c_rates = [0.1, 0.2, 0.5, 1.0])

Prints the voltage losses for the given C-rates.

Parameters

c_rates	The C-rates (as fraction of ”Typical current [A]”).

6.2.4 Member Data Documentation

6.2.4.1 $\mathbf{i}_{sen}$ ep_bolfi.models.assess_effective_parameters.Effective_Parameters.i_{sen}

SPM(e) negative electrode exchange-current.

6.2.4.2 $\mathbf{i}_{sep}$ ep_bolfi.models.assess_effective_parameters.Effective_Parameters.i_{sep}

SPM(e) positive electrode exchange-current.

6.2.4.3 parameters ep_bolfi.models.assess_effective_parameters.Effective_Parameters.parameters

The parameter dictionary converted to PyBaMM form.

Convert SubstitutionDict to dict by iterating over it.

6.2.4.4 Q_n ep_bolfi.models.assess_effective_parameters.Effective_Parameters.Q_n

Negative electrode theoretical capacity.

6.2.4.5 Q_p ep_bolfi.models.assess_effective_parameters.Effective_Parameters.Q_p

Positive electrode theoretical capacity.

6.2.4.6 R_bar_c_{en}_1 ep_bolfi.models.assess_effective_parameters.Effective_Parameters.R_bar_c_{en}_1

Electrolyte concentration / I at the negative electrode.

6.2.4.7 R_bar_c_{ep}_1 ep_bolfi.models.assess_effective_parameters.Effective_Parameters.R_bar_c_{ep}_1

Electrolyte concentration / I at the positive electrode.

6.2.4.8 R_bar_φ_{sn}_1 ep_bolfi.models.assess_effective_parameters.Effective_Parameters.R_bar_φ_{sn}_1

Effective resistance of the negative electrode.

6.2.4.9 R_bar_φ_{sp}_1 ep_bolfi.models.assess_effective_parameters.Effective_Parameters.R_bar_φ_{sp}_1

Effective resistance of the positive electrode.

6.2.4.10 R_c_e_0_1 ep_bolfi.models.assess_effective_parameters.Effective_Parameters.R_c_e_0_1

Integration constant for the electrolyte concentration / I.

6.2.4.11 R_e ep_bolfi.models.assess_effective_parameters.Effective_Parameters.R_e

Electrolyte resistance (as calculated by the SPMe(S)).

6.2.4.12 R_{ns} ep_bolfi.models.assess_effective_parameters.Effective_Parameters.R_{ns}

Negative electrode resistance (as calculated by the SPMe(S)).

6.2.4.13 R_{ps} ep_bolfi.models.assess_effective_parameters.Effective_Parameters.R_{ps}

Positive electrode resistance (as calculated by the SPMe(S)).

6.2.4.14 working_electrode ep_bolfi.models.assess_effective_parameters.Effective_Parameters.working_electrode

Sets whether full- or half-cell parameters are calculated.

The documentation for this class was generated from the following file:

- models/assess_effective_parameters.py

6.3 ep_bolfi.models.electrolyte.Electrolyte Class Reference

Electrolyte model assuming a symmetric Li-metal cell.

Inheritance diagram for ep_bolfi.models.electrolyte.Electrolyte:

6.4 ep_bolfi.models.electrolyte.Electrolyte_internal Class Reference

Defining equations for a symmetric Li cell with electrolyte.

Inheritance diagram for ep_bolfi.models.electrolyte.Electrolyte_internal:

Collaboration diagram for ep_bolfi.models.electrolyte.Electrolyte_internal:

Public Member Functions

- def __init__ (self, param, pybamm_control=False, options={}, build=True)

  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

- pybamm_control

  Current is fixed if False and a variable if True.

6.4.1 Detailed Description

Defining equations for a symmetric Li cell with electrolyte.

6.4.1.1 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 ep_bolfi.models.electrolyte.Electrolyte_internal.__init__ (
self,
param,
pybamm_control = False,
options = {},
build = True)

Sets the model properties.

Parameters

param	A class containing all the relevant parameters for this model. For example, models.standard_parameters represents a valid choice for this parameter.
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.
options	Not used; only here for compatibility with the base class.
build	Not used; only here for compatibility with the base class.

6.4.3 Member Function Documentation

6.4.3.1 get_coupled_variables() def ep_bolfi.models.electrolyte.Electrolyte_internal.get_coupled_variables (
self,
variables)

Builds all model symbols that rely on other models.

Parameters

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 ep_bolfi.models.electrolyte.Electrolyte_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 ep_bolfi.models.electrolyte.Electrolyte_internal.set_algebraic (
self,
variables)

Sets up the algebraic equations in self.algebraic.

6.4.3.4 set_boundary_conditions() def ep_bolfi.models.electrolyte.Electrolyte_internal.set_boundary_conditions (
self,
variables)

Sets the (self.)boundary(_)conditions.

6.4.3.5 set_events() def ep_bolfi.models.electrolyte.Electrolyte_internal.set_events (
self,
variables)

Sets up the termination switches in self.events.

6.4.3.6 set_initial_conditions() def ep_bolfi.models.electrolyte.Electrolyte_internal.set_initial_conditions (
self,
variables)

Sets the (self.)initial(_)conditions.

6.4.3.7 set_rhs() def ep_bolfi.models.electrolyte.Electrolyte_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 pybamm_control ep_bolfi.models.electrolyte.Electrolyte_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/electrolyte.py

6.5 ep_bolfi.optimization.EP_BOLFI.EP_BOLFI Class Reference

Expectation Propagation and Bayesian Optimization.

Public Member Functions

- def __init__ (self, simulators, experimental_datasets, feature_extractors, fixed_parameters, free_parameters=None, initial_covariance=None, free_parameters_boundaries=None, boundaries_in_deviations=0, Q=None, r=None, Q_features=None, r_features=None, transform_parameters={}, weights=None, display_current_feature=None, fixed_parameter_order=None)
- def result_to_json (self, seed=None)
- def log_to_json (self)
- def visualize_parameter_distribution (self)

  Plots the features and visualizes the correlation.
- def run (self, bolfi_initial_evidence=None, bolfi_total_evidence=None, bolfi_posterior_samples=None, ep_iterations=3, ep_dampener=None, final_dampening=None, ep_dampener_reduction_steps=-1, gelman_rubin_threshold=None, ess_ratio_resample=5.0, ess_ratio_sampling_from_zero=-1.0, ess_ratio↩_abort=20.0, max_heuristic_steps=10, posterior_sampling_increase=1.2, model_resampling_increase=1.↩1, independent_mcmc_chains=4, scramble_ep_feature_order=True, show_trials=False, verbose=True, seed=None)

  Runs Expectation Propagation together with BOLFI.

Public Attributes

- simulators
- experimental_datasets
- feature_extractors
- fixed_parameters
- free_parameters
- initial_covariance
- free_parameters_boundaries
- boundaries_in_deviations

  Uses boundaries as multiples of the standard deviation.
- transform_parameters
- weights

  Set the weights to unity if None are given.
- display_current_feature
- fixed_parameter_order

  If the parameter order is explicitly given, use that instead.
- log_of_tried_parameters

  Stores all parameter combinations that have been tried.
- experimental_features

  Experimental features.
- input_dim

  Input dimension of the estimation task.
- output_dim

  Output dimension of the estimation task (sum of features).
- simulator_index_by_feature

  Mapping of index by all features to corresponding simulator.
- sub_index_by_feature

  Mapping of index by all features to that by one set of them.
- initial_guesses

  Container for the initial expectation values.
- log_of_raw_tried_parameters

  Stores all raw parameter evaluation points.
- log_of_discrepancies

  Stores all discrepancies of the sampled parameters.
- final_expectation

  Stores the inference mean (empty at first).
- final_covariance

  Stores the inference covariance (empty at first).
- final_correlation

  Stores the inference correlation (empty at first).
- initial_Q

  Expectation Propagation covariance matrix (prior).
- Q

  Expectation Propagation covariance matrix (posterior).
- initial_r

  Expectation Propagation expectation value (prior).
- r

  Expectation Propagation expectation value (posterior).
- Q_features

  Expectation Propagation itemized covariance matrices.
- r_features

Expectation Propagation itemized expectation values.

- inferred_parameters

  The inferred model parameters.
- final_error_bounds

  The 95% confidence bounds (which don't reflect the cross-correlations, but are easier to interpret).

6.5.1 Detailed Description

Expectation Propagation and Bayesian Optimization.

Sets up and runs these two algorithms to infer model parameters. Use the variables ”Q”, ”r”, ”Q_features” and ”r_features” to copy the state of another estimator. Do not use them in any other case. Always use either all of them or none of them.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 __init__()

```
def ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.__init__ (
        self,
        simulators,
        experimental_datasets,
        feature_extractors,
        fixed_parameters,
        free_parameters = None,
        initial_covariance = None,
        free_parameters_boundaries = None,
        boundaries_in_deviations = 0,
        Q = None,
        r = None,
        Q_features = None,
        r_features = None,
        transform_parameters = {},
        weights = None,
        display_current_feature = None,
        fixed_parameter_order = None )
```

Parameters

simulators	A list of functions that take one argument: a dictionary of combined 'fixed_parameters' and 'free_parameters'. They return the simulated counterpart to the experimental data. Most of the time, one function will be sufficient. Additional functions may be used to combine simulators which each give a subset of the total experimental method.
experimental_datasets	A list of the experimental data. Each entry corresponds to the simulator in 'simulators' with the same index and has the same structure as its output.
feature_extractors	A list of functions which each take the corresponding data entry and return a list of numbers, which represent its reduced features.
fixed_parameters	Dictionary of parameters that stay fixed and their values.

