Galore Documentation

Release 0.7.0

Scanlon Materials Theory Group

Jul 16, 2021

CONTENTS:

CHAPTER

ONE

README

1.1 Introduction

Galore is a package which applies Gaussian and Lorentzian broadening to data from *ab initio* calculations. The two main intended applications are

1. Gaussian and Lorentzian broadening of electronic density-of-states, with orbital weighting to simulate UPS/XPS/HAXPES measurements.
2. Application of Lorentzian instrumental broadening to simulated Raman spectra from DFPT calculations.

1.2 Documentation

A brief overview is given in this README file. A full manual, including tutorials and API documentation, is available online at readthedocs.io. You can build a local version using Sphinx with `make html` from the *docs* directory of this project.

An brief formal overview of the background and purpose of this code has been published in *The Journal of Open Source Software*.

1.3 Usage

Broadening, weighting and plotting are accessed with the `galore` program. For full documentation of the command-line flags, please use the in-built help:

```
galore -h
```

1.3.1 Instrumental broadening

Data may be provided as a set of X,Y coordinates in a text file of comma-separated values (CSV). Whitespace-separated data is also readable, in which case a *.txt* file extension should be used.

To plot a CSV file to the screen with default Lorentzian broadening (2 cm^{-1}), use the command:

```
galore MY_DATA.csv -l -p
```

and to plot to a file with more generous 10 cm^{-1} broadening:

```
galore MY_DATA.csv -l 10 -p MY_PLOT.png
```

will provide the additional data needed.

Other file formats are supported, including IR and Raman intensity simulation output. See the Tutorials for usage examples.

1.3.2 Photoelectron spectra

UPS, XPS or HAXPES spectra can be simulated using Galore. This requires several inputs:

- Orbital-projected density of states data. - This may be provided as an output file from the VASP or GPAW codes. - Formatted text files may also be used.
- Instrumental broadening parameters. The Lorentzian and Gaussian broadening widths are input by the user as before.
- Photoionization cross section data, which is used to weight the contributions of different orbitals.
  - Galore includes data for valance band orbitals at Al k- (XPS) and He II (UPS) energies, drawn from a more extensive table computed by Yeh and Lindau (1985). An alternative dataset may be provided as a JSON file; it is only necessary to include the elements and orbitals used in the DOS input files.
  - Cross-sections for high-energy (1-1500 keV) photons have been fitted from tabulated data computed by Scofield (1973).

See the Tutorials for a walkthrough using sample data.

The orbital data can also be accessed without working on a particular spectrum with the `galore-get-cs` program. For example:

```
galore-get-cs 4 Sn O
```

will print a set of valence orbital weightings for Sn and O corresponding to a 4 keV hard x-ray source. These values have been converted from atomic orbital data to *per electron* cross-sections.

The `galore-plot-cs` program is provided for plotting over a range of energies using the high-energy fitted data:

```
galore-plot-cs Pb S --emin 2 --emax 10 -o PbS.pdf
```

generates a publication-quality plot of cross-sections which may help in the selection of appropriate HAXPES energies for experiments with a given material.

1.4 Requirements

Galore is currently compatible with Python versions 3.5 and newer. Galore uses Numpy to apply convolution operations. Matplotlib is required for plotting.

Galore uses Pip and setuptools for installation. You *probably* already have this; if not, your GNU/Linux package manager will be able to oblige with a package named something like `python-setuptools`. On Max OSX, the Python distributed with Homebrew includes setuptools and Pip.

1.5 Installation

1.5.1 Windows user installation

Anaconda is recommended for managing the Python environment and dependencies on Windows. From the Anaconda shell:

```
pip3 install .
```

1.5.2 Linux/Mac developer installation

From the directory containing this README:

```
pip3 install --user -e .
```

which installs an *editable* (`-e`) version of galore in your userspace. The executable program `galore` goes to a user directory like `~/.local/bin` (which may need to be added to your PATH) and the galore library should be available on your PYTHONPATH. These are links to the project source folder, which you can continue to edit and update using Git.

To import data from VASP calculations you will need the Pymatgen library. If you don't have Pymatgen yet, the requirements can be added to the Galore installation with by adding `[vasp]` to the pip command e.g.:

```
pip3 install --user -e .[vasp]
```

1.5.3 Installation for documentation

If you need to build the documentation you can add `[docs]` to the pip command to ensure you have all the Sphinx requirements and extensions:

```
pip3 install --upgrade .[docs]
```

1.6 Support

If you're having trouble with Galore or think you've found a bug, please report it using the Github issue tracker. Issues can also be used for questions and discussion about the Galore methodology/implementation.

1.7 Development

This code is developed by the Scanlon Materials Theory Group based at University College London. Suggestions and contributions are welcome; please read the CONTRIBUTING guidelines and use the Github issue tracker.

1.8 How to cite Galore

If you use Galore in your research, please consider citing the following work:

> Adam J. Jackson, Alex M. Ganose, Anna Regoutz, Russell G. Egdell, David O. Scanlon (2018). *Galore: Broadening and weighting for simulation of photoelectron spectroscopy.* Journal of Open Source Software, 3(26), 773, doi: 10.21105/joss.007733

Galore includes a machine-readable citation file in an emerging standard format with citation details for the actual code, but as conventions for software citation are still developing the JOSS paper is a more reliable method of giving credit.

1.9 License

Galore is made available under the GNU Public License, version 3.

1.10 Acknowledgements

Development work by Adam J. Jackson took place in the course of research into new transparent conducting materials, led by David O. Scanlon and funded by EPSRC (project code EP/N01572X/1). Work by Alex M. Ganose was supported by a studentship co-sponsored by the Diamond Light Source at the EPSRC Centre for Doctoral Training in Molecular Modelling and Materials Science (EP/L01582/1). Anna Ragoutz was our expert advisor on all things PES, guiding the feature-set and correcting the implementation of weighting, and was supported by an Imperial College Research Fellowship.

We acknowledge useful discussions with Alexey Sokol (who proposed that a code such as this would be useful), Katie Inzani, and Tim Veal. Feature requests and user testing came from Benjamin Williamsion, Christopher Savory and Winnie L. Leung.

This would have been much more painful if not for the excellent scientific Python ecosystem, and the Python Materials Genome project spared us the pain of writing Yet Another Vasp Parser.

CHAPTER

TWO

TUTORIALS

2.1 Simulated IR

The first-order infra-red absorption spectrum can be simulated by performing lattice dynamics calculations to obtain the -point vibrational mode frequencies. The dielectric response of the system determines the relative intensities of the modes, and some will be inactive for symmetry reasons.

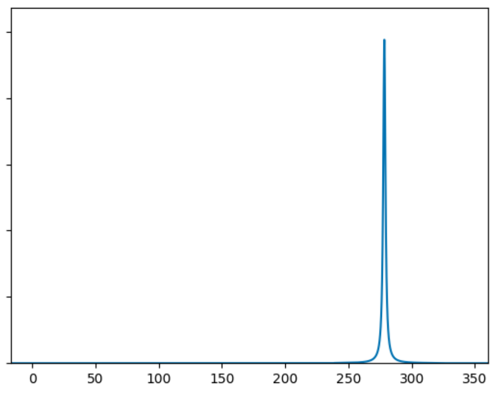
If you have the VASP quantum chemistry code, the simplest way to compute these properties is with a single DFPT calculation (e.g. `IBRION = 7`, `LEPSILON = .TRUE.`, `NWRITE = 3`) and follow-up with David Karhanek's analysis script. A sample output file is provided for CaF_2 (computed within the local-density approximation (LDA) using a 700 eV plane-wave cutoff) is included as *test/CaF2/ir_lda_700.txt*.

This file (found as *intensities/results/results.txt* after running the script) uses a three-column space-separated format understood by Galore. To plot the spectrum to screen with some broadening then we can use:

```
galore test/CaF2/ir_lda_700.txt -g 0.5 -l --spikes --plot
```

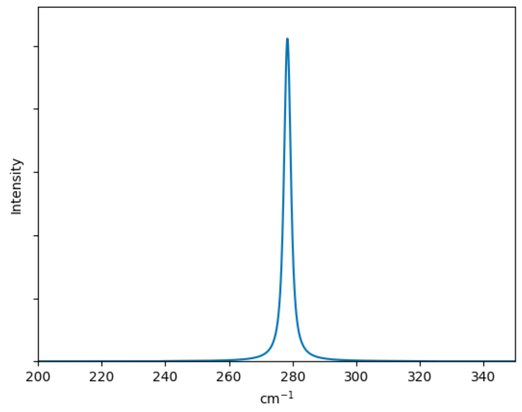
Breaking down this command: First we provide the path to a data file. This can also appear elsewhere in the argument string, but as many flags take optional arguments it is safest to put it first. `-g` applies Gaussian broadening; here we specify a width of 0.5. This will use the same units as the x-axis; in this case cm^{-1}. `-l` applies Lorentzian broadening; as no width is specified, the default 2 cm^{-1} will be used. This is generally a sensible value for optical measurements, but some tuning may be needed. `--spikes` prevents interpolation when the data is resampled; this is required for datasets where the values between data points should be set to zero before broadening. Finally `--plot` will cause Galore to print to the screen using Matplotlib. (The abbreviation `-p` can also be used.)

