SoftwareX 7 (2018) 287–293

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/.

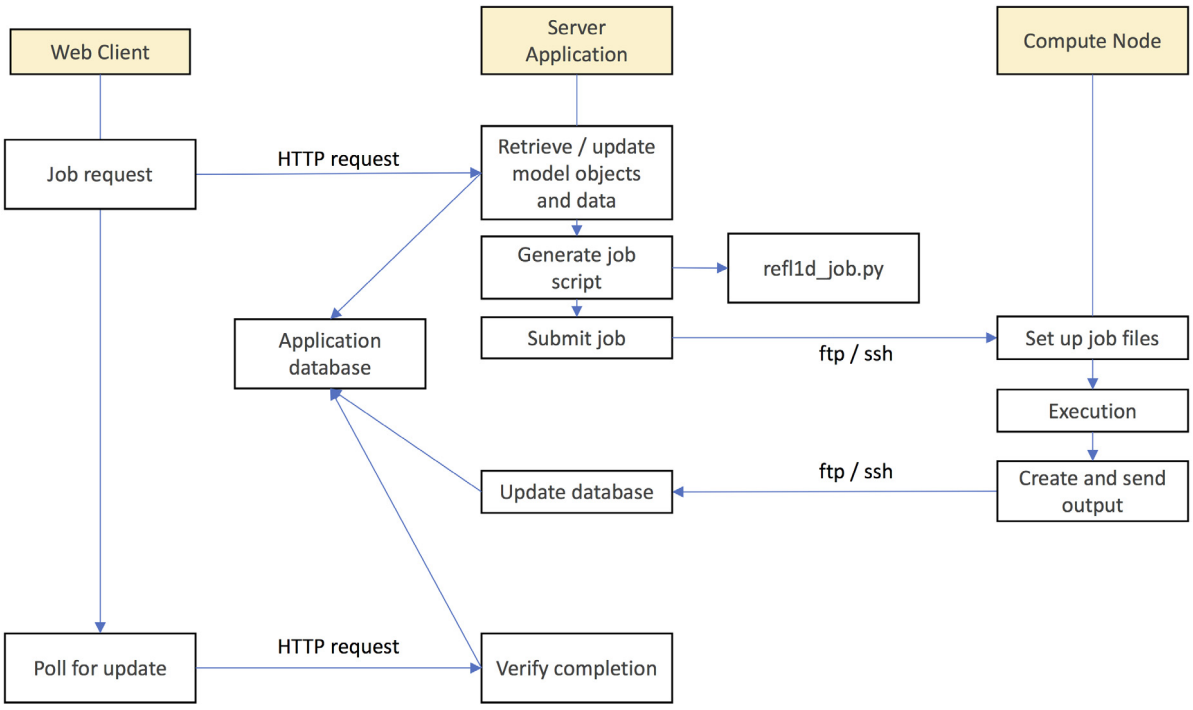

Fig. 1. Main activity diagram of the application. Upon receiving an HTTP request, the application retrieves the model objects from the database, updates them according to the request, generates a job script, and submits it to the compute node through ssh. Once the job is completed, the database is updated with the results of the fit. The client polls the server to know when a job is completed.