Parameters

free_parameters	Dictionary of parameters which shall be inferred and their initial guesses or, more accurately, their expected values. Please note that these values live in the transformed space. Optionally, the values may be a 2-tuple where the second entry would be the variance of that parameter. For finer tuning with covariances, use 'initial_covariance' (will take precedence). Alternatively, you may set 'free_parameters_boundaries' to set the expected values and variances by confidence intervals.
initial_covariance	Initial covariance of the parameters. Has to be a symmetric matrix (list of list or numpy 2D array). 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. If the diagonal entries are $σ_i^2$ and the bounds are symmetric, the probability distribution of each parameter is 95% within these bounds. Please note that the same does not hold for the whole probability distribution.
free_parameters_boundaries	Optional hard boundaries of the space in which optimal parameters are searched for. They are given as a dictionary with values as 2-tuples with the left and right boundaries. Boundaries need to be given for either none or all parameters. If None are given, boundaries will be set by 'boundaries_in_deviations' relative to the covariance. The default then is the 95 % confidence ellipsoid. If neither 'initial_covariance' nor 'free_parameters' set the covariance, this parameter sets it according to the example in the description of 'initial_covariance'.
boundaries_in_deviations	When <= 0, the boundaries are set as described above. When > 0, the boundaries are this multiple of the standard deviation. This scales with the shrinking covariance as the algorithm progresses. 'free_parameters_boundaries' takes precedence, i.e., the covariance gets set and then the boundaries in the optimization are in standard deviations.
Q	If you want to restore a previous EP-BOLFI instance from a dump of its data (see "result_to_json" method), put the "Q" attribute stored therein into this parameter.
r	Same as Q, but use the "r" attribute.
Q_features	Same as Q, but use the "Q_features" attribute.
r_features	Same as Q, but use the "r_features" attribute.
transform_parameters	Optional transformations between the parameter space that is used for searching for optimal parameters and the model parameters. Any missing free parameter is not transformed. The values are 2-tuples. The first entry is a function taking the search space parameter and returning the model parameter. The second entry is the inverse function. For convenience, any value may also be one of the following: • 'none' => (identity, identity) • 'log' => (exp, log) Please note that, for performance reasons, the returned inferred values are directly back-transformed from the mean of the internal standard distribution. This means that they represent the median of the actual distribution.
weights	Optional weights to assign to the features. Higher weights give more importance to a feature and vice versa. A list of lists which correspond to the 'feature_extractors'.
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 output.

Parameters

fixed_parameter_order	Establish a numerical order to the parameter names. This prevents errors arising from internal reordering of the dictionaries. Only necessary when using the same model in different contexts.

6.5.3 Member Function Documentation

6.5.3.1 log_to_json()

def ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.log_to_json (
self)

6.5.3.2 result_to_json()

def ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.result_to_json (
self,
seed = None)

6.5.3.3 run()

def ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.run (
self,
bolfi_initial_evidence = None,
bolfi_total_evidence = None,
bolfi_posterior_samples = None,
ep_iterations = 3,
ep_dampener = None,
final_dampening = None,
ep_dampener_reduction_steps = -1,
gelman_rubin_threshold = None,
ess_ratio_resample = 5.0,
ess_ratio_sampling_from_zero = -1.0,
ess_ratio_abort = 20.0,
max_heuristic_steps = 10,
posterior_sampling_increase = 1.2,
model_resampling_increase = 1.1,
independent_mcmc_chains = 4,
scramble_ep_feature_order = True,
show_trials = False,
verbose = True,
seed = None)

Runs Expectation Propagation together with BOLFI.

This function can be called multiple times; the estimation will take off from where it last stopped.

Parameters

bolfi_initial_evidence	Number of evidence samples BOLFI will take for each feature before using Bayesian Optimization sampling. Default: 1 + 2 ^ "number of estimated parameters".

Parameters

bolfi_total_evidence	Number of evidence samples BOLFI will take for each feature in total (including initial evidence). Default: 2 ∗ ”bolfi_initial_evidence”.
bolfi_posterior_samples	Effective number of samples BOLFI will take from the posterior distribution. These are then used to fit a Gaussian to the posterior. Fit convergence scales with $1/\sqrt{n}$. Default: $I^2 + 3 * I$ with I as the number of estimated parameters. This is the number of the metaparameters of the underlying probability distribution times 2. The ”times 2” considers the warmup samples.
ep_iterations	The number of iterations of the Expectation Propagation algorithm, i.e., the number of passes over each feature. Default: 3.
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. For values up to 1, that fraction of the pseudo-prior remains in each site update. Default: with ”a” as the number of features and ”b” as ”ep_iterations”, $1 - a * (1 - \sqrt[ab]{\text{”final\_dampening”}})$.
final_dampening	Alternative way to set ”ep_dampener”. 0 means no dampening. For values up to 1, that fraction of the prior remains after the whole estimation. Default: if ”ep_dampener” is not set, 0.5. Else, ”ep_dampener” takes precedence.
ep_dampener_reduction_steps	Number of iterations over which the 'ep_dampener' gets reduced to 0. In each iteration, an equal fraction of it gets subtracted. Set to a negative number to disable the reduction. Default: -1.
gelman_rubin_threshold	Optional threshold on top of the effective sample size. Values close to one indicate a converged estimate of the pseudo-posteriors. Never set to exactly one.
ess_ratio_sampling_from_zero	Threshold in the ratio of effective sample size to samples in the pseudo-posterior estimation, at which the sampling defaults to starting at the center of the pseudo-prior. Set higher than ”ess_ratio_resample” to disable this behaviour.
ess_ratio_resample	Threshold in the ratio of effective sample size to samples in the pseudo-posterior estimation, at which before sampling the model gets resampled. Set higher than ”ess_ratio_abort” to disable this behaviour.
ess_ratio_abort	Threshold in the ratio of effective sample size to samples in the pseudo-posterior estimation, at which the sampling aborts and the pseudo-posterior update is skipped.
max_heuristic_steps	The heuristics that are set by the ”ess_ratio_∗” arguments could effectively run forever. This parameter limits the amount of times these heuristics get employed in on EP iteration before it terminates.
posterior_sampling_increase	The factor by which the ratio of the effective sample size to samples in the pseudo-posterior estimation is multiplied each loop (cumulatively). Never set to exactly one or lower, as it might result in an infinite loop.
model_resampling_increase	The factor by which 'bolfi_total_evidence' gets multiplied each time the model gets resampled.
independent_mcmc_chains	The number of independent Markov-Chain Monte Carlo chains that are used for the estimation of the pseudo-posterior. Since we did not implement parallelization, more chains will not be faster, but more stable.
scramble_ep_feature_order	True randomizes the order that the EP features are iterated over. Their order is still consistent across EP iterations. False uses the order that the ”feature_extractors” define.
show_trials	True plots the log of tried parameters live. Please note that each plot blocks the execution of the program, so do not use this when running the estimation in the background.

Parameters

verbose	True shows verbose error messages and logs of the estimation process. With False, you need to get the estimation results from self.final_expectation and self.final_covariance. Default: True.
seed	Optional seed that is used in the RNG. If None is given, the results will be slightly different each time.

Returns

The BOLFI instance of the last EP iteration. As such, it contains the Posterior of the overall inference procedure.

6.5.3.4 visualize_parameter_distribution() def ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.visualize_parameter_←↩ distribution (*self*)

Plots the features and visualizes the correlation.

Please note that this function requires that the output of the individual simulators and the individual experimental data give an x- and y-axis when indexed with [0] and [1], respectively. Lists of lists in [0] and [1] with segmented data works as well. EP_BOLFI.run() does not have these restrictions. Visualizes the comparison that EP_BOLFI was set up to infer model parameters with. May be used to check if everything works as intended. Additionally, the 95% confidence error bounds for the parameters are visualized to check for reasonable bounds. If called after 'run()', the expected parameter set, the correlation and error bounds are from the finished estimation.

6.5.4 Member Data Documentation

6.5.4.1 boundaries_in_deviations ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.boundaries_in_deviations

Uses boundaries as multiples of the standard deviation.

6.5.4.2 display_current_feature ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.display_current_feature

6.5.4.3 experimental_datasets ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.experimental_datasets

6.5.4.4 experimental_features ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.experimental_features

Experimental features.

6.5.4.5 feature_extractors ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.feature_extractors

6.5.4.6 final_correlation ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.final_correlation

Stores the inference correlation (empty at first).

6.5.4.7 final_covariance ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.final_covariance

Stores the inference covariance (empty at first).

6.5.4.8 final_error_bounds ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.final_error_bounds

The 95% confidence bounds (which don't reflect the cross-correlations, but are easier to interpret).

6.5.4.9 final_expectation ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.final_expectation

Stores the inference mean (empty at first).

6.5.4.10 fixed_parameter_order ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.fixed_parameter_order

If the parameter order is explicitly given, use that instead.

6.5.4.11 fixed_parameters ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.fixed_parameters

6.5.4.12 free_parameters ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.free_parameters

6.5.4.13 free_parameters_boundaries ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.free_parameters_boundaries

6.5.4.14 inferred_parameters ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.inferred_parameters

The inferred model parameters.

6.5.4.15 initial_covariance ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.initial_covariance

6.5.4.16 initial_guesses ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.initial_guesses

Container for the initial expectation values.

6.5.4.17 initial_Q ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.initial_Q

Expectation Propagation covariance matrix (prior).

6.5.4.18 initial_r ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.initial_r

Expectation Propagation expectation value (prior).

6.5.4.19 input_dim ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.input_dim

Input dimension of the estimation task.

6.5.4.20 log_of_discrepancies ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.log_of_discrepancies

Stores all discrepancies of the sampled parameters.

6.5.4.21 log_of_raw_tried_parameters ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.log_of_raw_tried_parameters

Stores all raw parameter evaluation points.

6.5.4.22 log_of_tried_parameters ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.log_of_tried_parameters

Stores all parameter combinations that have been tried.

6.5.4.23 output_dim ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.output_dim

Output dimension of the estimation task (sum of features).

6.5.4.24 Q ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.Q

Expectation Propagation covariance matrix (posterior).

6.5.4.25 Q_features ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.Q_features

Expectation Propagation itemized covariance matrices.

6.5.4.26 r ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.r

Expectation Propagation expectation value (posterior).

6.5.4.27 r_features ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.r_features

Expectation Propagation itemized expectation values.

6.5.4.28 simulator_index_by_feature ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.simulator_index_by_feature

Mapping of index by all features to corresponding simulator.

6.5.4.29 simulators ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.simulators

6.5.4.30 sub_index_by_feature ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.sub_index_by_feature

Mapping of index by all features to that by one set of them.

6.5.4.31 transform_parameters ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.transform_parameters

6.5.4.32 weights ep_bolfi.optimization.EP_BOLFI.EP_BOLFI.weights

Set the weights to unity if None are given.

The documentation for this class was generated from the following file:

- optimization/EP_BOLFI.py

6.6 ep_bolfi.utility.dataset_formatting.Impedance_Measurement Class Reference

Contains basic impedance data.

