py-droplets Documentation

Release unknown

David Zwicker

Sep 03, 2026

CONTENTS

1 Contents **3**

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

1.2.1.4 Tracking simulations . . . 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

1.3.1.2 droplets.droplet_tracks module . . . 12

1.3.1.3 droplets.droplets module . . . 19

1.3.1.4 droplets.emulsions module . . . 28

1.3.1.5 droplets.image_analysis module . . . 36

1.3.1.6 droplets.trackers module . . . 39

2 Indices and tables **43**

Python Module Index **45**

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

CHAPTER

ONE

CONTENTS

1.1 Installation

This *py-droplets* package is developed for python 3.10+ 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
```

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
h5py	>=2.10	Reading and writing data
matplotlib	>=3.1	Visualizing results
numpy	>=1.22	Array library used for storing data
numba	>=0.59	Just-in-time compilation to accelerate numerics
scipy	>=1.4	Miscellaneous scientific functions
py-pde	>=0.50	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. 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

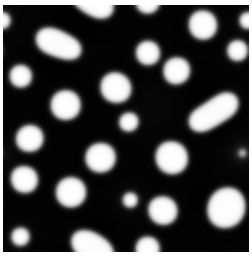


Fig. 1: An emulsion image

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.

1.2.1.4 Tracking simulations

We also provide a tracker, which can be used to monitor simulations:

```
from pde import CahnHilliardPDE, ScalarField, UnitGrid

from droplets.emulsions import EmulsionTimeCourse

field = ScalarField.random_uniform(UnitGrid([32, 32]), -1, 1)
pde = CahnHilliardPDE()

etc = EmulsionTimeCourse()
```

(continues on next page)

(continued from previous page)

```
pde.solve(field, t_range=10, backend="numpy", tracker=etc.tracker())