To see the full list of command-line arguments you can use `galore -h` or check the *Command-line interface* section in this manual.



Admittedly, it isn't the most exciting spectrum, with a single peak around 280 cm^{-1}. Let's make some adjustments: we'll add a touch more Gaussian broadening, zoom in on the peak by limiting the axis range, add axis labels and write to a file.

```
galore test/CaF2/ir_lda_700.txt -g 1.2 -l  --spikes \
  --plot ir_lda_700_better.png \
  --xmin 200 --xmax 350 --units cm-1 --ylabel Intensity
```



Now the plot is more publication-ready! If you would like to use another plotting program, the broadened data can be output to a CSV file by simply replacing `--plot` with `--csv`:

```
galore test/CaF2/ir_lda_700.txt -g 1.2 -l -k --csv --xmin 200 --xmax 350
```

(Here we have also replaced `--spikes` with its short form `-k`.) This will write a csv file to the standard output as no filename was given. We can also write space-separated text data, so for example

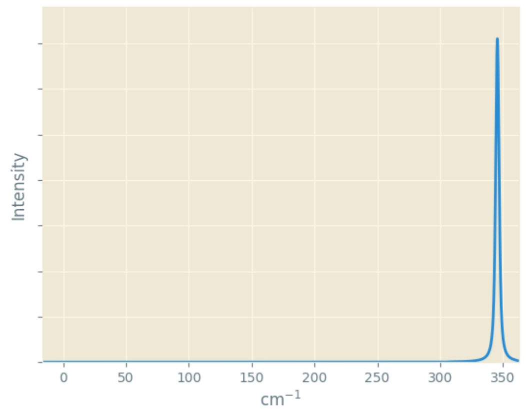
```
galore test/CaF2/ir_lda_700.txt -g 1.2 -l -k --txt ir_CaF2_broadened.txt
```

generates a file with two columns (i.e. energy and broadened intensity).

2.2 Simulated Raman

Broadening a simulated Raman spectrum is very similar to broadening a simulated IR spectrum. Galore recognises the output format of the vasp_raman.py code, which automates the process of following vibrational modes and calculating the polarisability change on each displacement. The output file has a simple format and Galore recognises them by inspecting the header. Sample data is included (computed with LDA using VASP with a 500 eV cutoff) as *test/CaF2/raman_lda_500.dat*. We generate a plot in the same way as before:

```
galore test/CaF2/raman_lda_500.dat -g -l -k --plot --units cm-1 --ylabel Intensity --
↪style Solarized_Light2
```



This time we set an alternative appearance style with `--style Solarized_Light2`. Galore uses Matplotlib styles, and you can use inbuilt styles or create custom stylesheets. The `dark_background` style may be useful for slide presentations. Note that for the same material we are seeing a single peak again, but at a different frequency to the IR plot. This is not a shift; the peak at 280 cm^{-1} is still present but has zero activity, while the peak calculated at 345 cm^{-1} has zero IR activity.

2.3 Simulated Photoelectron Spectroscopy

Photoelectron measurements allow valence band states to be probed fairly directly; energy is absorbed by an incident photon as it ejects an electron from the sample, and the shift in energy is measured relative to a monochromatic photon source. Ultraviolet photoelectron spectroscopy (UPS), x-ray photoelectron spectroscopy (XPS) and Hard x-ray photoelectron spectroscopy (HAXPES) are fundamentally similar techniques, differing in the energy range of the incident photons.

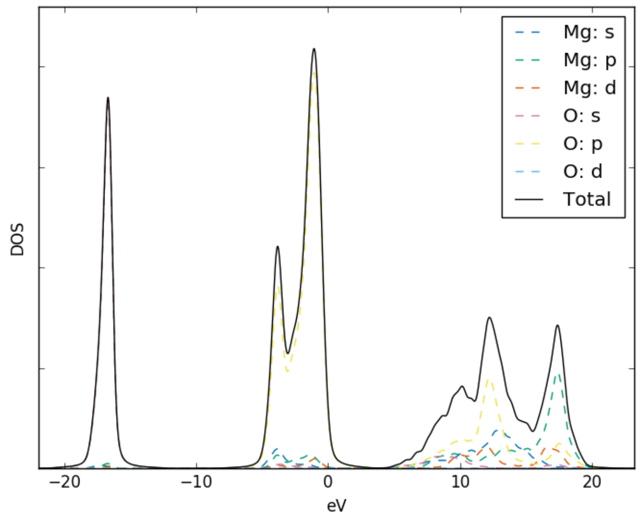
These binding energies may be compared with the full density of states (DOS) computed with *ab initio* methods. However, the intensity of interaction will vary depending on the character of the energy states and the energy of the radiation source. The relevant interaction parameter ("photoionization cross-section") has been calculated systematically over the periodic table and relevant energy values; Galore includes some such data from Yeh and Lindau (1985).

In *ab initio* codes it is often possible to assign states to particular orbital characters; often this is limited to s-p-d-f (i.e. the second quantum number) but in principle an all-electron code can also assign the first quantum number. Directional character is also sometimes assigned, usually relative to Cartesian axes. These various schemes are used to construct a "projected density of states" (PDOS).

The construction of a PDOS in *ab initio* calculations is slightly arbitrary and lies beyond the scope of Galore. However, when the orbital assignment has been made the DOS elements can be weighted to simulate the photoionization spectrum.

We begin by plotting a PDOS from sample data in *test/MgO*. This was computed using VASP with standard pseudopotentials and the revTPSS exchange-correlation functional.

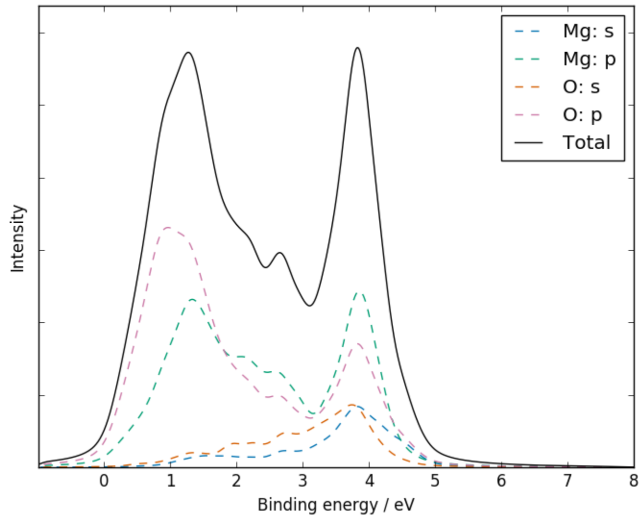
```
galore test/MgO/MgO_Mg_dos.dat test/MgO/MgO_O_dos.dat \
  --plot --pdos -g 0.5 -l 0.2 --ylabel DOS --units eV
```



Note that the `--pdos` flag is required to enable the reading of orbital-projected input files. The element identity is read from these filenames, and is expected between two underscore characters. The orbital names are determined from the column headers in this file. It is possible to use spin-polarised data by using a header form such as 's(up)' and 's(down)' for spin-polarised s-orbitals, in which case the contributions of the spin channels will be summed together.

Let's turn this into a useful XPS plot. The flag `--weightings` can be used to pass a data file with cross-section data; for some cases data is build into Galore. Here we will use the data for Al k- radiation which is denoted `alka` is built into Galore. We also flip the x-axis with `--flipx` to match the usual presentation of XPS data as positive ionisation or binding energies rather than the negative energy of the stable electron states. Finally the data is written to a CSV file with the `--csv` option.

```
galore test/MgO/MgO_Mg_dos.dat test/MgO/MgO_O_dos.dat \
  --plot mgo_xps.png --pdos -g 0.2 -l 0.2 --weighting alka \
  --units ev --xmin -1 --xmax 8 --ylabel Intensity \
  --csv mgo_xps.csv --flipx
```



Plotting the CSV file with a standard plotting package should give a similar result to the figure above; if not, please report this as a bug!

