pyMETHES Documentation

Release 0.1.1

Alise Chachereau, Kerry Jansen, Markus Niese, Martin Vahlensiec

Jan 22, 2021

CONTENTS:

1 pyMETHES **1**
- 1.1 Documentation . . . 1
- 1.2 License . . . 1

2 Installation **3**
- 2.1 Stable release . . . 3
- 2.2 From sources . . . 3

3 Usage **5**

4 Overview **7**

5 API Documentation **9**
- 5.1 pyMETHES package . . . 9

6 Contributing **31**
- 6.1 Types of Contributions . . . 31
- 6.2 Get Started! . . . 32
- 6.3 Merge Request Guidelines . . . 33
- 6.4 Tips . . . 33
- 6.5 Deploying . . . 33

7 Authors **35**

8 History **37**
- 8.1 0.1.0 (2020-06-19) . . . 37
- 8.2 0.1.1 (2020-06-24) . . . 37
- 8.3 0.2.0 (2021-01-20) . . . 37

9 Indices and tables **39**

Python Module Index **41**

Index **43**

CHAPTER

ONE

PYMETHES

A Monte Carlo simulation of electron transport in low temperature plasmas.

Python version of the MATLAB's METHES package

1.1 Documentation

read pyMETHES documentation at RTD

1.2 License

GNU General Public License v3 (GPLv3)

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install pyMETHES, run this command in your terminal:

```
$ pip install pymethes
```

This is the preferred method to install pyMETHES, as it will always install the most recent stable release.

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

2.2 From sources

The sources for pyMETHES can be downloaded from the GitLab repo.

You can either clone the repository:

```
$ git clone git@gitlab.com:ethz_hvl/pymethes.git
```

Or download the tarball:

```
$ curl -OL https://gitlab.com/ethz_hvl/pymethes/-/archive/master/pymethes.tar.gz
```

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

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

CHAPTER

THREE

USAGE

To use pyMETHES in a project:

```
import pyMETHES

sim = pyMETHES.Simulation('config.json')
sim.run()
```

See more examples

CHAPTER

FOUR

OVERVIEW

Class diagram:

Simulation flow diagram:

CHAPTER

FIVE

API DOCUMENTATION

5.1 pyMETHES package

5.1.1 Submodules

pyMETHES.config module

Module for the Config class of the simulation.

class pyMETHES.config.**Config**(*config: Union[str, dict]*)
Bases: object

Configuration class for the Monte-Carlo simulation.

The configuration can be loaded from, or saved to, json or json5 files. Alternatively, it can be provided as, or exported to, a (nested) dictionary. The gas number density is not a configuration parameter, but a cached property of the Config class, which is computed from the pressure and temperature.

paths_to_cross_section_files
paths to the cross section files in txt format

Type list

gases
sum formulae of gases

Type list

fractions
proportions of the gases in the gas mixture

Type list

max_cross_section_energy
maximum cross section energy (eV)

Type float

output_directory
path to the output directory

Type str

base_name
prefix of the output filename

Type str

save_simulation_pickle
save the simulation as pickle file

Type bool

save_temporal_evolution
save temporal evolution

Type bool

save_swarm_parameters
save swarm parameters

Type bool

save_energy_distribution
save energy distribution

Type bool

EN
E/N ratio in (Td)

Type float

_pressure
gas pressure in Pa

Type float

_temperature
gas temperature in K

Type float

_gas_number_density
gas number density in m-3

Type float

num_e_initial
initial number of electrons

Type int

initial_pos_electrons
initial position [x, y, z] of the electrons' center of mass

Type list

initial_std_electrons
initial broadening of gaussian distributed electrons in x, y and z direction

Type list

initial_energy_distribution
The initial energy distribution of the electrons. Can be either `"zero"` (all electrons have zero kinetic energy), `"fixed"` (all electrons have the same energy) or `"maxwell-boltzmann"` (at temperature *initial_temperature*). Maxwell-Boltzmann support is experimental, check *pyMETHES.utils.maxwell_boltzmann_random()* and its test case to see if the required precision is achieved. The default is `"zero"`. If the initial distribution is *"fixed"*, *initial_energy* and *initial_direction* must be set.

Type str

initial_energy
The initial energy of the electrons in eV.

Type float

initial_direction
The initial direction of the electrons. Either the string `"random"` to give each electron a random direction or a tuple with three elements x, y, z specifying a single direction for all electrons.

Type Union[Tuple[float, float, float], str]

initial_temperature
The initial temperature in K. Used for the Maxwell-Boltzmann distribution.

Type Union[Float, int]

num_energy_bins
number of energy bins to group the electrons for the energy distribution

Type int

energy_sharing_factor
energy sharing factor for ionization collisions

Type float

isotropic_scattering
scattering: isotropic (true), non-isotropic according to Vahedi et al. (false)

Type bool

conserve
conservation of the number of electrons

Type bool

num_e_max
maximum allowed electron number (when it is reached, the number of electrons is then conserved until simulation ends)

Type int

seed
optional. If set to an integer it is used to seed the Simulation. If set to the string *"random"* no seeding occurs. Default value is *"random"*.

Type int, str

end_condition_type
Specifies the end condition. Can be `"steady-state"`, `"num_col_max"`, `"w_tol+ND_tol"` or `"custom"`. The `"custom"` end condition requires *is_done* to be set as well. Defaults to `"w_tol+ND_tol"`

Type str

w_tol
tolerance on the flux drift velocity. simulation ends when w_err/w < w_tol

Type float

DN_tol
tolerance on the flux diffusion coefficient. simulation ends when DN_err/w < DN_tol

Type float

num_col_max
maximum number of collisions during the simulation, simulation ends when it is reached

Type int

is_done
This function gets called to determine whether to end the simulation or not. Gets passed the simulation object as argument. Return `True` to stop the simulation, `False` otherwise.

Type Callable

timeout
End the simulation after `timeout` seconds. Zero means no timeout. Defaults to zero.

Type int

__init__(*config: Union[str, dict]*)
Instantiate the config.

Parameters **config** (*str, dict*) – path to a json or json5 config file, or dictionary.

property gas_number_density

property pressure

save_json(*path: str = 'config.json'*) → None
Saves the current configuration to a json file.

Parameters **path** (*str*) – path including the file name and extension, example: 'data/config.json'

save_json5(*path: str = 'config.json5'*) → None
Saves the current configuration to a json5 file.

Parameters **path** (*str*) – path including the file name and extension, example: 'data/config.json5'

property temperature

to_dict() → dict
Returns the current configuration as a dictionary.

Returns: dict of configuration

pyMETHES.cross_section module

Module for importing cross section data, based on the 'lxcat_data_parser' package.

Data points are added to each cross section at zero energy and at max_energy (which is user-defined). The classes 'CrossSection' and 'CrossSectionSet' of the lxcat_data_parser are extended to include 'interp1d' linear interpolations of each cross section.

class pyMETHES.cross_section.**InterpolatedCrossSection**(*cross_section_type: Union[str, lxcat_data_parser.import_tools.CrossSectionTypes], species: str, data: pandas.core.frame.DataFrame, mass_ratio: float, threshold: float, **kwargs*)

Bases: `lxcat_data_parser.import_tools.CrossSection`

Extension of the CrossSection class to add a linear interpolation.

type
type of collision

Type str, CrossSectionType

species
chemical formula of the species, example: N2

Type str

mass_ratio
ratio of electron mass to molecular mass

Type float

threshold
energy threshold (eV) of the cross section

Type float

database
name of the database

Type str

data
pandas DataFrame with columns "energy" and "cross section"

Type DataFrame

info
optional additional information on the cross section given via kwargs

interpolation
linear interpolation of the cross section

Type interp1d

__init__(*cross_section_type: Union[str, lxcat_data_parser.import_tools.CrossSectionTypes]*, *species: str*, *data: pandas.core.frame.DataFrame*, *mass_ratio: float*, *threshold: float*, ***kwargs*)
Instantiate an InterpolatedCrossSection.