print(etc[-1].get_size_statistics())
```

The script prints statistics of the emulsion in the last tracked frame, which should n something like `{'count': 10, 'radius_mean': 2.3, 'radius_std': 0.8, 'volume_mean': 19.2, 'volume_std': 12.6}`. However, the *EmulsionTimeCourse* object `etc` provides much more detailed information. In particular, it can be converted to a *DropletTrackList* object by calling *from_emulsion_time_course()*.

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 ruff, based on the styles suggest by 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.misc module

Miscellaneous functions.

enable_scalar_args	Decorator that makes vectorized methods work with scalars.

enable_scalar_args(*method*)

Decorator that makes vectorized methods work with scalars.

This decorator allows to call functions that are written to work on numpy arrays to also accept python scalars, like *int* and *float*. Essentially, this wrapper turns them into an array and unboxes the result. Note that the dtype of the returned value will always be double or cdouble even if the function is called with an integer.

Parameters
method (*TFunc*) – The method being decorated

Returns
The decorated method

Return type
TFunc

droplets.tools.spherical module

Module collecting functions for handling spherical geometry.

The coordinate systems use the following convention for polar coordinates (r, ϕ), where r is the radial coordinate and ϕ 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, θ, ϕ), where r is the radial coordinate, θ is the azimuthal angle, and ϕ 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.

radius_from_volume	Return the radius of a sphere with a given volume.
volume_from_radius	Return the volume of a sphere with a given radius.
surface_from_radius	Return the surface area of a sphere with a given radius.
radius_from_surface	Return the radius of a sphere with a given surface area.
make_radius_from_volume_compiled	Return a function calculating the radius of a sphere with a given volume.
make_volume_from_radius_compiled	Return a function calculating the volume of a sphere with a given radius.
make_surface_from_radius_compiled	Return a function calculating the surface area of a sphere.
points_cartesian_to_spherical	Convert points from Cartesian to spherical coordinates.
points_spherical_to_cartesian	Convert points from spherical to Cartesian coordinates.
polar_coordinates	Return polar coordinates associated with grid points.
spherical_index_k	Returns the mode *k* from the degree *degree* and order *order*
spherical_index_lm	Returns the degree *l* and the order *m* from the mode *k*
spherical_index_count	Return the number of modes for all indices <= l.
spherical_index_count_optimal	Checks whether the modes captures all orders for maximal degree.
spherical_harmonic_symmetric	Axisymmetric spherical harmonics with degree *degree*, so *m=0*.
spherical_harmonic_real	Real spherical harmonics of degree l and order m.
spherical_harmonic_real_k	Real spherical harmonics described by mode k.

make_radius_from_volume_compiled(*dim*)

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_radius_from_volume_nd_compiled()

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

Returns
A function that calculate the radius from a volume and dimension

Return type
function

make_surface_from_radius_compiled(*dim*)

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*)

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

make_volume_from_radius_nd_compiled()

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

Returns
A function that calculates the volume using a radius and dimension

Return type
function

points_cartesian_to_spherical(*points*)

Convert points from Cartesian to spherical coordinates.

Parameters
points (ndarray) – Points in Cartesian coordinates

Returns
Points (r, θ, φ) in spherical coordinates

Return type
ndarray

points_spherical_to_cartesian(*points*)

Convert points from spherical to Cartesian coordinates.

Parameters
points (ndarray) – Points in spherical coordinates (r, θ, φ)

Returns
Points in Cartesian coordinates

Return type
ndarray

polar_coordinates(*grid: GridBase*, *, *origin: ndarray[tuple[int, ...], dtype[integer | floating]] | None = None*, *ret_angle: Literal[False] = False*) → ndarray[tuple[int, ...], dtype[integer | floating]]

polar_coordinates(*grid: GridBase*, *, *origin: ndarray[tuple[int, ...], dtype[integer | floating]] | None = None*, *ret_angle: Literal[True]*) → tuple[ndarray[tuple[int, ...], dtype[integer | floating]], ...]

Return polar coordinates associated with grid points.

Parameters

- **grid** (GridBase) – The grid whose cell coordinates are used.
- **origin** (ndarray, optional) – Cartesian coordinates of the origin at which polar coordinates are anchored.
- **ret_angle** (*bool*) – Determines whether angles are returned alongside the distance. If *False* only the distance to the origin is returned for each support point of the grid. If *True*, the distance and angles are returned. For a 1d system system, the angle is defined as the sign of

the difference between the point and the origin, so that angles can either be 1 or -1. For 2d systems and 3d systems, polar coordinates and spherical coordinates are used, respectively.

Returns
Coordinates values in polar coordinates

Return type
ndarray or tuple of ndarray

radius_from_surface(*surface*, *dim*)

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*, *dim*)

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*, *order*, θ, φ)

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
- **θ** (*float*) – Azimuthal angle (in $[0, \pi]$) at which function is evaluated.
- **ψ** (*float*) – Polar angle (in $[0, 2\pi]$) at which function is evaluated.

Returns
The value of the spherical harmonics

Return type
float

spherical_harmonic_real_k(k, θ, φ)

Real spherical harmonics described by mode k.

Parameters

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

- **φ** (*float*) – Polar angle (in $[0, 2\pi]$) at which function is evaluated.

Returns
The value of the spherical harmonics

Return type
float

spherical_harmonic_symmetric(*degree*, θ)

Axisymmetric spherical harmonics with degree *degree*, so *m=0*.

Parameters

