qpsphere Documentation

Release 0.4.0

Paul Müller

Jul 27, 2018

Contents:

1 Introduction **3**
- 1.1 The decoupling problem . . . 3
- 1.2 Why qpsphere? . . . 3
- 1.3 Citing qpsphere . . . 4

2 Getting started **5**
- 2.1 Installing qpsphere . . . 5
- 2.2 User API . . . 5
  - 2.2.1 Basic usage . . . 5
  - 2.2.2 Choosing method and model . . . 6

3 Examples **7**
- 3.1 Refractive index and radius determination . . . 7
  - 3.1.1 OPD edge-detection approach with a single cell . . . 7
  - 3.1.2 Comparison of light-scattering models . . . 9
  - 3.1.3 OPD projection image fit applied to an OPD simulation . . . 11
  - 3.1.4 Rytov-SC image fit applied to a Mie simulation . . . 12
- 3.2 Other examples . . . 13

4 Code reference **15**
- 4.1 analyze (Generic fitting wrapper) . . . 15
- 4.2 cnvnc (Convenience methods) . . . 15
- 4.3 edgefit (Contour-based fitting) . . . 17
- 4.4 imagefit (2D phase image fitting) . . . 19
  - 4.4.1 imagefit.alg (Algorithms) . . . 20
  - 4.4.2 imagefit.interp (Image interpolation logic) . . . 22
- 4.5 models (Scattering models) . . . 24
- 4.6 util (Helper methods) . . . 24

5 Bilbliography **25**

6 Indices and tables **27**

Bibliography **29**

Python Module Index **31**

Qpsphere is a Python3 library for analyzing spherical objects in quantitative phase imaging. This includes the extraction of average refractive index and object radius, for which several methods are implemented. This is the documentation of qpsphere version 0.4.0.

CHAPTER 1

Introduction

1.1 The decoupling problem

Quantitative phase imaging (QPI) measures the phase retardation $\phi(x, y)$ introduced by an object, such as a cell, to a light field, usually a plane wave. Under the assumption that light travels along straight lines, the measured phase $\phi(x, y)$ can be described as the projection of the refractive index (RI) $n(x, y, z)$ of the imaged object onto the detector plane

$$\phi(x, y) = \frac{2\pi}{\lambda} \int (n(x, y, z) - n_{\text{med}})\, dz,$$

with the vacuum wavelength λ of the imaging light and the RI of the medium surrounding the object n_{med}. This equation describes a coupling between the RI of the imaged object and its shape: From a single phase image, it is not possible to compute the RI of an object without knowing its shape and vice versa. Moreover, it is not possible to infer the RI of an object from its shape if the RI is not constant. Thus, in general, it is not possible to solve the coupling problem in QPI. However, if the object has a spherical shape and if its RI is constant, it is possible to infer the radius R and the RI n from a single phase measurement. The equation above then reduces to

$$\phi(x, y) = \frac{4\pi}{\lambda}(n - n_{\text{med}}) \cdot \sqrt{R^2 - (x - x_0)^2 - (y - y_0)^2}$$

with the lateral position of the sphere (x_0, y_0). Note that this approach is often applied to suspended (spherical) cells. Even though cells are complex structured objects, this approach yields good estimates for the average RI and radius. Qpsphere offers several approaches to address this decoupling problem, ranging from simple edge-detection to image fits with a 2D Mie model.

1.2 Why qpsphere?

The purpose of qpsphere is to make our QPI image analysis tools (*[SSM+15] [SSM+16] [MSG+18]*) available to other research groups. Qpsphere makes heavy use of qpimage, a resourceful QPI image manager and is a key component of our QPI analysis software DryMass.

1.3 Citing qpsphere

If you are using qpsphere in a scientific publication, please cite it with:

```
(...) using qpsphere version X.X.X (available at
https://pypi.python.org/pypi/qpsphere).
```

or in a bibliography

```
Paul Müller (2017), qpsphere version X.X.X: Phase image analysis
[Software]. Available at https://pypi.python.org/pypi/qpsphere.
```

and replace `X.X.X` with the version of qpsphere that you used.

Furthermore, several ideas implemented in qpsphere have been described and published in scientific journals:

- Retrieval of RI and radius using the OPD edge-detection approach is described in *[SSM+15]* and *[SSM+16]* (`method="edge"` in `qpsphere.analyze()`).
- Retrieval of RI and radius by fitting 2D models (OPD projection, Rytov approximation, systematically corrected Rytov approximation, Mie) to phase images is described in *[MSG+18]*. (`method="image"` in `qpsphere.analyze()` and the corresponding scattering models in `qpsphere.models`).

CHAPTER 2

Getting started

2.1 Installing qpsphere

Qpsphere is written in pure Python and supports Python version 3.6 and higher. Qpsphere depends on several other scientific Python packages, including:

- numpy,
- scipy,
- lmfit (contour fitting),
- scikit-image (segmentation).
- qpimage (phase data manipulation),

To install qpsphere, use one of the following methods (package dependencies will be installed automatically):

- **from PyPI:** `pip install qpsphere`
- **from sources:** `pip install .` or `python setup.py install`

2.2 User API

The methods in qpsphere for determining radius and refractive index of spherical objects from quantitative phase images can be divided into edge-based (*`qpsphere.edgefit`*) and image-based (*`qpsphere.imagefit`*). The method *`qpsphere.analyze()`* combines both in a user-convenient interface.

2.2.1 Basic usage

The high-level API of qpsphere heavily relies on qpimage for accessing meta data to convert in-between units (e.g. pixel size in meters). If the experimental data file format is supported by qpformat, this leads to particularly clean and simple code;

```
import qpformat
import qpsphere

# load an experimental data set
ds = qpformat.load_data(path="path/to/data_file",
                        # set pixel size in meters
                        meta_data={"pixel size": .12e-6}
                        )
# get QPImage instance (also contains meta data, e.g. pixel size)
qpi_sensor = ds.get_qpimage()
# obtain the region of interest containing a spherical object
qpi_object = qpi_sensor[100:300, 50:250]
# determine the refractive index and radius of the object
n, r = qpsphere.analyze(qpi=qpi_object,
                        # estimate of the object's radius in meters
                        r0=10e-6,
                        # OPD edge-detection approach
                        method="edge"
                        # OPD projection model
                        model="projection"
                        )
```

where `n` is the average refractive index (RI) and `r` is the radius of the object in meters estimated by the optical path difference (OPD) edge-detection approach.

