

---

# **quanfima Documentation**

***Release 0.1a1***

**Roman Shkarin, Andrei Shkarin**

**Apr 26, 2020**



---

## Contents

---

<b>1</b>	<b>User Guide</b>	<b>3</b>
1.1	Overview . . . . .	3
1.2	Quickstart . . . . .	4
1.3	API Reference . . . . .	14
1.4	ChangeLog . . . . .	18
	<b>Python Module Index</b>	<b>19</b>
	<b>Index</b>	<b>21</b>



*Quanfima* (**q**uantitative **a**nalysis of **f**ibrous **m**aterials) is a collection of useful routines for morphological analysis and visualization of 2D/3D data from various areas of material science. The aim is to simplify the analysis process by providing functionality for frequently required tasks in the same place.

- Analysis of fibrous structures by tensor-based method in 2D / 3D datasets.
- Estimation of structure diameters in 2D / 3D by a ray-casting method.
- Counting of particles in 2D / 3D datasets and providing a detailed report in pandas.DataFrame format.
- Calculation of porosity measure for each material in 2D / 3D datasets.
- Visualization in 2D / 3D using matplotlib, visvis packages.



## 1.1 Overview

### 1.1.1 Requirements

- Python 2.7
- PyCUDA >= 2017.1.1
- Numpy >= 1.13
- SciPy >= 0.19
- scikit-image >= 0.12
- scikit-learn >= 0.18
- Pandas >= 0.19
- Matplotlib >= 2.0

### 1.1.2 Installation

The easiest way to install the latest version is by using pip:

```
$ pip install quanfima
```

You may also use Git to clone the repository and install it manually:

```
$ git clone https://github.com/rshkarin/quantfima.git
$ cd quanfima
$ python setup.py install
```

