Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original Software Publication

Web interface for reflectivity fitting

M. Doucet [*], R.M. Ferraz Leal, T.C. Hobson

Neutron Scattering Division, Oak Ridge National Laboratory, P.O. Box 2008, Oak Ridge, TN 37831, USA

ARTICLE INFO

Article history:
Received 25 January 2018
Received in revised form 9 May 2018
Accepted 3 September 2018

Keywords:
Neutron scattering
Reflectometry
User interface

ABSTRACT

The Liquids Reflectometer at Oak Ridge National Laboratory's Spallation Neutron Source provides neutron reflectometry capability for an average of about 30 experiments each year. In recent years, there has been a large effort to streamline the data processing and analysis for the instrument. While much of the data reduction can be automated, data analysis remains something that needs to be done by scientists. For this purpose, we present a reflectivity fitting web interface that captures the process of setting up and executing fits while reducing the need for installing software or writing Python scripts.

Code metadata

Current code version	*v1.0*
Permanent link to code/repository used of this code version	https://github.com/ElsevierSoftwareX/SOFTX_2018_44
Legal Code License	*Apache License, 2.0*
Code versioning system used	*git*
Software code languages, tools, and services used	*python*
Compilation requirements, operating environments & dependencies	https://web-reflectivity.readthedocs.io/en/latest/testing.html
If available Link to developer documentation/manual	https://web-reflectivity.readthedocs.io/en/latest/
Support email for questions	doucetm@ornl.gov

1. Motivation and significance

The Liquids Reflectometer (LR) is one of the two reflectometers at the Oak Ridge National Laboratory's Spallation Neutron Source (SNS) [1]. Specular reflectivity allows us to probe the depth profile of thin planar films by measuring the reflected distribution of beams of neutrons or X-ray photons scattered off their surface. In the case of the LR, neutron reflectivity measurements allow us to probe layer structures with length scales between a few Angstroms and a few thousand Angstroms.

The reflectivity distribution $R(Q)$ is measured as a function of momentum transfer $Q = 4\pi \sin(\theta)/\lambda$, where θ is the scattering angle and λ is the neutron wavelength. $R(Q)$ depends on the scattering length density (SLD) distribution along the axis perpendicular to scattering plane and is approximately given by [2]:

$$R(Q) \approx \frac{16\pi}{Q^4}\left|\int_{-\infty}^{\infty}\frac{d\beta}{dz}e^{-izQ}dz\right|^2$$

For a given compound, the scattering length density β is related to the scattering length of each atom in the system:

$$\beta = \rho N_A/m\sum_{i=1}^{n} b_i$$

where ρ is the mass density of the compound, m is its mass, N_A is Avogadro's number, and b_i is the coherent scattering length of the ith atom in the system. Fitting the reflectivity distribution thus gives us the SLD profile of our sample as a function of depth. This profile in turn gives us information about the chemical composition of our layered system.

The calculation of the reflectivity profile from a layered system is generally done using a solution of the Schrödinger equation for the reflected and transmitted particle waves at the layer boundaries [3,4]. This approach gives an accurate reflectivity value in all Q ranges, as opposed to the $R(Q)$ equation above which holds only in the region where the first Born approximation is valid.

The measured reflectivity $R(Q)$ is generally modelled by defining an initial SLD profile and using a fitting engine to optimize its parameters. This SLD profile is defined as a series of layers, each with a thickness, a uniform SLD, and an interfacial mixing

[*] Corresponding author.
E-mail address: doucetm@ornl.gov (M. Doucet).

https://doi.org/10.1016/j.softx.2018.09.001

depth with the layer above. The mixing depth is often referred to as 'roughness' and is commonly modelled as an error function, which describes how the SLD varies at the transition from one layer to the next. The reflectivity is computed for the SLD profile and compared to the measured data. A fitting engine is used to minimize the least-squares difference between the calculated reflectivity and the measurement. In this text, we refer to this process as minimization, or refinement, the outcome of which is a system of layers with an estimate for their thickness, SLD, and roughness, each with an uncertainty value.