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

Returns
The value of the spherical harmonics

Return type
float

spherical_index_count(*l*)

Return the number of modes for all indices <= 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*)

Checks whether the modes captures all orders for maximal degree.

Parameters
k_count (*int*) – The number of modes considered

Returns
indicates whether *k_count* is optimally chosen.

Return type
bool

spherical_index_k(*degree*, *order=0*)

Returns the mode *k* from the degree *degree* and order *order*

Parameters

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

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

Returns
a combined index k

Return type
int

spherical_index_lm(*k*)

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*, *dim*)

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

volume_from_radius(*radius*, *dim*)

Return the volume of a sphere with a given radius.

Parameters

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

Returns
Volume of the sphere

Return type
float or ndarray

droplets.tools.typing module

Miscellaneous types.

1.3.1.2 droplets.droplet_tracks module

Classes representing the time evolution of droplets.

DropletTrack	Information about a single droplet over multiple time steps.
DropletTrackList	A list of instances of *DropletTrack*

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*, *time=None*)

Append a new droplet with a time code.

Parameters

- **droplet** (*droplets.droplets.SphericalDroplet*) – The droplet to append
- **time** (*float, optional*) – The associated time point

Return type
None

property data: RealArray | None

an array containing the data of the full track.

Type
ndarray

property dim: int | None

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*)

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()*.

Return type
DropletTrack

get_position(*time*)

ndarray: returns the droplet position at a specific time.

Parameters
time (*float*)

Return type
RealArray

get_radii(*smoothing=0*)

ndarray: returns the droplet radius for each time point.

Parameters
smoothing (*float*) – Determines the length scale for some gaussian smoothing of the trajectory. The default value of zero disables smoothing.

Return type
RealArray

get_trajectory(*smoothing=0*, ***, *attribute='position'*)

Return a the time-evolution of a droplet attribute (e.g., the position)

Parameters

- **smoothing** (*float*) – Determines the scale for some gaussian smoothing of the trajectory. The default value of zero disables smoothing.
- **attribute** (*str*) – The attribute to consider (default: "position").

Returns

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

Return type
ndarray

get_volumes(*smoothing=0*)

ndarray: returns the droplet volume for each time point.

Parameters
smoothing (*float*) – Determines the volume scale for some gaussian smoothing of the trajectory. The default value of zero disables smoothing.

Return type
RealArray

items()

Iterate over all times and droplets, returning them in pairs.

property last: ***SphericalDroplet***

last droplet instance

Type
SphericalDroplet

plot(*attribute='radius'*, *smoothing=0*, *t_max=None*, **args*, *title=None*, *filename=None*, *action='auto'*, *ax_style=None*, *fig_style=None*, *ax=None*, ***kwargs*)

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.
- **smoothing** (*float*) – Determines the scale for some gaussian smoothing of the trajectory. The default value of zero disables smoothing.
- **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 *auto* or *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.
- **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 i 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.
- **t_max** (*float | None*)

Returns
Information about the plot

Return type
`PlotReference`

plot_positions(*grid=None*, *arrow=True*, **args*, *title=None*, *filename=None*, *action='auto'*, *ax_style=None*, *fig_style=None*, *ax=None*, ***kwargs*)

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 *auto* or *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.
- **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 i 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*)
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*, *info=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

Return type
None

class DropletTrackList(*iterable=()*, */*)
Bases: list

A list of instances of *DropletTrack*

classmethod from_emulsion_time_course(*time_course*, *, *method='overlap'*, *grid=None*, *progress=False*, ***kwargs*)
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).
- **grid** (GridBase) – The grid on which the droplets are defined, which is necessary if periodic boundary conditions should be respected for measuring 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*, *, *progress=True*)

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()*.
- **progress** (*bool*) – Whether to show the progress of the process in a progress bar

Returns
an instance describing the droplet track list

Return type
DropletTrackList

classmethod from_storage(*storage*, *, *method='overlap'*, *refine=False*, *num_processes=1*, *progress=None*)

Obtain droplet tracks from stored scalar field data.

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

Parameters

- **storage** (StorageBase) – The phase fields for many time instances
- **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).
- **refine** (*bool*) – Flag determining whether the droplet properties should be refined using fitting. This is a potentially slow procedure.
- **num_processes** (*int or "auto"*) – Number of processes used for the refinement. If set to "auto", the number of processes is chosen automatically.
- **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='radius'*, **args*, *title=None*, *filename=None*, *action='auto'*, *ax_style=None*, *fig_style=None*, *ax=None*, ***kwargs*)

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 *auto* or *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.
- **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 i 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=None*, *filename=None*, *action='auto'*, *ax_style=None*, *fig_style=None*, *ax=None*, ***kwargs*)

