qpimage Documentation

Release 0.2.0

Paul Müller

Jun 18, 2018

Contents:

1 Introduction **3**
- 1.1 The Problem . . . 3
- 1.2 Why qpimage? . . . 3
- 1.3 What are the alternatives? . . . 3
- 1.4 Citing qpimage . . . 4

2 Getting started **5**
- 2.1 Installing qpimage . . . 5
- 2.2 User API . . . 5
  - 2.2.1 Basic usage . . . 6
  - 2.2.2 Storing QPImage data on disk . . . 6
  - 2.2.3 Dealing with measurement series . . . 7
  - 2.2.4 Notes . . . 7

3 Examples **9**
- 3.1 Simple phase . . . 9
- 3.2 Background image tilt correction . . . 10
- 3.3 Background image offset correction . . . 11
- 3.4 Background image binary mask correction . . . 13
- 3.5 Background image 2nd order polynomial correction . . . 15
- 3.6 Digital hologram of a single cell . . . 17
- 3.7 Hologram filter choice . . . 19

4 Code reference **23**
- 4.1 bg_estimate (background-estimation) . . . 23
  - 4.1.1 Constants . . . 23
  - 4.1.2 Methods . . . 23
- 4.2 core (QPImage) . . . 24
  - 4.2.1 Constants . . . 24
  - 4.2.2 Classes . . . 24
  - 4.2.3 Methods . . . 28
- 4.3 holo (hologram analysis) . . . 28
  - 4.3.1 Methods . . . 28
- 4.4 image_data (basic image management) . . . 30
  - 4.4.1 Constants . . . 30
  - 4.4.2 Classes . . . 30
  - 4.4.3 Methods . . . 32

4.5 integrity_check (check QPImage data) . . . 33
4.5.1 Exceptions . . . 33
4.5.2 Methods . . . 33
4.6 meta (definitions for QPImage meta data) . . . 33
4.6.1 Constants . . . 33
4.6.2 Exceptions . . . 33
4.6.3 Classes . . . 33
4.7 series (QPSeries) . . . 34
4.7.1 Classes . . . 34

5 Bilbliography **35**

6 Indices and tables **37**

Bibliography **39**

Python Module Index **41**

Qpimage is a Python3 library that provides a convenient interface to many common functionalities that are used in quantitative phase imaging. This is the documentation of qpimage version 0.2.0.

CHAPTER 1

Introduction

1.1 The Problem

Quantitative phase imaging (QPI) is a fundamental imaging technique that visualizes the retardation of electromagnetic radiation as it passes through an object. The parameter that governs this retardation is called refractive index. In biological imaging, QPI is an important tool to measure the dry mass or the refractive index (related to mass density *[Bar52] [DW52]*) of single cells and tissues, which enables a profound characterization of the investigated samples.

1.2 Why qpimage?

In the Guck group, we make heavy use of QPI and thus require a reliable and well-documented software library that, independent of the particular QPI setup used, allows us to address QPI-related research questions. Qpimage attempts to unify QPI analysis by providing a unique and user-friendly API for working with QPI data, including the choice of input data (complex field, phase with amplitude or intensity, hologram), memory-efficient and fast storage of large data sets (using HDF5, phase and amplitude data are stored separately), or robust and extendable background correction techniques (tilt and second order polynomial fits, binary mask). The main reason for the development of qpimage is our QPI analysis software DryMass.

1.3 What are the alternatives?

There are other open-source Python libraries that address quantitative phase imaging analysis with varying scopes and motivations.

- HoloPy is an established Python library for digital holographic microscopy (DHM) that comes with several additional features such as scattering calculations and model fitting. The overlap between HoloPy and qpimage is the computation of phase and amplitude from raw hologram data. The main difference is that HoloPy is focused on DHM analysis with a rich set of tools while qpimage is only focused on managing quantitative phase data (data conversion and storage as well as an extended set of background correction algorithms). However, there is a broad set of additional tools in the "qpimage universe", including qpformat for loading experimental

data, qpsphere for scattering calculations and model fitting (focus is on cell-sized objects), and DryMass as a user interface to these libraries.

- The Python package shampoo focuses on DHM reconstruction and detection and tracking of biological cells. The overlap between shampoo and the "qpimage universe" (see above) is quite large. The difference is mostly the scope of the projects; While shampoo is an optimized library for DHM analysis, the "qpimage universe" encompasses other quantitative phase imaging (QPI) techniques with the aim to becoming a generic tool in QPI analysis. Experimental .tif files from the shampoo project can be opened with qpformat (see Hologram from tif file).
- If you are using electron holography, HyperSpy might be worth looking at. If you are storing your hologram data in the HyperSpy file format, you can still load it with qpformat (see HyperSpy hologram file format) and analyze it with qpimage.

1.4 Citing qpimage

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

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

or in a bibliography

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

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

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

- Phase retrieval from holographic images with a gaussian filter is implemented according to *[SSM+15]*.
- Phase background image correction with a tilt fitted to a border of the image data was used in *[SSM+15]* and *[SSM+16]*.
- Phase background image correction with a polynomial fitted to known background regions was introduced for DHM in *[CCC+06]* (in this reference the phase correction is applied to the hologram data before field reconstruction).
- Intensity background correction by dividing by a reference intensity image for tomographic imaging was used in *[SCG+17]*.

CHAPTER 2

Getting started

2.1 Installing qpimage

Qpimage is written in pure Python and supports Python version 3.5 and higher. Qpimage depends on several other scientific Python packages, including:

- numpy,
- scipy,
- h5py (caching),
- lmfit (background estimation),
- nrefocus (numerical focusing), and
- scikit-image (phase unwrapping using `skimage.restoration.unwrap_phase()`).

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

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

2.2 User API

The qpimage API is built upon the hdf5 file format using h5py. That means that each instance of *`qpimage.QPImage`* generates an hdf5 file, either on disk or in memory, depending on the preferences of the user. This approach has the advantage that phase and amplitude data can be cached on disk, including all parameters that were used for background correction, which allows to transparently recapture any steps that were performed on a specific data set at a later time point.

2.2.1 Basic usage

A typical use case of qpimage is

```
qpi = qpimage.QPImage(data=phase_ndarray, which_data="phase")
# perform phase-tilt background correction
qpi.compute_bg(which_data="phase",  # correct phase image
               fit_offset="fit",  # use bg offset from tilt fit
               fit_profile="tilt",  # perform 2D tilt fit
               border_px=5,  # use 5 px border around image
                                 )
# save the background-corrected phase to a text file
numpy.savetxt("out.txt", qpi.pha)
```

which creates an instance of *QPImage* containing otherwise experimentally obtained phase data, performs a phase-tilt background correction and then saves the corrected phase data to the text file "out.txt". In this case, all data are stored in memory.

2.2.2 Storing QPImage data on disk

To cache the QPImage data on disk, use the `with` statement in combination with the `h5file` keyword argument

```
with qpimage.QPImage(data=phase_ndarray, which_data="phase", h5file="/path/to/file.h5
↪") as qpi:
    qpi.compute_bg(which_data="phase",
                   fit_offset="fit",
                   fit_profile="tilt",
                   border_px=5,
                   )
```

where all data is stored in `/path/to/file.h5`. This will create an hdf5 file on disk that, at a later time point, can be used to create an instance of *QPImage*:

```
# open previously cached data for reading
qpi = qpimage.QPImage(h5file="/path/to/file.h5", h5mode="r")

# or open cached data for writing (e.g. for changing the background)
with qpimage.QPImage(h5file="/path/to/file.h5", h5mode="a") as qpi:
    # do something here
```

The default value of `h5mode` is "a", which means that data will be overridden. In the hdf5 file, the following data is stored:

- all data for reproducing the background-corrected phase (`qpi.pha`) and amplitude (`qpi.amp`) (and thus field `qpi.field`), including
  - the experimental phase data
  - the experimental background data
  - the parameters for reproducing the result of *qpi.compute_bg*
- all measurement specific meta data, given by the keyword argument `meta_data`

2.2.3 Dealing with measurement series

Qpimage also comes with a *QPSeries* class for handling multiple instances of QPImage in one hdf5 file. For instance, to combine two QPImages in one series file, one could use:

```
paths = ["file_a.h5", "file_b.h5", "file_c.h5"]

with qpimage.QPSeries(h5file="/path/to/series_file.h5", h5mode="w") as qps:
    for ii, pp in enumerate(paths):
        qpi = qpimage.QPImage(h5file="/path/to/file.h5", h5mode="r")
        qps.add_qpimage(qpi=qpi, identifier="my_name_{}".frmat(ii))
```

Note that the function *add_qpimage* accepts the optional keyword argument "identifier" (overriding the identifier of the QPImage) which can also be used for indexing later:

```
with qpimage.QPSeries(h5file="/path/to/series_file.h5", h5mode="r") as qps:
    # these two are equivalent
    qpi = qps[0]
    qpi = qps["my_name_0"]
```

