Simulation Science Laboratory 2018

An Analysis Tool for Materials Design

Students:
Praneeth Katta Venkatesh Babu
Christian Partmann
Johannes Wasmer

Supervisors:
Stefan Blügel Prof. Dr.
Stefan Rost MSc
Quantum Theory of Materials PGI-1
Forschungszentrum Jülich

Abstract — Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est eleifend mi, non fermentum diam nisl sit amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim. Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras vestibulum bibendum augue. Praesent egestas leo in pede. Praesent blandit odio eu enim. Pellentesque sed dui ut augue blandit sodales. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam nibh. Mauris ac mauris sed pede pellentesque fermentum. Maecenas adipiscing ante non diam sodales hendrerit. Ut velit mauris, egestas sed, gravida nec, ornare ut, mi. Aenean ut orci vel massa suscipit pulvinar. Nulla sollicitudin. Fusce varius, ligula non tempus aliquam, nunc turpis ullamcorper nibh, in tempus sapien eros vitae ligula. Pellentesque rhoncus nunc et augue. Integer id felis.

Keywords Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor.

AICES
Schinkelstr. 2
Rogowski Building
4th Floor
52062 Aachen

Acknowledgments

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est eleifend mi, non fermentum diam nisl sit amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim. Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras vestibulum bibendum augue. Praesent egestas leo in pede. Praesent blandit odio eu enim. Pellentesque sed dui ut augue blandit sodales. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam nibh. Mauris ac mauris sed pede pellentesque fermentum. Maecenas adipiscing ante non diam sodales hendrerit. Ut velit mauris, egestas sed, gravida nec, ornare ut, mi. Aenean ut orci vel massa suscipit pulvinar. Nulla sollicitudin. Fusce varius, ligula non tempus aliquam, nunc turpis ullamcorper nibh, in tempus sapien eros vitae ligula. Pellentesque rhoncus nunc et augue. Integer id felis.

Contents

List of Figures

List of Abbreviations

Chapter 1

Introduction

Problem Statement

Solid State physics deals with the study of large scale properties of solid materials resulting from the atomic scale properties. A solid state physicist can get to know the atomic scale properties from the experiments conducted on the material. One such experiment is DFT(Density Functional theory). This method or experiment is computational Quantum mechanical modelling method to investigate the electronic structure of many body systems of an atom or molecule. Electron density, Inter molecular forces, charge transfer excitations, calculation of band gap are few extractions from DFT simulations.

Fleur is one such DFT simulation which is made by physicist at Juelich Forchungzentrum and most importantly its an open source and anyone can access it and use it. Like any other simulation, DFT simulation also outputs a lot of data so that a solid state physicist can understand and determine the properties of a cell structure of a molecule and from there any physical properties of the material can be determined thereafter. Fleur can be run by digitally specifying the cell structure of a molecule and so can be run to any material and such outputs.

Generally it is used to find/simulate properties of solids with impurities in cell structure. Fluer outputs the data of DFT simulation. It gives the data of ground state and excited state properties of solids. These data which are raw are generally not accessible directly and have to be processed through steps for the solid state physicist to understand the raw data extracted from Fleur DFT simulation. This becomes the major problem and which is dealt in this project. The goal of this project was to implement a complete data analysis pipeline for this application. The steps include preprocessing followed by data exploration and visualization.

Motivation and Requirements

An API which solves the physicist's problem of understanding the simulation data which transforms raw data to useful data such that it is process and visualizes. This API has to process the output files from fleur directly so that it is easier for physicists to get the data processed without external effort. Having said this, modularization and easy maintainability of code also matters since the simulation output keeps varying over the time with the development of simulation code and more data getting collected from simulation.

This API shouldn't only be solving the physicist's problem but also should be fast enough in terms of computation. As known, using of computer code to get outputs and may cause confusions. So an front end GUI is needed such that a physicist can use the GUI to output the processed data. As this is used for research purpose and the features such as plots, images and other should be of high quality so that there is no uncertainty of the results.

Steps

As a project, step by step procedure has completed to complete the task of processing and visualization.

- Understanding the problem: Firstly the theory of DFT simulation, Fleur code, band plots, density plots, file format of data and other necessary theory needed are learnt and understanding about the problem.
- Pre Processing: Once problem is clearly understood, first step of project comes to preprocessing the data. Reading the data, sorting the data and storing it sorted which can be used in further stages of implementation.
- Exploring the data: From the raw data, not just band plots but many features can be extracted from the raw simulation data. Finding out through exploring the data and trying to understand and extract those features.
- Visualization: The data which is preprocesed and extracted need to be visualized into plots such that any user with solid state physics background can understand it by a glance.
- Front End: A GUI is developed so that its easier in future for any physicist to just run the GUI and get the plots instead of going through whole code.
- Results and lookup: Getting to know the features extracted, studying it, reporting the same.

Chapter 2

Theoretical Background

...some introduction scentence depending on into chapter...

Fleur computes the electronic structure in crystals using the Density Functional Theory approach (DFT), which is the state of the art method for this problem. The many-body Schrödinger equation, that can be used to describe electrons in solids, is almost impossible to solve directly, because the storage of the wavefunctions of each of the N electrons in the system at each spatial coordinate exceeds the memory of any currently available computer for even quite small N. This motivates the DFT approach, that uses two fundamental theorems to reduce the computational complexity of the many-body problem significantly: The Hohenberg-Kohn theorem allows to use the electron density instead of the N-electron wavefunctions to uniquely characterize the ground state of a system. With the Kohn-Sham equations, the interacting Hamiltonian of the system can be replaced by non-interacting equations with an effective potential. This approach reduces the dimensionality of the problem from N^3 to 3 and trades the interacting Hamiltonian for a set of non-interacting equations that have to be solved self-consistently. In general, the DFT approach is not just limited to computations of electrons in crystals, but is also for example used in chemistry to compute nonperiodic molecules. The output of a DFT calculation includes various physical quantities (for example the 3-dimensional spatial electron density, which is the central quantity in DFT calculations), but for this project we focus on two quantities that are easy to interpret, already reduced in dimensionality and frequently used in both experimental and theoretical physics.

The band structure $E(\mathbf{k})$ represents the eigenenergies of the eigenfunctions of the Hamiltonian for each crystal momentum $\mathbf{k}$. It is the dispersion relation of electrons in the crystal and relates allowed momenta and energies. In general, $E(\mathbf{k})$ is defined at any point inside the Brillouin zone, which is a special choice of the unit cell in the reciprocal lattice of the crystal. Both, the real-space lattice and the reciprocal lattice are shown for a face centered cubic (fcc) crystal in figure 2.1. For larger unit cells with fewer symmetries, the Brillouin zone can be much more complicated than in the shown example. In order to reduce the dimensionality of $E(\mathbf{k})$ with

$k \in \mathbb{R}^3$, the dispersion relation is only sampled along a discrete one-dimensional path between high symmetry points in the Brillouin zone. This path still contains most of the relevant physical features.

