qpformat Documentation

Release 0.2.1

Paul Müller

Jun 20, 2018

Contents:

Qpformat is a Python3 library for opening quantitative phase imaging data file formats. This is the documentation of qpformat version 0.2.1.

CHAPTER 1

Introduction

1.1 Phase image retrieval

1.2 Why qpformat?

CHAPTER 2

Getting started

2.1 Installing qpformat

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

- numpy,
- scipy,
- qpimage (phase data manipulation),

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

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

2.2 User API

Qpformat supports several file formats (*qpformat.file_formats.formats*), which are divided into `qpformat.file_formats.SingleData` (the experimental data file format contains only one phase image) and *qpformat.file_formats.SeriesData* (the experimental data file format supports multiple phase images). From these base classes, all data file formats are derived. The idea is that experimental data is not loaded into memory until the *get_qpimage* method is called which returns a `qpimage.QPImage` object.

2.2.1 Basic Usage

The file format type is determined automatically by qpformat. If the file format is implemented in qpformat, experimental data can be loaded with the `qpformat.load_data()` method.

```
# Obtain a qpformat.file_formats.SingleData object
# (the data is not loaded into memory, only the meta data is read)
ds = qpformat.load_data(path="/path/to/SID PHA_xxx.tif")
# Get the quantitative phase data (a qpimage.QPImage is returned)
qpi = ds.get_qpimage()
```

2.2.2 SingleData

class qpformat.file_formats.**SeriesData**(*path*, *meta_data={}*, *holo_kw={}*, *as_type='float32'*)

background_identifier = None
Unique string that identifies the background data that was set using *set_bg*.

identifier
Return a unique identifier for the given data set

get_identifier(*idx*)
Return an identifier for the data at index *idx*

get_name(*idx*)
Return name of data at index *idx*

get_qpimage(*idx*)
Return background-corrected QPImage of data at index *idx*

get_qpimage_raw(*idx*)
Return QPImage without background correction

get_time(*idx*)
Return time of data at index *idx*

saveh5(*h5file*)
Save the data set as an hdf5 file (QPImage format)

set_bg(*dataset*)
Set background data

Parameters **dataset** (*DataSet*, *qpimage.QPImage*, or int) – If the `len(dataset)` matches `len(self)`, then background correction is performed element-wise. Otherwise, `len(dataset)` must be one and is used for all data of `self`.

See also:

get_qpimage() obtain the background corrected QPImage

static verify(*path*)
Verify that *path* has this file format

Returns *True* if the file format matches. The implementation of this method should be fast and memory efficient, because e.g. the "GroupFolder" file format depends on it.

2.2.3 SeriesData

class qpformat.file_formats.**SeriesData**(*path*, *meta_data={}*, *holo_kw={}*, *as_type='float32'*)

background_identifier = None
Unique string that identifies the background data that was set using *set_bg*.

identifier
Return a unique identifier for the given data set

get_identifier(*idx*)
Return an identifier for the data at index *idx*

get_name(*idx*)
Return name of data at index *idx*

get_qpimage(*idx*)
Return background-corrected QPImage of data at index *idx*

get_qpimage_raw(*idx*)
Return QPImage without background correction

get_time(*idx*)
Return time of data at index *idx*

saveh5(*h5file*)
Save the data set as an hdf5 file (QPImage format)

set_bg(*dataset*)
Set background data

Parameters **dataset** (*DataSet*, *qpimage.QPImage*, or int) – If the `len(dataset)` matches `len(self)`, then background correction is performed element-wise. Otherwise, `len(dataset)` must be one and is used for all data of `self`.

See also:

get_qpimage() obtain the background corrected QPImage

static verify(*path*)
Verify that *path* has this file format

Returns *True* if the file format matches. The implementation of this method should be fast and memory efficient, because e.g. the "GroupFolder" file format depends on it.