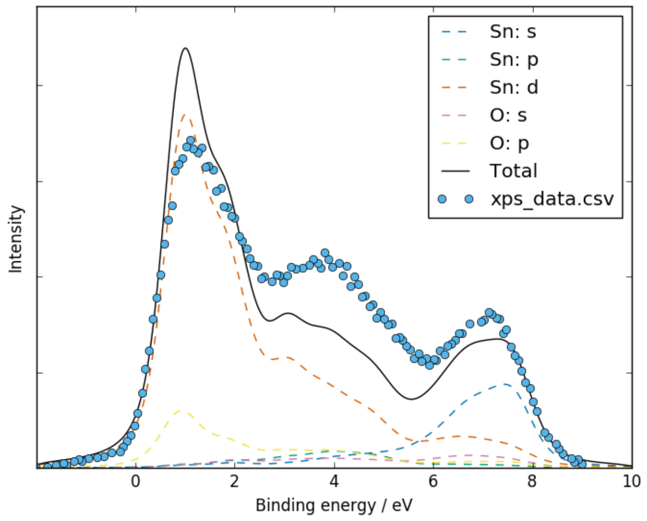
2.4 Compare with literature: tin dioxide

Sample VASP output data is included for rutile tin dioxide. This was computed with the PBE0 functional, using a 4x4x5 **k**-point mesh and 700 eV basis-set cutoff. The structure was optimised to reduce forces to below 1E-3 eV $^{-1}$ using 0.05 eV of Gaussian broadening and the DOS was computed on an automatic tetrahedron mesh with Blöchl corrections. Instead of the separate .dat files used above, we will take advantage of Galore's ability to read a compressed *vasprun.xml* file directly. This requires the Pymatgen library to be installed:

```
pip3 install --user pymatgen
```

If the GPAW Python library is available, it is also possible to import this data from *.gpw* output files.

2.4.1 XPS



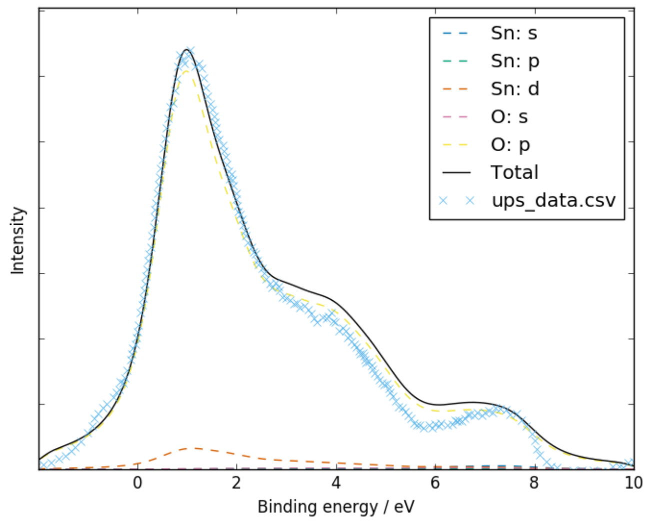
We have digitised the experimental data plotted in Fig.3 of Farahani et al. (2014) in order to aid a direct comparison:

```
galore test/SnO2/vasprun.xml.gz --plot -g 0.5 -l 0.4 \
  --pdos --w alka --flipx --xmin -2 --xmax 10 \
  --overlay test/SnO2/xps_data.csv  --overlay_offset -4 \
  --overlay_scale 5 --units ev --ylabel Intensity
```

(Note that here the shorter alias -w is used for the XPS weighting.) The general character and peak positions match well, but the relative peak intensities could be closer; the first peak is very strong even with cross-section weightings applied. We can see that the dip in between the second an third main peak corresponds to a crossing point between the contributions of the Sn-s and Sn-d orbitals.

A slightly more generous assignment of 'p' vs 'd' character by the orbital projection scheme would have made for a better fit! The published results seem to fit better despite using similar calculation parameters; we can't see if the orbital breakdown is indeed the determining factor.

2.4.2 UPS



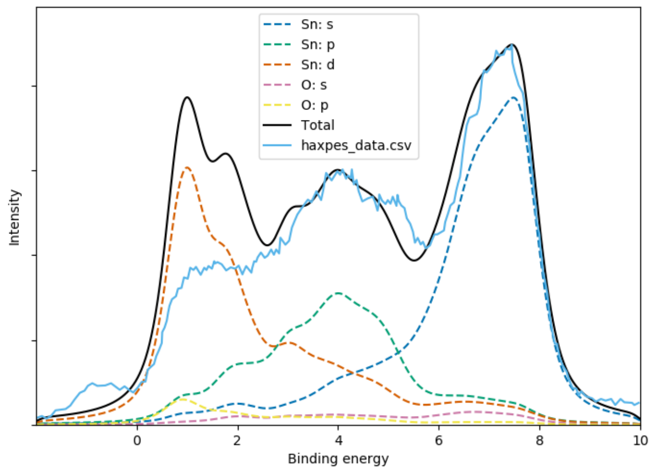
Experimental UPS data was digitized from Fig. 1 of Themlin et al. (1990). A satisfactory fit is obtained for the three main peaks, but the "bump" below zero suggests the presence of some phenomenon in the bandgap which was not captured by the *ab initio* calculation:

```
galore test/SnO2/vasprun.xml.gz --plot -g 0.5 -l 0.9 \
  --pdos --w he2 --flipx --xmin -2 --xmax 10 \
  --overlay test/SnO2/ups_data.csv --overlay_offset 0.44 \
  --units ev --ylabel Intensity --overlay_style x
```

The authors noted this in their own comparison to a DOS from tight-binding calculations:

> The location of the VBM in out UPS data was complicated by the presence of a slowly varying photo-electron signal, resulting from a surface-state band.

2.4.3 HAXPES



A HAXPES spectrum was obtained by digitizing Fig. 1 of Nagata et al. (2011). These experiments were performed with 5.95 keV x-rays; we set `--w 5.95` to estimate corresponding cross-sections by fitting to Scofield (1973):

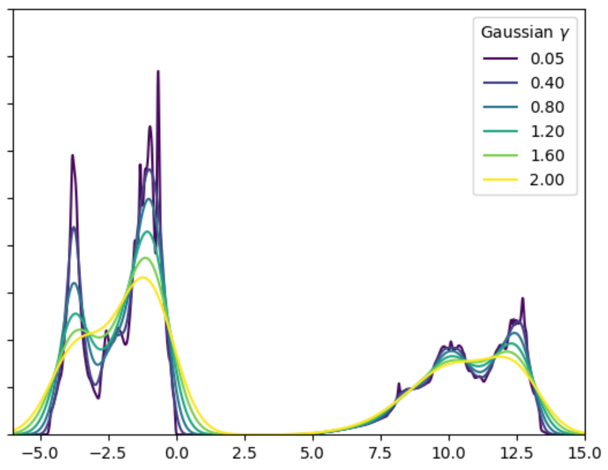
```
galore test/SnO2/vasprun.xml.gz --plot -g 0.3 -l 0.5 --pdos \
  --w 5.95 --flipx --xmin -2 --xmax 10 \
  --overlay test/SnO2/haxpes_data.csv --overlay_offset -3.7 \
  --ylabel Intensity --overlay_style -
```

We see that the weighting goes some way to rebalancing the peak intensities but once again the Sn-d states are over-represented. Surface states above the valence band are seen in the experimental data.

2.5 Python API

The Python API allows custom plots to be produced by generating data sets Galore's core functions and passing Matplotlib axes to the plotting functions. A few examples are provided below. It is not especially difficult to access the lines plotted to an axis and manipulate their appearance, rescale them etc.