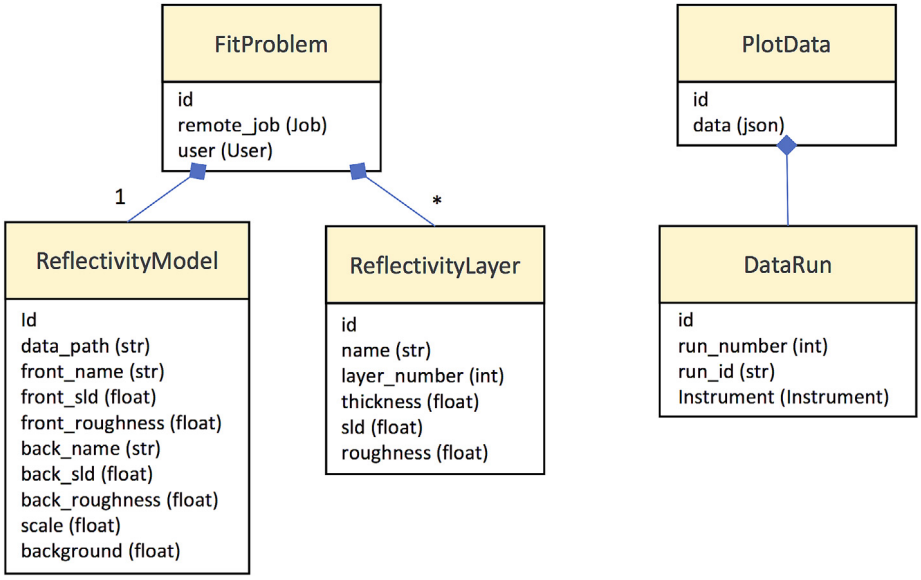

Fig. 2. Main class diagram for the application. The FitProblem is comprised of the initial layer structure and the remote job information. The layer structure is defined by a ReflectivityModel object and any number of ReflectivityLayer objects. The data is stored as json in a PlotData object, with a DataRun object containing addition metadata. All classes inherit from django.db.Model. The User class is provided by Django and the Job class is provided by Django Remote Submission [14]. The Instrument class only contains identification information for neutron instruments and was left out of the diagram for simplicity.

2.3.2. Fitting jobs

We define a fit problem as all the information needed to perform a minimization. The fit problem is the central concept of the design. At its core, the application mainly generates, stores, retrieves, and updates fit problems. Fig. 2 shows a class diagram for the main classes involved. Those classes inherit from Django's Model class, which abstracts out the database queries. Django also provides a mechanism to generate form objects to be presented to the user through the framework's templating system (see the example of Fig. 3). Model classes were chosen to minimize the complexity of the queries needed to extract the information from the database.

2.3.3. Interface to REFL1D, execution, and result storage

REFL1D is executed by calling it with a reflectivity model script. This script is responsible for loading the data and instantiating the objects REFL1D needs to define the minimization problem. The web application uses a templating system to generate a job script. The model objects shown in Fig. 2 and described in the previous section are used to create code that will be inserted in

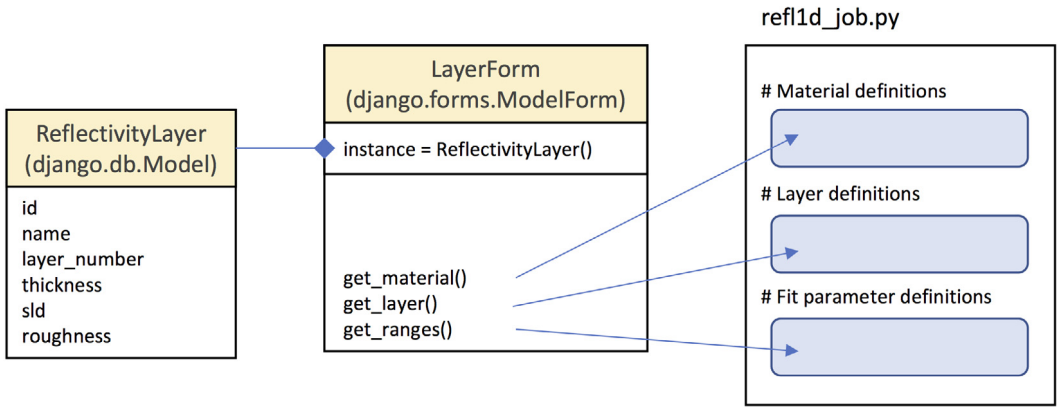

Fig. 3. Example relationship between a model class, its view class, and the REFL1D script template. The ReflectivityLayer class is used to store the parameters of a layer. Django provides utilities to create a ModelForm class that provides the functionality to represent itself as HTML. Our LayerForm class extends this functionality to provide code fragments to be used in our REFL1D script template.

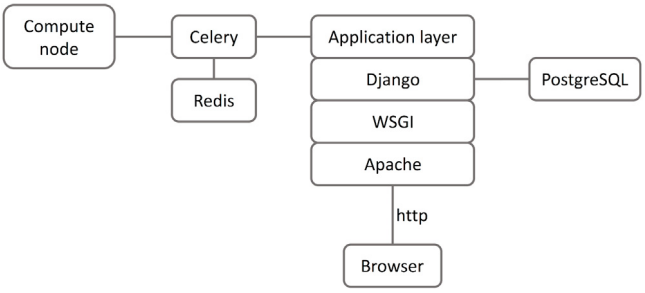

Fig. 4. Overview of the reflectivity fitting web interface, showing the software packages used to build the application.

