XSVS_example

October 30, 2014

1 PYXSVS - python X-ray Speckle Visibility Spectroscopy data analysis tool

This notebook gives a description of the pyxsvs code, together with an example of usage. The code is available at GitHub: https://github.com/pawel-kw/pyxsvs

1.1 Introduction

X-ray Speckle Visibility Spectroscopy (XSVS) is a technique which allows to measure dynamics of a disordered sample from a series of coherent diffraction patterns (speckle patterns). The basic quantity measured is the visibility of speckles, quantified by the normalized variance of the intensity I in the speckle pattern, measured as a function of exposure time t_e

$$V_2(t_e) = \langle I^2 \rangle_{t_e} / \langle I \rangle_{t_e}^2 - 1 \tag{1}$$

The pyxsvs code calculates the variance assuming that $V_2(t_e) = 1/M$, where M is the number of coherent modes, derived from the recorded speckle pattern by fitting the intensity distribution $P(K)$ with the Poisson-Gamma (negative-binomial) model,

$$P(K) = \frac{\Gamma(K+M)}{\Gamma(K+1)\Gamma(M)} \left(\frac{\langle K \rangle}{\langle K \rangle + M} \right)^K \left(\frac{M}{\langle K \rangle + M} \right)^M, \tag{2}$$

K denoting the number of photons, and Γ - the gamma function.

1.1.1 Requirements

Running pyxsvs requires the following, non standard Python libraries to be installed:

1. scipy http://www.scipy.org/
2. fabio - I/O library for images produced by 2D X-ray, https://pypi.python.org/pypi/fabio
3. pyFAI - tool for fast azimuthal integration, http://www.esrf.eu/UsersAndScience/Publications/Highlights/2012/et/et3
4. lmfit - A library for least-squares minimization and data fitting, https://pypi.python.org/pypi/lmfit/

In principle, the code should be able to run on any operating system, but it was only tested on Archlinux.

1.2 Data

The code was written to handle data acquired with the single-chip MEDIPIX detector at ID10. Data from the 2x2 MEDIPIX could be also handled without any changes to the code, provided that the appropriate flat field and mask is given.

1. A complete XSVS data set is composed of several sub-sets of data acquired with different exposure times.
2. Each sub-set for a given exposure time contains exposures taken on different (but equivalent) scattering volumes, in order to avoid radiation damage (the sample was translated after a certain number of exposures).

1.2.1 Data description

A data set is described in an *.ini style input file, containing all the information needed for the calculations. The input file contains several sections. The [Main] section holds all the data common for the different exposures, like the result directory, data directory, flat field file, mask file, q partitioning description, experimental geometry description. The following sections are names [Exp_n], with n being an order number of the exposure. These sections contain exposure-specific metadata: file prefix and suffix, first and last file number and the exposure time. *Important*: the exposure time is not automatically deduced from the files. It has to be provided manually in the input file.

Below, a listing of an example input file.

```
In [1]: with open('./analysis/xsvs/xsvs_input.txt','r') as input_file:
            lines = input_file.readlines()

        for line in lines:
            print line
```

```
[Main]

save dir = ./

data dir = ../../data/xsvs-series/

flat field = ../maxipix1_flatfield_2013.edf

default mask = ../maxipix1_mask_2013.edf

mask = ../mask.edf

q1 = 0.002

qs = 0.001

q2 = 0.010

dq = 4e-04

sample name = SiD_1-1_40C_

wavelength = 1.53

cenx = 125.01

ceny = 118.31

pix = 0.055

sddist = 2140

figure title = SiD_1-1 40 degC



[Exp_1]
```

```
data prefix = sid1-1_40C_
data suffix = .edf.gz
first data file = 1
last data file = 360
exp time = 5e-5

[Exp_2]
data prefix = sid1-1_40C_
data suffix = .edf.gz
first data file = 361
last data file = 720
exp time = 1e-4

[Exp_3]
data prefix = sid1-1_40C_
data suffix = .edf.gz
first data file = 721
last data file = 1080
exp time = 2e-4

[Exp_4]
data prefix = sid1-1_40C_
data suffix = .edf.gz
first data file = 1081
last data file = 1440
exp time = 4e-4
```

```
[Exp_5]

data prefix = sid1-1_40C_

data suffix = .edf.gz

first data file = 1441

last data file = 1800

exp time = 8e-4


[Exp_6]

data prefix = sid1-1_40C_

data suffix = .edf.gz

first data file = 1801

last data file = 2160

exp time = 1.6e-3


[Exp_7]

data prefix = sid1-1_40C_

data suffix = .edf.gz

first data file = 2161

last data file = 2520

exp time = 3.2e-3


[Exp_8]

data prefix = sid1-1_40C_

data suffix = .edf.gz

first data file = 2521

last data file = 2700

exp time = 6.4e-3
```

The input file is created manually. In the case of multiple, similarly looking data sets, I usually write a Python script creating the input files for all the data sets.