## 1.2 Quickstart

### 1.2.1 Analysis of fibrous 2D data

Open a grayscale image, perform segmentation, estimate porosity, analyze fiber orientation and diameters, and plot the results.

```
import numpy as np
from skimage import io, filters
from quanfima import morphology as mrph
from quanfima import visualization as vis
from quanfima import utils

img = io.imread('../data/polymer_slice.tif')

th_val = filters.threshold_otsu(img)
img_seg = (img > th_val).astype(np.uint8)


# estimate porosity
pr = mrph.calc_porosity(img_seg)
for k,v in pr.items():
    print 'Porosity ({}): {}'.format(k, v)

# prepare data and analyze fibers
data, skeleton, skeleton_thick = utils.prepare_data(img_seg)
cskel, fskel, omap, dmap, ovals, dvals = \
    mrph.estimate_fiber_properties(data, skeleton)

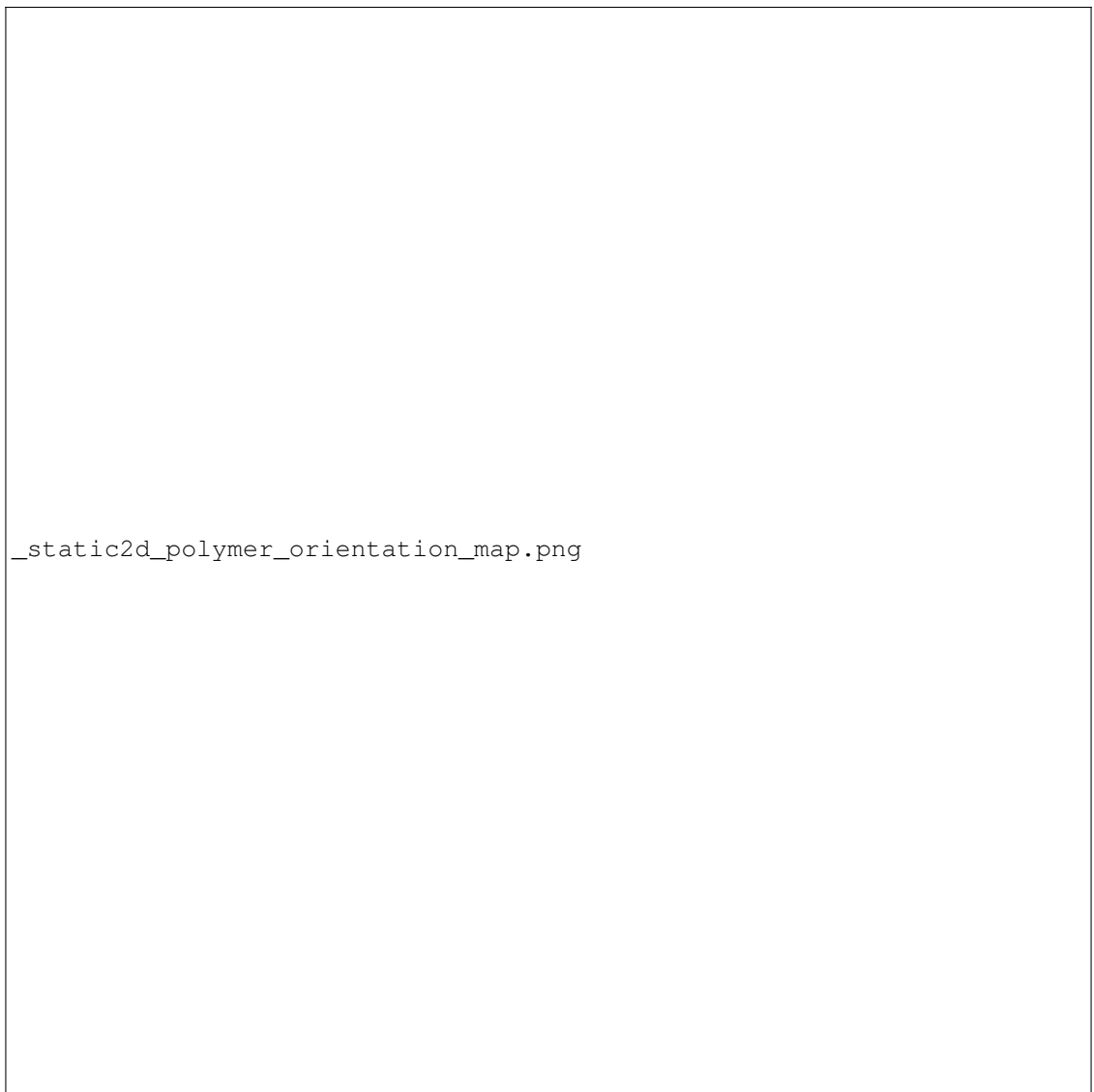
# plot results
vis.plot_orientation_map(omap, fskel, min_label=u'0°', max_label=u'180°',
                        figsize=(10,10),
                        name='2d_polymer',
                        output_dir='../data/results')
vis.plot_diameter_map(dmap, cskel, figsize=(10,10), cmap='gist_rainbow',
                    name='2d_polymer',
                    output_dir='../data/results')
```

```
>> Porosity (Material 1): 0.845488888889
```





`_static2d_polymer_data.png`



`_static2d_polymer_orientation_map.png`

\_static2d\_polymer\_diameter\_map.png

### 1.2.2 Analysis of 3D data of fibrous material

Open a micro-CT dataset, perform slice-wise segmentation, estimate porosity, analyze 3D fiber orientation and diameters, and visualize the results.

```
import numpy as np
from skimage import filters
from quantfima import morphology as mrph
from quantfima import visualization as vis
from quantfima import utils

data = np.memmap('../data/polymer3d_8bit_128x128x128.raw',
                  shape=(128,128,128), dtype=np.uint8, mode='r')

data_seg = np.zeros_like(data, dtype=np.uint8)
for i in xrange(data_seg.shape[0]):
    th_val = filters.threshold_otsu(data[i])
```

(continues on next page)

(continued from previous page)

```

data_seg[i] = (data[i] > th_val).astype(np.uint8)

# estimate porosity
pr = mrph.calc_porosity(data_seg)
for k,v in pr.items():
    print 'Porosity ({}): {}'.format(k, v)

# prepare data and analyze fibers
pdata, pskel, pskel_thick = utils.prepare_data(data_seg)
oprops = mrph.estimate_tensor_parallel('polymer_orientation_w32', pskel,
                                       pskel_thick, 32,
                                       '../../data/results')

odata = np.load(oprops['output_path']).item()
lat, azth, skel = odata['lat'], odata['azth'], odata['skeleton']


dprops = mrph.estimate_diameter_single_run('polymer_diameter',
                                           '../../data/results',
                                           pdata, skel, lat, azth)
dmtr = np.load(dprops['output_path']).item()['diameter']

# plot results
vis.plot_3d_orientation_map('polymer_w32', lat, azth,
                           output_dir='../../data/results',
                           camera_azth=40.47,
                           camera_elev=32.5,
                           camera_fov=35.0,
                           camera_loc=(40.85, 46.32, 28.85),
                           camera_zoom=0.005124)


vis.plot_3d_diameter_map('polymer_w32', dmtr,
                        output_dir='../../data/results',
                        measure_quantity='vox',
                        camera_azth=40.47,
                        camera_elev=32.5,
                        camera_fov=35.0,
                        camera_loc=(40.85, 46.32, 28.85),
                        camera_zoom=0.005124,
                        cb_x_offset=5,
                        width=620)

```

```
>> Porosity (Material 1): 0.855631351471
```



`_staticpolymer_w32_3d_orientation.png`



\_staticpolymer\_w32\_3d\_diameter.png

### 1.2.3 Estimation of p-values

Estimate p-values between several groups of samples with corresponding measurements of some material's property.

```
import pandas as pd
from quanfima import utils

prop_vals = [174.93, 182.42, 194.61, 234.6, 229.73, 242.6, 38.78, 37.79,
             32.06, 14.81, 15.23, 13.84]
mat_groups = ['PCL_cl', 'PCL_cl', 'PCL_cl', 'PCL_wa', 'PCL_wa', 'PCL_wa',
              'PCL_SiHA_cl', 'PCL_SiHA_cl', 'PCL_SiHA_cl', 'PCL_SiHA_wa',
              'PCL_SiHA_wa', 'PCL_SiHA_wa']
df_elongation = pd.DataFrame({'elongation': prop_vals, 'type': mat_groups})

_, _ = utils.calculate_tukey_posthoc(df_elongation, 'elongation',
                                     name='samples_elongation',
                                     write=True,
                                     output_dir='../data/results')
```

```
>> Tukey post-hoc (elongation)
>> Multiple Comparison of Means - Tukey HSD,FWER=0.05
```

(continues on next page)

(continued from previous page)

```
>> =====
>>      group1      group2  meandiff  lower    upper    reject
>> -----
>> PCL_SiHA_cl PCL_SiHA_wa -21.5833 -37.8384 -5.3282   True
>> PCL_SiHA_cl   PCL_cl    147.7767 131.5216 164.0318   True
>> PCL_SiHA_cl   PCL_wa    199.4333 183.1782 215.6884   True
>> PCL_SiHA_wa   PCL_cl     169.36  153.1049 185.6151   True
>> PCL_SiHA_wa   PCL_wa    221.0167 204.7616 237.2718   True
>>      PCL_cl      PCL_wa    51.6567  35.4016  67.9118   True
>> -----
>> ['PCL_SiHA_cl' 'PCL_SiHA_wa' 'PCL_cl' 'PCL_wa']
>> PCL_SiHA_cl-PCL_SiHA_wa :  0.011919282004
>> PCL_SiHA_wa-PCL_cl :  0.001
>> PCL_SiHA_wa-PCL_wa :  0.001
>> PCL_cl-PCL_wa :  0.001
>> PCL_SiHA_cl-PCL_wa :  0.001
>> PCL_SiHA_cl-PCL_cl :  0.001
```

## 1.2.4 Simulate and count particles in 3D data

Count and estimate properties of particles in a generated dataset comprised of spheres of varying radius.

```
from quanfima import simulation
from quanfima import morphology as mrph

volume, diameter, _, _ = simulation.simulate_particles((512,512,512),
                                                       n_particles=1000)

stats, labeled_volume = mrph.object_counter(volume)
```

```
>>      Label      Area      Perimeter  Sphericity
>> 0         1.0    20479.0    2896.958728    1.249533
>> 1         2.0    5575.0    1184.028571    1.284158
>> 2         3.0    57777.0   5816.142853    1.242660
>> 3         4.0    17077.0   2545.194226    1.260001
>> 4         5.0     5575.0   1184.028571    1.284158
>> 5         6.0    65267.0   6348.926691    1.234752
>> ..         ...         ...         ...         ...
>> 791      792.0     2109.0    605.185858    1.314154
>> 792      793.0      257.0    134.225397    1.456369
>> 793      794.0      257.0    134.225397    1.456369
>> 794      795.0     123.0     78.627417    1.521179

>> [795 rows x 4 columns]
```