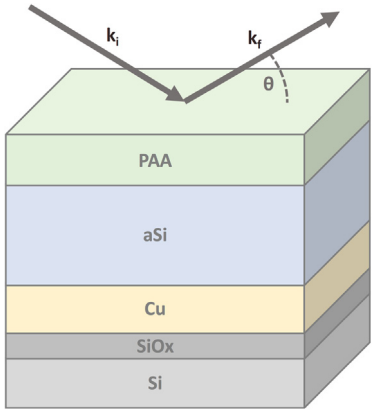

Fig. 5. Thin film structure for the example data set, consisting of a deposited layer of Cu on a silicon substrate, a deposited layer of amorphous silicon, and a spin-coated layer of PAA. SiOx denotes the silicon substrate's native oxide.

the job template. Fig. 3 shows an example of this process for the ReflectivityLayer object. The created script, which also packages the data used in the minimization, is copied to the execution node through ftp and the execution is done through ssh. The Django Remote Submission package [14] takes care of the job handling and creates entries in the application database to track each job.

2.4. Implementation

The web application was developed in Python using the Django web framework. It is hosted on a Red Hat Enterprise Linux node running Apache with mod_wsgi[4] and uses a PostgreSQL database[5] (see Fig. 4).

Fits are performed on one of the SNS analysis nodes for the user. The Django remote submission package manages remote jobs using the Celery distributed task queue[6] and provides real-time monitoring of remote jobs and their associated logs. Celery uses message brokers to pass messages between the Django application and compute nodes. The Redis[7] in-memory data structure store is used as the message broker. Interactive plots are generated using Plotly[8] for Python.

2.5. Performance and user testing

The length of time needed to execute a REFL1D fit will depend on the complexity of the layer structure, the number of fit parameters, and the choice of fitting engine. As an example, fitting data containing 200 data points with a 6-layer model and 14 free parameters will complete in 5 to 8 s on a 32-core AMD machine with 264 GB of memory when using the Nelder–Mead algorithm [12]. The same fit using the DREAM [13] fitting engine needs about 4 min to complete. Benchmarking revealed that the handling of the remote submission added up to 1.5 s to the execution time. This includes the time needed to transfer data and launch the job.

The application was deployed at the LR instrument and given to our users to try, which at the time of writing numbered to about 15. The conclusion from feedback received was that it is indeed a convenient tool that helps scientists understand their data as they are acquiring it. Since the community of potential users of the application is not very large, the real impact of the tool will be judged by its adoption over time.

3. Illustrative Examples

The web application is best demonstrated with examples using LR data. In this section, we describe an example of a simple fit, and the co-refinement of two related data sets.

3.1. Single reflectivity model

The following example is a reflectivity measurement made on a thin film (see Fig. 5) with layers of copper and amorphous silicon

4 https://github.com/GrahamDumpleton/mod_wsgi/.
5 https://www.postgresql.org/.
6 http://www.celeryproject.org/.
7 https://redis.io/.
8 https://plot.ly/python/.