2.2.4 Notes

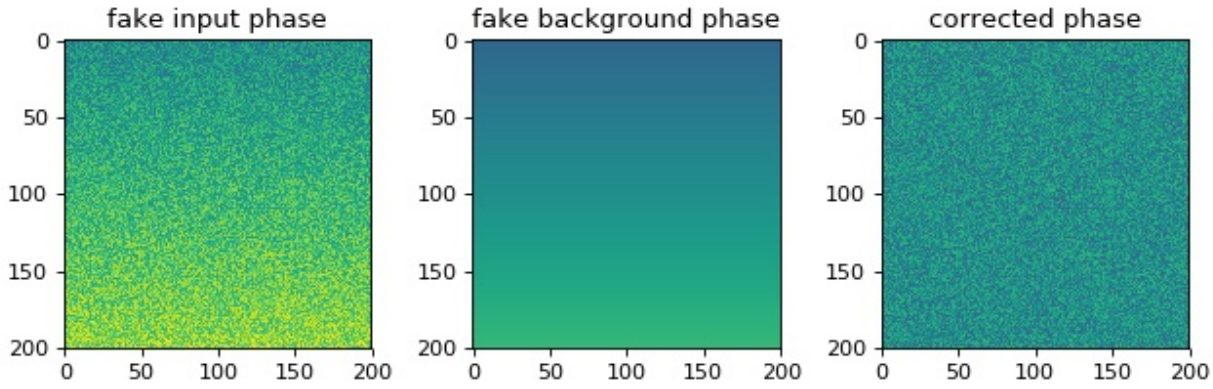
- Even though the hdf5 data is stored as gzip-compressed single precision floating point values, using qpimage hdf5 files may result in file sizes that are considerably larger compared to when only the output of e.g. `qpi.pha` is stored using e.g. `numpy.save()`.
- Units in qpimage follow the international system of units (SI).
- *qpimage.QPSeries* provides a convenient way to manage multiple *qpimage.QPImage*, optionally storing them in a single hdf5 file.

CHAPTER 3

Examples

3.1 Simple phase

This example illustrates the simple usage of the *qpimage.QPImage* class for reading and managing quantitative phase data. The attribute `QPImage.pha` yields the background-corrected phase data and the attribute `QPImage.bg_pha` yields the background phase image.



simple_phase.py

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

size = 200
# background phase image with a tilt
bg = np.repeat(np.linspace(0, 1, size), size).reshape(size, size)
# phase image with random noise
```

(continues on next page)

(continued from previous page)

```
phase = np.random.rand(size, size) + bg

# create QPImage instance
qpi = qpimage.QPImage(data=phase, bg_data=bg, which_data="phase")

# plot the properties of `qpi`
plt.figure(figsize=(8, 3))
plot_kw = {"vmin": -1,
           "vmax": 2}

plt.subplot(131, title="fake input phase")
plt.imshow(phase, **plot_kw)

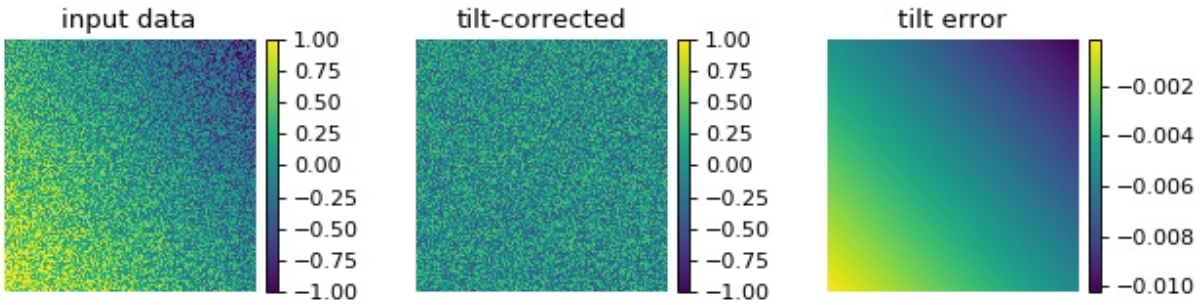
plt.subplot(132, title="fake background phase")
plt.imshow(qpi.bg_pha, **plot_kw)

plt.subplot(133, title="corrected phase")
plt.imshow(qpi.pha, **plot_kw)

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

3.2 Background image tilt correction

This example illustrates background tilt correction with qpimage. In contrast to the 'simple_phase.py' example, the known background data is not given to the *qpimage.QPImage* class. In this particular example, the background tilt correction achieves an error of about 1% which is sufficient in most quantitative phase imaging applications.



background_tilt.py

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

size = 200
# background phase image with a tilt
bg = np.repeat(np.linspace(0, 1, size), size).reshape(size, size)
bg = .6 * bg - .8 * bg.transpose() + .2
# phase image with random noise
```

(continues on next page)

(continued from previous page)

```
rsobj = np.random.RandomState(47)
phase = rsobj.rand(size, size) - .5 + bg

# create QPImage instance
qpi = qpimage.QPImage(data=phase, which_data="phase")
# compute background with 2d tilt approach
qpi.compute_bg(which_data="phase",  # correct phase image
               fit_offset="fit",  # use bg offset from tilt fit
               fit_profile="tilt",  # perform 2D tilt fit
               border_px=5,  # use 5 px border around image
               )

# plot the properties of `qpi`
fig = plt.figure(figsize=(8, 2.5))
plot_kw = {"vmin": -1,
           "vmax": 1}

ax1 = plt.subplot(131, title="input data")
map1 = ax1.imshow(phase, **plot_kw)
plt.colorbar(map1, ax=ax1, fraction=.046, pad=0.04)

ax2 = plt.subplot(132, title="tilt-corrected")
map2 = ax2.imshow(qpi.pha, **plot_kw)
plt.colorbar(map2, ax=ax2, fraction=.046, pad=0.04)

ax3 = plt.subplot(133, title="tilt error")
map3 = ax3.imshow(bg - qpi.bg_pha)
plt.colorbar(map3, ax=ax3, fraction=.046, pad=0.04)

# disable axes
[ax.axis("off") for ax in [ax1, ax2, ax3]]

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

3.3 Background image offset correction

This example illustrates the different background offset correction methods implemented in qpimage. The phase image data contains two gaussian noise distributions for which these methods yield different background phase offsets.

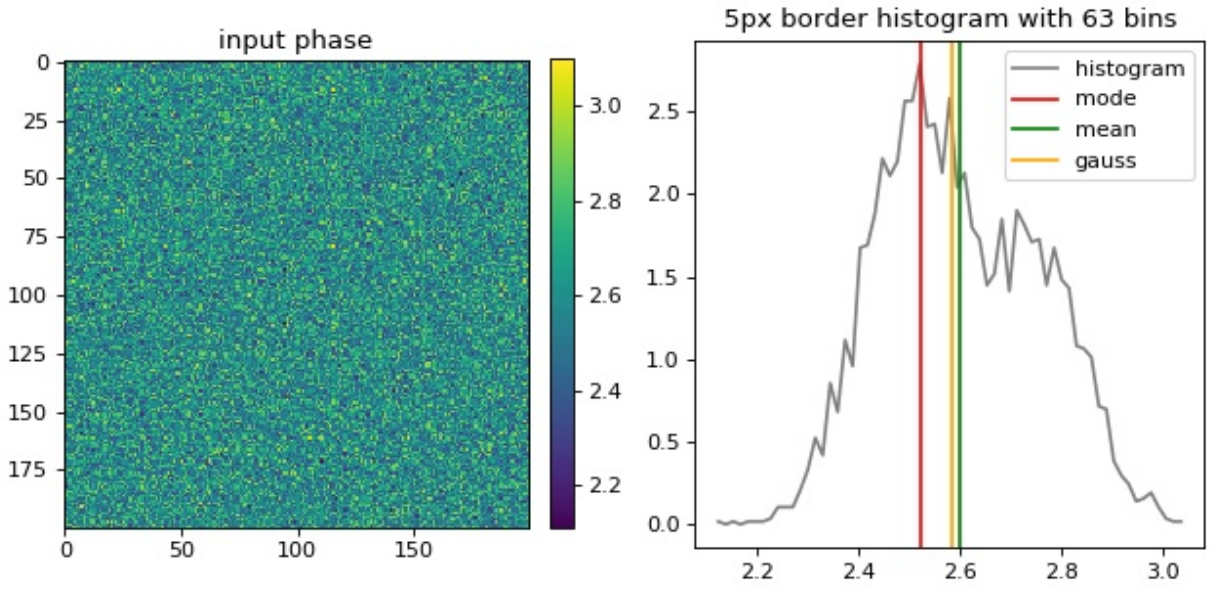
background_offset.py

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

size = 200  # the size of the image
bg = 2.5  # the center of the background phase distribution
scale = .1  # the spread of the background phase distribution