Parameters

- **cross_section_type** (*str, CrossSectionType*) – type of collision
- **species** (*str*) – chemical formula of the species, example: N2
- **data** (*DataFrame*) – pandas DataFrame with columns "energy" and "cross section"
- **mass_ratio** (*float*) – ratio of electron mass to molecular mass
- **threshold** (*float*) – energy threshold (eV) of the cross section

class pyMETHES.cross_section.**InterpolatedCrossSectionSet**(*max_cross_section_energy: float*, *input_file: str*, *imposed_species: str*, *imposed_database: Optional[str] = None*)
Bases: lxcat_data_parser.import_tools.CrossSectionSet

Extension of the CrossSectionSet class to use InterpolatedCrossSections.

species
chemical formula of the species, example: N2

Type str

database
name of the database

Type str

cross_sections
list of CrossSection instances

Type list

__init__(*max_cross_section_energy: float*, *input_file: str*, *imposed_species: str*, *imposed_database: Optional[str] = None*)
Instantiate an InterpolatedCrossSectionSet.

Parameters

- **max_cross_section_energy** (*float*) – maximum cross section energy (eV), which is appended to the end of the cross section data
- **input_file** (*str*) – path to the cross section data
- **imposed_species** (*str*) – chemical formula of the species, example: N2
- **imposed_database** (*str*) – (optional) name of the database

Raises **CrossSectionReadingError** – if the provided file does not contain a valid set.

effective_to_elastic() → None
If effective cross section is given instead of elastic, calculate elastic cross section and replace the effective cross section.

plot(*block=True*) → None
Plot the cross section data.

Parameters **block** (*bool*) – block execution or not when showing plot

pyMETHES.electrons module

Module for the Electrons class.

The electrons class stores the position, velocity and acceleration of electrons in (3,n)-sized ndarrays. The class provides numerous class properties to compute electrons-related data such as the mean electron energy using cached class properties.

class pyMETHES.electrons.**Electrons**(*num_e_initial: int*, *init_pos: list*, *initial_std: list*, *e_field: float*, *initial_energy_distribution: str = 'zero'*, *initial_energy: Optional[float] = None*, *initial_direction: Optional[Union[Tuple[float, float, float], str]] = None*, *initial_temperature: Optional[float] = None*)
Bases: object

Stores information on a electron swarm in motion.

position
xyz-coordinates of each electron, dim=(num_e,3)

Type ndarray

velocity
xyz-velocities of each electron, dim=(num_e,3)

Type ndarray

acceleration
xyz-accelerations of each electron, dim=(num_e,3)

Type ndarray

__init__(*num_e_initial: int*, *init_pos: list*, *initial_std: list*, *e_field: float*, *initial_energy_distribution: str = 'zero'*, *initial_energy: Optional[float] = None*, *initial_direction: Optional[Union[Tuple[float, float, float], str]] = None*, *initial_temperature: Optional[float] = None*)
Instantiates an electron swarm.

Parameters

- **num_e_initial** (*int*) – Initial number of electrons
- **init_pos** (*list*) – initial xyz position (m) of electron swarm
- **initial_std** (*list*) – initial xyz std (m) of electron swarm
- **e_field** (*float*) – electric field strength (V/m) along z-direction
- **initial_energy_distribution** (*str, optional*) – The initial energy distribution. See *pyMETHES.config.Config.initial_energy_distribution*. `"maxwell-boltzmann"` requires temperature to be set. Defaults to `"zero"`
- **initial_energy** (*float, optional*) – Initial electron energy in eV. Needed if initial_energy_distribution is not `"zero"`.
- **initial_direction** (*Union[Tuple[float, float, float], str], optional*) – See *pyMETHES.config.Config.initial_direction*.
- **temperature** (*float, optional*) – Starting temperature in K. Required for Maxwell-Boltzmann distribution.

accelerate(*e_field: float*) → None
Calculates the electrons acceleration in the electric field.

Parameters **e_field** (*float*) – electric field strength (V/m) along z direction

property acceleration_norm

apply_scatter(*pos: numpy.ndarray*, *vel: numpy.ndarray*, *e_field: float*) → None
Updates the attributes of Electrons after scattering by the gas.

Parameters

- **pos** (*ndarray*) – new positions of the electrons (attachment and ionization)
- **vel** (*ndarray*) – new velocities of the electrons (scattering)
- **e_field** (*float*) – electric field strength (V/m) along z direction

property energy

property energy_distribution

free_flight(*duration: float*) → None
Update the attributes of Electrons after a free-flight in the electric field.

Parameters **duration** (*float*) – duration (s) of the free flight

property max_acceleration_norm

property max_energy

property max_velocity_norm

property mean_energy

property mean_position

property mean_velocity

property mean_velocity_moment

property num_e

plot_all(*show: bool = True*, *block: bool = True*) → None
Produces three Matplotlib Figures to visualize the position, velocity, and energy distribution of the electrons.

Parameters

- **show** (*bool*) – calls plt.show() if True, else does nothing
- **block** (*bool*) – if show is True, plt.show(block) is called

plot_energy(*show: bool = True*, *block: bool = True*) → matplotlib.figure.Figure
Produces a figure showing the electron energy distribution function and the electron energy probability function.

Parameters

- **show** (*bool*) – calls plt.show() if True, else does nothing
- **block** (*bool*) – if show is True, plt.show(block) is called

Returns: Matplotlib figure object

plot_position(*show: bool = True*, *block: bool = True*) → matplotlib.figure.Figure
Produces a figure showing the electron positions. The figure contains four subplots showing the 3D scatter, and histograms along x, y, and z directions of the electron positions.

Parameters

- **show** (*bool*) – calls plt.show() if True, else does nothing
- **block** (*bool*) – if show is True, plt.show(block) is called

Returns: Matplotlib figure object

plot_velocity(*show: bool = True*, *block: bool = True*) → matplotlib.figure.Figure
Produces a figure showing the electron positions in velocity space. The figure contains four subplots showing the 3D scatter, and histograms along v_x, v_y, and v_z directions.

Parameters

- **show** (*bool*) – calls plt.show() if True, else does nothing
- **block** (*bool*) – if show is True, plt.show(block) is called

Returns: Matplotlib figure object

reset_cache() → None
Resets the cache to ensure it is updated.

property std_energy

property var_position

property velocity_norm

pyMETHES.energy_distribution module

Module for the InstantaneousEnergyDistribution and TimeAveragedEnergyDistribution classes.