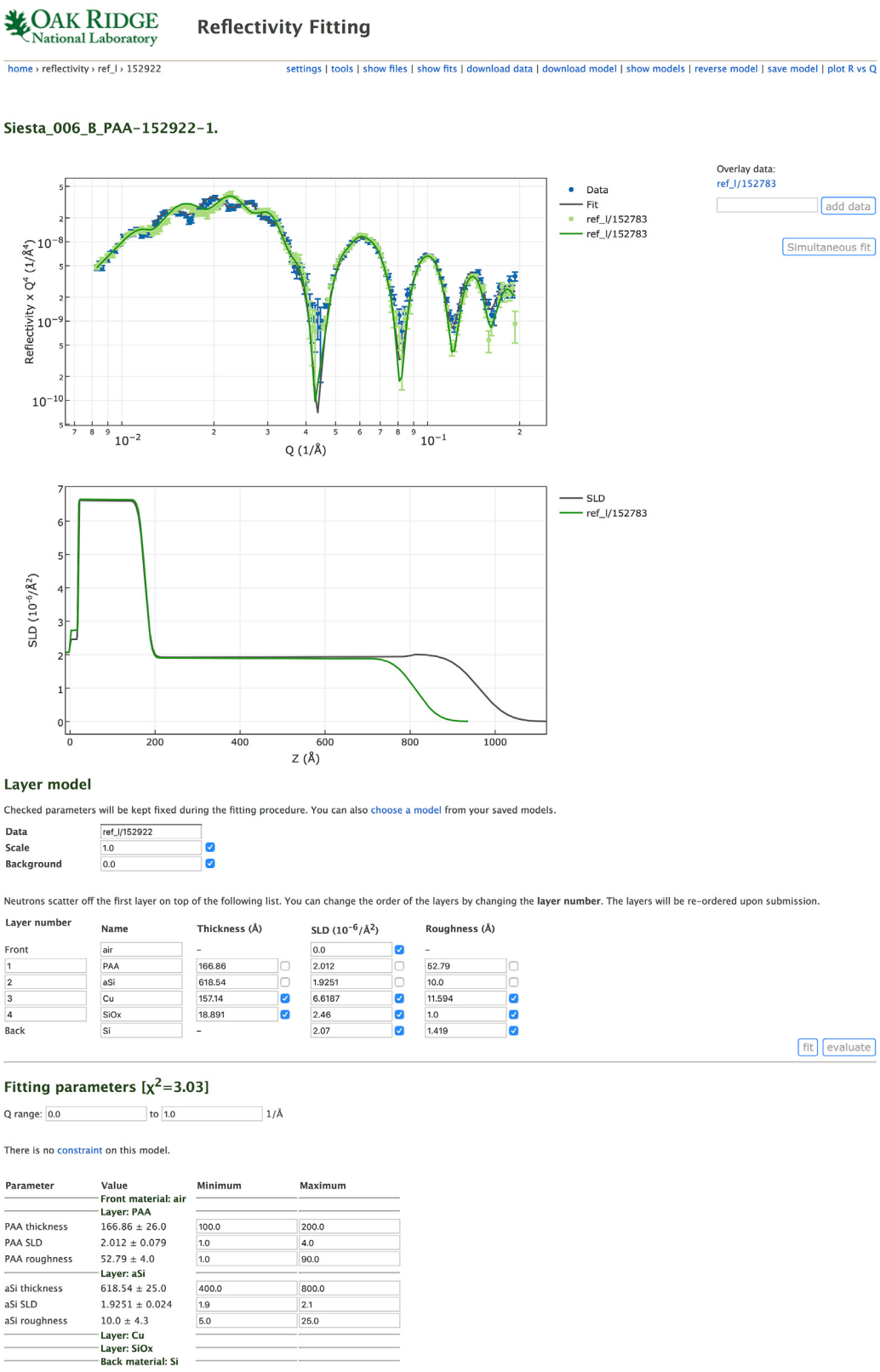

Fig. 6. Model fitting form for our example thin film with PAA (identified as run ref_l/152922 on this figure). The same film measured before the PAA deposition (labelled ref_l/152783) is also plotted for comparison.

deposited on a silicon substrate, topped by a spin-coated layer of polyacrylic acid (PAA). When modelling the structure of the film, we also allow for a native oxide layer on the surface of the silicon substrate. The reflectivity measurement was done at the LR with the incident beam coming through air and impinging on the PAA surface of the film. This data was automatically reduced by the SNS post-processing system so that it was immediately ready to fit. The model form for this data set is shown on Fig. 6. For each