Inheritance diagram for ep_bolfi.utility.dataset_formatting.Impedance_Measurement:

Collaboration diagram for ep_bolfi.utility.dataset_formatting.Impedance_Measurement:

Public Member Functions

- def __init__ (self, frequencies, real_impedances, imaginary_impedances, phases, other_columns={}, indices=None)
- def __len__ (self)
- def complex_impedances (self)
- def to_json (self)
- def from_json (cls, json_string)
- def table_descriptors (self)
- def table_mapping (cls)
- def segment_tables (self, start=0, stop=None, step=1)
- def example_table_row (self)
- 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.
- def extend (self, other)

Public Attributes

- frequencies

  The frequencies at which impedances were measured.
- real_impedances

  The real part of the impedances measured at those frequencies.
- imaginary_impedances

  The imaginary part of the impedances measured at those frequencies.
- phases

  The phases of the impedance measured at those frequencies.
- other_columns

  The contents of any other columns.
- indices

  The indices of the individual segments.

6.6.1 Detailed Description

Contains basic impedance data.

Each member variable is a list and has the same length as the other ones.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 __init__()

def ep_bolfi.utility.dataset_formatting.Impedance_Measurement.__init__ (
self,
frequencies,
real_impedances,
imaginary_impedances,
phases,
other_columns = {},
indices = None)

6.6.3 Member Function Documentation

6.6.3.1 __len__()

def ep_bolfi.utility.dataset_formatting.Impedance_Measurement.__len__ (
self)

6.6.3.2 complex_impedances()

def ep_bolfi.utility.dataset_formatting.Impedance_Measurement.complex_impedances (
self)

6.6.3.3 example_table_row()

def ep_bolfi.utility.dataset_formatting.Impedance_Measurement.example_table_row (
self)

Reimplemented from ep_bolfi.utility.dataset_formatting.Measurement.

6.6.3.4 extend()

def ep_bolfi.utility.dataset_formatting.Impedance_Measurement.extend (
self,
other)

Reimplemented from ep_bolfi.utility.dataset_formatting.Measurement.

6.6.3.5 from_json() def ep_bolfi.utility.dataset_formatting.Impedance_Measurement.from_json (
cls,
json_string)

Reimplemented from ep_bolfi.utility.dataset_formatting.Measurement.

6.6.3.6 segment_tables() def ep_bolfi.utility.dataset_formatting.Impedance_Measurement.segment_tables (
self,
start = 0,
stop = None,
step = 1)

Reimplemented from ep_bolfi.utility.dataset_formatting.Measurement.

6.6.3.7 subarray() def ep_bolfi.utility.dataset_formatting.Impedance_Measurement.subarray (
self,
array)

Selects the data segments with the given indices.

Parameters

array	The indices of the segments that are to be returned.

Returns

A new Impedance_Measurement object containing the subset.

Reimplemented from ep_bolfi.utility.dataset_formatting.Measurement.

6.6.3.8 subslice() def ep_bolfi.utility.dataset_formatting.Impedance_Measurement.subslice (
self,
start,
stop,
step = 1)

Selects a subslice of the data segments.

The arguments exactly match the slice(...) notation.

Parameters

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 Impedance_Measurement object containing the slice.

Reimplemented from ep_bolfi.utility.dataset_formatting.Measurement.

6.6.3.9 table_descriptors() def ep_bolfi.utility.dataset_formatting.Impedance_Measurement.table_descriptors (*self*)

Reimplemented from ep_bolfi.utility.dataset_formatting.Measurement.

6.6.3.10 table_mapping() def ep_bolfi.utility.dataset_formatting.Impedance_Measurement.table_mapping (*cls*)

Reimplemented from ep_bolfi.utility.dataset_formatting.Measurement.

6.6.3.11 to_json() def ep_bolfi.utility.dataset_formatting.Impedance_Measurement.to_json (*self*)

Reimplemented from ep_bolfi.utility.dataset_formatting.Measurement.

6.6.4 Member Data Documentation

6.6.4.1 frequencies ep_bolfi.utility.dataset_formatting.Impedance_Measurement.frequencies

The frequencies at which impedances were measured.

Usually a list of lists where each list corresponds to a different equilibrium.

6.6.4.2 imaginary_impedances ep_bolfi.utility.dataset_formatting.Impedance_Measurement.imaginary_impedances

The imaginary part of the impedances measured at those frequencies.

6.6.4.3 indices ep_bolfi.utility.dataset_formatting.Impedance_Measurement.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 colourcoding the segments by cycle.

6.6.4.4 other_columns ep_bolfi.utility.dataset_formatting.Impedance_Measurement.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.

6.6.4.5 phases ep_bolfi.utility.dataset_formatting.Impedance_Measurement.phases

The phases of the impedance measured at those frequencies.

6.6.4.6 real_impedances ep_bolfi.utility.dataset_formatting.Impedance_Measurement.real_impedances

The real part of the impedances measured at those frequencies.

The documentation for this class was generated from the following file:

- utility/dataset_formatting.py

6.7 ep_bolfi.utility.dataset_formatting.Measurement Class Reference

Defines common methods for measurement objects.

Inheritance diagram for ep_bolfi.utility.dataset_formatting.Measurement:

Public Member Functions

- def to_json (self)
- def from_json (cls, json_string)
- def table_descriptors (self)
- def table_mapping (cls)
- def segment_tables (self, start=0, stop=None, step=1)
- def example_table_row (self)
- def subslice (self, start, stop, step=1)
- def subarray (self, array)
- def extend (self, other)

6.7.1 Detailed Description

Defines common methods for measurement objects.

6.7.2 Member Function Documentation

6.7.2.1 example_table_row() def ep_bolfi.utility.dataset_formatting.Measurement.example_table_row (
self)

Reimplemented in ep_bolfi.utility.dataset_formatting.Impedance_Measurement, and ep_bolfi.utility.dataset_formatting.Cycling_In

6.7.2.2 extend() def ep_bolfi.utility.dataset_formatting.Measurement.extend (
self,
other)

Reimplemented in ep_bolfi.utility.dataset_formatting.Impedance_Measurement, ep_bolfi.utility.dataset_formatting.Static_Informa
and ep_bolfi.utility.dataset_formatting.Cycling_Information.

6.7.2.3 from_json() def ep_bolfi.utility.dataset_formatting.Measurement.from_json (
cls,
json_string)

Reimplemented in ep_bolfi.utility.dataset_formatting.Impedance_Measurement, and ep_bolfi.utility.dataset_formatting.Cycling_In

6.7.2.4 segment_tables() def ep_bolfi.utility.dataset_formatting.Measurement.segment_tables (
self,
start = 0,
stop = None,
step = 1)

Reimplemented in ep_bolfi.utility.dataset_formatting.Impedance_Measurement, and ep_bolfi.utility.dataset_formatting.Cycling_In

6.7.2.5 subarray() def ep_bolfi.utility.dataset_formatting.Measurement.subarray (
self,
array)

Reimplemented in ep_bolfi.utility.dataset_formatting.Impedance_Measurement, ep_bolfi.utility.dataset_formatting.Static_Informa
and ep_bolfi.utility.dataset_formatting.Cycling_Information.

6.7.2.6 subslice() def ep_bolfi.utility.dataset_formatting.Measurement.subslice (
self,
start,
stop,
step = 1)

Reimplemented in ep_bolfi.utility.dataset_formatting.Impedance_Measurement, ep_bolfi.utility.dataset_formatting.Static_Informa
and ep_bolfi.utility.dataset_formatting.Cycling_Information.

6.7.2.7 table_descriptors()
def ep_bolfi.utility.dataset_formatting.Measurement.table_descriptors (
self)

Reimplemented in ep_bolfi.utility.dataset_formatting.Impedance_Measurement, and ep_bolfi.utility.dataset_formatting.Cycling_In

6.7.2.8 table_mapping()
def ep_bolfi.utility.dataset_formatting.Measurement.table_mapping (
cls)

Reimplemented in ep_bolfi.utility.dataset_formatting.Impedance_Measurement, and ep_bolfi.utility.dataset_formatting.Cycling_In

6.7.2.9 to_json()
def ep_bolfi.utility.dataset_formatting.Measurement.to_json (
self)

Reimplemented in ep_bolfi.utility.dataset_formatting.Impedance_Measurement, ep_bolfi.utility.dataset_formatting.Static_Informa
and ep_bolfi.utility.dataset_formatting.Cycling_Information.

The documentation for this class was generated from the following file:

- utility/dataset_formatting.py

6.8 ep_bolfi.optimization.EP_BOLFI.NDArrayEncoder Class Reference

Inheritance diagram for ep_bolfi.optimization.EP_BOLFI.NDArrayEncoder:

Collaboration diagram for ep_bolfi.optimization.EP_BOLFI.NDArrayEncoder:

Public Member Functions

- def default (self, item)

6.8.1 Member Function Documentation

6.8.1.1 default()
def ep_bolfi.optimization.EP_BOLFI.NDArrayEncoder.default (
self,
item)

The documentation for this class was generated from the following file:

- optimization/EP_BOLFI.py

6.9 ep_bolfi.utility.fitting_functions.NDArrayEncoder Class Reference

Inheritance diagram for ep_bolfi.utility.fitting_functions.NDArrayEncoder:

Collaboration diagram for ep_bolfi.utility.fitting_functions.NDArrayEncoder:

Public Member Functions

- def default (self, item)

6.9.1 Member Function Documentation

6.9.1.1 default() def ep_bolfi.utility.fitting_functions.NDArrayEncoder.default (

self,
item)

The documentation for this class was generated from the following file:

- utility/fitting_functions.py

6.10 ep_bolfi.utility.fitting_functions.OCV_fit_result Class Reference

Contains OCV fit parameters and related information.

Inheritance diagram for ep_bolfi.utility.fitting_functions.OCV_fit_result:

Collaboration diagram for ep_bolfi.utility.fitting_functions.OCV_fit_result:

Public Member Functions

- def __init__ (self, fit, SOC, OCV, SOC_offset=1.0, SOC_scale=1.0, optimize_result=None, spline_interpolation_knots=None, spline_interpolation_coefficients=None, function_string=None)
- def to_json (self)
- def SOC_adjusted (self, soc=None)

  Gives the adjusted SOC values.
- def SOC_other_electrode (self, soc=None)

  Relates the SOCs of the two electrodes to each other.