(a) Lattice in real space

(b) Brillouin zone

Figure 2.1: Brillouin zone of a fcc lattice. The red curve in the reciprocal lattice represents a possible sampling path of $E(\mathbf{k})$ in the reciprocal lattice.

The second central quantity in this project is the Density of States (DOS) $D(E)$, which describes the number of states per energy interval that is independent from the crystal momentum. In this sense, the DOS is also derived from $E(\mathbf{k})$, but instead of just selecting a subset, the $\mathbf{k}$ dependency is summed out. Both complementary quantities together are a common choice for comprehensive visualizations of electronic structure data while still capturing most of the important physics.

For applications, it is useful to investigate where the contributions to $E(\mathbf{k})$ and $D(E)$ come from. Therefore, the data contains the weights of each basis function of the DFT calculation belonging to the individual atom groups and the orbitals. This means, spatial information about the system can be restored by considering only contributions from certain atom groups. (An atom group contains all atoms that are equivalent with respect to symmetries of the real space lattice.) On the other hand, the projection on the hydrogen orbitals s, p, d, f encode information about the shape of the wavefunction at each atom. These contributions are stored in the form of relative weights that can be summed to include contributions for multiple groups and orbitals. In case of distinct spins in the crystal, $E(\mathbf{k})$, $D(E)$ and the weights can be different for both spins and are therefore stored individually.

Chapter 3

Implementation

As per the requirements expounded upon in the introduction, the deliverable of the project should be a finished software product. The software is written in Python so as to integrate easily with the research group's ongoing software projects around the Fleur code [Blü+18]. These are chiefly the group's materials science tool collection `masci-tools` [RBR18] , where also this project's code is hosted, and the 'Automated Interactive Infrastructure and Database for Computational Science' (AiiDA) [Piz+16]. The product stakeholders split into frontend users and code developers. In order to accommodate this, the product is organized into three unidirectionally dependent subpackages or -modules, see Figure 3.1a.

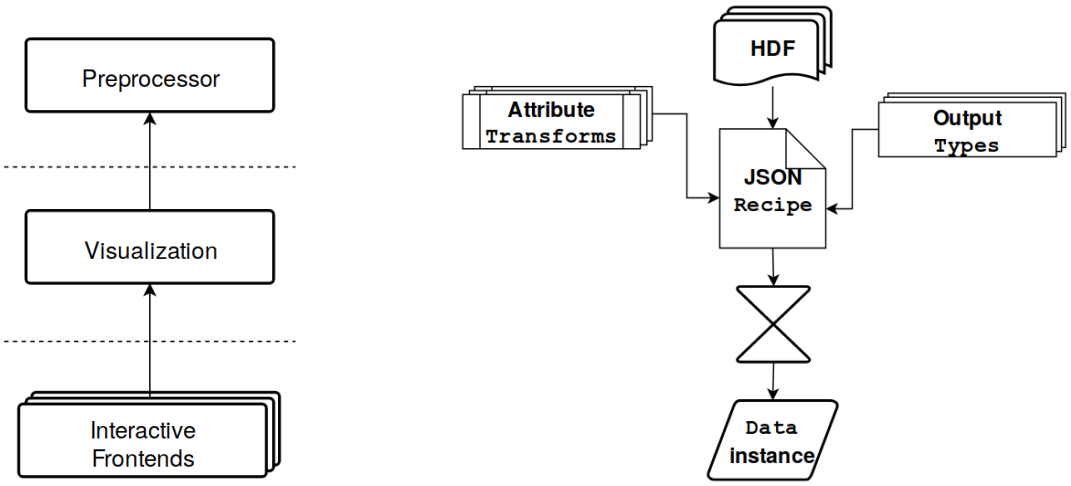
An important design consideration was to account for unknown use cases. This has been realized in each submodule by the decoupling of **interface** and **implementation**. The interfaces do not rely on any specific input file format, visualization method or package, unlike the implementations for a specific task or **application**. The word 'application' in this section denotes the band structure and density of states visualization, and for these, implementations are provided.

This design choice was also one reason why the product does not reuse any of the `masci-tools` routines which partly solve quite similar problems, but seemed to be too specialized in an cursory code review. For these developers, one added value of the project product could be to inspire the hopefully easy integration into a common interface, where the current abstraction level could serve as a starting point.

HDF Preprocessor Module

Interface

This is the 'backend' of the tool. It is basically a file reader for the input data, for example a Fleur simulation output. Supported formats are the Hierarchical Data Format (HDF) [Kor11]



(a) Submodules (b) Preprocessor

Figure 3.1: Module Design.

for the band structure, and a simple Fleur-specific comma-separated values (CSV) format for the density of states (DOS).

The HDF format is basically a binary flexible container for all kinds of common binary and text file formats, each of which constitutes a `Dataset` inside the HDF file. The format supports metadata annotation and high-throughput input/output (I/O). As a consequence, it is considered by some developers in some application domains relying on numerical simulation codes, to be one possible base for the establishment of common domain-specific rich data exchange standards in order to increase code interoperability. These developers are in the process of extending their codes' I/O capabilities towards that end. However, HDF's flexibility comes at the price of a relatively complex Application Programming Interface (API) as the keyhole for all operations.

The preprocessor module tries to hide that complexity by offering the `Recipes` interface, see Figure 3.1b. A specific application `Recipe` is a dictionary that aims to describe a complete Extract-Transform-Load (ETL) pipeline for one specific application. The 'extract' is the reading of a dataset from HDF, the 'transform' a sequence of once-through functions applied to the the dataset, and the 'load' the aggregation of all transformed datasets into one runtime object that has all the methods for operations on the data that are going to be used later on in the intended application.

The 'transform' and 'output' type methods are defined in hierarchical `Transform` and `Output_Type` classes, which sort them from general to application-specific applicability. This structure is built using Python's `AbstractBaseClass` (`ABC`) interface and multiple inheritance. The advantages of the 'Recipes approach' are:

- All ETL processes for one application are collected in one simple list (the recipe), not

locked in different code locations with conflicting contexts. In this list, entries can be sorted in any manner, e.g. alphabetical for perusal.

- Recipes are de/serializable (can be read from and saved to disk) and thus be created and manipulated by code.
- The ETL processes declared in this way can be easily reused across applications. A recipe can combine different output types into a new type.

The feature that enables this flexibility is **type introspection**: the preprocessor processes the datasets listed in the recipe in the order of their mutual dependencies as found in the used transform and output methods. When all transformed datasets have been added to the object, all specified output types are searched and all their methods and attributes added. Thus the output object's type is defined at runtime, when the preprocessing is finished.