# compute random phase data
rsobj = np.random.RandomState(42)
data = rsobj.normal(loc=bg, scale=scale, size=size**2)
# Add a second distribution `data2` at random positions `idx`,
```

(continues on next page)



(continued from previous page)

```
# such that there is no pure gaussian distribution.
# (otherwise 'mean' and 'gaussian' cannot be distinguished)
data2 = rsobj.normal(loc=bg*1.1, scale=scale, size=size**2//2)
idx = rsobj.choice(data.size, data.size//2)
data[idx] = data2
# reshape `data` to get a 2D array
data = data.reshape(size, size)

qpi = qpimage.QPImage(data=data, which_data="phase")

cpkw = {"which_data": "phase",  # correct the input phase data
        "fit_profile": "offset",  # perform offset correction only
        "border_px": 5,  # use a border of 5px of the input phase
        "ret_binary": True,  # return the binary image for visualization
        }

binary = qpi.compute_bg(fit_offset="mode", **cpkw)
bg_mode = np.mean(qpi.bg_pha[binary])

qpi.compute_bg(fit_offset="mean", **cpkw)
bg_mean = np.mean(qpi.bg_pha[binary])

qpi.compute_bg(fit_offset="gauss", **cpkw)
bg_gauss = np.mean(qpi.bg_pha[binary])

bg_data = (qpi.pha + qpi.bg_pha)[binary]
# compute histogram
nbins = int(np.ceil(np.sqrt(bg_data.size)))
mind, maxd = bg_data.min(), bg_data.max()
histo = np.histogram(bg_data, nbins, density=True, range=(mind, maxd))
dx = abs(histo[1][1] - histo[1][2]) / 2
hx = histo[1][1:] - dx
hy = histo[0]
```

(continues on next page)

(continued from previous page)

```

# plot the properties of `qpi`
plt.figure(figsize=(8, 4))

ax1 = plt.subplot(121, title="input phase")
map1 = plt.imshow(data)
plt.colorbar(map1, ax=ax1, fraction=.046, pad=0.04)


t2 = "{}px border histogram with {} bins".format(cpkw["border_px"], nbins)
plt.subplot(122, title=t2)
plt.plot(hx, hy, label="histogram", color="gray")
plt.axvline(bg_mode, 0, 1, label="mode", color="red")
plt.axvline(bg_mean, 0, 1, label="mean", color="green")
plt.axvline(bg_gauss, 0, 1, label="gauss", color="orange")
plt.legend()

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

3.4 Background image binary mask correction

This example illustrates background correction with qpimage using a binary mask to exclude regions that do not contain background information.

The phase image of a microgel bead (top left) has two artifacts; there is a tilt-like phase profile added along the vertical axis and there is a second microgel bead in close proximity to the center bead. A regular phase tilt background correction using the image values around a frame of five pixels (see "background_tilt.py" example) does not yield a flat background, because the second bead is fitted into the background which leads to a horizontal background phase profile (top right). By defining a binary mask (bottom left image), the phase values of the second bead can be excluded from the background tilt fit and a flat background phase is achieved (bottom right).

`background_binary_mask.py`

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

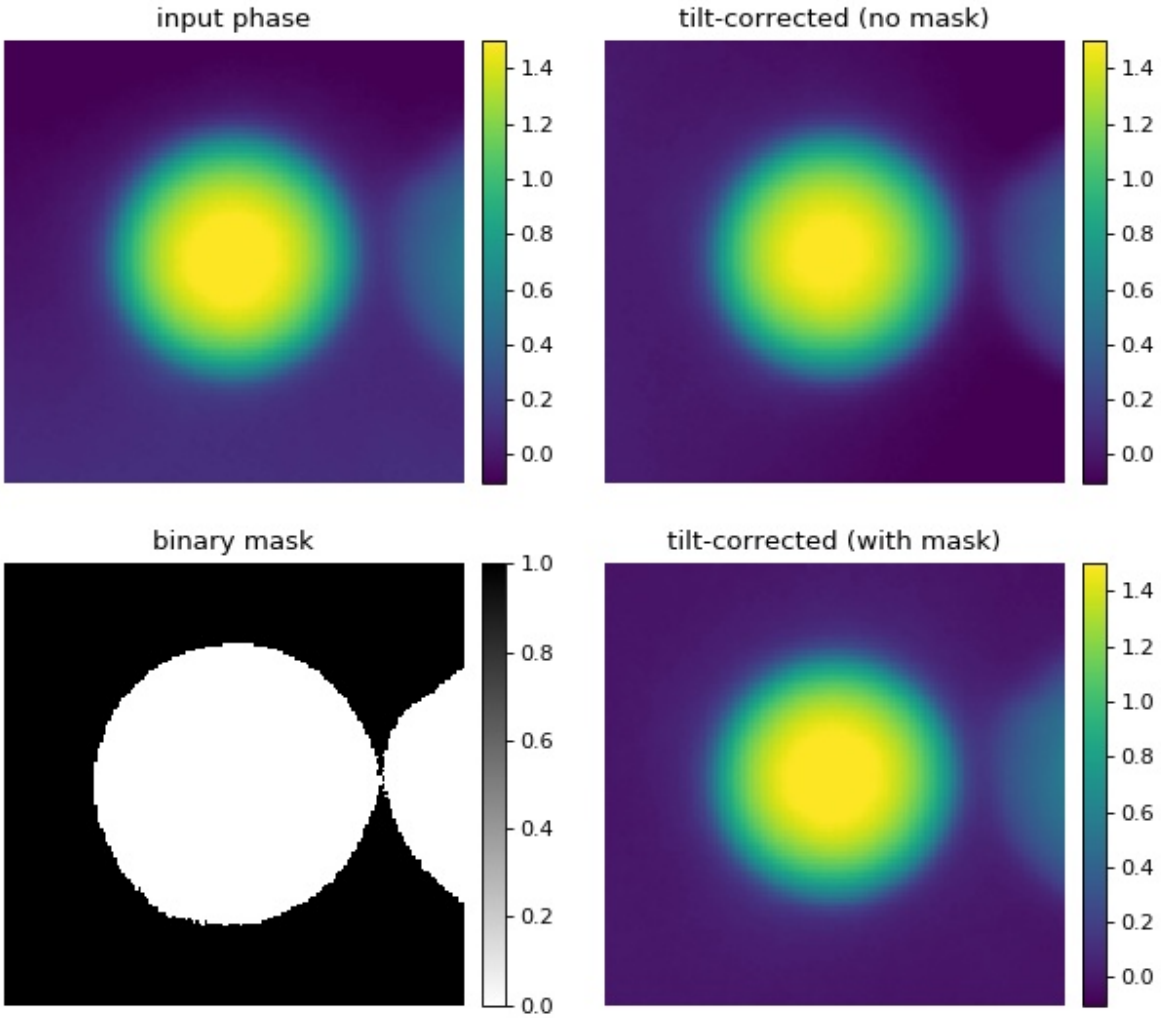

# load the experimental data
input_phase = np.load("./data/phase_beads_close.npz")["phase"].astype(float)

# create QPImage instance
qpi = qpimage.QPImage(data=input_phase,
                      which_data="phase")

# background correction without mask
qpi.compute_bg(which_data="phase",
               fit_offset="fit",
               fit_profile="tilt",
               border_px=5,
               )
pha_nomask = qpi.pha

```

(continues on next page)


input phase
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
tilt-corrected (no mask)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
binary mask
1.0
0.8
0.6
0.4
0.2
0.0
tilt-corrected (with mask)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0

(continued from previous page)

```
# educated guess for binary mask
mask = input_phase < input_phase.max() / 10

# background correction with mask
# (the intersection of `mask` and the 5px border is used for fitting)
qpi.compute_bg(which_data="phase",
               fit_offset="fit",
               fit_profile="tilt",
               border_px=5,
               from_binary=mask
               )
pha_mask = qpi.pha

# plot
fig = plt.figure(figsize=(8, 7))
plot_kw = {"vmin": -.1,
           "vmax": 1.5}

ax1 = plt.subplot(221, title="input phase")
map1 = ax1.imshow(input_phase, **plot_kw)
plt.colorbar(map1, ax=ax1, fraction=.044, pad=0.04)

ax2 = plt.subplot(222, title="tilt-corrected (no mask)")
map2 = ax2.imshow(pha_nomask, **plot_kw)
plt.colorbar(map2, ax=ax2, fraction=.044, pad=0.04)

ax3 = plt.subplot(223, title="binary mask")
map3 = ax3.imshow(mask, cmap="gray_r")
plt.colorbar(map3, ax=ax3, fraction=.044, pad=0.04)

ax4 = plt.subplot(224, title="tilt-corrected (with mask)")
map4 = ax4.imshow(pha_mask, **plot_kw)
plt.colorbar(map4, ax=ax4, fraction=.044, pad=0.04)