Both classed inherit from the abstract class EnergyDistribution. The InstantaneousEnergyDistribution class can be used to calculate the eedf and eepf at a given point in time. The TimeAveragedEnergyDistribution class allows for averaging the eedf and eepf over time. This is done by updating an histogram of electron energies at every time-step.

class pyMETHES.energy_distribution.**EnergyDistribution**
Bases: abc.ABC

Abstract base class for electron energy distributions.

energy_bins
bins to construct the histogram of electron energies

Type ndarray

energy_bin_centers
centers of the energy bins (length - 1)

Type ndarray

eedf
electron energy distribution function (eV-1)

Type ndarray

eepf
electron energy probability function (eV-3/2)

Type ndarray

calculate_bin_centers() → None
Calculate the center values of the energy bins.

class pyMETHES.energy_distribution.**InstantaneousEnergyDistribution**
Bases: *pyMETHES.energy_distribution.EnergyDistribution*

Stores the energy distribution of electrons.

calculate_distribution(*energies*, *energy_bins=None*) → None
Calculates the eedf and eepf associated to the input electron energies.

Parameters

- **energies** (*ndarray*) – array of electron energies
- **energy_bins** (*ndarray*) – (optional) energy bins for calculating the eedf and eepf. If None, the energy bins are automatically determined with the Freedman Diaconis Estimator.

class pyMETHES.energy_distribution.**TimeAveragedEnergyDistribution**
Bases: *pyMETHES.energy_distribution.EnergyDistribution*

Stores the energy distribution of electrons (averaged over time after swarm equilibration).

cumulative_energy_histogram
energy histogram cumulated over every time-step since steady-state

Type ndarray

calculate_distribution(*mean_energies: numpy.ndarray*) → None
Calculates the eedf and eepf using the cumulative energy histogram. Calculates the mean electron energy averaged over time.

Parameters **mean_energies** (*ndarray*) – mean energies at every time-step after swarm equilibration

collect_histogram(*energies*) → None
Updates the current eedf and eepf, using the electron energies given as input. The energy bins should be already defined and are re-used. This method should be called at every time-step.

Parameters **energies** (*ndarray*) – array of electron energies

generate_bins(*num_bins: int*, *max_energy: float*) → None
Generate fixed energy bins for averaging the eedf over time. The bins are spaced linearly and range from 0 to 150% of the maximum electron energy at the step where the method is called.

Parameters

- **num_bins** (*int*) – number of energy bins
- **max_energy** (*float*) – maximum electron energy (eV)

pyMETHES.gas_mixture module

Module for the GasMixture class, which aggregates data of different gases.

The GasMixture uses the cross_section module to read cross section data. It stores linear interpolations of each cross section, as well as the mass ratio, energy threshold and type of cross section, which are needed for the simulation. The proportions of the gas mixture should sum up to 1. A common energy_vector for all cross sections is calculated, as well as the total_cross_section.

class pyMETHES.gas_mixture.**GasMixture**(*gas_formulas: list*, *path_to_cross_section_files: list*, *proportions: list*, *max_cross_section_energy: float*)

Bases: object

Class representing a gas mixture.

cross_sections
array of cross section interpolations (from all gases)

Type ndarray

types
type of each cross section of each species

Type ndarray

thresholds
threshold of each cross section of each species

Type ndarray

mass_ratios
mass ratios (repeated for each cross section)

Type ndarray

is_attachment
boolean mask for ATTACHMENT cross sections

Type ndarray

is_ionization
boolean mask for IONIZATION cross sections

Type ndarray

energy_vector
common energy vector, containing all distinct energies of all cross sections

Type ndarray

total_cross_section
sum of all cross sections of all species, linearly interpolated at energy_vector

Type ndarray

__init__(*gas_formulas: list*, *path_to_cross_section_files: list*, *proportions: list*, *max_cross_section_energy: float*)
Instantiates a GasMixture.

Parameters

- **gas_formulas** (*list*) – list of the chemical formulas of the species
- **path_to_cross_section_files** (*list*) – path to the cross section data of each species
- **proportions** (*list*) – proportion of each species in the mixture
- **max_cross_section_energy** (*float*) – maximum cross section energy (eV)

Raises **ValueError** – If the proportions do not sum up to 1

property number_of_cross_sections
Total number of cross sections, all gases considered.

Type int

pyMETHES.monte_carlo module

Module for the MonteCarlo class.

The MonteCarlo class implements all random-number based methods to simulate the motion of electrons. The simulation time-step is determined with the null collision technique (calculate_max_coll_freq creates a lookup table for the choice of the trial collision frequency, determine_timestep calculates the time-step based on the current maximum electron energy and acceleration). The collision processes are randomly chosen based on the collision frequency of each process. The scattering angles are randomly chosen with either an isotropic or an anisotropic model.

class pyMETHES.monte_carlo.**MonteCarlo**(*cfg:* pyMETHES.config.Config)
Bases: object

Class implementing all random-number based simulation methods

config
configuration of the simulation

Type *Config*

trial_coll_freq
trial collision frequency for the null collision technique

Type float

max_coll_freq
cumulative maximum of the collision frequency as a function of the electron energy

Type interp1d

max_coll_period
array inversely proportional to the cumulative maximum of the collision frequency

Type ndarray

max_coll_period_squared
array inversely proportional to the square of the cumulative maximum of the collision frequency

Type ndarray

collision_by_electron
array of collision index for each electron (starting at 0), an index equal to the number of cross sections indicates a null collision

Type ndarray

__init__(*cfg:* pyMETHES.config.Config)
Instantiates the MonteCarlo class.

Parameters **cfg** (Config) – configuration of the simulation

calculate_max_coll_freq(*gas_mixture:* pyMETHES.gas_mixture.GasMixture)
Calculates the maximum collision frequency in the given gas mixture.

Parameters **gas_mixture** (GasMixture) – gas mixture

determine_collisions(*gas_mixture:* pyMETHES.gas_mixture.GasMixture, *velocity_norm:* *numpy.ndarray*, *energy:* *numpy.ndarray*) → None
Calculates the collision frequencies for all electrons with all cross sections, and chooses a collision type via a random number.

Parameters

- **gas_mixture** (GasMixture) – cross section data
- **velocity_norm** (*ndarray*) – norm of the velocity of each electron
- **energy** (*ndarray*) – energy of each electron

determine_timestep(*max_velocity: float*, *max_acceleration: float*) → float
Determine the duration of the next time-step in the simulation with the null-collision technique.

Parameters

- **max_velocity** (*float*) – current maximum electron velocity
- **max_acceleration** (*float*) – current maximum electron acceleration

Returns: time-step duration (s)

perform_collisions(*gas_mixture:* pyMETHES.gas_mixture.GasMixture, *position:* *numpy.ndarray*, *velocity:* *numpy.ndarray*, *energy:* *numpy.ndarray*) → Tuple[numpy.ndarray, numpy.ndarray, int, int, int]
Calculates the electrons positions (created/removed) and velocities (scattered) after the collisions listed in the collision_by_electron array.

Parameters

- **gas_mixture** (GasMixture) – cross section data
- **position** (*ndarray*) – coordinates (x,y,z) of each electron (m)
- **velocity** (*ndarray*) – velocity of each electron in (x,y,z) directions (m.s-1)
- **energy** (*ndarray*) – energy of each electron (eV)

Returns: the new position and velocity of electrons, the number of collisions, cations and anions produced

static scattering_angles(*energy: numpy.ndarray, iso: bool) -> (<class 'numpy.ndarray'>, <class 'numpy.ndarray'>, <class 'numpy.ndarray'>, <class 'numpy.ndarray'>)*

Generates values for the polar (chi) and azimuthal (phi) isotropic or anisotropic scattering angles according to Vahedi (1995).

