

---

# **py-droplets Documentation**

*Release 0.10.0*

**David Zwicker**

**Jun 03, 2022**



# CONTENTS

<b>1</b>	<b>Contents</b>	<b>3</b>
1.1	Installation	3
1.1.1	Installing from source	3
1.1.1.1	Prerequisites	3
1.1.1.2	Downloading the package	4
1.2	Getting started	4
1.2.1	Examples	4
1.2.1.1	Basic droplets	4
1.2.1.2	Plotting emulsions	4
1.2.1.3	Analyze images	5
1.2.2	Contributing code	6
1.2.2.1	Structure of the package	6
1.2.2.2	Extending functionality	6
1.2.2.3	Coding style	6
1.2.2.4	Running unit tests	6
1.3	droplets package	6
1.3.1	Subpackages	7
1.3.1.1	droplets.tools package	7
1.3.2	droplets.conftest module	11
1.3.3	droplets.droplet_tracks module	11
1.3.4	droplets.droplets module	17
1.3.5	droplets.emulsions module	24
1.3.6	droplets.image_analysis module	29
1.3.7	droplets.trackers module	32
1.3.8	droplets.version module	34
<b>2</b>	<b>Indices and tables</b>	<b>35</b>
	<b>Python Module Index</b>	<b>37</b>
	<b>Index</b>	<b>39</b>



The *py-droplets* python package provides methods and classes useful for studying phase separation phenomena using phase field methods.



## CONTENTS

### 1.1 Installation

This *py-droplets* package is developed for python 3.7+ and should run on all common platforms. The code is tested under Linux, Windows, and macOS.

Since the package is available on [pypi](#), the installation is in principle as simple as running

```
pip install py-droplets
```

In order to have all features of the package available, you might also want to install the following optional packages:

```
pip install h5py pyfftw tqdm
```

#### 1.1.1 Installing from source

Installing from source can be necessary if the pypi installation does not work or if the latest source code should be installed from github.

##### 1.1.1.1 Prerequisites

The code builds on other python packages, which need to be installed for *py-droplets* to function properly. The required packages are listed in the table below:

Package	Version	Usage
matplotlib	>= 3.1.0	Visualizing results
numpy	>=1.16	Array library used for storing data
numba	>=0.43	Just-in-time compilation to accelerate numerics
scipy	>=1.2	Miscellaneous scientific functions
sympy	>=1.4	Dealing with user-defined mathematical expressions
py-pde	>=0.14	Simulating partial differential equations

These package can be installed via your operating system's package manager, e.g. using **macports**, **homebrew**, **conda**, or **pip**. The package versions given above are minimal requirements, although this is not tested systematically. Generally, it should help to install the latest version of the package. The *py-pde* package is available on *pip*, but if this is inconvenient the package can also be installed from github sources, as [described in its documentation](#).

A small subset of the package will only be available if extra optional packages are installed. Currently, this only concerns the *h5py* package for reading hdf files.

### 1.1.1.2 Downloading the package

The package can be simply checked out from [github.com/zwicker-group/py-droplets](https://github.com/zwicker-group/py-droplets). To import the package from any python session, it might be convenient to include the root folder of the package into the `PYTHONPATH` environment variable.

This documentation can be built by calling the `make html` in the `docs` folder. The final documentation will be available in `docs/build/html`. Note that a LaTeX documentation can be build using `make latexpdf`.

## 1.2 Getting started

### 1.2.1 Examples

We here collect examples for using the package to demonstrate some of its functionality.

#### 1.2.1.1 Basic droplets

The basic droplet classes can be used as follows

```
from droplets import DiffuseDroplet, Emulsion, SphericalDroplet

# construct two droplets
drop1 = SphericalDroplet(position=[0, 0], radius=2)
drop2 = DiffuseDroplet(position=[6, 8], radius=3, interface_width=1)

# check whether they overlap
print(drop1.overlaps(drop2)) # prints False

# construct an emulsion and query it
e = Emulsion([drop1, drop2])
e.get_size_statistics()
```

We first create two droplets represented by two different classes. The basic class `SphericalDroplet` represents a droplet by its position and radius, while the more advanced class `DiffuseDroplet` also keeps track of the interface width. Finally, we combine the two droplets in an emulsion, which then allows further analysis.

#### 1.2.1.2 Plotting emulsions

To visualize an emulsions, one can simply use the `plot()`:

```
import numpy as np

from droplets import DiffuseDroplet, Emulsion

# create 10 random droplets
droplets = [
    DiffuseDroplet(
        position=np.random.uniform(0, 100, 2),
        radius=np.random.uniform(5, 10),
        interface_width=1,
    )
    for _ in range(10)
```

(continues on next page)



(continued from previous page)

```
]
# remove overlapping droplets in emulsion and plot it
emulsion = Emulsion(droplets)
emulsion.remove_overlapping()
emulsion.plot()
```

Note that the emulsion class can also keep track of the space in which droplets are defined, e.g, the boundaries of a simulation grid. For this, the *Emulsion* supports the *grid* argument, which can for instance be an instance of *CartesianGrid*.

### 1.2.1.3 Analyze images

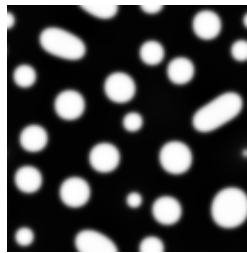


Fig. 1: An emulsion image

The package also allows analyzing images of emulsions like the one shown on the right. The code below loads the image, locates the droplets, and then displays some of their properties

```
from pathlib import Path

from pde.fields import ScalarField

from droplets.image_analysis import locate_droplets

img_path = Path(__file__).parent / "resources" / "emulsion.png"
field = ScalarField.from_image(img_path)
emulsion = locate_droplets(field)

# visualize the result
emulsion.plot(field=field, fill=False, color="w")
```

Note that the determined positions and sizes of the droplets are only roughly determined by default. If more accurate data is desired, *locate\_droplets()* supports the *refine* arguments, which fits the model image of droplet to the actual image to obtain more accurate parameter estimates.

## 1.2.2 Contributing code

### 1.2.2.1 Structure of the package

The functionality of the *droplets* package is split into multiple modules. In particular, we distinguish classes that deal with single droplets from those classes that represent collections (emulsions). The functions analyzing images are collected in a separate module.

### 1.2.2.2 Extending functionality

All code is build on a modular basis, making it easy to introduce new classes that integrate with the rest of the package. For instance, it is simple to define a droplet class that stores additional information by subclassing *SphericalDroplet*.

### 1.2.2.3 Coding style

The coding style is enforced using *isort* and *black*. Moreover, we use *Google Style docstrings*, which might be best *learned by example*. The documentation, including the docstrings, are written using *reStructuredText*, with examples in the following *cheatsheet*. To ensure the integrity of the code, we also try to provide many test functions, which are typically contained in separate modules in sub-packages called *tests*. These tests can be ran using scripts in the *tests* subfolder in the root folder. This folder also contain a script *tests\_types.sh*, which uses *mypy* to check the consistency of the python type annotations. We use these type annotations for additional documentation and they have also already been useful for finding some bugs.

### 1.2.2.4 Running unit tests

The *droplets* package contains several unit tests, typically contained in sub-module *tests* in the folder of a given module. These tests ensure that basic functions work as expected, in particular when code is changed in future versions. To run all tests, there are a few convenience scripts in the root directory *tests*. The most basic script is *tests\_run.sh*, which uses *pytest* to run the tests in the sub-modules of the *droplets* package. Clearly, the python package *pytest* needs to be installed. There are also additional scripts that for instance run tests in parallel (need the python package *pytest-xdist* installed), measure test coverage (need package *pytest-cov* installed), and make simple performance measurements. Moreover, there is a script *test\_types.sh*, which uses *mypy* to check the consistency of the python type annotations and there is a script *codestyle.sh*, which checks the coding style.

Before committing a change to the code repository, it is good practice to run the tests, check the type annotations, and the coding style with the scripts described above.

## 1.3 droplets package

Functions and classes for analyzing emulsions and droplets

## 1.3.1 Subpackages

### 1.3.1.1 droplets.tools package

#### droplets.tools.spherical module

Module collecting functions for handling spherical geometry

The coordinate systems use the following convention for polar coordinates  $(r, \phi)$ , where  $r$  is the radial coordinate and  $\phi$  is the polar angle:

$$\begin{cases} x = r \cos(\phi) \\ y = r \sin(\phi) \end{cases} \quad \text{for } r \in [0, \infty] \text{ and } \phi \in [0, 2\pi)$$

Similarly, for spherical coordinates  $(r, \theta, \phi)$ , where  $r$  is the radial coordinate,  $\theta$  is the azimuthal angle, and  $\phi$  is the polar angle, we use

$$\begin{cases} x = r \sin(\theta) \cos(\phi) \\ y = r \sin(\theta) \sin(\phi) \\ z = r \cos(\theta) \end{cases} \quad \text{for } r \in [0, \infty], \theta \in [0, \pi], \text{ and } \phi \in [0, 2\pi)$$

The module also provides functions for handling spherical harmonics. These spherical harmonics are described by the degree  $l$  and the order  $m$  or, alternatively, by the mode  $k$ . The relation between these values is

$$k = l(l + 1) + m$$

and

$$\begin{aligned} l &= \text{floor}(\sqrt{k}) \\ m &= k - l(l + 1) \end{aligned}$$

We will use these indices interchangeably, although the mode  $k$  is preferred internally. Note that we also consider axisymmetric spherical harmonics, where the order is always zero and the degree  $l$  and the mode  $k$  are thus identical.