2.2.4 Constants

qpformat.file_formats.**formats = [<class 'qpformat.file_formats.SeriesFolder'>, <class 'qpf**
list() -> new empty list list(iterable) -> new list initialized from iterable's items

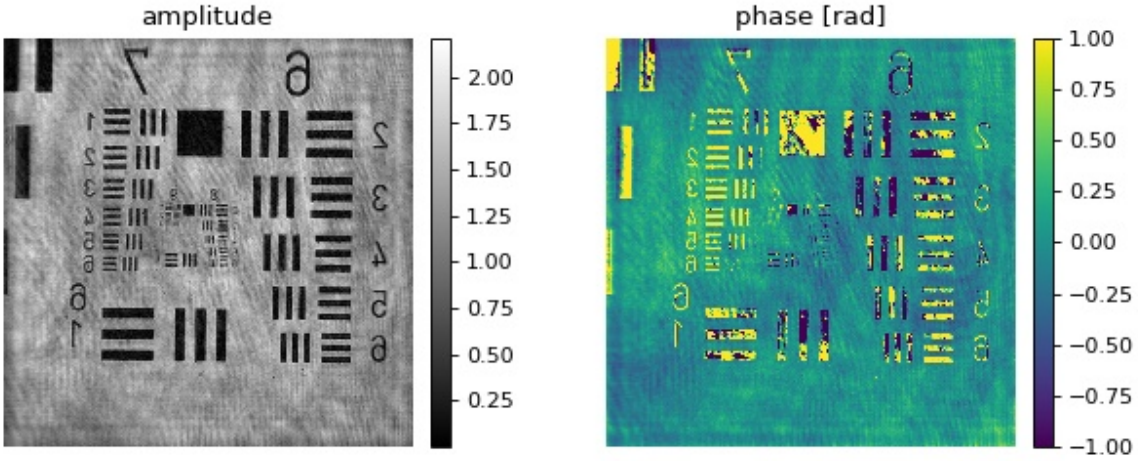
CHAPTER 3

Examples

3.1 Hologram from tif file

This example illustrates how to retrieve phase and amplitude from a hologram stored as a tif file. The experimental hologram is a U.S. Air Force test target downloaded from the Submersible Holographic Astrobiology Microscope with Ultraresolution (SHAMU) project *[BBL+17]*. The values for pixel resolution, wavelength, and reconstruction distance are taken from the corresponding Python example.

The object returned by the `get_qpimage <qpformat.file_formats.dataset.SingleData.get_qpimage()` function is an instance of `qpimage.QPImage` which allows for field refocusing. The refocused QPImage is background-corrected using a polynomial fit to the phase data at locations where the amplitude data is not attenuated (bright regions in the amplitude image).

