Using pyXS to process protein solution scattering data

Lin Yang
Jan, 2011

What is pyXS

pyXS is a set of python scripts that implement functions similar to those available in view.gtk. It offers more flexibility since these codes can be utilized in python code written by users to process the data or control the experiment.

This documentation describes how to use the pyXS package to process solution scattering data. While data collected on the SAXS and WAXS detectors will be used as examples, the same procedures should also apply for data collected elsewhere, as long as the image files that contain the scattering patterns are readable by the python image library.

Availability and OS Compatibility

The pyXS package can be downloaded from the website of beamline X9 at NSLS: http://x9dev.nsls.bnl.gov/software/pyXS.htm . Since pyXS is based on python, in principle it can run under all platforms that support python.

Installation

The pyXS package depends on the following software packages: python, numpy, python image library (PIL), matplotlib. A portion of the code (RQconv) is written in C. Therefore you will need to compile this module if the package from the web link above does not contain a binary suitable for your OS. Doing so will also require the SWIG software.

If you use Linux, you should use the package manager to install these required software packages. If you use Mac, you can install these packages using fink (http://www.finkproject.org/). Alternatively, you may be able to use the native python packages that come OS X (see discussion here: http://www.python.org/download/mac/). But I have not tested this. And the fink works fine for me. If you use Windows, you may have to download these packages individually, and make sure that the python executables are included in your PATH variable. The package from X9 web site includes a WIN32 binary compiled using MinGW.

The web links for the require packages are listed below:

Python: http://www.python.org/download/
Numpy: http://www.scipy.org/Download
Matplotlib: http://matplotlib.sourceforge.net/
PIL: http://www.pythonware.com/products/pil/
(NOTE: An older version of PIL has a bug and cannot read some TIFF files correctly. Make sure you have the most recent version the X9 WAXS data you see do not make sense.)
SWIG: http://www.swig.org/

Overview of pyXS

The pyXS package include the following modules: Data2D.py, RQconv.py, and slnXS.py.

The Data2D module reads (PIL) and displays (matplotlib) 2D scattering patterns and perform the conversion between pixel position and receiprocal space coordinates, q or (q_r, q_z) (see more descriptions of X-ray scattering in the documentation of view.gtk). Data2D relies on C code RQconv.c to convert data, based on scattering geometry defined by the structure ExpPara. The line profile extraction and annotation functions of view.gtk can be similarly accomplished using functions defined in this module.

The slnXS module is used to process 1D solution scattering data produced by Data2D. It can average from multiple scattering patterns, perform background subtraction (dark current and buffer scattering) based on transmitted beam intensity, and combine the SAXS and WAXS data simutaneously collected on the two detectors at X9. This module also provides rudimentary capabilities for producing Gunier plots and for approximate p(r) function calculations.

Setting up for data processing

For data conversion to work properly, the parameters that describe the scattering geometry must be defined. At X9, these parameters are usually defined in a file named exp_setup.py, produced by the beamline staff before user experiments. You can simply import it into your python codes that utlize pyXS. An example of the exp_setup.py file is shown in the appendix.

In exp_setup.py, the global variables that affect the behavior of pyXS are first defined:

```
slnXS.DETECTORS_CONFIG=slnXS.DET_SAXSWAXS
# The options are DET_SAXSWAXS and DET_SAXSONLY. pyXS will not look for the second
```

```
# set of data (WAXS) is DETECTOR_CONFIG=SAXSONLY .

slnXS.PROTEIN_SOLUTION_SCATTERING=True
# If True, the protein volume fraction is calculated based on the concentration
# and an assumed protein density and used for buffer scattering subtraction.

slnXS.BEAM_SIZE_hW=5
slnXS.BEAM_SIZE_hH=4
# These are useful for the data collected at X9 only. The beam stop at X9 is
# semi-transparent to X-rays. Therefore each SAXS pattern also contain the transmitted
# beam intensity, which is obtained by integrating the intensity within the rectangle
# around the direct beam and with the half-width and half-height defined here.
```