1.3 Data analysis

The pyxsvs code can be executed as a script, with the input file name given as an input parameter:

```
> python2 pyxsvs.py -i ./xsvs_input.txt
```

The script works as follows.

1. The input file name is parsed and passed as an argument to the pyxsvs object constructor.
2. An object of pyxsvs class is created. This includes calculating an averaged 2D SAXS image.
3. Direct beam position is determined
4. Speckle visibility is calculated:
   a) For each exposure, each data frame is divided into q partitions.
   b) Histograms are calculated for each of these partitons and averaged over all frames.
   c) The resulting histograms are fitted with the negative-binomial distribution function and the number of modes is determined

The results are saved in a binary data file (pickle). A series of figures is also produced.

1.3.1 Example

The following example executes the same commands which are executed by calling the pyxsvs script

```
In [2]: import sys
```

```
In [3]: sys.path.append('/home/kwasniew/Tools/Python/pyxsvs/pyxsvs-src/')
        import pyxsvs
```

```
ERROR:pyFAI.opencl:Unable to import pyOpenCl. Please install it from: http://pypi.python.org/pypi/pyoper
ERROR:pyFAI.azimuthalIntegrator:Unable to import pyFAI.ocl_azim
ERROR:pyFAI.azimuthalIntegrator:Unable to import pyFAI.ocl_azim_lut for Look-up table based azimuthal in
```

```
In [4]: cd ./analysis/xsvs
```

```
/home/kwasniew/Tools/Python/pyxsvs/examples/analysis/xsvs
```

```
In [5]: calculator = pyxsvs.pyxsvs('./xsvs_input.txt',useFlatField=True)
```

```
In [6]: calculator.findDB()
```

```
[[Variables]]
     xi:     125.009901 +/- 0.030955 (0.02%) initial =  125.010000
     yi:     118.309403 +/- 0.068321 (0.06%) initial =  118.310000
[[Correlations]] (unreported correlations are <  0.100)
    C(xi, yi)                    = -0.632
```

```
In [7]: calculator.calculateVisibility()
```

```
20
Histograming Exp_1
99%
Calculations took 1.53 s
Done for Exp_1, now plotting and fitting...
20
Histograming Exp_2
99%
Calculations took 1.52 s
Done for Exp_2, now plotting and fitting...
20
Histograming Exp_3
99%
Calculations took 1.64 s
Done for Exp_3, now plotting and fitting...
20
Histograming Exp_4
99%
Calculations took 1.71 s
Done for Exp_4, now plotting and fitting...
20
Histograming Exp_5
99%
Calculations took 2.22 s
Done for Exp_5, now plotting and fitting...
20
Histograming Exp_6
99%
Calculations took 1.94 s
Done for Exp_6, now plotting and fitting...
20
Histograming Exp_7
99%
Calculations took 1.98 s
Done for Exp_7, now plotting and fitting...
20
Histograming Exp_8
99%
Calculations took 0.96 s
Done for Exp_8, now plotting and fitting...

/home/kwasniew/Tools/Python/pyxsvs/pyxsvs-src/pyxsvs.py:628: RuntimeWarning: invalid value encountered
  numpy.logErr = yerr/(ydata*numpy.log(10))
```

The execution often spits out some warnings about zeros appearing in log or divisions. Normally this is not a problem.

1.3.2 Reporting results

The script produces a lot of figures and a binary pickle file containing all the data. To get an overview of the fit resutls I wrote another Python script, which collects all the figures produced by pyxsvs and puts them together into a single PDF repot. The pickle result file can be easily used for further data fitting and reporting. Pickle is not an optimal data storage format (see https://wiki.python.org/moin/UsingPickle). Eventually, pyxsvs should save the results into something more secure and readable, like good, old CSV type ASCII file or an hdf5.