2.5.1 Overlaying different amounts of broadening



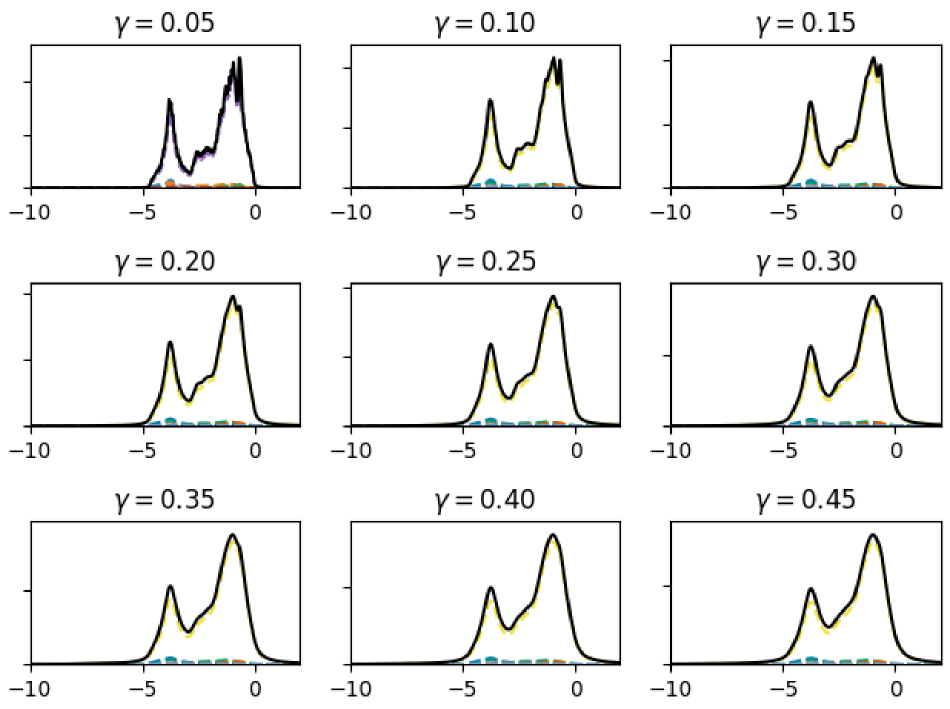
```python
#! /usr/bin/env python3

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.cm import viridis as cmap
import galore
import galore.plot

vasprun = 'test/MgO/vasprun.xml.gz'
xmin, xmax = (-6, 15)

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

widths = np.arange(0, 2.01, 0.4)
```

(continues on next page)

(continued from previous page)

```
widths[0] = 0.05  # Use a finite width in smallest case

for g in widths:

    x_values, broadened_data = galore.process_1d_data(input=vasprun,
                                                      gaussian=g,
                                                      xmin=xmin, xmax=xmax)

    broadened_data /= g  # Scale values by broadening width to conserve area

    galore.plot.plot_tdos(x_values, broadened_data, ax=ax)
    line = ax.lines[-1]
    line.set_label("{0:2.2f}".format(g))
    line.set_color(cmap(g / max(widths)))

ax.set_ylim(0, 1800)
legend = ax.legend(loc='best')
legend.set_title('Gaussian $\gamma$')
plt.show()
```

2.5.2 Plotting different amounts of broadening on tiles



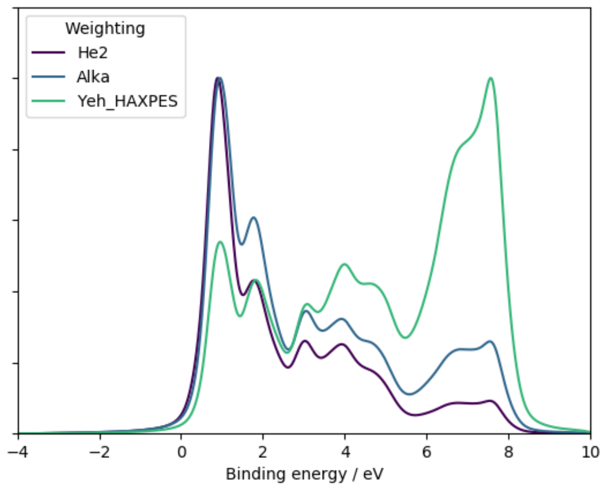
```
#! /usr/bin/env python3

import numpy as np
import matplotlib.pyplot as plt
plt.style.use("seaborn-colorblind")
import galore
```

(continues on next page)

(continued from previous page)

```
import galore.plot

vasprun = './test/MgO/vasprun.xml.gz'
xmin, xmax = (-10, 2)

fig = plt.figure()

for i, l in enumerate(np.arange(0.05, 0.50, 0.05)):
    ax = fig.add_subplot(3, 3, i + 1)
    ax.set_title("$\gamma = {0:4.2f}$".format(l))
    plotting_data = galore.process_pdos(input=[vasprun], lorentzian=l,
                                        xmin=xmin, xmax=xmax)
    galore.plot.plot_pdos(plotting_data, ax=ax)
    ax.legend().set_visible(False)

fig.tight_layout()
plt.show()
```

2.5.3 Comparing different photoemission weightings



```
#! /usr/bin/env python3
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.cm import viridis as cmap
import galore
import galore.plot

vasprun = 'test/SnO2/vasprun.xml.gz'
xmin, xmax = (-10, 4)

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

weightings = ('He2', 'Alka', 'Yeh_HAXPES')

```

(continues on next page)

(continued from previous page)

```
for i, weighting in enumerate(weightings):

    plotting_data = galore.process_pdos(input=[vasprun],
                                        gaussian=0.3, lorentzian=0.2,
                                        xmin=xmin, xmax=xmax,
                                        weighting=weighting)
    galore.plot.plot_pdos(plotting_data, ax=ax, show_orbitals=False,
                          units='eV', xmin=-xmin, xmax=-xmax,
                          flipx=True)

    line = ax.lines[-1]
    line.set_label(weighting)
    line.set_color(cmap(i / len(weightings)))

    ymax = max(line.get_ydata())
    line.set_data(line.get_xdata(), line.get_ydata() / ymax)

ax.set_ylim((0, 1.2))
legend = ax.legend(loc='best')
legend.set_title('Weighting')
plt.show()
```

CHAPTER THREE

THEORY

A short academic paper is in preparation which gives an overview of Galore's applications. Some of that information is repeated here; this section of the user guide aims to provide essential information and refer to the academic literature for those seeking more depth and context.

3.1 Photoelectron spectroscopy

3.1.1 History

Photoelectron spectroscopy (PES) is a family of methods used to characterise the chemical nature and electronic structure of materials. PES is based on the photoelectric effect, which was discovered by Hertz. [1] It was explored extensively by Rutherford and colleagues [2] and within a few years researchers including de Broglie [3] and Robinson [4] were using the technique to measure electron binding energies through the relationship

$$E_\mathrm{k} = h\nu - E_\mathrm{B}.$$

Photons with energies $h\nu$ ranging from 5 eV up to 12 keV eject electrons (referred to as "photoelectrons") from the occupied orbitals of a sample. The kinetic energy E_k of each photoelectron therefore depends on its binding energy E_B. The names of various PES methods refer to the photon energy range used:

- ultraviolet photoelectron spectroscopy (UPS): 5–100 eV
- X-ray photoelectron spectroscopy (XPS): 0.3–2 keV
- hard X-ray photoelectron spectroscopy (HAXPES, HE-PES, HXPS, HX-PES): above 2 keV

3.1.2 Broadening

Major sources of broadening include:

- Intrinsic lifetime broadening (Lorentzian)
  - While this can play a significant role, the lifetime broadening is energy-dependent and care should be taken when applying it across the full data set.
- Franck–Condon phonon broadening (Gaussian)
  - This is caused by vibrations in the system which lead to a distribution of accessible photoionization energies.
  - In oxides this is associated with as much as 0.8 eV broadening
- Instrumental broadening (Gaussian)
  - Typical values are in the range 0.2–0.3 eV.

3.1.3 Weighting

The Gelius model

The Gelius model was originally developed to describe molecular systems. [5][6][7]

Asymmetry corrections

3.1.4 References

CHAPTER

FOUR

COMMAND-LINE INTERFACE

The main interface for Galore is the `galore` command. Additionally, the `galore-get-cs` program is provided for convenient access to cross-section data.

4.1 Style files

Advanced plot styling can be managed with style files. Galore uses Matplotlib for plotting. The `--style` option allows you to pass in the name of a default style (try *dark_background*) or the path to a file containing keywords and values. For more information and a sample file see the Matplotlib docs here.