# disable axes
[ax.axis("off") for ax in [ax1, ax2, ax3, ax3, ax4]]

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

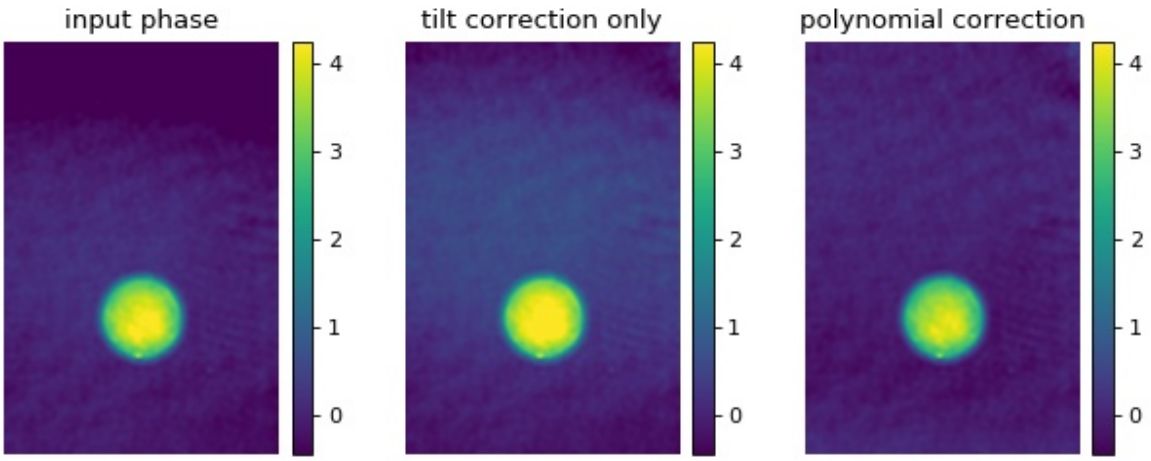
3.5 Background image 2nd order polynomial correction

This example extends the tilt correction ('background_tilt.py') to a second order polynomial correction for samples that exhibit more sophisticated phase aberrations. The phase background correction is computed from a ten pixel wide frame around the image. The phase data shown are computed from a hologram of a single myeloid leukemia cell (HL60) recorded using digital holographic microscopy (DHM) (see *[SSM+15]*).

`background_poly2o.py`

```
import matplotlib.pylab as plt
import numpy as np
# The data are stored in a .jpg file (lossy compression).
# If `PIL` is not found, try installing the `pillow` package.
from PIL import Image
```

(continues on next page)



(continued from previous page)

```
import qpimage

edata = np.array(Image.open("./data/hologram_cell_curved_bg.jpg"))

# create QPImage instance
qpi = qpimage.QPImage(data=edata, which_data="hologram")
pha0 = qpi.pha

# background correction using tilt only
qpi.compute_bg(which_data=["phase"],
               fit_offset="fit",
               fit_profile="tilt",
               border_px=10,
               )
pha_tilt = qpi.pha

# background correction using polynomial
qpi.compute_bg(which_data=["phase"],
               fit_offset="fit",
               fit_profile="poly2o",
               border_px=10,
               )
pha_poly2o = qpi.pha

# plot phase data
fig = plt.figure(figsize=(8, 3.3))

phakw = {"cmap": "viridis",
         "interpolation": "bicubic",
         "vmin": pha_poly2o.min(),
         "vmax": pha_poly2o.max()}

ax1 = plt.subplot(131, title="input phase")
map1 = ax1.imshow(pha0, **phakw)
plt.colorbar(map1, ax=ax1, fraction=.067, pad=0.04)

ax2 = plt.subplot(132, title="tilt correction only")
```

(continues on next page)

(continued from previous page)

```
map2 = ax2.imshow(pha_tilt, **phakw)
plt.colorbar(map2, ax=ax2, fraction=.067, pad=0.04)

ax3 = plt.subplot(133, title="polynomial correction")
map3 = ax3.imshow(pha_poly2o, **phakw)
plt.colorbar(map3, ax=ax3, fraction=.067, pad=0.04)

# disable axes
[ax.axis("off") for ax in [ax1, ax2, ax3]]

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

3.6 Digital hologram of a single cell

This example illustrates how qpimage can be used to analyze digital holograms. The hologram of a single myeloid leukemia cell (HL60) shown was recorded using digital holographic microscopy (DHM). Because the phase-retrieval method used in DHM is based on the discrete Fourier transform, there always is a residual background phase tilt which must be removed for further image analysis. The setup used for recording these data is described in reference *[SSM+15]*, which also contains a description of the hologram-to-phase conversion and phase background correction algorithms which qpimage is based on.

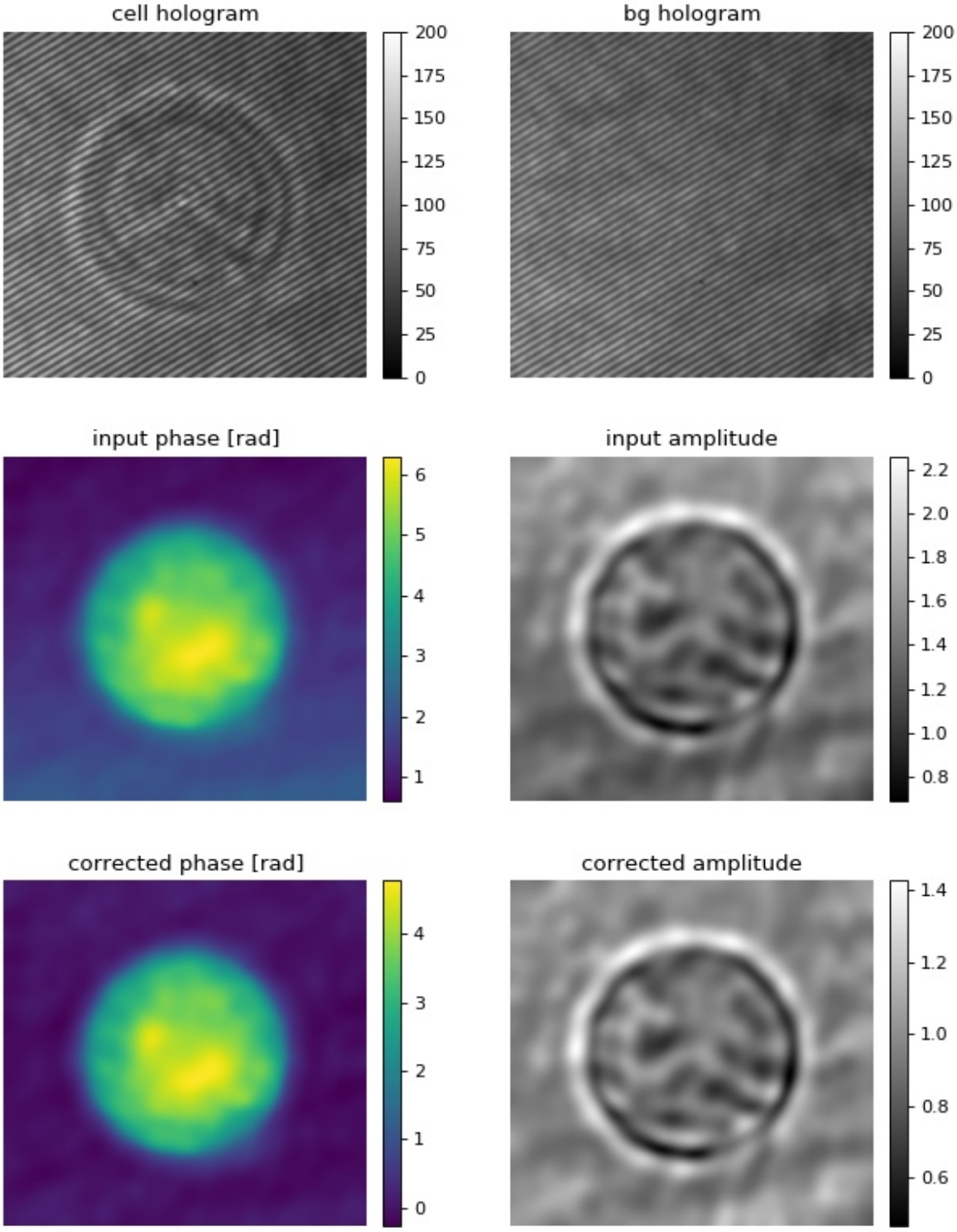
`hologram_cell.py`

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

# 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",
                      # This parameter allows to pass arguments to the
                      # hologram-analysis algorithm of qpimage.
                      # (see qpimage.holo.get_field)
                      holo_kw={
                               # For this hologram, the "smooth disk"
                               # filter yields the best trade-off
                               # between interference from the central
                               # band and image resolution.
                               "filter_name": "smooth disk",
                               # As can be seen in the hologram image,
                               # the sidebands are not positioned at
                               # an angle of 45° from the central band.
                               # If the filter size is 1/3 (default),
                               # the central band introduces line-
                               # artifacts to the reconstructed image.
                               "filter_size": 1/4
                               }
                      )
```

