Spectroscopy Scientific and Technical Advisory Panel

Instrument Data Scientist Report

Gregory Tucker

April 18, 2023

1 Pixel mapping

The last ESS Spectroscopy STAP Report [1] included a request for information about pixel mapping plans.

Comprehensive details are available from the Experiment Control and Data Curation (ECDC) Instrument Status Overview page within ESS Confluence. The CSPEC documentation there does not yet reflect the change to ^{3}He detectors, but should instead be similar to that for BIFROST.

A high-level summary of the planned pixelation scheme follows: 1. signals from the ends of a wire, A and B, are digitized by one of series of Front End Nodes; 2. a Readout Master collects the digitized signals, bundles them into packets along with timing information, and sends the packets to a server room located in the Central Utility Building, H01, on the ESS site; 3. an instance of the Event Formation Unit (EFU) software uses linear charge division to pixelate each wire.

Conversion to pixels from the continuous charge-division signal, $x = A/(A + B)$, will require two values per tube to identify its ends. The EFU extracts x between the specified end-points and optionally applies a nonlinear correction with calibrated polynomial coefficients before subdividing it into a configurable number of pixels. The number of EFU pixels should match the position resolution of the tube.

Since the ESS data transformation for spectroscopy will keep neutron event data and avoid unnecessarily producing histograms, combining pixels to, e.g., match instrumental resolution conditions, will only be performed preceding the final histogram-creating step.

2 Project planning

Since the October 2022 STAP meeting, extra emphasis has been placed on project planning at the IDS level. As with other DMSC projects, a JIRA project, DMSC Spectroscopy, has been created to track the DMSC deliverables for BIFROST and CSPEC. Currently efforts are focused on aligning the project with requirements as seen by groups within DMSC as well as the BIFROST and CSPEC teams.

3 Simulations

Efforts to integrate McStas simulations into the ESS event pipeline continue [2]. Since the last report:

- The detector component has been extended to support other arrangements of multiple ^{3}He tubes, like the constant-radius MultiTube arrays now-expected for CSPEC. Figure 1 shows multiple tube arrays now possible with the McStas component.
- The readout master component has been refactored. The changes have made installing and using the shared library and McStas component easier on new systems, and allow for automatic EFU control even in parallel MPI runtime environments.
- The full BIFROST spectrometer can now be used via McStasScript to simulate neutrons from the ESS source through to pixel identification in the EFU.
- BIFROST primary spectrometer Monte Carlo Particle List (MCPL) files exist for a small subset of possible chopper settings, which can be used to significantly improve the simulated instrument throughput with, e.g., complex Union samples.

Planned next-steps include:

- `DMSCSPEC-8` Complete the readout chain by getting simulated events from the EFU through to a NeXus file.
- `DMSCSPEC-9` Simulate a BIFROST experimenti, including produing at least one (Q, E) map.
- `DMSCSPEC-10` Pass the simulated experiment data through the data transformation workflow.

4 Flexible user tools

Some tasks are made easier through graphical user interfaces (GUI) to scripts or other routines. A user may request a tool to help carry out a measurement, which must be flexible enough to support their specific use case, and must be available quickly if it is to be useful.

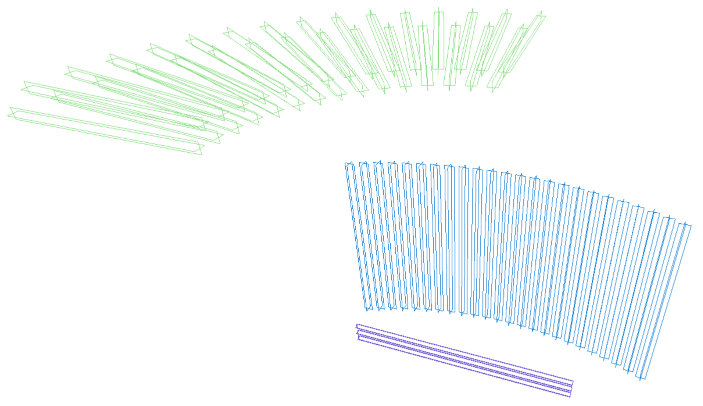
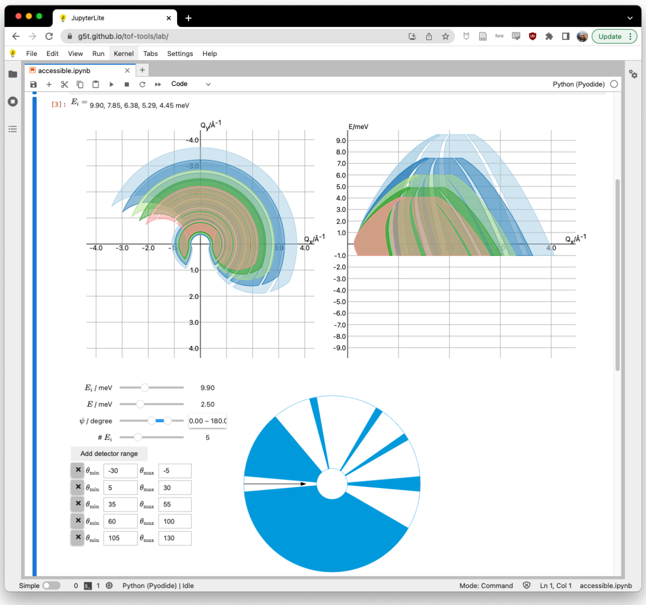