Implementation for Band Structure Visualization

TODO describe how bandstructure data is preprocessed for visualization including user selections AND optimizations

The frontend has to draw three kinds of plots: a 3D atom plot of the unit cell or supercell, and a combined band structure and DOS plot sharing the same vertical energy axis. If no DOS data is present, the DOS plot will be omitted. All three plots are controlled by one set of widgets for varying the parameters. In the current implementation, the data for the first two plots come from a HDF file, while the data for DOS plot come from CSV files.

The band structure plot is a scatter plot and plots discrete $E(\mathbf{k})$ data from the simulation. It first needs the k-path (where $|\mathbf{k}| = k$) for the horizontal axis. The preprocessor, having received the recipe `FleurBands`, computes it from the k-points in the HDF in a transform. Next, the plot needs the eigenergies for every point on the k-path, labeled by the band index ν, and its associated l-like charge $n_{s,k,\nu,g,l}$. It is fivedimensional, and represents the contribution of spin s, point k point on k-path, band ν, atom group g, and character or orbital l (here: only s,p,d,f) to the specific eigenergy. In order to visualize this, resolves the processed data into the like-named output type `FleurBands`. This type has a data filter method. The respective `BandPlot` type calls this filter with a user selection of subsets of all (s, k, ν, g, l). The method then computes the according **effective weight** shown in Equation 3.1. This is used for the dot size of each $E(k)$ in the plot. Before rendering, the plotter normalizes the energies to the Fermi Energy.

$$W^{\text{eff}}_{s,k,\nu} = \left(\frac{\sum\limits_{\substack{g \in \text{groups} \\ l \in \text{characters}}} n_{s,k,\nu,g,l}\, N_g}{\sum\limits_{\substack{g \in \text{all groups} \\ l \in \text{all characters}}} n_{s,k,\nu,g,l}\, N_g} \right) \left(W^{\text{unf}}_{s,k,\nu} \right)^{\alpha} \tag{3.1}$$

N_g denotes the number of atoms in a group. $W^{\text{unf}}_{s,k,\nu}$ is the unfolding weight and α its exponent. The effect of unfolding is illustrated in Figure 3.2 for a toy example of a monatomic chain, with a two-atom supercell of size a' representing $\alpha = 0$ and a one-atom unit cell of size a representing $\alpha = 1$. A use-case is discussed in the Application Chapter 5.

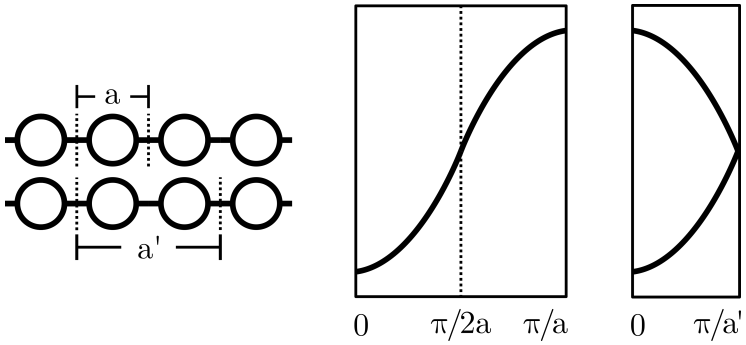


Figure 3.2: Band Unfolding: Example for a simple monatomic chain [Hof87].

Even for small band structures, the filter has to access on the order 10^7 individual data points for every selection change. Optimizations were introduced which included: a cutoff skipping effective weights too small,use of optimized Numpy functions like `np.tensor`, data buffering, and array reshaping. Together, these achieve a speedup of approximately 10^2 in plotting speed. Thanks to that, the tool remains usable even when input data is in the 10^2 MB range.

Visualization Module & Interactive Graphical Frontends

TODO Frontends: combine into one subsection when finished. usage will be in user manual next section.

Visualization Module

The Python visualization landscape abounds with a rapidly evolving plethora of plotting libraries for different application contexts and technology stacks [VR17]. Thus the project's visualization module first design objective was to account for that by decoupling from specific library use, and modularizing applications. This structure again is built using Python's `AbstractBaseClass` (`ABC`) interface and multiple inheritance. Each application is represented by an abstract base class that contains the common plotting method signatures. Each plotting library is represented by an abstract base class that contains library-specifics. An implementation inherits both from one library base class and one or more application base classes. See Fig. 3.3 for an impression. Thus switching the library in a frontend should require minimal adjustment, and a new application can be build using existing ones.

The second design objective was for the plotting methods to to hide all interactions with the actual plotting library used under the hood, while the method arguments only relate to the preprocessed data being plotted. Thus different frontend implementations only all call one common method for one specific plot and receive the identical visualization with identical interactive behavior.

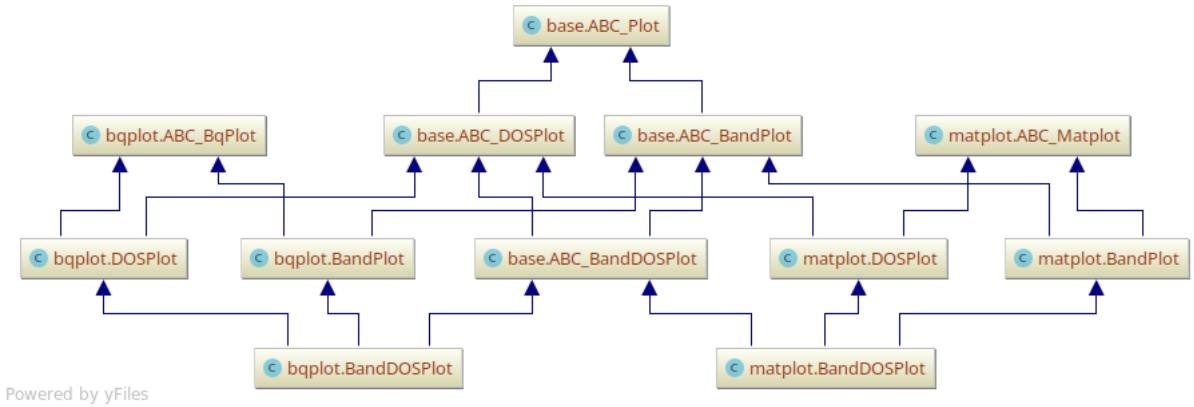


Figure 3.3: Visualization Module Design: Example Structure for two applications `BandPlot` and `DOSPlot`, and two plotting libraries `matplotlib` and `bqplot`.