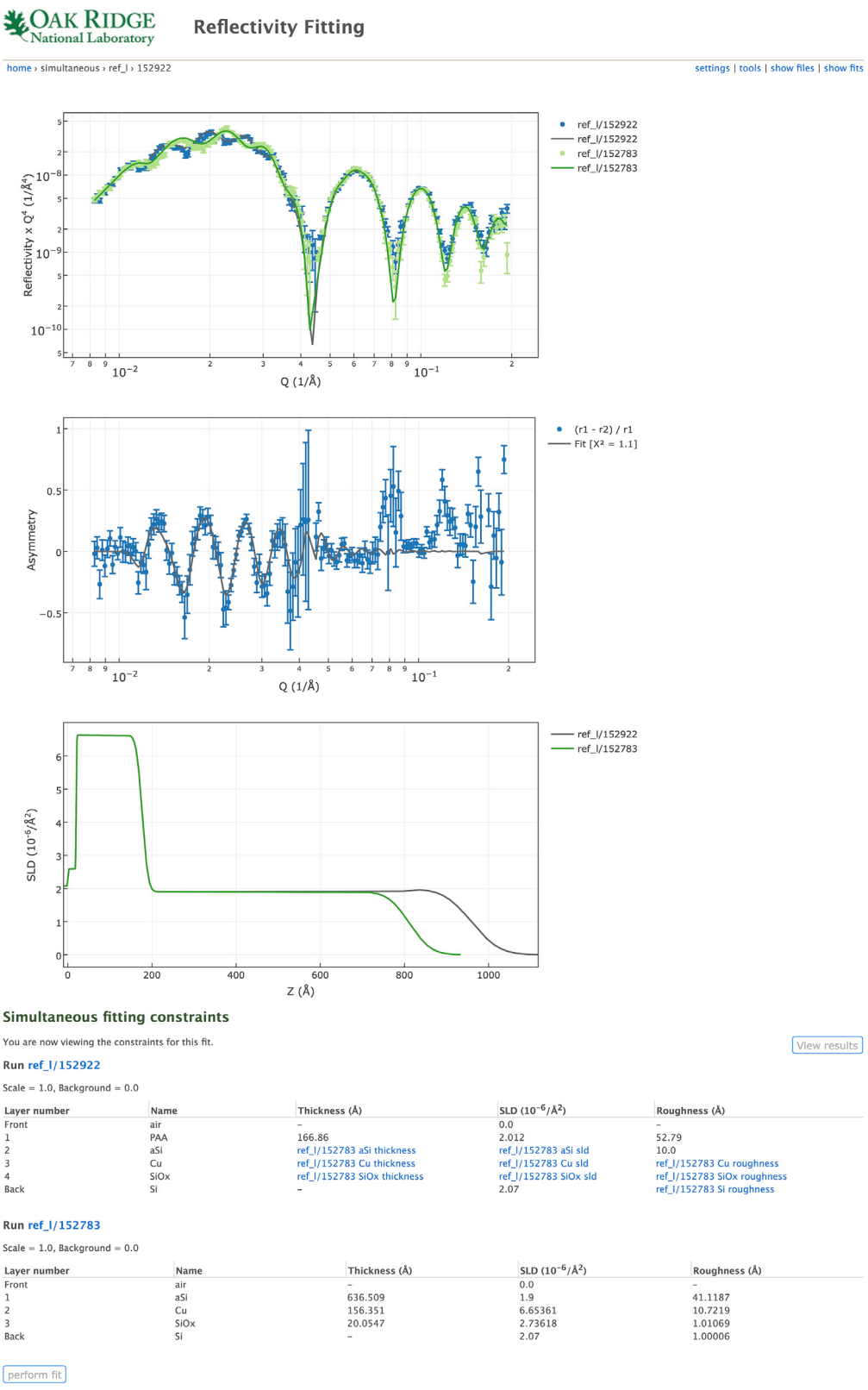

Fig. 7. The simultaneous fit page, showing the data and models of two data sets. The constraint matrix for the simultaneous fit job is shown at the bottom of the page. Constraints are added by dragging and dropping parameters between the two models. Tied parameters are indicated by a blue label. Clicking this label will remove the constraint and revert the parameter to its original value. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

selected model parameter, a value range can be entered in the bottom section of the form. The overall Q range used for the fit can also be modified.

The fit page notifies the user once the REFL1D fit is complete. Once the fit page is refreshed, the parameter values are updated in the model table and in the fit parameter range table, as shown

on Fig. 6. The R(Q) data plot is updated with the output model, and the SLD profile of this model is also shown. If the DREAM algorithm is used, the uncertainty on each fit parameter is displayed. Results from several fit jobs can be overlaid for comparison.

Varying parameters can be selected with using the check box next to each parameter. Once fit results are available, the uncertainty on each fit parameter is displayed in the "Fitting parameters" section if the chosen minimization technique estimates it.

3.2. Co-refinement

Sometimes two measurements taken with the same sample in two configurations can help extract better information about the structure of a film. In the example above, we have tried to measure the thicknesses, SLDs, and roughness values of our native oxide, copper, amorphous silicon, and PAA. To constrain the fit, we can also use a measurement done on the film before spin coating the PAA. Since depositing the PAA is not expected to change the rest of the film, we can impose that the unmodified layers have the same parameters in both models. We can fit the two data sets in a single minimization problem and tie together parameters that should remain the same before and after the PAA deposition. If we allow for the surface roughness parameter of the amorphous silicon to change after depositing of PAA, we are left with nine constraints between the two models. Using the simultaneous fitting form shown in Fig. 7, such constraints can be entered by dragging and dropping parameters between the two models.