Public Attributes

- SOC_range

  The SOC range of the data.
- fit

  The fit parameters of the OCV function from Birkl.
- E_0

  The E_0 (plateau voltages) parameters.
- a

  The a (inverse plateau widths) parameters.
- Δx

  The Δx (phase proportion) parameters.
- SOC

  The SOC data points.
- OCV

  The OCV data points.
- SOC_offset

  If another electrode was factored out in the data, this may contain its SOC at SOC 0 of the electrode of interest.
- SOC_scale

  If another electrode was factored out in the data, this may contain the the rate of change of its SOC to that of the electrode of interest.
- optimize_result

  The scipy.optimize.OptimizeResult that led to the fit.
- spline_interpolation_knots

  The knots of the interpolating spline fitted to the inverse.
- spline_interpolation_coefficients

  The coefficients of the interpolating spline fitted to the inverse.
- function_string

  The string representation of the interpolating spline fitted to the inverse.

6.10.1 Detailed Description

Contains OCV fit parameters and related information.

6.10.1.1 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

6.10.2 Constructor & Destructor Documentation

6.10.2.1 __init__()

```
def ep_bolfi.utility.fitting_functions.OCV_fit_result.__init__ (
        self,
        fit,
        SOC,
        OCV,
        SOC_offset = 1.0,
        SOC_scale = 1.0,
        optimize_result = None,
        spline_interpolation_knots = None,
        spline_interpolation_coefficients = None,
        function_string = None )
```

6.10.3 Member Function Documentation

6.10.3.1 SOC_adjusted() def ep_bolfi.utility.fitting_functions.OCV_fit_result.SOC_adjusted (
self,
soc = None)

Gives the adjusted SOC values.

Parameters

soc	The SOC as assigned in the original data. This usually corresponds to the range available during a measurement.

Returns

The SOC as corrected by the OCV model. These values will try to correspond to the level of lithiation.

6.10.3.2 SOC_other_electrode() def ep_bolfi.utility.fitting_functions.OCV_fit_result.SOC_other_electrode (
self,
soc = None)

Relates the SOCs of the two electrodes to each other.

If the original data was of a full cell and the other electrode was factored out, this may contain the function that takes the SOC of the electrode of interest and gives the SOC of the other electrode, i.e., the stoichiometric relation.

Parameters

soc	The SOC of the electrode of interest.

Returns

The SOC of the other electrode that was factored out.

6.10.3.3 to_json() def ep_bolfi.utility.fitting_functions.OCV_fit_result.to_json (
self)

6.10.4 Member Data Documentation

6.10.4.1 a ep_bolfi.utility.fitting_functions.OCV_fit_result.a

The a (inverse plateau widths) parameters.

6.10.4.2 E_0 ep_bolfi.utility.fitting_functions.OCV_fit_result.E_0

The E_0 (plateau voltages) parameters.

6.10.4.3 fit ep_bolfi.utility.fitting_functions.OCV_fit_result.fit

The fit parameters of the OCV function from Birkl.

et al., excluding the estimated SOC range.

6.10.4.4 function_string ep_bolfi.utility.fitting_functions.OCV_fit_result.function_string

The string representation of the interpolating spline fitted to the inverse.

6.10.4.5 OCV ep_bolfi.utility.fitting_functions.OCV_fit_result.OCV

The OCV data points.

May be adjusted from the original data.

6.10.4.6 optimize_result ep_bolfi.utility.fitting_functions.OCV_fit_result.optimize_result

The scipy.optimize.OptimizeResult that led to the fit.

6.10.4.7 SOC ep_bolfi.utility.fitting_functions.OCV_fit_result.SOC

The SOC data points.

6.10.4.8 SOC_offset ep_bolfi.utility.fitting_functions.OCV_fit_result.SOC_offset

If another electrode was factored out in the data, this may contain its SOC at SOC 0 of the electrode of interest.

6.10.4.9 SOC_range ep_bolfi.utility.fitting_functions.OCV_fit_result.SOC_range

The SOC range of the data.

6.10.4.10 SOC_scale ep_bolfi.utility.fitting_functions.OCV_fit_result.SOC_scale

If another electrode was factored out in the data, this may contain the the rate of change of its SOC to that of the electrode of interest.

6.10.4.11 spline_interpolation_coefficients ep_bolfi.utility.fitting_functions.OCV_fit_result.spline_interpolation_↩ coefficients

The coefficients of the interpolating spline fitted to the inverse.

6.10.4.12 spline_interpolation_knots ep_bolfi.utility.fitting_functions.OCV_fit_result.spline_interpolation_knots

The knots of the interpolating spline fitted to the inverse.

6.10.4.13 Δx ep_bolfi.utility.fitting_functions.OCV_fit_result.Δx

The Δx (phase proportion) parameters.

The documentation for this class was generated from the following file:

- utility/fitting_functions.py

6.11 ep_bolfi.optimization.EP_BOLFI.Optimizer_State Class Reference

Handles the heuristics for the EP-BOLFI operation modes.

Public Member Functions

- def __init__ (self, input_dim, mcmc_chains, total_evidence, posterior_samples, gelman_rubin_threshold, ess_ratio_resample=5.0, ess_ratio_sampling_from_zero=-1.0, ess_ratio_abort=20.0, posterior_sampling_increase=1.↩ 2, model_resampling_increase=1.2)
- def calculate_next_step (self, ess_ratio, action=None)

Public Attributes

- input_dim
- mcmc_chains
- total_evidence
- posterior_samples
- gelman_rubin_threshold
- ess_ratio_resample
- ess_ratio_sampling_from_zero
- ess_ratio_abort
- posterior_sampling_increase
- model_resampling_increase
- finished
- sampling_from_zero
- resampling
- initials
- order_of_actions
- current_action
- verbose_actions

6.11.1 Detailed Description

Handles the heuristics for the EP-BOLFI operation modes.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 __init__()

def ep_bolfi.optimization.EP_BOLFI.Optimizer_State.__init__ (
self,
input_dim,
mcmc_chains,
total_evidence,
posterior_samples,
gelman_rubin_threshold,
ess_ratio_resample = 5.0,
ess_ratio_sampling_from_zero = -1.0,
ess_ratio_abort = 20.0,
posterior_sampling_increase = 1.2,
model_resampling_increase = 1.2)

6.11.3 Member Function Documentation

6.11.3.1 calculate_next_step()

def ep_bolfi.optimization.EP_BOLFI.Optimizer_State.calculate_next_step (
self,
ess_ratio,
action = None)

6.11.4 Member Data Documentation

6.11.4.1 current_action ep_bolfi.optimization.EP_BOLFI.Optimizer_State.current_action

6.11.4.2 ess_ratio_abort ep_bolfi.optimization.EP_BOLFI.Optimizer_State.ess_ratio_abort

6.11.4.3 ess_ratio_resample ep_bolfi.optimization.EP_BOLFI.Optimizer_State.ess_ratio_resample

6.11.4.4 ess_ratio_sampling_from_zero ep_bolfi.optimization.EP_BOLFI.Optimizer_State.ess_ratio_sampling_from_zero

6.11.4.5 finished ep_bolfi.optimization.EP_BOLFI.Optimizer_State.finished

6.11.4.6 gelman_rubin_threshold ep_bolfi.optimization.EP_BOLFI.Optimizer_State.gelman_rubin_threshold

6.11.4.7 initials ep_bolfi.optimization.EP_BOLFI.Optimizer_State.initials

6.11.4.8 input_dim ep_bolfi.optimization.EP_BOLFI.Optimizer_State.input_dim

6.11.4.9 mcmc_chains ep_bolfi.optimization.EP_BOLFI.Optimizer_State.mcmc_chains

6.11.4.10 model_resampling_increase ep_bolfi.optimization.EP_BOLFI.Optimizer_State.model_resampling_increase

6.11.4.11 order_of_actions ep_bolfi.optimization.EP_BOLFI.Optimizer_State.order_of_actions

6.11.4.12 posterior_samples ep_bolfi.optimization.EP_BOLFI.Optimizer_State.posterior_samples

6.11.4.13 posterior_sampling_increase ep_bolfi.optimization.EP_BOLFI.Optimizer_State.posterior_sampling_increase

6.11.4.14 resampling ep_bolfi.optimization.EP_BOLFI.Optimizer_State.resampling

6.11.4.15 sampling_from_zero ep_bolfi.optimization.EP_BOLFI.Optimizer_State.sampling_from_zero

6.11.4.16 total_evidence ep_bolfi.optimization.EP_BOLFI.Optimizer_State.total_evidence

6.11.4.17 verbose_actions ep_bolfi.optimization.EP_BOLFI.Optimizer_State.verbose_actions

The documentation for this class was generated from the following file:

- optimization/EP_BOLFI.py

6.12 ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator Class Reference

Normalizes sampling to a standard normal distribution.

Public Member Functions

- def __init__ (self, simulator, fixed_parameters, free_parameters_names, r, Q, experimental_data, feature_extractor, transform_parameters={}, fixed_parameter_order=None)
- def search_to_transformed_trial (self, search_space_parameters)

    Transforms search space parameters to model ones.
- def transformed_trial_to_search (self, model_space_parameters)

    Transforms model space parameters to model ones.
- def undo_transformation (self, transformed_trial_parameters)

    Undo the transforms in 'self.transform_parameters'.
- def apply_transformation (self, trial_parameters)

    Apply the transforms in 'self.transform_parameters'.
- def elfi_simulator (self, ∗args, ∗∗kwargs)

    A battery model simulator that can be used with ELFI.

Public Attributes

- simulator
- fixed_parameters
- free_parameters_names
- fixed_parameter_order
- r
- Q
- experimental_data
- feature_extractor
- transform_parameters
- log_of_tried_parameters

  Stores all parameter combinations that have been tried.
- experimental_features

  Extract the features from the experimental data.
- input_dim

  Input dimension of the estimation task.
- output_dim

  Output dimension of the estimation task (number of features).
- add_parameters

  Create a function to combine the free and fixed parameters.
- back_transform_matrix

  Compute the linear transformation of parameters for which the covariance of the underlying multivariate normal distribution is a diagonal matrix.
- variances

  Variances of the model parameters.
- transform_matrix

  Inverse of back_transform_matrix.
- transformed_means

  transform_matrix @ Q @ back_transform_matrix is diagonal.
- norm_factor

  Now that the multivariate normal distribution is decomposed into various univariate ones, norm them to have equal variance 1.
- un_norm_factor

  Inverse of norm_factor.
- normed_means

  Expectation values of the normed univariate normal distributions.

6.12.1 Detailed Description

Normalizes sampling to a standard normal distribution.