(continues on next page)


cell hologram
200
175
150
125
100
75
50
25
0
bg hologram
200
175
150
125
100
75
50
25
0
input phase [rad]
6
5
4
3
2
1
input amplitude
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
corrected phase [rad]
4
3
2
1
0
corrected amplitude
1.4
1.2
1.0
0.8
0.6

(continued from previous page)

```
amp0 = qpi.amp
pha0 = qpi.pha

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

# plot the properties of `qpi`
fig = plt.figure(figsize=(8, 10))

matplotlib.rcParams["image.interpolation"] = "bicubic"
holkw = {"cmap": "gray",
         "vmin": 0,
         "vmax": 200}

ax1 = plt.subplot(321, title="cell hologram")
map1 = ax1.imshow(edata["data"], **holkw)
plt.colorbar(map1, ax=ax1, fraction=.046, pad=0.04)

ax2 = plt.subplot(322, title="bg hologram")
map2 = ax2.imshow(edata["bg_data"], **holkw)
plt.colorbar(map2, ax=ax2, fraction=.046, pad=0.04)

ax3 = plt.subplot(323, title="input phase [rad]")
map3 = ax3.imshow(pha0)
plt.colorbar(map3, ax=ax3, fraction=.046, pad=0.04)

ax4 = plt.subplot(324, title="input amplitude")
map4 = ax4.imshow(amp0, cmap="gray")
plt.colorbar(map4, ax=ax4, fraction=.046, pad=0.04)

ax5 = plt.subplot(325, title="corrected phase [rad]")
map5 = ax5.imshow(qpi.pha)
plt.colorbar(map5, ax=ax5, fraction=.046, pad=0.04)

ax6 = plt.subplot(326, title="corrected amplitude")
map6 = ax6.imshow(qpi.amp, cmap="gray")
plt.colorbar(map6, ax=ax6, fraction=.046, pad=0.04)

# disable axes
[ax.axis("off") for ax in [ax1, ax2, ax3, ax4, ax5, ax6]]

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

3.7 Hologram filter choice

There are several parameters that influence the quality of phase and amplitude data retrieved from holograms. This example demonstrates the advantages and disadvantages of a three hologram filters in qpimage. For more information, please have a look at *qpimage.holo.get_field()*.

Several observations can be made:

- There appears to be a "bleed-through" of phase data into the amplitude data.
- A (sharp) disk filter introduces ringing artifacts in the amplitude and phase images.
- A smooth disk filter does not lead to such artifacts, but a dark halo is introduced around the coins in the amplitude image.
- The amplitude reconstruction with the gaussian filter does not exhibit the dark halo but, due to blurring, reveals less details.

To correctly interpret the data shown, please note that:

- This is a simulated hologram with *no* central band. For real data, the "filter_size" parameter also affects the reconstruction quality. Contributions from the central band can lead to strong artifacts. A balance between high resolution (large filter size) and small contributions from the central band (small filter size) usually has be found.
- It is not trivial to compare a gaussian filter with a disk filter in terms of filter size (sigma vs. radius). The gaussian filter takes into account larger frequencies and suppresses low frequencies. In qpimage, the actual gaussian filter size is chosen such that the resolution approximately matches that of the disk filter with a corresponding radius. In general however, the filter size parameter has to be examined when comparing the two.
- There is an inherent loss of information (resolution) in the holographic reconstruction process. The side band is isolated with a low-pass filter in Fourier space. The size and shape of this filter determine the resolution of the phase and amplitude images. As a result, the level of detail of all reconstructions shown is lower than that of the original images.

hologram_filters.py

```
import matplotlib.pylab as plt
import numpy as np
import qpimage
from skimage import color, data

# image of a galaxy recorded with the Hubble telescope
img1 = color.rgb2grey(data.hubble_deep_field())[354:504, 70:220]
# image of a coin
img2 = color.rgb2grey(data.coins())[150:300, 70:220]

pha = img1/img1.max() * 2 * np.pi
amp = img2/img2.mean()

# create a hologram
x, y = np.mgrid[0:150, 0:150]
hologram = 2 * amp * np.cos(-2 * (x + y) + pha)

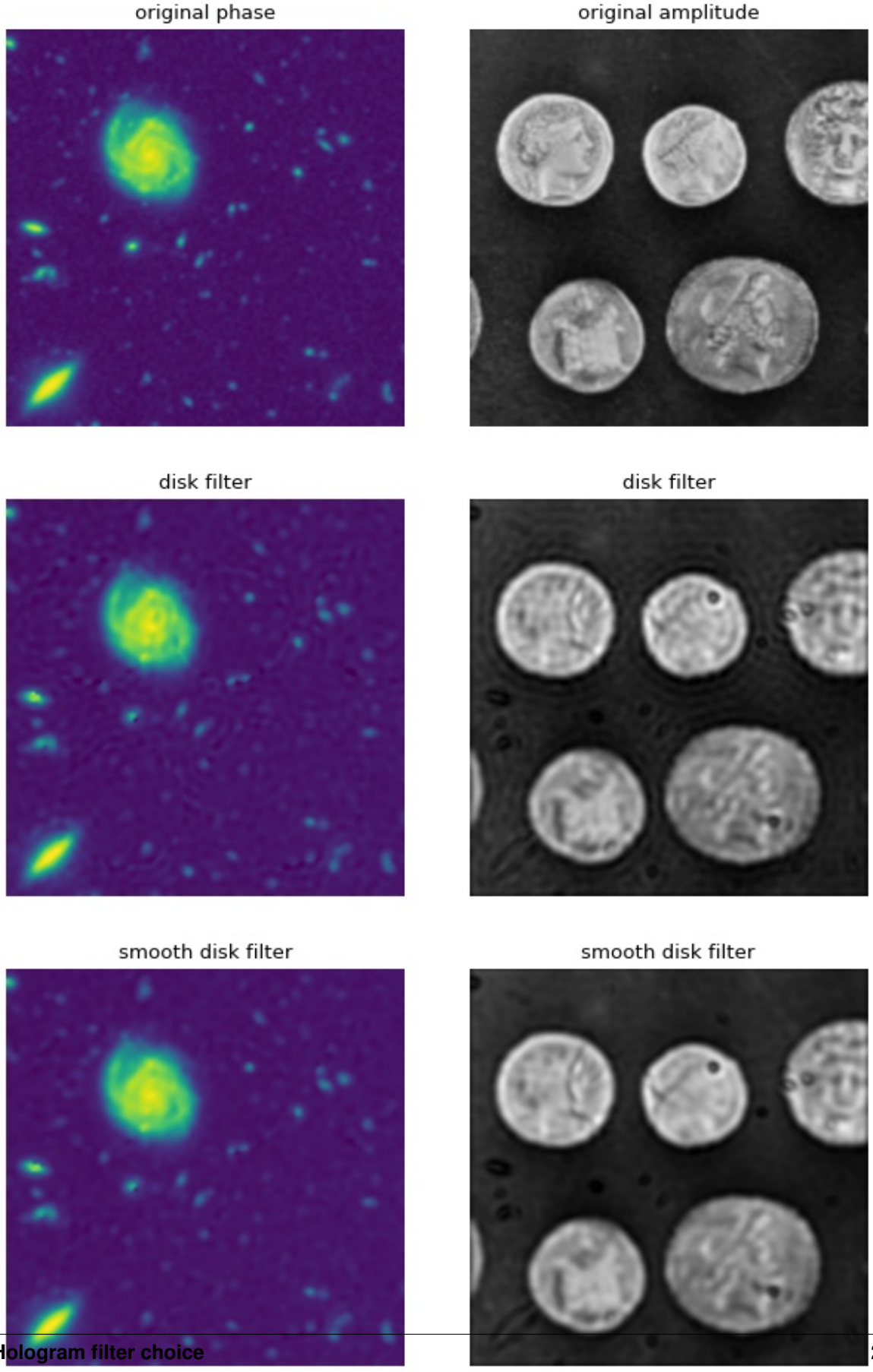
filters = ["disk", "smooth disk", "gauss"]
qpis = []

for filter_name in filters:
    qpi = qpimage.QPImage(data=hologram,
                          which_data="hologram",
                          holo_kw={"filter_size": .5,
                                   "filter_name": filter_name})
    qpis.append(qpi)

fig = plt.figure(figsize=(8, 16))

phakw = {"interpolation": "bicubic",
```

(continues on next page)


original phase
original amplitude
disk filter
disk filter
smooth disk filter
smooth disk filter
gauss filter
gauss filter

(continued from previous page)