4.1.1 galore

```
usage: galore [-h] [-l [LORENTZIAN]] [-g [GAUSSIAN]] [-w WEIGHTING]
              [--units {cm,cm-1,thz,THz,ev,eV,ry,Ry,ha,Ha}] [--ylabel YLABEL]
              [--txt [TXT]] [--csv [CSV]] [-p [PLOT]] [-d SAMPLING] [-k]
              [--pdos] [--flipx] [--xmin XMIN] [--xmax XMAX] [--ymin YMIN]
              [--ymax YMAX] [--style STYLE [STYLE ...]] [--overlay OVERLAY]
              [--overlay_scale OVERLAY_SCALE]
              [--overlay_offset OVERLAY_OFFSET]
              [--overlay_style OVERLAY_STYLE] [--overlay_label OVERLAY_LABEL]
              input [input ...]
```

Positional Arguments

input	Input data file. The supported formats are vasprun.xml (VASP output), *.gpw (GPAW output), *.csv (comma-delimited text) and *.txt (space-delimited text). Default: “vasprun.xml”

Named Arguments

-l, --lorentzian	Apply Lorentzian broadening with specified width. Default: False
-g, --gaussian	Apply Gaussian broadening with specified width. Default: False
-w, --weighting	Apply cross-section weighting to data. “alka”, “he2” and “yeh_haxpes” select tabulated data for valence band at 1486.6 eV, 40.8 eV and 8047.8 eV respectively. Numerical values will be interpreted as an energy in keV; for energies from 1-1500 eV cross-sections will be determined using a parametrisation from tabulated data. Alternatively, provide path to a JSON file with cross-section data.
--units, --x_units	Possible choices: cm, cm-1, thz, THz, ev, eV, ry, Ry, ha, Ha Units for x axis (usually frequency or energy) Default: “”
--ylabel	Label for plot y-axis
--txt	Write broadened output as space-delimited text; file if path provided, otherwise write to standard output. Default: False
--csv	Write broadened output as comma-separated values; file if path provided, otherwise write to standard output. Default: False
-p, --plot	Plot broadened spectrum. Plot to filename if provided, otherwise display to screen. Default: False
-d, --sampling	Width, in units of x, of x-axis resolution. If not specified, default value is based on units. If units are not specified, default value is 1e-2. Default: False
-k, --spikes, --spike	Resample data as “spikes” on a zero baseline. The default is to interpolate linearly between y-values, which is reasonable for distributions such as DOS. If the input data set only contains active energies/frequencies (e.g. IR modes) then you should use –spike mode. See tutorials for examples. Default: False
--pdos	Use orbital-projected data Default: False
--flipx, --xflip	Negate x-values in output; this may be helpful for comparison with binding energy measurments. Default: False
--xmin	Minimum x axis value
--xmax	Maximum x axis value
--ymin	Minimum y axis value Default: 0

--ymax	Maximum y axis value
--style	Plotting style: a sequence of matplotlib styles and paths to style files. The default palette is called "seaborn-colorblind". Default: ['seaborn-colorblind']
--overlay	Data file for overlay
--overlay_scale	Y-axis scale factor for data overlay
--overlay_offset	X-axis offset for data overlay Default: 0
--overlay_style	Matplotlib line style for overlay data. Default "o" for circles, "x:" for crosses joined by dotted lines, etc. Default: "o"
--overlay_label	Legend label for data overlay

4.1.2 galore-get-cs

```
usage: galore-get-cs [-h] energy elements [elements ...]
```

Positional Arguments

energy	Photon energy, expressed as source type: "he2" for He (II), "alka" for Al k-alpha, (values from Yeh/Lindau (1985)) or as energy in keV (values from polynomial fit to Scofield (1973)).
elements	Space-separated symbols for elements in material.

4.1.3 galore-plot-cs

```
usage: galore-plot-cs [-h] [--emin EMIN] [--emax EMAX] [--megabarn]
                      [--size SIZE SIZE] [--output OUTPUT]
                      [--fontsize FONTSIZE] [--style STYLE [STYLE ...]]
                      elements [elements ...]
```

Positional Arguments

elements	Space-separated symbols for elements in material.

Named Arguments

--emin	Minimum energy in keV Default: 1
--emax	Maximum energy in keV Default: 20
--megabarn	Set y-axis unit to megabarn/electron Default: False
--size	Figure dimensions in cm
--output, -o	Output filename. If not given, plot to screen.
--fontsize	Font size in pt Default: 12
--style	Plotting style: a sequence of matplotlib styles and paths to style files. The default palette is called "seaborn-colorblind". Default: ['seaborn-colorblind']

CHAPTER

FIVE

PYTHON API

5.1 galore package

5.1.1 Submodules

galore.cross_sections module

`galore.cross_sections.`**`cross_sections_info`**(*cross_sections*, *logging=None*)

Log basic info from cross-sections dict.

Parameters

- **`cross_sections`** (*`dict`*) – The keys 'energy', 'citation', 'link' and 'warning' are checked for relevant information
- **`logging`** (*`module`*) – Active logging module from Python standard library. If None, logging will be set up.

Returns Active logging module from Python standard library

Return type module

`galore.cross_sections.`**`get_cross_sections`**(*weighting*, *elements=None*)

Get photoionization cross-section weighting data.

For known sources, data is based on tabulation of Yeh/Lindau (1985).[1] Otherwise, energies in keV from 1-1500 are used with log-log polynomial parametrisation of data from Scofield.[2]

References

1. Yeh, J.J. and Lindau, I. (1985) Atomic Data and Nuclear Data Tables 32 pp 1-155
2. J. H. Scofield (1973) Lawrence Livermore National Laboratory Report No. UCRL-51326

Parameters

- **`weighting`** (*`str or float`*) – Data source for photoionization cross-sections. If the string is a known keyword then data will be drawn from files included with Galore. Otherwise, the string will be interpreted as a path to a JSON file containing data arranged in the same way as the output of this function.
- **`elements`** (*`iterable or None`*) – Collection of element symbols to include in the data set. If None, a full set of available elements will be included. When using a JSON dataset (including the inbuilt Yeh/Lindau) this parameter will be ignored as the entire dataset has already been loaded into memory.

Returns

Photoionization cross-section weightings arranged by element and orbital as nested dictionaries of floats, i.e.:

```
{el1: {orb1: cs11, orb2: cs12, ...},
 el2: {orb1: cs21, orb2: cs22, ...}, ... }
```

In addition the keys "reference", "link", "energy" and "warning" may be used to store metadata.

Return type dict

`galore.cross_sections.`**`get_cross_sections_json`**`(`*path*`)`

Get valence-band cross-sections from JSON file

Read photoionization data from a JSON file. File is expected to contain data for multiple elements and orbitals in the form `{El1: {orb1: c1, orb2: c2, ...}, ...}`. While it is expected that Galore will be used to examine valence-band orbitals labelled (s, p, d, f) it may be helpful in some cases to prepare a file with alternative orbital labels corresponding to the pDOS labels.

The labels 'citation', 'energy' and 'link' are reserved for metadata which may be displayed in the program log. The label 'comment' may be used for additional material in the JSON file; it is recommended to use this repeatedly for line-breaks, e.g.:

```
{"comment": "First line of text",
 "comment": "which is continued.",
 ...}
```

Parameters **`path`** (*`str`*) – Path to JSON file

Returns

Weighted photoionization cross-sections for each element and orbital in form:

```
{el1: {'s': c11, 'p': c12, ... },
 el2: {'s': c21, 'p': c22, ... }, ... }
```

in tabulated units.

Return type dict

`galore.cross_sections.`**`get_cross_sections_scofield`**`(`*energy*, *elements=None*`)`

Get valence-band cross-sections from fitted data

Energy-dependent cross-sections have been averaged and weighted for the uppermost s, p, d, f orbitals from data tabulated by Scofield. The energy/cross-section relationship was fitted to an order-8 polynomial on a log-log scale.

Multiple energy values can be evaluated simultaneously by passing an array-like group of energies as `energy`. In this case the cross-section values will be arrays with the same shape as the energy arrays.

Parameters

- **`energy`** (*`float or array-like`*) – Incident energy in keV
- **`element`** (*`iterable or None`*) – Iterable (e.g. list) of element symbols. If None, include all available elements (1 <= Z <= 100).

Returns

Weighted photoionization cross-sections in Barns/electron for each orbital in form:

```
{el1: {'s': c11, 'p': c12, ... },
 el2: {'s': c21, 'p': c22, ... }, ... }
```

Return type dict

Raises **ValueError** – Energy values must lie within interpolation range 1–1500keV

`galore.cross_sections.`**`get_cross_sections_yeh`**(*source*)

Get valence-band cross-sections from tabulated data

Tabulated values of photoionization cross-sections were drawn from ref [1] for energy values corresponding to relevant radiation sources: - 1486.6 eV, corresponding to Al k-alpha (laboratory XPS) - 40.8 eV, corresponding to He II (laboratory UPS) - 8047.8 eV, corresponding to a possible HAXPES source

References

1. Yeh, J.J. and Lindau, I. (1985) Atomic Data and Nuclear Data Tables 32 pp 1-155

Parameters **source** (*str*) – Label corresponding to radiation source. Accepted values 'alka' (1486.6 eV), 'he2' (40.8 eV), 'yeh_haxpes' (8047.8). These keys are not case-sensitive and correspond to Al k-alpha, He(II) and hard x-ray sources.