The scattering geometry is then established for the SAXS and WAXS detectors. If only one detector is used (DETECTOR_CONFIG=SAXSONLY), only one ExpPara needs to be defined. In order to define the parameters in ExpPara, the scattering pattern from a standard powder sample is measured. The simulated powder rings based on these parameters are then compared to the measured standard pattern. The parameter are adjusted to best match the two. This process can be done interactively suing view.gtk (see the documentation of view.gtk). An automatic refinement routine is planned but not yet implemented.

```
es = ExpPara()                # create a new ExpPara
es.wavelength = 0.886         # X-ray wavelength
es.bm_ctr_x = 425             # pixel position of the X-ray beam
es.bm_ctr_y = 480             #
es.ratioDw = 49.8             # this is the sample-to-detector distance divided by
                              #       the width of the detector
es.det_orient = 0             # orientation of the detector
es.det_tilt = 0                       #       0, 0, 0 for the SAXS detector
es.det_phi = 0                #
es.grazing_incident = False   # always false for solution scattering
es.incident_angle = 0.2       # only meaningful for grazing incident scattering
es.sample_normal = 0          # only meaningful for grazing incident scattering
```

Next the *q*-grids for the 1D SAXS and WAXS scattering profile are defined. Note that the grids do not have to be evenly spaced.

```
qsaxs = np.hstack((np.arange(0.004,0.150,0.001),np.arange(0.152,0.187,0.002)))
```

The masks are also defined to block off parts of the scattering pattern that should not be included in the conversion from the 2D scattering patterns to 1D scattering profiles.

```
msaxs = Mask(487,619)
# This should match the size of the 2D scattering pattern. Otherwise you will enconter
# an error when converting the 2D data.

msaxs.read_file("mask.SAXS")
```

The format of the mask file is similar to that used in view.gtk, except that polygons can also be used. Here is an example of the mask file (used for the WAXS data collected at X9):

```
# A mask file should contain geometric shapes, each defined on one line.
# "h" defines a "hole", i.e. inverse of the circle, with pixels outside of the cicle
# blocked off. The parameters are the center (x,y) and the radius.
h       518     518     645
# "p" defines a polygon. The parameters define the vertices (x,y) of the polygon.
p       0       0       0       250     250     0       0       0
p       750     0       1042    0       1042    292     750     0
p       750     1042    1042    1042    1042    750     750     1042
# "r" defines a rectangle. The parameters are the center (x,y), width and height, and the
# rotation of the rectangle about its center.
r       536     521     4       1042    0
r       197     521     1       1042    0
r       518     521     1       1042    0
r       791     521     1       1042    0
```

Finally, the dark current from the detectors are defined and will be used for background subtraction. The dark current data are averaged from several 2D images and saved in a pickled file. The pickled file will be read if already exist when exp_setup.py is imported. Otherwise is will be created from the 2D images specified. Note that the exposure time for the dark current data should equal that for the actual data.

In the event that you suspect that the exp_setup.py file provided by the beamline staff is incorrect, you can compare the simulated powder rings to the standard patterns using view.gtk or the disp.py script in the appendix of this document and revise the parameters or the mask accordingly.

Data processing

Data processing

1. Conversion of 2D data into 1D scattering profiles

Conversion of the 2D images into 1D scattering profiles can be accomplished in one single line by calling Data1d.avg(). pyXS permits several 2D images to be averged together to produce the 1D profile. The first argument therefore can be a string of multiple file names separated by spaces. The 1D profile corresponding to each image as well as the average can be plotted, if plot_data=True, so that the user can decide any of them should be discarded (e.g. because sample ran out during the exposure.) The averaged 1D profile for each individual 2D image can also be saved if save_ave=True. The rest of arguments when calling Data1d.avg() are usually defined in exp_setup.py.

```
dsamp.avg(sys.argv[1].split(),es,ew,msaxs,mwaxs,plot_data=False,save_ave=True)
```

```
dbuf.avg(sys.argv[2].split(),es,ew,msaxs,mwaxs,plot_data=False,save_ave=True)
dempty.avg(sys.argv[3].split(),es,ew,msaxs,mwaxs,plot_data=False,save_ave=True)
```

