McSAS Documentation

Release 1.0

Brian R. Pauw

March 14, 2013

CONTENTS

CHAPTER

ONE

MCSAS QUICK USAGE GUIDE

1.1 Introduction

This guide is intended as an aid to getting the first fits using McSAS. For comprehensive details of what goes on under the hood, please refer as a baseline to the paper. Additionally, the code is open source, and provides the best "documentation" of what actually takes place.

When publishing results using this code, please be so kind as to cite the paper as:

> Pauw, B. R., Pedersen, J. S., Tardif, S., Takata, M. and Iversen, B. B., J. Appl. Cryst. 46 (2013), *in press*

This document assumes a modicum of proficiency with the Python language, to a level sufficient for reading in a dataset.

1.1.1 Changes since the paper include:

A replacement of p_c by $Rpfactor$, where $Rpfactor = 1 - \frac{p_c}{6}$. Setting $Rpfactor = \frac{1}{2}$ will work in most cases.

1.1.2 Scope of the code capabilities

The McSAS code at the moment can:

1. Fit supplied data to a set of polydisperse spheres
2. Plot the data and fit alongside the distribution
3. Rebin the result
4. Export the result to a semilcolon-delimited CSV file.

These aspects will be discussed in that order in this document.

1.2 -1. Loading the Python functions

Assuming you have a suitable Python prompt, such as provided by iPython from the Enthought Python Distribution, you can load the McSAS functions using:

```
execfile('McSAS.py')
```

1.3 0. Loading test data

A set of test data has been included in the directory, with concatenated data from two measurements (measured over different angular ranges). The sample is a porous organic material. The uncertainty has been estimated as outlined in the paper.

The dataset can be loaded using the pickle functions provided in McSAS:

```
QIE = pickle_load('test_data.pydat')
q = QIE[0,:]
I = QIE[1,:]
E = QIE[2,:]
```

1.4 1. Fitting a dataset

It is assumed that a dataset has been loaded into Python, and that the 1D q-vector, accompanying 1D intensity and uncertainty (“error”, or standard deviation) is available as “q”, “I”, and “E”, respectively. For consistency throughout the calculation and its parameters, all length units are given in meters, which is particularly important if the measurement is done in absolute units, and retrieval of the actual volume fraction is required.

This implies that “q” must have the units of $\left[\frac{1}{m}\right]$ and “I” and “E” must have the units of $\left[\frac{1}{m \cdot sr}\right]$. Additionally, the optional size bounds are to be given in meters, and the scattering contrast, delta rho 2 squared, should be given in $\left[\frac{1}{m}\right]$.

If, on the other hand, the intensity is in relative units, or if the size distribution can be in relative volume fraction, the units do not matter much. They should be consistent, but q can then be given in 1/nm, and the size bounds likewise in nm. The resulting distribution will then have the same units.

The most straightforward way of fitting is by running:

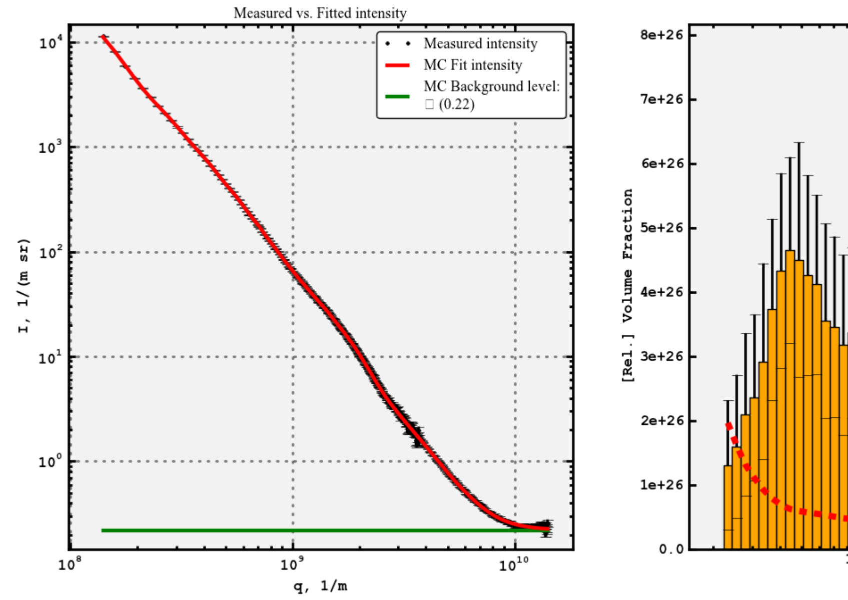
```
Result = Analyze_1D(q, I, E)
```

Which after a minute or so (or about five minutes for the provided extended dataset on an Intel i7 at 1.8 GHz (MacBook Air) will result in a dictionary called Result with all the fitting details in it.

1.5 2. Plotting the result

The result can be visualised using the built-in plotting function, which puts the data, its fit and the resulting size distribution alongside one another. This can be automatically started by using the option Plot=True in the Analyze_1D code, but can also be run separately. To plot, simply type:

```
McPlot(q,I,E,Result)
```



Which should give you the plot shown in *test_data.pdf*:
The left-hand plot shows the data in black with error bars showing the uncertainty "E", the MC fit in red, and a green line indicating the fitted background level. The width of the green line also serves to indicate the fitting limits in q, and the background value is furthermore indicated in the legend.

The right-hand plot shows the resulting volume-weighted size histogram, with uncertainties on the bars, and the red dashed line indicating the minimum level required for each bin to contribute a measurable amount (i.e. more than the uncertainty) to the scattering pattern.

As is clear from the vertical axis on that plot, the partial volume fractions are unrealistic if the scattering contrast has not been set. One may also want to rebin the plot in fewer bins to reduce the relative uncertainties on the bins.

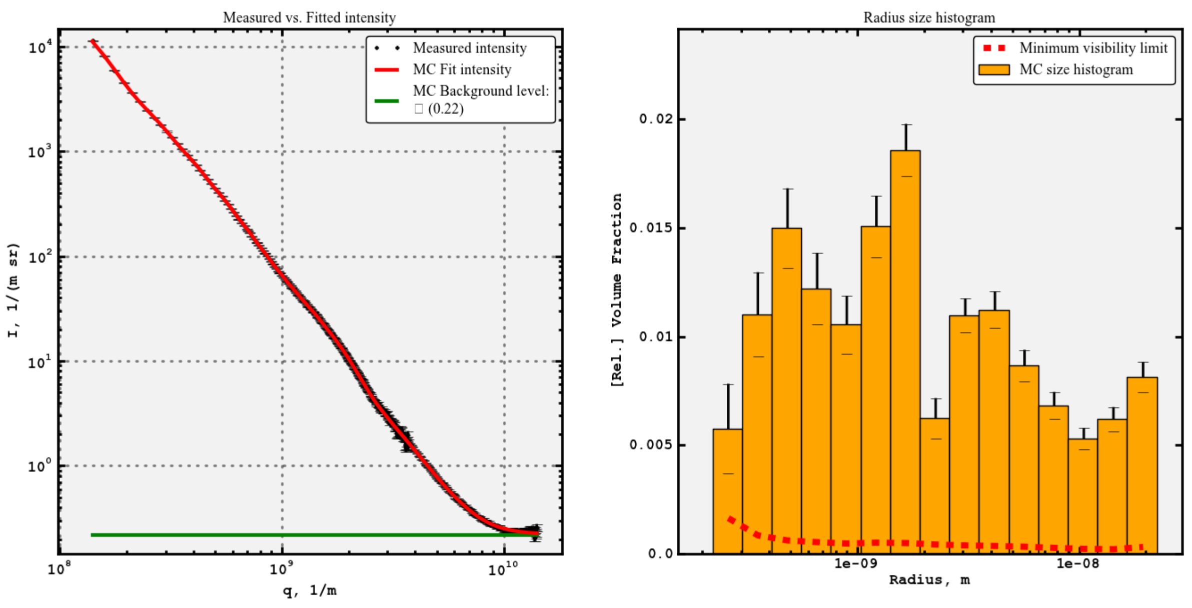
1.6 3. Rebinning the result

These things can be achieved through rebinning of the result. The rebinning process takes similar arguments as Analyze_1D, and we can thus rebin and replot using (with empty line delimiting the for-loop):

```
B = observability3(q, I, E, Rrep = Result['Rrep'], Rpfactor = 0.5,
                   Histbins = 15, Histscale = 'log', drhosqr = 1.0e29)

#copy all content of the result of observability3 to the output matrix
for keyname in B.keys():
    Result[keyname] = B[keyname]
    McPlot(q, I, E, Result)
```

This should give the following figure (as shown in *test_data_plot2.pdf*):



This plot shows more reasonable values for the relative volume fraction, the total volume fraction can be calculated from the 100 repetitions using:

```
numpy.mean(Result['Vft'])
```

which should result in about 15%.

1.7 3.5 Everything in one step

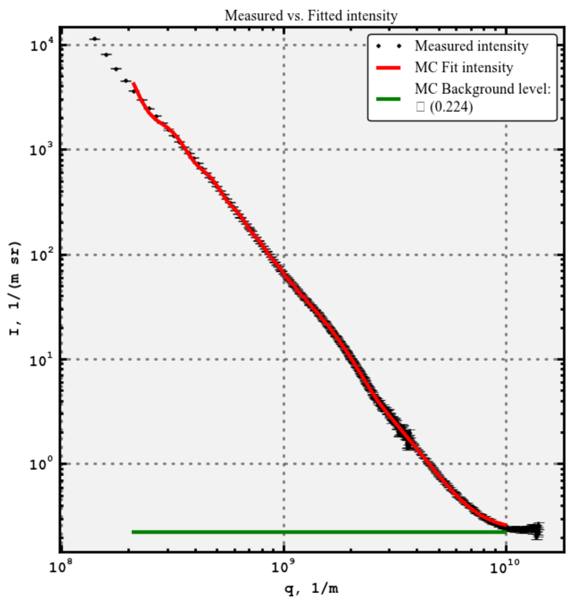
If you have a reasonable idea about what you want to do:

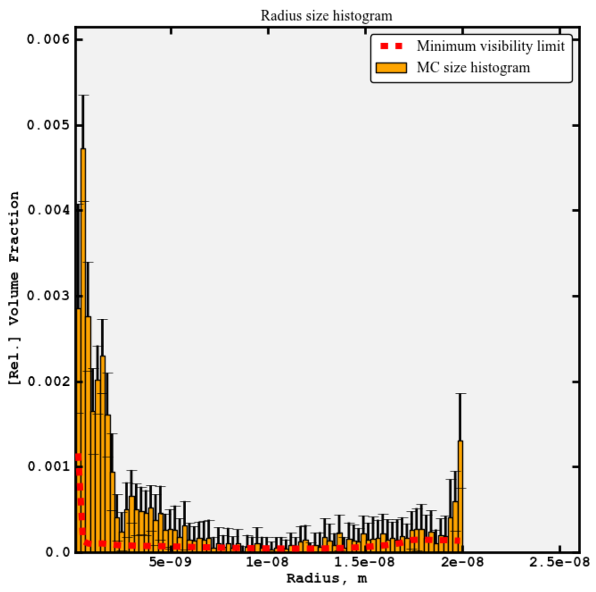
1. fit to within a chi-squared of 5.
2. Only 25 repetitions
3. Radius size bounds of [1e-10,2e-8] (it is not recommended to use '0' as starting point especially when using log-scaled histogram bins, but also because such sized scatterers are unphysical)
4. q-limits between 2e8 and 1e10 reciprocal meters (0.2 and 10 reciprocal nanometers or 0.02 and 1 reciprocal ångström)
5. A scattering contrast of $5 \cdot 10^{29} m^{-2}$
6. 80 bins, linearly scaled
7. using 100 sphere contributions

(there are more options but I am running out of ideas here:) Then you can do this in one step (with plotting) using:

```
Result = Analyze_1D(q, I, E, Convcrit = 5., Nreps = 25,
                    Bounds = [1e-10, 2e-8], qlims = [2e8, 1e10],
                    drhosqr = 5.0e29, Histbins = 80,
                    Histscale = 'lin', Nsph = 100, Plot = True)
```

Which returns a plot that looks a little wonky (given the low convergence criterion restrictions (the higher Convcrit, the more relaxed the criterion). But is otherwise perfectly valid (*test_data_plot3.pdf*):





1.8 4. Exporting the result

At this point it may be a good idea to get the histogram data out of python and into another plotting program. You can export whatever information you want, but to get a useful set of four columns indicating left bin edge, bin width, bin value (height) and bin uncertainty (standard deviation), you can use:

```
McCSV('hist.csv', Result['Hx'], Result['Hwidth'], Result['Hmean'], Result['Hstd'])
```

Which writes just that to a file named *hist.csv*, semicolon delimited. It actually contains one more bin edge, which is the trailing edge and is superfluous.

1.9 5. What's next?

If you have the ability and interest in improving the code, please consider joining the development effort, which will work on making the code object-oriented, including more shapes besides spheres, and adding slit-smearing options.

If you have more questions that are not answered in either 1) the paper, 2) the code, and 3) this document, feel free to send me an e-mail which you can find on the http://lookingatnothing.com/ weblog.

Good luck!

CHAPTER

TWO

SOURCE CODE DOCUMENTATION

This is a file with small programs for Monte-Carlo fitting of SAXS patterns. It is released under a Creative Commons CC-BY-SA license. Please cite as:

Brian R. Pauw, 2012, http://arxiv.org/abs/1210.5304 arXiv:1210.5304.

Also available open access at J. Appl. Cryst. 46, (2013) with doi: http://dx.doi.org/10.1107/S0021889813001295

Test Function: $\frac{s}{\sqrt{N}}$

Another Test:

$$\begin{aligned} y &= ax^2 + bx + c \qquad (2.1) \\ f(x) &= x^2 + 2xy + y^2 \qquad (2.2) \end{aligned}$$

Contents (updated 2013-01-16): *needs to be updated after finishing the OO variant** Analyze_1D: A wrapper for the MC code which repeatedly runs the optimization and

computes the final volume-weighted sphere distribution.

FF_sph_1D: computes the rayleigh form factor for a sphere radius or array of sphere radii

observability3: histogramming and uncertainty calculation for the Analyze_1D code. also calculates the minimum number of required contributions and the volume fractions in absolute units.

MCFit_sph: Monte-carlo fitting of the data to extract polydispersity assuming spheres, by changing the radius of a fixed number of spheres

binning_1D: bins the data and propagates errors, or calculates errors if not initially provided

binning_weighted_1D: Weighted binning, where the intensities of a pixel are divided between the two neighbouring bins depending on the distances to the centres. If error provided is empty, the standard deviation of the intensities in the bins are computed.

McPlot: A procedure for generating a data-fit plot and size histogram based on Analyze_1D's results

McCSV: Function to write an arbitrary number of semicolon-separated values to a file FixBounds: Internal function for estimating minimum and maximum size bounds based on

q values.

csqr: least-squares error to use with scipy.optimize.leastsq Iopt: Optimize the scaling factor and background level of modeled data vs. intensity csqr_v1: least-squares for data with known error, size of parameter-space not taken

into account

Iopt_v1: old intensity scaling factor optimisation, more robust but slower than Iopt pickle_read: Reads in pickled data from a file (by filename) pickle_write: write a block or dictionary to a file (by filename)

(asterisk * indicates code to be depreciated)

Made possible with help from (amongst others): Samuel Tardif - Derivations (mostly observability) and checking of mathematics Jan Skov Pedersen - checking of mathematics Pawel Kwasniewski <kwasniew@esrf.fr> - Code cleanup and documentation

`McSAS.`**`Icyl_InPlaneAverage_RandomSampling`**(*q*, *psi=array([0., 90.])*, *nsamples='auto'*, *R1=1.0*, *R2=1.0*)

calculates the in-plane average of a shape (rotational average, but only rotated around the beam axis, perpendicular to the detector plane). This calculation works by uniform random number generation for q and psi, within the bounds dictated by the input. Set "nsamples" to a number of samples. 'auto' sounds cool but has not been implemented yet. input 'q' is supposed to be a vector of q values for which the intensity is requested

`McSAS.`**`Integrate_Shape_as_Radial_Isotropic`**(*q*, *psirange=array([1.00000000e-01, 3.60100000e+02])*, *Func=''*, *parameters=array([0])*, *psidiv=303*, *randfact=0*)

Generate a 1D intensity of a radially isotropic shape. This is not identical to a fully rotationally isotropic shape, but merely isotropic as if the shape is rotated around the axis perpendicular to the detector.

`McSAS.`**`Integrate_Shape_as_Spherical_Isotropic`**(*q*, *psirange=array([1.00000000e-01, 3.60100000e+02])*, *Func=''*, *parameters=array([0])*, *psidiv=303*, *randfact=0*)

Generate a 1D intensity of a spherically isotropic shape. This is identical to a fully rotationally isotropic averaged shape, for radially isotropic shapes, use Integrate_Shape_as_Radial_Isotropic, which averages the shape around the axis perpendicular to the detector.

class `McSAS.`**`McSAS`**(***kwargs*)

Core Monto-Carlo Small Angle Scattering calculations.

`CSVwrite`(*filename*, **args*)

This function writes a semicolon-separated csv file to [filename] containing an arbitrary number of output variables *args. in case of variable length columns, empty fields will contain ''.

Input arguments should be names of fields in "self.results". For example: A.McCSV('hist.csv','Hx','Hwidth','Hmean','Hstd')

i.e. just stick on as many columns as you'd like. They will be flattened by default. a header with the names will be added.

existing files with the same filename will be overwritten by default.

`EllBounds_2D`()

This function will take the q and psi input bounds and outputs properly formatted two-element size bounds for ellipsoids. Ellipsoids are defined by their equatorial radius, meridional radius and axis misalignment (default -45 to 45 degrees in PSI).

`FF_ell_2D`(*Rset=[ ]*, *Q=[ ]*, *PSI=[ ]*)

all 2D functions should be able to potentially take externally supplied Q and PSI vectors

`FF_sph_1D`(*Rset*)

Calculate the Rayleigh function for a sphere

`Histogram`()

Observability calculation for a series of spheres, over a range of q. Additional intensity and errors may be supplied for error-weighted observability. Intensity is used for determining the intesity scaling and background levels.

Now with rebinning as well, so we can keep track of which contribution ends up in which bin and calculate the correct minimum required contribution accordingly.

`MCFit`(*OutputI=False*, *OutputDetails=False*, *OutputIterations=False*, *Prior=[ ]*)

Object-oriented and hopefully shape-flexible form of the MC procedure.

Plot (*AxisMargin=0.3*)
: This function plots the output of the Monte-Carlo procedure in two windows, with the left window the measured signal versus the fitted intensity (on double-log scale), and the righthand window the size distribution

SphBounds ()
: This function will take the q and input bounds and outputs properly formatted two-element size bounds.

TwoDGenI ()
: this function is run after the histogram procedure for anisotropic images, and will calculate the MC fit intensity in imageform

check_parameters ()
: in here we can place checks for the parameters, for example to make sure histbins is defined for all, or to check if all parameters fall within their limits

clip_dataset ()
: if q and/or psi limits are supplied in self.parameters, clips the dataset to within the supplied limits. Copies data to self.fitdata if no limits are set.

getdata (*parname=*[ ], *dataset='fit'*)
: gets the values of a dataset, retrieves from fitdata (clipped) by default. If the original data is wanted, use "dataset='original" as kwarg.

getpar (*parname=*[ ])
: gets the value of a parameter, so simple it is probably superfluous

random_logR_ell (*Nell=1*)
: like uniform, but with a higher likelihood of sampling smaller sizes. May speed up some fitting procedures.

reshape_fitdata ()
: This function ensures that q, I, PSI and E are in 1-by-n shape

set_defaults ()
: Populates the default parameter settings

setdata (***kwargs*)
: Sets the supplied data in the proper location.

setfunctions (***kwargs*)
: functions are defined here. In particular here the following is specified: -1. The parameter bounds estimation function 'BOUNDS'. Should be able to take input argument Bounds to update, should set the parameter bounds in self.parameter['Bounds'] 0. The random number generation function 'RAND' This must take its parameters from self, and have an optional input argument specifying the number of sets to return (for MC initialization). It should return a set of Nsets-by-nvalues to be used directly in 'FF' 1. The Form-factor function 'FF'. If called, this should get the required information from self and a supplied Nsets-by-nvalues shape-specifying parameter array. It should return an Nsets-by-q array. Orientational averaging should be part of the form-factor function (as it is most efficiently calculated there), so several form factor functions can exist for non-spherical objects. 2. The shape volume calculation function 'VOL', which must be able to deal with input argument "Rpfactor", ranging from 0 to 1. Should accept an Nsets-by-nvalues array returning an Nsets number of (Rpfactor-compensated)-volumes. 3. The smearing function 'SMEAR'. Should take information from self, and an input Icalc, to output an Ismear of the same length.

: This function will actually cast the supplied function name into a function pointer.

setpar (***kwargs*)
: Sets the supplied parameters given in keyword-value pairs for known setting keywords (unknown key-value pairs are skipped) If a supplied parameter is one of the function names, it is stored in the self.functions dict.

`setresult`(***kwargs*)

Sets the supplied keyword-value pairs to the result. These can be arbitrary. Varnum is the sequence number of the variable for which data is stored. Default is set to 0, which is where the output of the MC routine is put before histogramming. The Histogramming procedure may populate more variables if so needed.

`vol_ell`(*Rset*, *Rpfactor*=[ ])

calculates the volume of an ellipsoid, taking Rpfactor from input or preset parameters

`vol_sph`(*Rset*, *Rpfactor*=[ ])

calculates the volume of a sphere, taking Rpfactor from input or preset parameters

`McSAS.`**`binning_array`**(*Q*, *PSI*, *I*, *IERR*, *S=2*)

this function applies a simple S-by-S binning routine on images. Calculates new error based on old error superseded by standard deviation in a bin

CHAPTER THREE

HOW TO GENERATE THE DOCUMENTATION

3.1 Requirements

- Python, of course
- Sphinx package
- For Latex/PDF generation, there should be a latex environment installed

3.2 Generate a PDF document

```
cd <mcsas>/doc
make latexpdf
```

The resulting `McSAS.pdf` can be found in `<mcsas>/doc/_build/latex/`.

3.3 Generate HTML pages

```
cd <mcsas>/doc
make html
```

The entry point `index.html` can be found in `<mcsas>/doc/_build/html/`.

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *search*

PYTHON MODULE INDEX

INDEX