Status report on inclusion of SESANS data analysis in SASVIEW during Sasview Code Camp III (February 2015)

Jurrian Bakker (j.h.bakker@tudelft.nl)
Wim Bouwman (w.g.bouwman@tudelft.nl)
all participants of the code camp 2015

October 6, 2016

1 The aim of the code camp

The aim of this work package is to add the ability to analyse SESANS data using SasView. SasView is a software package designed to analyze scattering data by graphical visualization and model fitting in reciprocal (Fourier) space. We will extend this ability by allowing fitting in real space using Hankel transformation of existing SANS models and direct programming of real space scattering models.

The long-term goal is to enable co-analysis of SANS and SESANS data. SESANS can be used to increase the accuracy of low-Q data in a SANS experiment; conversely, SANS data can be used to decrease experimental error at low length scales in a SESANS experiment. If structural models could be fitted in both real space and Fourier space simultaneously, SANS and SESANS data would provide structural information at complementary length scales.

The work was divided into several stages:

1. Definition of data format (Section 2).
2. Enable dataloader package to read SESANS data and represent it in a suitable structure.
3. Wrap SANS models into a SESANS data analysis wrapper to calculate the projected correlation functions numerically from model parameters.
4. Combine the dataloader and SESANS data analysis wrapper with BUMPS in fitting scripts.
5. Enable fitting of SESANS data through SasView GUI

The first four tasks have been completed to some extent: a data format is proposed and a dataloader was created for this format (Section 2). Further scripts will need to be written to convert instrument output data into this data format. SESANS wrapper is able to combine with BUMPS to fit SESANS data read in by the dataloader with some tweaks (Section 6). It was decided that inclusion of SESANS analysis into the Sasview GUI would be too early at this stage, since the Sasview GUI will undergo heavy rewriting to clean up the code and create a more user-friendly interface.

So in conclusion: we have now the possibility to use the models available in sasmodels to analyse SESANS-measurements in the command line in a developers environment. It is good for the experts, but not yet for users.

2 Data format

The current file extension is .ses or .sesans (not case sensitive).

Our preliminary suggestion for the file format is to have a list of name-value pairs as a header at the top of the file, detailing general experimental parameters necessary for fitting and analyzing data. This list should contain all information necessary for the file to be 'portable' between users.

Following that is a 6 column list of instrument experimental variables:

- Spin echo length (δ, in Angstroms)
- Spin echo length error ($\Delta\delta$, in Angstroms) (experimental resolution)
- neutron wavelength (λ, in Angstroms) (essential for ToF instruments)
- neutron wavelength error ($\Delta\lambda$, in Angstroms)
- Normalized polarization (P/P_0, unitless)
- Normalized polarization error ($\Delta(P/P_0)$, unitless) (measurement error)

The neutron wavelength appears in both scattering power (Σ) and spin echo length (δ), making data analysis more complicated for Time-of-Flight SESANS than for Monochromatic SESANS.

$$\Sigma = \lambda^2 t (\Delta\rho_0)^2 \phi(1-\phi)\xi \tag{1}$$

$$\delta = \frac{c\lambda^2 BL}{2\pi \tan\theta_0} \tag{2}$$

After discussion with partners (DUT/ISIS/ESS/ORNL, etc.), we might add more items to the header. Extending the header will not be a problem, since the dataloader has been written to allow for a 2-column header of arbitrary length.

2.1 Example Data File

At the moment, the data file is a tab-separated text file. Using tab as a separator elegantly avoids problems with whitespaces and special characters when parsing the data (keeps the dataloader code short, simple and flexible).

```
DataFileTitle  Polystyrene of Markus Strobl,  Full Sine, ++ only
Sample  Polystyrene 2 um in 53% H2O, 47% D2O
Settings  D1=D2=20x8 mm,Ds = 16x10 mm (WxH), GF1 =scanning, GF2 = 2.5 A. 2 um polystyrene i
Operator  CPD
Date  ma 7 jul 2014 18:54:43
ScanType  sine one element scan
Thickness [mm]  2
Q_zmax [\AA^-1]  0.05
Q_ymax [\AA^-1]  0.05

spin echo length [nm]   error SEL   wavelength [nm]   error wavelength   polarisation   erro
49.778 2.4889 0.211 0.01055 0.99782 0.0044367
63.041 3.152 0.211 0.01055 1.0026 0.0047862
76.487 3.8244 0.211 0.01055 0.99601 0.0060598
89.847 4.4924 0.211 0.01055 0.99175 0.0058257
103.41 5.1705 0.211 0.01055 0.99543 0.0060966
116.95 5.8475 0.211 0.01055 0.99512 0.0048106
etc....
```

3 Dataloader

The data loader uses the data file and parses it for use in Sasview (plotting only for now) and loading into BUMPS (for data fitting). Future changes to the Sasview source code should have minimal or no impact on the code of the dataloader.