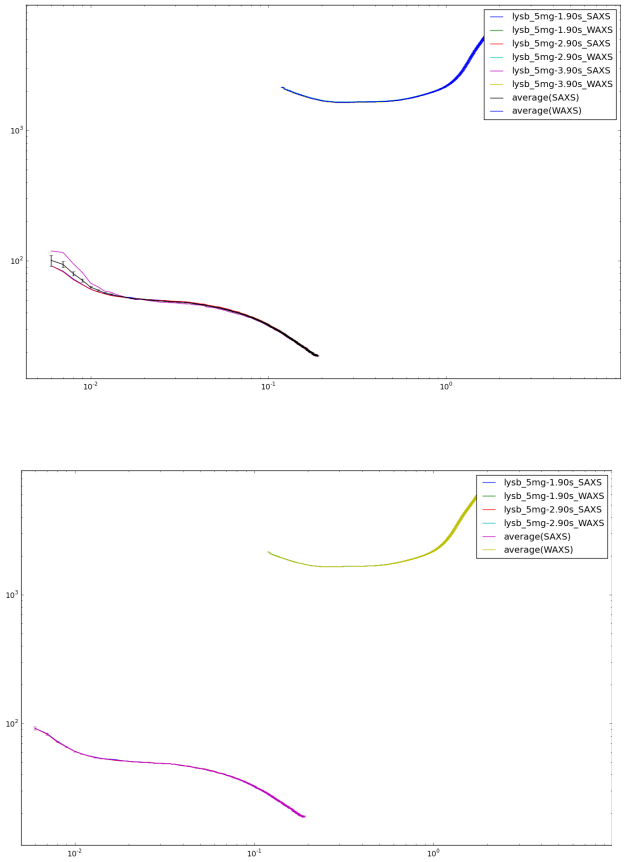

Fig.1 Plots generated by Data1d.avg(). It is clear from the plot on the left that one of the scattering profiles is significantly different from the other two at low q. This set of data is therefore discarded. The reason for the difference is likely streaking around the beamstop.

2. Background subtraction

Subtraction of dark current is done implicitly. The user only needs to specify which is the buffer scattering data to be subtracted (see proc.py in the Appendix). It is also possible to subtract the empty cell scattering (see proc-empty.py in the Appendix). In this case the empty cell scattering is first subtracted by calling Data1d.cor(). The buffer scattering, which is also empty cell-subtracted, can then be subtracted using Data1d.cor2(). When subtracting one set of data from another, the two sets of data are first normalized for transmitted beam intensity, which can be either obtained by integrating the intensity within the direct beam stop in the SAXS scattering pattern (default), or explicitly specified by calling Data1d.set_trans().

If slnXS.PROTEIN_SOLUTION_SCATTERING=True, pyXS will take into account of the volume fraction of the protein when subtracting buffer scattering. This volume is calculated using the protein concentration supplied when calling Data1d.cor() or Data1d.cor2(), and the assumed protein density, PROTEIN_WATER_DENSITY_RATIO = 1.35, which is defined slnXS.py. Omitting the conc parameter when calling Data1d.cor() or Data1d.cor2() turns off the volume fraction correction (zero concentration and therefore zero volume fraction).