Data overview In order to use the reportResults.py script, in addition to pyxsvs requirements you need to have ReportLab, an engine for creating complex, data-driven PDF documents. An opensource version can be obtained from http://www.reportlab.com/opensource/ Then, all you need to do is to run:

```
> python2 reportResults.py -i ./xsvs_input.txt
```

The PDF gives a good overview, but real data analysis requires looking into the pickle data file.

Data plotting and fitting Pickle is a standard Python library. The results can be loaded from the file in a very simple way.

```
In [9]: from pylab import *
        import pickle
        fname = './SiD_1-1_40C_sid1-1_40C_results.p'
        with open(fname,'rb') as handle:
            res_data = pickle.load(handle)
```

The data container is a multi-level dictionary with exposure labels as keys:

```
In [10]: res_data.keys()

Out[10]: ['Exp_5', 'Exp_4', 'Exp_7', 'Exp_6', 'Exp_1', 'Exp_3', 'Exp_2', 'Exp_8']
```

For each exposure, the value containes another dictionary:

```
In [11]: res_data['Exp_1'].keys()

Out[11]: ['data', 'expTime']
```

One of them stores the exposure time value in [s]

```
In [12]: res_data['Exp_1']['expTime']

Out[12]: 5e-05
```

The other is the data container, with data grouped by the scattering vector partitions:

```
In [13]: res_data['Exp_1']['data'].keys()

Out[13]: ['q008', 'q007', 'q006', 'q005', 'q004', 'q003', 'q002', 'q001', 'q000']

In [14]: res_data['Exp_1']['data']['q000'].keys()

Out[14]: ['R-1',
          'histStdDev',
          'trace',
          'K',
          'M',
          'histogram',
          'q',
          'histogramBins',
          'beta']
```

- 'R-1': An estimate of contrast β based on the observed probabilities of 1 and 2 photon events: $R = 2P(2)[1-P(1)]/P(1)^2 = 1+\beta$. This should work well for small average photon counts $\langle K \rangle$.
- 'histogramBins': number of photons K
- 'histogram': probability value $P(K)$ of observing K photons

- 'histStdDev': standard deviation from the mean of $P(K)$, calculated using all the frames for the given exposure time
- 'K': average number of photons for the given q partition
- 'M': the fitted number of modes contributing to the speckle pattern
- 'beta': $\beta = 1/M$, speckle contrast
- 'q': the value of the momentum transfer vector q in $[\text{A}^{-1}]$
- 'trace': average number of photons in the given q partition for each frame

Below, an example showing how the results can be plotted.

