Spectroscopy Scientific and Technical Advisory Panel

Revision: v0.2.0

Instrument Data Scientist Report

Gregory Tucker

September 30, 2021

Investigations aimed at defining the experiment data format for ESS spectrometers are underway. Preliminary results suggest that storing events and minimal normalization information will be a viable solution. Forethought has been given to how a typical user will interact with the data transformation workflow and how graphical user interfaces may help.

1 Experimental event-data format

Modern neutron spectrometers record time and location information for individual neutron absorption events. This information can be used along with information from the primary spectrometer and sample orientation to define the momentum and energy transferred between the neutron and the sample in the scattering process. The ratio of the number of events with a specific momentum and energy transfer to the number of neutrons incident on the sample is the double differential cross section, $\frac{\partial^2\sigma}{\partial Q \partial E}$, which can be compared to theory. Determining $I(Q, E) \propto \frac{\partial^2\sigma}{\partial Q \partial E}$ is therefore a major goal of any inelastic neutron scattering experiment; and typically involves creating histograms of the events which are subsequently normalized to a measure of the incident neutron flux. The histograms necessitate a choice of bin size which, if too large, could muddle fine details in the spectra or, if too small, could produce overly-large data files which occupy more storage space and take longer to process than is stricly necessary. By keeping the event information until a user requests $I(Q, E)$ the instrumental resolution can be preserved without producing oversized data files.

A growing proportion of modern time-of-flight (ToF) neutron spectrometers make use of fixed-energy multiplexing. For fixed-incident-energy direct-geometry (DG) ToF spectrometers, the chosen method is often 'rep-rate' multiplication in which multiple monochromatic neutron pulses are directed onto the sample for each source pulse.

Multiple fixed-energies necessarily probe different, but perhaps overlapping, regions of reciprocal space. Due to the typically-large difference between the multiple-energies on current instruments and the inability to handle multiple energies with existing analysis software, the fixed-energies are treated separately in reduction and analysis, which can lead to a plethora of related data files. The resulting proliferation of histogram files is, at best, a challenging book-keeping exercise and, at worst, a hindrance to a user interested in extracting salient information from their neutron spectroscopy data.

The neutron spectrometers of the ESS will make extensive use of multiplexing to leverage the long-pulse structure of the proton source. It is likely that five to ten comparable fixed-energies will be utilized in most experiments and users may analyze the multiplexed data together or separately, depending on the scientific case and instrument configuration. Current histogram methods would produce thousands of individual data files for a typical BIFROST or CSPEC experiment, which will present an untennable challenge for most users if current techiques and formats are adhered to.

Taken together, the desires to delay histogramming and simplify data handling, present a clear case for a new data format to store transformed event information. The new format will allow for 1. flexible per-detector-element aggregation of events, e.g., within the instrumental timing resolution, 2. per-detector-element, fixed-energy, and energy-transfer corrections to the event intensity (and variance), 3. translation to formats used by the community, e.g., `Mantid MDEventWorkspace`, `Horace SQW`, `NeXuS NXS`, etc., 4. deduplication of parameter information and metadata through use of references, potentially leading to less storage utilization.

A ToF spectrometer detector which is held fixed during a data acquisition collects events along a kinematically-constrained trajectory through (Q, E). Similar to histogram spectra file formats, the new format could leverage the trajectories to maintain a hierarchy between each detector-element and its events. Such a hierarchy, however, is likely to lead to performance issues when collecting $I(Q, E)$, especially in a distributed computing environment.

A more-parallelizable approach would store events in a coarse `scipp` grid, allowing the possibility of locating grid cells on different compute nodes and simplifying dis-

tributed analysis. This has the disadvantage of severing the implicit connection between event and incident-spectrometer state, making it more complex to calculate the incident-flux normalization. Currently, investigations are ongoing to determine the feasibility of this approach.

The new format is partly-defined and has benefitted from feedback from external inelastic neutron scattering data experts. Their feeback will be solicited again as the format specification becomes more concrete.

