# Excitations Data Analysis Training Course – Euphonic

## Euphonic Introduction

Euphonic is a package to visualize phonons from a force constants matrix. It can load force constants from several common modelling codes and produce publication quality outputs from them. Euphonic is mostly written in Python and much of this session will be spent learning to drive it through Python. However, it also plays nicely with Horace within Matlab and we will show you how to drive it here as well.

## Euphonic, driving from Python

The worksheet for running Euphonic from Python is available on Github at <https://github.com/pace-neutrons/edatc/blob/main/worksheets/edatc21_worksheet_09_euphonic_python.ipynb>, it can also be found on IDAaaS at /mnt/ceph/auxiliary/excitations/edatc/edatc21\_worksheet\_09\_euphonic\_python.ipynb. It is a Jupyter notebook, first make sure you have a copy saved to your home directory so you can save any changes. To open a Jupyter notebook on IDAaaS, go to “Applications” (bottom left) → Software → Jupyter notebook. A web browser should open. Navigate to the worksheet file and click to open it, then follow the worksheet.

## Euphonic with Horace Setup

The Horace-Euphonic Interface docs are available at <https://horace-euphonic-interface.readthedocs.io>

To use Euphonic with Horace, add the following to your startup.m file:

% Tells Matlab to use the Python version with Euphonic installed

pyenv('Version', '/usr/local/virtualenvs/euphonicenv/bin/python3');

% Prevents library clashes with Matlab and Numpy

py.sys.setdlopenflags(int32(10));

Now make sure you have the correct version of Horace-Euphonic-Interface installed, a version was released recently so it may not have appeared on IDAaaS yet. Open Matlab, and click HOME -> Add-Ons -> Manage Add-Ons. You should see **horace\_euphonic\_interface** **version 0.3.0** is installed. If you have a lower version, go to <https://github.com/pace-neutrons/horace-euphonic-interface/releases/tag/v0.3.0>, download the horace\_euphonic\_interface.mltbx file from the release assets and open it in Matlab. This should install v0.3.0.

## Euphonic with Horace

We have now seen the power of euphonic to produce phonon dispersions and neutron weight them. This is great for planning an experiment but what if we already have our data? We are now going to drive euphonic through Horace and use it to see if the quartz simulations you have been looking at have any resemblance to reality. To facilitate this we have collected a dataset on a large single crystal of α-quartz and have built the SQW file for you. You can access it at /mnt/ceph/auxiliary/excitations/edatc/quartz\_5K.sqw.

To begin with, **take some cuts through the dataset**. Quartz is trigonal (P3121) and in this case it was aligned with the in the horizontal scattering plane (note that and are not perpendicular to each other). Try taking some cuts through the data, making sure that at least one of them is either at or is along the direction. Now, we are going to try simulating these cuts. For the moment please use the ‘-nopix’ option in your cuts!!! Two or three nice colour maps should be more than sufficient.

We are now going to setup euphonic. First, we need to load the force constant model

You will need the quartz force constants quartz.castep\_bin which you can find again in the /mnt/ceph/auxiliary/excitations/edatc/ folder. We load them with

fc = euphonic.ForceConstants.from\_castep('fname')

This done, we set up a crystal object

% Set up model

coh\_model = euphonic.CoherentCrystal(...

fc, 'debye\_waller\_grid', [6 6 6],'temperature', 5, ...

'asr', 'reciprocal','use\_c', true);

Here we give it the density of points for the debye-waller factor, temperature and acoustic sum rule. To produce an output we simply need

cut\_sim = disp2sqw\_eval(...

cut1, @coh\_model.horace\_disp, {scale\_factor}, effective\_fwhm);

plot(cut\_sim);

**Choose whichever value you want/feel is appropriate for scale\_factor and the FWHM. How do your cuts look? Look especially carefully at cuts that have values of , are they correct?**

Of course they are not, quartz has no centre of inversion and so it is possible to put the crystal (or calculate it) upside down. When we setup the CoherentCrystal we can add an extra term to define a translation matrix. This is handy as calculations are often performed in the primitive cell (for convenience) while the experiment will use a larger but simpler to visualize cell. This matrix lets us convert one into the other. For this case we need to use a matrix of the form . Using the help put this in and try the calculation again. Is it better?

By eye the agreement should be fairly strong if you have done everything correctly. Colour plots though can be misleading and it is always hard to compare individual points to each other. **Now, try taking some 1D cuts through the data. How does it look?** Depending on your cut sizes and FWHM it might look really good however, it might also look poor with some peaks much broader in reality than expected and perhaps worse at higher energy transfers.

There are two things happening here, first, the cuts you have made have a finite width, however euphonic knows nothing of them as we did not preserve the pixel information. This width will broaden dispersive excitations and if we want it to be captured in our model we will need to give euphonic all the pixel information. It can handle this but even a small cut can be 100,000 pixels so please do not start off trying to model a large colour plot, stick to the line plots. **Now, repeat your line cut with the pixel information and rerun the simulation. Is it better?**

The final thing you might need is to properly capture the resolution function. The resolution on our instruments is not the same as a function of energy transfer (or scattering angle). The syntax for this is shown below

is\_crystal = true; xgeom = [0,0,1]; ygeom = [0,1,0]; shape = 'cuboid'; shape\_pars = [0.01,0.05,0.01];

cut1 = set\_sample(cut1, IX\_sample(is\_crystal, xgeom, ygeom, shape, shape\_pars));

ei = 45.12; freq = 350; chopper = 'g';

cut1 = set\_instrument(cut1, merlin\_instrument(ei, freq, chopper));

intrinsic\_fwhm = 0.1;

kk = tobyfit(cut1);

kk = kk.set\_fun(@disp2sqw, {@coh\_model.horace\_disp, {[tt scale\_factor]}, [intrinsic\_fwhm]});

kk = kk.set\_options('selected',false)

sim = kk.simulate('fore');

keep\_figure;

plot(cut1); pl(sim)