horace-euphonic-interface

Feb 16, 2023

Contents

This is a simple interface to allow simulation of inelastic neutron scattering data from phonons in Horace using Euphonic. This is done using Horace simulation functions.

CHAPTER 1

General Installation

1.1 1. Prerequisites

Ensure you have both Horace, Euphonic and the Python package `psutil` installed:

- Horace docs
- Euphonic docs
- psutil

1.2 2. Set up Python in Matlab

The Python executable that you installed Euphonic with needs to be specified in MATLAB. You can find the executable location in Python with:

```
>>> import sys
>>> print(sys.executable)
```

You can then set this executable in MATLAB (2019b or later) using:

```
>> pyenv('Version', '/path/to/python')
```

Or in MATLAB 2019a or earlier:

```
>> pyversion('/path/to/python')
```

Note: The Python version used in Matlab can only be changed if it has not yet been loaded. If you have already installed Horace-Euphonic-Interface, Python might be automatically loaded on startup. To prevent this, disable Horace-Euphonic-Interface first in **Add-Ons > Manage Add-Ons** then click the ⋮ symbol to the right of the add-on to

bring up the settings, and untick the **Enabled** box, then restart Matlab. Python will no longer be loaded. Remember to re-enable Horace-Euphonic-Interface afterwards.

1.3 3. Download and Install

Latest version (recommended)

Horace-Euphonic-Interface is packaged as a Matlab toolbox (`.mltbx`), which allows easy installation from a single file as a Matlab Add-On. In Matlab, go to the **Home** tab, and in the **Environment** section, click **Add-Ons**, and then **Get Add-Ons**. Search for **horace-euphonic-interface**, select it and then click **Add > Add to MATLAB**. That's it!

See here for more information on Matlab Add-Ons.

Older versions

The `.mltbx` file for each release is also available at https://github.com/pace-neutrons/horace-euphonic-interface/releases. Open the `.mltbx` file in Matlab and it should automatically be installed.

1.4 4. Test installation

To test everything has been installed ok, try:

```
>> euphonic.ForceConstants
```

If everything worked, you should see the Python type description `<class 'euphonic.force_constants.ForceConstants'>`.

CHAPTER 2

IDAaaS Installation

Euphonic is installed in a Python virtual environment at `/opt/euphonic` and `horace-euphonic-interface` is already installed in Matlab as an add-on. To use Horace-Euphonic-Interface, you just have to make sure the Python version you are using in Matlab has a compatible version of Euphonic installed. To avoid Python/Matlab library collisions, you also need to set some library loading flags and import Euphonic as soon as Matlab is started. To do this, just add the following to your `startup.m`:

```
pyenv('Version', '/opt/euphonic/bin/python3');
py.sys.setdlopenflags(int32(10));
py.importlib.import_module('euphonic');
```

CHAPTER 3

Usage

3.1 Quick Guide

To view the available functions and classes, try:

```
import euphonic.help
help euphonic
import euphonic.doc
doc euphonic
```

Because Euphonic is actually a Python program which is wrapped to be used in Matlab, its online documentation is in Python, and the default Matlab `help` function is not able to read this. The `import` commands above instead overrides the default `help` and `doc` functions to use the Python help system for Euphonic functions instead. (You can also use `import euphonic.help euphonic.doc` instead of using two separate `import` commands.)

1. Read force constants

First, the force constants must be read. The usage is very similar to Euphonic, for example to read a CASTEP `.castep_bin` file:

```
fc = euphonic.ForceConstants.from_castep('quartz.castep_bin')
```

Or, to read from Phonopy files:

```
fc = euphonic.ForceConstants.from_phonopy('path', 'quartz', ...
                                          'summary_name', 'phonopy.yaml')
```

To get help on these functions type:

```
help euphonic.ForceConstants.from_castep
help euphonic.ForceConstants.from_phonopy
help euphonic.ForceConstants.from_json_file
help euphonic.ForceConstants.from_dict
```

You can also type `help euphonic` or `doc euphonic` and follow the hyperlinks.

Note that in Matlab usage, a Matlab `struct` should be used for the dictionary in the `from_dict` function.