The LR provides data for an average of about 30 experiments each year as part of the SNS User Program, producing about 10 000 data sets per year. Each experiment typically takes three to five days, with some taking only a single day and some taking up to a week. It is important to automate and simplify data treatment as much as possible. Once acquired, the LR data is automatically reduced using the Mantid framework for data analysis and visualization [5]. In most cases, users are able to leave the laboratory with reduced *R(Q)* reflectivity data ready to be analysed. Subsequent modelling of reflectivity data is usually done using Motofit [6] or REFL1D [7]. Other software packages such as GenX [8] or custom user software are also used. Motofit is a package that provides reflectivity modelling within the IGOR Pro environment.[1] In addition to its minimization and error analysis capabilities, Motofit provides a graphical user interface. REFL1D is a Python package originally developed by the DANSE Project[2] that also provides minimization and error analysis of reflectivity models. It does not provide a graphical user interface.

The choice of reflectivity fitting software is generally done with the help of SNS staff. Instrument scientists also train users in the use of the chosen software. For this reason, developing tools to simplify that process and empower visiting scientists to fit their data has been a focus of the LR staff for the past few years.

In the present article, we describe a web application developed as a front end to REFL1D. It integrates with the existing web monitoring site available at the SNS [9] and allows users to visualize and fit their reflectivity data using a simple set of web forms. It allows an easy transition from modelling activities done during their experiment and analysis work done at their own institution without the installation of commercial software. The application code is available at https://github.com/neutrons/web_reflectivity.

2. Software description

2.1. Non-functional requirements

The goal of this work is to provide reflectometry users with a simple data modelling application that does not require software installation. For this purpose, the main non-functional requirements were three-fold. First, the application needs to be available without software installation. Second, the application should allow users to upload their data while be flexible enough for an institution to use their own mechanism to access data. Third, the application needs to use an existing and well-established package to do the calculations.

2.2. Software functionalities

The functionality provided by the application can be organized in four main topics:

2.2.1. REFL1D job setup and execution

The application is designed to let users define a model for a given data set using a web form to set up their initial layer structure and fit parameters. Once a fit job is submitted, a Python script is created to perform the minimization using REFL1D. Information related to the model and fit status is kept up to date in the application database, which allows users to come back to a particular data set, view fit results, modify their model and resubmit as necessary.

2.2.2. Available engines

The reflectivity application can generate REFL1D scripts that use three fitting engines: a Levenberg–Marquardt algorithm [10,11], a Nelder–Mead algorithm [12], and the DREAM algorithm [13]. The DREAM algorithm is a hybrid algorithm that uses differential evolution to adapt the evolution of Markov chains. REFL1D uses the DREAM algorithm to infer the probability distribution of fit parameters and provide an error analysis that gives us uncertainty estimates for each fit parameter and a correlation plot between those parameters. The uncertainties are reported on the fit result page of the web application. The additional files produced by REFL1D, which include the correlation plot, theoretical SLD profiles, and theoretical reflectivity profiles, are saved in the user's home directory on the compute node.

2.2.3. Functional constraints

One strength of REFL1D is the ability to process complex constraints written as a function of fit parameters. The web interface allows users to add multiple constraints between the layer parameters of a model. Equations written in Python can be entered. A dedicated page is available to validate and manage the constraints for a given model.

2.2.4. Simultaneous fitting

The web application offers an interface to set up the simultaneous fit of multiple data sets. The results of those simultaneous fits are stored separately in the application database so that several fits involving the same data sets can be done without interfering with each other. Users can tie model parameters from data sets selected to be co-refined by simple drag-and-drop. Once those constraints are set and the fit is executed, the reflectivity and SLD plots of the resulting fits are presented along with the output parameters.

2.3. Software architecture

We chose to build a web application to address the required functionalities listed above. That choice enables us to cleanly separate the computations from the user interface, and it allows us to reach a large fraction of our users regardless of their location or their preferred platform. The Django web framework[3] was chosen as the basis of the application for its flexibility and simplicity. Django follows a model-view-template architectural pattern, which drove the design. Fig. 1 shows the central flow diagram of the application. It details the flow from the user's job request to the completion of the calculation. The following details the different parts of the diagram:

2.3.1. Data management

The software comes with a simple data manager that lets users upload their data to the server, which parses each file and stores the data in the application database as json. Json was chosen because it is easily serializable, which allows us serve data quickly to web clients for plotting. The data manager component, named *datahandler*, was packaged as a Django "application" so that it can easily be replaced.

[1] https://www.wavemetrics.com/.
[2] http://danse.us/.
[3] https://www.djangoproject.com/.