2.2.2 Choosing method and model

Although the OPD edge-detection approach is fast, it is inaccurate because it is resolution-dependent and because it approximates light scattering by an integral over the RI along straight lines. Higher accuracy can be achieved by fitting a 2D model to the experimental phase image. When setting `method="image"` in the example above, the following models are available:

- "mie": a full Mie model (very slow)
- "mie-avg": a polarization-averaged Mie model (faster than "mie")
- "projection": an OPD projection model
- "rytov": the Rytov approximation
- "rytov-sc": the systematically corrected Rytov approximation

A comparison of these models can be found in *[MSG+18]*.

CHAPTER 3

Examples

3.1 Refractive index and radius determination

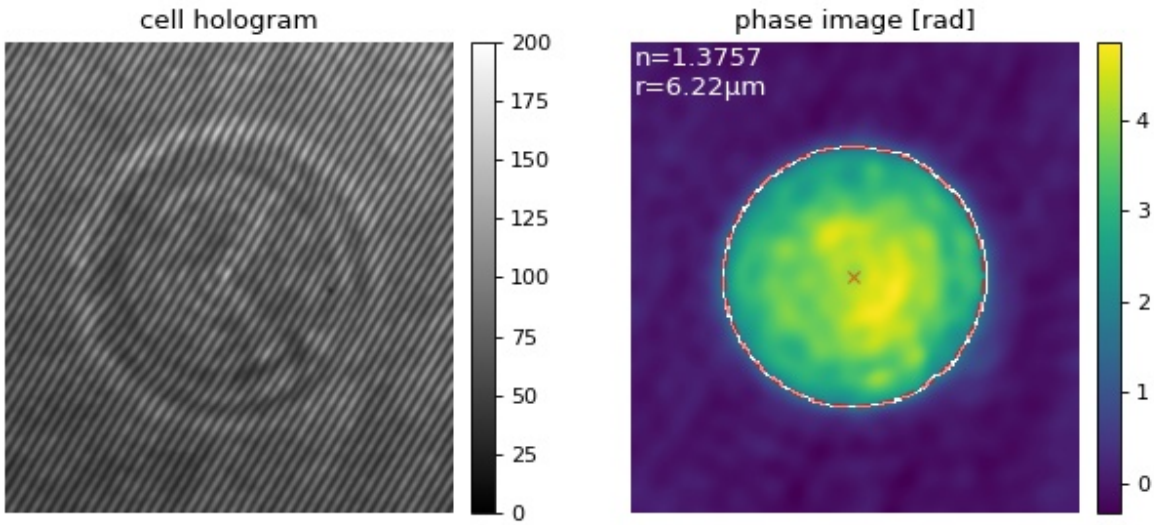
3.1.1 OPD edge-detection approach with a single cell

This example illustrates how qpsphere can be used to determine the radius and the refractive index of a spherical cell. The hologram of the myeloid leukemia cell (HL60) on the left was recorded using digital holographic microscopy (DHM). In the quantitative phase image on the right, the detected cell contour (white) and the subsequent circle fit (red) as well as the resulting average radius and refractive index of the cell are shown. The setup used for recording these data is described in *[SSM+15]* which also contains a description of the basic steps to determine the position and radius of the cell and to subsequently compute the average refractive index from the experimental phase data. The experimental data is loaded and background-corrected using `qpimage`.

`cell_edge.py`

```
import matplotlib
import matplotlib.pylab as plt
import numpy as np
import qpimage
import qpsphere

# load the experimental data
edata = np.load("./data/hologram_cell.npz")

# create QPImage instance
qpi = qpimage.QPImage(data=edata["data"],
                      bg_data=edata["bg_data"],
                      which_data="hologram",
                      meta_data={"wavelength": 633e-9,
                                 "pixel size": 0.107e-6,
                                 "medium index": 1.335
                                 }
                      )
```

(continues on next page)



(continued from previous page)

```

# background correction
qpi.compute_bg(which_data=["amplitude", "phase"],
               fit_offset="fit",
               fit_profile="tilt",
               border_px=5,
               )

# determine radius and refractive index, guess the cell radius: 10µm
n, r, (cx, cy), edge = qpsphere.edgefit.analyze(qpi=qpi,
                                                r0=10e-6,
                                                ret_center=True,
                                                ret_edge=True)

# plot results
fig = plt.figure(figsize=(8, 4))
matplotlib.rcParams["image.interpolation"] = "bicubic"
holkw = {"cmap": "gray",
         "vmin": 0,
         "vmax": 200}
# hologram image
ax1 = plt.subplot(121, title="cell hologram")
map1 = ax1.imshow(edata["data"].T, **holkw)
plt.colorbar(map1, ax=ax1, fraction=.048, pad=0.04)
# phase image
ax2 = plt.subplot(122, title="phase image [rad]")
map2 = ax2.imshow(qpi.pha.T)
# edge
edgeplot = np.ma.masked_where(edge == 0, edge)
ax2.imshow(edgeplot.T, cmap="gray_r", interpolation="none")
# fitted circle center
plt.plot(cx, cy, "xr", alpha=.5)
# fitted circle perimeter
```