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 *auto* or *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.
- **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 i 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=0*)

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.

Return type
None

to_file(*path*, *info=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

Return type
None

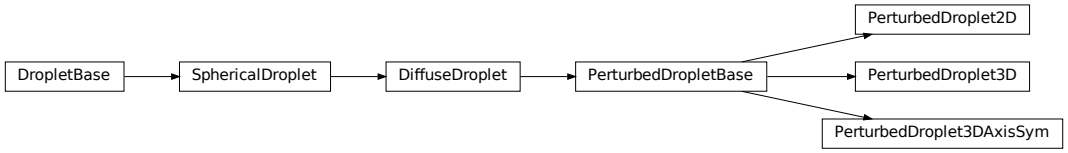
1.3.1.3 droplets.droplets module

Classes representing (perturbed) droplets in various dimensions.

The classes differ in what features of a droplet they track. In the simplest case, only the position and radius of a spherical droplet are stored. Other classes additionally keep track of the interfacial width or shape perturbations (of various degrees).

SphericalDroplet	Represents a single, spherical droplet.
DiffuseDroplet	Represents a single, spherical droplet with a diffuse interface.
PerturbedDroplet2D	Represents a single 2D droplet with a perturbed shape.
PerturbedDroplet3D	Represents a single 3D droplet with a perturbed shape.
PerturbedDroplet3DAxisSym	Represents a 3D droplet with axisymmetrically perturbed shape.

Inheritance structure of the classes:



The details of the classes are explained below:

class DiffuseDroplet(*position*, *radius*, *interface_width=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: np.recarray

property data_bounds: tuple[RealArray, RealArray]

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, determining the space dimension.

Returns
the (structured) dtype associated with this class

Return type
numpy.dtype

property interface_width: float | None

the width of the interface of this droplet

Type
float

class PerturbedDroplet2D(*position*, *radius*, *interface_width=None*, *amplitudes=None*)

Bases: PerturbedDropletBase

Represents a single 2D droplet 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 ϕ. 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: np.recarray

dim = 2

interface_curvature(φ)

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
φ (float or `RealArray`) – The angle in the polar coordinate system that describing the interface

Returns
Array with curvature at the interfacial points associated with each angle φ

Return type
RealArray

interface_distance(φ)

Calculates the distance of the droplet interface to the origin.

Parameters
φ (float or `RealArray`) – The angle in the polar coordinate system that describing the interface

Returns
Array with distances of the interfacial points associated with each angle φ

Return type
RealArray

interface_position(φ)

Calculates the position of the interface of the droplet.

Parameters
φ (float or `RealArray`) – The angle in the polar coordinate system that describing the interface

Returns
Array with coordinates of interfacial points associated with each angle φ

Return type
RealArray

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*, *radius*, *interface_width=None*, *amplitudes=None*)

Bases: `PerturbedDropletBase`

Represents a single 3D droplet 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 θ and the polar angle ϕ. 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: np.recarray

dim = 3

interface_curvature(*θ*, *φ=None*)

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

- **θ** (float or RealArray) – Azimuthal angle (in $[0, \pi]$)
- **φ** (float or RealArray) – Polar angle (in $[0, 2\pi]$); 0 if omitted

Returns
Array with curvature at the interfacial points associated with the angles

Return type
RealArray

interface_distance(*θ*, *φ=None*)

Calculates the distance of the droplet interface to the origin.

Parameters

- **θ** (float or RealArray) – Azimuthal angle (in $[0, \pi]$)
- **φ** (float or RealArray) – Polar angle (in $[0, 2\pi]$); 0 if omitted

Returns
Array with distances of the interfacial points associated with the angles

Return type
RealArray

interface_position(*θ*, *φ=None*)

Calculates the position of the interface of the droplet.

Parameters

- **θ** (float or RealArray) – Azimuthal angle (in $[0, \pi]$)
- **φ** (float or RealArray) – Polar angle (in $[0, 2\pi]$); 0 if omitted

Returns

Array with coordinates of the interfacial points associated with the angles

Return type

RealArray

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 PerturbedDroplet3DAxisSym(*position*, *radius*, *interface_width=None*, *amplitudes=None*)

Bases: PerturbedDropletBase

Represents a 3D droplet with axisymmetrically perturbed shape.

The shape is described using the distance $R(\theta)$ of the interface from the origin as a function of the azimuthal angle θ, while polar symmetry is assumed. This function is developed as a truncated series of spherical harmonics $Y_{l,m}(\theta, 0)$:

$$R(\theta) = R_0 \left[1 + \sum_{l=1}^{N_l} \epsilon_l Y_{l,0}(\theta, 0)\right]$$

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) – The amplitudes of the perturbations

check_data()

Method that checks the validity and consistency of self.data.

data: np.recarray

dim = 3

interface_curvature(θ)

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
θ (float or `RealArray`) – Azimuthal angle (in $[0, \pi]$)

Returns
Array with curvature at the interfacial points associated with the angles

Return type
RealArray

interface_distance(θ)

Calculates the distance of the droplet interface to the origin.

Parameters
θ (float or `RealArray`) – Azimuthal angle (in $[0, \pi]$)

Returns
Array with distances of the interfacial points associated with the angles

Return type
RealArray

property volume_approx: float

approximate volume to linear order in the perturbation

Type
float

class SphericalDroplet(*position*, *radius*)

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: np.recarray

property data_bounds: tuple[RealArray, RealArray]

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*, *volume*, ***kwargs*)

Construct a droplet from given volume instead of radius.

Parameters

- **position** (ndarray) – Center of the droplet
- **volume** (*float*) – Volume of the droplet
- ****kwargs** – Additional arguments are forwarded to the class initializer. This can for instance be used to set the interfacial width.

classmethod get_dtype(***kwargs*)