From proc.py:

```
# if plot_data=True, a plot of the data, the background, and the background-subtracted
data
# will be plotted
dsamp.cor(dbuf,ddark,conc=float(sys.argv[3]),plot_data=True)
```

From proc-empty.py:

```
# subtract empty cell scattering first
dsamp.cor(dempty,ddark,plot_data=True)
dbuf.cor(dempty,ddark,plot_data=True)
# now subtract buffer scattering from sample scattering
dsamp.cor2(dbuf,conc=float(sys.argv[4]))
```

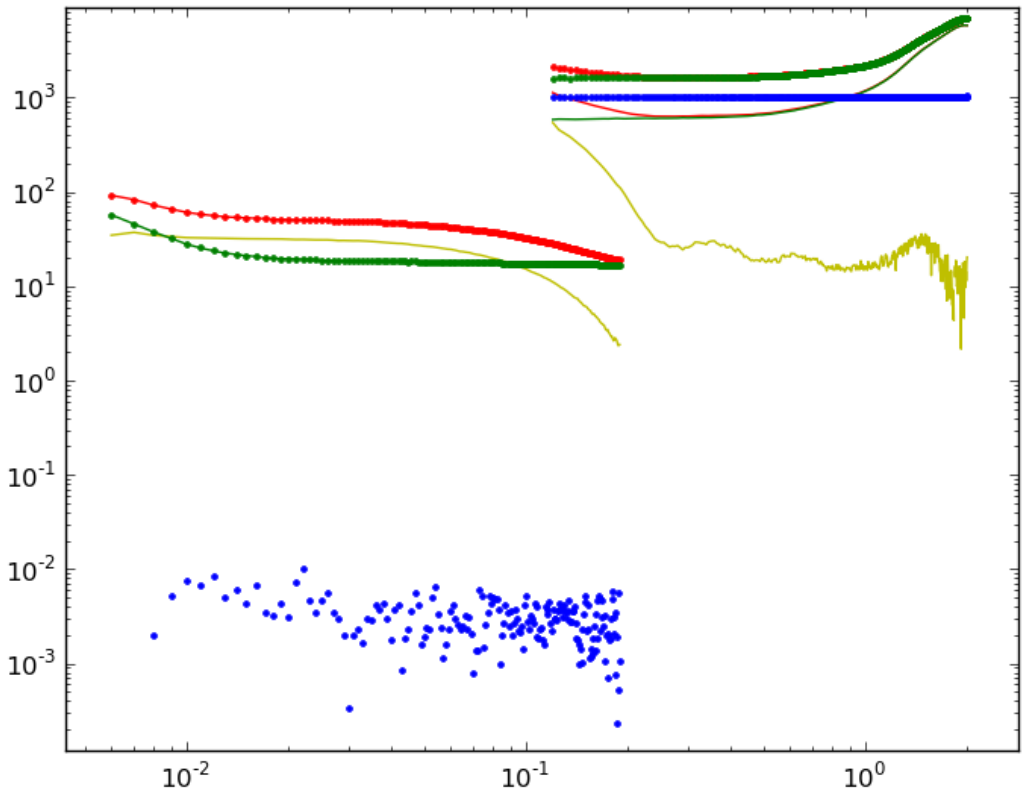

Fig.2 A plot generated by Data1d.cor(). The data (red), background (green), dark current (blue) and

the result of background subtraction (yellow) are shown for both SAXS and WAXS data.

3. Merging SAXS/WAXS data

Once the SAXS and WAXS data are background-subtracted, separately, they can be merged to produce a single scattering profile. The SAXS and WAXS data must be scaled relative to each other because of the differences in the solid angle that each pixel opens to as well and in the sensitivity of the detector. When there is a substantial overlap between the *q*-ranges of the SAXS and WAXS data, it is possible simply optimize the scaling factor so that the two sets of data coincide within the overlapping *q*-range. This is accomplished by calling Data1d.join_swaxs(), which arguments that specify the *q*-range within which the best-fit scaling factor should be obtained.

```
dsamp.join_swaxs(qmin=0.122,qmax=0.178)
dsamp.save(sys.argv[4],save_SW=1)
dsamp.plot_swaxs()
```