One new file was created in the Sasview source code to load in data:

sesans_reader.py (location: C://Users/[User name]/sasview-code/src/sas/dataloader/readers/).

The code for this file has been added to this document in Appendix 7

3 existing files of the source code were modified:

data_info.py (location: C://Users/[User name]/sasview-code/src/sas/dataloader/): added the SESANSData1D class to handle SESANS data as opposed to Data1D or Data2D for SANS/SAXS data. Added the plottable_sesans1D class to handle SESANS plots.

associations.py (location: C://Users/[User name]/sasview-code/src/sas/dataloader/readers/): added sesans_reader to the registry to allow use in Sasview.

defaults.json (location: C://Users/[User name]/sasview-code/src/sas/dataloader/readers/): added default file extension .ses to allow automatic detection of SESANS data files; this is not yet working as intended.

4 Wrapper from SAS model to SESANS in absolute units

The conversion from SANS into SESANS in absolute units is a simple Hankel transformation when all the small-angle scattered neutrons are detected. First we calculate the Hankel transform including the absolute intensities by

$$\widetilde{G}(\delta) = 2\pi \int_0^\infty J_0(Q\delta) \frac{d\Sigma}{d\Omega}(Q) Q dQ, \tag{3}$$

in which J_0 is the zeroth order Bessel function, δ the spin-echo length, Q the wave vector transfer and $\frac{d\Sigma}{d\Omega}(Q)$ the scattering cross section in absolute units. This is a 1-dimensional integral, which can be rather fast. In the numerical calculation we integrate from $Q_{min} = 0.1 \times 2\pi/R_{max}$ in which R_{max} will be model dependent. We determined the factor 0.1 by varying its value until the value of the integral was stable. This happened at a value of 0.3. The have a safety margin of a factor of three we have choosen the value 0.1. For the solid sphere we took 3 times the radius for R_{max}. The real integration is performed to Q_{max} which is an instrumental parameter that is read in from the measurement file. From 3 we can calculate the polarisation that we measure in a SESANS experiment:

$$P(\delta) = e^{t\left(\frac{\lambda}{2\pi}\right)^2 \left(\widetilde{G}(\delta) - \widetilde{G}(0)\right)}, \tag{4}$$

in which t is the thickness of the sample and λ is the wavelength of the neutrons.

For collimation by a supermirror analyser the Q_{max} will be fixed.

2 new files were created for fitting the SESANS data:

The file containing the fit model and parameter boundaries to be used in the fitting, sesansfit.py (location: C://Users/[User name]/sasmodels/example/)

The file containing the Hankel transform, sesans.py (location: C://Users/[User name]/sasmodels/sasmodels/)

5 Combination of dataloader with wrapper to fit in command line with bumps

Install the sasview and sasmodels source code (description in Section 6 Make sure you have a Git Bash prompt open and that your path is in the sasmodels folder.

Now it's time to use BUMPS for SESANS fitting: Place the .ses (or .sesans) file that you wish to fit into the 'sasmodels/example/' folder and enter the following into the command-line:

```
./bumps.sh example/sesansfit.py --preview (for a quick preview of the fit)
(something should appear on your screen if the installation was successful)
```

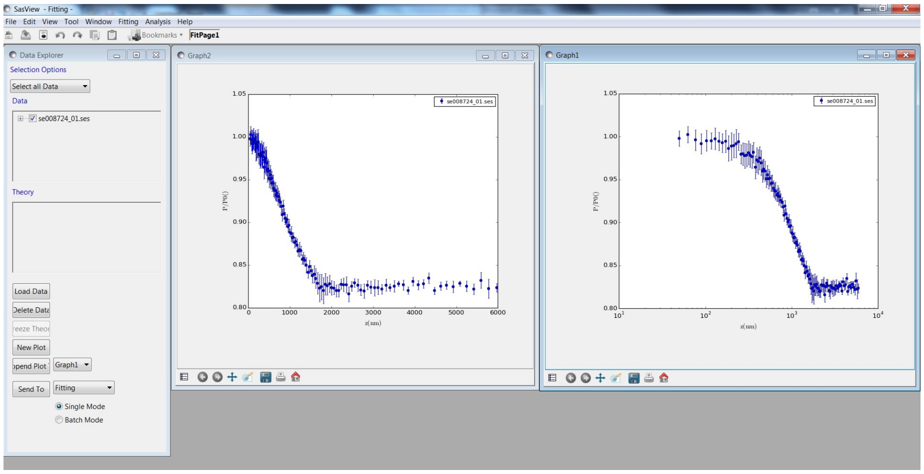


Figure 1: a window showing a plot of SESANS data in the current version of Sasview

```
./bumps.sh example/sesansfit.py --edit (for interactive GUI mode).
./bumps.sh example/sesansfit.py --fit=amoeba --edit (a very good fitting algorithm).
```

Note that you can also call up a help menu using the -help option, we give no guarantees that any of the options given by help, other than those listed above, will work though: BUMPS is still a work in progress. Best of luck with your SESANS fitting!