Desktop Frontend

A Desktop Frontend is always a helping had for the physicists to just run it on the computer when one wants to go through the plots of the raw data that is already in computer. Since the reading of HDF files and preprocessing of the data is done using Python, it is decided that it would be better to use python for front end development. Considering various packages for front end in python such as PyQt, Tkinter and other packages available for Desktop frontend, the conventional tkinter is selected based on the maintainability of the code. It is a package where every button can be designed and can be assigned to function. Functions like label, button, checkbutton, listbox, canvas for plots, tab for viewing each plot in different tab are used to make a simple Desktop front end. It is simple to use with limited options of what any physicist needed and also easy to convert it into a executable software and run in any system without any installation.

Web Frontend

As Web Frontends increasingly replace traditional Desktop Frontends in the modern software environment [Dou+08], so Python-based Frontends and visualizations are increasingly moving towards the browser. There, GUIs with interactive visualizations are often called **Dashboards**.

In the project context, a survey was undertaken in order to choose the most suitable technology stack. The full survey is documented in [Was19]. The requirements for the solution stack, added to those in the introduction, were as follows:

1. **openness**: relies solely on Open-Source-Software (OSS) with licensing suitable for academic use, sports a stable release cycle, developer base and documentation,
2. **dashboarding**: features graphical control elements (widgets) that interact with InfoVis[1] plotting libraries,
3. **deployment**: ideally works like any web service, i.e. only a modern web browser is required to use it,
4. **maintenance**: requires only Python and no Web Development knowledge like e.g. Javascript, with respect to the product stakeholders.

The last point implies a client-server model where the dashboard app is hosted by a remote service. This model requires a communication framework and protocol between the Python interpreter running on the server and the JavaScript interpreter running in the client browser. As per requirement number two, unlike a generic Python web framework like e.g. Flask, the framework should take care of that communication by itself. Four major frameworks were identified which fulfill the first three requirements: Project Jupyter, PyViz, Bokeh, and Dash by Plotly. The last two only partially fulfilled the last requirement, so they were discarded. PyViz is the newest contender among the four. Its expressed goal is to untangle the Python visualization jungle by providing one high-level API that ties together all major Python InfoVis libraries and data formats, including dashboarding. That ambitious goal comes at the price of sacrificing support for 3D plotting [aut18], which was needed in this project for the atoms plot. So PyViz had to be discarded.

That left Project Jupyter. By now, it's widget library `ipywidgets` is integrated to work with a wide variety of popular plotting libraries. However, Jupyter only partially fulfills the third requirement – a Jupyter notebook (app) cannot, by itself, be published (deployed) as a standalone website outside a live Jupyter environment [Bed18]:

> [...] "However, despite their web-based interactivity, the ipywidgets-based libraries (ipyleaflet, pythreejs, ipyvolume, bqplot) are difficult to deploy as public-facing apps because the Jupyter protocol allows arbitrary code execution" [...].

To avoid requiring users to setup a working Jupyter environment on their machine, the go-to solution for this problem is to setup a JupyterHub multi-user server. This still requires users to register an account there. Fortunately though, the intended users are contributors to the AiiDA project, and so should have access to the JupyterHub-based AiiDaLab service where the app can be registered. Details on this procedure and alternative hosting solutions can be found

[1] InfoVis libraries: visualizations of information in arbitrary spaces, not necessarily the three-dimensional physical world. Example: matplotlib. SciVis libraries: visualizing physically situated data. Example: VTK [Bed18].

in the developer Section 4 on page 14.

Chapter 4

Manual

The project code and documentation is hosted on the `masci-tools` repository [RBR18] under the branch `studentproject18ws`. All of the project code resides in the folders `binder` (for the Web Frontend Demo) and `studentproject18w` (all code and documentation). The `README.md` serves as the manual. Therefore, the remaining part of this chapter is a TEX-ified version of that `README.md`.

SiScLab 2018 Student Project **Analysis Tool for Materials Design**. Written in Python3.

Authors: Johannes Wasmer, Christian Partmann, and Praneeth Katta.

Overview

This subfolder `studentproject18ws` is currently a largely independent side-project accompanying the main module `masci-tools`. It was created in a student project, and consists of three submodules:

- preprocessor: a HDF reader interface, and one implementation for Fleur band structure simulation output
- visualization: a plotting interface, and one implementation for Fleur bandstructure+DOS plots
- frontends: a Desktop GUI and a Web Dashboard (Tk and Jupyter) for interactive Fleur bandDOS plots.

A more thorough description and example use cases can be found in the project report and presentation.

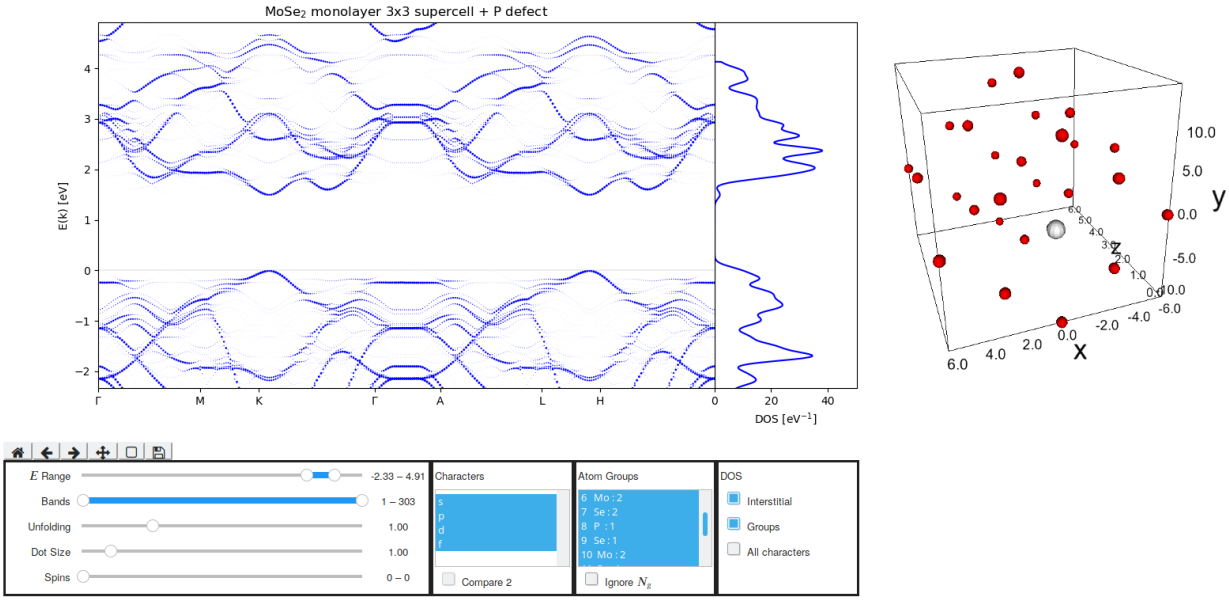


Figure 4.1

For Frontend Users

The Desktop GUI executable can be received from the developers on request. Otherwise, it can be built using PyInstaller from this repo.