2. Set up model

Next, the model must be set up. Currently, only the `CoherentCrystal` model is available. The force constants must be passed in, then any other optional parameters. For example:

```
coh_model = euphonic.CoherentCrystal(...
   fc, ...
   'conversion_mat', [1 0 0; 0 1 0; 0 0 -1],
   'debye_waller_grid', [6 6 6], ...
   'temperature', 100, ...
   'asr', 'reciprocal', ...
   'use_c', true);
```

To see all the available optional parameters, try one of:

```
help euphonic.CoherentCrystal
doc euphonic.CoherentCrystal
```

Note: conversion_mat

Pay particular attention to this parameter, this is a 3x3 matrix to convert from the q-points in Horace to the q-points in the modelling code. This will be required if you've used a different unit cell convention/orientation in Horace and your modelling code, and will depend on the cells chosen. If set incorrectly, the results will not make sense (or worse, may happen to make sense at first in certain cuts due to symmetry, but give incorrect results in other cuts later on!)

---

3. Simulate cut

In Horace, the `disp2sqw_eval` simulation function is used to simulate experimental data with Euphonic. This requires a function handle, which is provided by `CoherentCrystal.horace_disp`. Help on the `horace_disp` function can be seen by with `help euphonic.CoherentCrystal.horace_disp`.

`horace_disp` has 2 optional arguments, `intensity_scale` and `frequency_scale` which can be used to multiply the intensities and frequencies by a constant scaling factor. These can be used as positional arguments (note they must be in the correct order). For example:

```
intensity_scale = 100;
frequency_scale = 0.9
effective_fwhm = 1;

cut_sim = disp2sqw_eval(cut, @coh_model.horace_disp, [intensity_scale, frequency_
↪scale], effective_fwhm);
```

They can also be used as named arguments, for example:

```
iscale = 100;
fscale = 0.9
effective_fwhm = 1;

cut_sim = disp2sqw_eval(cut, @coh_model.horace_disp, {'intensity_scale', iscale,
↪'frequency_scale', fscale}, effective_fwhm);
```

If the scaling parameters are to be used in fitting (e.g. in Multifit or Tobyfit), they must be used as positional arguments, for example:

```
iscale = 100;
fcale = 0.9
fwhm = 1;

kk = multifit_sqw(cut);
kk = kk.set_fun(@disp2sqw, {@coh_model.horace_disp, [iscale, fscale], fwhm});
cut_sim = kk.fit();
```

3.2 Full Example

An example script simulating a simple cut is below:

```
% Read in experimental cut
cut = read_horace('quartz.d2d');

% Read force constants
fc = euphonic.ForceConstants.from_castep('quartz.castep_bin')

% Set up model
coh_model = euphonic.CoherentCrystal(...
   fc, ...
   'conversion_mat', [1 0 0; 0 1 0; 0 0 -1],
   'debye_waller_grid', [6 6 6], ...
   'temperature', 100, ...
   'dipole_parameter', 0.75, ...
   'asr', 'reciprocal', ...
   'use_c', true, ...
   'n_threads', 4);


% Simulate
intensity_scale = 100;
frequency_scale = 0.9;
effective_fwhm = 1;
cut_sim = disp2sqw_eval(...
   cut, @coh_model.horace_disp, {'intensity_scale', intensity_scale, 'frequency_scale
↪', frequency_scale}, effective_fwhm);

% Plot
plot(cut_sim);
```

3.3 Faster Interpolation with Brille

From version `1.2.0`, Euphonic can use the Brille library to perform linear (rather than Fourier) interpolation of phonon frequencies and eigenvectors. Linear interpolation may be less accurate than the Fourier interpolation performed by `ForceConstants`, but should be faster for large unit cells, particularly those that require the expensive dipole correction calculation. You should test this on your particular machine and material first to see if it provides a performance benefit. For more details on how this works, and what the various options mean, see the Euphonic BrilleInterpolator docs