Figure 1: Multiple tube arrays in a test McStas instrument, including a BIFROST triplet (purple); a curved array (blue), like those planned for CSPEC but much shorter to fit in this image; a double-layer fan array (green), like that used on CAMEA.



```
[3]: E_i = 9.90, 7.85, 6.38, 5.29, 4.45 meV
```

E_i / meV 9.90
E / meV 2.50
ψ / degree 0.00 – 180.0
E_i 5

$\theta_{\min}$	-30	$\theta_{\max}$	-5
$\theta_{\min}$	5	$\theta_{\max}$	30
$\theta_{\min}$	35	$\theta_{\max}$	55
$\theta_{\min}$	60	$\theta_{\max}$	100
$\theta_{\min}$	105	$\theta_{\max}$	130

Figure 2: The magnet opening-angle visualization tool, running in a Jupyterlite notebook.

Python is the language of choice for all ESS software interfaces since it is free, open source, and has an extensive catalog of community-provided modules to augment its capabilities. These qualities make it well suited to rapid prototyping and development, which in turn makes it an ideal language for producing user-facing tools.

Thanks to the vast module catalog, there are many options for graphical interfaces in Python. One option that requires little development effort to use in simple GUIs is Jupyter Widgets which provide graphical interfaces to Jupyter notebooks. Previously the use of voila has been discussed for stand-alone GUIs [3].

To make any GUI tool written in Python available to users we can 1. distribute installers, 2. host it on ESS hardware, 3. distribute source files, or 4. make use of client-side web browser based interfaces.

Producing and distributing installers for any application can be resource intensive. Such an investment may be worthwhile for large stable programs or performance sensitive applications, but is likely wasteful for one-off tools.

Hosting on ESS hardware can reduce a large part of the development effort compared to distributing installers; mainly from having machine control and only targeting a single machine (or multiple identical machines). Still, configuring and running the hosting system requires relatively high resource use, which may not be suitable for in-development tools.

Distributing the source files which constitute a Python tool confers all of the advantages of open-source software. There may exist a high entry-barrier, however, since Python's modularity can make it challenging to use code written on another machine due to differing computing environments. Semi-automatic configuration of computing environments is possible via a variety of tools, notably mamba, but as they require installation and subsequent command-line use they may not be suitable for all users.

Web browser based interfaces have the significant advantage that they run on top of software which is almost-certainly already present on a user's computer. Through WebAssembly and the Pyodide Python distribution it is possible to run Python-based tools on a user's machine with no user-configuration and near-native performace. It is even possible to run notebooks in a browser via Jupyterlite. For simple GUI tools this can allow for both rapid development and ease of use, even for novice users.

Recently, to support design of a new magnet a tool was required to help visualize the effect different opening-angles would have on accessible reciprocal space for a direct-geometry spectrometer (DGS) like CSPEC. The intended audience for the tool were the engineers, technicians, and scientists involved in setting the magnet design criteria, but a similar tool could prove useful for *users* of a magnet on a DGS as well. The source code of the tool is hosted on Github which also provides free webhosting for the Jupyterlite interface. The tool running after interacting with some of the GUI elements is shown in figure 2.

References

[1] G. Ehlers, *ESS Spectroscopy STAP Report*, tech. rep. (2022-10).

[2] G. Tucker, *IDS Report*, tech. rep. (2022-10).

[3] G. Tucker, *IDS Report*, tech. rep. (2022-04).