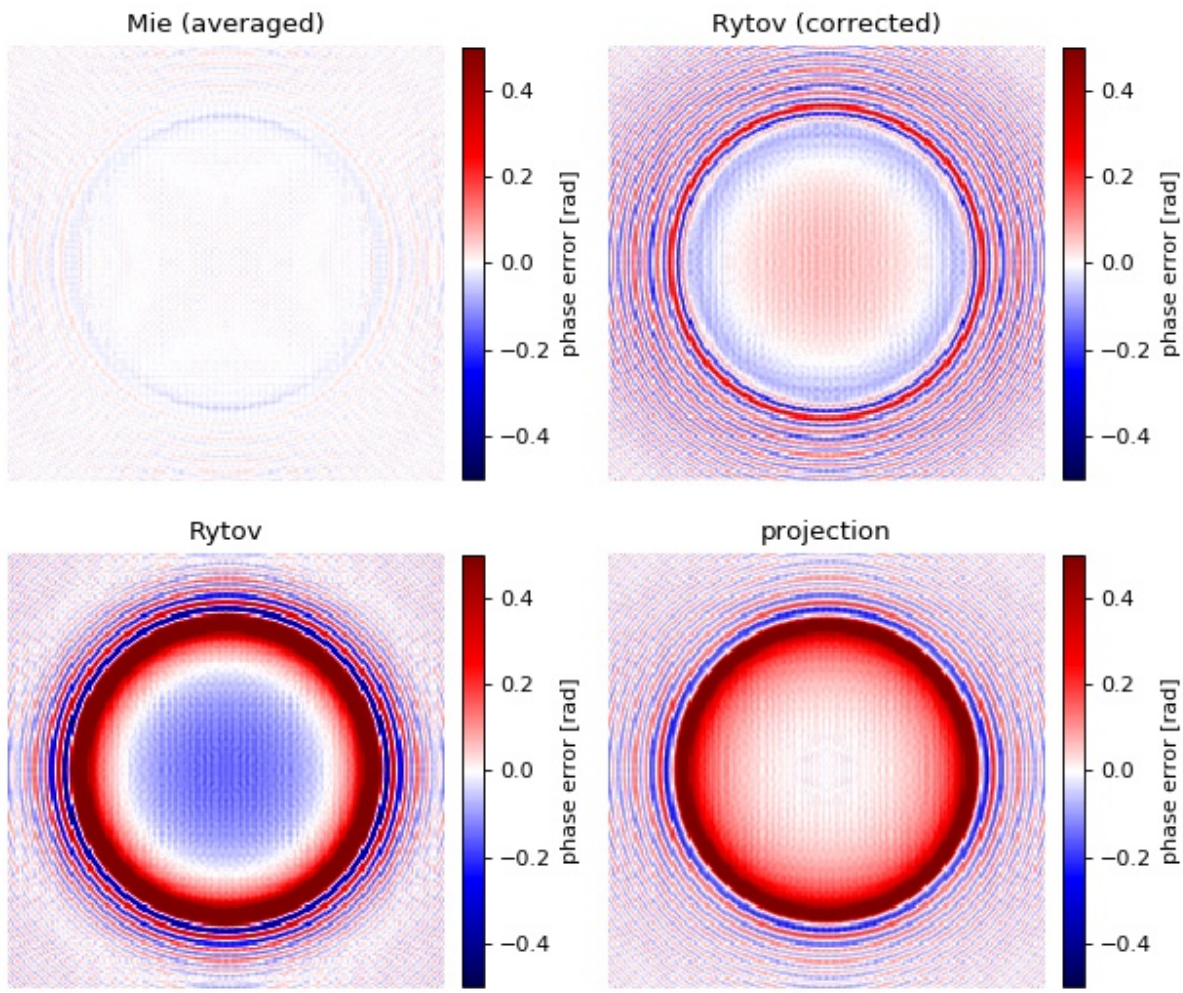
(continues on next page)

(continued from previous page)

```
circle = plt.Circle((cx, cy), r / qpi["pixel size"],
                    color='r', fill=False, ls="dashed", lw=2, alpha=.5)
ax2.add_artist(circle)
# fitting results as text
info = "n={:.4F}\nr={:.2f}µm".format(n, r * 1e6)
ax2.text(.8, .8, info, color="w", fontsize="12", verticalalignment="top")
plt.colorbar(map2, ax=ax2, fraction=.048, pad=0.04)
# disable axes
[ax.axis("off") for ax in [ax1, ax2]]

plt.tight_layout()
plt.show()
```

3.1.2 Comparison of light-scattering models

The phase error map allows a comparison of the ability of the modeling methods implemented in qpsphere to reproduce the phase delay introduced by a dielectric sphere. For a quantitative comparison, see reference *[MSG+18]*.

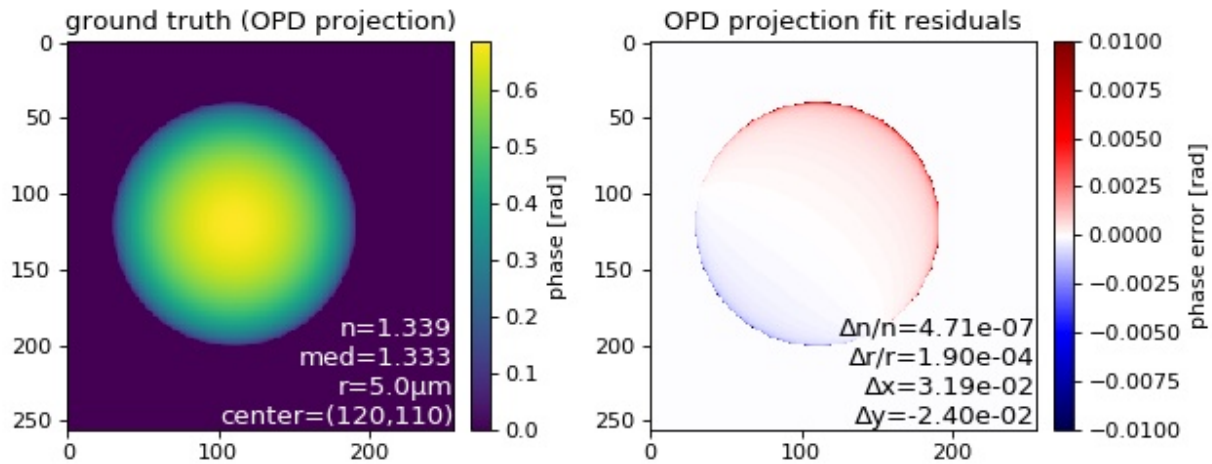


sphere_models.py

```
import matplotlib.pylab as plt
import qpsphere

kwargs = {"radius": 10e-6,  # 10µm
          "sphere_index": 1.380,  # cell
          "medium_index": 1.335,  # PBS
          "wavelength": 647.1e-9,  # krypton laser
          "grid_size": (200, 200),
          }

px_size = 3 * kwargs["radius"] / kwargs["grid_size"][0]
kwargs["pixel_size"] = px_size

# mie (long computation time)
qpi_mie = qpsphere.simulate(model="mie", **kwargs)

# mie averaged
qpi_mie_avg = qpsphere.simulate(model="mie-avg", **kwargs)

# rytov corrected
qpi_ryt_sc = qpsphere.simulate(model="rytov-sc", **kwargs)

# rytov
qpi_ryt = qpsphere.simulate(model="rytov", **kwargs)

# projection
qpi_proj = qpsphere.simulate(model="projection", **kwargs)

kwargs = {"vmin": -.5,
          "vmax": .5,
          "cmap": "seismic"}

plt.figure(figsize=(8, 6.8))

ax1 = plt.subplot(221, title="Mie (averaged)")
pmap = plt.imshow(qpi_mie.pha - qpi_mie_avg.pha, **kwargs)

ax2 = plt.subplot(222, title="Rytov (corrected)")
plt.imshow(qpi_mie.pha - qpi_ryt_sc.pha, **kwargs)

ax3 = plt.subplot(223, title="Rytov")
plt.imshow(qpi_mie.pha - qpi_ryt.pha, **kwargs)

ax4 = plt.subplot(224, title="projection")
plt.imshow(qpi_mie.pha - qpi_proj.pha, **kwargs)

# disable axes
for ax in [ax1, ax2, ax3, ax4]:
    ax.axis("off")
    plt.colorbar(pmap, ax=ax, fraction=.045, pad=0.04,
                 label="phase error [rad]")

plt.tight_layout(w_pad=0, h_pad=0)
plt.show()
```