<code>radius_from_volume</code>	Return the radius of a sphere with a given volume
<code>volume_from_radius</code>	Return the volume of a sphere with a given radius
<code>surface_from_radius</code>	Return the surface area of a sphere with a given radius
<code>radius_from_surface</code>	Return the radius of a sphere with a given surface area
<code>make_radius_from_volume_compiled</code>	Return a function calculating the radius of a sphere with a given volume
<code>make_volume_from_radius_compiled</code>	Return a function calculating the volume of a sphere with a given radius
<code>make_surface_from_radius_compiled</code>	Return a function calculating the surface area of a sphere
<code>points_cartesian_to_spherical</code>	Convert points from Cartesian to spherical coordinates
<code>points_spherical_to_cartesian</code>	Convert points from spherical to Cartesian coordinates
<code>spherical_index_k</code>	returns the mode $k$ from the degree $degree$ and order $order$
<code>spherical_index_lm</code>	returns the degree $l$ and the order $m$ from the mode $k$
<code>spherical_index_count</code>	return the number of modes for all indices $\leq l$
<code>spherical_index_count_optimal</code>	checks whether the modes captures all orders for maximal degree
<code>spherical_harmonic_symmetric</code>	axisymmetric spherical harmonics with degree $degree$ , so $m=0$ .
<code>spherical_harmonic_real</code>	real spherical harmonics of degree $l$ and order $m$
<code>spherical_harmonic_real_k</code>	real spherical harmonics described by mode $k$

**make\_radius\_from\_volume\_compiled** (*dim: int*) → Callable[[TNumArr], TNumArr]

Return a function calculating the radius of a sphere with a given volume

**Parameters**

**dim** (*int*) – Dimension of the space

**Returns**

A function that takes a volume and returns the radius

**Return type**

function

**make\_surface\_from\_radius\_compiled** (*dim: int*) → Callable[[TNumArr], TNumArr]

Return a function calculating the surface area of a sphere

**Parameters**

**dim** (*int*) – Dimension of the space

**Returns**

A function that takes a radius and returns the surface area

**Return type**

function

**make\_volume\_from\_radius\_compiled** (*dim: int*) → Callable[[TNumArr], TNumArr]

Return a function calculating the volume of a sphere with a given radius

**Parameters**

**dim** (*int*) – Dimension of the space

**Returns**

A function that takes a radius and returns the volume

**Return type**

function

**points\_cartesian\_to\_spherical** (*points: ndarray*) → ndarray

Convert points from Cartesian to spherical coordinates

**Parameters**

**points** (*ndarray*) – Points in Cartesian coordinates

**Returns**

Points (r,  $\theta$ ,  $\varphi$ ) in spherical coordinates

**Return type**

*ndarray*

**points\_spherical\_to\_cartesian** (*points: ndarray*) → ndarray

Convert points from spherical to Cartesian coordinates

**Parameters**

**points** (*ndarray*) – Points in spherical coordinates (r,  $\theta$ ,  $\varphi$ )

**Returns**

Points in Cartesian coordinates

**Return type**

*ndarray*

**radius\_from\_surface** (*surface: TNumArr, dim: int*) → TNumArr

Return the radius of a sphere with a given surface area

**Parameters**

- **surface** (float or `ndarray`) – Surface area of the sphere
- **dim** (`int`) – Dimension of the space

**Returns**

Radius of the sphere

**Return type**

float or `ndarray`

**radius\_from\_volume** (*volume: TNumArr, dim: int*) → TNumArr

Return the radius of a sphere with a given volume

**Parameters**

- **volume** (float or `ndarray`) – Volume of the sphere
- **dim** (`int`) – Dimension of the space

**Returns**

Radius of the sphere

**Return type**

float or `ndarray`

**spherical\_harmonic\_real** (*degree: int, order: int,  $\theta$ : float,  $\varphi$ : float*) → float

real spherical harmonics of degree  $l$  and order  $m$

**Parameters**

- **degree** (`int`) – Degree  $l$  of the spherical harmonics
- **order** (`int`) – Order  $m$  of the spherical harmonics
- **$\theta$**  (`float`) – Azimuthal angle (in  $[0, \pi]$ ) at which the spherical harmonics is evaluated.
- **$\varphi$**  (`float`) – Polar angle (in  $[0, 2\pi]$ ) at which the spherical harmonics is evaluated.

**Returns**

The value of the spherical harmonics

**Return type**

float

**spherical\_harmonic\_real\_k** (*k: int,  $\theta$ : float,  $\varphi$ : float*) → float

real spherical harmonics described by mode  $k$

**Parameters**

- **k** (`int`) – Combined index determining the degree and order of the spherical harmonics
- **$\theta$**  (`float`) – Azimuthal angle (in  $[0, \pi]$ ) at which the spherical harmonics is evaluated.
- **$\varphi$**  (`float`) – Polar angle (in  $[0, 2\pi]$ ) at which the spherical harmonics is evaluated.

**Returns**

The value of the spherical harmonics

**Return type**

float

**spherical\_harmonic\_symmetric** (*degree: int,  $\theta$ : float*)  $\rightarrow$  float

axisymmetric spherical harmonics with degree *degree*, so  $m=0$ .

**Parameters**

- **degree** (*int*) – Degree of the spherical harmonics
- **$\theta$**  (*float*) – Azimuthal angle at which the spherical harmonics is evaluated (in  $[0, \pi]$ )

**Returns**

The value of the spherical harmonics

**Return type**

float

**spherical\_index\_count** (*l: int*)  $\rightarrow$  int

return the number of modes for all indices  $\leq l$

The returned value is one less than the maximal mode  $k$  required.

**Parameters**

- **l** (*int*) – Maximal degree of the spherical harmonics

**Returns**

The number of modes

**Return type**

int

**spherical\_index\_count\_optimal** (*k\_count: int*)  $\rightarrow$  bool

checks whether the modes captures all orders for maximal degree

**Parameters**

- **k\_count** (*int*) – The number of modes considered

**spherical\_index\_k** (*degree: int, order: int = 0*)  $\rightarrow$  int

returns the mode  $k$  from the degree *degree* and order *order*

**Parameters**

- **degree** (*int*) – Degree of the spherical harmonics
- **order** (*int*) – Order of the spherical harmonics

**Raises**

**ValueError** – if  $order < -degree$  or  $order > degree$

**Returns**

a combined index  $k$

**Return type**

int

**spherical\_index\_lm** (*k: int*)  $\rightarrow$  Tuple[int, int]

returns the degree  $l$  and the order  $m$  from the mode  $k$

**Parameters**

- **k** (*int*) – The combined index for the spherical harmonics

**Returns**

The degree  $l$  and order  $m$  of the spherical harmonics associated with the combined index

**Return type**

tuple

**surface\_from\_radius** (*radius: TNumArr, dim: int*) → TNumArr

Return the surface area of a sphere with a given radius

**Parameters**

- **radius** (float or `ndarray`) – Radius of the sphere
- **dim** (`int`) – Dimension of the space

**Returns**

Surface area of the sphere

**Return type**

float or `ndarray`

## 1.3.2 droplets.conftest module

## 1.3.3 droplets.droplet\_tracks module

Classes representing the time evolution of droplets

<code>DropletTrack</code>	information about a single droplet over multiple time steps
<code>DropletTrackList</code>	a list of instances of <code>DropletTrack</code>

**class DropletTrack** (*droplets=None, times=None*)

Bases: `object`

information about a single droplet over multiple time steps

**Parameters**

- **emulsions** (`list`) – List of emulsions that describe this time course
- **times** (`list`) – Times associated with the emulsions

**append** (*droplet: SphericalDroplet, time: float = None*) → `None`

append a new droplet with a time code

**Parameters**

- **droplet** (*droplets.droplets.SphericalDroplet*) – the droplet
- **time** (`float, optional`) – The time point

**property data:** `Optional[ndarray]`

an array containing the data of the full track

**Type**

`ndarray`

**property dim:** `Optional[int]`

return the space dimension of the droplets

**property duration:** `float`

total duration of the track

**Type**

`float`

**property end:** `float`

last time point

**Type**

`float`

**property first:** `SphericalDroplet`

first droplet instance

**Type**

`SphericalDroplet`

**classmethod from\_file** (*path: str*) → `DropletTrack`

create droplet track by reading from file

**Parameters**

**path** (*str*) – The path from which the data is read. This function assumes that the data was written as an HDF5 file using `to_file()`.

**get\_position** (*time: float*) → `ndarray`

`ndarray`: returns the droplet position at a specific time

**get\_radii** () → `ndarray`

`ndarray`: returns the droplet radius for each time point

**get\_trajectory** (*smoothing: float = 0*) → `ndarray`

return a list of positions over time

**Parameters**

**smoothing** (*float*) – Determines the length scale for some gaussian smoothing of the trajectory. Setting this to zero disables smoothing.