Once constraints between models have been set, clicking the "perform fit" button submits the job for execution. The starting parameters for the models will be those from the existing individual fits. Once the simultaneous fit is completed, users can see their fit results on the simultaneous fit page. The fit results for R(Q) are plotted, and the model table of Fig. 7 now shows the output values of the fit parameters. Uncertainties are also shown if they are estimated by the minimization.

4. Impact

- This software provides a user interface for a powerful reflectivity fitting package (REFL1D), removing the need to install software or write Python scripts.
- This software allows facility users to start the analysis of their data as soon as it is acquired, enabling them to adjust their experiment planning as they go.
- By providing reflectivity fitting as a web application, this software will allow users to transition easily from the laboratory to their home institution.
- Although this software is now only in use at ORNL, it was designed to be used at any neutron or X-ray facility.

5. Conclusions

The number of experiments performed at the Liquids Reflectometer at the SNS demands a streamlining of data processing so that users leave the laboratory with quality data sets and a clear plan for how they are going to analysis them. An important part of this effort is to offer analysis tools that empower users to work independently regardless of whether they have a background in scientific computing or not. To address this need, we developed a user interface that simplifies the work of modelling reflectivity data. By deploying it as a web application, we have also greatly reduced the problem of installing software on various platforms, and we have made the transition from the laboratory to the home institution seamless. The reflectivity fitting application lets users manage their fit jobs, their models, and their data files. It provides a simple interface that lets users concentrate on the science by removing the need to install software or write Python scripts.

Acknowledgements

This research was sponsored by the Scientific User Facilities Division, Office of Basic Energy Sciences, US Department of Energy. A portion of this research used data from the Liquids Reflectometer at the Spallation Neutron Source. The REFL1D package used in this work was originally developed as part of the DANSE Project under NSF award DMR-0520547. Special thanks to Jim Browning and John Ankner for immediately starting to use the interface and giving us feedback. MD would like to thank collaborators Jim Browning, Josh Kim, Katie Browning and Gabe Veith for allowing us to use the data shown in this article. Finally, MD would also like to thank Paul Kienzle for developing and maintaining REFL1D, and for all the discussions on the topic of software and minimization. This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

Declarations of interest

none

References

[1] Mason TE, et al. The spallation neutron source in oak ridge: a powerful tool for materials research. Physica B 2006;385:955–60.
[2] Sivia DS. Elementary scattering theory: for X-ray and neutron users. Oxford University Press; 2011.
[3] Abeles F. Sur la propagation des ondes electromagnetiques dans les milieux stratifies. Ann Phys 1948;3:504–20.
[4] Parratt LG. Surface studies of solids by total reflection of x-rays. Phys Rev 1954;95:359–69.
[5] Arnold O, et al. Mantid—data analysis and visualization package for neutron scattering and SR experiments. Nucl Instrum Methods Phys Res A 2014;764:156–66.
[6] Nelson A. Co-refinement of multiple-contrast neutron/x-ray reflectivity data using MOTOFIT. J Appl Cryst 2006;39(2):273–6.
[7] Kienzle PA, Krycka JA, Patel N. Refl1D: Interactive depth profile modeler.
[8] Björck M, Andersson G. Genx: an extensible x-ray reflectivity refinement program utilizing differential evolution. J Appl Cryst 2007;40:1174–8.
[9] Shipman GM et al. Accelerating data acquisition, reduction, and analysis at the spallation neutron source. In: IEEE 10th international conference on eScience. 2014.
[10] Levenberg K. A method for the solution of certain non-linear problems in least squares. Quart Appl Math 1944;2:164–8.
[11] Marquardt DW. An algorithm for least-squares estimation of nonlinear parameters. J Soc Indust Appl Math 1963;11(2):431–41.
[12] Nelder JA, Mead R. A simplex method for function minimization. Computer J 1965;7(4):308–13.
[13] Vrugt JA, et al. Accelerating Markov chain Monte Carlo simulation by differential evolution with self-adaptive randomized subspace sampling. Int J Nonlin Sci Num 2009;10(3):273.
[14] Hobson TC, Doucet M, Ferraz Leal RM. Django remote submission. J Open Source Softw 2017;2(16).