A `BrilleInterpolator` object can be created from a `ForceConstants` object, and can then be used in `horace_disp` just like `ForceConstants`. A full example showing this is below:

```
% Read in experimental cut
cut = read_horace('quartz.d2d');

% Read force constants
fc = euphonic.ForceConstants.from_castep('quartz.castep_bin')

% Create BrilleInterpolator from the force constants
% Note that any arguments you would pass to
% ForceConstants.calculate_qpoint_phonon_modes are passed here as
% 'interpolation_kwargs' to be used when creating the Brille grid
bri = euphonic.BrilleInterpolator.from_force_constants(...
    fc, ...
    'grid_npts', 5000, ...
    'interpolation_kwargs', struct('dipole_parameter', 0.75, ...
                                   'use_c', true, ...
                                   'n_threads', 4, ...));

% Set up model
% Pass in BrilleInterpolator here instead of force constants
coh_model = euphonic.CoherentCrystal(...
   bri, ...
   'conversion_mat', [1 0 0; 0 1 0; 0 0 -1],
   'debye_waller_grid', [6 6 6], ...
   'temperature', 100, ...
   'useparallel', true, ...
   'threads', 4);

% Simulate
intensity_scale = 100;
frequency_scale = 0.9;
effective_fwhm = 1;
cut_sim = disp2sqw_eval(...
   cut, @coh_model.horace_disp, {'intensity_scale', intensity_scale, 'frequency_scale
↪', frequency_scale}, effective_fwhm);

% Plot
plot(cut_sim);
```

3.4 Performance and Memory Tips

The following are a few tips to help make sure you have the optimum settings for the type of work you're doing and your computing resources.

Number of Threads

Euphonic will make use of the C Extension by default, and automatically choose the number of threads using the Python function `multiprocessing.cpu_count`. However, this can be overridden by the `use_c` and `n_threads` arguments to `CoherentCrystal` (or `useparallel` and `threads` if you're using `BrilleInterpolator`). Ensure that if these arguments are used, they are appropriate to the computing resource you are using. Generally `use_c` should be `true` and `n_threads` should be the same as the number of cores (or the same as the number of logical cores if your system has hyperthreading).

Chunking

The phonon eigenvectors, which are an intermediate step in calculating the scattering intensities, are particularly memory intensive, requiring $18n^2$ floating point numbers per q-point, where n is the number of atoms in the unit

cell of your calculation. To reduce memory consumption, the intensity calculation can be chunked with the `chunk` argument to `CoherentCrystal`. This defines the number of q-points that are calculated at once. Generally it is best to use the largest chunk you can get away with based on the amount of memory available and the number of atoms in the unit cell, but this depends on the system architecture. If no `chunk` is provided to `CoherentCrystal`, a recommended chunk size will automatically be set depending on the available memory. This estimate is conservative to cover most use-cases and avoid running out of memory (which can cause mysterious crashes!). Therefore it is possible on some systems using a higher chunk size might be slightly more efficient, but it is a good starting estimate.

Reducing Q-points

The most time consuming part of the intensity calculation is the calculation of the phonon frequencies and eigenvectors. Fortunately these are periodic from one Brillouin Zone to the next. If the `reduce_qpts` argument to `CoherentCrystal` is set to `true` (this is the default), Euphonic will look for q-points in other Brillouin Zones e.g. if there are points `[0.5, 0.5, 0.]` and `[1.5, 1.5, 2.]` Euphonic will only calculate frequencies/eigenvectors for one of those q-points. However, there is some overhead to finding these q-points, and Euphonic will only look at q-points in the same chunk, so setting `reduce_qpts` to `true` will not always be beneficial. It is most likely to be useful if you are simulating a `dnd` object with commensurate bin spacing. If you are simulating per pixel, or are using Tobyfit to apply resolution convolution, the q-points are likely to be irregular so `reduce_qpts` may not provide a benefit.

Dipole Parameter

If your simulation cell is polar (i.e. you have Born charges and dielectric permittivity tensors), there is an extra computationally expensive correction that must be applied when calculating the phonon frequencies and eigenvectors. This correction is based on an Ewald sum, so includes both real space and reciprocal space sums. The limit of these sums can be tuned so that the optimum balance of real and reciprocal space terms is used to reduce the computation required. This can be done with the `dipole_parameter` argument to `CoherentCrystal`. Euphonic has a Python command-line tool, euphonic-optimise-dipole-parameter which can help to tune this argument.