In order to help BOLFI to work efficiently with the least amount of setup required, this class mediates between the model parameters and a standard normal distribution for sampling. In a sense, the simulator output gets transformed into covariance eigenvectors.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 __init__() def ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.__init__ (
self,
simulator,
fixed_parameters,
free_parameters_names,
r,
Q,
experimental_data,
feature_extractor,
transform_parameters = {},
fixed_parameter_order = None)

Parameters

simulator	The function that returns results given parameters.
fixed_parameters	Dictionary of parameters that stay fixed and their values.
free_parameters_names	List of the names of parameters which shall be inferred.
r	'Q' times the mean of the distribution of free parameters.
Q	Inverse covariance matrix of free parameters, i.e. precision. It is used to transform the free parameters given to the 'simulator' into the ones used in the 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 that the model will be fitted to. It has to have the same structure as the 'simulator' output.
feature_extractor	A function that takes the output of 'simulator' / the 'experimental_data' and returns a list of numbers, the features.
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 2-tuples. The first entry is a function taking the search space parameter and returning the model parameter. The second entry is the inverse function.
fixed_parameter_order	Optional fixed parameter order. Prevents erroneous behaviour when the parameter dictionaries get reshuffled. Highly recommended.

6.12.3 Member Function Documentation

6.12.3.1 apply_transformation() def ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.apply_transformation (
self,
trial_parameters)

Apply the transforms in 'self.transform_parameters'.

Parameters

trial_parameters	A dictionary. The keys are the 'free_parameters_names' and the values are the actual model parameters.

Returns

The given dictionary with the vales transformed to the modified parameter space as specified in 'self.↩ transform_parameters'.

6.12.3.2 elfi_simulator() def ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.elfi_simulator (

self,
∗ *args,*
∗∗ *kwargs*)

A battery model simulator that can be used with ELFI.

Parameters

∗*args*	The parameters as given by the prior nodes. Their order has to correspond to that of the parameter 'free_parameters' given to 'return_simulator'.
∗∗*kwargs*	Keyword parameters batch_size and random_state, but both are unused (they just get passed by BOLFI).

Returns

Simulated features for the given free parameters.

6.12.3.3 search_to_transformed_trial() def ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.search_to_↩ transformed_trial (

self,
search_space_parameters)

Transforms search space parameters to model ones.

Parameters

search_space_parameters	A list of lists which each contain a single search space parameter sample as it is returned by the sample functions of ELFI. In the case of only sample, a list also works.

Returns

A dictionary with its keys as the names of the parameters. Their order in the 'search_space_parameters' is given by the order of 'self.free_parameters_names'. The values yield the model parameters when passed through the functions in 'self.transform_parameters'.

6.12.3.4 transformed_trial_to_search() def ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.transformed_trial_↩ to_search (

self,
model_space_parameters)

Transforms model space parameters to model ones.

Parameters

model_space_parameters	A dictionary. The keys are the 'self.free_parameters_names' and the values are the model parameters after applying the transformations given in 'self.transform_parameters'.

Returns

A list (of lists) which each contain a single search space parameter sample as it is returned by the sample functions of ELFI. If the 'model_space_parameters' dictionary values are numbers, the returned value is a list. If they are lists, the returned value is a list of corresponding lists. In that case, each and every list must have the same length.

6.12.3.5 undo_transformation() def ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.undo_transformation (

self,
transformed_trial_parameters)

Undo the transforms in 'self.transform_parameters'.

Parameters

transformed_trial_parameters	A dictionary. The keys are the 'free_parameters_names' and the values are the model parameters after they have been transformed as specified in 'self.transform_parameters'.

Returns

The given dictionary with the values transformed back to the actual model parameter values.

6.12.4 Member Data Documentation

6.12.4.1 add_parameters ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.add_parameters

Create a function to combine the free and fixed parameters.

6.12.4.2 back_transform_matrix ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.back_transform_matrix

Compute the linear transformation of parameters for which the covariance of the underlying multivariate normal distribution is a diagonal matrix.

That is, compute the eigenvectors of Q. It is more stable since Q has growing eigenvectors in convergence.

6.12.4.3 experimental_data ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.experimental_data

6.12.4.4 experimental_features ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.experimental_features

Extract the features from the experimental data.

6.12.4.5 feature_extractor ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.feature_extractor

6.12.4.6 fixed_parameter_order ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.fixed_parameter_order

6.12.4.7 fixed_parameters ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.fixed_parameters

6.12.4.8 free_parameters_names ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.free_parameters_names

6.12.4.9 input_dim ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.input_dim

Input dimension of the estimation task.

6.12.4.10 log_of_tried_parameters ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.log_of_tried_parameters

Stores all parameter combinations that have been tried.

6.12.4.11 norm_factor ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.norm_factor

Now that the multivariate normal distribution is decomposed into various univariate ones, norm them to have equal variance 1.

6.12.4.12 normed_means ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.normed_means

Expectation values of the normed univariate normal distributions.

6.12.4.13 output_dim ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.output_dim

Output dimension of the estimation task (number of features).

6.12.4.14 Q ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.Q

6.12.4.15 r ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.r

6.12.4.16 simulator ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.simulator

6.12.4.17 transform_matrix ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.transform_matrix

Inverse of back_transform_matrix.

6.12.4.18 transform_parameters ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.transform_parameters

6.12.4.19 transformed_means ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.transformed_means

transform_matrix @ Q @ back_transform_matrix is diagonal.

The correct transformation for vectors v is then transform_matrix @ v. The product below corresponds to Q^{-1} @ r. It is just expressed in the eigenvector space of Q for efficiency.

6.12.4.20 un_norm_factor ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.un_norm_factor

Inverse of norm_factor.

6.12.4.21 variances ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator.variances

Variances of the model parameters.

The documentation for this class was generated from the following file:

- optimization/EP_BOLFI.py

6.13 ep_bolfi.utility.dataset_formatting.Static_Information Class Reference

Contains additional informations, e.g.

Inheritance diagram for ep_bolfi.utility.dataset_formatting.Static_Information:

Collaboration diagram for ep_bolfi.utility.dataset_formatting.Static_Information:

Public Member Functions

- def __init__ (self, timepoints, currents, voltages, other_columns={}, indices=None)
- def to_json (self)
- 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.
- def extend (self, other)

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.

6.13.1 Detailed Description

Contains additional informations, e.g.

for GITT. Each member variable is a list and has the same length as the other ones.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 __init__()

def ep_bolfi.utility.dataset_formatting.Static_Information.__init__ (
self,
timepoints,
currents,
voltages,
other_columns = {},
indices = None)

Reimplemented from ep_bolfi.utility.dataset_formatting.Cycling_Information.

6.13.3 Member Function Documentation

6.13.3.1 extend() def ep_bolfi.utility.dataset_formatting.Static_Information.extend (

self,
other)

Reimplemented from ep_bolfi.utility.dataset_formatting.Cycling_Information.

6.13.3.2 subarray() def ep_bolfi.utility.dataset_formatting.Static_Information.subarray (

self,
array)

Selects the data segments with the given indices.

Parameters

array	The indices of the segments that are to be returned.

Returns

A new Static_Information object containing the subset.

Reimplemented from ep_bolfi.utility.dataset_formatting.Cycling_Information.

6.13.3.3 subslice() def ep_bolfi.utility.dataset_formatting.Static_Information.subslice (

self,
start,
stop,
step = 1)

Selects a subslice of the data segments.

The arguments exactly match the slice(...) notation.

Parameters

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.

Reimplemented from ep_bolfi.utility.dataset_formatting.Cycling_Information.

6.13.3.4 to_json() def ep_bolfi.utility.dataset_formatting.Static_Information.to_json (*self*)

Reimplemented from ep_bolfi.utility.dataset_formatting.Cycling_Information.

6.13.4 Member Data Documentation

6.13.4.1 asymptotic_voltages ep_bolfi.utility.dataset_formatting.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.

6.13.4.2 exp_I_decays ep_bolfi.utility.dataset_formatting.Static_Information.exp_I_decays

Same as exp_U_decays for current decays (PITT).

6.13.4.3 exp_U_decays ep_bolfi.utility.dataset_formatting.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). """

6.13.4.4 ir_steps ep_bolfi.utility.dataset_formatting.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/dataset_formatting.py

6.14 ep_bolfi.utility.preprocessing.SubstitutionDict Class Reference

A dictionary with some automatic substitutions.

Inheritance diagram for ep_bolfi.utility.preprocessing.SubstitutionDict:

Collaboration diagram for ep_bolfi.utility.preprocessing.SubstitutionDict:

Public Member Functions

- def __init__ (self, storage, substitutions={})
- def __delitem__ (self, key)
- def __getitem__ (self, key)
- def __iter__ (self)
- def __len__ (self)
- def __setitem__ (self, key, value)
- def __str__ (self)
- def __repr__ (self)
- def log_lock (self)
- def dependent_variables (self, parameters)

6.14.1 Detailed Description

A dictionary with some automatic substitutions.

"substitutions" is a dictionary that extends "storage" with automatic substitution rules depending on its value types:

- string, which serves the value of "storage" at that value.
- callable which takes one parameter, which will get passed its SubstitutionDict instance and serves its return value.
- any other type, which serves the value as-is. Assigning values to keys afterwards will overwrite substitutions.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 __init__()

def ep_bolfi.utility.preprocessing.SubstitutionDict.__init__ (
self,
storage,
substitutions = {})

6.14.3 Member Function Documentation

6.14.3.1 __delitem__()

def ep_bolfi.utility.preprocessing.SubstitutionDict.__delitem__ (
self,
key)

6.14.3.2 __getitem__()

def ep_bolfi.utility.preprocessing.SubstitutionDict.__getitem__ (
self,
key)

6.14.3.3 __iter__()

def ep_bolfi.utility.preprocessing.SubstitutionDict.__iter__ (
self)

6.14.3.4 __len__()

def ep_bolfi.utility.preprocessing.SubstitutionDict.__len__ (
self)

6.14.3.5 __repr__()

def ep_bolfi.utility.preprocessing.SubstitutionDict.__repr__ (
self)

6.14.3.6 __setitem__()

def ep_bolfi.utility.preprocessing.SubstitutionDict.__setitem__ (
self,
key,
value)

6.14.3.7 __str__()

def ep_bolfi.utility.preprocessing.SubstitutionDict.__str__ (
self)

6.14.3.8 dependent_variables()

def ep_bolfi.utility.preprocessing.SubstitutionDict.dependent_variables (
self,
parameters)

6.14.3.9 log_lock()

def ep_bolfi.utility.preprocessing.SubstitutionDict.log_lock (
self)

The documentation for this class was generated from the following file:

- utility/preprocessing.py

7 File Documentation

7.1 __init__.py File Reference

Namespaces

- ep_bolfi

7.2 models/__init__.py File Reference

Namespaces

- ep_bolfi.models

7.3 optimization/__init__.py File Reference

Namespaces

- ep_bolfi.optimization

7.4 utility/__init__.py File Reference

Namespaces

- ep_bolfi.utility

7.5 models/assess_effective_parameters.py File Reference

Classes

- class ep_bolfi.models.assess_effective_parameters.Effective_Parameters
  Calculates, stores and prints effective parameters.