6 Appendix 1: Installation Guide

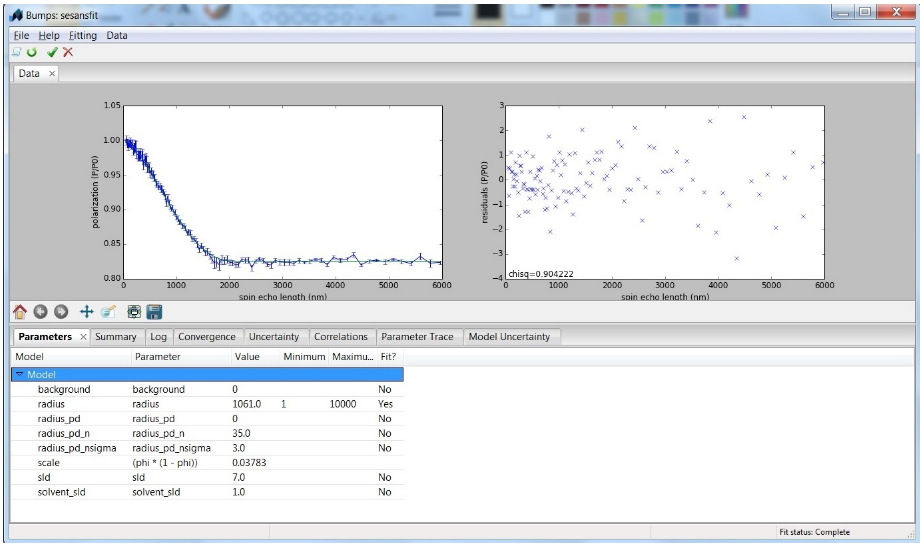
For the moment, Sasview, and with it, the SESANS fitting module can only be used on Windows operating systems. Firstly, follow the 'Easy Developer Setup on Windows Using Anaconda' on the Sasview wiki (trac.sasview.org/wiki/AnacondaSetup) with one modification: in step 3, select to install Git, not Subversion. Be sure to create a Github account if you have none. After completing the installation of sasview, perform the following steps:

Clone the sasmodels package from github by entering the following into the command-line:

```
https://github.com/Sasview/sasmodels.git sasmodels
```

Download Intel Opencl drivers from: https://software.intel.com/en-us/articles/opencl-drivers#win64 and install it (this is to make the BUMPS package work).

From http://www.lfd.uci.edu/~gohlke/pythonlibs/#pyopencl, Download pyopencl-2015.1-cp27-none-win32.whl Enter the following into the command-line: pip install [THIS FILE] (make sure to be in the correct directrory, otherwise you need to enter the entire path to it).



Model	Parameter	Value	Minimum	Maximu...	Fit?
Model					
background	background	0			No
radius	radius	1061.0	1	10000	Yes
radius_pd	radius_pd	0			No
radius_pd_n	radius_pd_n	35.0			No
radius_pd_nsigma	radius_pd_nsigma	3.0			No
scale	(phi * (1 - phi))	0.03783			
sld	sld	7.0			No
solvent_sld	solvent_sld	1.0			No

Figure 2: a window showing the result of fitting the Hankel transform of the SANS form factor of a sphere to a SESANS measurement of dilute polystyrene spheres in deuterium oxide using the BUMPS software. The fitting algorithm used was the Nelder-Mead Simplex.

For the moment, you will need to perform a small hack of Anaconda. You need to modify [Anaconda directory]/Lib/site-packages/bumps/gui/fit_thread.py, line 153: Change line to If False to turn off multi-core support (There is no multi-core support in the current implementation)

In Git Bash (Unix prompt from Git, comes with the Git installation) , navigate to sasmodels folder (should be C:/Users/[User name]/sasmodels.