Returns

Weighted photoionization cross-sections in megaBarns/electron for each orbital in form:

```
{el1: {'s': c11, 'p': c12, ... },
 el2: {'s': c21, 'p': c22, ... }, ... }
```

Return type dict

galore.formats module

`galore.formats.`**`is_complete_dos`**(*pdos*)

Determine whether the object is a pymatgen CompleteDos object

`galore.formats.`**`is_csv`**(*filename*)

Determine whether file is CSV by checking extension

`galore.formats.`**`is_doscar`**(*filename*)

Determine whether file is a DOSCAR by checking fourth line

`galore.formats.`**`is_gpw`**(*filename*)

Determine whether file is GPAW calculation by checking extension

`galore.formats.`**`is_vasp_raman`**(*filename*)

Determine if file is raman-sc/vasp_raman.py data by checking header

`galore.formats.`**`is_xml`**(*filename*)

Determine whether file is XML by checking extension

`galore.formats.`**`read_csv`**(*filename*)

Read a txt file containing frequencies and intensities

If input file contains three columns, the first column is ignored. (It is presumed to be a vibrational mode index.)

Parameters **filename** (*str*) – Path to data file

Returns n x 2 Numpy array of frequencies and intensities

`galore.formats.`**`read_doscar`**(*filename='DOSCAR'*)
Read an x, y series of frequencies and DOS from a VASP DOSCAR file

Parameters **`filename`** (*`str`*) – Path to DOSCAR file

Returns Tuple containing x values and y values as lists

Return type data (2-tuple)

`galore.formats.`**`read_gpaw_pdos`**(*filename*, *npts=50001*, *width=0.001*, *ref='vbm'*)
Read orbital-projected DOS from GPAW with minimal broadening.

This requires GPAW to be installed and on your PYTHONPATH!

Parameters

- **`filename`** (*`str`*) – Path to GPAW calculation file. This should be a .gpw file generated with `calc.write('myfilename.gpw')`.
- **`npts`** (*`int`*) – Number of DOS samples
- **`width`** (*`float`*) – Gaussian broadening parameter applied by GPAW. Default is minimal so that broadening is dominated by choices in Galore. Beware that there is a strong interaction between this parameter and npts; with a small npts and small width, many energy levels will be missed from the DOS!
- **`ref`** (*`str`*) – Reference energy for DOS. 'vbm' or 'efermi' are accepted for the valence-band maximum or the Fermi energy, respectively. VBM is determined from calculation eigenvalues and not DOS values. If set to None, raw values are used.

Returns

PDOS data formatted as nestled OrderedDict of: {element: {'energy': energies, 's': densities, 'p' … }

Return type pdos_data (OrderedDict)

`galore.formats.`**`read_gpaw_totaldos`**(*filename*, *npts=50001*, *width=0.001*, *ref='vbm'*)
Read total DOS from GPAW with minimal broadening

This requires GPAW to be installed and on your PYTHONPATH!

Parameters

- **`filename`** (*`str`*) – Path to GPAW calculation file. This should be a .gpw file generated with `calc.write('myfilename.gpw')`.
- **`npts`** (*`int`*) – Number of DOS samples
- **`width`** (*`float`*) – Gaussian broadening parameter applied by GPAW. Default is minimal so that broadening is dominated by choices in Galore. Beware that there is a strong interaction between this parameter and npts; with a small npts and small width, many energy levels will be missed from the DOS!
- **`ref`** (*`str`*) – Reference energy for DOS. 'vbm' or 'efermi' are accepted for the valence-band maximum or the Fermi energy, respectively. VBM is determined from calculation eigenvalues and not DOS values. If set to None, raw values are used.

Returns 2D array of energy and DOS values

Return type data (np.ndarray)

`galore.formats.`**`read_pdos_txt`**(*filename*, *abs_values=True*)
Read a text file containing projected density-of-states (PDOS) data

The first row should be a header identifying the orbitals, e.g. "# Energy s p d f". The following rows contain the corresponding energy and DOS values. Spin channels indicated by (up) or (down) suffixes will be combined.

Parameters

- **`filename`** (*`str`*) – Path to file for import
- **`abs_values`** (*`bool, optional`*) – Convert intensity values to absolute numbers. This is primarily for compatibility with spin-polarised .dat files from Sumo. Set to False if negative values in spectrum are resonable.

Returns

Numpy structured array with named columns corresponding to input data format.

Return type data (np.ndarray)

`galore.formats.`**`read_txt`** (*filename*, *delimiter=None*)

Read a txt file containing frequencies and intensities

If input file contains three columns, the first column is ignored. (It is presumed to be a vibrational mode index.)

Parameters **`filename`** (*`str`*) – Path to data file

Returns n x 2 Numpy array of frequencies and intensities

`galore.formats.`**`read_vasp_raman`** (*filename='vasp_raman.dat'*)

Read output file from Vasp raman simulation

Parameters **`filename`** (*`str`*) – Path to formatted data file generated by https://github.com/raman-sc/VASP - Raman intensities are computed by following vibrational modes and calculating polarisability. The generated output file is named *vasp_raman.dat* but can be renamed if desired. The format is five space-separated columns, headed by `# mode freq(cm-1) alpha beta2 activity`.

Returns Only the columns corresponding to frequency and activity are retained.

Return type 2-D np.array

`galore.formats.`**`read_vasprun`** (*filename='vasprun.xml'*)

Read a VASP vasprun.xml file to obtain the density of states

Pymatgen must be present on the system to use this method

Parameters **`filename`** (*`str`*) – Path to vasprun.xml file

Returns A pymatgen Dos object

Return type data (pymatgen.electronic_structure.dos.Dos)

`galore.formats.`**`read_vasprun_pdos`** (*filename='vasprun.xml'*)

Read a vasprun.xml containing projected density-of-states (PDOS) data

Pymatgen must be present on the system to use this method

Parameters **`filename`** (*`str or CompleteDos`*) – Path to vasprun.xml file or pymatgen CompleteDos object.

Returns

PDOS data formatted as nestled OrderedDict of: {element: {'energy': energies, 's': densities, 'p' … }

Return type pdos_data (np.ndarray)

`galore.formats.`**`read_vasprun_totaldos`**(*filename='vasprun.xml'*)

Read an x, y series of energies and DOS from a VASP vasprun.xml file

Pymatgen must be present on the system to use this method

Parameters **`filename`** (*`str`*) – Path to vasprun.xml file

Returns 2D array of energy and DOS values

Return type data (np.ndarray)

`galore.formats.`**`write_csv`**(*x_values*, *y_values*, *filename='galore_output.csv'*, *header=None*)

Write output to a simple space-delimited file

Parameters

- **`x_values`** (*`iterable`*) – Values to print in first column
- **`y_value`** (*`iterable`*) – Values to print in second column
- **`filename`** (*`str`*) – Path to output file, including extension. If None, write to standard output instead.
- **`header`** (*`iterable`*) – Additional line to prepend to file. If None, no header is used.