Namespaces

- ep_bolfi.models.assess_effective_parameters
  Evaluates a parameter set for, e.g., overpotential and capacity.

7.6 models/electrolyte.py File Reference

Classes

- class ep_bolfi.models.electrolyte.Electrolyte_internal
  Defining equations for a symmetric Li cell with electrolyte.
- class ep_bolfi.models.electrolyte.Electrolyte
  Electrolyte model assuming a symmetric Li-metal cell.

Namespaces

- ep_bolfi.models.electrolyte

  Contains a PyBaMM-compatible electrolyte model.

7.7 models/solversetup.py File Reference

Namespaces

- ep_bolfi.models.solversetup

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

Functions

- def ep_bolfi.models.solversetup.solver_setup (model, parameters, submesh_types, var_pts, spatial_↩ methods, geometry=None, reltol=1e-6, abstol=1e-6, root_tol=1e-3, dt_max=None, free_parameters=[ ], verbose=False, logging_file=None)

  Processes the model and returns a runnable solver.
- def ep_bolfi.models.solversetup.simulation_setup (model, operation_input, parameters, submesh_types, var_pts, spatial_methods, geometry=None, reltol=1e-6, abstol=1e-6, root_tol=1e-3, dt_max=None, free_↩ parameters=[ ], verbose=False, logging_file=None)

  Processes the model and returns a runnable solver.
- def ep_bolfi.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 ep_bolfi.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.

7.8 models/standard_parameters.py File Reference

Namespaces

- ep_bolfi.models.standard_parameters

  Comprehensive list of parameters of every model.

Functions

- def ep_bolfi.models.standard_parameters.SOC_n_dim_init (x)

  Initial SOC of the anode.
- def ep_bolfi.models.standard_parameters.SOC_n_init (x)

  Non-dimensionalized initial SOC of the anode.
- def ep_bolfi.models.standard_parameters.SOC_p_dim_init (x)

  Initial SOC of the cathode.
- def ep_bolfi.models.standard_parameters.SOC_p_init (x)

  Non-dimensionalized initial SOC of the cathode.
- def ep_bolfi.models.standard_parameters.D_e_dim (c_e_dim, T_dim)

  Electrolyte diffusivity.
- def ep_bolfi.models.standard_parameters.D_e (c_e, T)

  Non-dimensionalized electrolyte diffusivity.

- def ep_bolfi.models.standard_parameters.κ_e_dim (c_e_dim, T_dim)

  Electrolyte conductivity.
- def ep_bolfi.models.standard_parameters.κ_e (c_e, T)

  Non-dimensionalized electrolyte conductivity.
- def ep_bolfi.models.standard_parameters.t_plus_dim (c_e_dim)

  Transference number.
- def ep_bolfi.models.standard_parameters.t_plus (c_e)

  Non-dimensionalized (referring to the input) transference number.
- def ep_bolfi.models.standard_parameters.one_plus_dlnf_dlnc_dim (c_e_dim)

  Thermodynamic factor.
- def ep_bolfi.models.standard_parameters.one_plus_dlnf_dlnc (c_e)

  Non-dimensionalized (referring to the input) thermodynamic factor.
- def ep_bolfi.models.standard_parameters.D_n_dim (SOC_n, T_dim)

  Anode diffusivity.
- def ep_bolfi.models.standard_parameters.D_n (SOC_n, T)

  Non-dimensionalized anode diffusivity.
- def ep_bolfi.models.standard_parameters.D_p_dim (SOC_p, T_dim)

  Cathode diffusivity.
- def ep_bolfi.models.standard_parameters.D_p (SOC_p, T)

  Non-dimensionalized cathode diffusivity.
- def ep_bolfi.models.standard_parameters.i_{sn}_0_dim (c_n_dim, SOC_n_surf_dim, c_n_max, T_dim)

  Anode exchange current density.
- def ep_bolfi.models.standard_parameters.i_{sn}_0 (c_n, SOC_n_surf, c_n_max, T)

  Non-dimensionalized anode exchange current density.
- def ep_bolfi.models.standard_parameters.d_c_n_i_{sn}_0_dim (c_n_dim, SOC_n_surf_dim, c_n_max, T_dim)

  ∂ anode exchange current density / ∂ electrolyte concentration.
- def ep_bolfi.models.standard_parameters.d_c_n_i_{sn}_0 (c_n, SOC_n_surf, c_n_max, T)

  The non-dimensionalized version of the prior variable.
- def ep_bolfi.models.standard_parameters.i_{sep}_0_dim (c_{ep}_dim, SOC_p_surf_dim, c_p_max, T_dim)

  Cathode exchange current density.
- def ep_bolfi.models.standard_parameters.i_{sep}_0 (c_{ep}, SOC_p_surf, c_p_max, T)

  Non-dimensionalized cathode exchange current density.
- def ep_bolfi.models.standard_parameters.d_c_{ep}_i_{sep}_0_dim (c_{ep}_dim, SOC_p_surf_dim, c_p_max, T_dim)

  ∂ cathode exchange current density / ∂ electrolyte concentration.
- def ep_bolfi.models.standard_parameters.d_c_{ep}_i_{sep}_0 (c_{ep}, SOC_p_surf, c_p_max, T)

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

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

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

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

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

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

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

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

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

- def ep_bolfi.models.standard_parameters.OCV_n_dim (SOC_n, T_dim)

  Anode OCV.
- def ep_bolfi.models.standard_parameters.OCV_n (SOC_n, T)

  Non-dimensionalized anode OCV.
- def ep_bolfi.models.standard_parameters.dOCV_n_dim_dSOC_n (SOC_n, T_dim)

  ∂ anode OCV / ∂ anode SOC.
- def ep_bolfi.models.standard_parameters.dOCV_n_dSOC_n (SOC_n, T)

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

  Cathode OCV.
- def ep_bolfi.models.standard_parameters.OCV_p (SOC_p, T)

  Non-dimensionalized cathode OCV.
- def ep_bolfi.models.standard_parameters.dOCV_p_dim_dSOC_p (SOC_p, T_dim)

  ∂ cathode OCV / ∂ cathode SOC.
- def ep_bolfi.models.standard_parameters.dOCV_p_dSOC_p (SOC_p, T)

  Non-dimensionalized ∂ cathode OCV / ∂ cathode SOC.

Variables

- ep_bolfi.models.standard_parameters.R = Scalar(constants.R)
- ep_bolfi.models.standard_parameters.F = Scalar(constants.physical_constants["Faraday constant"][0])
- ep_bolfi.models.standard_parameters.k_B = constants.physical_constants["Boltzmann constant"][0]
- ep_bolfi.models.standard_parameters.q_e = constants.physical_constants["electron volt"][0]
- ep_bolfi.models.standard_parameters.T_ref = Parameter("Reference temperature [K]")
- ep_bolfi.models.standard_parameters.T_init = Parameter("Initial temperature [K]")
- ep_bolfi.models.standard_parameters.thermal_voltage = R ∗ T_ref / F
- ep_bolfi.models.standard_parameters.ΔT = Scalar(1)
- ep_bolfi.models.standard_parameters.L_n_dim = Parameter("Negative electrode thickness [m]")
- ep_bolfi.models.standard_parameters.L_s_dim = Parameter("Separator thickness [m]")
- ep_bolfi.models.standard_parameters.L_p_dim = Parameter("Positive electrode thickness [m]")
- ep_bolfi.models.standard_parameters.L_dim = L_n_dim + L_s_dim + L_p_dim
- ep_bolfi.models.standard_parameters.A = Parameter("Current collector perpendicular area [m2]")
- ep_bolfi.models.standard_parameters.V = Parameter("Cell volume [m3]")
- ep_bolfi.models.standard_parameters.L_x = L_dim
- ep_bolfi.models.standard_parameters.L_y = Parameter("Electrode width [m]")
- ep_bolfi.models.standard_parameters.L_z = Parameter("Electrode height [m]")
- ep_bolfi.models.standard_parameters.C = Parameter("Typical current [A]")
- ep_bolfi.models.standard_parameters.U_l = pybamm.Parameter("Lower voltage cut-off [V]")
- ep_bolfi.models.standard_parameters.U_u = pybamm.Parameter("Upper voltage cut-off [V]")
- ep_bolfi.models.standard_parameters.c_e_typ = pybamm.Parameter("Typical electrolyte concentration [mol.m-3]")
- ep_bolfi.models.standard_parameters.c_n = pybamm.Parameter("Maximum concentration in negative electrode [mol.m-3]")
- ep_bolfi.models.standard_parameters.c_p = pybamm.Parameter("Maximum concentration in positive electrode [mol.m-3]")
- ep_bolfi.models.standard_parameters.σ_n_dim = pybamm.Parameter("Negative electrode conductivity [S.↩ m-1]")
- ep_bolfi.models.standard_parameters.σ_p_dim = pybamm.Parameter("Positive electrode conductivity [S.↩ m-1]")
- ep_bolfi.models.standard_parameters.a_n_dim = Parameter("Negative electrode surface area to volume ratio [m-1]")

