# Authors

# Affiliations

[1] ISIS Neutron and Muon source, Rutherford Appleton Laboratory, Harwell Campus, OX11 0QX

[2]

# Abstract

# Introduction

# Methods

## Experiment Simulation

1. Define a refnx structure with thickness, SLD and roughness for each layer (can have multiple contrasts). Also, choose a measurement angle.
2. Load the directbeam\_wavelength.dat file which records incident neutrons as a function of wavelength for OFFSPEC (not binned). We can multiply this by a constant to effectively shorten the experimental counting time, and to give us the number of incident neutrons for a simulated experiment.
3. Adjust the flux with given measurement angle ( in radians) to account for slits.
4. Calculate Q value for each wavelength point using given measurement angle and formula
5. With NumPy, bin calculated q values in equally log-spaced bins (number of bins can be set to an arbitrary number of points < 460ish) using flux as weighting. Get the new bin centres from the returned bin edges.
6. With refnx, use the structure and given dq, scale and background (we’re using 2%, 1, 1e-6 respectively) to create a (ReflectModel) model.
7. Iterate over each q bin and for each bin:

* Calculate the model reflectivity scaled [0, 1] using refnx (Abeles matrix formalism?)
* Add background noise (background rate is 5e-7)
* Multiply reflectivity and flux to get counts for the bin
* The error bar in count-space is the square root of the count. Use the count and error bar as the mean and standard deviation of a normal distribution respectively to obtain a noisy count.
* Obtain the noisy reflectivity and associated reflectivity error by dividing both the noisy count and its error bar by the flux.

You now have a range of Q values and for each value you have a noisy reflectivity, reflectivity error and flux.

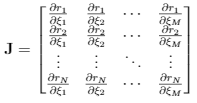
## Comparison

1. Provide a structure (such as one from the structures.py file) and simulate an experiment using the method above using a chosen angle, time and number of points.
2. Pass the resulting model(s) and dataset(s) to the Fitting class which will create a global objective that can be later fit.
3. Fit the objective using differential evolution / L-BFGS-B, MCMC or nested sampling. For MCMC, you can provide the number of samples for the initial burn-in and also the number of steps and thinning for the main sampling.
4. Each method will provide a fit of the data and an estimate of the parameter covariance matrix. MCMC and nested sampling will additionally provide a corner plot.

## Fisher Information

**Assumptions**: 1) We have used the fact that all of our errors are Poisson, and are treating every experimental data point independently as a way of analytically calculating the Fisher information, and therefore the variance in the model parameters. 2) We have the correct model before fitting.

1. Vary the parameters of interest of the model either from the experiment simulation or from the model describing a real dataset.
2. Create a refnx Objective using the model and simulated data and create a CurveFitter object using this Objective.
3. Use the data from the experiment simulation and a traditional fitting algorithm to find the optimal model parameter values, as judged by chi squared or similar. Fitting algorithms include L-BFGS-B, differential evolutions, MCMC and nested sampling. We used differential evolution as a global optimiser and then “polished” with the local optimiser L-BFGS-B. This intrinsically assumes we have the right model. From here on, we no longer need the data, since the model + Poisson statistics describe the data well enough. Therefore from here all mention of data point refers to the model.
4. We calculate for every model data point (e.g. every Q value which the data was taken for every condition), the gradient of the model reflectivity (*r*) with every model parameter (*ξ*). This is calculated using a two-point gradient with 0.5% step. This forms our Jacobian:



1. Using this Jacobian, and the diagonal matrix (**M**) of incident counts divided by the reflectivity (zeros for off-diagonals), we can calculate the Fisher information matrix (FIM):

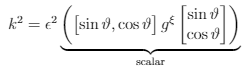


This matrix contains all of the information about parameter variances and covariances we just need to extract them.

1. For a single parameter’s error bar, ε, the Fisher information is just the multiplicative inverse of the variance, so we say

For multiple parameters, we take the square root of multiplicative inverse (reciprocal) of the diagonal elements of the Fisher information matrix.

1. If we want the covariance between two parameters then our equation is:



We can graphically solve (plot) this by inserting a number of theta values in and plotting the points

In summary, we will need to fit a dataset (complexity dependant) and then calculate the gradients of N (points) \* M (parameters), i.e. at least 2\*N\*M reflectivity points, but this is fast and fairly optimised; these calculations do not add any noticeable overhead to getting results in real time over the actual data fitting.

# Results and Discussion

* Comparison of different fitting methods (L-BFGS-B/differential evolution, MCMC and nested sampling)
* Relationship of traditional fitting errors and Fisher errors with increasing time constant. The time dependence for any one error bar (and indeed the covariance) is determined by sqrt(measurement time)
* Corner plots and Fisher confidence ellipses.
* Comparison of multiplicative inverse of Fisher information (variance) with variance in parameter estimation using traditional fitting for many fits (1000+).
* Fisher information approach applied to two or more real datasets.

# Future Work

* Experiment design: using Fisher information gain to influence choice of measurement angle and/or contrast choice.
* Using the Fisher information to determine how parameter uncertainties will change with time and estimate how much longer data needs to be counted in order to reduce the uncertainty in given parameters to a desired level.

# Conclusions

# Acknowledgements

# References

## refnx

“Nelson, A.R.J. & Prescott, S.W. (2019). J. Appl. Cryst. 52, <https://doi.org/10.1107/S1600576718017296>.”

## dynesty

**Code and Methods:**

Speagle (2019): ui.adsabs.harvard.edu/abs/2019arXiv190402180S

**Nested Sampling:**

Skilling (2004): ui.adsabs.harvard.edu/abs/2004AIPC..735..395S

Skilling (2006): projecteuclid.org/euclid.ba/1340370944

**Bounding Method:**

Feroz, Hobson & Bridges (2009): ui.adsabs.harvard.edu/abs/2009MNRAS.398.1601F

## corner.py

https://dx.doi.org/10.21105/joss.00024

## matplotlib

J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.

## NumPy

Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 0.1038/s41586-020-2649-2. (Publisher link).

## L-BFGS-B

To do