```
          "cmap": "viridis",
          "vmin": pha.min(),
          "vmax": pha.max(),
          }

ampkw = {"interpolation": "bicubic",
          "cmap": "gray",
          "vmin": amp.min(),
          "vmax": amp.max()
          }

numrows = len(filters) + 1

plt.subplot(numrows, 2, 1, title="original phase")
plt.imshow(pha, **phakw)

ax2 = plt.subplot(numrows, 2, 2, title="original amplitude")
plt.imshow(amp, **ampkw)

for ii in range(len(filters)):
    # phase
    plt.subplot(numrows, 2, 2*ii+3, title=filters[ii]+" filter")
    plt.imshow(qpis[ii].pha, **phakw)
    # amplitude
    plt.subplot(numrows, 2, 2*ii+4, title=filters[ii]+" filter")
    plt.imshow(qpis[ii].amp, **ampkw)

# disable axes
for ax in fig.get_axes():
    ax.axis("off")

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

CHAPTER 4

Code reference

4.1 bg_estimate (background-estimation)

4.1.1 Constants

qpimage.bg_estimate.**VALID_FIT_OFFSETS = ['fit', 'gauss', 'mean', 'mode']**
valid values for keyword argument *fit_offset* in *estimate()*

qpimage.bg_estimate.**VALID_FIT_PROFILES = ['offset', 'poly2o', 'tilt']**
valid values for keyword argument *fit_profile* in *estimate()*

4.1.2 Methods

qpimage.bg_estimate.**estimate**(*data*, *fit_offset='mean'*, *fit_profile='tilt'*, *border_px=0*, *from_binary=None*, *ret_binary=False*)
Estimate the background value of an image

Parameters

- **data** (*np.ndarray*) – Data from which to compute the background value
- **fit_profile** (*str*) – The type of background profile to fit:
  - "offset": offset only
  - "poly2o": 2D 2nd order polynomial with mixed terms
  - "tilt": 2D linear tilt with offset (default)
- **fit_offset** (*str*) – The method for computing the profile offset
  - "fit": offset as fitting parameter
  - "gauss": center of a gaussian fit
  - "mean": simple average

  - ”mode”: mode (see *qpimage.bg_estimate.mode*)

- **border_px** (*float*) – Assume that a frame of *border_px* pixels around the image is background.
- **from_binary** (*boolean np.ndarray or None*) – Use a boolean array to define the background area. The binary image must have the same shape as the input data. *True* elements are used for background estimation.
- **ret_binary** (*bool*) – Return the binary image used to compute the background.

Notes

If both *border_px* and *from_binary* are given, the intersection of the two is used, i.e. the positions where both, the binary frame and *from_binary*, are *True*.

qpimage.bg_estimate.**offset_gaussian**(*data*)
Fit a gaussian model to *data* and return its center

qpimage.bg_estimate.**offset_mode**(*data*)
Compute Mode using a histogram with *sqrt(data.size)* bins

qpimage.bg_estimate.**profile_tilt**(*data*, *binary*)
Fit a 2D tilt to *data[binary]*

qpimage.bg_estimate.**profile_poly2o**(*data*, *binary*)
Fit a 2D 2nd order polynomial to *data[binary]*

qpimage.bg_estimate.**poly2o_model**(*params*, *shape*)
lmfit 2nd order polynomial model

qpimage.bg_estimate.**poly2o_residual**(*params*, *data*, *binary*)
lmfit 2nd order polynomial residuals

qpimage.bg_estimate.**tilt_model**(*params*, *shape*)
lmfit tilt model

qpimage.bg_estimate.**tilt_residual**(*params*, *data*, *binary*)
lmfit tilt residuals

4.2 core (QPImage)

4.2.1 Constants

qpimage.core.**VALID_INPUT_DATA = ['field', 'hologram', 'phase', ('phase', 'amplitude'), ('pl**
valid combinations for keyword argument *which_data*

4.2.2 Classes

class qpimage.core.**QPImage**(*data=None*, *bg_data=None*, *which_data='phase'*, *meta_data={}*, *holo_kw={}*, *h5file=None*, *h5mode='a'*, *h5dtype='float32'*)
Quantitative phase image manipulation

This class implements various tasks for quantitative phase imaging, including phase unwrapping, background correction, numerical focusing, and data export.

Parameters

- **data** (*2d ndarray (float or complex) or list*) – The experimental data (see *which_data*)
- **bg_data** (2d ndarray (float or complex), list, or *None*) – The background data (must be same type as *data*)
- **which_data** (*str*) – String or comma-separated list of strings indicating the order and type of input data. Valid values are "hologram", "field", "phase", "phase,amplitude", or "phase,intensity", where the latter two require an indexable object with the phase data as first element.
- **meta_data** (*dict*) – Meta data associated with the input data. see *qpimage.meta.META_KEYS*
- **holo_kw** (*dict*) – Special keyword arguments for phase retrieval from hologram data (*which_data="hologram"*). See *qpimage.holo.get_field()* for valid keyword arguments.

  New in version 0.1.6.
- **h5file** (*str, h5py.Group, h5py.File, or None*) – A path to an hdf5 data file where all data is cached. If set to *None* (default), all data will be handled in memory using the "core" driver of the `h5py`'s `File` class. If the file does not exist, it is created. If the file already exists, it is opened with the file mode defined by *hdf5_mode*. If this is an instance of h5py.Group or h5py.File, then this will be used to internally store all data.
- **h5mode** (*str*) – Valid file modes are (only applies if *h5file* is a path)
  - "r": Readonly, file must exist
  - "r+": Read/write, file must exist
  - "w": Create file, truncate if exists
  - "w-" or "x": Create file, fail if exists
  - "a": Read/write if exists, create otherwise (default)
- **h5dtype** (*str*) – The datatype in which to store the image data. The default is "float32" which is sufficient for 2D image analysis and consumes only half the disk space of the numpy default "float64".

Notes

QPImage is slicable; the following returns a new QPImage with the same meta data, but with all background corrections merged into the raw data:

```
qpi = QPImage(data=...)
qpi_scliced = qpi[10:20, 40:30]
```

bg_amp
: background amplitude image

bg_pha
: background phase image

amp
: background-corrected amplitude image

field
: background-corrected complex field

info
list of tuples with QPImage meta data

meta
dictionary with imaging meta data

pha
background-corrected phase image

raw_amp
raw amplitude image

raw_pha
raw phase image

shape
size of image dimensions

clear_bg(*which_data=('amplitude', 'phase')*, *keys='fit'*)
Clear background correction

Parameters

- **which_data** (*str or list of str*) – From which type of data to remove the background information. The list contains either "amplitude", "phase", or both.
- **keys** (*str or list of str*) – Which type of background data to remove. One of:
  - "fit": the background data computed with `qpimage.QPImage.compute_bg()`
  - "data": the experimentally obtained background image

compute_bg(*which_data='phase'*, *fit_offset='mean'*, *fit_profile='tilt'*, *border_m=0*, *border_perc=0*, *border_px=0*, *from_binary=None*, *ret_binary=False*)
Compute background correction

Parameters

- **which_data** (*str or list of str*) – From which type of data to remove the background information. The list contains either "amplitude", "phase", or both.
- **fit_profile** (*str*) – The type of background profile to fit:
  - "offset": offset only
  - "poly2o": 2D 2nd order polynomial with mixed terms
  - "tilt": 2D linear tilt with offset (default)
- **fit_offset** (*str*) – The method for computing the profile offset
  - "fit": offset as fitting parameter
  - "gauss": center of a gaussian fit
  - "mean": simple average
  - "mode": mode (see *qpimage.bg_estimate.mode*)
- **border_m** (*float*) – Assume that a frame of *border_m* meters around the image is background. The value is converted to pixels and rounded.
- **border_perc** (*float*) – Assume that a frame of *border_perc* percent around the image is background. The value is converted to pixels and rounded. If the aspect ratio of the image is not one, then the average of the data's shape is used to compute the percentage in pixels.

- **border_px** (*float*) – Assume that a frame of *border_px* pixels around the image is background.
- **from_binary** (*boolean np.ndarray or None*) – Use a boolean array to define the background area. The binary image must have the same shape as the input data. *True* elements are used for background estimation.
- **ret_binary** (*bool*) – Return the binary image used to compute the background.

Notes

The *border_** values are translated to pixel values and the largest pixel border is used to generate a binary image for background computation.

If any of the *border_** arguments are non-zero and *from_binary* is given, the intersection of the two is used, i.e. the positions where both, the binary frame and *from_binary*, are *True*.

See also:

qpimage.bg_estimate.estimate()

copy(*h5file=None*)
Create a copy of the current instance

This is done by recursively copying the underlying hdf5 data.