3.1.3 OPD projection image fit applied to an OPD simulation

This examples illustrates how the refractive index and radius of a sphere can be determined using the 2D (image-based) phase fitting algorithm.



imagefit_projection.py

```
import matplotlib.pylab as plt

import qpsphere

# run simulation with projection model
r = 5e-6
n = 1.339
med = 1.333
c = (120, 110)
qpi = qpsphere.simulate(radius=r,
                        sphere_index=n,
                        medium_index=med,
                        wavelength=550e-9,
                        grid_size=(256, 256),
                        model="projection",
                        center=c)

# fit simulation with projection model
n_fit, r_fit, c_fit, qpi_fit = qpsphere.analyze(qpi=qpi,
                                                r0=4e-6,
                                                method="image",
                                                model="projection",
                                                imagekw={"verbose": 1},
                                                ret_center=True,
                                                ret_qpi=True)

# plot results
fig = plt.figure(figsize=(8, 3.5))
txtkwargs = {"verticalalignment": "bottom",
             "horizontalalignment": "right",
             "fontsize": 12}
```

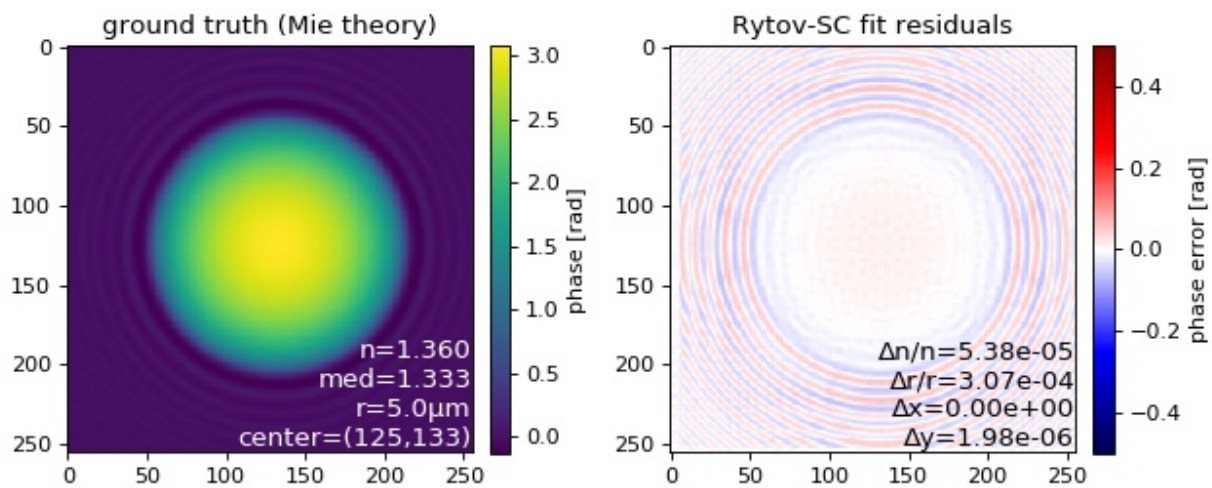
(continues on next page)

(continued from previous page)

```

ax1 = plt.subplot(121, title="ground truth (OPD projection)")
map1 = ax1.imshow(qpi.pha)
plt.colorbar(map1, ax=ax1, fraction=.046, pad=0.04, label="phase [rad]")
t1 = "n={:.3f}\nmed={:.3f}\nr={:.1f}µm\ncenter=({:d},{:d})".format(
    n, med, r * 1e6, c[0], c[1])
ax1.text(1, 0, t1, transform=ax1.transAxes, color="w", **txtkwargs)

ax2 = plt.subplot(122, title="OPD projection fit residuals")
map2 = ax2.imshow(qpi.pha - qpi_fit.pha, vmin=-.01, vmax=.01, cmap="seismic")
plt.colorbar(map2, ax=ax2, fraction=.046, pad=0.04, label="phase error [rad]")
t2 = "Δn/n={:.2e}\nΔr/r={:.2e}\nΔx={:.2e}\nΔy={:.2e}".format(
    abs(n - n_fit) / n, abs(r - r_fit) / r,
    c_fit[0] - c[0], c_fit[1] - c[1])
ax2.text(1, 0, t2, transform=ax2.transAxes, color="k", **txtkwargs)

plt.tight_layout()
plt.show()
```

3.1.4 Rytov-SC image fit applied to a Mie simulation

This examples illustrates how the refractive index and radius of a sphere can be determined accurately using the 2D phase fitting algorithm with the systematically corrected Rytov approximation.



imagefit_rytov_sc.py

```
import matplotlib.pylab as plt

import qpsphere

# run simulation with averaged Mie model
r = 5e-6
n = 1.360
med = 1.333
c = (125, 133)
```

(continues on next page)

(continued from previous page)

```
qpi = qpsphere.simulate(radius=r,
                        sphere_index=n,
                        medium_index=med,
                        wavelength=550e-9,
                        grid_size=(256, 256),
                        model="mie-avg",
                        center=c)

# Fitting Mie simulations with the systematically corrected Rytov
# approximation (`model="rytov sc"`) yields lower parameter errors
# compared to the non-corrected Rytov approximation (`model="rytov"`).
n_fit, r_fit, c_fit, qpi_fit = qpsphere.analyze(qpi=qpi,
                                                r0=4e-6,
                                                method="image",
                                                model="rytov-sc",
                                                imagekw={"verbose": 1},
                                                ret_center=True,
                                                ret_qpi=True)

# plot results
fig = plt.figure(figsize=(8, 3.5))
txtkwargs = {"verticalalignment": "bottom",
             "horizontalalignment": "right",
             "fontsize": 12}

ax1 = plt.subplot(121, title="ground truth (Mie theory)")
map1 = ax1.imshow(qpi.pha)
plt.colorbar(map1, ax=ax1, fraction=.046, pad=0.04, label="phase [rad]")
t1 = "n={:.3f}\nmed={:.3f}\nr={:.1f}µm\ncenter=({:d},{:d})".format(
    n, med, r * 1e6, c[0], c[1])
ax1.text(1, 0, t1, transform=ax1.transAxes, color="w", **txtkwargs)

ax2 = plt.subplot(122, title="Rytov-SC fit residuals")
map2 = ax2.imshow(qpi.pha - qpi_fit.pha, vmin=-.5, vmax=.5, cmap="seismic")
plt.colorbar(map2, ax=ax2, fraction=.046, pad=0.04, label="phase error [rad]")
t2 = "Δn/n={:.2e}\nΔr/r={:.2e}\nΔx={:.2e}\nΔy={:.2e}".format(
    abs(n - n_fit) / n, abs(r - r_fit) / r,
    c_fit[0] - c[0], c_fit[1] - c[1])
ax2.text(1, 0, t2, transform=ax2.transAxes, color="k", **txtkwargs)

plt.tight_layout()
plt.show()
```

3.2 Other examples

- Background correction based on the mask image determined with the convenience method *qpsphere.cnvnc.bg_phase_mask_for_qpi()*.