**Returns**

**An array giving the position of the droplet at each time instance**

**Return type**

`ndarray`

**get\_volumes** () → `ndarray`

`ndarray`: returns the droplet volume for each time point

**items** ()

iterate over all times and droplets, returning them in pairs

**property last:** `SphericalDroplet`

last droplet instance

**Type**

`SphericalDroplet`

**plot** (*attribute: str = 'radius', \*args, title: str = None, filename: str = None, action: str = 'auto', ax\_style: Optional[Dict[str, Any]] = None, fig\_style: Optional[Dict[str, Any]] = None, ax=None, \*\*kwargs*) → `PlotReference`

plot the time evolution of the droplet

**Parameters**

- **attribute** (*str*) – The attribute to plot. Typical values include *radius* and *volume*, but others might be defined on the droplet class.



- **title** (*str*) – Title of the plot. If omitted, the title might be chosen automatically.
- **filename** (*str*, *optional*) – If given, the plot is written to the specified file.
- **action** (*str*) – Decides what to do with the final figure. If the argument is set to *show*, `matplotlib.pyplot.show()` will be called to show the plot. If the value is *none*, the figure will be created, but not necessarily shown. The value *close* closes the figure, after saving it to a file when *filename* is given. The default value *auto* implies that the plot is shown if it is not a nested plot call.
- **ax\_style** (*dict*) – Dictionary with properties that will be changed on the axis after the plot has been drawn by calling `matplotlib.pyplot.setp()`. A special item in this dictionary is *use\_offset*, which is flag that can be used to control whether offset are shown along the axes of the plot.
- **fig\_style** (*dict*) – Dictionary with properties that will be changed on the figure after the plot has been drawn by calling `matplotlib.pyplot.setp()`. For instance, using `fig_style={'dpi': 200}` increases the resolution of the figure.
- **ax** (`matplotlib.axes.Axes`) – Figure axes to be used for plotting. The special value “create” creates a new figure, while “reuse” attempts to reuse an existing figure, which is the default.
- **\*\*kwargs** – All remaining parameters are forwarded to the *ax.plot* method. For example, passing *color=None*, will use different colors for different droplets.

**Returns**

Information about the plot

**Return type**

`PlotReference`

**plot\_positions** (*grid*: *Optional*[*GridBase*] = *None*, *arrow*: *bool* = *True*, *\*args*, *title*: *str* = *None*, *filename*: *str* = *None*, *action*: *str* = 'auto', *ax\_style*: *Optional*[*Dict*[*str*, *Any*]] = *None*, *fig\_style*: *Optional*[*Dict*[*str*, *Any*]] = *None*, *ax=None*, *\*\*kwargs*) → `PlotReference`

plot the droplet track

**Parameters**

- **grid** (*GridBase*, *optional*) – The grid on which the droplets are defined. If given, periodic boundary conditions can be respected in the plotting.
- **arrow** (*bool*, *optional*) – Flag determining whether an arrow head is shown to indicate the direction of the droplet drift.
- **title** (*str*) – Title of the plot. If omitted, the title might be chosen automatically.
- **filename** (*str*, *optional*) – If given, the plot is written to the specified file.
- **action** (*str*) – Decides what to do with the final figure. If the argument is set to *show*, `matplotlib.pyplot.show()` will be called to show the plot. If the value is *none*, the figure will be created, but not necessarily shown. The value *close* closes the figure, after saving it to a file when *filename* is given. The default value *auto* implies that the plot is shown if it is not a nested plot call.
- **ax\_style** (*dict*) – Dictionary with properties that will be changed on the axis after the plot has been drawn by calling `matplotlib.pyplot.setp()`. A special item in this dictionary is *use\_offset*, which is flag that can be used to control whether offset are shown along the axes of the plot.

- **fig\_style** (*dict*) – Dictionary with properties that will be changed on the figure after the plot has been drawn by calling `matplotlib.pyplot.setp()`. For instance, using `fig_style={'dpi': 200}` increases the resolution of the figure.
- **ax** (`matplotlib.axes.Axes`) – Figure axes to be used for plotting. The special value “create” creates a new figure, while “reuse” attempts to reuse an existing figure, which is the default.
- **\*\*kwargs** – Additional keyword arguments are passed to the matplotlib plot function to affect the appearance. For example, passing `color=None`, will use different colors for different droplets.

**Returns**

Information about the plot

**Return type**

`PlotReference`

**property start:** `float`

first time point

**Type**

`float`

**time\_overlaps** (*other: DropletTrack*) → `bool`

determine whether two `DropletTrack` instances overlaps in time

**Parameters**

**other** (`DropletTrack`) – The other droplet track

**Returns**

True when both tracks contain droplets at the same time step

**Return type**

`bool`

**to\_file** (*path: str, info: Optional[Dict[str, Any]] = None*) → `None`

store data in hdf5 file

The data can be read using the classmethod `DropletTrack.from_file()`.

**Parameters**

- **path** (*str*) – The path to which the data is written as an HDF5 file.
- **info** (*dict*) – Additional data stored alongside the droplet track list

**class DropletTrackList** (*iterable=(), /*)

Bases: `list`

a list of instances of `DropletTrack`

**classmethod from\_emulsion\_time\_course** (*time\_course: EmulsionTimeCourse, method: str = 'overlap', progress: bool = False, \*\*kwargs*) → `DropletTrackList`

obtain droplet tracks from an emulsion time course

**Parameters**

- **time\_course** (*droplets.emulsions.EmulsionTimeCourse*) – A collection of temporally arranged emulsions

- **method** (*str*) – The method used for tracking droplet identities. Possible methods are “overlap” (adding droplets that overlap with those in previous frames) and “distance” (matching droplets to minimize center-to-center distances).
- **progress** (*bool*) – Whether to show the progress of the process.
- **\*\*kwargs** – Additional parameters for the tracking algorithm. Currently, one can only specify a maximal distance (using *max\_dist*) for the “distance” method.

**Returns**

the resulting droplet tracks

**Return type**

*DropletTrackList*

**classmethod from\_file** (*path: str*) → *DropletTrackList*

create droplet track list by reading file

**Parameters**

**path** (*str*) – The path from which the data is read. This function assumes that the data was written as an HDF5 file using *to\_file()*.

**Returns**

an instance describing the droplet track list

**Return type**

*DropletTrackList*

**classmethod from\_storage** (*storage: StorageBase, refine: bool = False, method: str = 'overlap', progress: Optional[bool] = None*) → *DropletTrackList*

obtain droplet tracks from stored scalar field data

This method first determines an emulsion time course and then collects tracks by tracking droplets.

**Parameters**

- **storage** (*StorageBase*) – The phase fields for many time instances
- **refine** (*bool*) – Flag determining whether the droplet properties should be refined using fitting. This is a potentially slow procedure.
- **method** (*str*) – The method used for tracking droplet identities. Possible methods are “overlap” (adding droplets that overlap with those in previous frames) and “distance” (matching droplets to minimize center-to-center distances).
- **progress** (*bool*) – Whether to show the progress of the process. If *None*, the progress is not shown, except for the first step if *refine* is *True*.

**Returns**

the resulting droplet tracks

**Return type**

*DropletTrackList*

**plot** (*attribute: str = 'radius', \*args, title: str = None, filename: str = None, action: str = 'auto', ax\_style: Optional[Dict[str, Any]] = None, fig\_style: Optional[Dict[str, Any]] = None, ax=None, \*\*kwargs*) → *PlotReference*

plot the time evolution of all droplets

**Parameters**

- **attribute** (*str*) – The attribute to plot. Typical values include *radius* and *volume*, but others might be defined on the droplet class.

- **title** (*str*) – Title of the plot. If omitted, the title might be chosen automatically.
- **filename** (*str*, *optional*) – If given, the plot is written to the specified file.
- **action** (*str*) – Decides what to do with the final figure. If the argument is set to *show*, `matplotlib.pyplot.show()` will be called to show the plot. If the value is *none*, the figure will be created, but not necessarily shown. The value *close* closes the figure, after saving it to a file when *filename* is given. The default value *auto* implies that the plot is shown if it is not a nested plot call.
- **ax\_style** (*dict*) – Dictionary with properties that will be changed on the axis after the plot has been drawn by calling `matplotlib.pyplot.setp()`. A special item in this dictionary is *use\_offset*, which is flag that can be used to control whether offset are shown along the axes of the plot.
- **fig\_style** (*dict*) – Dictionary with properties that will be changed on the figure after the plot has been drawn by calling `matplotlib.pyplot.setp()`. For instance, using `fig_style={'dpi': 200}` increases the resolution of the figure.
- **ax** (`matplotlib.axes.Axes`) – Figure axes to be used for plotting. The special value “create” creates a new figure, while “reuse” attempts to reuse an existing figure, which is the default.
- **\*\*kwargs** – Additional keyword arguments are passed to the matplotlib plot function to affect the appearance. The special value *color="cycle"* implies that the default color cycle is used for the tracks, using different colors for different tracks.