Parameters **h5file** (*str, h5py.File, h5py.Group, or None*) – see *QPImage.__init__*

refocus(*distance*, *method='helmholtz'*, *h5file=None*, *h5mode='a'*)
Compute a numerically refocused QPImage

Parameters

- **distance** (*float*) – Focusing distance [m]
- **method** (*str*) – Refocusing method, one of ["helmholtz","fresnel"]
- **h5file** (*str, h5py.Group, h5py.File, or None*) – A path to an hdf5 data file where the QPImage is cached. If set to *None* (default), all data will be handled in memory using the "core" driver of the `h5py`'s `File` class. If the file does not exist, it is created. If the file already exists, it is opened with the file mode defined by *hdf5_mode*. If this is an instance of h5py.Group or h5py.File, then this will be used to internally store all data.
- **h5mode** (*str*) – Valid file modes are (only applies if *h5file* is a path)
  - "r": Readonly, file must exist
  - "r+": Read/write, file must exist
  - "w": Create file, truncate if exists
  - "w-" or "x": Create file, fail if exists
  - "a": Read/write if exists, create otherwise (default)

Returns **qpi** – Refocused phase and amplitude data

Return type *QPImage*

See also:

nrefocus library used for numerical focusing

set_bg_data(*bg_data*, *which_data=None*)
Set background amplitude and phase data

Parameters

- **bg_data** (2d ndarray (float or complex), list, QPImage, or *None*) – The background data (must be same type as *data*). If set to *None*, the background data is reset.
- **which_data** (*str*) – String or comma-separated list of strings indicating the order and type of input data. Valid values are "field", "phase", "phase,amplitude", or "phase,intensity", where the latter two require an indexable object for *bg_data* with the phase data as first element.

4.2.3 Methods

qpimage.core.**copyh5**(*inh5*, *outh5*)
Recursively copy all hdf5 data from one group to another

Parameters

- **inh5** (*str, h5py.File, or h5py.Group*) – The input hdf5 data. This can be either a file name or an hdf5 object.
- **outh5** (*str, h5py.File, h5py.Group, or None*) – The output hdf5 data. This can be either a file name or an hdf5 object. If set to *None*, a new hdf5 object is created in memory.

Notes

All data in outh5 are overridden by the inh5 data.

4.3 holo (hologram analysis)

4.3.1 Methods

qpimage.holo.**find_sideband**(*ft_data*, *which=1*, *copy=True*)
Find the side band position of a hologram

The hologram is Fourier-transformed and the side band is determined by finding the maximum amplitude in Fourier space.

Parameters

- **ft_data** (*2d ndarray*) – Fourier transform of the hologram image
- **which** (*+1 or -1*) – which sideband to search for:
  - +1: upper half
  - -1: lower half
- **copy** (*bool*) – copy *ft_data* before modification

Returns **fsx, fsy** – coordinates of the side band in Fourier space frequencies

Return type tuple of floats

qpimage.holo.**fourier2dpad**(*data*, *zero_pad=True*)
Compute the 2D Fourier transform with zero padding

Parameters

- **data** (*2d fload ndarray*) – real-valued image data
- **zero_pad** (*bool*) – perform zero-padding to next order of 2

qpimage.holo.**get_field**(*hologram*, *sideband=1*, *filter_name='disk'*, *filter_size=0.3333333333333333*, *subtract_mean=True*, *zero_pad=True*, *copy=True*)
Compute the complex field from a hologram using Fourier analysis

Parameters

- **hologram** (*real-valued 2d ndarray*) – hologram data
- **sideband** (*+1, -1, or tuple of (float, float)*) – specifies the location of the sideband:
  - +1: sideband in the upper half in Fourier space, exact location is found automatically
  - -1: sideband in the lower half in Fourier space, exact location is found automatically
  - (float, float): sideband coordinates in frequencies in interval [1/"axes size", .5]
- **filter_name** (*str*) – specifies the filter to use, one of
  - "disk": binary disk with radius *filter_size*
  - "smooth disk": disk with radius *filter_size* convolved with a radial gaussian (*sigma=filter_size/5*)
  - "gauss": radial gaussian (*sigma=0.6*filter_size*)
  - "square": binary square with side length *filter_size*
  - "smooth square": square with side length *filter_size* convolved with square gaussian (*sigma=filter_size/5*)
  - "tukey": a square tukey window of width *2*filter_size* and *alpha=0.1*
- **filter_size** (*float*) – Size of the filter in Fourier space in fractions of the distance between central band and sideband. See *filter_shape* for interpretation of *filter_size*.
- **subtract_mean** (*bool*) – If True, remove the mean of the hologram before performing the Fourier transform. This setting is recommended as it can reduce artifacts from frequencies around the central band.
- **zero_pad** (*bool*) – Perform zero-padding before applying the FFT. Setting *zero_pad* to *False* increases speed but might introduce image distortions such as tilts in the phase and amplitude data or dark borders in the amplitude data.
- **copy** (*bool*) – If set to True, input *data* is not edited.
- **and y0 are center of the filter** (*x0*) –
- **is factor for "radius" of filter (sqrt(x0^2 + y0^2)/np.pi)** (*R*) –
- **can be "disk" or "gauss"** (*filter_type*) –

Notes

Even though the size of the "gauss" filter approximately matches the frequencies of the "disk" filter, it takes into account higher frequencies as well and thus suppresses ringing artifacts for data that contain jumps in the phase image.

4.4 image_data (basic image management)

4.4.1 Constants

qpimage.image_data.**COMPRESSION = {'compression': 'gzip', 'compression_opts': 9}**
default hdf5 compression keyword arguments

qpimage.image_data.**VALID_BG_KEYS = ['data', 'fit']**
valid background data identifiers

4.4.2 Classes

class qpimage.image_data.**Amplitude**(*h5*, *h5dtype='float32'*)
Bases: *qpimage.image_data.ImageData*

Dedicated class for amplitude image data

For amplitude image data, background correction is defined by dividing the raw image by the background image.

Parameters

- **h5** (*h5py.Group*) – HDF5 group where all data is kept
- **h5dtype** (*str*) – The datatype in which to store the image data. The default is "float32" which is sufficient for 2D image analysis and consumes only half the disk space of the numpy default "float64".

class qpimage.image_data.**Phase**(*h5*, *h5dtype='float32'*)
Bases: *qpimage.image_data.ImageData*

Dedicated class for phase image data

For phase image data, background correction is defined by subtracting the background image from the raw image.

Parameters

- **h5** (*h5py.Group*) – HDF5 group where all data is kept
- **h5dtype** (*str*) – The datatype in which to store the image data. The default is "float32" which is sufficient for 2D image analysis and consumes only half the disk space of the numpy default "float64".

class qpimage.image_data.**ImageData**(*h5*, *h5dtype='float32'*)
Base class for image management

See also:

Amplitude ImageData with amplitude background correction

Phase ImageData with phase background correction

Parameters

- **h5** (*h5py.Group*) – HDF5 group where all data is kept
- **h5dtype** (*str*) – The datatype in which to store the image data. The default is "float32" which is sufficient for 2D image analysis and consumes only half the disk space of the numpy default "float64".

bg
combined background image data

image
background corrected image data

info
list of background correction parameters

raw
raw (uncorrected) image data

del_bg(*key*)
Remove the background image data

Parameters **key** (*str*) – One of *VALID_BG_KEYS*

estimate_bg(*fit_offset='mean'*, *fit_profile='tilt'*, *border_px=0*, *from_binary=None*, *ret_binary=False*)
Estimate image background

Parameters

- **fit_profile** (*str*) – The type of background profile to fit:
  - "offset": offset only
  - "poly2o": 2D 2nd order polynomial with mixed terms
  - "tilt": 2D linear tilt with offset (default)
- **fit_offset** (*str*) – The method for computing the profile offset
  - "fit": offset as fitting parameter
  - "gauss": center of a gaussian fit
  - "mean": simple average
  - "mode": mode (see *qpimage.bg_estimate.mode*)
- **border_px** (*float*) – Assume that a frame of *border_px* pixels around the image is background.
- **from_binary** (*boolean np.ndarray or None*) – Use a boolean array to define the background area. The binary image must have the same shape as the input data.'True' elements are used for background estimation.
- **ret_binary** (*bool*) – Return the binary image used to compute the background.

Notes

If both *border_px* and *from_binary* are given, the intersection of the two resulting binary images is used.

The arguments passed to this method are stored in the hdf5 file *self.h5* and are used for optional integrity checking using *qpimage.integrity_check.check*.

See also:

qpimage.bg_estimate.estimate()

get_bg(*key=None*, *ret_attrs=False*)
Get the background data

Parameters

- **key** (*None or str*) – A user-defined key that identifies the background data. Examples are "data" for experimental data, or "fit" for an estimated background correction (see *VALID_BG_KEYS*). If set to *None*, returns the combined background image (*ImageData.bg*).
- **ret_attrs** (*bool*) – Also returns the attributes of the background data.

set_bg(*bg*, *key='data'*, *attrs={}*)
Set the background data

Parameters

- **bg** (*numbers.Real, 2d ndarray, ImageData, or h5py.Dataset*) – The background data. If *bg* is an *h5py.Dataset* object, it must exist in the same hdf5 file (a hard link is created). If set to *None*, the data will be removed.
- **key** (*str*) – One of *VALID_BG_KEYS*)
- **attrs** (*dict*) – List of background attributes

See also:

del_bg() removing background data

4.4.3 Methods

qpimage.image_data.**write_image_dataset**(*group*, *key*, *data*, *h5dtype=None*)
Write an image to an hdf5 group as a dataset

This convenience function sets all attributes such that the image can be visualized with HDFView, sets the compression and fletcher32 filters, and sets the chunk size to the image shape.