CHAPTER 4

Code reference

4.1 analyze (Generic fitting wrapper)

qpsphere.**analyze**()

alias of *qpsphere.cnvnc.analyze()*

4.2 cnvnc (Convenience methods)

qpsphere.cnvnc.**analyze**(*qpi*, *r0*, *method='edge'*, *model='projection'*, *edgekw={}*, *imagekw={}*, *ret_center=False*, *ret_pha_offset=False*, *ret_qpi=False*)

Determine refractive index and radius of a spherical object

Parameters

- **qpi** (*qpimage.QPImage*) – Quantitative phase image data
- **r0** (*float*) – Approximate radius of the sphere [m]
- **method** (*str*) – The method used to determine the refractive index can either be "edge" (determine the radius from the edge detected in the phase image) or "image" (perform a 2D phase image fit).
- **model** (*str*) – The light-scattering model used by *method*. If *method* is "edge", only "projection" is allowed. If *method* is "image", *model* can be one of "mie", "projection", "rytov", or "rytov-sc".
- **edgekw** (*dict*) – Keyword arguments for tuning the edge detection algorithm, see *qpsphere.edgefit.contour_canny()*.
- **imagekw** (*dict*) – Keyword arguments for tuning the image fitting algorithm, see *qpsphere.imagefit.alg.match_phase()*
- **ret_center** (*bool*) – If True, return the center coordinate of the sphere.
- **ret_pha_offset** (*bool*) – If True, return the phase image background offset.

- **ret_qpi** (*bool*) – If True, return the modeled data as a `qpimage.QPImage`.

Returns

- **n** (*float*) – Computed refractive index
- **r** (*float*) – Computed radius [m]
- **c** (*tuple of floats*) – Only returned if *ret_center* is True; Center position of the sphere [px]
- **pha_offset** (*float*) – Only returned if *ret_pha_offset* is True; Phase image background offset
- **qpi_sim** (*qpimage.QPImage*) – Only returned if *ret_qpi* is True; Modeled data

Notes

If *method* is "image", then the "edge" method is used as a first step to estimate initial parameters for radius, refractive index, and position of the sphere using *edgekw*. If this behavior is not desired, please make use of the method `qpsphere.imagefit.analyze()`.

`qpsphere.cnvnc.`**`bg_phase_mask_from_sim`**(*sim*, *radial_clearance=1.1*)

Return the background phase mask of a qpsphere simulation

Parameters

- **sim** (`qpimage.QPImage`) – Quantitative phase data simulated with qpsphere; The simulation keyword arguments "sim center", "sim radius", and "pixel size" must be present in *sim.meta*.
- **radial_clearance** (`float`) – Multiplicator to the fitted radius of the sphere; modifies the size of the mask; set to "1" to use the radius determined by `qpsphere.analyze()`. The circular area containing the phase object is set to *False* in the output *mask* image.

Returns **mask** – The mask is *True* for background regions and *False* for object regions.

Return type boolean 2d np.ndarray

`qpsphere.cnvnc.`**`bg_phase_mask_for_qpi`**(*qpi*, *r0*, *method='edge'*, *model='projection'*, *edgekw={}*, *imagekw={}*, *radial_clearance=1.1*)

Determine the background phase mask for a spherical phase object

The position and radius of the phase object are determined with `analyze()`, to which the corresponding keyword arguments are passed. A binary mask is created from the simulation results via `bg_phase_mask_from_sim()`.

Parameters

- **qpi** (`qpimage.QPImage`) – Quantitative phase image data
- **r0** (`float`) – Approximate radius of the sphere [m]
- **method** (`str`) – The method used to determine the refractive index can either be "edge" (determine the radius from the edge detected in the phase image) or "image" (perform a 2D phase image fit).
- **model** (`str`) – The light-scattering model used by *method*. If *method* is "edge", only "projection" is allowed. If *method* is "image", *model* can be one of "mie", "projection", "rytov", or "rytov-sc".
- **edgekw** (`dict`) – Keyword arguments for tuning the edge detection algorithm, see `qpsphere.edgefit.contour_canny()`.
- **imagekw** (`dict`) – Keyword arguments for tuning the image fitting algorithm, see `qpsphere.imagefit.alg.match_phase()`

- **radial_clearance** (*float*) – Multiplicator to the fitted radius of the sphere; modifies the size of the mask; set to "1" to use the radius determined by `qpsphere.analyze()`. The circular area containing the phase object is set to *False* in the output *mask* image.

Returns **mask** – The mask is *True* for background regions and *False* for object regions.

Return type boolean 2d np.ndarray

4.3 edgefit (Contour-based fitting)

Canny edge detection approach for QPI analysis of spheres

exception qpsphere.edgefit.**EdgeDetectionError**

exception qpsphere.edgefit.**EdgeDetectionWarning**

exception qpsphere.edgefit.**RadiusExceedsImageSizeError**

qpsphere.edgefit.**analyze**(*qpi*, *r0*, *edgekw={}*, *ret_center=False*, *ret_edge=False*)

Determine refractive index and radius using Canny edge detection

Compute the refractive index of a spherical phase object by detection of an edge in the phase image, a subsequent circle fit to the edge, and finally a weighted average over the phase image assuming a parabolic phase profile.

Parameters

- **qpi** (*QPImage*) – Quantitative phase image information
- **r0** (*float*) – Approximate radius of the sphere [m]
- **edgekw** (*dict*) – Additional keyword arguments for `contour_canny()`
- **ret_center** (*bool*) – Return the center coordinate of the sphere

Returns

- **n** (*float*) – Computed refractive index
- **r** (*float*) – Computed radius [m]
- **center** (*tuple of floats*) – Center position of the sphere [px], only returned if *ret_center* is *True*

qpsphere.edgefit.**average_sphere**(*image*, *center*, *radius*, *weighted=True*, *ret_crop=False*)

Compute the weighted average phase from a phase image of a sphere

Parameters

- **image** (*2d ndarray*) – Quantitative phase image of a sphere
- **center** (*tuble (x,y)*) – Center of the sphere in *image* in ndarray coordinates
- **radius** (*float*) – Radius of the sphere in pixels
- **weighted** (*bool*) – If *True*, return average phase density weighted with the height profile obtained from the radius, otherwise return simple average phase density. Weighting gives data points at the center of the sphere more weight than those points at the boundary of the sphere, avoiding edge artifacts.
- **ret_crop** (*bool*) – Return the cropped image.