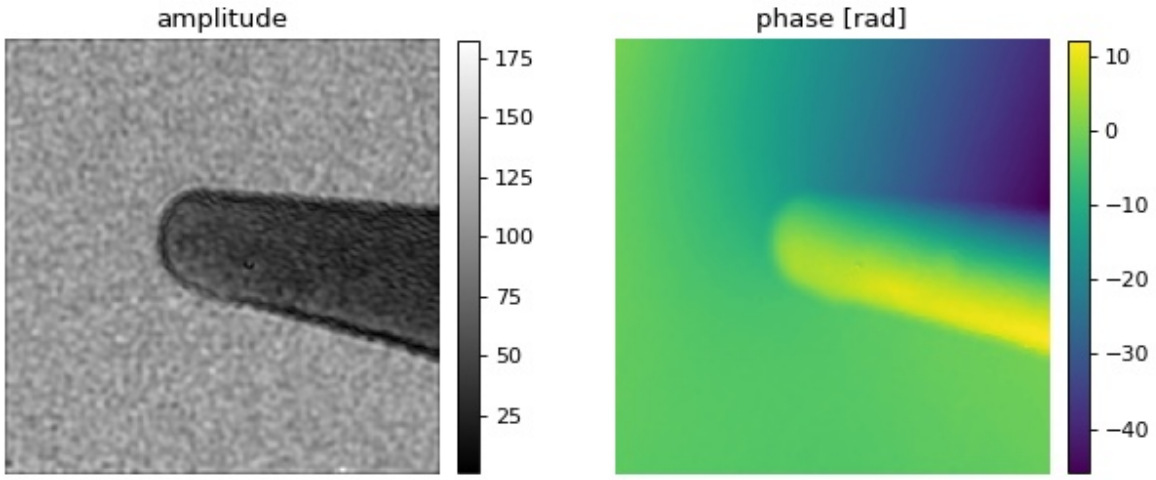


`tif_hologram.py`

```python
import urllib.request
import os

import matplotlib.pylab as plt
import qpformat


# load the experimental data
dl_loc = "https://github.com/bmorris3/shampoo/raw/master/data/"
dl_name = "USAF_test.tif"
if not os.path.exists(dl_name):
    urllib.request.urlretrieve(dl_loc + dl_name, dl_name)


ds = qpformat.load_data(dl_name,
                        # manually set meta data
                        meta_data={"pixel size": 3.45e-6,
                                   "wavelength": 405e-9,
                                   "medium index": 1},
                        # set filter size to 1/2 (defaults to 1/3)
                        # which increases the image resolution
                        holo_kw={"filter_size": .5})

# retrieve the qpimage.QPImage instance
qpi = ds.get_qpimage()
# refocus `qpi` to 0.03685m
qpi_foc = qpi.refocus(0.03685)
# perform an offset-based amplitude correction
qpi_foc.compute_bg(which_data="amplitude",
                   fit_profile="offset",
                   fit_offset="mode",
                   border_px=10,
                   )
# perform a phase correction using
# - those pixels that are not dark in the amplitude image (amp_bin) and
# - a 2D second order polynomial fit to the phase data
amp_bin = qpi_foc.amp > 1  # bright regions
qpi_foc.compute_bg(which_data="phase",
                   fit_profile="poly2o",
                   from_binary=amp_bin,
                   )

# plot results
plt.figure(figsize=(8, 3.5))
# ampltitude
ax1 = plt.subplot(121, title="amplitude")
map1 = plt.imshow(qpi_foc.amp, cmap="gray")
plt.colorbar(map1, ax=ax1, fraction=.0455, pad=0.04)
# phase in interval [-1rad, 1rad]
ax2 = plt.subplot(122, title="phase [rad]")
map2 = plt.imshow(qpi_foc.pha, vmin=-1, vmax=1)
plt.colorbar(map2, ax=ax2, fraction=.0455, pad=0.04)
# disable axes
[ax.axis("off") for ax in [ax1, ax2]]
plt.tight_layout()
plt.show()
```

3.2 HyperSpy hologram file format

This example demonstrates the import of hologram images in the HyperSpy hdf5 file format. The off-axis electron hologram shows an electrically biased Fe needle *[MLFDB15]*. The corresponding HyperSpy demo can be found here.



hyperspy_hologram.py

```
import urllib.request
import os

import matplotlib.pylab as plt
import qpformat

# load the experimental data
dl_loc = "https://github.com/hyperspy/hyperspy/raw/RELEASE_v1.3/" \
         + "hyperspy/misc/holography/example_signals/"
dl_name = "01_holo_Vbp_130V_0V_bin2_crop.hdf5"
if not os.path.exists(dl_name):
    urllib.request.urlretrieve(dl_loc + dl_name, dl_name)

ds = qpformat.load_data(dl_name,
                        holo_kw={
                            # reduces ringing artifacts in the amplitude image
                            "filter_name": "smooth disk",
                            # select correct sideband
                            "sideband": -1,
                            })

# retrieve the qpimage.QPImage instance
qpi = ds.get_qpimage(0)

# plot results
plt.figure(figsize=(8, 3.5))
# ampltitude
ax1 = plt.subplot(121, title="amplitude")
map1 = plt.imshow(qpi.amp, cmap="gray")
plt.colorbar(map1, ax=ax1, fraction=.0455, pad=0.04)
```

(continues on next page)

(continued from previous page)

```
# phase in interval [-1rad, 1rad]
ax2 = plt.subplot(122, title="phase [rad]")
map2 = plt.imshow(qpi.pha)
plt.colorbar(map2, ax=ax2, fraction=.0455, pad=0.04)
# disable axes
[ax.axis("off") for ax in [ax1, ax2]]
plt.tight_layout()
plt.show()
```

3.3 Conversion of external file formats to .npy files

Sometimes the data recorded are not in a file format supported by qpformat or it is not feasible to implement a reader class for a very unique data set. In this example, QPI data, stored as a tuple of files ("*_intensity.txt" and "*_phase.txt") with commas as decimal separators, are converted to the numpy file format which is supported by qpformat.