The Web Frontend is a Jupyter Dashboard. It is in experimental phase (no fileupload yet). You can try it out here on Binder. You can run it locally (see developer section). If you have an AiiDaLab account: the dashboard is planned to be published as an app there.

For Developers

Installation

Though `masci-tools` is availabe via PyPI, there is currently no plan to integrate `studentproject18ws`. If you want to use it in your code, clone the repo, use it in an IDE, or append the path to your `sys.path`:

```
import sys
if path_repo not in sys.path:
    sys.path.append(path_repo)

# now import works
from studentproject18w.hdf.reader import Reader
```

```
# ...
```

Create project virtual environment

With conda (recommended): - Install Anaconda (3 recommended) - Install the environment `masci-stupro` with the necessary and recommended dependencies:

```
conda create -f environment.yml
source activate masci-stupro
```

With virtualenv (untested):

```
virtualenv masci-stupro
source masci-stupro/bin/activate
pip install -r requirements_pip.txt # install requirements
```

Programmatic use

In this example, a Fleur HDF file is preprocessed using the Recipe `FleurBands`. The resulting output `data` with the extracted and transformed HDF datasets and attached load methods (Extract-Transform-Load) is then passed to a plotter, alongside some DOS CSV files for a bandstructure plot using `matplotlib` as backend library.

```
import matplotlib.pyplot as plt
from studentproject18w.hdf.reader import Reader
from studentproject18w.hdf.recipes import Recipes
from studentproject18w.plot.matplot import BandDOSPlot

data = None
reader = Reader(filepath=filepath_hdf)
with reader as h5file:
    data = reader.read(recipe=Recipes.FleurBands)
    #
    # Note:
    # Inside the with statement (context manager),
    # all data attributes that are type h5py Dataset are available (in-file
        ↪access)
    # When the statement is left,the HDF5 file gets closed and the datasets are
        ↪ closed.
    #
    # Use data outside the with-statement (in-memory access: all HDF5 datasets
        ↪converted to numpy ndarrays):
```

```
    data.move_datasets_to_memory()

plotter = BandDOSPlot(plt, data, filepaths_dos)
(fig, ax_bands, ax_dos) = plter.setup_figure(fig_ratio=[12,6], fig_scale=1,
                    ↪fig_title="BandDOS")
data_selection = some_selection_process()
plotter.plot_bandDOS(*data_selection)
plt.show()
```

Try out Web Frontend locally

The demo notebook with the Dashboard is studentproject18w/frontend/jupyter/demo/demo.ipynb.

If using Jupyter Notebook Notebook

If using Windows, omit keyword source.

```
source activate masci-stupro
cd mypath/masci-tools/studentproject18ws/
jupyter-notebook .
# if Home is not set to this dir, try this instead:
# /home/you/anaconda3/envs/myenv/bin/python /home/you/anaconda3/envs/myenv/bin/
                    ↪jupyter-notebook .
```

If using Jupyter Lab

Additional installation step needed:

```
source activate masci-stupro
jupyter labextension install @jupyter-widgets/jupyterlab-manager jupyter-
                    ↪matplotlib ipyvolume
cd mypath/masci-tools/studentproject18ws/
jupyter-lab
```

To-do list for publishing the Web Frontend

- (recommended: create frontend/jupyter/Dashboard.py widget and put code of demo_back.ipynb notebook inside it. Use aiidalab-widgets-base > StructureUpload-

Widget as a template. Create `frontend/jupyter/Dashboard.ipynb` notebook. Use StructureUploadWidget Demo Notebook as a template.)

- Add fileupload to widget (again, like in StructureUploadWidget. See binder_fileupload_test.ipynb notebook for a demo that works with Binder.)
- Now the Web Frontend should work on Binder.
- For publishing the app on AiiDA Lab, the app has to be registered in the aiidalab-registry.
  - The project code is in Python3, but aiidalab requires Python2. So the code has to first be backported by hand using the `future` package. If this takes too long, maybe try the tool 3to2.
  - Use the simplest app in the registry, aiidalab-units as a template. Adapt code.
  - Try it out first in the Quantum Mobile Virtual Machine, which has aiidalab installed and configured. Else try it in a virtual environment with aiidalab installed from PyPI.
  - Register the app.

Note: other publishing options besides Binder and AiiDALab are listed here. For instance, Google Colaboratory is a free Notebook hosting service that allows file upload.

Chapter 5

Applications

To illustrate the use of the graphical user interface, two different physical applications are shown in the desktop and the web frontend, respectively. From the physics point of view, the example in the desktop version focuses more on the density of states and the visualization of spin contributions, while the dataset in web frontend focuses on the band structure $E(\mathbf{k})$ and the visualization of defect states in supercells.

Web Frontend: $\mathrm{MoSe_2}$ Crystal

Figure 5.2 shows the visualization of a band structure calculation of a 3 dimensional Molybdenum diselenide ($\mathrm{MoSe_2}$ bulk) crystal using the default settings of the GUI. Even with the default settings the band structure plots clearly indicate, that $\mathrm{MoSe_2}$ is a semiconductor, since there are no states at the Fermi level. Because the minimum of the conduction band is located at an other k as the maximum of the valence band, the plot shows that $\mathrm{MoSe_2}$ has an indirect band gap. This indicates that for the transition with the smallest energy difference between valence and the conduction band, both, energy and momentum have to change.

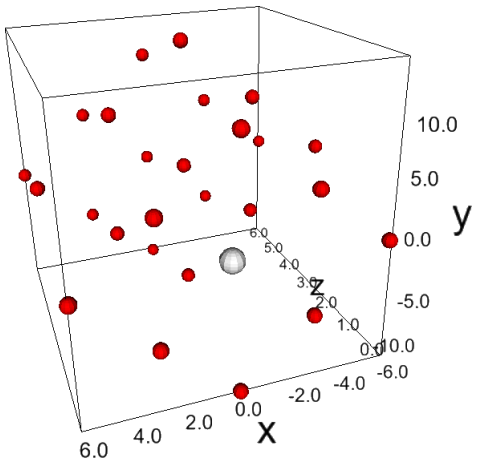


Figure 5.1: Atom Plot of a $\mathrm{MoSe_2}$ monolayer with the defect atom selected in the web frontend

In contrast to the 3 dimensional extended $\mathrm{MoSe_2}$ crystal, a $\mathrm{MoSe_2}$ monolayer (see Fig. 5.3) has a direct band gap but is still a semiconductor. Furthermore, the $\mathrm{MoSe_2}$ monolayer has a defect atom in every 9th unit cell and the DFT computation is therefore done in a 3×3 supercell to restore periodicity. This is the reason for the much greater number of states