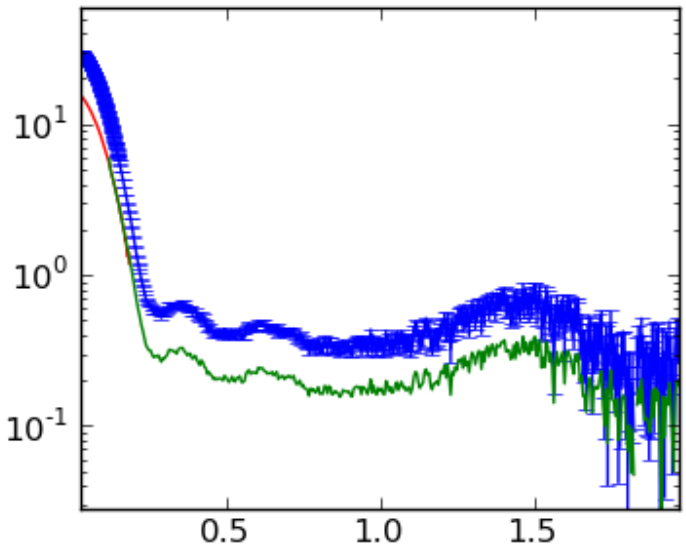

Fig.3 A plot generated by Data1d.plot_swaxs(). The SAXS (red) and WAXS (green) scattering profiles are background-subtracted separately. The WAXS data is then scaled to match the SAXS data in the *q*-range specified in Data1d.join_swaxs(). In the merged data (blue), half of the SAXS data and half of the WAXS data are retained in this overlapping *q*-range.

4. Gunier fit

A Gunier plot can be generated using Data1d.plot_Guinier(). The Guinier fit is performed within the specified *q*-range. If fix_qe=False, pyXS will adjust the ending

point of the q-range until it is below $1/R_g$. A call to Data1d.plot_Gunier() returns the best-fit I_0 and R_g values.

```
(I0, Rg) = dsamp.plot_Guinier(qs=0.02,qe=0.05,fix_qe=False)
```

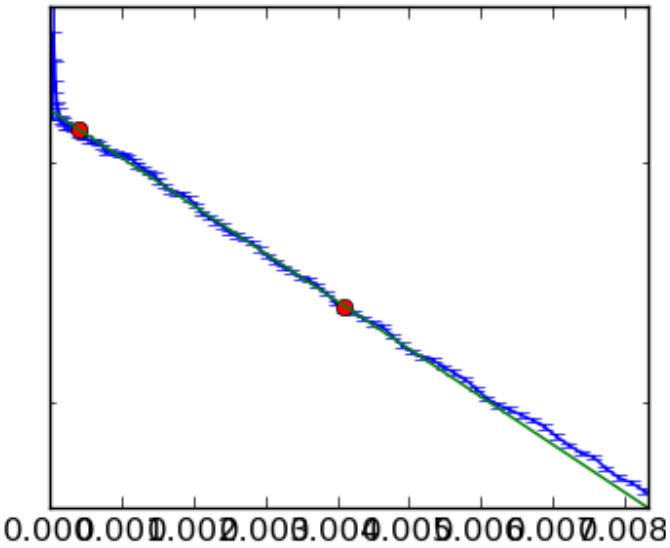

Fig.4 The plot generated by Data1d.plot_Guinier(). The fit is performed within the range specified by the qs (start) and qe (end) arguments of Data1d.join_swaxs(), shown as the red dots.