#### Returns

Information about the plot

#### Return type

`PlotReference`

**plot\_positions** (*\*args*, *title: str = None*, *filename: str = None*, *action: str = 'auto'*, *ax\_style: Optional[Dict[str, Any]] = None*, *fig\_style: Optional[Dict[str, Any]] = None*, *ax=None*, *\*\*kwargs*) → `PlotReference`

plot all droplet tracks

#### Parameters

- **title** (*str*) – Title of the plot. If omitted, the title might be chosen automatically.
- **filename** (*str*, *optional*) – If given, the plot is written to the specified file.
- **action** (*str*) – Decides what to do with the final figure. If the argument is set to *show*, `matplotlib.pyplot.show()` will be called to show the plot. If the value is *none*, the figure will be created, but not necessarily shown. The value *close* closes the figure, after saving it to a file when *filename* is given. The default value *auto* implies that the plot is shown if it is not a nested plot call.
- **ax\_style** (*dict*) – Dictionary with properties that will be changed on the axis after the plot has been drawn by calling `matplotlib.pyplot.setp()`. A special item in this dictionary is *use\_offset*, which is flag that can be used to control whether offset are shown along the axes of the plot.
- **fig\_style** (*dict*) – Dictionary with properties that will be changed on the figure after the plot has been drawn by calling `matplotlib.pyplot.setp()`. For instance, using `fig_style={'dpi': 200}` increases the resolution of the figure.
- **ax** (`matplotlib.axes.Axes`) – Figure axes to be used for plotting. The special value “create” creates a new figure, while “reuse” attempts to reuse an existing figure, which is the default.

- **\*\*kwargs** – Additional keyword arguments are passed to the matplotlib plot function to affect the appearance.

**Returns**

Information about the plot

**Return type**

`PlotReference`

**remove\_short\_tracks** (*min\_duration: float = 0*) → `None`

remove tracks that a shorter than a minimal duration

**Parameters**

**min\_duration** (*float*) – The minimal duration a droplet track must have in order to be retained. This is measured in actual time and not in the number of time steps stored in the track.

**to\_file** (*path: str, info: Optional[Dict[str, Any]] = None*) → `None`

store data in hdf5 file

The data can be read using the classmethod `DropletTrackList.from_file()`.

**Parameters**

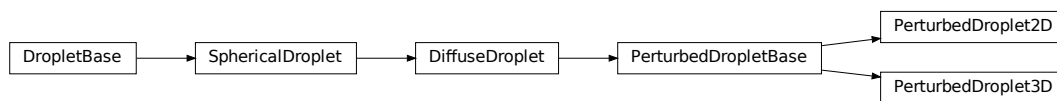
- **path** (*str*) – The path to which the data is written as an HDF5 file.
- **info** (*dict*) – Additional data stored alongside the droplet track list

### 1.3.4 droplets.droplets module

Classes representing (perturbed) droplets in various dimensions

<code>SphericalDroplet</code>	Represents a single, spherical droplet
<code>DiffuseDroplet</code>	Represents a single, spherical droplet with a diffuse interface
<code>PerturbedDroplet2D</code>	Represents a single droplet in two dimensions with a perturbed shape
<code>PerturbedDroplet3D</code>	Represents a single droplet in two dimensions with a perturbed shape

Inheritance structure of the classes:



The details of the classes are explained below:

**class DiffuseDroplet** (*position: ndarray, radius: float, interface\_width: float = None*)

Bases: `SphericalDroplet`

Represents a single, spherical droplet with a diffuse interface

**Parameters**

- **position** (`ndarray`) – Position of the droplet center
- **radius** (`float`) – Radius of the droplet
- **interface\_width** (`float`, *optional*) – Width of the interface

**data:** `recarray`

**property data\_bounds:** `Tuple[ndarray, ndarray]`

lower and upper bounds on the parameters

**Type**

`tuple`

**classmethod get\_dtype** (*\*\*kwargs*)

determine the dtype representing this droplet class

**Parameters**

**position** (`ndarray`) – The position vector of the droplet. This is used to determine the space dimension.

**Returns**

the (structured) dtype associated with this class

**Return type**

`numpy.dtype`

**property interface\_width:** `Optional[float]`

the width of the interface of this droplet

**Type**

`float`

**class PerturbedDroplet2D** (*position: ndarray, radius: float, interface\_width: float = None, amplitudes: Optional[ndarray] = None*)

Bases: `PerturbedDropletBase`

Represents a single droplet in two dimensions with a perturbed shape

The shape is described using the distance  $R(\phi)$  of the interface from the *position*, which is a function of the polar angle  $\phi$ . This function is expressed as a truncated series of harmonics:

$$R(\phi) = R_0 + R_0 \sum_{n=1}^N \left[ \epsilon_n^{(1)} \sin(n\phi) + \epsilon_n^{(2)} \cos(n\phi) \right]$$

where  $N$  is the number of perturbation modes considered, which is given by half the length of the *amplitudes* array. Consequently, amplitudes should always be an even number, to consider both *sin* and *cos* terms.

**Parameters**

- **position** (`ndarray`) – Position of the droplet center
- **radius** (`float`) – Radius of the droplet
- **interface\_width** (`float`, *optional*) – Width of the interface
- **amplitudes** (`ndarray`) – (dimensionless) perturbation amplitudes  $\{\epsilon_1^{(1)}, \epsilon_1^{(2)}, \epsilon_2^{(1)}, \epsilon_2^{(2)}, \epsilon_3^{(1)}, \epsilon_3^{(2)}, \dots\}$ . The length of the array needs to be even to capture perturbations of the highest mode consistently.

**data:** `recarray`

**dim = 2**

**interface\_curvature** ( $\varphi$ )

calculates the mean curvature of the interface of the droplet

For simplicity, the effect of the perturbations are only included to linear order in the perturbation amplitudes  $\epsilon_n^{(1/2)}$ .

**Parameters**

$\varphi$  (*float or array*) – The angle in the polar coordinate system that is used to describe the interface

**Returns**

An array with the curvature at the interfacial points associated with each angle given by  $\varphi$ .

**interface\_distance** ( $\varphi$ )

calculates the distance of the droplet interface to the origin

**Parameters**

$\varphi$  (*float or array*) – The angle in the polar coordinate system that is used to describe the interface

**Returns**

An array with the distances of the interfacial points associated with each angle given by  $\varphi$ .

**interface\_position** ( $\varphi$ )

calculates the position of the interface of the droplet

**Parameters**

$\varphi$  (*float or array*) – The angle in the polar coordinate system that is used to describe the interface

**Returns**

An array with the coordinates of the interfacial points associated with each angle given by  $\varphi$ .

**property surface\_area: float**

surface area of the droplet

**Type**

float

**property surface\_area\_approx: float**

surface area of the droplet (quadratic in amplitudes)

**Type**

float

**property volume: float**

volume of the droplet

**Type**

float

**class PerturbedDroplet3D** (*position: ndarray, radius: float, interface\_width: float = None, amplitudes: Optional[ndarray] = None*)

Bases: PerturbedDropletBase

Represents a single droplet in two dimensions with a perturbed shape

The shape is described using the distance  $R(\theta, \phi)$  of the interface from the origin as a function of the azimuthal angle  $\theta$  and the polar angle  $\phi$ . This function is developed as a truncated series of spherical harmonics  $Y_{l,m}(\theta, \phi)$ :

$$R(\theta, \phi) = R_0 \left[ 1 + \sum_{l=1}^{N_l} \sum_{m=-l}^l \epsilon_{l,m} Y_{l,m}(\theta, \phi) \right]$$

where  $N_l$  is the number of perturbation modes considered, which is deduced from the length of the *amplitudes* array.

#### Parameters

- **position** (`ndarray`) – Position of the droplet center
- **radius** (`float`) – Radius of the droplet
- **interface\_width** (`float`, *optional*) – Width of the interface
- **amplitudes** (`ndarray`) – Perturbation amplitudes  $\epsilon_{l,m}$ . Note that the zero-th mode, which would only change the radius, is skipped. Consequently, the length of the array needs to be 0, 3, 8, 15, 24, ... to capture perturbations of the highest mode consistently.

**data:** `recarray`

**dim = 3**

**interface\_curvature** ( $\theta, \varphi$ )

calculates the mean curvature of the interface of the droplet

For simplicity, the effect of the perturbations are only included to linear order in the perturbation amplitudes  $\epsilon_{l,m}$ .

#### Parameters

- $\theta$  (`float` or `array`) – Azimuthal angle (in  $[0, \pi]$ )
- $\varphi$  (`float` or `array`) – Polar angle (in  $[0, 2\pi]$ )

#### Returns

An array with the curvature at the interfacial points associated with the angles

**interface\_distance** ( $\theta, \varphi$ )

calculates the distance of the droplet interface to the origin

#### Parameters

- $\theta$  (`float` or `array`) – Azimuthal angle (in  $[0, \pi]$ )
- $\varphi$  (`float` or `array`) – Polar angle (in  $[0, 2\pi]$ )

#### Returns

An array with the distances of the interfacial points associated with the angles.