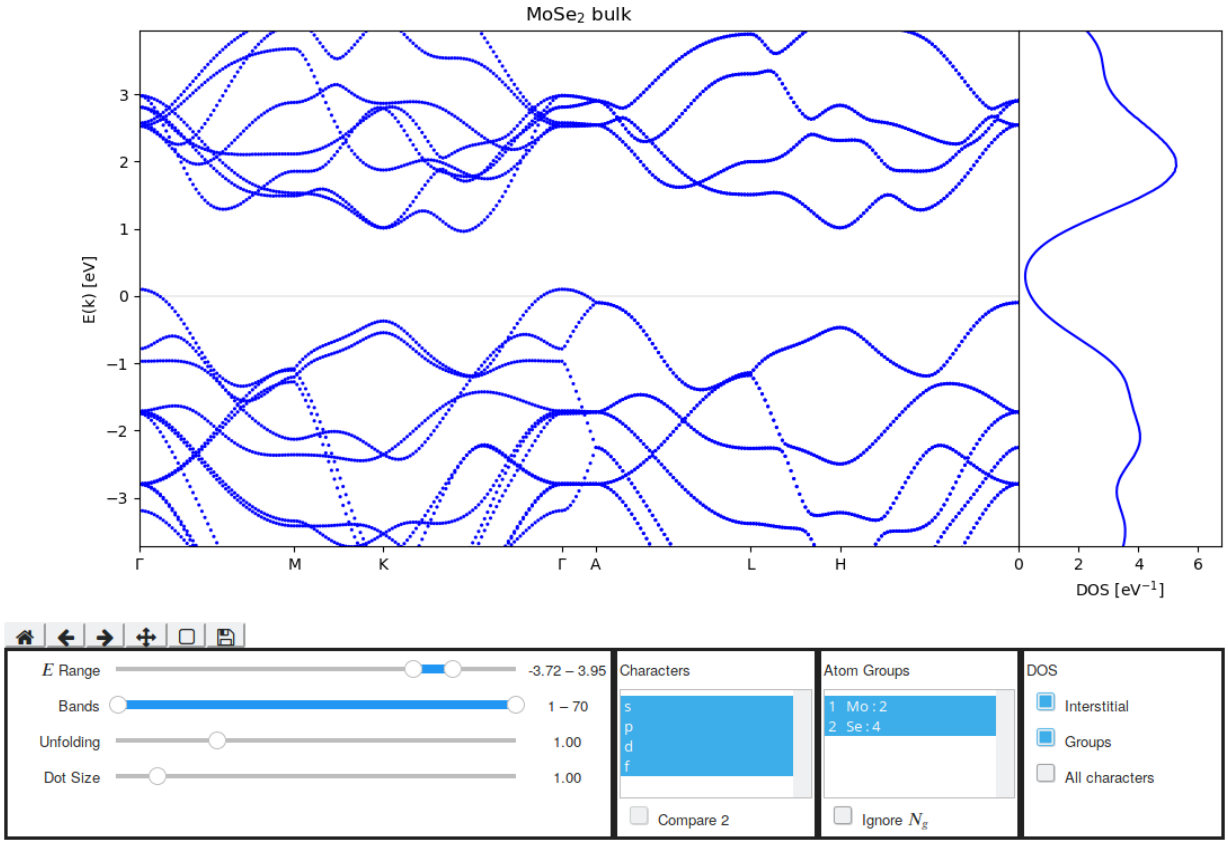


Figure 5.2: Bandstructure of a $MoSe_2$ crystal using the default settings of the web frontend

in the band structure plot.

Since the Brillouin zone of the supercell is smaller than the Brillouin zone of the same crystal without the defect, the supercell Brillouin zone is unfolded to the same size as the Brillouin zone of the unperturbed lattice. To account for the fact that the defect is only present in every 9th cell and its relative importance for the spectrum is therefore degraded, an unfolding weight is introduced to visualize the relative importance of bands in the unfolded Brillouin zone. By default, the unfolding weight is used by our visualization tool, but it can gradually be turned off in order to highlight the impact of defect states. This is shown in figure 5.4. To even better visualize the defect state, it would also be possible to select the atom group belonging the defect atoms only. In this example, the analysis with reduced band unfolding shows, that there are many more direct band gaps origination from the defect state.