In [18]:
```
import itertools
from pylab import *
import pickle
sys.path.append('/home/kwasniew/Tools/Python/pyxsvs/pyxsvs-src/')
import pyxsvs
%pylab inline

mcolors = itertools.cycle(['b', 'g', 'r', 'c', 'm', 'y', 'k'])
markers = itertools.cycle(list(Line2D.filled_markers))
lstyles = itertools.cycle(['-', '--', '-.', ':'])


def plot_xsvs_results(res_data,ax1,ax2,**kwargs):
    '''Plotting results of the pyxsvs code
    '''
    exp_list = sort(res_data.keys()) # Get a sorted list of exposures
    q_list = sort(res_data[exp_list[0]]['data'].keys()) # Sorted list of qs
    for j in xrange(len(q_list[:])):
        beta_data = zeros((len(exp_list),3)) # Container for contrast values
        for i in xrange(len(exp_list)):
            marker = markers.next()
            color = mcolors.next()
            histogramBins = res_data[exp_list[i]]['data'][q_list[j]]['histogramBins']
            hist_data = res_data[exp_list[i]]['data'][q_list[j]]['histogram']
            K_ave = res_data[exp_list[i]]['data'][q_list[j]]['K']['value']
            # Plot the histogram for the first q value for all exposures
            if j == 0:
                # First, get the q value
                q_val = res_data[exp_list[i]]['data'][q_list[j]]['q']['value']
                q_txt = r'$q = %.2f \times 10^{-2} \; \AA^{-1}$' % (q_val*1e2)
                # Get the exposure time
                exp_val = res_data[exp_list[i]]['expTime']
                ax1.set_title(q_txt)
                # Plot the fitted negative binomial distribution
                # The fitted M value
                M = res_data[exp_list[i]]['data'][q_list[j]]['M']['value']
                xx = arange(0,100,0.1) # generating a dense mesh for the fitted curve
                # evaluate the negative binomial distribution
                neg_binom = pyxsvs.nbinomPMF(xx,K_ave,M)
                # Plot the data and the fitted curve
                ax1.plot(histogramBins[:-1]/K_ave,hist_data[:-1],'o',ms=3,
                        marker=marker,color=color,label=r'%.2f ms' % (exp_val*1e3))
                ax1.plot(xx/K_ave,neg_binom,'-',color=color)
            # Extract the contrast as a function of exposure for each q value
```

```
                beta_data[i,0] = res_data[exp_list[i]]['expTime']
                beta_data[i,1] = res_data[exp_list[i]]['data'][q_list[j]]['beta']['value']
                beta_data[i,2] = res_data[exp_list[i]]['data'][q_list[j]]['beta']['stddev']
            sep_par = j*0.1 # additive factor to separate the data sets in the plot
            qval = res_data[exp_list[i]]['data'][q_list[j]]['q']['value']
            # Plot the contrast as a function of exposure time for all q values
            ax2.errorbar(beta_data[:,0],beta_data[:,1]+sep_par,beta_data[:,2],
                    color=color,marker=marker,
                    label=r'$%.1f$' % (qval*1e2),**kwargs)
        # Adjusting the plots
        ax2.set_xscale('log')
        ax2.set_xlim(4e-5,1e-2)
        ax2.set_ylim(0,1.0)
        ax2l = ax2.legend(frameon=0,bbox_to_anchor=(1.3,1.0),
                        title=r'$q \times 10^{-2}$ [$\AA^{-1}$]')
        ax2l.get_title().set_fontsize('10')
        ax2.set_xlabel(r'$t_e$ [s]')
        ax2.set_ylabel(r'$1/M$')
        ax1.set_xlim(-1,20)
        ax1.set_ylim(1e-4,5)
        ax1.set_yscale('log')
        ax1.legend(frameon=0,bbox_to_anchor=(1.2,1),title=r'$t_e$')
        ax1.set_xlabel(r'$K/ \langle K \rangle$')
        ax1.set_ylabel(r'$P(K)$')
```

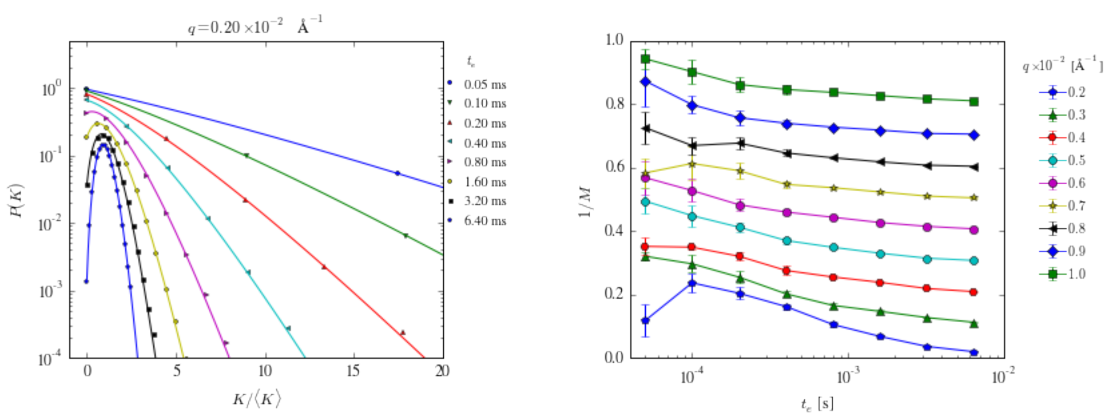
```
Populating the interactive namespace from numpy and matplotlib

WARNING: pylab import has clobbered these variables: ['power', 'draw_if_interactive', 'random', 'fft', '
`%matplotlib` prevents importing * from pylab and numpy
```

```
In [17]: results_file = './SiD_1-1_40C_sid1-1_40C_results.p'

         # Prepare the figure
         fig1 = figure(figsize=(12,4))
         ax1 = fig1.add_subplot(121)
         ax2 = fig1.add_subplot(122)
         subplots_adjust(wspace=0.5)

         # Plot the results
         with open(results_file,'rb') as handle:
             res_data = pickle.load(handle)
         plot_xsvs_results(res_data,ax1,ax2)
```



In [ ]: