Software manual - MCFit

Brian R. Pauw
(Dated: January 11, 2011)

CONTENTS

GENERAL INTRODUCTION

This document attempts to explain the usage of some software packages written in Matlab revolving around the analysis of small-angle scattering patterns. The functions mostly take a few required input arguments, and optional arguments are supplied in so-called "parameter-value" pairs. This means that two arguments are supplied to the function, one "parameter", usually a string identifying the second argument, the "value". This looks like this in Matlab (where `>>` is the matlab prompt):

```
>>  [output_variable_1,output_variable_2,...,output_variable_n]=
          function_name(required_input_1,required_input_2,'parameter_1',1234,'parameter_2',4567);
```

These parameter-value pairs can be placed in any order (with one exception in "perfectpattern" programs), as long as the correct value follows the parameter. Limited input checks are done, but it is mostly up to the user to try to not make mistakes.

The code should be well-commented and readable, and no mistakes should remain. If, however, you do see room for improvement, please let me know so we can work on it!

MCFIT

purpose

The purpose of this program is to explore Monte-Carlo based fitting routines. The method implemented is the method which demonstratively works. Other methods have been attempted, but have not yielded success so far.

Method

The method is quite straightforward:

- The program starts with a number of spheres (400 spheres by default) uniformly distributed in radius, spanning a range (which is either predefined or estimates for the limits can be obtained from the q-limits of the data) of 0-40 Ångström for measurements before number 22 and 0-100 for measurements 22 and higher. The range governs the speed at which it arrives at a reasonable solution.
- The scattering intensity from these spheres is calculated, assuming infinite dilution. This intensity is then fit to the data, with a two-parameter non-linear least-squares minimisation routine (the implementation of which was the culprit for its dramatic reduction in speed). The calculated intensity is scaled (the first parameter) and a flat background level is added (the second parameter).
- A single sphere is picked and its radius is changed to another radius within the range.
- The intensity and the goodness-of-fit is recalculated
- If it is an improvement, the radius change is kept, otherwise discarded (there are arguments for accepting some bad choices occasionally, but they have not been implemented yet).
- This process is to be repeated about 1e5 times, depending on the number of spheres.

drawbacks

There is one major drawback with this method, which is that if a very wide distribution is present in your sample, there are not enough spheres in the small-radius region to compensate for the volume weighting. When that is the case, it is easily observed that a good fit has not been obtained, and that the number of spheres is to be increased.

Besides this drawback, the method also cannot be considered fast, running at about 50-60 iterations per second on my 2009 MacBook Pro. As mentioned, the culprit is the implementation of the non-linear least-squares fit. As such, the method can take about 20 minutes to fit a scattering pattern to a reasonable degree. There must be a better solution than the implementation of the least-squares fit, but the intention was to stick with the original chi-squared goodness-of-fit measure so that a variety of weightings could be applied to the data, and the only method to come to mind to implement this was by means of the least-squares fit. In the future, help from mathematicians or statisticians may resolve this.

usage

```
[Icalc,R,varargout]=MCfit_sph(q,I,varargin)
```

Input consists of a vector of q and I, which are to be of equal length. Optional arguments in are listed below. Output consists of $Icalc$, the calculated intensity of the last successful MC step, and a list of sphere radii in R. Optional arguments out are for the third parameter: a structured array with settings, and for the fourth parameter: the vector $qfit$, in case q limits have been applied.

TABLE I. Parameter-value pairs and explanation.

Parameter	Value type	Description
weighting	string or vector	Can be either of "relative", "unit" or "poisson", or a vector of size q or I, with the errors for each intensity value.
nsph	integer	number of spheres to be simulated, default 400. More spheres increases flexibility but also increases the number of required iterations.
nitr	integer	number of iterations to be carried out, default 1×10^5.
convcrit	float	convergence criterium, default is 1, suitable for Poisson weighting.
graphics	bool	if set to 1, shows progress every 100 steps, but slows down progress considerably.
exportgraphics	bool	if set to 1, stores graphic images of the progress for later use in a movie
prior	vector	vector of length nsph, with prior guesses of the radii, for example from a prior computation
bounds	vector	two-element vector with lower and upper radius bounds.
qlim	vector	two-element vector with lower and upper limits of the range of q to fit.
param	struct	a structured array previously output by the program can be used to load a set of settings. Any changes should come after this parameter-value pair.
tolerance	float	a tolerance limit for the least-squares minimisation routine, default set to 10 % of the convergence criterium. If the convergence criterium is set to zero, the tolerance must be set!

Optional parameter-value pairs

examples

One initial example can be generated using the "perfectpattern_spheres.m" program, to simulate a scattering pattern from a set of narrowly polydisperse spheres:

```
[qsph,Isph,Rsph]=perfectpattern_spheres([0.01 0.5],'dparam',[50 10]);
```

This pattern (which is normalised to $I_0 = 1$), can then be fit using the "montecarlofit_sph.m" program to a first attempt through:

```
[Ifit,Rfit]=MCFit_sph(qsph,Isph.*1e6,'nitr',1e5);
```

Notice that we have increased the intensity to $I_0 = 1 \times 10^6$ so that the default Poisson weighting has sufficient simulated intensity to reach a reasonable fit. The Monte-Carlo fit will now fit until the reduced chi square has dropped below 1 (indicating that a fit to within the errors has been achieved), or until it has gone through 100000 monte-carlo fitting steps.

In one run, the fit converged to a χ^2 of 0.8749 in 15659 steps and 170 seconds. The resulting histogram, compared to the original "perfectpattern" histogram, is shown in Figure 1.

This example shows that the original size distribution has been partially retrieved, albeit slightly "rough". The entire q-range has been used for this fit, and the automatically determined bounds have been applied, i.e. random particles have been generated in a range of $3.1 \leq R \leq 314$ (the minimum bound is set lower than the sampling theorem limit ($\pi/q_{\max}$) to allow "overflowing" spheres to have a noncontributing space to end up in).

One could consider running the monte-carlo fit for a longer time, but this would only fit within the noise of the scattering pattern, and not extract any more information from it. Getting data with correct errors, therefore, is very useful towards a correct application of the method.

A second example can be shown with a wider particle size distribution, using distribution parameters of $R_\mu = 50$ and $R_\sigma = 30$:

```
[qsph,Isph,Rsph]=perfectpattern_spheres([0.01 0.5],'dparam',[50 30]);
```

In Figure 2, the comparison of the original and retrieved histogram is shown again, revealing that even for wider distributions, some distribution information has been retrieved.

The smoothness of the monte-carlo distribution can be improved by increasing the number of spheres in the monte-carlo simulation, at the penalty of an increased computational time. By increasing the number of spheres (from the default 400 to e.g. 5000) like so:

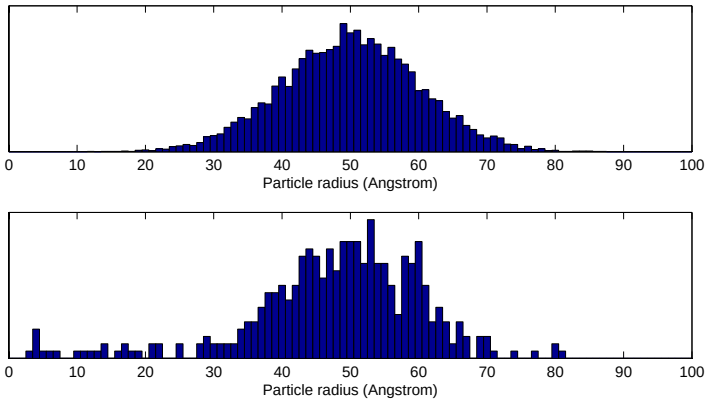

FIG. 1. Original (top) and retrieved (bottom) particle size distribution by fitting the simulated scattering pattern using a monte-carlo method.

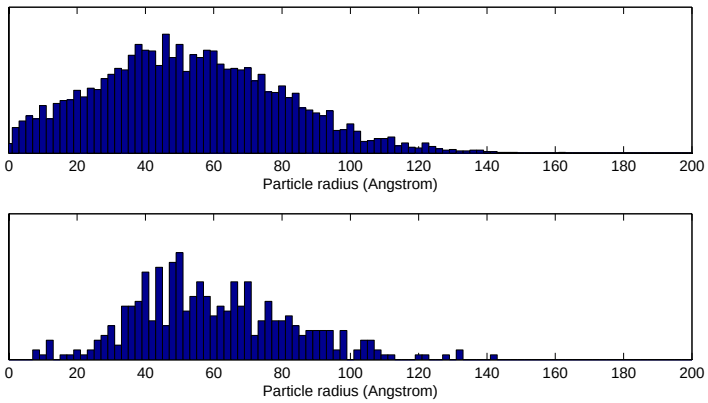

FIG. 2. Original (top) and retrieved (bottom) particle size distribution by fitting the simulated scattering pattern using a monte-carlo method.

```
[Ifit,Rfit]=MCFit_sph(qsph,Isph.*1e6,'nitr',1e5,'nsph',5000);
```

If we do so, and wait a bit longer for the result, we see that a reasonably smooth result can be obtained (Figure 3). For a better agreement, more virtual "counts" are required. Thus, this combination of programs can be used as well for design of experiments to determine the angular range and number of counts required to retrieve the desired information.

A full example where most of the options are used (but for which an Ierr (error values) is needed, for example, by stating $Ierr = 10. * \mathrm{sqrt}(Isph)$) is:

```
[Ifit,Rfit,param,qfit]=MCFit_sph(qsph,Isph,'weighting',Ierr,...
'nsph',1000,'nitr',1e6,'convcrit',1,'graphics',1,...
'exportgraphics',1,'prior',Rfit,'bounds',[0 100],'qlim',[0.05 0.3],'tolerance',0.01);
```

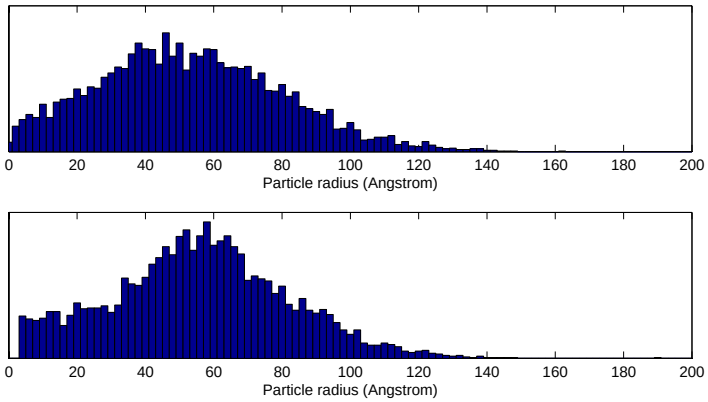

FIG. 3. Original (top) and retrieved (bottom) particle size distribution by fitting the simulated scattering pattern using a monte-carlo method.