Determine the dtype representing this droplet class.

Parameters
position (ndarray) – The position vector of the droplet, determining space dimension.

Returns
the (structured) dtype associated with this class

Return type
numpy.dtype

get_phase_field(*grid*, *, *vmin=0*, *vmax=1*, *label=None*)

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=1*)

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*)

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 θ (in $[0, \pi]$) and the polar angle φ (in $[0, 2\pi]$) need to be specified.

Returns
An array with the coordinates of the interfacial points associated with each angle given by φ.

Return type
ndarray

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

overlaps(*other*, *grid=None*)

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 Euclidean distance is assumed.

Returns
whether the droplets overlap or not

Return type
bool

plot(*value=None*, *autoscale=True*, **args*, *title=None*, *filename=None*, *action='auto'*, *ax_style=None*, *fig_style=None*, *ax=None*, ***kwargs*)

Plot the droplet.

Tip

The default settings shows the droplet as a blue disk. The color can be set using the `color` argument or via more detailed arguments, such as `facecolor` or `edgecolor`. To only show an outline, use `fill=False`.

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 *auto* or *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.
- **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 i 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.
- **autoscale** (*bool*) – Determines whether the axes limits should be scaled automatically. If True `autoscale()` is called.
- ****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: RealArray
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

1.3.1.4 droplets.emulsions module

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

Emulsion	Class representing a collection of droplets in a common system.
EmulsionTimeCourse	Represents emulsions as a function of time.

class Emulsion(*droplets=None*, ***, *copy=True*, *dtype=None*, *force_consistency=False*)

Bases: `list`

Class representing a collection of droplets in a common system.

Parameters

- **droplets** (*Iterable[SphericalDroplet] | None*) – A list or generator of instances of *SphericalDroplet*.
- **copy** (*bool, optional*) – Whether to make a copy of the droplet or not
- **dtype** (`DTypeLike`) – The dtype describing what droplets are stored in the emulsion. Providing this is usually only necessary for creating empty emulsions. Instead of a dtype, an array or an example droplet can also be supplied.
- **force_consistency** (*bool, optional*) – Whether to ensure that all droplets are of the same type, i.e., their data is described by the same dtype and matches *dtype* if given.

append(*droplet*, ***, *copy=True*, *force_consistency=False*)

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
- **force_consistency** (*bool, optional*) – Whether to ensure that all droplets are of the same type

Return type
None

property bbox: Cuboid

bounding box of the emulsion.

Type
`Cuboid`

copy(*min_radius=-1*)

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.

Return type
Emulsion

property data: RealArray

an array containing the data of the full emulsion.

> **Warning**
>
> This requires all droplets to be of the same class. The returned array is a copy of all the data and writing to it will thus not change the underlying data.

Type
ndarray

property dim: int | None

dimensionality of space in which droplets are defined

Type
int

dtype: dtype[Any] | None | type[Any] | _SupportsDType[dtype[Any]] | str | tuple[Any, int] | tuple[Any, SupportsIndex | Sequence[SupportsIndex]] | list[Any] | _DTypeDict | tuple[Any, Any]

Type of the data, which determines the type of droplets in the emulsion.

classmethod empty(*droplet*)

Create empty emulsion with particular droplet type.

Parameters
droplet (*SphericalDroplet*) – An example for a droplet, which defines the type of

Returns
The empty emulsion

Return type
Emulsion

extend(*droplets*, *, *copy=True*, *force_consistency=False*)

Add many droplets to the emulsion.

Parameters

- **droplet** (list of *droplets.droplets.SphericalDroplet*) – List of droplets to add to the emulsion
- **copy** (*bool, optional*) – Whether to make a copy of the droplet or not
- **force_consistency** (*bool, optional*) – Whether to ensure that all droplets are of the same type
- **droplets** (*Iterable[SphericalDroplet]*)

Return type
None

filter(*, *min_radius=-inf*, *max_radius=inf*)

Filter droplets by various criteria

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
- **max_radius** (*float*) – The maximal radius of the droplets that are retained.

Returns
Emulsion The emulsion with droplets that are retained

Return type
Emulsion

classmethod from_file(*path*)

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()*.

Returns
The emulsion read from the file

Return type
Emulsion

classmethod from_random(*num*, *grid_or_bounds*, *radius*, *, *remove_overlapping=True*, *droplet_class=<class 'droplets.droplets.SphericalDroplet'>*, *rng=None*)

Create an emulsion with random droplets.

Parameters

- **num** (*int*) – The (maximal) number of droplets to generate
- **grid_or_bounds** (GridBase or list of float tuples) – Determines the space in which droplets are placed. This is either a GridBase describing the geometry or a sequence of tuples with lower and upper bounds for each axes, so the length of the sequence determines the space dimension.
- **radius** (*float or tuple of float*) – Radius of the droplets that are created. If two numbers are given, they specify the bounds of a uniform distribution from which the radius of each individual droplet is chosen.
- **remove_overlapping** (*bool*) – Flag determining whether overlapping droplets are removed. If enabled, the resulting element might contain less than *num* droplets.
- **droplet_class** (*SphericalDroplet*) – The class that is used to create droplets.
- **rng** (Generator) – Random number generator (default: default_rng())

Returns
The emulsion containing the random droplets

Return type
Emulsion

get_linked_data()

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=False*)