Returns

- **average** (*float*) – The average phase value of the sphere from which the refractive index can be computed

- **cropped_image** (*2d ndarray*) – Returned if *ret_crop* is True

`qpsphere.edgefit.`**`circle_fit`**(*edge*, *ret_dev=False*)

Fit a circle to a boolean edge image

Parameters

- **edge** (*2d boolean ndarray*) – Edge image
- **ret_dev** (*bool*) – Return the average deviation of the distance from contour to center of the fitted circle.

Returns

- **center** (*tuple of (float, float)*) – Coordinates of the circle center
- **radius** (*float*) – Radius of the circle [px]
- *rdev* – Only returned if *ret_dev* is True Average deviation of the radius from the circle

`qpsphere.edgefit.`**`circle_radii`**(*params*, *xedge*, *yedge*)

Compute the distance to the center from cartesian coordinates

This method is used for fitting a circle to a set of contour points.

Parameters

- **params** (*lmfit.Parameters*) – Must contain the keys:
  - "cx": origin of x coordinate [px]
  - "cy": origin of y coordinate [px]
- **xedge** (*1D np.ndarray*) – Edge coordinates x [px]
- **yedge** (*1D np.ndarray*) – Edge coordinates y [px]

Returns **radii** – Radii corresponding to edge coordinates relative to origin

Return type 1D np.ndarray

`qpsphere.edgefit.`**`circle_residual`**(*params*, *xedge*, *yedge*)

Residuals for circle fitting

Parameters

- **params** (*lmfit.Parameters*) – Must contain the keys:
  - "cx": origin of x coordinate [px]
  - "cy": origin of y coordinate [px]
- **xedge** (*1D np.ndarray*) – Edge coordinates x [px]
- **yedge** (*1D np.ndarray*) – Edge coordinates y [px]

Returns **rad_dev** – Deviation of radii from average radius

Return type 1D np.ndarray

`qpsphere.edgefit.`**`contour_canny`**(*image*, *radius*, *mult_coarse=0.4*, *mult_fine=0.1*, *clip_rmin=0.9*, *clip_rmax=1.1*, *maxiter=20*, *verbose=True*)

Heuristic Canny edge detection for circular objects

Two Canny-based edge detections with different filter sizes are performed to find the outmost contour of an object in a phase image while keeping artifacts at a minimum.

Parameters

- **image** (*2d ndarray*) – Image containing an approximately spherically symmetric object

- **radius** (*float*) – The approximate object radius in pixels (required for filtering)
- **mult_coarse** (*float*) – The coarse edge detection has a filter size of `sigma = mult_coarse * radius`
- **mult_fine** (*float*) – The fine edge detection has a filter size of `sigma = mult_fine * radius`
- **clip_rmin** (*float*) – Removes edge points that are closer than *clip_rmin* times the average radial edge position from the center of the image.
- **clip_rmax** (*float*) – Removes edge points that are further than *clip_rmin* times the average radial edge position from the center of the image.
- **maxiter** (*int*) – Maximum number iterations for coarse edge detection, see Notes
- **verbose** (*bool*) – If set to *True*, issues EdgeDetectionWarning where applicable

Returns **edge** – The detected edge positions of the object.

Return type 2d boolean ndarray

Notes

If no edge is found using the filter size defined by *mult_coarse*, then the coarse filter size is reduced by a factor of 2 until an edge is found or until *maxiter* is reached.

The edge found using the filter size defined by *mult_fine* is heuristically filtered (parts at the center and at the edge of the image are removed). This heuristic filtering assumes that the circular object is centered in the image.

See also:

skimage.feature.canny() Canny edge detection algorithm used

4.4 imagefit (2D phase image fitting)

`qpsphere.imagefit.`**analyze**(*qpi*, *model*, *n0*, *r0*, *c0=None*, *imagekw={}*, *ret_center=False*, *ret_pha_offset=False*, *ret_qpi=False*)

Fit refractive index and radius to a phase image of a sphere

Parameters

- **qpi** (*QPImage*) – Quantitative phase image information
- **model** (*str*) – Name of the light-scattering model (see *qpsphere.models.available*)
- **n0** (*float*) – Approximate refractive index of the sphere
- **r0** (*float*) – Approximate radius of the sphere [m]
- **c0** (*tuple of (float, float)*) – Approximate center position in ndarray index coordinates [px]; if set to *None* (default), the center of the image is used.
- **imagekw** (*dict*) – Additional keyword arguments to *qpsphere.imagefit.alg.match_phase()*.
- **ret_center** (*bool*) – Return the center coordinate of the sphere
- **ret_pha_offset** (*bool*) – If True, return the phase image background offset.

- **ret_qpi** (*bool*) – If True, return the modeled data as a `qpimage.QPImage`.

Returns

- **n** (*float*) – Computed refractive index
- **r** (*float*) – Computed radius [m]
- **c** (*tuple of floats*) – Only returned if *ret_center* is True Center position of the sphere [px]
- **pha_offset** (*float*) – Only returned if *ret_pha_offset* is True Phase image background offset
- **qpi_sim** (*qpimage.QPImage*) – Only returned if *ret_qpi* is True Modeled data

4.4.1 imagefit.alg (Algorithms)

`qpsphere.imagefit.alg.`**`match_phase`**(*qpi, model, n0, r0, c0=None, pha_offset=0, fix_pha_offset=False, nrel=0.1, rrel=0.05, crel=0.05, stop_dn=0.0005, stop_dr=0.001, stop_dc=1, min_iter=3, max_iter=100, ret_center=False, ret_pha_offset=False, ret_qpi=False, ret_num_iter=False, ret_interim=False, verbose=0, verbose_h5path='./match_phase_error.h5'*)