`galore.formats.`**`write_pdos`**(*pdos_data*, *filename=None*, *filetype='txt'*, *flipx=False*)

Write PDOS or XPS data to CSV file

Parameters

- **`pdos_data`** (*`dict`*) – Data for pdos plot in format:

```
{'el1': {'energy': values, 's': values, 'p': values ...},
 'el2': {'energy': values, 's': values, ...}, ...}
```

  where DOS values are 1D numpy arrays. For deterministic output, use ordered dictionaries!
- **`filename`** (*`str or None`*) – Filename for output. If None, write to stdout
- **`filetype`** (*`str`*) – Format for output; “csv” or “txt.
- **`flipx`** (*`bool`*) – Negate the x-axis (i.e. energy) values to make binding energies

`galore.formats.`**`write_txt`**(*x_values*, *y_values*, *filename='galore_output.txt'*, *header=None*)

Write output to a simple space-delimited file

Parameters

- **`x_values`** (*`iterable`*) – Values to print in first column
- **`y_value`** (*`iterable`*) – Values to print in second column
- **`filename`** (*`str`*) – Path to output file, including extension. If None, write to standard output instead.
- **`header`** (*`str`*) – Additional line to prepend to file. If None, no header is used.

galore.plot module

Plotting routines with Matplotlib

`galore.plot.`**`add_overlay`**(*plt*, *overlay*, *overlay_scale=None*, *overlay_offset=0.0*, *overlay_style='o'*, *overlay_label=None*)

Overlay data points from file over existing plot

Parameters

- **`plt`** (*`matplotlib.pyplot`*) – Pyplot object with target figure/axes active
- **`overlay`** (*`str`*) – Path to overlay data file
- **`overlay_scale`** (*`float`*) – y-axis scale factor for overlay data. If None, scale to match maximum and print this value.
- **`overlay_offset`** (*`float`*) – x-xaxis offset for overlay data
- **`overlay_style`** (*`str`*) – Matplotlib short code for marker/line style
- **`overlay_label`** (*`str`*) – Legend label for data overlay (default: filename)

`galore.plot.`**`guess_xlabel`**(*units=None*, *flipx=False*, *energy_label=None*)

Infer a decent x-xaxis label from available information

Parameters

- **`units`** (*`str`*) – Energy or frequency unit string
- **`flipx`** (*`bool`*) – Is energy scale negated to form binding energy
- **`energy_label`** (*`str`*) – Header from .dat file if used

`galore.plot.`**`plot_pdos`**(*pdos_data*, *ax=None*, *total=True*, *show_orbitals=True*, *offset=0.0*, *flipx=False*, ***kwargs*)

Plot a projected density of states (PDOS)

Parameters

- **`pdos_data`** (*`dict`*) – Data for pdos plot in format:

  ```
  {'el1': {'energy': values, 's': values, 'p': values ...},
   'el2': {'energy': values, 's': values, ...}, ...}
  ```

  where DOS values are 1D numpy arrays. For deterministic plots, use ordered dictionaries!
- **`ax`** (*`matplotlib.Axes`*) – Use existing Axes object for plot. If None, a new figure and axes will be created.
- **`total`** (*`bool`*) – Include total DOS. This is sum over all others. Input x-values must be consistent, no further resampling is done.
- **`show_orbitals`** (*`bool`*) – Show orbital contributions. If False, they will not be plotted but are still used to calculate the total DOS.
- **`offset`** (*`float`*) – Bias x-axis values (e.g. to account for XPS E-Fermi),
- **`flipx`** (*`bool`*) – Negate x-axis values to express negative VB energies as positive binding energies.

Returns The pyplot state machine. Can be queried to access current figure and axes.

Return type (matplotlib.pyplot)

`galore.plot.`**`plot_tdos`**(*xdata*, *ydata*, *ax=None*, *offset=0.0*, ***kwargs*)

Plot a total DOS (i.e. 1D dataset)

Parameters

- **xdata** (*iterable*) – x-values (energy, frequency etc.)
- **ydata** (*iterable*) – Corresponding y-values (DOS or measurement intensity)
- **show** (*bool*) – Display plot
- **offset** (*float*) – Energy shift to x-axis
- **ax** (*matplotlib.Axes*) – If provided, plot onto existing Axes object. If None, a new Figure will be created and the pyplot instance will be returned.

Returns The pyplot state machine. Can be queried to access current figure and axes.

Return type (matplotlib.pyplot)

galore.cli package

These submodules implement the command line tools. For usage information see *Command-line interface*. Full API documentation is not provided; it is expected that Python users will use the main Galore API.

Submodules

galore.cli.galore module

galore.cli.galore.**get_parser**()

Parse command-line arguments. Function is used to build the CLI docs.

galore.cli.galore.**main**()

galore.cli.galore.**pdos_from_files**(*return_plt=False*, ***kwargs*)

Read input data, process for PDOS before plotting and/or writing

Parameters

- **return_plt** (*bool*) – If True, return the pyplot object instead of writing or displaying plot output.
- ****kwargs** – See command reference for full argument list

galore.cli.galore.**run**(***kwargs*)

galore.cli.galore.**simple_dos_from_files**(*return_plt=False*, ***kwargs*)

Generate a spectrum or DOS over one data series

kwargs['input'] can be a string or a list containing one string. In addition to main kwargs documented for CLI

Parameters

- **return_plt** (*bool*) – If True, return the pyplot object instead of writing or displaying plot output.
- ****kwargs** – See command reference for full argument list

galore.cli.galore_get_cs module

`galore.cli.galore_get_cs.`**`get_parser`**`()`

`galore.cli.galore_get_cs.`**`main`**`()`

`galore.cli.galore_get_cs.`**`run`**`(`*energy*, *elements*`)`

galore.cli.galore_plot_cs module

`galore.cli.galore_plot_cs.`**`get_parser`**`()`

`galore.cli.galore_plot_cs.`**`main`**`()`

`galore.cli.galore_plot_cs.`**`run`**`(`*elements*, *emin=1*, *emax=10*, *megabarn=False*, *size=None*, *output=None*, *fontsize=10*, *style=None*`)`

Module contents

5.1.2 Module contents

`galore.`**`apply_orbital_weights`**`(`*pdos_data*, *cross_sections*`)`

Weight orbital intensities by cross-section for photoemission simulation

Parameters

- **`pdos_data`** (*`dict`*) – DOS data in format:

```
{'el1': {'energy': values, 's': values, 'p': values ...},
 'el2': {'energy': values, 's': values, ...}, ...}
```

  where DOS values are 1D numpy arrays. Orbital labels must match cross_sections data. It is recommended to use collections.OrderedDict instead of regular dictionaries, to ensure consistent output.

  In addition, the fields “citation”, “link” and “energy” are recognised and logged as “INFO”.

- **`cross_sections`** (*`dict`*) – Weightings in format:

```
{'el1': {'1s': x1, '2s': x2, '2p': x3 ...},
 'el2': {'3s': y1, '3p': y2 ...}, ...}
```

  The labels should correspond to the headers in the input data. It is fine not so specify the level (e.g. use ‘s’, ‘p’, etc.) as is done in the sample data; however, this means that all levels are being treated equally and hence probably the core levels will be weighted incorrectly. It is possible to set the cross-section of undesired orbitals (e.g. projection onto d-orbital for early elements) to None; in this case the orbital will be dropped from the returned data set.

Returns Weighted data in same format as input

Return type weighted_pdos_data (dict)

`galore.`**`auto_limits`**`(`*data_1d*, *padding=0.05*`)`

Return limiting values outside data range

Parameters

- **`data_1d`** (*`iterable`*) – Data to obtain range from

- **padding** (*float*) – Scale factor for padding relative to data range

Returns (2-tuple) (xmin, xmax)

galore.**broaden**(*data*, *dist='lorentz'*, *width=2*, *pad=False*, *d=1*)

Given a 1d data set, use convolution to apply a broadening function

Parameters

- **data** (*np.array*) – 1D array of data points to broaden
- **dist** (*str*) – Type of distribution used for broadening. Currently only "Lorentz" is supported.
- **width** (*float*) – Width parameter for broadening function. Determines the full-width at half-maximum (FWHM) of the broadening function.
- **pad** (*float*) – Distance sampled on each side of broadening function.
- **d** (*float*) – x-axis distance associated with each sample in 1D data

galore.**delta**(*f1*, *f2*, *w=1*)

Compare two frequencies, return 1 if close

galore.**gaussian**(*f*, *f0=0*, *fwhm=1*)

Gaussian function with height 1 centered on f0

f (np.array): 1D array of x-values (e.g. frequencies) f0 (float): Origin of function fwhm (float): full-width half-maximum (FWHM); i.e. the width of the function at half its maximum value.

galore.**lorentzian**(*f*, *f0=0*, *fwhm=1*)

Lorentzian function with height 1 centered on f0.