**interface\_position** ( $\theta, \varphi$ )

calculates the position of the interface of the droplet

#### Parameters

- $\theta$  (`float` or `array`) – Azimuthal angle (in  $[0, \pi]$ )
- $\varphi$  (`float` or `array`) – Polar angle (in  $[0, 2\pi]$ )

#### Returns

An array with the coordinates of the interfacial points associated with the angles.



**property volume:** `float`

volume of the droplet (determined numerically)

**Type**

`float`

**property volume\_approx:** `float`

approximate volume to linear order in the perturbation

**Type**

`float`

**class SphericalDroplet** (*position: ndarray, radius: float*)

Bases: `DropletBase`

Represents a single, spherical droplet

**Parameters**

- **position** (`ndarray`) – Position of the droplet center
- **radius** (`float`) – Radius of the droplet

**property bbox:** `Cuboid`

bounding box of the droplet

**Type**

`Cuboid`

**check\_data** ()

method that checks the validity and consistency of `self.data`

**data:** `recarray`

**property data\_bounds:** `Tuple[ndarray, ndarray]`

lower and upper bounds on the parameters

**Type**

`tuple`

**property dim:** `int`

the spatial dimension this droplet is embedded in

**Type**

`int`

**classmethod from\_volume** (*position: ndarray, volume: float*)

Construct a droplet from given volume instead of radius

**Parameters**

- **position** (`ndarray`) – center of the droplet
- **volume** (`float`) – volume of the droplet
- **interface\_width** (`float, optional`) – width of the interface

**classmethod get\_dtype** (*\*\*kwargs*)

determine the dtype representing this droplet class

**Parameters**

**position** (`ndarray`) – The position vector of the droplet. This is used to determine the space dimension.

**Returns**

the (structured) dtype associated with this class

**Return type**

`numpy.dtype`

**get\_phase\_field** (*grid*: *GridBase*, \*, *vmin*: *float* = 0, *vmax*: *float* = 1, *label*: *str* = None) → *ScalarField*

Creates an image of the droplet on the *grid*

**Parameters**

- **grid** (*GridBase*) – The grid used for discretizing the droplet phase field
- **vmin** (*float*) – Minimal value the phase field will attain (far away from droplet)
- **vmax** (*float*) – Maximal value the phase field will attain (inside the droplet)
- **label** (*str*) – The label associated with the returned scalar field

**Returns**

A scalar field representing the droplet

**Return type**

*ScalarField*

**get\_triangulation** (*resolution*: *float* = 1) → *Dict*[*str*, *Any*]

obtain a triangulated shape of the droplet surface

**Parameters**

**resolution** (*float*) – The length of a typical triangulation element. This affects the resolution of the triangulation.

**Returns**

A dictionary containing information about the triangulation. The exact details depend on the dimension of the problem.

**Return type**

*dict*

**property interface\_curvature**: *float*

the mean curvature of the interface of the droplet

**Type**

*float*

**interface\_position** (\**args*) → *ndarray*

calculates the position of the interface of the droplet

**Parameters**

**\*args** (*float* or *ndarray*) – The angles identifying the interface points. For 2d droplets, this is simply the angle in polar coordinates. For 3d droplets, both the azimuthal angle  $\theta$  (in  $[0, \pi)$ ) and the polar angle  $\varphi$  (in  $[0, 2\pi]$ ) need to be specified.

**Returns**

An array with the coordinates of the interfacial points associated with each angle given by  $\varphi$ .

**Return type**

*ndarray*

**Raises**

**ValueError** – If the dimension of the space is not 2

**overlaps** (*other*: `SphericalDroplet`, *grid*: `Optional[GridBase] = None`) → `bool`

determine whether another droplet overlaps with this one

Note that this function so far only compares the distances of the droplets to their radii, which does not respect perturbed droplets correctly.

#### Parameters

- **other** (`SphericalDroplet`) – instance of the other droplet
- **grid** (`GridBase`) – grid that determines how distances are measured, which is for instance important to respect periodic boundary conditions. If omitted, an Euclidian distance is assumed.

#### Returns

whether the droplets overlap or not

#### Return type

`bool`

**plot** (*value*: `Callable = None`, *\*args*, *title*: `str = None`, *filename*: `str = None`, *action*: `str = 'auto'`, *ax\_style*: `Optional[Dict[str, Any]] = None`, *fig\_style*: `Optional[Dict[str, Any]] = None`, *ax*=`None`, *\*\*kwargs*) → `PlotReference`

Plot the droplet

#### Parameters

- **title** (`str`) – Title of the plot. If omitted, the title might be chosen automatically.
- **filename** (`str`, `optional`) – If given, the plot is written to the specified file.
- **action** (`str`) – Decides what to do with the final figure. If the argument is set to `show`, `matplotlib.pyplot.show()` will be called to show the plot. If the value is `none`, the figure will be created, but not necessarily shown. The value `close` closes the figure, after saving it to a file when `filename` is given. The default value `auto` implies that the plot is shown if it is not a nested plot call.
- **ax\_style** (`dict`) – Dictionary with properties that will be changed on the axis after the plot has been drawn by calling `matplotlib.pyplot.setp()`. A special item in this dictionary is `use_offset`, which is flag that can be used to control whether offset are shown along the axes of the plot.
- **fig\_style** (`dict`) – Dictionary with properties that will be changed on the figure after the plot has been drawn by calling `matplotlib.pyplot.setp()`. For instance, using `fig_style={'dpi': 200}` increases the resolution of the figure.
- **ax** (`matplotlib.axes.Axes`) – Figure axes to be used for plotting. The special value “create” creates a new figure, while “reuse” attempts to reuse an existing figure, which is the default.
- **value** (`callable`) – Sets the color of the droplet. This could be either a `matplotlib color` or a function that takes the droplet instance and returns a color in which this droplet is drawn. If given, it overwrites the `color` argument.
- **\*\*kwargs** – Additional keyword arguments are passed to the class that creates the patch that represents the droplet. For instance, to only draw the outlines of the droplets, you may need to supply `fill=False`.

#### Returns

Information about the plot

**Return type**

`PlotReference`

**property position:** `ndarray`

the position of the droplet

**Type**

`ndarray`

**property radius:** `float`

the radius of the droplet

**Type**

`float`

**property surface\_area:** `float`

surface area of the droplet

**Type**

`float`

**property volume:** `float`

volume of the droplet

**Type**

`float`

### 1.3.5 droplets.emulsions module

Classes that describe collections of droplets, i.e. emulsions, and their temporal dynamics.

<code>Emulsion</code>	class representing a collection of droplets in a common system
<code>EmulsionTimeCourse</code>	represents emulsions as a function of time

**class Emulsion** (*droplets: Optional[Union[Generator, Sequence[SphericalDroplet]]] = None, grid: Optional[GridBase] = None, copy: bool = True*)

Bases: `list`

class representing a collection of droplets in a common system

**Parameters**

- **droplets** – A list or generator of instances of `SphericalDroplet`.
- **grid** (`GridBase`) – The grid on which the droplets are defined. This information can help to measure distances between droplets correctly.
- **copy** (`bool`, *optional*) – Whether to make a copy of the droplet or not

**append** (*droplet: SphericalDroplet, copy: bool = True*) → `None`

add a droplet to the emulsion

**Parameters**

- **droplet** (`droplets.droplets.SphericalDroplet`) – Droplet to add to the emulsion
- **copy** (`bool`, *optional*) – Whether to make a copy of the droplet or not

**property bbox:** `Cuboid`

bounding box of the emulsion

**Type**

`Cuboid`

**copy** (*min\_radius: float = -1*) → *Emulsion*

return a copy of this emulsion

**Parameters**

**min\_radius** (*float*) – The minimal radius of the droplets that are retained. Droplets with exactly *min\_radius* are removed, so *min\_radius == 0* can be used to filter vanished droplets.

**property data:** `Optional[ndarray]`

an array containing the data of the full emulsion

This requires all droplets to be of the same class

**Type**

`ndarray`

**extend** (*droplets: Union[Generator, Sequence[SphericalDroplet]]*, *copy: bool = True*) → *None*

add many droplets to the emulsion

**Parameters**

- **droplets** (*list*) – List of droplets to add to the emulsion
- **copy** (*bool, optional*) – Whether to make a copy of the droplets or not

**classmethod from\_file** (*path: str*) → *Emulsion*

create emulsion by reading file

**Parameters**

**path** (*str*) – The path from which the data is read. This function assumes that the data was written as an HDF5 file using *to\_file()*.

**get\_linked\_data** () → *ndarray*

link the data of all droplets in a single array

**Returns**

**The array containing all droplet data. If entries in this array are modified, it will be reflected in the droplets.**

**Return type**

`ndarray`

**get\_neighbor\_distances** (*subtract\_radius: bool = False*) → *ndarray*

calculates the distance of each droplet to its nearest neighbor

Warning: Nearest neighbors are defined by comparing the distances between the centers of the droplets, not their surfaces.

**Parameters**

**subtract\_radius** (*bool*) – Determines whether to subtract the radius from the distance, i.e., whether to return the distance between the surfaces instead of the positions

**Returns**

a vector with a distance for each droplet

**Return type**

`ndarray`

**get\_pairwise\_distances** (*subtract\_radius: bool = False*) → ndarray

return the pairwise distance between droplets

**Parameters**

**subtract\_radius** (*bool*) – determines whether the distance is measured from interface to interface (for round droplets) or center to center.