## 1.2.5 Simulate fibers and estimate properties

Simulate a 3D dataset containing some number of fibers, estimate their properties and visualize.

```
import numpy as np
from scipy import ndimage as ndi
from skimage import morphology
from quanfima import simulation
```

(continues on next page)

(continued from previous page)

```

from quanfima import morphology as mrph
from quanfima import utils
from quanfima import visualization as vis

volume, lat_ref, azth_ref, diameter, _, _ = \
    simulation.simulate_fibers((128,128,128), n_fibers=30, max_fails=100,
                              radius_lim=(2, 3), gap_lim=(3,5))

volume = volume.astype(np.uint8)
volume = ndi.binary_fill_holes(volume)
volume = ndi.median_filter(volume, footprint=morphology.ball(2))
lat_ref = ndi.median_filter(lat_ref, footprint=morphology.ball(2))
azth_ref = ndi.median_filter(azth_ref, footprint=morphology.ball(2))

# prepare data and analyze fibers
pdata, pskel, pskel_thick = utils.prepare_data(volume)
oprops = mrph.estimate_tensor_parallel('dataset_orientation_w36',
                                       pskel, pskel_thick, 36,
                                       '../..data/results')

odata = np.load(oprops['output_path']).item()
lat, azth, skel = odata['lat'], odata['azth'], odata['skeleton']


dprops = mrph.estimate_diameter_single_run('dataset_diameter',
                                           '../..data/results',
                                           pdata, skel, lat, azth)
dmtr = np.load(dprops['output_path']).item()['diameter']

# plot results
vis.plot_3d_orientation_map('dataset_w36', lat, azth,
                           output_dir='../..data/results',
                           camera_azth=40.47,
                           camera_elev=32.5,
                           camera_fov=35.0,
                           camera_loc=(40.85, 46.32, 28.85),
                           camera_zoom=0.005124)

vis.plot_3d_diameter_map('dataset_w36', dmtr,
                        output_dir='../..data/results',
                        measure_quantity='vox',
                        camera_azth=40.47,
                        camera_elev=32.5,
                        camera_fov=35.0,
                        camera_loc=(40.85, 46.32, 28.85),
                        camera_zoom=0.005124,
                        cb_x_offset=5,
                        width=620)

```





`_staticdataset_w36_3d_orientation.png`



## 1.3 API Reference

### 1.3.1 quanfima.morphology module

### 1.3.2 quanfima.simulation module

`quanfima.simulation.additive_noise` (*params*, *noise\_lvl*, *smooth\_lvl*, *use\_median=True*, *median\_rad=3*)

Contaminates datasets with a specified additive Gaussian noise and smoothing level.

Contaminates the datasets (generated with *generate\_datasets* function) with the specified level of additive Gaussian noise and smoothing, uneven illumination can be added by extracting *blobs* from *params* tuple with some other arguments.

**params** [tuple] Contains *name*, *dataset\_path*, *blobs*, *output\_dir* arguments passed from *generate\_noised\_data*.

**noise\_level** [float] Indicates the standard deviations of noise.

**smooth\_level** [float] Indicates the sigma value of Gaussian filter.

**use\_median** [boolean] Specifies if the median filter should be applied after addition of noise.

**median\_rad** [integer] Indicates the size of median filter.

**datasets\_props** [dict] The dictionary containing the path to the reference dataset, the path to the contaminated dataset, the generated name, the SNR level, the precision, the recall and f1-score, and the level of noise and smoothing.

```
quanfima.simulation.generate_blobs(volume_size, blob_size_fraction=0.1, trans-
                                parency_ratio=0.5, sigma=90.0)
```

Generates random blobs smoothed with Gaussian filter in a volume.

Generates several blobs of random size in a volume using function from scikit-image, which are subsequently smoothed with a Gaussian filter of a large sigma to imitate 3D uneven illumination of the volume.

**volume\_size** [tuple] Indicates the size of the volume.

**blob\_size\_fraction** [float] Indicates the fraction of volume occupied by blobs.

**transparency\_ratio** [float] Indicates the transparency of blobs in a range [0, 1].

**sigma** [float] Indicates the sigma of Gaussian filter.

**blobs\_smeared** [ndarray] The volume with smoothed blobs of a specified transparency.

```
quanfima.simulation.generate_datasets(volume_size=(512, 512, 512), n_fibers=50, ra-
                                dius_lim=(4, 10), length_lim=(0.2, 0.8), gap_lim=(3,
                                10), max_fails=100, median_rad=3, intersect=False,
                                output_dir=None, params=None)
```

Simulates specified configurations of fibers and stores in a npy file.