This example must be executed with a directory as an command line argument, i.e. `python convert_txt2npy.py /path/to/folder/`

`convert_txt2npy.py`

```
import pathlib
import sys

import numpy as np


def get_paths(folder):
    '''Return *_phase.txt files in `folder`'''
    folder = pathlib.Path(folder).resolve()
    files = folder.rglob("*_phase.txt")
    return sorted(files)


def load_file(path):
    '''Load a txt data file'''
    path = pathlib.Path(path)
    data = path.open().readlines()
    # remove comments and empty lines
    data = [l for l in data if len(l.strip()) and not l.startswith("#")]
    # determine data shape
    n = len(data)
    m = len(data[0].strip().split())
    res = np.zeros((n, m), dtype=np.dtype(float))
    # write data to array, replacing comma with point decimal separator
    for ii in range(n):
        res[ii] = np.array(data[ii].strip().replace(",", ".").split(),
                           dtype=float)
    return res


def load_field(path):
    '''Load QPI data using *_phase.txt files'''
    path = pathlib.Path(path)
    phase = load_file(path)
```

(continues on next page)

(continued from previous page)

```
    inten = load_file(path.parent / (path.name[:-10] + "_intensity.txt"))
    ampli = np.sqrt(inten)
    return ampli * np.exp(1j * phase)


if __name__ == "__main__":
    path = pathlib.Path(sys.argv[-1])
    if not path.is_dir():
        raise ValueError("Command line argument must be directory!")
    # output directory
    pout = path.parent / (path.name + "_npy")
    pout.mkdir(exist_ok=True)
    # get input *_phase.txt files
    files = get_paths(path)
    # conversion
    for ff in files:
        field = load_field(ff)
        np.save(str(pout / (ff.name[:-10] + ".npy")), field)
```

3.4 Conversion of external holograms to .tif files

Qpformat can load hologram data from .tif image files. If your experimental hologram data are stored in a different file format, you can either request its implementation in qpformat by **'creating an issue<https://github.com/RI-imaging/qpformat/issues/new>'_** or you can modify this example script to your needs.

This example must be executed with a directory as an command line argument, i.e. `python convert_txt2tif.py /path/to/folder/`

convert_txt2tif.py

```
import pathlib
import sys

import numpy as np
from skimage.external import tifffile

# File names ending with these strings are ignored
ignore_endswith = ['.bmp', '.npy', '.opj', '.png', '.pptx', '.py', '.svg',
                   '.tif', '.txt', '_RIdist', '_parameter', '_parameter_old',
                   '_phase', 'n_array', 'n_array_real', '~']
# uncomment this line to keep background hologram files
ignore_endswith += ['_bg']


def get_paths(folder, ignore_endswith=ignore_endswith):
    '''Return hologram file paths

    Parameters
    ----------
    folder: str or pathlib.Path
        Path to search folder
    ignore_endswith: list
        List of filename ending strings indicating which
        files should be ignored.
```

(continues on next page)

(continued from previous page)

```
    '''
    folder = pathlib.Path(folder).resolve()
    files = folder.rglob("*")
    for ie in ignore_endswith:
        files = [ff for ff in files if not ff.name.endswith(ie)]
    return sorted(files)


if __name__ == "__main__":
    path = pathlib.Path(sys.argv[-1])
    if not path.is_dir():
        raise ValueError("Command line argument must be directory!")
    # output directory
    pout = path.parent / (path.name + "_tif")
    pout.mkdir(exist_ok=True)
    # get input hologram files
    files = get_paths(path)
    # conversion
    for ff in files:
        # convert image data to uint8 (most image sensors)
        hol = np.loadtxt(str(ff), dtype=np.uint8)
        tifout = str(pout / (ff.name + ".tif"))
        # compress image data
        tifffile.imsave(tifout, hol, compress=9)
```

CHAPTER 4

Code reference

CHAPTER 5

Bilbliography

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Bibliography

[BBL+17] Manuel Bedrossian, Casey Barr, Chris A. Lindensmith, Kenneth Nealson, and Jay L. Nadeau. Quantifying microorganisms at low concentrations using digital holographic microscopy (DHM). *Journal of Visualized Experiments*, nov 2017. doi:10.3791/56343.

[MLFDB15] V. Migunov, A. London, M. Farle, and R. E. Dunin-Borkowski. Model-independent measurement of the charge density distribution along an fe atom probe needle using off-axis electron holography without mean inner potential effects. *Journal of Applied Physics*, 117(13):134301, apr 2015. doi:10.1063/1.4916609.

Index