Parameters

- **energy** – array of electron energies
- **iso** – isotropic scattering or not

Returns: 4 arrays cos(chi), sin(chi), cos(phi), sin(phi)

static unit_scattered_velocity(*energy: numpy.ndarray, velocity: numpy.ndarray, iso: bool*) → Tuple[numpy.ndarray, numpy.ndarray]

Calculates the new direction of the velocity vector after scattering.

Parameters

- **energy** (*ndarray*) – energy of each electron (eV)
- **velocity** (*ndarray*) – velocity of each electron in (x,y,z) directions before the collision (m.s-1)
- **iso** (*bool*) – isotropic scattering (True) or anisotropic scattering (False)

Returns: normed velocities after collisions, cosine of polar scattering angle

version = 'pyMETHES version 0.1.1\n'

pyMETHES.output module

Module for the Output class.

class pyMETHES.output.**Output**(*cfg: pyMETHES.config.Config, ver: str, electrons: pyMETHES.electrons.Electrons*)

Bases: object

The Output class instantiates all output-related classes: TimeSeries, TimeAveragedEnergyDistribution, FluxData, BulkData, ConvolutedRates, CountedRates. It also provides methods to save or to plot the output data. Finally, it provides the check_sst method which test if the swarm is at equilibrium based on the evolution of the mean electron energy.

config
configuration of the simulation

Type *Config*

version
version of pyMETHES as a string

Type str

time_series
temporal evolution data

Type *TimeSeries*

energy_distribution
energy distribution data

Type *TimeAveragedEnergyDistribution*

flux
flux transport data

Type *FluxData*

bulk
flux transport data

Type *BulkData*

rates_conv
convoluted rate coefficients

Type *ConvolutedRates*

rates_count
counted rate coefficients

Type *CountedRates*

__init__ (*cfg:* pyMETHES.config.Config, *ver: str*, *electrons:* pyMETHES.electrons.Electrons)
Instantiates the Output class.

Parameters

- **cfg** (Config) – configuration of the simulation
- **ver** (*str*) – version of pyMETHES as a string
- **electrons** (Electrons) – electron data

check_sst () → bool
Checks if the swarm energy is at equilibrium (steady-state). This is done by checking if the mean electron energy has dropped during the last 10% of the elapsed time, compared to the period between 80 and 90%.

Returns: True if equilibrium was reached, False otherwise

plot_energy_distribution (*show: bool = True*, *block: bool = True*) → matplotlib.figure.Figure
Produces a figure showing the time-averaged electron energy distribution function and electron energy probability function.

Parameters

- **show** (*bool*) – calls plt.show() if True, else does nothing
- **block** (*bool*) – if show is True, plt.show(block) is called

Returns: Matplotlib figure object

static plot_sst_line (*axes*, *t_sst*) → None
Plots a vertical line at the time instant where equilibrium is reached.

Parameters

- **axes** – axes to plot the line on
- **t_sst** – equilibration time

plot_temporal_evolution (*show: bool = True*, *block: bool = True*) → matplotlib.figure.Figure
Produces a figure showing the temporal evolution of the swarm. The figure contains five subplots showing the number of particles, the mean electron energy, the number of collisions, the mean electron position and the variance of electron positions.

Parameters

- **show** (*bool*) – calls plt.show() if True, else does nothing

- **block** (*bool*) – if show is True, plt.show(block) is called

Returns: Matplotlib figure object

save_energy_distribution(*name: Optional[str] = None*) → None
Saves the final time-averaged electron energy distribution function and electron energy probability function to a json file.

Parameters **name** – name of the json file created

save_swarm_parameters(*name: Optional[str] = None*) → None
Saves the final swarm parameters to a json file.

Parameters **name** – name of the json file created

save_temporal_evolution(*name: Optional[str] = None*) → None
Saves the temporal evolution of the swarm data to a json file.

Parameters **name** – name of the json file created

pyMETHES.rate_coefficients module

Module for the CountedRates and ConvolutedRates classes.

Both classes are based on the RateCoefficients abstract class. They implement two different methods for calculating ionization/attachment rate coefficients. The CountedRates class calculates the rate coefficients by averaging over time the number of ionization and attachment collisions. The ConvolutedRates class calculates the rate coefficients by calculating the convolution product of the cross section with the time-averaged electron energy distribution of the electrons.

class pyMETHES.rate_coefficients.**ConvolutedRates**
Bases: *pyMETHES.rate_coefficients.RateCoefficients*

Ionization, attachment and effective ionization rate coefficients calculated by convolution product of eedf and cross section.

static calculate_convolution(*interp*, *distri:* pyMETHES.energy_distribution.TimeAveragedEnergyDistribution) → float
Calculates the reaction rate of a collision type by calculating the convolution product of the eedf and the corresponding cross sections.

Parameters

- **interp** – linear interpolation of the cross section (m2)
- **distri** (TimeAveragedEnergyDistribution) – energy distribution of the electrons (eV-1)

Returns: reaction rate coefficient (m3.s-1)

calculate_data(*mix:* pyMETHES.gas_mixture.GasMixture, *distri:* pyMETHES.energy_distribution.TimeAveragedEnergyDistribution) → None
Calculates the effective, ionization and attachment rate coefficients.

Parameters

- **mix** (GasMixture) – GasMixture object containing the cross section data
- **distri** (TimeAveragedEnergyDistribution) – electron energy distribution (eV-1)

class pyMETHES.rate_coefficients.**CountedRates**(*gas_number_density: float*, *conserve: bool*)
Bases: *pyMETHES.rate_coefficients.RateCoefficients*

Ionization, attachment and effective ionization rate coefficients, calculated by counting ionization and attachment collisions.

gas_number_density
gas number density (m-3)

Type float

conserve
conservation of the electron number

Type bool

__init__ (*gas_number_density: float*, *conserve: bool*)
Instantiation of CountedRates.

Parameters

- **gas_number_density** (*float*) – gas number density (m-3) for normalization
- **conserve** (*bool*) – bool indicating conservation of the electron number (affects the statistics when counting the collisions)

calculate_data (*time_series:* pyMETHES.temporal_evolution.TimeSeries) → None
Calculates the ionization, attachment and effective ionization rate coefficients values by counting collision events.

Parameters **time_series** (TimeSeries) – temporal evolution of some quantities

class pyMETHES.rate_coefficients.**RateCoefficients**
Bases: abc.ABC

Ionization, attachment and effective ionization rate coefficients

ionization
ionization reaction rate coefficient (m3.s-1)

attachment
attachment reaction rate coefficient (m3.s-1)

effective
effective ionization reaction rate coefficient (m3.s-1)

pyMETHES.simulation module

Module for the Simulation class.

class pyMETHES.simulation.**Simulation**(*config: Union[str, dict]*)
Bases: object

Main class of the pyMETHES simulation tool.

The Simulation can be initialized providing the path to a configuration file, or a configuration dictionary. A different configuration can be applied at later stages using the apply_config method. A single simulation can be run with the run method, a series of simulation with the run_series method.

config
configuration of the simulation

Type *Config*

mc
Monte-Carlo methods

Type *MonteCarlo*

gas_mixture
GasMixture object containing the cross section data

Type *GasMixture*

electric_field
electric field strength (V.m-1)

Type float

electrons
electron related data

Type *Electrons*

output
output data of the simulation

Type *Output*

time_passed
Number of second that have passed since the start of the simulation (measured with `time.time()`). Needed for *pyMETHES.config.Config.timeout*. Is reset every time *run()* is called. Updated only once per iteration.