Simulates a number of fiber configurations specified in *params* with *n\_fibers* of the radii and lengths in ranges *radius\_lim* and *length\_lim*, separated with gaps in a range of *gap\_lim*. The simulation process stops if the number of attempts to generate a fiber exceeds *max\_fails*.

**volume\_size** [tuple] Indicates the size of the volume.

**n\_fibers** [integer] Indicates the number of fibers to be generated.

**radius\_lim** [tuple] Indicates the range of radii for fibers to be generated.

**length\_lim** [tuple] Indicates the range of lengths for fibers to be generated.

**gap\_lim** [tuple] Indicates the range of gaps separating the fibers from each other.

**max\_fails** [integer] Indicates the maximum number of failures during the simulation process.

**median\_rad** [integer] Indicates the radius of median filter to fill holes occurred due to rounding of coordinates of the generated fibers.

**intersect** [boolean] Specifies if generated fibers can intersect.

**output\_dir** [str] Indicates the path to the output folder where the data will be stored.

**params** [dict] Indicates the configurations of orientation of fibers to be generated.

**out** [dict] The dictionary of generated datasets of specified configurations.

```
quanfima.simulation.generate_noised_data(datasets_path, noise_levels=[0.0, 0.15, 0.3],
                                smooth_levels=[0.0, 1.0, 2.0], blobs=None,
                                use_median=True, median_rad=3, out-
                                put_dir=None, n_processes=9)
```

Contaminates datasets with a specified additive Gaussian noise and smoothing level.

Contaminates the datasets (generated with *generate\_datasets* function) with the specified level of additive Gaussian noise and smoothing, uneven illumination can be added if *blobs* is provided. The contaminating process can be performed in a parallel *n\_processes* processes.

**datasets\_path** [str] Indicates the path to dataset.

**noise\_levels** [array] Indicates the array of standard deviations of noise.

**smooth\_levels** [array] Indicates the array of sigma values of Gaussian filter.

**blobs** [ndarray] Indicates the volume of uneven illumination generated by *generate\_blobs*.

**use\_median** [boolean] Specifies if the median filter should be applied after addition of noise.

**median\_rad** [integer] Indicates the size of median filter.

**output\_dir** [str] Indicates the path to the output folder where the data will be stored.

**n\_processes** [integer] Indicates the number of parallel processes.

**results** [array of dicts] The array of dictionaries containing paths to contaminated datasets, and other properties.

```
quanfima.simulation.generate_particle_dataset (volume_size=(512, 512, 512),
                                              n_particles=500, radius_lim=(4, 10),
                                              max_fails=100, output_dir=None)
```

Simulates a specified number of particles and stores complete dataset in a npy file.

**volume\_size** [tuple] Indicates the size of the volume.

**n\_particles** [integer] Indicates the number of particles to be generated.

**radius\_lim** [tuple] Indicates the range of radii for particles to be generated.

**max\_fails** [integer] Indicates the maximum number of failures during the simulation process.

**output\_dir** [str] Indicates the path to the output folder where the data will be stored.

**out** [dict] The dictionary of generated dataset.

```
quanfima.simulation.mkfiber (dims_size, length, radius, azth, lat, offset_xyz)
```

Computes fiber coordinates and its length.

Computes a fiber of specified *length*, *radius*, oriented under azimuth *azth* and latitude / elevation *lat* angles shifted to *offset\_xyz* from the center of a volume of size *dims\_size*.

**dims\_size** [tuple] Indicates the size of the volume.

**length** [integer] Indicates the length of the simulated fiber.

**radius** [integer] Indicates the radius of the simulated fiber.

**azth** [float] Indicates the azimuth component of the orientation angles of the fiber in radians.

**lat** [float] Indicates the latitude / elevation component of the orientation angles of the fiber in radians.

**offset\_xyz** [tuple] Indicates the offset from the center of the volume where the fiber will be generated.

**fiber\_pts, fiber\_len** [tuple of array and number] The array of fiber coordinates and the length.

```
quanfima.simulation.random_in (rng, number=1)
```

Returns a random value within a given range.

```
quantfima.simulation.simulate_fibers(volume_shape, n_fibers=1, radius_lim=(4, 10),
                                     length_lim=(0.2, 0.8), lat_lim=(0, 3.141592653589793),
                                     azth_lim=(0, 3.141592653589793), gap_lim=(3, 10),
                                     max_fails=10, max_len_loss=0.5, intersect=False)
```

Simulates fibers in a volume.