**Returns**

a matrix with the distances between all droplets

**Return type**

ndarray

**get\_phasefield** (*grid: Optional[GridBase] = None, label: str = None*) → ScalarField

create a phase field representing a list of droplets

**Parameters**

- **grid** (`pde.grids.base.GridBase`) – The grid on which the phase field is created. If omitted, the grid associated with the emulsion is used.
- **label** (*str*) – Optional label for the returned scalar field

**Returns**

the actual phase field

**Return type**

ScalarField

**get\_size\_statistics** () → Dict[str, float]

determine size statistics of the current emulsion

**Returns**

a dictionary with various size statistics

**Return type**

dict

**property interface\_width: Optional[float]**

the average interface width across all droplets

This averages the interface widths of the individual droplets weighted by their surface area, i.e., the amount of interface

**Type**

float

**plot** (*field: ScalarField = None, image\_args: dict = None, repeat\_periodically: bool = True, color\_value: Callable = None, cmap=None, norm=None, colorbar: Union[bool, str] = True, \*args, title: str = None, filename: str = None, action: str = 'auto', ax\_style: Optional[Dict[str, Any]] = None, fig\_style: Optional[Dict[str, Any]] = None, ax=None, \*\*kwargs*) → PlotReference

plot the current emulsion together with a corresponding field

If the emulsion is defined in a 3d geometry, only a projection on the first two axes is shown.

**Parameters**

- **title** (*str*) – Title of the plot. If omitted, the title might be chosen automatically.
- **filename** (*str, optional*) – If given, the plot is written to the specified file.
- **action** (*str*) – Decides what to do with the final figure. If the argument is set to *show*, `matplotlib.pyplot.show()` will be called to show the plot. If the value is *none*,

the figure will be created, but not necessarily shown. The value `close` closes the figure, after saving it to a file when `filename` is given. The default value `auto` implies that the plot is shown if it is not a nested plot call.

- **ax\_style** (*dict*) – Dictionary with properties that will be changed on the axis after the plot has been drawn by calling `matplotlib.pyplot.setp()`. A special item in this dictionary is `use_offset`, which is flag that can be used to control whether offset are shown along the axes of the plot.
- **fig\_style** (*dict*) – Dictionary with properties that will be changed on the figure after the plot has been drawn by calling `matplotlib.pyplot.setp()`. For instance, using `fig_style={'dpi': 200}` increases the resolution of the figure.
- **ax** (`matplotlib.axes.Axes`) – Figure axes to be used for plotting. The special value “create” creates a new figure, while “reuse” attempts to reuse an existing figure, which is the default.
- **field** (`pde.fields.scalar.ScalarField`) – provides the phase field that is shown as a background
- **image\_args** (*dict*) – additional arguments determining how the phase field in the background is plotted. Acceptable arguments are described in `plot()`.
- **repeat\_periodically** (*bool*) – flag determining whether droplets are shown on both sides of periodic boundary conditions. This option can slow down plotting
- **color\_value** (*callable*) – Function used to determine the color of a droplet. The function is called with individual droplet objects and must return a single scalar value, which is then mapped to a color using the colormap given by `cmap` and a suitable normalization given by `norm`.
- **cmap** (*str* or `Colormap`) – The colormap used to map normalized data values to RGBA colors.
- **norm** (`Normalize`) – The normalizing object which scales data, typically into the interval `[0, 1]`. If `None`, `norm` defaults to a `colors.Normalize` object which maps the range of values obtained from `color_value` to `[0, 1]`.
- **colorbar** (*bool* or *str*) – Determines whether a colorbar is shown when `color_value` is supplied. If a string is given, it is used as a label for the colorbar.
- **\*\*kwargs** – Additional keyword arguments are passed to the function creating the patch that represents the droplet. For instance, to only draw the outlines of the droplets, you may need to supply `fill=False`.

#### Returns

Information about the plot

#### Return type

`PlotReference`

**remove\_overlapping** (*min\_distance: float = 0*) → `None`

remove all droplets that are overlapping.

If a pair of overlapping droplets was found, the smaller one of these is removed from the current emulsion. This method modifies the emulsion in place and thus does not return anything.

#### Parameters

**min\_distance** (*float*) – The minimal distance droplets need to be apart. The default value of 0 corresponds to just remove overlapping droplets. Larger values ensure that droplets keep a distance, while negative values allow for some overlap.

**remove\_small** (*min\_radius*: *float* = - *inf*) → *None*

remove droplets that are very small

The emulsions is modified in-place.

**Parameters**

**min\_radius** (*float*) – The minimal radius of the droplets that are retained. Droplets with exactly *min\_radius* are removed, so *min\_radius* == 0 can be used to filter vanished droplets. The default value does not remove any droplets

**to\_file** (*path*: *str*) → *None*

store data in hdf5 file

The data can be read using the classmethod *Emulsion.from\_file()*.

**Parameters**

**path** (*str*) – The path to which the data is written as an HDF5 file.

**property total\_droplet\_volume**: *float*

the total volume of all droplets

**Type**

*float*

**class EmulsionTimeCourse** (*emulsions*=*None*, *times*=*None*)

Bases: *object*

represents emulsions as a function of time

**Parameters**

- **emulsions** (*list*) – List of emulsions that describe this time course
- **times** (*list*) – Times associated with the emulsions

**append** (*emulsion*: *Emulsion*, *time*: *float* = *None*, *copy*: *bool* = *True*) → *None*

add an emulsion to the list

**Parameters**

- **emulsions** (*Emulsion*) – An *Emulsion* instance that is added to the time course
- **time** (*float*) – The time point associated with this emulsion
- **copy** (*bool*) – Whether to copy the emulsion

**clear** () → *None*

removes all data stored in this instance

**classmethod from\_file** (*path*: *str*, *progress*: *bool* = *True*) → *EmulsionTimeCourse*

create emulsion time course by reading file

**Parameters**

- **path** (*str*) – The path from which the data is read. This function assumes that the data was written as an HDF5 file using *to\_file()*.
- **progress** (*bool*) – Whether to show the progress of the process in a progress bar

**Returns**

an instance describing the emulsion time course

**Return type**

*EmulsionTimeCourse*



**classmethod from\_storage** (*storage: StorageBase, refine: bool = False, progress: Optional[bool] = None, \*\*kwargs*) → *EmulsionTimeCourse*

create an emulsion time course from a stored phase field

#### Parameters

- **storage** (*StorageBase*) – The phase fields for many time instances
- **refine** (*bool*) – Flag determining whether the droplet properties should be refined using fitting. This is a potentially slow procedure.
- **progress** (*bool*) – Whether to show the progress of the process. If *None*, the progress is only shown when *refine* is *True*.
- **\*\*kwargs** – All other parameters are forwarded to the *locate\_droplets()*.

#### Returns

an instance describing the emulsion time course

#### Return type

*EmulsionTimeCourse*

**items** ()

iterate over all times and emulsions, returning them in pairs

**to\_file** (*path: str, info: Optional[Dict[str, Any]] = None*) → *None*

store data in hdf5 file

The data can be read using the classmethod *EmulsionTimeCourse.from\_file()*.

#### Parameters

- **path** (*str*) – The path to which the data is written as an HDF5 file.
- **info** (*dict*) – Additional data stored alongside the droplet track list

**tracker** (*interval: Union[int, float, ConstantIntervals] = 1, filename: str = None*) → *DropletTracker*

return a tracker that analyzes emulsions during simulations

#### Parameters

- **interval** – Determines how often the tracker interrupts the simulation. Simple numbers are interpreted as durations measured in the simulation time variable. Alternatively, a string using the format ‘hh:mm:ss’ can be used to give durations in real time. Finally, instances of the classes defined in *intervals* can be given for more control.
- **filename** (*str*) – determines where the *EmulsionTimeCourse* data is stored

## 1.3.6 droplets.image\_analysis module

Functions for analyzing phase field images of emulsions.

<i>locate_droplets</i>	Locates droplets in the phase field
<i>refine_droplet</i>	Refines droplet parameters by fitting to phase field
<i>get_structure_factor</i>	Calculates the structure factor associated with a field
<i>get_length_scale</i>	Calculates a length scale associated with a phase field

**get\_length\_scale** (*scalar\_field*: ScalarField, *method*: str = 'structure\_factor\_maximum', *\*\*kwargs*) → Union[float, Tuple[float, Any]]

Calculates a length scale associated with a phase field

**Parameters**

- **scalar\_field** (ScalarField) – The scala field being analyzed
- **method** (*str*) – A string determining which method is used to calculate the length scale. Valid options are *structure\_factor\_maximum* (numerically determine the maximum in the structure factor) and *structure\_factor\_mean* (calculate the mean of the structure factor).

Additional supported keyword arguments depend on the chosen method. For instance, the methods involving the structure factor allow for a boolean flag *full\_output*, which also returns the actual structure factor. The method *structure\_factor\_maximum* also allows for some smoothing of the radially averaged structure factor. If the parameter *smoothing* is set to *None* the amount of smoothing is determined automatically from the typical discretization of the underlying grid. For the method *droplet\_detection*, additional arguments are forwarded to *locate\_droplets()*.