Appendix: example codes

1. exp_setup.py

```
import numpy as np
import pickle

PYXS_PATH='/Users/lyang/pro/pyXS'
import sys, os
PYXS_PATH in sys.path or sys.path.append(PYXS_PATH)

import slnXS
slnXS.DETECTORS_CONFIG=slnXS.DET_SAXSWAXS
slnXS.PROTEIN_SOLUTION_SCATTERING=True
slnXS.BEAM_SIZE_hW=5
slnXS.BEAM_SIZE_hH=4

from Data2D import *
from RQconv import *
from slnXS import *
import matplotlib.pyplot as plt

es = ExpPara()
es.wavelength = 0.886
es.bm_ctr_x = 425
es.bm_ctr_y = 480
es.ratioDw = 49.8
```

```
es.det_orient = 0
es.det_tilt = 0
es.det_phi = 0
es.grazing_incident = False
es.incident_angle = 0.2
es.sample_normal = 0

ew = ExpPara()
ew.wavelength = 0.886
ew.bm_ctr_x = 3
ew.bm_ctr_y = 984
ew.ratioDw = 2.05
ew.det_orient = 45
ew.det_tilt = 18
ew.det_phi = 0
ew.grazing_incident = False
ew.incident_angle = 0
ew.sample_normal = 0

qsaxs = np.hstack((np.arange(0.004,0.150,0.001),np.arange(0.152,0.187,0.002)))
msaxs = Mask(487,619)
msaxs.read_file("mask.SAXS")

qwaxs = np.hstack((np.arange(0.164,0.180,0.002),np.arange(0.185,4.30,0.005)))
mwaxs = Mask(1042,1042)
mwaxs.read_file("mask.WAXS")

dsamp = Data1d(qsaxs,qwaxs)
dbuf = Data1d(qsaxs,qwaxs)
dempty = Data1d(qsaxs,qwaxs)

DARK_FILE="dark-90s.pkl"
if os.path.isfile(DARK_FILE):
    pkl_file = open(DARK_FILE, 'rb')
    ddark = pickle.load(pkl_file)
    pkl_file.close()
else:
    ddark = Data1d(qsaxs,qwaxs)
    # change the list of files below for each unique exposure time
ddark.avg(["dark-1.90s",
               "dark-2.90s",
               "dark-3.90s",
               "dark-4.90s",
               "dark-5.90s",
               "dark-6.90s"],
               es,ew,msaxs,mwaxs,
               plot_data=False,
               save_ave=True)
    pkl_file = open(DARK_FILE, 'wb')
    pickle.dump(ddark, pkl_file)
    pkl_file.close()
```

2. disp.py

```
#!/sw/bin/python2.6

from exp_setup import *
import matplotlib as mpl
import sys

if (len(sys.argv) < 2):
    print "Usage: disp.py file_name_root"
    exit()
else:

    saxs_current_file = sys.argv[1]+'_SAXS'
    waxs_current_file = sys.argv[1]+'_WAXS'

    plt.figure(figsize=(10,5))
```

```
    plt.subplot(121)
    ax = plt.gca()
    dsaxs = Data2d(saxs_current_file)
    dsaxs.set_exp_para(es)
    pax = Axes2dplot(ax,dsaxs)
    pax.plot(mask=msaxs)
    pax.set_color_scale(mpl.cm.get_cmap('jet'),3.5)
    pax.img.set_clim(-10,600)

    if (len(sys.argv)>2):
        pax.add_dec("Q 0.1076 72 r-")
        pax.add_dec("Q 0.2152 72 r-")
        pax.add_dec("Q 0.3228 72 r-")
        pax.add_dec("Q 0.4304 72 r-")

    plt.subplot(122)
    ax = plt.gca()
    dwaxs = Data2d(waxs_current_file)
    dwaxs.set_exp_para(ew)
    pax = Axes2dplot(ax,dwaxs)
    pax.plot(mask=mwaxs)
    pax.set_color_scale(mpl.cm.get_cmap('jet'),3.5)
    pax.img.set_clim(1000,2400)

    # these are simulated rings corresponding to the structure of silver behenates
    if (len(sys.argv)>2):
        pax.add_dec("Q 0.1076 72 r-")
        pax.add_dec("Q 0.2152 72 r-")
        pax.add_dec("Q 0.3228 72 r-")
        pax.add_dec("Q 0.4304 72 r-")
        pax.add_dec("Q 0.5380 72 r-")
        pax.add_dec("Q 0.6456 72 r-")
        pax.add_dec("Q 0.7532 72 r-")
        pax.add_dec("Q 0.8608 72 r-")
        pax.add_dec("Q 0.9684 72 r-")
        pax.add_dec("Q 1.0760 72 r-")
        pax.add_dec("Q 1.37 72 r-")

    plt.show()
```