Simulates *n\_fibers* of the radii and lengths in ranges *radius\_lim* and *length\_lim*, oriented in a range of azimuth *azth\_lim* and latitude elevation ‘lat\_lim’ angles, separated with a gap in a range of *gap\_lim*. The simulation process stops if the number of attempts to generate a fiber exceeds *max\_fails*.

**volume\_shape** [tuple] Indicates the size of the volume.

**n\_fibers** [integer] Indicates the number of fibers to be generated.

**radius\_lim** [tuple] Indicates the range of radii for fibers to be generated.

**length\_lim** [tuple] Indicates the range of lengths for fibers to be generated.

**lat\_lim** [tuple] Indicates the range of latitude / elevation component of the orientation angles of the fibers to be generated.

**azth\_lim** [tuple] Indicates the range of azimuth component of the orientation angles of the fibers to be generated.

**gap\_lim** [tuple] Indicates the range of gaps separating the fibers from each other.

**max\_fails** [integer] Indicates the maximum number of failures during the simulation process.

**max\_len\_loss** [float] Indicates the maximum fraction of the generated fiber placed out of volume, exceeding which the fiber is counted as failed.

**intersect** [boolean] Specifies if generated fibers can intersect.

**(volume, lat\_ref, azth\_ref, diameter, n\_generated, elapsed\_time)** [tuple of arrays and numbers] The binary volume of generated fibers, the volumes of latitude / elevation and azimuth angles at every fiber point, the volume of diameters at every fibers point, the number of generated fibers and the simulation time.

```
quantfima.simulation.simulate_particles(volume_shape, n_particles=1, radius_lim=(3, 30),
                                     max_fails=10)
```

Simulates particles in a volume.

Simulates *n\_particles* of the radii in a range *radius\_lim*. The simulation process stops if the number of attempts to generate a particle exceeds *max\_fails*.

**volume\_shape** [tuple] Indicates the size of the volume.

**n\_particles** [integer] Indicates the number of particles to be generated.

**radius\_lim** [tuple] Indicates the range of radii for particles to be generated.

**max\_fails** [integer] Indicates the maximum number of failures during the simulation process.

**(volume, diameter, n\_generated, elapsed\_time)** [tuple of arrays and numbers] The binary volume of generated particles, the volume of diameters at every point of particles, the number of generated particles and the simulation time.

```
quantfima.simulation.unpack_additive_noise(args)
Unpack arguments and return result of additive_noise function.
```

### 1.3.3 quanfima.utils module

### 1.3.4 quanfima.visualization module

## 1.4 ChangeLog

### 1.4.1 Version 0.1a1

Features:

- Orientation and diameter estimation for 2D / 3D data
- Counting of particles with subsequent estimation of properties
- Porosity calculation for ND data
- Visualization of 2D orientation and diameter with MPL package
- Visualization of 3D orientation and diameter with VisVis package
- Simulation of fibers and particles with various scenarios

### 1.4.2 Version 0.1a2

Fix:

- Incorrect display of measure units at a legend of a pie chart

### 1.4.3 Version 0.1a3

New features:

- Block-wise orientation estimation with 2D FFT and PCA-approach [estimate\_fourier\_orientation]
- Visualization of block-wise orientation with MPL package [plot\_fourier\_orientation]

## q

`quantfima.simulation`, [14](#)





## A

`additive_noise()` (*in module `quanfima.simulation`*), 14

## G

`generate_blobs()` (*in module `quanfima.simulation`*), 15

`generate_datasets()` (*in module `quanfima.simulation`*), 15

`generate_noised_data()` (*in module `quanfima.simulation`*), 15

`generate_particle_dataset()` (*in module `quanfima.simulation`*), 16

## M

`mkfiber()` (*in module `quanfima.simulation`*), 16

## Q

`quanfima.simulation` (*module*), 14

## R

`random_in()` (*in module `quanfima.simulation`*), 16

## S

`simulate_fibers()` (*in module `quanfima.simulation`*), 16

`simulate_particles()` (*in module `quanfima.simulation`*), 17

## U

`unpack_additive_noise()` (*in module `quanfima.simulation`*), 17