2 Inverse scattering problem

In the laboratory frame, a spectrometer with fixed incident or final neutron energy can only probe $I(Q, E)$ on a kinematically constrained 3-D manifold. For the standard choice of axes, with the incident neutron beam along $\hat{z}$ and $\hat{y}$ pointing vertically-up, that manifold is given by

$$Q_{lab} = k_\mathrm{i} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} - k_\mathrm{f} \begin{bmatrix} \cos\phi \sin 2\theta \\ \sin\phi \sin 2\theta \\ \cos 2\theta \end{bmatrix} \tag{1}$$

where 2θ is both the angle between k_i and k_f as well as the polar angle of a spherical coordinate system, and the azimuthal angle ϕ is zero for $k_\mathrm{f} \parallel \hat{x}$ and $\pi/2$ for $k_\mathrm{f} \parallel \hat{y}$, with the last dimension along energy-transfer $\propto k_\mathrm{i}^2 - k_\mathrm{f}^2$.

Instruments add one or more sample degree of freedom, allowing more-flexible selection of (Q, E) in the sample coordinate system. Often this comes in the form of a sample-stage rotation, ψ, installed such that it points along $\hat{y}$. With such a setup, the accessible reciprocal space becomes a torus for fixed incident and final energies. In the sample stage coordinate system, Q is given by

$$Q = k_\mathrm{i} \begin{bmatrix} -\sin\psi \\ 0 \\ \cos\psi \end{bmatrix} - k_\mathrm{f} \begin{bmatrix} \cos\psi\cos\phi\sin 2\theta - \sin\psi\cos 2\theta \\ \sin\phi\sin 2\theta \\ \sin\psi\cos\phi\sin 2\theta + \cos\psi\cos 2\theta \end{bmatrix} \tag{2}$$

The direction to every detector in the laboratory coordinate system and the stage orientation will be known for each neutron event, allowing unambiguous conversion from $(2\theta, \psi, \phi, k_\mathrm{i}, k_\mathrm{f})$ to (Q, E) on a per-event level. When a user then requests $I(Q, E)$, collecting the events within the (Q, E) region is straightforward but determining the normalisation is not. For this, we must invert the problem and determine all $(2\theta, \psi, \phi, k_\mathrm{i}, k_\mathrm{f})$ that correspond to a given (Q, E), to then identify which detectors and aquisition periods have $(2\theta, \phi)$ and $(\psi, k_\mathrm{i}, k_\mathrm{f})$, respectively, within the (Q, E) region.

Since a user will typically not limit their desired $I(Q, E)$ to only that which has been or can be measured, the inverse algorithm must ensure that only physically accessible (Q, E) regions are returned. This is accomplished by requiring that a user define their region of interest as one or more 4-D (Q, E) bins (e.g., as a cut); these bins are then replaced by their union with the 4-D (Q, k_i) or (Q, k_f) torus, and the resulting 4-polytope regions are used to define extents in $(2\theta, \phi, \psi)$ which envelop each bin.

A prototype `python` module is being developed to perform the inversion for an arbitrary spectrometer with a number of fixed neutron energies, to use the resultant angle regions to select detector-acquisition pairs, and to determine the normalization constant for user-defined (Q, E) bins from a pre-sample beam monitor spectrum. The prototype makes use of `scipp` data structures and is partially implemented in `pybind11` wrapped C++ to hasten the integration of the monitor counts. If the method works and the prototype is sufficiently quick, it will provide concrete support for the non-hierarchical transformed event data format discussed above.

3 Event transformation configuration

Converting raw event data from ESS to a more-usable format will be done in `python` by a script or module built on top of `scipp`. For the spectroscopy instruments, most of the transformation will be the same for every experiment and so could be accomplished by a top-level gateway function which accepts a small number of experiment parameters and user options. While some users will be happy to set these parameters in a `python` script, others will prefer a graphical user interface (GUI) approach.

The gateway transformation function will be written to take a validated set of input parameters in the form of a `python` a class. To please everyone, the validated input will be constructable in a straight-forward way from within a script, or through interaction with graphical interface built on top of `jupyter` widgets.

The widget-based interface will likely present a tabbed structure to the user, with each tab containing settings for different aspects of the data transformation. As an example, one tab for a DG ToF spectrometer would contain settings which control how chopper and beam monitor information is used in the transformation. When provided with some preliminary instrument settings, from a raw data file or the user, the expected monochromatic incident pulse times could be presented and compared with one or more monitor spectra. A user could then select which peaks to use to determine incident neutron energies and, on a detailed sub-tab, control how each peak is fit.

By using a class to define the transformation parameters, a sensible default can be provided to the user to accept or modify as they see fit. By validating the parameters before starting the transformation, a user can be made aware of potential pitfalls at the start of a possibly-long-running process. By using `jupyter` widgets for the graphical elements, the interface can be tailored to changing use cases more easily than a traditonal stand-alone GUI.