Fit a scattering model to a quantitative phase image

Parameters

- **qpi** (*qpimage.QPImage*) – QPI data to fit (e.g. experimental data)
- **model** (*str*) – Name of the light-scattering model (see *qpsphere.models.available*)
- **n0** (*float*) – Initial refractive index of the sphere
- **r0** (*float*) – Initial radius of the sphere [m]
- **c0** (*tuple of (float, float)*) – Initial center position of the sphere in ndarray index coordinates [px]; if set to *None* (default), the center of the image is used.
- **pha_offset** (*float*) – Initial phase offset [rad]
- **fix_pha_offset** (*bool*) – If True, do not fit the phase offset *pha_offset*. The phase offset is determined from the mean of all pixels whose absolute phase is
  - below 1% of the modeled phase and
  - within a 5px or 20% border (depending on which is larger) around the phase image.
- **nrel** (*float*) – Determines the border of the interpolation range for the refractive index: [n-(n-nmed)*nrel, n+(n-nmed)*nrel] with nmed=qpi["medium_index"] and, initially, n=n0.
- **rrel** (*float*) – Determines the border of the interpolation range for the radius: [r*(1-rrel), r*(1+rrel)] with, initially, r=r0.
- **crel** (*float*) – Determines the border of the interpolation range for the center position: [cxy - dc, cxy + dc] with the center position (along x or y) cxy, and the interval radius dc defined by dc=max(lambda, crel * r0) with the vacuum wavelength lambda=qpi["wavelenght"].
- **stop_dn** (*float*) – Stopping criterion for refractive index
- **stop_dr** (*float*) – Stopping criterion for radius
- **stop_dc** (*float*) – Stopping criterion for lateral offsets
- **min_iter** (*int*) – Minimum number of fitting iterations to perform

- **max_iter** (*int*) – Maximum number of fitting iterations to perform
- **ret_center** (*bool*) – If True, return the fitted center coordinates
- **ret_pha_offset** (*bool*) – If True, return the fitted phase offset
- **ret_qpi** (*bool*) – If True, return the final fit as a data set
- **ret_num_iter** (*bool*) – If True, return the number of iterations
- **ret_interim** (*bool*) – If True, return intermediate parameters of each iteration
- **verbose** (*int*) – Higher values increase verbosity
- **verbose_h5path** (*str*) – Path to hdf5 output file, created when *verbosity* >= *2*

Returns

- **n** (*float*) – Fitted refractive index
- **r** (*float*) – Fitted radius [m]
- **c** (*tuple of (float, float)*) – Only returned if *ret_center* is True Center position of the sphere in ndarray index coordinates [px]
- **pha_offset** (*float*) – Only returned if *ret_pha_offset* is True Fitted phase offset [rad]
- **qpi** (*qpimage.QPImage*) – Only returned if *ret_qpi* is True Simulation using *model* with the final fit parameters
- **num_iter** (*int*) – Only returned if *ret_num_iter* is True Number of iterations performed; negative number is returned when iteration fails
- **interim** (*list*) – Only returned if *ret_interim* is True Intermediate fitting parameters

`qpsphere.imagefit.alg.`**`sq_phase_diff`**(*pha_a*, *pha_b*)

Compute sum of squares error between two arrays

Parameters **pha_b** (*pha_a,*) – Phase data to compare

Returns **sumsq** – Sum of squares of differences

Return type float

`qpsphere.imagefit.alg.`**`export_phase_error_hdf5`**(*h5path*, *identifier*, *index*, *phase*, *mphase*, *model*, *n0*, *r0*, *spi_params*)

Export the phase error to an hdf5 file

Parameters

- **h5path** (*str or pathlib.Path*) – path to hdf5 output file
- **identifier** (*str*) – unique identifier of the input phase (e.g. *qpimage.QPImage["identifier"]*)
- **index** (*int*) – iteration index
- **phase** (*2d real-valued np.ndarray*) – phase image
- **mphase** (*2d real-valued np.ndarray*) – reference phase image
- **model** (*str*) – sphere model name
- **n0** (*float*) – initial object index
- **r0** (*float*) – initial object radius [m]
- **spi_params** (*dict*) – parameter dictionary of `SpherePhaseInterpolator()`

4.4.2 imagefit.interp (Image interpolation logic)

class qpsphere.imagefit.interp.**SpherePhaseInterpolator**(*model*, *model_kwargs*, *pha_offset=0*, *nrel=0.1*, *rrel=0.05*, *verbose=0*)

Interpolation in-between modeled phase images

Parameters

- **model** (*str*) – Name of the light-scattering model (see *qpsphere.models.available*)
- **model_kwargs** (*dict*) – Keyword arguments for the sphere model; must contain:
  - **radius: float** Radius of the sphere [m]
  - **sphere_index: float** Refractive index of the object
  - **medium_index: float** Refractive index of the surrounding medium
  - **wavelength: float** Vacuum wavelength of the imaging light [m]
  - **pixel_size: float** Pixel size [m]
  - **grid_size: tuple of floats** Resulting image size in x and y [px]
  - **center: tuple of floats** Center position in image coordinates [px]
- **pha_offset** (*float*) – Phase offset added to the interpolation result
- **nrel** (*float*) – Determines the border of the interpolation range for the refractive index: [n-(n-nmed)*nrel, n+(n-nmed)*nrel] with n=model_kwargs["sphere_index"] and nmed=model_kwargs["medium_index"]
- **rrel** (*float*) – Determines the border of the interpolation range for the radius: [r*(1-rrel), r*(1+rrel)] with r=model_kwargs["radius"]
- **verbose** (*int*) – Increases verbosity.

model = None
scattering model

sphere_method = None
scattering model function

model_kwargs = None
scattering model keyword arguments

radius = None
current sphere radius [m]

sphere_index = None
current sphere index

pha_offset = None
current background phase offset

posx_offset = None
current pixel offset in x

posy_offset = None
current pixel offset in y

dn = None
half of current search interval size for refractive index

dr = None
half of current search interval size for radius [m]

params
Current interpolation parameter dictionary

range_n
Current interpolation range of refractive index

range_r
Current interpolation range of radius

compute_qpi()
Compute model data with current parameters

Returns **qpi** – Modeled phase data

Return type qpimage.QPImage

Notes

The model image might deviate from the fitted image because of interpolation during the fitting process.

get_border_phase(*idn=0*, *idr=0*)
Return one of nine border fields

Parameters

- **idn** (*int*) – Index for refractive index. One of -1 (left), 0 (center), 1 (right)
- **idr** (*int*) – Index for radius. One of -1 (left), 0 (center), 1 (right)

get_phase(*nintp=None*, *rintp=None*, *delta_offset_x=0*, *delta_offset_y=0*)
Interpolate from the border fields to new coordinates

Parameters

- **nintp** (*float or None*) – Refractive index of the sphere
- **rintp** (*float or None*) – Radius of sphere [m]
- **delta_offset_x** (*float*) – Offset in x-direction [px]
- **delta_offset_y** (*float*) – Offset in y-direction [px]

Returns **phase_intp** – Interpolated phase at the given parameters

Return type 2D real-valued np.ndarray

Notes

Not all combinations are poosible, e.g.