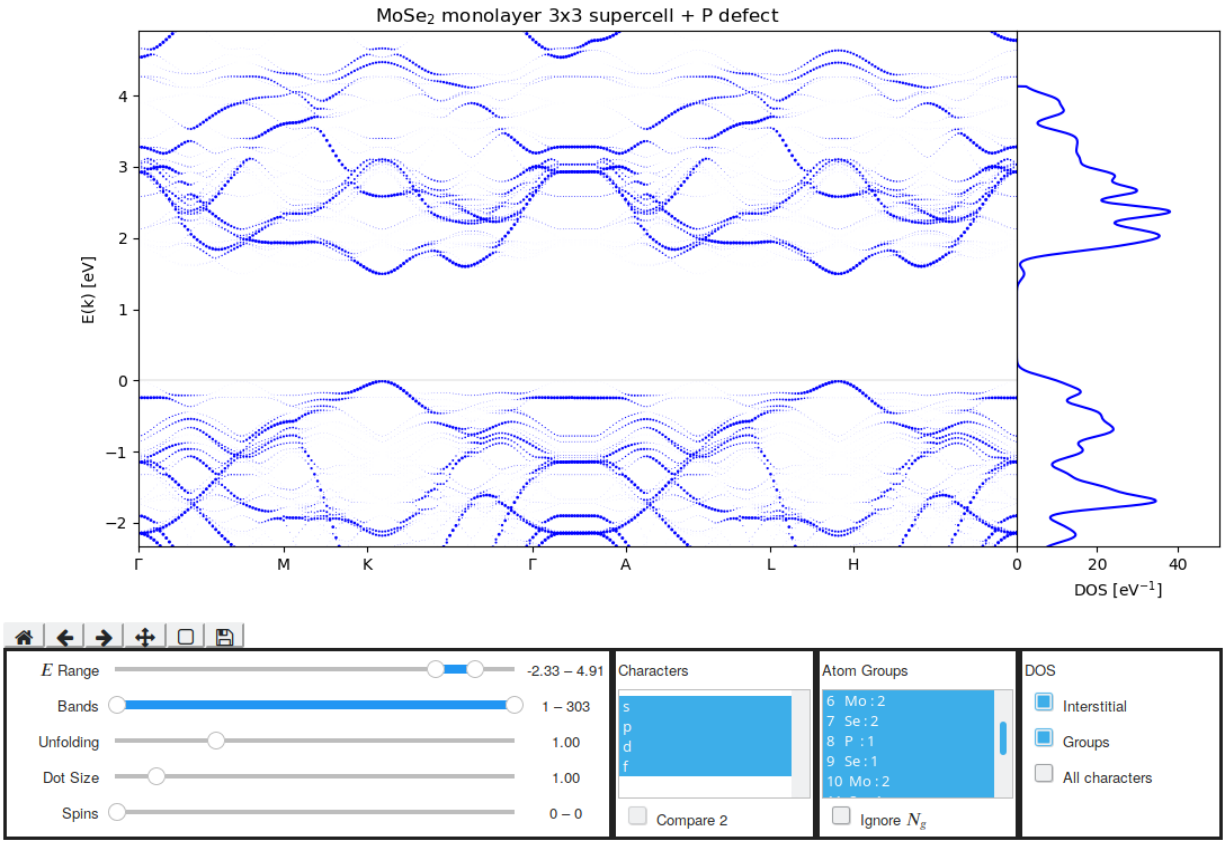


Figure 5.3: Bandstructure of a $MoSe_2$ monolayer using the default settings of the web frontend

Desktop Frontend: Co Crystal

Figure **TODO** shows the visualization of a band structure calculation of a 3 dimensional Co crystal with settings of dual spin, one atom group(since it is the only crystal in the cell structure) with all characters and whole band selected with exponent being at 0.0. We can see in the figure that band plot of any spin intersecting with the Fermi energy level. This means that many electrons excite to valance band which is a characteristic of a conductor. Hence the Co Crystal can be considered and concluded as conductor material.

One more physical property of Co crystal can be determined by considering and reading the density of state(DOS) files and plotting the density of state along with the band plots. In the figure **TODO**, on the right one can see the Density of states of both the spins. If the area covered under the density is considered, it is clear that under the Fermi energy, electrons of one spin covers more area than other spin in the density of states plot. This means there are more electrons with positive spin under the Fermi energy. This is the character exhibited by the Ferro magnetic substance, so it be considered and concluded that this Co crystal is a Ferro magnetic.

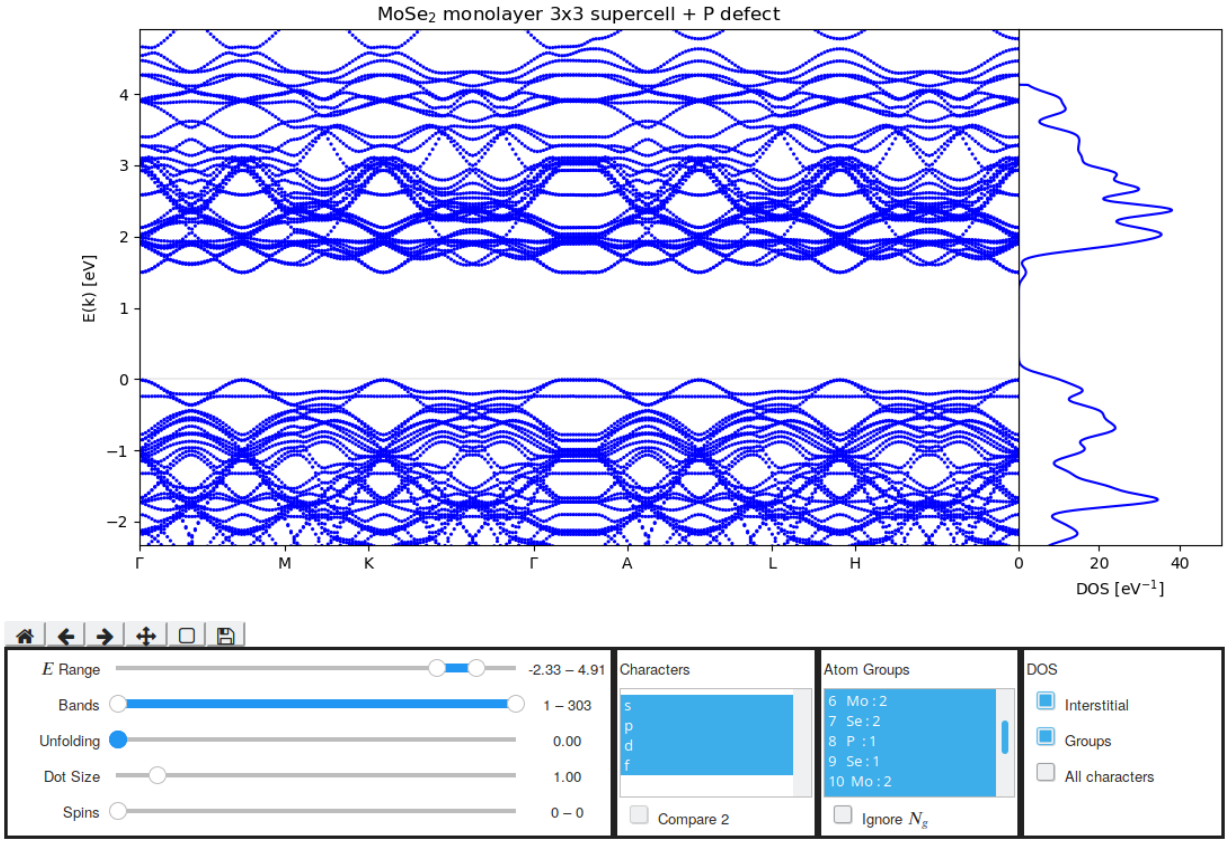


Figure 5.4: Bandstructure of a $\mathrm{MoSe_2}$ monolayer without unfolding weights

There are two properties which are concluded by observing the band-dos plot but depending on the material and band-dos plot many properties can be extracted and concluded.

Derived physical Quantities: Differentiation

The kind of datasets handled in the scope of the project did not lend themselves readily to applications of automatic differentiation techniques. Nevertheless, it is possible to derive meaningful physical quantities from the band structure using numerical differentiation techniques.

The effective mass m^* represents the mass, that an electron appears to have due to the interatomic forces in the crystal. At every $\mathbf{k}_0$, where $E(\mathbf{k})$ has a local extremum, $E(\mathbf{k})$ can be expanded in a Taylor series with a vanishing first order term $E(\mathbf{k}) = E_0 + \frac{\partial E(\mathbf{k})}{\partial \mathbf{k}} \cdot (\mathbf{k} - \mathbf{k}_0)^2$. Comparing this to the dispersion relation of a free electron $E(\mathbf{k})_{free} = \frac{\hbar^2 \mathbf{k}^2}{2m_e}$ motivates the

general definition

$$m^*_{k_i,k_j} = \hbar^2 \left(\frac{\partial^2 E(\mathbf{k})}{\partial k_i \partial k_j} \right)^{-1} \tag{5.1}$$

Since $\frac{\partial^2 E(\mathbf{k})}{\partial \mathbf{k}_i \partial \mathbf{k}_j}$ depends on the direction of the partial derivatives, $m^*_{k_i,k_j}$ is a tensor. Because the band structure files only contain a discrete sampled path in the Brillouin zone, only the derivatives that correspond to the direction from one high-symmetry point to the next can be computed. We are only interested in the diagonal terms of $m^*\big|_{i,j}$.