Use Linear Interpolation with Brille

See *Faster Interpolation with Brille*

3.4.1 Release Notes

Unreleased

v1.1.0

- New Features:
  - You can now perform linear interpolation of phonon frequencies and eigenvectors with the Brille library using the new `euphonic.brille.BrilleInterpolator` object. This should provide performance improvements for large unit cells which require the dipole correction.
- Dependency changes:
  - Euphonic version dependency increased from >=0.6.0 to >=1.2.0

v1.0.0

- Changes:
  - `psutil` has been added as a Python dependency for automatic chunking
- Improvements:

  - `n_threads` will now automatically be converted to an integer (using e.g. `int32(4)` is no longer needed)
  - Warn rather than error if the incorrect Python version or Python module versions are used
  - If `chunk` isn't provided to `euphonic.CoherentCrystal`, a recommended chunk size will be set depending on the available memory
  - `install_python_modules.m` will now install the latest version of dependencies instead of the oldest
- Bug fixes:
  - In some distributions of MATLAB the automatic conversion from Numpy `ndarray` using MATLAB's `double` does not work. If it fails, convert to a regular `py.array` first, this should be more reliable.
  - In some MATLAB versions `py.sys.executable` actually points to the MATLAB executable so the `install_python_modules` script wouldn't work. This has been fixed.

v0.3.3

- Improvements:
  - A `CITATION.cff` file has been created and is now bundled with the `.mltbx` distribution
  - The `LICENSE` file is now bundled with the `.mltbx` distribution
  - Only warn once about slow Numpy array conversion with old Matlab versions
- Bug fixes:
  - Fix bug which made the required version checks fail with Euphonic 1.0.0
  - Fix bug with Numpy array conversion with Matlab 2018

v0.3.2

- Bug fixes:
  - Use of temperature=0 will now calculate the 0K Debye-Waller and Bose population factors - previously these temperature dependent effects were not calculated at 0K
- Improvements:
  - There are now `help` and `doc` commands which override the built-in Matlab commands to display richer help information (from the Python on-line documentation) for Euphonic commands. To use them, you must first import them with `import euphonic.help` or `import euphonic.doc` to override the built-in commands. Then use it as normal, e.g. `help euphonic.ForceConstants`. If this is used without the import, the original Matlab help is displayed which has been modified to suggest that the import is used.

v0.3.1

- Improvements:
  - There is a new `frequency_scale` argument to `horace_disp` which allows the output frequencies to be scaled

v0.3.0

- Dependency changes:
  - Euphonic version dependency increased from >=0.5.0 to >=0.6.0
- Breaking changes:
  - The default units of `StructureFactor.structure_factors` in Euphonic have been changed from `angstrom**2` per unit cell to `mbarn` per sample atom, and are now in absolute units including a previously omitted 1/2 factor. So the structure factors produced by `CoherentCrystal.horace_disp` have increased by a factor of `1e11/(2*N_atoms)`
- Other changes:
  - The `eta_scale` keyword argument to `CoherentCrystal` has been deprecated, `dipole_parameter` should be used instead
  - A Python ValueError will now be raised if an unrecognised keyword argument is passed to `CoherentCrystal`

v0.2.2

This release has no code changes, this just updates the IDAaaS installation documentation

v0.2.1

This release has no code changes, this update is only to fix the .mltbx upload to the MATLAB File Exchange

v0.2.0

There has been a major refactor, which means changes to how Horace-Euphonic-Interface is installed. There are also major changes to how Euphonic is used, the API has been updated to make it more object-oriented.

- Dependency changes:
  - Euphonic version dependency increased to >=0.5.0
- Installation changes:
  - Horace-euphonic-interface is now distributed as a Matlab toolbox (`.mltbx`) which is available in the Matlab File Exchange as an Add-On
- Usage changes:
  - `euphonic_sf` has been removed
  - `euphonic_on` has been removed
  - Force constants are now a separate object (`ForceConstants`) rather than passing these arguments to `euphonic_sf`
  - The model parameters are set in a `CoherentCrystal` model object, rather than passing these parameters to `euphonic_sf`
  - The function handle to be passed to `disp2sqw_eval` is `CoherentCrystal.horace_disp` rather than `euphonic_sf`
  - The `dw_grid` argument has been renamed to `debye_waller_grid`
  - `fall_back_on_python` argument has been removed as this has been removed in Euphonic

For more detailed help see the documentation

v0.1.0

- First release