Type int

__init__(*config: Union[str, dict]*)
Instantiates a Simulation.

Parameters **config** (*str, dict*) – path to a json or json5 config file, or dictionary.

Raises **TypeError** – If config is None

advance_one_step() → None
Advances the simulation by one time step.

apply_config(*config: Optional[Union[str, dict]] = None*) → None
Applies the config to (re-)initialize all attributes of Simulation.

Parameters **config** (*str, dict*) – path to a json or json5 config file, or dictionary containing the configuration for the simulation

calculate_final_output() → None
Calculates the final output data at the end of the simulation.

collect_output_data(*dt*, *nc*, *ni*, *na*)
Collects and store current data for the simulation output.

Parameters

- **dt** (*float*) – duration of the current time-step (s)
- **nc** (*int*) – number of collisions during the current time-step
- **ni** (*int*) – number of cations produced during the current time-step
- **na** (*int*) – number of anions produced during the current time-step

end_simulation() → bool
Check end conditions for the simulation. See *pyMETHES.config.Config.end_condition_type* on what options are available.

Returns: True if simulation is finished, False otherwise.

print_step_info() → None
Prints information on the current simulation step: mean electron energy, relative error of the flux drift velocity in z direction, and relative error of the flux diffusion coefficient (maximum of x, y, z directions).

run() → None
Runs the simulation, until one of the end conditions is fulfilled.

run_series(*param: str*, *values: numpy.ndarray*) → None
Runs a series of simulations.

Parameters

- **param** (*str*) – name of the configuration parameter to be varied
- **values** (*ndarray*) – list of values of the configuration parameter for the different simulations

save() → None
Save the data specified in the configuration: pickle file of the simulation, temporal evolution of the swarm, swarm parameters, electron energy distribution.

save_pickle(*name: Optional[str] = None*) → None
Save the MonteCarlo class instance to a pickle file.

Parameters **name** – name of the pickle file created

version = 'pyMETHES version 0.1.1\n'

pyMETHES.temporal_evolution module

Module for the TimeSeries class.

class pyMETHES.temporal_evolution.**TimeSeries**(*electrons:* pyMETHES.electrons.Electrons)
Bases: object

Temporal evolution of some quantities during the simulation.

The TimeSeries class stores the temporal evolution of some quantities in arrays of ever-increasing length. The TimeSeries can be exported to a pandas DataFrame for further processing.

ind_equ
index of equilibration time

Type int

time
simulated time

Type ndarray

num_collisions
cumulative number of collisions

Type ndarray

num_electrons
number of electrons

Type ndarray

num_cations
cumulative number of cations

Type ndarray

num_anions
cumulative number of anions

Type ndarray

mean_energy
mean energy of electrons

Type ndarray

mean_position
mean position of electrons

Type ndarray

var_position
variance of electrons positions

Type ndarray

mean_velocity
mean velocity of electrons

Type ndarray

mean_velocity_moment
mean velocity moment of electrons

Type ndarray

__init__(*electrons:* pyMETHES.electrons.Electrons)
Instantiates a TimeSeries.

Parameters **electrons** (Electrons) – electron related data

append_data(*electrons:* pyMETHES.electrons.Electrons, *dt: float*, *n_coll: int*, *n_cations: int*, *n_anions: int*) → None
Appends latest data to the arrays storing the temporal evolution of some quantities.

Parameters

- **electrons** (Electrons) – current data related to the electrons
- **dt** (*float*) – duration of the current time-step (s)
- **n_coll** (*int*) – number of collisions during the current time-step
- **n_cations** (*int*) – number of cations produced during the current time-step
- **n_anions** (*int*) – number of anions produced during the current time-step

to_dataframe() → pandas.core.frame.DataFrame
Creates a pandas DataFrame with the data containe in the TimeSeries and returns it.

Returns: pandas DataFrame containing the TimeSeries data.

pyMETHES.transport_data module

Module for the FluxData and BulkData classes.

Both classes are based on the TransportData abstract class, and calculate the drift velocity and diffusion coefficient of electrons based on the temporal evolution of their position and velocity. The BulkData class calculates the bulk transport parameters, which describe the transport of the center-of-mass of the swarm (time derivative of spatial averages). The FluxData class calculates the flux transport parameters (spatial averages of time derivatives).

class pyMETHES.transport_data.**BulkData**(*gas_number_density: float*)
Bases: *pyMETHES.transport_data.TransportData*

Stores the bulk drift velocity and bulk diffusion coefficient.

w
bulk drift velocity along x, y and z directions (m.s-1)

Type ndarray

w_err
bulk drift velocity error along x, y and z directions (m.s-1)

Type ndarray

DN
bulk diffusion coeff. along x, y and z directions (m-1.s-1)

Type ndarray

DN_err
bulk diffusion coeff. error along x, y and z directions (m-1.s-1)

Type ndarray

calculate_data(*time_series:* pyMETHES.temporal_evolution.TimeSeries) → None
Calculates the bulk drift velocity and bulk diffusion coefficient values (time derivative of spatial averages).

Parameters **time_series** (TimeSeries) – temporal evolution of some quantities

class pyMETHES.transport_data.**FluxData**(*gas_number_density: float*)
Bases: *pyMETHES.transport_data.TransportData*

Flux drift velocity and flux diffusion coefficient.

w
flux drift velocity along x, y and z directions (m.s-1)

Type ndarray

w_err
flux drift velocity error along x, y and z directions (m.s-1)

Type ndarray

DN
flux diffusion coeff. along x, y and z directions (m-1.s-1)

Type ndarray

DN_err
flux diffusion coeff. error along x, y and z directions (m-1.s-1)

Type ndarray

calculate_data(*time_series:* pyMETHES.temporal_evolution.TimeSeries) → None
Calculates the flux drift velocity and the flux diffusion coefficient (spatial averages of time derivatives).

Parameters **time_series** (TimeSeries) – temporal evolution of some quantities

class pyMETHES.transport_data.**TransportData**(*gas_number_density: float*)
Bases: abc.ABC

Stores the drift velocity and diffusion coefficient.

w
drift velocity along x, y and z directions (m.s-1)

Type ndarray

w_err
drift velocity error along x, y and z directions (m.s-1)

Type ndarray

DN
diffusion coeff. along x, y and z directions (m-1.s-1)

Type ndarray

DN_err
diffusion coeff. error along x, y and z directions (m-1.s-1)

Type ndarray

pyMETHES.utils module

Module with some utility methods.

pyMETHES.utils.**acceleration_from_electric_field**(*electric_field: Union[int, float, numpy.ndarray]*) → Union[int, float, numpy.ndarray]
Calculates the acceleration of electrons, based on the local electric field.

Parameters **electric_field** – local electric field strength (V.m-1)

Returns: acceleration (m.s-2)

pyMETHES.utils.**energy_from_velocity**(*velocity: Union[int, float, numpy.ndarray]*) → Union[int, float, numpy.ndarray]
Calculate the kinetic energy of electrons, based on the norm of their velocity.