A second potentially interesting quantity is the group velocity $v_G(\mathbf{k})$ associated to each band. The group velocity $v_G(\mathbf{k})$ at the Fermi energy $E = E_F$ is called Fermi velocity.

$$v_G(\mathbf{k})_{k_i} = \frac{1}{\hbar} \frac{\partial E(\mathbf{k})}{\partial \mathbf{k_i}} \tag{5.2}$$

Differentiation

Since the k-mesh in DFT calculations is potentially very sparse, low order finite difference schemes are not expected to work well. Alternatively, one way to exploit all data points efficiently is to use fast Fourier transform methods, which are equivalent to the derivation of a truncated Fourier series. This method is expected be well-suited for the problem since the graph of the band structure $(k, E(|k|))$ with $k \in \{-\Gamma, ..., H, ..., \Gamma\}$ is periodic, where Γ and H are arbitrary representatives of the high symmetry points. This periodicity in the reciprocal space is a direct consequence of the periodicity of the crystal.

In Fourier space, spatial derivatives transform into multiplications, which can easily be shown by partial integration.

$$f^{(n)}(x) = \mathcal{F}^{-1}\left((ik)^n \mathcal{F}(f(x))\right) \tag{5.3}$$

To test the FFT differentiation method, a band was selected that did not have intersections with other bands within the interval between two high symmetry points and a stationary point at the high symmetry points. Then a resolution study was done to investigate the impact of the number of points within the interval.

The comparison between the FFT and a first-order central difference approximation of the second derivative (FD) is shown in Fig. 5.5, where different k-mesh resolutions are compared to a derivative that is almost fully converged. The comparison indicates that for small N, the error of the FFT method is significantly smaller than the error of the FD derivative. This is especially striking in the vicinity of the high symmetry points, where the error of the differentiation is

required to be small in order to get good approximations for m^*, which is only meaningful close to the high symmetry points.

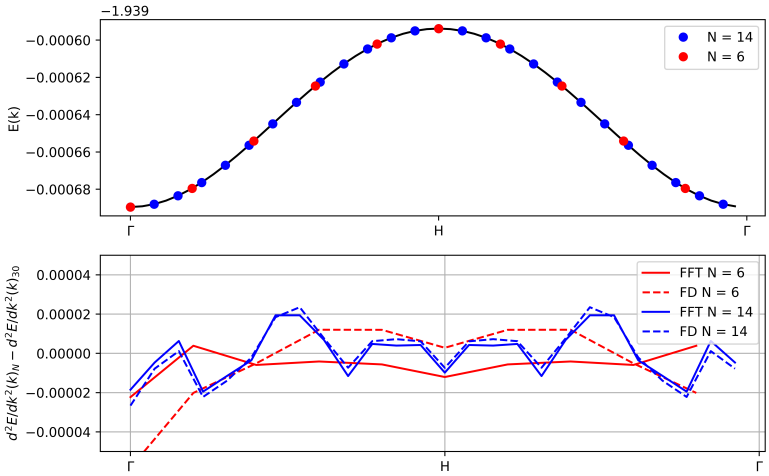


Figure 5.5: Comparison between FFT differentiation method and central finite difference method

When using more points, the difference between both approximation schemes is not significant. It is interesting to note, that the FFT method does not work anymore in the limit of extremely many points. In this case the derivative is dominated by Gibbs oscillations. These are most likely caused by small discontinuities in $E(\mathbf{k})$, due to basis changes inside the DFT computation.

At the current state the question remains to be answered weather the computation of m^* and v_g is useful. The number of bands, which the computation can be applied to is extremely limited. Since the bands in the data files are not labeled according to their corresponding eigenfunction but sorted by value, there are discontinuities at each point, where two bands intersect. This problem might be solved in the future.

Chapter 6

Conclusion

Outlook

TODO Module design:

- Preprocessor module: dataset dependencies for new recipes have to be explicitly stated. This could also be automatically resolved by type inspection.
- Frontends:
  - PyViz Param [Ste+19] makes it possible to decouple the formal description of a particular GUI from the GUI library used. This would serve to separate interface and implementation like it is done in the preprocessor and visualization submodules.

Bibliography

[aut18] PyViz authors. *PyViz FAQ*. http://pyviz.org/FAQ.html. PyViz team. 2018.

[Bed18] James Bednar. *Python Data Visualization 2018*. Blog. https://www.anaconda.com/python-data-visualization-2018-why-so-many-libraries/ and https://www.anaconda.com/python-data-visualization-2018-moving-toward-convergence/ and https://www.anaconda.com/python-data-visualization-2018-where-do-we-go-from-here/. 2018.

[Blü+18] Stefan Blügel et al. *Fleur: full potential linearized augmented planewave code*. http://www.judft.de. 2018.

[Dou+08] John R Douceur et al. "Leveraging Legacy Code to Deploy Desktop Applications on the Web." In: *OSDI*. Vol. 8. 2008, pp. 339–354.

[Hof87] Roald Hoffmann. "How chemistry and physics meet in the solid state". In: *Angewandte Chemie International Edition in English* 26.9 (1987), pp. 846–878.

[Kor11] Sandeep Koranne. "Hierarchical Data Format 5 : HDF5". In: *Handbook of Open Source Tools*. Boston, MA: Springer US, 2011, pp. 191–200. ISBN: 978-1-4419-7719-9. DOI: 10.1007/978-1-4419-7719-9_10. URL: https://doi.org/10.1007/978-1-4419-7719-9_10.

[Piz+16] Giovanni Pizzi et al. "AiiDA: automated interactive infrastructure and database for computational science". In: *Computational Materials Science* 111 (2016), pp. 218–230. ISSN: 0927-0256. DOI: https://doi.org/10.1016/j.commatsci.2015.09.013. URL: http://www.sciencedirect.com/science/article/pii/S0927025615005820.

[RBR18] Philipp Rüßmann, Jens Bröder, and Stefan Rost. *masci-tools*. https://github.com/JuDFTteam/masci-tools. 2018.

[Ste+19] Jean-Luc Stevens et al. *PyViz Param*. https://github.com/pyviz/param. 2019.

[VR17] Jake VanderPlas and Nicolas Rougier. *Python Visualization Landscape*. https://github.com/rougier/python-visualization-landscape. 2017.

[Was19] Johannes Wasmer. *Software Development Notes*. https://github.com/Irratzo/notes. 2019.