- One of nintp or rintp must be None
- The current interpolation range must include the values for rintp and nintp

4.5 models (Scattering models)

`qpsphere.models.available = ['mie', 'mie-avg', 'projection', 'rytov', 'rytov-sc']`
available light-scattering models

`qpsphere.models.simulate(radius=5e-06, sphere_index=1.339, medium_index=1.333, wavelength=5.5e-07, grid_size=(80, 80), model='projection', pixel_size=None, center=None)`
Simulate scattering at a sphere

Parameters

- **radius** (*float*) – Radius of the sphere [m]
- **sphere_index** (*float*) – Refractive index of the object
- **medium_index** (*float*) – Refractive index of the surrounding medium
- **wavelength** (*float*) – Vacuum wavelength of the imaging light [m]
- **grid_size** (*tuple of ints or int*) – Resulting image size in x and y [px]
- **model** (*str*) – Sphere model to use (see *available*)
- **pixel_size** (*float or None*) – Pixel size [m]; if set to *None* the pixel size is chosen such that the radius fits at least three to four times into the grid.
- **center** (*tuple of floats or None*) – Center position in image coordinates [px]; if set to None, the center of the image (grid_size - 1)/2 is used.

Returns **qpi** – Quantitative phase data set

Return type qpimage.QPImage

4.6 util (Helper methods)

`qpsphere.util.CACHE_PATH = PosixPath('/home/docs/.cache/python-qpsphere')`
User's cache directory

`qpsphere.util.RESCR_PATH = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/qps`
Qpsphere package *resources* directory.

`qpsphere.util.download_binaries(package_dir=False)`
Download all binaries for the current platform

Parameters **package_dir** (*bool*) – If set to *True*, the binaries will be downloaded to the *resources* directory of the qpsphere package instead of to the users application data directory. Note that this might require administrative rights if qpsphere is installed in a system directory.

Returns **paths** – List of paths to binaries. This will always return binaries in the *resources* directory of the qpsphere package (if binaries are present there), in disregard of the parameter *package_dir*.

Return type list of pathlib.Path

`qpsphere.util.remove_binaries(package_dir=False)`
Remove all binaries for the current platform

Parameters **package_dir** (*bool*) – If True, remove all binaries from the *resources* directory of the qpsphere package. If False, remove all binaries from the user's cache directory.

CHAPTER 5

Bilbliography

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Bibliography

[MSG+18] P. Müller, M. Schürmann, S. Girardo, G. Cojoc, and Guck J. Accurate evaluation of size and refractive index for spherical objects in quantitative phase imaging. *Optics Express*, 26(8):10729–10743, 2018. doi:10.1364/OE.26.010729.

[SSM+15] M. Schürmann, J. Scholze, P. Müller, C. J. Chan, A. E. Ekpenyong, K. J. Chalut, and J. Guck. Chapter 9 - Refractive index measurements of single, spherical cells using digital holographic microscopy. In Ewa K Paluch, editor, *Biophysical Methods in Cell Biology*, volume 125 of Methods in Cell Biology, pages 143–159. Academic Press, 2015. doi:10.1016/bs.mcb.2014.10.016.

[SSM+16] M. Schürmann, J. Scholze, P. Müller, J. Guck, and C. J. Chan. Cell nuclei have lower refractive index and mass density than cytoplasm. *Journal of Biophotonics*, 9(10):1068–1076, oct 2016. doi:10.1002/jbio.201500273.

Python Module Index

q

qpsphere.cnvnc, 15
qpsphere.edgefit, 17
qpsphere.imagefit, 19
qpsphere.imagefit.alg, 20
qpsphere.imagefit.interp, 22
qpsphere.models, 24
qpsphere.util, 24

Index

A

analyze() (in module qpsphere.cnvnc), 15
analyze() (in module qpsphere.edgefit), 17
analyze() (in module qpsphere.imagefit), 19
available (in module qpsphere.models), 24
average_sphere() (in module qpsphere.edgefit), 17

B

bg_phase_mask_for_qpi() (in module qpsphere.cnvnc), 16
bg_phase_mask_from_sim() (in module qpsphere.cnvnc), 16

C

CACHE_PATH (in module qpsphere.util), 24
circle_fit() (in module qpsphere.edgefit), 18
circle_radii() (in module qpsphere.edgefit), 18
circle_residual() (in module qpsphere.edgefit), 18
compute_qpi() (qpsphere.imagefit.interp.SpherePhaseInterpolator method), 23
contour_canny() (in module qpsphere.edgefit), 18

D

dn (qpsphere.imagefit.interp.SpherePhaseInterpolator attribute), 22
download_binaries() (in module qpsphere.util), 24
dr (qpsphere.imagefit.interp.SpherePhaseInterpolator attribute), 22

E

EdgeDetectionError, 17
EdgeDetectionWarning, 17
export_phase_error_hdf5() (in module qpsphere.imagefit.alg), 21

G

get_border_phase() (qpsphere.imagefit.interp.SpherePhaseInterpolator method), 23
get_phase() (qpsphere.imagefit.interp.SpherePhaseInterpolator method), 23

M

match_phase() (in module qpsphere.imagefit.alg), 20
model (qpsphere.imagefit.interp.SpherePhaseInterpolator attribute), 22
model_kwargs (qpsphere.imagefit.interp.SpherePhaseInterpolator attribute), 22

P

params (qpsphere.imagefit.interp.SpherePhaseInterpolator attribute), 23
pha_offset (qpsphere.imagefit.interp.SpherePhaseInterpolator attribute), 22
posx_offset (qpsphere.imagefit.interp.SpherePhaseInterpolator attribute), 22
posy_offset (qpsphere.imagefit.interp.SpherePhaseInterpolator attribute), 22

Q

qpsphere.analyze() (built-in function), 15
qpsphere.cnvnc (module), 15
qpsphere.edgefit (module), 17
qpsphere.imagefit (module), 19
qpsphere.imagefit.alg (module), 20
qpsphere.imagefit.interp (module), 22
qpsphere.models (module), 24
qpsphere.util (module), 24

R

radius (qpsphere.imagefit.interp.SpherePhaseInterpolator attribute), 22
RadiusExceedsImageSizeError, 17
range_n (qpsphere.imagefit.interp.SpherePhaseInterpolator attribute), 23
range_r (qpsphere.imagefit.interp.SpherePhaseInterpolator attribute), 23
remove_binaries() (in module qpsphere.util), 24

RESCR_PATH (in module qpsphere.util), 24

S

simulate() (in module qpsphere.models), 24
sphere_index (qpsphere.imagefit.interp.SpherePhaseInterpolator attribute), 22
sphere_method (qpsphere.imagefit.interp.SpherePhaseInterpolator attribute), 22
SpherePhaseInterpolator (class in qpsphere.imagefit.interp), 22
sq_phase_diff() (in module qpsphere.imagefit.alg), 21