Calculates the distance of each droplet to its nearest neighbor.

> **Warning**
>
> This function does not take periodic boundary conditions into account.

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=False*, *grid=None*)

Return the pairwise distance between droplets.

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
- **grid** (GridBase) – The grid on which the droplets are defined, which is necessary if periodic boundary conditions should be respected for measuring distances

Returns
a matrix with the distances between all droplets

Return type
ndarray

get_phasefield(*grid*, *label=None*)

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*) – Optional label for the returned scalar field

Returns
the actual phase field

Return type
ScalarField

get_size_statistics(*incl_vanished=True*)

Determine size statistics of the current emulsion.

Parameters
incl_vanished (*bool*) – Whether to include droplets with vanished radii

Returns
a dictionary with various size statistics

Return type
dict

property interface_width: float | None

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=None*, *image_args=None*, *repeat_periodically=True*, *grid=None*, *set_bounds=True*, *color_value=None*, *cmap=None*, *norm=None*, *colorbar=True*, **args*, *title=None*, *filename=None*, *action='auto'*, *ax_style=None*, *fig_style=None*, *ax=None*, ***kwargs*)

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 *auto* or *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.
- **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 i 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
- **grid** (`GridBase`) – The grid on which the droplets are defined, which is necessary if periodic boundary conditions should be respected for measuring distances
- **set_bounds** (*bool*) – Determines whether the axis bounds are set explicitly (and autoscale) is disabled. If True, the bounds are determined from the supplied field or grid. If these are omitted, the bounding box is determined from all droplets.
- **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=0*, *grid=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.
- **grid** (GridBase) – The grid on which the droplets are defined, which is necessary if periodic boundary conditions should be respected for measuring distances

Return type
None

remove_small(*min_radius=-inf*)

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

Return type
None

to_file(*path*)

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.

Return type
None

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*, *time=None*, *copy=True*)

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
- **emulsion** (Emulsion)

Return type
None

clear()

Removes all data stored in this instance.

Return type
None

classmethod from_file(*path*, *, *progress=True*)

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*, *, *num_processes=1*, *refine=False*, *progress=None*, **kwargs*)

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.

- **num_processes** (*int or "auto"*) – Number of processes used for the refinement. If set to "auto", the number of processes is chosen automatically.
- **progress** (*bool*) – Whether to show the progress of the process. If *None*, the progress is only shown when *refine* is *True*. Progress bars are only shown for serial calculations (where *num_processes == 1*).
- ****kwargs** – All other parameters are forwarded to the *locate_droplets()*.

Returns
an instance describing the emulsion time course

Return type
EmulsionTimeCourse

get_emulsion(*time*)

Returns the emulsion closest to a specific time point.

Parameters
time (*float*) – The time point

Returns
Emuslion

Return type
Emulsion

items()

Iterate over all times and emulsions, returning them in pairs.

Return type
Iterator[tuple[float, *Emulsion*]]

to_file(*path*, *info=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

Return type
None

tracker(*interrupts=1*, *filename=None*, ***kwargs*)

Return a tracker that analyzes emulsions during simulations.

Parameters

- **interrupts** (*InterruptData*) – {ARG_TRACKER_INTERRUPTS}
- **filename** (*str*) – determines where the EmulsionTimeCourse data is stored
- ****kwargs** – Arguments forwarded to *DropletTracker*

Return type
DropletTracker

1.3.1.5 droplets.image_analysis module

Functions for analyzing phase field images of emulsions.

locate_droplets	Locates droplets in the phase field.
refine_droplets	Refines many droplets by fitting to phase field.
refine_droplet	Refines droplet parameters by fitting to phase field.
get_structure_factor	Calculates the structure factor associated with a field.
get_length_scale	Calculates a length scale associated with a phase field.
threshold_otsu	Find the threshold value for a bimodal histogram using the Otsu method.

get_length_scale(*scalar_field*, *method='structure_factor_maximum'*, ***kwargs*)

Calculates a length scale associated with a phase field.

Parameters

- **scalar_field** (*ScalarField*) – The scalar field to analyze
- **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), *structure_factor_mean* (calculate the mean of the structure factor), and *droplet_detection* (determine the number of droplets and estimate average separation).

Return type
float | tuple[float, *Any*]

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

Return type
float

Parameters

- **scalar_field** (*ScalarField*)
- **method** (*Literal['structure_factor_mean', 'structure_factor_maximum', 'droplet_detection']*)

See also

LengthScaleTracker: Tracker measuring length scales

get_structure_factor(*scalar_field*, *smoothing='auto'*, *wave_numbers='auto'*, *add_zero=False*)

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 regridding 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. Wave numbers k are related to distances by $2\pi/k$.

Return type

(numpy.ndarray, numpy.ndarray)

locate_droplets(*phase_field*, *threshold=0.5*, ***, *minimal_radius=0*, *modes=0*, *interface_width=None*, *refine=False*, *refine_args=None*, *num_processes=1*)

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.

Example

To determine the position, radius, and interfacial width of an arbitrary droplet, the following call can be used