Parameters **velocity** – velocity norm of the electrons (m.s-1)

Returns: Energy of the electrons (eV)

pyMETHES.utils.**maxwell_boltzmann_eedf**(*points: Union[int, float, numpy.ndarray]*, *temperature: Union[int, float]*) → Union[int, float, numpy.ndarray]
Calculates the Maxwell-Boltzmann EEDF of the given points or point

Parameters

- **points** (*Union[int, float, np.ndarray]*) – Scalar or vector of energies in eV
- **temperature** (*Union[int, float]*) – Temperature at which to calculate the distribution

pyMETHES.utils.**maxwell_boltzmann_random**(*num: int*, *temperature: Union[float, int]*) → list
Returns a list of *num* Maxwell Boltzmann distributed values (in eV) at temperature *temperature*

Parameters

- **num** (*int*) – Number of electrons
- **temperature** (*float*) – Temperature

pyMETHES.utils.**velocity_from_energy**(*energy: Union[int, float, numpy.ndarray]*) → Union[int, float, numpy.ndarray]

Calculate the velocity norm of electrons, based on their kinetic energy.

Parameters **energy** – Energy of the electron (eV)

Returns: Velocity norm of the electrons (m.s-1)

5.1.2 Module contents

CHAPTER

SIX

CONTRIBUTING

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

You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://gitlab.com/ethz_hvl/pymethes/issues.

If you are reporting a bug, please include:

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

6.1.2 Fix Bugs

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

6.1.3 Implement Features

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

6.1.4 Write Documentation

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

6.1.5 Submit Feedback

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

If you are proposing a feature:

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

6.2 Get Started!

Ready to contribute? Here's how to set up *pyMETHES* for local development.

1. Clone *pyMETHES* repo from GitLab.

   $ git clone git@gitlab.ethz.ch:your_name_here/pyMETHES.git

2. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pyMETHES
$ cd pyMETHES/
$ python setup.py develop
$ pip install -r requirements_dev.txt
```

3. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 pyMETHES tests
$ python setup.py test # or py.test
$ tox # test all supported Python environments
```

5. Commit your changes and push your branch to GitLab:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

6. Submit a merge request through the GitLab website.

6.3 Merge Request Guidelines

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

1. The merge request should include tests.
2. If the merge request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The merge request should work for Python 3.7. Check https://gitlab.com/ethz_hvl/pymethes/merge_requests and make sure that the tests pass for all supported Python versions.

6.4 Tips

- To run tests from a single file:

```
$ py.test tests/test_sample.py
```

  or a single test function:

```
$ py.test tests/test_sample.py::test_pass
```

- To add dependency, edit appropriate `*requirements` variable in the `setup.py` file and re-run:

```
$ python setup.py develop
```

- To generate a PDF version of the Sphinx documentation instead of HTML use:

```
$ rm -rf docs/pyMETHES.rst docs/modules.rst docs/_build && sphinx-apidoc -o docs/␣
↪pyMETHES && python -msphinx -M latexpdf docs/ docs/_build
```

  This requires a local installation of a LaTeX distribution, e.g. MikTeX.