Parameters

- **f** (*np.array*) – 1D array of x-values (e.g. frequencies)
- **f0** (*float*) – Origin of function
- **fwhm** (*float*) – full-width half-maximum (FWHM); i.e. the width of the function at half its maximum value.

galore.**process_1d_data**(*input=['vasprun.xml']*, *gaussian=None*, *lorentzian=None*, *sampling=0.01*, *xmin=None*, *xmax=None*, *spikes=False*, ***kwargs*)

Read 1D data series from files, process for output

Parameters

- **input** (*str or 1-list*) – Input data file. Pass as either a string or a list containing one string
- ****kwargs** – See main command reference

Returns Resampled x-values and corresponding broadened data as 1D numpy arrays

Return type 2-tuple (np.ndarray, np.ndarray)

galore.**process_pdos**(*input=['vasprun.xml']*, *gaussian=None*, *lorentzian=None*, *weighting=None*, *sampling=0.01*, *xmin=None*, *xmax=None*, *flipx=False*, ***kwargs*)

Read PDOS from files, process for output

Parameters

- **input** (*list or str*) –

Files for processing. Vasp output or space-separated files with XXX_EL_YYY.EXT filename pattern where EL is the element label. We recommend SYSTEM_EL_dos.dat. Alternatively, a *pymatgen.electronic_structure.dos.CompleteDos* can be provided. Spin channels indicated by an (up) or (down) suffix in file header will be combined for each orbital type.

- `**kwargs` – See main command reference

Returns

Weighted and resampled orbital data in format:

```
{'el1': {'energy': values, 's': values, 'p': values ...},
 'el2': {'energy': values, 's': values, ...}, ...}
```

Return type dict

`galore.`**`xy_to_1d`**(*xy*, *x_values*, *spikes=False*)

Convert a set of x,y coordinates to 1D array

Data is resampled to a given sequence of regularly-spaced x-values. By default linear interpolation is used between neighbouring points, which is suitable for resampling distribution data.

Where the data consists of a set of discrete energy levels, it should be resampled to a series of "spikes". y-values are placed on the nearest x-value by subtracting d/2 and rounding up. d is determined by examining the first two elements of x_values.

Parameters

- **`xy`** – (ndarray) 2D numpy array of x, y values
- **`x_values`** – (iterable) An evenly-spaced x-value mesh

Returns re-sampled y values corresponding to x_values

Return type (np.array)

CHAPTER

SIX

CONTRIBUTING

Galore is managed by the Scanlon Materials Theory group at University College London, and is open to contributions from third parties. If you have an idea to make Galore better, please open an Issue outlining your idea on the Github issue tracker. This will allow a public discussion of how well this fits the project design and goals, and how it might be implemented.

6.1 Making changes

All contributions from third parties are managed using "Pull requests" on Github, as are major changes by the core developers. Create a copy of the project on your own Github account using the "Fork" button near the top-right of the web interface and make your changes on a new branch based on `master`. When you are ready to share these changes, create a pull request; this will open a public discussion here.

Before making substantial changes, *please begin discussion in an Issue* so you have some idea if your proposal is likely to be accepted! For minor corrections it may be easier to move straight to a pull request.

Galore is licensed under GPLv3 (see LICENSE file), including any third-party contributions.

6.1.1 Further reading

- A helpful unofficial guide to forking and pull requests on GitHub
- Another unofficial tutorial

6.2 Coding guidelines

- Please follow PEP8 including the 79-character line width limit. You can run a style checker on your code when you are done; a decent one called `pep8` can be obtained with `pip install pep8`.
- It is helpful to start commit messages with a short code indicating the type of change made; this makes it easier to scan the list of commits looking for e.g. documentation changes. The codes are loosely borrowed from this scheme used by the ASE project, but no strict scheme is enforced.
- Please ensure your changes pass the test suite (`python3 setup.py test`) and consider adding tests for any new behaviour. It can be helpful to write the test before you finish implementing the feature.

CHAPTER

SEVEN

CHANGE LOG

Notable changes are logged here by release. This project uses Semantic Versioning. The changelog format is inspired by keep-a-changelog.

7.1 Unreleased

7.2 [0.7.0] - 2021-07-06

- Continuous integration has been migrated from Travis to Github Actions
- Minimum Python version has been increased to 3.5.
  - Python 3.4 was not available through Github actions. It is not wise to support a platform we cannot test.
- BUGFIX: missing import affecting process_pdos

7.3 [0.6.2] - 2021-05-24

- Updated setup.py to add a [vasp] extra; this handles Pymatgen installation which can be tricky on older Python versions.
- Update the [vasp] extra to handle some compatibility breaks between dependencies and different Python versions.
- Fix some incorrect values in Al k-alpha XPS cross-sections
- BUGFIX: Pymatgen CompleteDOS was not correctly accepted by galore.process_pdos()
- Implement previously ineffective "offset" option in galore.plot.plot_pdos(), add a matching option to galore.plot.plot_tdos()

7.4 [0.6.1] - 2018-11-19

- BUGFIX: PDOS plot was failing for elemental systems

7.5 [0.6.0] - 2018-11-02

- Matplotlib styling is exposed to the user: `--style` option selects CLI style while Python API does not overrule existing style.
- The default TDOS line colour is now the first colour from the selected style, which is usually blue. The previous default was a bright red.
- Pymatgen CompleteDos objects can be processed directly.
- Dropped Python 2.7 compatability.
- Fixed PDOS bug introduced with Python3 cleanup

7.6 [0.5.1] - 2018-05-03

- galore-plot-sc can optionally show values in Megabarn/electron

7.7 [0.5.0] - 2018-04-22

- Resample with interpolation by default; use "spikes" only when requested

7.8 [0.4.0] - 2018-04-17

- Import (P)DOS from `.gpw` files generated with GPAW. This requires GPAW to be available.
- galore-plot-sc tool for convenient plotting of cross-section data

7.9 [0.3.0] - 2018-02-19

- BUGFIX: Yeh/Lindau weightings for partially-occupied orbitals
- BUGFIX: Odd behaviour in s orbitals including one transcription error
- Expanded weighting features: HAXPES data parametrised from Scofield tables
- Verbose output including cross sections, warnings and data source citations
- galore-get-sc tool for direct access to cross-section data
- Change Yeh labels from 'xps', 'ups', 'haxpes' to 'alka', 'he2', 'yeh_haxpes'

7.10 [0.2.0] - 2017-09-29

- Gaussian broadening added. In the CLI, this stacks with Lorentzian broadening. Specified as a FWHM.
- Text file output
- Support for PDOS plotting
- Read vasprun.xml files using Pymatgen
- XPS simulation tools; x-axis flipping and PDOS contributions weighted by cross-section.
- Documentation including tutorials, hosted at http://galore.readthedocs.io/en/latest/
- Fancy formatting of units
- Support for files from David Karhanek's IR analysis script and the raman-sc program for simulated optical spectra.
- Source repository made public
- Python API refactored for cleaner scripts

7.11 [0.1.0] - 2016-08-11

7.11.1 Added

- Initial prototype from random data
- Convolution algorithm used to apply Lorentzian broadening
- command-line interface
- Plotting to screen and file
- Test framework
- setuptools-based distribution

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[1] H. Hertz. Ueber einen Einfluss des ultravioletten Lichtes auf die electrische Entladung. *Ann. der Phys. und Chemie*, 267(8):983–1000, 1887. URL: http://doi.wiley.com/10.1002/andp.18872670827, doi:10.1002/andp.18872670827.

[2] E. Rutherford. The Connexion between the and Ray Spectra. *Phil. Mag.*, Sept 1914. There are a number of other papers from Rutherford's group in this volume of Phil. Mag. related to the ejection of -rays (electrons) from samples exposed to -rays (photons). doi:10.1080/14786440908635214.

[3] Maurice de Broglie. Les phénomènes photo-électriques pour les rayons x et les spectres corpusculaires des éléments. *J. Phys. Radium*, 2(9):265–287, sept 1921.

[4] H Robinson. The Secondary Corpuscular Rays Produced by Homogeneous X-Rays. *Proc. R. Soc. A Math. Phys. Eng. Sci.*, 104(727):455–479, nov 1923. URL: http://rspa.royalsocietypublishing.org/cgi/doi/10.1098/rspa.1923.0121, doi:10.1098/rspa.1923.0121.

[5] U. Gelius and K. Siegbahn. ESCA studies of molecular core and valence levels in the gas phase. *Faraday Discuss. Chem. Soc.*, 54:257, 1972. URL: http://xlink.rsc.org/?DOI=dc9725400257, doi:10.1039/dc9725400257.

[6] U. Gelius. Molecular Orbitals and Line Intensities in ESCA Spectra. In D. A. Shirley, editor, *Electron Spectrosc.*, pages 311. North-Holland, Amsterdam, 1972.

[7] U. Gelius. Recent progress in ESCA studies of gases. *J. Electron Spectros. Relat. Phenomena*, 5(1):985–1057, jan 1974. URL: http://linkinghub.elsevier.com/retrieve/pii/0368204874850644, doi:10.1016/0368-2048(74)85064-4.

PYTHON MODULE INDEX

INDEX