**Returns**

The determine length scale

If *full\_output = True*, a tuple with the length scale and an additional object with further information is returned.

**Return type**

float

**See also:**

*LengthScaleTracker*: Tracker measuring length scales

**get\_structure\_factor** (*scalar\_field*: ScalarField, *smoothing*: Union[None, float, str] = 'auto', *wave\_numbers*: Union[Sequence[float], str] = 'auto', *add\_zero*: bool = False) → Tuple[ndarray, ndarray]

Calculates the structure factor associated with a field

Here, the structure factor is basically the power spectral density of the field *scalar\_field* normalized so that re-gridding or rescaling the field does not change the result.

**Parameters**

- **scalar\_field** (ScalarField) – The scalar\_field being analyzed
- **smoothing** (*float*, *optional*) – Length scale that determines the smoothing of the radially averaged structure factor. If omitted, the full data about the discretized structure factor is returned. The special value *auto* calculates a value automatically.
- **wave\_numbers** (*list of floats*, *optional*) – The magnitude of the wave vectors at which the structure factor is evaluated. This only applies when smoothing is used. If *auto*, the wave numbers are determined automatically.
- **add\_zero** (*bool*) – Determines whether the value at k=0 (defined to be 1) should also be returned.

**Returns**

Two arrays giving the wave numbers and the associated structure factor

**Return type**

(numpy.ndarray, numpy.ndarray)

**locate\_droplets** (*phase\_field*: *ScalarField*, *threshold*: *Union[float, str] = 0.5*, *modes*: *int = 0*, *minimal\_radius*: *float = 0*, *refine*: *bool = False*, *interface\_width*: *float = None*) → *Emulsion*

Locates droplets in the phase field

This uses a binarized image to locate clusters of large concentration in the phase field, which are interpreted as droplets. Basic quantities, like position and size, are determined for these clusters.

#### Parameters

- **phase\_field** (*ScalarField*) – Scalar field that describes the concentration field of droplets
- **threshold** (*float or str*) – The threshold for binarizing the image. If a value is given it is used directly. Otherwise, the following algorithms are supported:
  - *extrema*: take mean between the minimum and the maximum of the data
  - *mean*: take the mean over the entire data
  - *otsu*: use Otsu’s method implemented in *threshold\_otsu()*

The special value *auto* currently defaults to the *extrema* method.

- **modes** (*int*) – The number of perturbation modes that should be included. If *modes=0*, droplets are assumed to be spherical. Note that the mode amplitudes are only determined when *refine=True*.
- **minimal\_radius** (*float*) – The smallest radius of droplets to include in the list. This can be used to filter out fluctuations in noisy simulations.
- **refine** (*bool*) – Flag determining whether the droplet properties should be refined using fitting. This is a potentially slow procedure.
- **interface\_width** (*float, optional*) – Interface width of the located droplets, which is also used as a starting value for the fitting if droplets are refined.

#### Returns

All detected droplets

#### Return type

*Emulsion*

**refine\_droplet** (*phase\_field*: *ScalarField*, *droplet*: *DiffuseDroplet*, *least\_squares\_params*: *Optional[Dict[str, Any]] = None*) → *DiffuseDroplet*

Refines droplet parameters by fitting to phase field

This function varies droplet parameters, like position, size, interface width, and potential perturbation amplitudes until the overlap with the respective phase field region is maximized. Here, we use a constraint fitting routine.

#### Parameters

- **phase\_field** (*ScalarField*) – *phase\_field* that is being used to refine the droplet
- **droplet** (*SphericalDroplet*) – Droplet that should be refined. This could also be a subclass of *SphericalDroplet*
- **least\_squares\_params** (*dict*) – Dictionary of parameters that influence the fitting; see the documentation of *scipy.optimize.least\_squares*

#### Returns

The refined droplet as an instance of the argument *droplet*

#### Return type

*droplets.droplets.DiffuseDroplet*

**threshold\_otsu** (*data: ndarray, nbins: int = 256*) → float

Find the threshold value for a bimodal histogram using the Otsu method.

If you have a distribution that is bimodal, i.e., with two peaks and a valley between them, then you can use this to find the location of that valley, which splits the distribution into two.

**Parameters**

- **data** (*ndarray*) – The data to be analyzed
- **nbins** (*int*) – The number of bins in the histogram, which defines the accuracy of the determined threshold.

Modified from <https://stackoverflow.com/a/71345917/932593>, which is based on the the SciKit Image `threshold_otsu` implementation: <https://github.com/scikit-image/scikit-image/blob/70fa904eee9ef370c824427798302551df57afa1/skimage/filters/thresholding.py#L312>

### 1.3.7 droplets.trackers module

Module defining classes for tracking droplets in simulations.

<i>LengthScaleTracker</i>	Tracker that stores length scales measured in simulations
<i>DropletTracker</i>	Detect droplets in a scalar field during simulations

**class DropletTracker** (*interval: Union[ConstantIntervals, float, int, str] = 1, filename: str = None, \*, emulsion\_timecourse=None, source: Union[None, int, Callable] = None, threshold: Union[float, str] = 0.5, minimal\_radius: float = 0, refine: bool = False, perturbation\_modes: int = 0*)

Bases: `TrackerBase`

Detect droplets in a scalar field during simulations

This tracker is useful when only the parameters of actual droplets are needed, since it stores considerably less information compared to the full scalar field.

**data**

Contains the data of the tracked droplets after the simulation is done.

**Type**

*EmulsionTimeCourse*

**Parameters**

- **interval** – Determines how often the tracker interrupts the simulation. Simple numbers are interpreted as durations measured in the simulation time variable. Alternatively, a string using the format ‘hh:mm:ss’ can be used to give durations in real time. Finally, instances of the classes defined in `intervals` can be given for more control.
- **filename** (*str, optional*) – Determines the file to which the final data is written as an HDF5 file.
- **emulsion\_timecourse** (`EmulsionTimeCourse`, optional) – Can be an instance of `EmulsionTimeCourse` that is used to store the data.
- **source** (*int or callable, optional*) – Determines how a field is extracted from `fields`. If `None`, `fields` is passed as is, assuming it is already a scalar field. This works for the simple, standard case where only a single `ScalarField` is treated. Alternatively, `source` can be an

integer, indicating which field is extracted from an instance of `FieldCollection`. Lastly, `source` can be a function that takes `fields` as an argument and returns the desired field.

- **threshold** (*float or str*) – The threshold for binarizing the image. The special value ‘auto’ takes the mean between the minimum and the maximum of the data as a guess.
- **minimal\_radius** (*float*) – Minimal radius of droplets that will be retained.
- **refine** (*bool*) – Flag determining whether the droplet coordinates should be refined using fitting. This is a potentially slow procedure.
- **perturbation\_modes** (*int*) – An option describing how many perturbation modes should be considered when refining droplets.

**finalize** (*info: Optional[Dict[str, Any]] = None*) → *None*

finalize the tracker, supplying additional information

#### Parameters

**info** (*dict*) – Extra information from the simulation

**handle** (*field: FieldBase, t: float*) → *None*

handle data supplied to this tracker

#### Parameters

- **field** (*FieldBase*) – The current state of the simulation
- **t** (*float*) – The associated time

**initialize** (*field: FieldBase, info: Optional[Dict[str, Any]] = None*) → *float*

#### Parameters

- **field** (*FieldBase*) – An example of the data that will be analyzed by the tracker
- **info** (*dict*) – Extra information from the simulation

#### Returns

The first time the tracker needs to handle data

#### Return type

*float*

**class LengthScaleTracker** (*interval: Union[ConstantIntervals, float, int, str] = 1, filename: str = None, \*, method: str = 'structure\_factor\_mean', source: Union[None, int, Callable] = None, verbose: bool = False*)

Bases: `TrackerBase`

Tracker that stores length scales measured in simulations

#### **times**

The time points at which the length scales are stored

#### Type

*list*

#### **length\_scales**

The associated length scales

#### Type

*list*

#### Parameters

- **interval** – Determines how often the tracker interrupts the simulation. Simple numbers are interpreted as durations measured in the simulation time variable. Alternatively, a string using the format ‘hh:mm:ss’ can be used to give durations in real time. Finally, instances of the classes defined in `intervals` can be given for more control.
- **filename** (*str*, *optional*) – Determines the file to which the data is written in JSON format
- **method** (*str*) – Method used for determining the length scale. Details are explained in the function `get_length_scale()`.
- **source** (*int or callable*, *optional*) – Determines how a field is extracted from *fields*. If *None*, *fields* is passed as is, assuming it is already a scalar field. This works for the simple, standard case where only a single `ScalarField` is treated. Alternatively, *source* can be an integer, indicating which field is extracted from an instance of `FieldCollection`. Lastly, *source* can be a function that takes *fields* as an argument and returns the desired field.
- **verbose** (*bool*) – Determines whether errors in determining the length scales are logged.