```
emulsion = droplets.locate_droplets(
    field,
    threshold="auto",
    refine=True,
    refine_args={"vmin": None, "vmax": None},
)
```

`field` is the scalar field, in which the droplets are located. The *refine_args* set flexible boundaries for the intensities inside and outside the droplet.

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.
- **minimal_radius** (*float*) – The smallest radius of droplets to include in the list. This can be used to filter out fluctuations in noisy simulations.

- **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*.
- **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.
- **refine** (*bool*) – Flag determining whether the droplet properties should be refined using fitting. This is a potentially slow procedure.
- **refine_args** (*dict*) – Additional keyword arguments passed on to *refine_droplet()*. Only has an effect if *refine=True*.
- **num_processes** (*int*) – Number of processes used for the refinement. If set to "auto", the number of processes is chosen automatically.

Returns
All detected droplets

Return type
Emulsion

refine_droplet(*phase_field*, *droplet*, ***, *vmin=0.0*, *vmax=1.0*, *adjust_values=False*, *tolerance=None*, *least_squares_params=None*)

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
- **vmin** (*float*) – The intensity value of the dilute phase surrounding the droplet. If *None*, the value will be determined automatically.
- **vmax** (*float*) – The intensity value inside the droplet. If *None*, the value will be determined automatically.
- **adjust_values** (*bool*) – Flag determining whether the intensity values will be included in the fitting procedure. The default value *False* implies that the intensity values are regarded fixed.
- **tolerance** (*float, optional*) – Sets the three tolerance values *ftol*, *xtol*, and *gtol* of the scipy.optimize.least_squares(), unless they are specified in detail by the *least_squares_params* argument.
- **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
DiffuseDroplet

refine_droplets(*phase_field*, *candidates*, ***, *num_processes=1*, ***kwargs*)

Refines many droplets by fitting to phase field.

Parameters

- **phase_field** (ScalarField) – Phase_field that is being used to refine the droplet
- **droplets** (sequence of *SphericalDroplet*) – Droplets that should be refined.
- **num_processes** (*int or "auto"*) – Number of processes used for the refinement. If set to "auto", the number of processes is chosen automatically.
- ****kwargs** (*dict*) – Additional keyword arguments are passed on to *refine_droplet()*.
- **candidates** (*Iterable[DiffuseDroplet]*)

Returns
The refined droplets

Return type
list of *DiffuseDroplet*

threshold_otsu(*data*, *nbins=256*)

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.

Return type
float

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

1.3.1.6 droplets.trackers module

Module defining classes for tracking droplets in simulations.

LengthScaleTracker	Tracker that stores length scales measured in simulations.
DropletTracker	Detect droplets in a scalar field during simulations.

class DropletTracker(*interrupts=1*, *filename=None*, ***, *emulsion_timecourse=None*, *source=None*, *threshold=0.5*, *minimal_radius=0*, *refine=False*, *refine_args=None*, *perturbation_modes=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. The file written when *filename* is supplied can be read in later using *from_file()*.

data

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

Type
EmulsionTimeCourse

Example

To track droplets and determine their position, radii, and interfacial widths, the following tracker can be used