Create file bumps.sh in the sasmodels main dir (easiest method is to open compare.sh and adjust the appropriate lines, then save as bumps.sh)) to easily use BUMPS fitting system (this is simply a text file): Contents are

```
#!/bin/sh

SASVIEW=$PWD/../sasview-code/src
PYTHONPATH=$PWD:$PWD/../bumps:$PWD/../periodictable:$SASVIEW
export PYOPENCL_CTX PYTHONPATH

echo PYTHONPATH=$PYTHONPATH
set -x

python -m bumps.cli $*
```

7 Appendix 2: Data-loader code

7.1 sesans_reader.py

```
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 20 10:28:18 2015

@author: jhbakker
"""

"""
    SESANS reader (based on ASCII reader)

    Reader for .ses or .sesans file format

    Jurrian Bakker
"""
import numpy
import os
from sas.dataloader.data_info import SESANSData1D

# Check whether we have a converter available
has_converter = True
try:
    from sas.data_util.nxsunit import Converter
except:
    has_converter = False
_ZERO = 1e-16

class Reader:
    """
    Class to load sesans files (6 columns).
    """
    ## File type
    type_name = "SESANS"

    ## Wildcards
    type = ["SESANS files (*.ses)|*.ses",
            "SESANS files (*..sesans)|*.sesans"]
    ## List of allowed extensions
    ext = ['.ses', '.SES', '.sesans', '.SESANS']

    ## Flag to bypass extension check
    allow_all = True
    def read(self, path):

#        print "reader triggered"

        """
        Load data file

        :param path: file path

        :return: SESANSData1D object, or None

        :raise RuntimeError: when the file can’t be opened
        :raise ValueError: when the length of the data vectors are inconsistent
        """
        if os.path.isfile(path):
            basename = os.path.basename(path)
            _, extension = os.path.splitext(basename)
            if self.allow_all or extension.lower() in self.ext:
                try:
                    # Read in binary mode since GRASP frequently has no-ascii
                    # characters that brakes the open operation
                    input_f = open(path,’rb’)
                except:
                    raise  RuntimeError, "sesans_reader: cannot open %s" % path
                buff = input_f.read()
#                print buff
                lines = buff.splitlines()

                x  = numpy.zeros(0)
                y  = numpy.zeros(0)
                dy = numpy.zeros(0)
                lam  = numpy.zeros(0)
                dlam = numpy.zeros(0)
                dx = numpy.zeros(0)

                output = SESANSData1D(x=x, y=y, lam=lam, dy=dy, dx=dx, dlam=dlam)
                self.filename = output.filename = basename


                data_conv_z = None
                data_conv_P = None
                if has_converter == True and output.x_unit != ’A’:
                    data_conv_z = Converter(’nm’)
                    # Test it
                    data_conv_z(1.0, output.x_unit)
```

```
if has_converter == True and output.y_unit != ' ':
    data_conv_P = Converter('a.u.')
    # Test it
    data_conv_P(1.0, output.y_unit)
paramnames=[]
paramvals=[]
zvals=[]
dzvals=[]
lamvals=[]
dlamvals=[]
Pvals=[]
dPvals=[]
for line in lines:
    line=line.strip()
    toks = line.split('\t')
    if len(toks)==2:
        paramnames.append(toks[0])
        paramvals.append(toks[1])
    if len(toks)>5:
        zvals.append(toks[0])
        dzvals.append(toks[1])
        lamvals.append(toks[2])
        dlamvals.append(toks[3])
        Pvals.append(toks[4])
        dPvals.append(toks[5])
    else:
        continue
x=[]
y=[]
lam=[]
dx=[]
dy=[]
dlam=[]
varheader=[zvals[0],dzvals[0],lamvals[0],dlamvals[0],Pvals[0],dPvals[0]]
valrange=range(len(zvals)-1)
for i in valrange:
    x.append(float(zvals[i+1]))
    y.append(float(Pvals[i+1]))
    lam.append(float(lamvals[i+1]))
    dy.append(float(dPvals[i+1]))
    dx.append(float(dzvals[i+1]))
    dlam.append(float(dlamvals[i+1]))

x,y,lam,dy,dx,dlam = [
   numpy.asarray(v, 'double')
   for v in (x,y,lam,dy,dx,dlam)
```

```
                ]
                input_f.close()
                output.x = x #[x != 0]
                output.y = y #[x != 0]
                output.dy = dy
                output.dx = dx
                output.lam = lam
                output.dlam = dlam
                output.xaxis("\\rm{z}", 'A')
                output.yaxis("\\rm{P/P0}", " ")
                output.meta_data['loader'] = self.type_name
                output.sample.thickness = float(paramvals[6])
                output.sample.name = paramvals[1]
                output.sample.ID = paramvals[0]
                output.sample.zacceptance=float(paramvals[7])
                output.vars=varheader
                if len(output.x) < 1:
                    raise RuntimeError, "%s is empty" % path
                return output

        else:
            raise RuntimeError, "%s is not a file" % path
        return None
```