6.5 Deploying

A reminder for the maintainers on how to deploy. Create release-N.M.K branch. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
$ make release
```

Merge the release branch into master and devel branches with `--no-ff` flag.

Optionally, go to https://gitlab.com/ethz_hvl/pymethes/tags/vM.N.P/release/edit and add release notes (e.g. changes lists).

CHAPTER

SEVEN

AUTHORS

pyMETHES was written by:

- Alise Chachereau <alisec@ethz.ch>, and
- Kerry Jansen <kjansen@student.ethz.ch>, and
- Markus Niese <mniese@student.ethz.ch>, and
- Martin Vahlensieck <martinva@student.ethz.ch>

Contributors:

- Mikolaj Rybinski <mikolaj.rybinski@id.ethz.ch>

CHAPTER

EIGHT

HISTORY

8.1 0.1.0 (2020-06-19)

- Simulation in homogeneous electric field
- Parametric sweep over one configuration parameter

8.2 0.1.1 (2020-06-24)

- Fix flow diagram not appearing in documentation
- More detailed USAGE.rst and README.rst
- Set electron scattering model to isotropic in the example configuration (anisotropic scattering is untested)

8.3 0.2.0 (2021-01-20)

- **Added new configuration options:**
  - Added a timeout for the simulation
  - Option for seeding random numbers (defaut: 'random')
  - Options to specify initial energy of electrons, including Maxwell-Boltzmann distributed values
  - New end conditions to choose from (**WARNING**: the default end condition has changed, the previous behaviour can be obtained by using the provided example *run_with_custom_end_condition.py*)
- **Diverse minor fixes and improvements**
  - Replaced outdated links to previous repository
  - Raise TypeError when Simulation configuration is None
  - Diverse fixes and improvements to Makefile
  - Improved gitlab-ci.yaml (added tests for Python 3.8 and 3.9, pylint)
  - Unified docstring style to google style docstrings
  - Improved documentation and code of config.py
- Parametrized and extended tests

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

pyMETHES, 30
pyMETHES.config, 9
pyMETHES.cross_section, 12
pyMETHES.electrons, 14
pyMETHES.energy_distribution, 17
pyMETHES.gas_mixture, 18
pyMETHES.monte_carlo, 19
pyMETHES.output, 21
pyMETHES.rate_coefficients, 23
pyMETHES.simulation, 24
pyMETHES.temporal_evolution, 26
pyMETHES.transport_data, 28
pyMETHES.utils, 29

INDEX

Symbols

__init__() (*pyMETHES.config.Config method*), 12
__init__() (*pyMETHES.cross_section.InterpolatedCrossSection method*), 13
__init__() (*pyMETHES.cross_section.InterpolatedCrossSectionSet method*), 14
__init__() (*pyMETHES.electrons.Electrons method*), 15
__init__() (*pyMETHES.gas_mixture.GasMixture method*), 19
__init__() (*pyMETHES.monte_carlo.MonteCarlo method*), 20
__init__() (*pyMETHES.output.Output method*), 22
__init__() (*pyMETHES.rate_coefficients.CountedRates method*), 24
__init__() (*pyMETHES.simulation.Simulation method*), 25
__init__() (*pyMETHES.temporal_evolution.TimeSeries method*), 27
_gas_number_density (*pyMETHES.config.Config attribute*), 10
_pressure (*pyMETHES.config.Config attribute*), 10
_temperature (*pyMETHES.config.Config attribute*), 10

A

accelerate() (*pyMETHES.electrons.Electrons method*), 15
acceleration (*pyMETHES.electrons.Electrons attribute*), 14
acceleration_from_electric_field() (*in module pyMETHES.utils*), 29
acceleration_norm() (*pyMETHES.electrons.Electrons property*), 15
advance_one_step() (*pyMETHES.simulation.Simulation method*), 25
append_data() (*pyMETHES.temporal_evolution.TimeSeries method*), 27
apply_config() (*pyMETHES.simulation.Simulation method*), 25
apply_scatter() (*pyMETHES.electrons.Electrons method*), 15
attachment (*pyMETHES.rate_coefficients.RateCoefficients attribute*), 24

B

base_name (*pyMETHES.config.Config attribute*), 9
bulk (*pyMETHES.output.Output attribute*), 22
BulkData (*class in pyMETHES.transport_data*), 28

C

calculate_bin_centers() (*pyMETHES.energy_distribution.EnergyDistribution method*), 17
calculate_convolution() (*pyMETHES.rate_coefficients.ConvolutedRates static method*), 23
calculate_data() (*pyMETHES.rate_coefficients.ConvolutedRates method*), 23
calculate_data() (*pyMETHES.rate_coefficients.CountedRates method*), 24
calculate_data() (*pyMETHES.transport_data.BulkData method*), 28
calculate_data() (*pyMETHES.transport_data.FluxData method*), 28
calculate_distribution() (*pyMETHES.energy_distribution.InstantaneousEnergyDistributio method*), 17
calculate_distribution() (*pyMETHES.energy_distribution.TimeAveragedEnergyDistributio method*), 17
calculate_final_output() (*pyMETHES.simulation.Simulation method*), 25
calculate_max_coll_freq() (*pyMETHES.monte_carlo.MonteCarlo method*), 20
check_sst() (*pyMETHES.output.Output method*), 22
collect_histogram() (*pyMETHES.energy_distribution.TimeAveragedEnergyDistributio method*), 18

collect_output_data() (*pyMETHES.simulation.Simulation method*), 25
collision_by_electron (*pyMETHES.monte_carlo.MonteCarlo attribute*), 20
Config (*class in pyMETHES.config*), 9
config (*pyMETHES.monte_carlo.MonteCarlo attribute*), 19
config (*pyMETHES.output.Output attribute*), 21
config (*pyMETHES.simulation.Simulation attribute*), 24
conserve (*pyMETHES.config.Config attribute*), 11
conserve (*pyMETHES.rate_coefficients.CountedRates attribute*), 24
ConvolutedRates (*class in pyMETHES.rate_coefficients*), 23
CountedRates (*class in pyMETHES.rate_coefficients*), 23
cross_sections (*pyMETHES.cross_section.InterpolatedCrossSectionSet attribute*), 14
cross_sections (*pyMETHES.gas_mixture.GasMixture attribute*), 18
cumulative_energy_histogram (*pyMETHES.energy_distribution.TimeAveragedEnergyDistribution attribute*), 17

D

data (*pyMETHES.cross_section.InterpolatedCrossSection attribute*), 13
database (*pyMETHES.cross_section.InterpolatedCrossSection attribute*), 13
database (*pyMETHES.cross_section.InterpolatedCrossSectionSet attribute*), 13
determine_collisions() (*pyMETHES.monte_carlo.MonteCarlo method*), 20
determine_timestep() (*pyMETHES.monte_carlo.MonteCarlo method*), 20
DN (*pyMETHES.transport_data.BulkData attribute*), 28
DN (*pyMETHES.transport_data.FluxData attribute*), 28
DN (*pyMETHES.transport_data.TransportData attribute*), 29
DN_err (*pyMETHES.transport_data.BulkData attribute*), 28
DN_err (*pyMETHES.transport_data.FluxData attribute*), 28
DN_err (*pyMETHES.transport_data.TransportData attribute*), 29
DN_tol (*pyMETHES.config.Config attribute*), 11

E

eedf (*pyMETHES.energy_distribution.EnergyDistribution attribute*), 17
eepf (*pyMETHES.energy_distribution.EnergyDistribution attribute*), 17
effective (*pyMETHES.rate_coefficients.RateCoefficients attribute*), 24
effective_to_elastic() (*pyMETHES.cross_section.InterpolatedCrossSectionSet method*), 14
electric_field (*pyMETHES.simulation.Simulation attribute*), 25
Electrons (*class in pyMETHES.electrons*), 14
electrons (*pyMETHES.simulation.Simulation attribute*), 25
EN (*pyMETHES.config.Config attribute*), 10
end_condition_type (*pyMETHES.config.Config attribute*), 11
end_simulation() (*pyMETHES.simulation.Simulation method*), 25
energy() (*pyMETHES.electrons.Electrons property*), 15
energy_bin_centers (*pyMETHES.energy_distribution.EnergyDistribution attribute*), 17
energy_bins (*pyMETHES.energy_distribution.EnergyDistribution attribute*), 17
energy_distribution (*pyMETHES.output.Output attribute*), 21
energy_distribution() (*pyMETHES.electrons.Electrons property*), 15
energy_from_velocity() (*in module pyMETHES.utils*), 29
energy_sharing_factor (*pyMETHES.config.Config attribute*), 11
energy_vector (*pyMETHES.gas_mixture.GasMixture attribute*), 18
EnergyDistribution (*class in pyMETHES.energy_distribution*), 17

F

flux (*pyMETHES.output.Output attribute*), 21
FluxData (*class in pyMETHES.transport_data*), 28
fractions (*pyMETHES.config.Config attribute*), 9
free_flight() (*pyMETHES.electrons.Electrons method*), 15

G

gas_mixture (*pyMETHES.simulation.Simulation attribute*), 25
gas_number_density (*pyMETHES.rate_coefficients.CountedRates attribute*), 24
gas_number_density() (*pyMETHES.config.Config property*), 12

gases (*pyMETHES.config.Config attribute*), 9
GasMixture (*class in pyMETHES.gas_mixture*), 18
generate_bins() (*pyMETHES.energy_distribution.TimeAveragedEnergyDistribution method*), 18

I