- ep_bolfi.models.standard_parameters.a_p_dim = Parameter("Positive electrode surface area to volume ratio [m-1]")
- ep_bolfi.models.standard_parameters.R_n = Parameter("Negative particle radius [m]")
- ep_bolfi.models.standard_parameters.R_p = Parameter("Positive particle radius [m]")
- ep_bolfi.models.standard_parameters.α_{nn} = Parameter("Negative electrode anodic charge-transfer coefficient")
- ep_bolfi.models.standard_parameters.α_{pn} = Parameter("Negative electrode cathodic charge-transfer coefficient")
- ep_bolfi.models.standard_parameters.α_{np} = Parameter("Positive electrode anodic charge-transfer coefficient")
- ep_bolfi.models.standard_parameters.α_{pp} = Parameter("Positive electrode cathodic charge-transfer coefficient")
- ep_bolfi.models.standard_parameters.β_n_scalar
- ep_bolfi.models.standard_parameters.β_{es}_scalar = Parameter("Separator Bruggeman coefficient (electrolyte)")
- ep_bolfi.models.standard_parameters.β_{ep}_scalar
- ep_bolfi.models.standard_parameters.β_n = pybamm.PrimaryBroadcast(β_n_scalar, "negative electrode")
- ep_bolfi.models.standard_parameters.β_{es} = pybamm.PrimaryBroadcast(β_{es}_scalar, "separator")
- ep_bolfi.models.standard_parameters.β_{ep} = pybamm.PrimaryBroadcast(β_{ep}_scalar, "positive electrode")
- ep_bolfi.models.standard_parameters.β_{sn}_scalar = Parameter("Negative electrode Bruggeman coefficient (electrode)")
- ep_bolfi.models.standard_parameters.β_{ss}_scalar = Parameter("Separator Bruggeman coefficient (electrode)")
- ep_bolfi.models.standard_parameters.β_{sp}_scalar = Parameter("Positive electrode Bruggeman coefficient (electrode)")
- ep_bolfi.models.standard_parameters.β_e = pybamm.Concatenation(β_n, β_{es}, β_{ep})
- ep_bolfi.models.standard_parameters.ε_n_scalar = Parameter("Negative electrode porosity")
- ep_bolfi.models.standard_parameters.ε_s_scalar = Parameter("Separator porosity")
- ep_bolfi.models.standard_parameters.ε_p_scalar = Parameter("Positive electrode porosity")
- ep_bolfi.models.standard_parameters.ε_n = pybamm.PrimaryBroadcast(ε_n_scalar, "negative electrode")
- ep_bolfi.models.standard_parameters.ε_s = pybamm.PrimaryBroadcast(ε_s_scalar, "separator")
- ep_bolfi.models.standard_parameters.ε_p = pybamm.PrimaryBroadcast(ε_p_scalar, "positive electrode")
- ep_bolfi.models.standard_parameters.ε = pybamm.Concatenation(ε_n, ε_s, ε_p)
- ep_bolfi.models.standard_parameters.ε^β = ε**β_e
- ep_bolfi.models.standard_parameters.z_n = Parameter("Negative electrode electrons in reaction")
- ep_bolfi.models.standard_parameters.z_p = Parameter("Positive electrode electrons in reaction")
- ep_bolfi.models.standard_parameters.c_e_dim_init = Parameter("Initial concentration in electrolyte [mol.↩ m-3]")
- ep_bolfi.models.standard_parameters.c_e_init = c_e_dim_init / c_e_typ
- def ep_bolfi.models.standard_parameters.D_e_typ = D_e_dim(c_e_typ, T_ref)
- def ep_bolfi.models.standard_parameters.κ_e_typ = κ_e_dim(c_e_typ, T_ref)
- tuple ep_bolfi.models.standard_parameters.κ_e_hat = (R ∗ T_ref / F) / (C / A ∗ L_dim / κ_e_typ)
- def ep_bolfi.models.standard_parameters.D_n_typ = D_n_dim(Scalar(0.5), T_ref)
- def ep_bolfi.models.standard_parameters.D_p_typ = D_p_dim(Scalar(0.5), T_ref)
- def ep_bolfi.models.standard_parameters.i_{sn}_0_ref = i_{sn}_0_dim(c_e_typ, 0.5 ∗ c_n, c_n, T_ref)
- def ep_bolfi.models.standard_parameters.i_{sep}_0_ref = i_{sep}_0_dim(c_e_typ, 0.5 ∗ c_p, c_p, T_ref)
- def ep_bolfi.models.standard_parameters.OCV_n_ref = OCV_n_dim(SOC_n_init(0), T_ref)
- def ep_bolfi.models.standard_parameters.OCV_p_ref = OCV_p_dim(SOC_p_init(1), T_ref)
- ep_bolfi.models.standard_parameters.a_n = a_n_dim ∗ R_n
- ep_bolfi.models.standard_parameters.a_p = a_p_dim ∗ R_p
- tuple ep_bolfi.models.standard_parameters.Q = (1 - ε_p_scalar) ∗ L_p_dim ∗ c_p ∗ z_p ∗ F ∗ A
- ep_bolfi.models.standard_parameters.τ^d = F ∗ c_p ∗ L_dim / (C / A)
- int ep_bolfi.models.standard_parameters.τ_e = L_dim∗∗2 / D_e_typ
- int ep_bolfi.models.standard_parameters.τ_n = R_n∗∗2 / D_n_typ
- int ep_bolfi.models.standard_parameters.τ_p = R_p∗∗2 / D_p_typ

- ep_bolfi.models.standard_parameters.τ_{rn} = F $*$ c_n / (i_{sn}_0_ref $*$ a_n_dim)
- ep_bolfi.models.standard_parameters.τ_{rp} = F $*$ c_p / (i_{sep}_0_ref $*$ a_p_dim)
- ep_bolfi.models.standard_parameters.timescale = τ^d
- int ep_bolfi.models.standard_parameters.C_e = τ_e / τ^d
- int ep_bolfi.models.standard_parameters.C_n = τ_n / τ^d
- int ep_bolfi.models.standard_parameters.C_p = τ_p / τ^d
- ep_bolfi.models.standard_parameters.C_{rn} = τ_{rn} / τ^d
- ep_bolfi.models.standard_parameters.C_{rp} = τ_{rp} / τ^d
- ep_bolfi.models.standard_parameters.γ_e = c_e_typ / c_p
- ep_bolfi.models.standard_parameters.γ_n = c_n / c_p
- ep_bolfi.models.standard_parameters.γ_p = c_p / c_p
- ep_bolfi.models.standard_parameters.L_n = L_n_dim / L_dim
- ep_bolfi.models.standard_parameters.L_s = L_s_dim / L_dim
- ep_bolfi.models.standard_parameters.L_p = L_p_dim / L_dim
- ep_bolfi.models.standard_parameters.L_e
- tuple ep_bolfi.models.standard_parameters.σ_n = (thermal_voltage / (C / A $*$ L_dim)) $*$ σ_n_dim
- tuple ep_bolfi.models.standard_parameters.σ_p = (thermal_voltage / (C / A $*$ L_dim)) $*$ σ_p_dim
- ep_bolfi.models.standard_parameters.I_extern_dim
- ep_bolfi.models.standard_parameters.I_extern = I_extern_dim / C
- ep_bolfi.models.standard_parameters.n_electrodes_parallel
- ep_bolfi.models.standard_parameters.n_cells = Parameter(”Number of cells connected in series to make a battery”)
- ep_bolfi.models.standard_parameters.I_typ = C
- ep_bolfi.models.standard_parameters.A_cc = A
- ep_bolfi.models.standard_parameters.current_with_time = I_extern
- ep_bolfi.models.standard_parameters.dimensional_current_with_time = I_extern_dim
- ep_bolfi.models.standard_parameters.dimensional_current_density_with_time = I_extern_dim / A
- ep_bolfi.models.standard_parameters.voltage_low_cut = U_l
- ep_bolfi.models.standard_parameters.voltage_high_cut = U_u
- ep_bolfi.models.standard_parameters.capacity = Parameter(”Nominal cell capacity [A.h]”)

7.9 optimization/EP_BOLFI.py File Reference

Classes

- class ep_bolfi.optimization.EP_BOLFI.NDArrayEncoder
- class ep_bolfi.optimization.EP_BOLFI.Preprocessed_Simulator

  Normalizes sampling to a standard normal distribution.
- class ep_bolfi.optimization.EP_BOLFI.Optimizer_State

  Handles the heuristics for the EP-BOLFI operation modes.
- class ep_bolfi.optimization.EP_BOLFI.EP_BOLFI

  Expectation Propagation and Bayesian Optimization.

Namespaces

- ep_bolfi.optimization.EP_BOLFI
- ep_bolfi.EP_BOLFI.EP_BOLFI

  This file contains functions to perform Expectation Propagation on simulator models using BOLFI (Bayesian Optimization).

Functions

- def ep_bolfi.optimization.EP_BOLFI.combine_parameters_to_try (parameters, parameters_to_try_dict)

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

  Returns a function which sets some parameters in advance.

7.10 utility/dataset_formatting.py File Reference

Classes

- class ep_bolfi.utility.dataset_formatting.Measurement

  Defines common methods for measurement objects.
- class ep_bolfi.utility.dataset_formatting.Cycling_Information

  Contains basic cycling informations.
- class ep_bolfi.utility.dataset_formatting.Static_Information

  Contains additional informations, e.g.
- class ep_bolfi.utility.dataset_formatting.Impedance_Measurement

  Contains basic impedance data.

Namespaces

- ep_bolfi.utility.dataset_formatting

  Defines datatypes for further processing.

Functions

- def ep_bolfi.utility.dataset_formatting.read_csv_from_measurement_system (path, encoding, number↩ _of_comment_lines, headers, delimiter='\t', decimal='.', datatype="cycling", segment_column=-1, segments_to_process=None, current_sign_correction={}, correction_column=-1, flip_voltage_sign=False, flip_imaginary_impedance_sign=False, max_number_of_lines=-1)

  Read the measurements as returned by common instruments.
- def ep_bolfi.utility.dataset_formatting.print_hdf5_structure (h5py_object, depth=1, table_limit=4, verbose_limit=64)

  brief Simple HDF5 structure viewer.
- def ep_bolfi.utility.dataset_formatting.convert_none_notation_to_slicing (h5py_object, index)

  Access a slice of an HDF5 object by transferable notation.
- def ep_bolfi.utility.dataset_formatting.get_hdf5_dataset_by_path (h5py_object, path)

  Follow the structure of a HDF object to get a certain part.
- def ep_bolfi.utility.dataset_formatting.read_hdf5_table (path, data_location, headers, datatype="cycling", segment_location=None, segments_to_process=None, current_sign_correction={}, correction↩ _location=None, flip_voltage_sign=False, flip_imaginary_impedance_sign=False)

  Read the measurements as stored in a HDF5 file.

7.11 utility/fitting_functions.py File Reference

Classes

- class ep_bolfi.utility.fitting_functions.NDArrayEncoder
- class ep_bolfi.utility.fitting_functions.OCV_fit_result

  Contains OCV fit parameters and related information.