7.2 sesansfit.py

```
import numpy as np
from bumps.names import *

from sasmodels import bumps_model as sas
kernel = sas.load_model('sphere', dtype='single')


if True: # fix when data loader exists
#    from sas.dataloader.readers\
    from sas.dataloader.loader import Loader
    loader=Loader()
    data=loader.load('testsasview1.ses')
    data.x /=10

#    data = load_sesans('mydatfile.pz')
#    sans_data = load_sans('mysansfile.xml')

else:
    SElength = np.linspace(0, 2400, 61) # [A]
    data = np.ones_like(SElength)
    err_data = np.ones_like(SElength)*0.03

    class Sample:
        zacceptance = 0.1 # [A^-1]
        thickness = 0.2 # [cm]

    class SESANSData1D:
        #q_zmax = 0.23 # [A^-1]
        lam = 0.2 # [nm]
        x = SElength
        y = data
        dy = err_data
        sample = Sample()
    data = SESANSData1D()

radius = 1000
data.Rmax = 3*radius # [A]

##  Sphere parameters

phi = Parameter(0.1, name="phi")
model = sas.BumpsModel(data, kernel,
    scale=phi*(1-phi), sld=7.0, solvent_sld=1.0, radius=radius)
phi.range(0.001,0.90)
#model.radius.pmp(40)
model.radius.range(100,10000)
#model.sld.pmp(5)
#model.background
#model.radius_pd=0
#model.radius_pd_n=0

if False: # have sans data
    sansmodel = sas.BumpsModel(sans_data, kernel, **model.parameters())
    problem = FitProblem([model, sansmodel])
else:
    problem = FitProblem(model)
```

7.3 sesans.py

```
"""
Conversion of scattering cross section from SANS in absolute
units into SESANS using a Hankel transformation
```

```
Everything is in units of metres except specified otherwise

Wim Bouwman (w.g.bouwman@tudelft.nl), June 2013
"""

from __future__ import division

import numpy as np
from numpy import pi, exp

from scipy.special import jv as besselj

def make_q(q_zmax, Rmax):
    q_min = dq = 0.1 * 2*pi / Rmax
    #q_min = 0.00003
    return np.arange(q_min, q_zmax, dq)

# TODO: dead code; for now the call to the hankel transform happens in BumpsModel
class SesansCalculator:
    def __init__(self, sans_kernel, q_zmax, Rmax, SElength, wavelength, thickness):
        self._set_kernel(sans_kernel, q_zmax, Rmax)
        self.SElength = SElength
        self.wavelength = wavelength
        self.thickness = thickness

    def _set_kernel(self, sans_kernel, q_zmax, Rmax):
        input = sans_kernel.make_input([make_q(q_zmax, Rmax)])
        self.sans_calculator = sans_kernel(input)

    def __call__(self, pars, pd_pars, cutoff=1e-5):
        Iq = self.sans_calculator(pars, pd_pars, cutoff)
        P = hankel(self.SElength, self.wavelength, self.thickness, self.q, Iq)
        self.Iq = Iq
        return P

def hankel(SElength, wavelength, thickness, q, Iq):
    """
    Compute the expected SESANS polarization for a given SANS pattern.

    Uses the hankel transform followed by the exponential.  The values
    for zz (or spin echo length, or delta), wavelength and sample thickness
    information should come from the dataset.  *q* should be chosen such
    that the oscillations in *I(q)* are well sampled (e.g., 5*2*pi/d_max).

    *SElength* [A] is the set of z points at which to compute the hankel transform
```

```
*wavelength* [m]  is the wavelength of each individual point *zz*

*thickness* [cm] is the sample thickness.

*q* [A^{-1}] is the set of q points at which the model has been computed.
These should be equally spaced.

*I* [cm^{-1}] is the value of the SANS model at *q*
"""
G = np.zeros(len(SElength), 'd')
for i in range(len(SElength)):
    integr = besselj(0,q*SElength[i])*Iq*q
    G[i] = np.sum(integr)
dq=(q[1]-q[0])*1e10   # [m^-1] step size in q, needed for integration
G *= dq*1e10*2*pi # integr step, conver q into [m**-1] and 2 pi circle integr
P = exp(thickness*wavelength**2/(4*pi**2)*(G-G[0]))

return P
```

8 Acknowledgements

We would like to thank everybody of the SASVIEW team for their patient help with introducing us to this great project. This report is the product of a very intensive collaboration during the code camp 2015 in Copenhagen.