ind_equ (*pyMETHES.temporal_evolution.TimeSeries attribute*), 26
info (*pyMETHES.cross_section.InterpolatedCrossSection attribute*), 13
initial_direction (*pyMETHES.config.Config attribute*), 11
initial_energy (*pyMETHES.config.Config attribute*), 10
initial_energy_distribution (*pyMETHES.config.Config attribute*), 10
initial_pos_electrons (*pyMETHES.config.Config attribute*), 10
initial_std_electrons (*pyMETHES.config.Config attribute*), 10
initial_temperature (*pyMETHES.config.Config attribute*), 11
InstantaneousEnergyDistribution (*class in pyMETHES.energy_distribution*), 17
InterpolatedCrossSection (*class in pyMETHES.cross_section*), 12
InterpolatedCrossSectionSet (*class in pyMETHES.cross_section*), 13
interpolation (*pyMETHES.cross_section.InterpolatedCrossSection attribute*), 13
ionization (*pyMETHES.rate_coefficients.RateCoefficients attribute*), 24
is_attachment (*pyMETHES.gas_mixture.GasMixture attribute*), 18
is_done (*pyMETHES.config.Config attribute*), 12
is_ionization (*pyMETHES.gas_mixture.GasMixture attribute*), 18
isotropic_scattering (*pyMETHES.config.Config attribute*), 11

M

mass_ratio (*pyMETHES.cross_section.InterpolatedCrossSection attribute*), 13
mass_ratios (*pyMETHES.gas_mixture.GasMixture attribute*), 18
max_acceleration_norm() (*pyMETHES.electrons.Electrons property*), 15
max_coll_freq (*pyMETHES.monte_carlo.MonteCarlo attribute*), 19
max_coll_period (*pyMETHES.monte_carlo.MonteCarlo attribute*), 19
max_coll_period_squared (*pyMETHES.monte_carlo.MonteCarlo attribute*), 20
max_energy (*pyMETHES.config.Config attribute*), 9
max_energy() (*pyMETHES.electrons.Electrons property*), 15
max_velocity_norm() (*pyMETHES.electrons.Electrons property*), 15
maxwell_boltzmann_eedf() (*in module pyMETHES.utils*), 29
maxwell_boltzmann_random() (*in module pyMETHES.utils*), 29
mc (*pyMETHES.simulation.Simulation attribute*), 24
mean_energy (*pyMETHES.temporal_evolution.TimeSeries attribute*), 27
mean_energy() (*pyMETHES.electrons.Electrons property*), 15
mean_position (*pyMETHES.temporal_evolution.TimeSeries attribute*), 27
mean_position() (*pyMETHES.electrons.Electrons property*), 15
mean_velocity (*pyMETHES.temporal_evolution.TimeSeries attribute*), 27
mean_velocity() (*pyMETHES.electrons.Electrons property*), 16
mean_velocity_moment (*pyMETHES.temporal_evolution.TimeSeries attribute*), 27
mean_velocity_moment() (*pyMETHES.electrons.Electrons property*), 16
module
    pyMETHES, 30
    pyMETHES.config, 9
    pyMETHES.cross_section, 12
    pyMETHES.electrons, 14
    pyMETHES.energy_distribution, 17
    pyMETHES.gas_mixture, 18
    pyMETHES.monte_carlo, 19
    pyMETHES.output, 21
    pyMETHES.rate_coefficients, 23
    pyMETHES.simulation, 24
    pyMETHES.temporal_evolution, 26
    pyMETHES.transport_data, 28
    pyMETHES.utils, 29
MonteCarlo (*class in pyMETHES.monte_carlo*), 19

N

num_anions (*pyMETHES.temporal_evolution.TimeSeries attribute*), 26
num_cations (*pyMETHES.temporal_evolution.TimeSeries attribute*), 26
num_col_max (*pyMETHES.config.Config attribute*), 11

num_collisions (*pyMETHES.temporal_evolution.TimeSeries attribute*), 26
num_e() (*pyMETHES.electrons.Electrons property*), 16
num_e_initial (*pyMETHES.config.Config attribute*), 10
num_e_max (*pyMETHES.config.Config attribute*), 11
num_electrons (*pyMETHES.temporal_evolution.TimeSeries attribute*), 26
num_energy_bins (*pyMETHES.config.Config attribute*), 11
number_of_cross_sections() (*pyMETHES.gas_mixture.GasMixture property*), 19

O

Output (*class in pyMETHES.output*), 21
output (*pyMETHES.simulation.Simulation attribute*), 25
output_directory (*pyMETHES.config.Config attribute*), 9

P

paths_to_cross_section_files (*pyMETHES.config.Config attribute*), 9
perform_collisions() (*pyMETHES.monte_carlo.MonteCarlo method*), 20
plot() (*pyMETHES.cross_section.InterpolatedCrossSectionSet method*), 14
plot_all() (*pyMETHES.electrons.Electrons method*), 16
plot_energy() (*pyMETHES.electrons.Electrons method*), 16
plot_energy_distribution() (*pyMETHES.output.Output method*), 22
plot_position() (*pyMETHES.electrons.Electrons method*), 16
plot_sst_line() (*pyMETHES.output.Output static method*), 22
plot_temporal_evolution() (*pyMETHES.output.Output method*), 22
plot_velocity() (*pyMETHES.electrons.Electrons method*), 16
position (*pyMETHES.electrons.Electrons attribute*), 14
pressure() (*pyMETHES.config.Config property*), 12
print_step_info() (*pyMETHES.simulation.Simulation method*), 25
pyMETHES
    module, 30
pyMETHES.config
    module, 9
pyMETHES.cross_section
    module, 12
pyMETHES.electrons
    module, 14
pyMETHES.energy_distribution
    module, 17
pyMETHES.gas_mixture
    module, 18
pyMETHES.monte_carlo
    module, 19
pyMETHES.output
    module, 21
pyMETHES.rate_coefficients
    module, 23
pyMETHES.simulation
    module, 24
pyMETHES.temporal_evolution
    module, 26
pyMETHES.transport_data
    module, 28
pyMETHES.utils
    module, 29

R

RateCoefficients (*class in pyMETHES.rate_coefficients*), 24
rates_conv (*pyMETHES.output.Output attribute*), 22
rates_count (*pyMETHES.output.Output attribute*), 22
reset_cache() (*pyMETHES.electrons.Electrons method*), 16
run() (*pyMETHES.simulation.Simulation method*), 26
run_series() (*pyMETHES.simulation.Simulation method*), 26

S

save() (*pyMETHES.simulation.Simulation method*), 26
save_energy_distribution (*pyMETHES.config.Config attribute*), 10
save_energy_distribution() (*pyMETHES.output.Output method*), 23
save_json() (*pyMETHES.config.Config method*), 12
save_json5() (*pyMETHES.config.Config method*), 12
save_pickle() (*pyMETHES.simulation.Simulation method*), 26
save_simulation_pickle (*pyMETHES.config.Config attribute*), 9
save_swarm_parameters (*pyMETHES.config.Config attribute*), 10
save_swarm_parameters() (*pyMETHES.output.Output method*), 23
save_temporal_evolution (*pyMETHES.config.Config attribute*), 10

save_temporal_evolution() (*pyMETHES.output.Output method*), 23
scattering_angles() (*pyMETHES.monte_carlo.MonteCarlo static method*), 20
seed (*pyMETHES.config.Config attribute*), 11
Simulation (*class in pyMETHES.simulation*), 24
species (*pyMETHES.cross_section.InterpolatedCrossSection attribute*), 13
species (*pyMETHES.cross_section.InterpolatedCrossSectionSet attribute*), 13
std_energy() (*pyMETHES.electrons.Electrons property*), 16

T

temperature() (*pyMETHES.config.Config property*), 12
threshold (*pyMETHES.cross_section.InterpolatedCrossSection attribute*), 13
thresholds (*pyMETHES.gas_mixture.GasMixture attribute*), 18
time (*pyMETHES.temporal_evolution.TimeSeries attribute*), 26
time_passed (*pyMETHES.simulation.Simulation attribute*), 25
time_series (*pyMETHES.output.Output attribute*), 21
TimeAveragedEnergyDistribution (*class in pyMETHES.energy_distribution*), 17
timeout (*pyMETHES.config.Config attribute*), 12
TimeSeries (*class in pyMETHES.temporal_evolution*), 26
to_dataframe() (*pyMETHES.temporal_evolution.TimeSeries method*), 27
to_dict() (*pyMETHES.config.Config method*), 12
total_cross_section (*pyMETHES.gas_mixture.GasMixture attribute*), 19
TransportData (*class in pyMETHES.transport_data*), 29
trial_coll_freq (*pyMETHES.monte_carlo.MonteCarlo attribute*), 19
type (*pyMETHES.cross_section.InterpolatedCrossSection attribute*), 12
types (*pyMETHES.gas_mixture.GasMixture attribute*), 18

U

unit_scattered_velocity() (*pyMETHES.monte_carlo.MonteCarlo static method*), 21

V

var_position (*pyMETHES.temporal_evolution.TimeSeries attribute*), 27
var_position() (*pyMETHES.electrons.Electrons property*), 16
velocity (*pyMETHES.electrons.Electrons attribute*), 14
velocity_from_energy() (*in module pyMETHES.utils*), 30
velocity_norm() (*pyMETHES.electrons.Electrons property*), 16
version (*pyMETHES.monte_carlo.MonteCarlo attribute*), 21
version (*pyMETHES.output.Output attribute*), 21
version (*pyMETHES.simulation.Simulation attribute*), 26

W

w (*pyMETHES.transport_data.BulkData attribute*), 28
w (*pyMETHES.transport_data.FluxData attribute*), 28
w (*pyMETHES.transport_data.TransportData attribute*), 29
w_err (*pyMETHES.transport_data.BulkData attribute*), 28
w_err (*pyMETHES.transport_data.FluxData attribute*), 28
w_err (*pyMETHES.transport_data.TransportData attribute*), 29
w_tol (*pyMETHES.config.Config attribute*), 11