3. proc.py

```
#!/sw/bin/python2.6
from exp_setup import *

import sys

if len(sys.argv)<3:
    print "Usage: proc.py sample buf protein_conc out_put_file"
    print "multiple files can be given for sample and buf"
    print "e.g. ./proc.py \"lys20 file2\" lysbuf4 3.7 lyso20.dat"
    exit()
else:
    # set plot_data=True to see curves from the individual files
    # a label can be given to the averaged curve
    dsamp.avg(sys.argv[1].split(),es,ew,msaxs,mwaxs,plot_data=False,save_ave=True)
    dbuf.avg(sys.argv[2].split(),es,ew,msaxs,mwaxs,plot_data=False,save_ave=True)
    dsamp.cor(dbuf,ddark,conc=float(sys.argv[3]),plot_data=True)

    plt.figure(figsize=(18,6))
    plt.subplot(131)
    # parameters can be given to adjust the q-range within which the
    # best fit is obtained to join SAXS/WAXS data
    # e.g. join_swaxs(qmin=0.12,qmax=0.2)
    dsamp.join_swaxs(qmin=0.13,qmax=0.2)
    dsamp.save(sys.argv[4])
    dsamp.plot_swaxs()

    plt.subplot(132)
```

```
    # the starting (qs) and ending (qe) q can be given for the Gunier fit
    # if fix_qe=False (default), the code will adjust qe automatically so
    # that Rg*qe<1 .
    (I0, Rg) = dsamp.plot_Guinier(qs=0.02,qe=0.08,fix_qe=True)
    print "I0=%f, Rg=%f" % (I0, Rg)

    plt.subplot(133)
    # qmax is the highest q of the joined SAXS/WAXS data to include in the
    # calculation. dmax is the highest r in the calculated p(r) funtion
    dsamp.plot_pr(I0,Rg,qmax=.9,dmax=200.)

    plt.show()
```

4. proc-empty.py (includes subtraction of empty cell scattering).

```
#!/sw/bin/python2.6
from exp_setup import *

import sys

if len(sys.argv)<4:
    print "Usage: proc.py sample buf empty protein_conc out_put_file"
    print "multiple files can be given for sample and buf"
    print "e.g. ./proc.py \"lys20 file2\" lysbuf4 empty 3.7 lyso20.dat"
    exit()
else:
    # set plot_data=True to see curves from the individual files
    # a label can be given to the averaged curve
    dsamp.avg(sys.argv[1].split(),es,ew,msaxs,mwaxs,plot_data=False,save_ave=True)
    dbuf.avg(sys.argv[2].split(),es,ew,msaxs,mwaxs,plot_data=False,save_ave=True)
    dempty.avg(sys.argv[3].split(),es,ew,msaxs,mwaxs,plot_data=False,save_ave=True)

    dsamp.cor(dempty,ddark,plot_data=True)
    dbuf.cor(dempty,ddark,plot_data=True)

    dsamp.cor2(dbuf,conc=float(sys.argv[4]))

    plt.figure()
    # parameters can be given to adjust the q-range within which the
    # best fit is obtained to join SAXS/WAXS data
    # e.g. join_swaxs(qmin=0.12,qmax=0.2)
    dsamp.join_swaxs(qmin=0.122,qmax=0.178)
    dsamp.save(sys.argv[5])
    dsamp.plot_swaxs()
    plt.show()
```