Namespaces

- ep_bolfi.utility.fitting_functions

  Various helper and fitting functions for processing measurement curves.

Functions

- def ep_bolfi.utility.fitting_functions.find_occurrences (sequence, value)

  Gives indices in sequence where it is closest to value.
- def ep_bolfi.utility.fitting_functions.smooth_fit (x, y, order=3, splits=None, w=None, s=None, display=False, derivatives=0)

  Calculates a smoothed spline with derivatives.
- def ep_bolfi.utility.fitting_functions.fit_exponential_decay (timepoints, voltages, recursive_depth=1, threshold=0.95)

  Extracts a set amount of exponential decay curves.
- def ep_bolfi.utility.fitting_functions.fit_sqrt (timepoints, voltages, threshold=0.95)

  Extracts a square root at the beginning of the data.
- def ep_bolfi.utility.fitting_functions.fit_drt (frequencies, impedances, lambda_value=-2.0)
- def ep_bolfi.utility.fitting_functions.laplace_transform (x, y, s)

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

  Calculates the conversion from "A parametric OCV model".
- def ep_bolfi.utility.fitting_functions.OCV_fit_function (E_OCV, ∗args, z=1.0, T=298.15, individual=False, fit_SOC_range=False, rescale=False)

  The OCV model from "A parametric OCV model".
- def ep_bolfi.utility.fitting_functions.d_dE_OCV_fit_function (E_OCV, ∗args, z=1.0, T=298.15, individual=False, fit_SOC_range=False, rescale=False)

  The derivative of fitting_functions.OCV_fit_function.
- def ep_bolfi.utility.fitting_functions.d2_dE2_OCV_fit_function (E_OCV, ∗args, z=1.0, T=298.15, individual=False, fit_SOC_range=False, rescale=False)

  The 2^{n} derivative of fitting_functions.OCV_fit_function.
- def ep_bolfi.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 ep_bolfi.utility.fitting_functions.inverse_d_dSOC_OCV_fit_function (SOC, ∗args, z=1.0, T=298.15, inverted=True)

  The derivative of the inverse of fitting_functions.OCV_fit_function.
- def ep_bolfi.utility.fitting_functions.inverse_d2_dSOC2_OCV_fit_function (SOC, ∗args, z=1.0, T=298.15, inverted=True)

  The 2nd derivative of the inverse of .OCV_fit_function.
- def ep_bolfi.utility.fitting_functions.fit_OCV (SOC, OCV, N=4, SOC_range_bounds=(0.2, 0.8), SOC_range↩ _limits=(0.0, 1.0), z=1.0, T=298.15, inverted=True, fit_SOC_range=True, distance_order=2, weights=None, initial_parameters=None, minimize_options=None)

  Fits data to fitting_functions.OCV_fit_function.
- def ep_bolfi.utility.fitting_functions.verbose_spline_parameterization (coeffs, knots, order, format='python', function_name="OCV", function_args="SOC", derivatives=0, spline_transformation='', verbose=False)

  Gives the monomic representation of a B-spline.

7.12 utility/preprocessing.py File Reference

Classes

- class ep_bolfi.utility.preprocessing.SubstitutionDict

  A dictionary with some automatic substitutions.

Namespaces

- ep_bolfi.utility.preprocessing

  Contains frequently used workflows in dataset preprocessing.

Functions

- def ep_bolfi.utility.preprocessing.fix_parameters (parameters_to_be_fixed)

  Returns a function which sets some parameters in advance.
- def ep_bolfi.utility.preprocessing.combine_parameters_to_try (parameters, parameters_to_try_dict)

  Give every combination as full parameter sets.
- def ep_bolfi.utility.preprocessing.calculate_means_and_standard_deviations (mean, covariance, free_↩ parameters_names, transform_parameters={}, bounds_in_standard_deviations=1, ∗∗kwargs)

  Calculate means and standard deviations.
- def ep_bolfi.utility.preprocessing.approximate_confidence_ellipsoid (parameters, free_parameters_names, covariance, mean=None, transform_parameters={}, refinement=True, confidence=0.95)

  Approximate a confidence ellipsoid.
- def ep_bolfi.utility.preprocessing.capacity (parameters, electrode="positive")

  Convenience function for calculating the capacity.
- def ep_bolfi.utility.preprocessing.calculate_SOC (timepoints, currents, initial_SOC=0, sign=1, capacity=1)

  Transforms applied current over time into SOC.
- def ep_bolfi.utility.preprocessing.calculate_both_SOC_from_OCV (parameters, negative_SOC_from_↩ cell_SOC, positive_SOC_from_cell_SOC, OCV)

  Calculates the SOC of both electrodes from their OCV.
- def ep_bolfi.utility.preprocessing.subtract_OCV_curve_from_cycles (dataset, parameters, starting_↩ SOC=None, starting_OCV=None, electrode="positive", current_sign=0, voltage_sign=0)

  Removes the OCV curve from a cycling measurement.
- def ep_bolfi.utility.preprocessing.subtract_both_OCV_curves_from_cycles (dataset, parameters, negative↩ _SOC_from_cell_SOC, positive_SOC_from_cell_SOC, starting_SOC=None, starting_OCV=None)

  Removes the OCV curve from a single cycle.
- def ep_bolfi.utility.preprocessing.laplace_transform (x, y, s)

  Performs a basic laplace transformation.
- def ep_bolfi.utility.preprocessing.find_occurrences (sequence, value)

  Gives indices in sequence where it is closest to value.
- def ep_bolfi.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=None, parameters_print=False)

  Tries to extract the OCV curve from CC-CV cycling data.
- def ep_bolfi.utility.preprocessing.calculate_desired_voltage (solution, t_eval, voltage_scale, overpotential, three_electrode=None, dimensionless_reference_electrode_location=0.5, parameters={})
- def ep_bolfi.utility.preprocessing.solve_all_parameter_combinations (model, t_eval, parameters, parameters_to_try, submesh_types, var_pts, spatial_methods, full_factorial=True, ∗∗kwargs)
- def ep_bolfi.utility.preprocessing.prepare_parameter_combinations (parameters, parameters_to_try, covariance, order_of_parameter_names, transform_parameters, confidence)

  Calculates all permutations of the parameter boundaries.
- def ep_bolfi.utility.preprocessing.parallel_simulator_with_setup (model, current_input, parameters, submesh_types, var_pts, spatial_methods, calc_esoh, inputs, t_eval, voltage_scale, overpotential, three_↩ electrode, dimensionless_reference_electrode_location, kwargs)
- def ep_bolfi.utility.preprocessing.simulate_all_parameter_combinations (model, current_input, submesh↩ _types, var_pts, spatial_methods, parameters, parameters_to_try=None, covariance=None, order_of_↩ parameter_names=None, additional_input_parameters=[ ], transform_parameters={}, confidence=0.↩ 95, full_factorial=True, calc_esoh=False, voltage_scale=1.0, overpotential=False, three_electrode=None, dimensionless_reference_electrode_location=0.5, ∗∗kwargs)

7.13 utility/visualization.py File Reference

Namespaces

- ep_bolfi.utility.visualization

    Various helper and plotting functions for common data visualizations.

Functions

- def ep_bolfi.utility.visualization.update_limits (ax, xmin=float('inf'), xmax=-float('inf'), ymin=float('inf'), ymax=-float('inf'))

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

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

    Makes sure that all items remain and all items show up.
- def ep_bolfi.utility.visualization.push_apart_text (fig, ax, text_objects, lock_xaxis=False, temp_↩ path="./temp_render.png")

    Push apart overlapping texts until no overlaps remain.
- def ep_bolfi.utility.visualization.make_segments (x, y)

    Create a list of line segments from x and y coordinates.
- def ep_bolfi.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 ep_bolfi.utility.visualization.nyquist_plot (fig, ax, ω, Z, cmap=plt.get_cmap('tab20b'), ls='-', lw=3, title↩ _text="Impedance Measurement", legend_text="impedance", colorbar_label="Frequency / Hz", add_↩ frequency_colorbar=True, equal_aspect=True)

    Plot an impedance measurement.
- def ep_bolfi.utility.visualization.bode_plot (fig, ax_real, ax_imag, ω, Z, cmap=plt.get_cmap('tab20b'), ls_↩ real='-', ls_imag='-.', lw=3, title_text="Impedance Measurement", legend_text="impedance")

    Plot an impedance measurement.
- def ep_bolfi.utility.visualization.plot_comparison (ax, solutions, errorbars, experiment, solution_↩ visualization=[ ], t_eval=None, title="", xlabel="", ylabel="", feature_visualizer=lambda ∗args:[ ], feature↩ _fontsize=12, interactive_plot=False, output_variables=None, voltage_scale=1.0, use_cycles=False, overpotential=False, three_electrode=None, dimensionless_reference_electrode_location=0.5, parameters=None)

    Tool for comparing simulation<->experiment with features.
- def ep_bolfi.utility.visualization.cc_cv_visualization (fig, ax, dataset, max_number_of_clusters=4, cmap=plt.get_cmap('tab20c'), check_location=[0.1, 0.7, 0.2, 0.225])

    Automatically labels and displays a CC-CV dataset.
- def ep_bolfi.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=None, parameters_print=False)

    Visualizes the OCV_fitting.OCV_from_CC_CV output.
- def ep_bolfi.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 ep_bolfi.utility.visualization.plot_measurement (fig, ax, dataset, title, cmap=plt.get_cmap('tab20c'), plot_current=True)

    Plots current and voltage curves in one diagram.

- def ep_bolfi.utility.visualization.fit_and_plot_OCV (ax, SOC, OCV, name, phases, SOC_range_↩ bounds=(0.2, 0.8), SOC_range_limits=(0.0, 1.0), z=1.0, T=298.15, fit=None, eval_SOC=[0, 1], eval_↩ points=200, spline_SOC_range=(0.01, 0.99), spline_order=2, spline_print=None, parameters_print=False, inverted=True, info_accuracy=True, normalized_xaxis=False, distance_order=2, weights=None, initial_↩ parameters=None, minimize_options=None)

  Fits an SOC(OCV)-model and an OCV(SOC)-evaluable spline.
- def ep_bolfi.utility.visualization.visualize_correlation (fig, ax, correlation, names=None, title=None, cmap=plt.get_cmap('BrBG'), entry_color='w')

  Produces a heatmap of a correlation matrix.