Parameters

- **group** (*h5py.Group*) – HDF5 group to store data to
- **key** (*str*) – Dataset identifier
- **data** (*np.ndarray of shape (M,N)*) – Image data to store
- **h5dtype** (*str*) – The datatype in which to store the image data. The default is the datatype of *data*.

Returns **dataset** – The created HDF5 dataset object

Return type h5py.Dataset

4.5 integrity_check (check QPImage data)

4.5.1 Exceptions

exception qpimage.integrity_check.**IntegrityCheckError**
Raised when a QPImage data set is incomplete or corrupt

4.5.2 Methods

qpimage.integrity_check.**check**(*qpi_or_h5file, checks=['attributes', 'background']*)
Checks various properties of a *qpimage.core.QPImage* instance

Parameters

- **qpi_or_h5file** (*qpimage.core.QPImage or str*) – A QPImage object or a path to an hdf5 file
- **checks** (*list of str*) – Which checks to perform ("attributes" and/or "background")

Raises *IntegrityCheckError* – if the checks fail

qpimage.integrity_check.**check_attributes**(*qpi*)
Check QPimage attributes

Parameters **qpi** (qpimage.core.QPImage) –

Raises *IntegrityCheckError* – if the check fails

qpimage.integrity_check.**check_background**(*qpi*)
Check QPimage background data

Parameters **qpi** (qpimage.core.QPImage) –

Raises *IntegrityCheckError* – if the check fails

4.6 meta (definitions for QPImage meta data)

4.6.1 Constants

qpimage.meta.**META_KEYS = ['medium index', 'pixel size', 'time', 'wavelength', 'dm exclude',**
valid *qpimage.core.QPImage* meta data keys

4.6.2 Exceptions

exception qpimage.meta.**MetaDataMissingError**
Raised when meta data is missing

4.6.3 Classes

class qpimage.meta.**MetaDict**(**args*, ***kwargs*)
Bases: dict

Management of meta data variables

Valid key names are combined in the *qpimage.meta.META_KEYS* variable.

4.7 series (QPSeries)

4.7.1 Classes

class qpimage.series.**QPSeries**(*qpimage_list=[]*, *meta_data={}*, *h5file=None*, *h5mode='a'*, *identifier=None*)

Quantitative phase image series

Parameters

- **qpimage_list** (*list of QPImage*) – A list of instances of `qpimage.QPImage`.
- **meta_data** (*dict*) – Meta data associated with the input data (see `qpimage.META_KEYS`). This overrides the meta data of the QPImages in *qpimage_list* and, if *h5file* is given and *h5mode* is not "r", overrides the meta data in *h5file*.
- **h5file** (*str, h5py.Group, h5py.File, or None*) – A path to an hdf5 data file where all data is cached. If set to *None* (default), all data will be handled in memory using the "core" driver of the `h5py`'s `File` class. If the file does not exist, it is created. If the file already exists, it is opened with the file mode defined by *hdf5_mode*. If this is an instance of h5py.Group or h5py.File, then this will be used to internally store all data. If *h5file* is given and *qpimage_list* is not empty, all QPImages in *qpimage_list* are appended to *h5file* in the given order.
- **h5mode** (*str*) – Valid file modes are (only applies if *h5file* is a path):
  - "r": Readonly, file must exist
  - "r+": Read/write, file must exist
  - "w": Create file, truncate if exists
  - "w-" or "x": Create file, fail if exists
  - "a": Read/write if exists, create otherwise (default)

identifier

unique identifier of the series

add_qpimage(*qpi*, *identifier=None*, *bg_from_idx=None*)

Add a QPImage instance to the QPSeries

Parameters

- **qpimage** (*QPImage*) – The QPImage that is added to the series
- **identifier** (*str*) – Identifier key for *qpi*
- **bg_from_idx** (*int or None*) – Use the background data from the data stored in this index, creating hard links within the hdf5 file. (Saves memory if e.g. all qpimages is corrected with the same data)

get_qpimage(*index*)

Return a single QPImage of the series

Parameters **index** (*int or str*) – Index or identifier of the QPImage

Notes

Instead of `qps.get_qpimage(index)`, it is possible to use the short-hand `qps[index]`.

CHAPTER 5

Bilbliography

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Bibliography

[Bar52] R. Barer. Interference Microscopy and Mass Determination. *Nature*, 169(4296):366–367, 1952. doi:10.1038/169366b0.

[CCC+06] T. Colomb, E. Cuche, F. Charrière, J. Kühn, N. Aspert, F. Montfort, P. Marquet, and C. Depeursinge. Automatic procedure for aberration compensation in digital holographic microscopy and applications to specimen shape compensation. *Applied Optics*, 45(5):851, feb 2006. doi:10.1364/ao.45.000851.

[DW52] H. G. Davies and M. H. F. Wilkins. Interference Microscopy and Mass Determination. *Nature*, 169(4300):541, 1952. doi:10.1038/169541a0.

[SCG+17] M. Schürmann, G. Cojoc, S. Girardo, E. Ulbricht, J. Guck, and P. Müller. 3d correlative single-cell imaging utilizing fluorescence and refractive index tomography. *Journal of Biophotonics*, pages n/a, aug 2017. doi:10.1002/jbio.201700145.

[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

qpimage, 3
qpimage.bg_estimate, 23
qpimage.holo, 28
qpimage.image_data, 30
qpimage.integrity_check, 33

Index

A

add_qpimage() (qpimage.series.QPSeries method), 34
amp (qpimage.core.QPImage attribute), 25
Amplitude (class in qpimage.image_data), 30

B

bg (qpimage.image_data.ImageData attribute), 31
bg_amp (qpimage.core.QPImage attribute), 25
bg_pha (qpimage.core.QPImage attribute), 25

C

check() (in module qpimage.integrity_check), 33
check_attributes() (in module qpimage.integrity_check), 33
check_background() (in module qpimage.integrity_check), 33
clear_bg() (qpimage.core.QPImage method), 26
COMPRESSION (in module qpimage.image_data), 30
compute_bg() (qpimage.core.QPImage method), 26
copy() (qpimage.core.QPImage method), 27
copyh5() (in module qpimage.core), 28

D

del_bg() (qpimage.image_data.ImageData method), 31

E

estimate() (in module qpimage.bg_estimate), 23
estimate_bg() (qpimage.image_data.ImageData method), 31

F

field (qpimage.core.QPImage attribute), 25
find_sideband() (in module qpimage.holo), 28
fourier2dpad() (in module qpimage.holo), 28

G

get_bg() (qpimage.image_data.ImageData method), 32
get_field() (in module qpimage.holo), 29
get_qpimage() (qpimage.series.QPSeries method), 34

I

identifier (qpimage.series.QPSeries attribute), 34
image (qpimage.image_data.ImageData attribute), 31
ImageData (class in qpimage.image_data), 30
info (qpimage.core.QPImage attribute), 25
info (qpimage.image_data.ImageData attribute), 31
IntegrityCheckError, 33

M

meta (qpimage.core.QPImage attribute), 26
META_KEYS (in module qpimage.meta), 33
MetaDataMissingError, 33
MetaDict (class in qpimage.meta), 33

O

offset_gaussian() (in module qpimage.bg_estimate), 24
offset_mode() (in module qpimage.bg_estimate), 24

P

pha (qpimage.core.QPImage attribute), 26
Phase (class in qpimage.image_data), 30
poly2o_model() (in module qpimage.bg_estimate), 24
poly2o_residual() (in module qpimage.bg_estimate), 24
profile_poly2o() (in module qpimage.bg_estimate), 24
profile_tilt() (in module qpimage.bg_estimate), 24

Q

QPImage (class in qpimage.core), 24
qpimage (module), 3
qpimage.bg_estimate (module), 23
qpimage.holo (module), 28
qpimage.image_data (module), 30
qpimage.integrity_check (module), 33
QPSeries (class in qpimage.series), 34

R

raw (qpimage.image_data.ImageData attribute), 31
raw_amp (qpimage.core.QPImage attribute), 26
raw_pha (qpimage.core.QPImage attribute), 26

refocus() (qpimage.core.QPImage method), 27

S

set_bg() (qpimage.image_data.ImageData method), 32
set_bg_data() (qpimage.core.QPImage method), 28
shape (qpimage.core.QPImage attribute), 26

T

tilt_model() (in module qpimage.bg_estimate), 24
tilt_residual() (in module qpimage.bg_estimate), 24

V

VALID_BG_KEYS (in module qpimage.image_data), 30
VALID_FIT_OFFSETS (in module qpimage.bg_estimate), 23
VALID_FIT_PROFILES (in module qpimage.bg_estimate), 23
VALID_INPUT_DATA (in module qpimage.core), 24

W

write_image_dataset() (in module qpimage.image_data), 32