**finalize** (*info: Optional[Dict[str, Any]] = None*) → *None*

finalize the tracker, supplying additional information

**Parameters**

**info** (*dict*) – Extra information from the simulation

**handle** (*field: FieldBase*, *t: float*)

handle data supplied to this tracker

**Parameters**

- **field** (`FieldBase`) – The current state of the simulation
- **t** (*float*) – The associated time

### 1.3.8 droplets.version module

Contains the version information of the package

## INDICES AND TABLES

- genindex
- modindex
- search





## PYTHON MODULE INDEX

### d

- droplets, 6
- droplets.droplet\_tracks, 11
- droplets.droplets, 17
- droplets.emulsions, 24
- droplets.image\_analysis, 29
- droplets.tools, 7
- droplets.tools.spherical, 7
- droplets.trackers, 32
- droplets.version, 34



## A

append() (*DropletTrack method*), 11  
 append() (*Emulsion method*), 24  
 append() (*EmulsionTimeCourse method*), 28

## B

bbox (*Emulsion property*), 24  
 bbox (*SphericalDroplet property*), 21

## C

check\_data() (*SphericalDroplet method*), 21  
 clear() (*EmulsionTimeCourse method*), 28  
 copy() (*Emulsion method*), 25

## D

data (*DiffuseDroplet attribute*), 18  
 data (*DropletTrack property*), 11  
 data (*DropletTracker attribute*), 32  
 data (*Emulsion property*), 25  
 data (*PerturbedDroplet2D attribute*), 18  
 data (*PerturbedDroplet3D attribute*), 20  
 data (*SphericalDroplet attribute*), 21  
 data\_bounds (*DiffuseDroplet property*), 18  
 data\_bounds (*SphericalDroplet property*), 21  
 DiffuseDroplet (*class in droplets.droplets*), 17  
 dim (*DropletTrack property*), 11  
 dim (*PerturbedDroplet2D attribute*), 18  
 dim (*PerturbedDroplet3D attribute*), 20  
 dim (*SphericalDroplet property*), 21  
 droplets  
   module, 6  
 droplets.droplet\_tracks  
   module, 11  
 droplets.droplets  
   module, 17  
 droplets.emulsions  
   module, 24  
 droplets.image\_analysis  
   module, 29  
 droplets.tools  
   module, 7  
 droplets.tools.spherical

  module, 7  
 droplets.trackers  
   module, 32  
 droplets.version  
   module, 34  
 DropletTrack (*class in droplets.droplet\_tracks*), 11  
 DropletTracker (*class in droplets.trackers*), 32  
 DropletTrackList (*class in droplets.droplet\_tracks*),  
   14  
 duration (*DropletTrack property*), 11

## E

Emulsion (*class in droplets.emulsions*), 24  
 EmulsionTimeCourse (*class in droplets.emulsions*),  
   28  
 end (*DropletTrack property*), 11  
 environment variable  
   PYTHONPATH, 4  
 extend() (*Emulsion method*), 25

## F

finalize() (*DropletTracker method*), 33  
 finalize() (*LengthScaleTracker method*), 34  
 first (*DropletTrack property*), 12  
 from\_emulsion\_time\_course() (*DropletTrack-  
 List class method*), 14  
 from\_file() (*DropletTrack class method*), 12  
 from\_file() (*DropletTrackList class method*), 15  
 from\_file() (*Emulsion class method*), 25  
 from\_file() (*EmulsionTimeCourse class method*), 28  
 from\_storage() (*DropletTrackList class method*), 15  
 from\_storage() (*EmulsionTimeCourse class method*),  
   28  
 from\_volume() (*SphericalDroplet class method*), 21

## G

get\_dtype() (*DiffuseDroplet class method*), 18  
 get\_dtype() (*SphericalDroplet class method*), 21  
 get\_length\_scale() (in *module  
 droplets.image\_analysis*), 29  
 get\_linked\_data() (*Emulsion method*), 25

get\_neighbor\_distances() (*Emulsion method*), 25  
 get\_pairwise\_distances() (*Emulsion method*), 25  
 get\_phase\_field() (*SphericalDroplet method*), 22  
 get\_phasefield() (*Emulsion method*), 26  
 get\_position() (*DropletTrack method*), 12  
 get\_radii() (*DropletTrack method*), 12  
 get\_size\_statistics() (*Emulsion method*), 26  
 get\_structure\_factor() (in module *droplets.image\_analysis*), 30  
 get\_trajectory() (*DropletTrack method*), 12  
 get\_triangulation() (*SphericalDroplet method*), 22  
 get\_volumes() (*DropletTrack method*), 12

## H

handle() (*DropletTracker method*), 33  
 handle() (*LengthScaleTracker method*), 34

## I

initialize() (*DropletTracker method*), 33  
 interface\_curvature (*SphericalDroplet property*), 22  
 interface\_curvature() (*PerturbedDroplet2D method*), 19  
 interface\_curvature() (*PerturbedDroplet3D method*), 20  
 interface\_distance() (*PerturbedDroplet2D method*), 19  
 interface\_distance() (*PerturbedDroplet3D method*), 20  
 interface\_position() (*PerturbedDroplet2D method*), 19  
 interface\_position() (*PerturbedDroplet3D method*), 20  
 interface\_position() (*SphericalDroplet method*), 22  
 interface\_width (*DiffuseDroplet property*), 18  
 interface\_width (*Emulsion property*), 26  
 items() (*DropletTrack method*), 12  
 items() (*EmulsionTimeCourse method*), 29

## L

last (*DropletTrack property*), 12  
 length\_scales (*LengthScaleTracker attribute*), 33  
 LengthScaleTracker (*class in droplets.trackers*), 33  
 locate\_droplets() (in module *droplets.image\_analysis*), 30

## M

make\_radius\_from\_volume\_compiled() (in module *droplets.tools.spherical*), 7

make\_surface\_from\_radius\_compiled() (in module *droplets.tools.spherical*), 8  
 make\_volume\_from\_radius\_compiled() (in module *droplets.tools.spherical*), 8  
 module  
     droplets, 6  
     droplets.droplet\_tracks, 11  
     droplets.droplets, 17  
     droplets.emulsions, 24  
     droplets.image\_analysis, 29  
     droplets.tools, 7  
     droplets.tools.spherical, 7  
     droplets.trackers, 32  
     droplets.version, 34

## O

overlaps() (*SphericalDroplet method*), 22

## P

PerturbedDroplet2D (*class in droplets.droplets*), 18  
 PerturbedDroplet3D (*class in droplets.droplets*), 19  
 plot() (*DropletTrack method*), 12  
 plot() (*DropletTrackList method*), 15  
 plot() (*Emulsion method*), 26  
 plot() (*SphericalDroplet method*), 23  
 plot\_positions() (*DropletTrack method*), 13  
 plot\_positions() (*DropletTrackList method*), 16  
 points\_cartesian\_to\_spherical() (in module *droplets.tools.spherical*), 8  
 points\_spherical\_to\_cartesian() (in module *droplets.tools.spherical*), 8  
 position (*SphericalDroplet property*), 24  
 PYTHONPATH, 4

## R

radius (*SphericalDroplet property*), 24  
 radius\_from\_surface() (in module *droplets.tools.spherical*), 8  
 radius\_from\_volume() (in module *droplets.tools.spherical*), 9  
 refine\_droplet() (in module *droplets.image\_analysis*), 31  
 remove\_overlapping() (*Emulsion method*), 27  
 remove\_short\_tracks() (*DropletTrackList method*), 17  
 remove\_small() (*Emulsion method*), 27

## S

spherical\_harmonic\_real() (in module *droplets.tools.spherical*), 9  
 spherical\_harmonic\_real\_k() (in module *droplets.tools.spherical*), 9  
 spherical\_harmonic\_symmetric() (in module *droplets.tools.spherical*), 9

spherical\_index\_count() (in module *droplets.tools.spherical*), 10  
 spherical\_index\_count\_optimal() (in module *droplets.tools.spherical*), 10  
 spherical\_index\_k() (in module *droplets.tools.spherical*), 10  
 spherical\_index\_lm() (in module *droplets.tools.spherical*), 10  
 SphericalDroplet (class in *droplets.droplets*), 21  
 start (*DropletTrack* property), 14  
 surface\_area (*PerturbedDroplet2D* property), 19  
 surface\_area (*SphericalDroplet* property), 24  
 surface\_area\_approx (*PerturbedDroplet2D* property), 19  
 surface\_from\_radius() (in module *droplets.tools.spherical*), 10

## T

threshold\_otsu() (in module *droplets.image\_analysis*), 31  
 time\_overlaps() (*DropletTrack* method), 14  
 times (*LengthScaleTracker* attribute), 33  
 to\_file() (*DropletTrack* method), 14  
 to\_file() (*DropletTrackList* method), 17  
 to\_file() (*Emulsion* method), 28  
 to\_file() (*EmulsionTimeCourse* method), 29  
 total\_droplet\_volume (*Emulsion* property), 28  
 tracker() (*EmulsionTimeCourse* method), 29

## V

volume (*PerturbedDroplet2D* property), 19  
 volume (*PerturbedDroplet3D* property), 20  
 volume (*SphericalDroplet* property), 24  
 volume\_approx (*PerturbedDroplet3D* property), 21