```
droplet_tracker = DropletTracker(
    1, refine=True, refine_args={"vmin": None, "vmax": None}
)
```

`field` is the scalar field, in which the droplets are located. The *refine_args* set flexible boundaries for the intensities inside and outside the droplet.

Parameters

- **interrupts** (*InterruptData*) – Determines when the tracker interrupts the simulation. A single numbers determines an interval (measured in the simulation time unit) of regular interruption. A string is interpreted as a duration in real time assuming the format 'hh:mm:ss'. A list of values is taken as explicit simulation time points. More fine- grained control is possible by passing an instance of classes defined in `interrupts`.
- **filename** (*str, optional*) – Determines the path to the HDF5 file to which the *EmulsionTimeCourse* data is written.
- **emulsion_timecourse** (EmulsionTimeCourse, optional) – Can be an instance of *EmulsionTimeCourse* that is used to store the data. If omitted, an empty class is initiated.
- **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. 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*: Otsu's method *threshold_otsu()*

  The special value *auto* currently defaults to the *extrema* method.
- **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.
- **refine_args** (*dict*) – Additional keyword arguments passed on to *refine_droplet()*. Only has an effect if *refine=True*.
- **perturbation_modes** (*int*) – An option describing how many perturbation modes should be considered when refining droplets. Only has an effect if *refine=True*.

finalize(*info=None*)

Finalize the tracker, supplying additional information.

Parameters

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

Return type

None

handle(*field*, *t*)

Handle data supplied to this tracker.

Parameters

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

Return type
None

class LengthScaleTracker(*interrupts=1*, *filename=None*, ***, *method='structure_factor_mean'*, *source=None*, *verbose=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

- **interrupts** (*InterruptData*) – {ARG_TRACKER_INTERRUPTS}
- **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=None*)

Finalize the tracker, supplying additional information.

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

Return type
None

handle(*field*, *t*)

Handle data supplied to this tracker.

Parameters

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

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

droplets, 6
droplets.droplet_tracks, 12
droplets.droplets, 19
droplets.emulsions, 28
droplets.image_analysis, 36
droplets.tools, 7
droplets.tools.misc, 7
droplets.tools.spherical, 7
droplets.tools.typing, 12
droplets.trackers, 39