SiScLab Project 8

Analysis Tool for Materials Design

Supervisors: Prof. Dr. Stefan Bluegel, Stefan Rost MSc, Jens Broeder MSc

Quantum Theory of Materials (PGI-1 / IAS-1)

Forschungszentrum Juelich

Katta, Partmann, Wasmer

January 25, 2019

Problem Statement

- Solid state physics: electronic structure computation
  - → Fleur: electronic structure of crystals using DFT
  - Fleur simulation code developed and maintained by Institute of Advanced Simulation-1 at FZJ and is open source
  - huge amount of data
  - physics not accessible unless structured / analysed / visualized

The goal of the project was to implement a complete data analysis pipeline for this application:

- preprocessing → data exploration → visualization

Motivation & Requirements

- to solve physicist's problems with the simulation data
- process Fleur output files
- modularization & easy maintainability of code
- fast computation time
- frontend: no installation required, intuitive usage
- high-quality export features

Visualisation

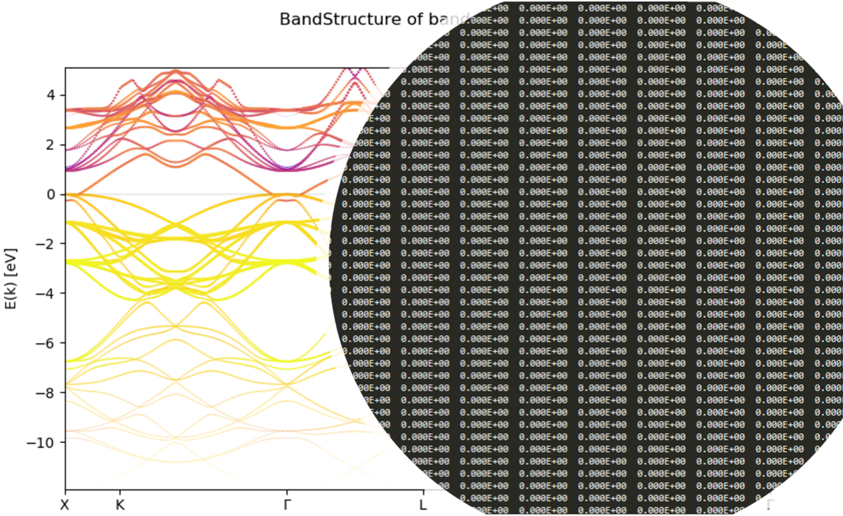


Figure: Transformation

Steps

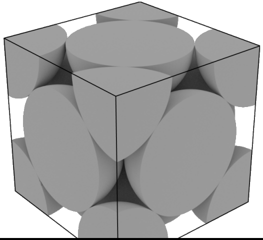
- understanding physics and problem
- preprocessing the data
- exploring the data(implementation)
- visualization & GUI
- results

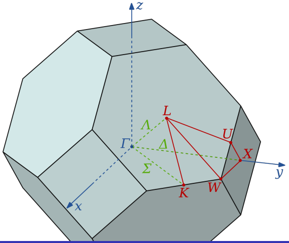
How is the data generated?

- Fleur computes electron density in crystals
- Density Functional Theory (DFT) approach:
  - Hohenberg-Kohn theorem: use electron density
  - Kohn-Sham equations: Solve one particle Schrödinger equations in effective potential (self consistent)
  - State of the art method for electronic structure computations in solids

What data is generated?

- Bandstructure $E_\nu(k)$:
    - Eigenenergies of eigenfunctions of the Hamiltonian for each (crystal-) momentum k
    - Dispersion relation: Relation between crystal momentum and energies of the Bloch electrons
    - Sampled along a 1D path between high symmetry points in 3D reciprocal space

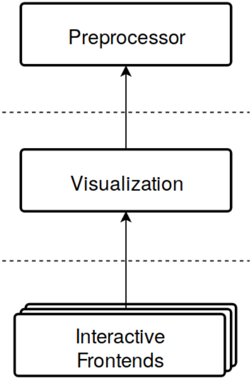




- Bandstructure D(E):
  - Density of electron states per energy interval

- Interesting for physicists: Where do the contributions to E(k) and D(E) come from?
  - Contributions from basis functions of the DFT calculation corresponding to different atomgroups and atomic orbitals (s, p, d, f)
  - User might be interested in any superposition of them (e.g. to locate states in real space)
  - Information stored in form of weights for all atom groups and the atomic orbitals s, p, d, f

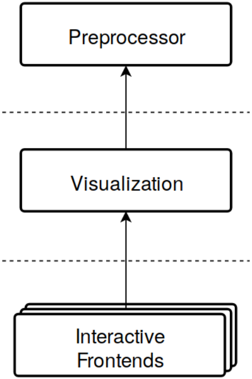
Module Design Goals



Multifunctionality:

- automated workflows like in AiiDA
- manual data analysis with Python

Module Design Goals

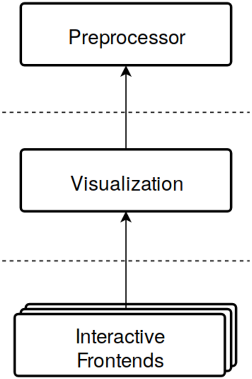


Multifunctionality:

- automated workflows like in AiiDA
- manual data analysis with Python

- no boilerplate code!

Module Design Goals



Multifunctionality:

- automated workflows like in AiiDA
- manual data analysis with Python

- no boilerplate code!

- Desktop 🖥
- Web → 🌍 like in AiiDAlab

Preprocessor Module

Input: Fleur calculation results stored in Hierarchical Data Format (HDF).

- ➔ attribute dependency resolution
- ➔ modular output types

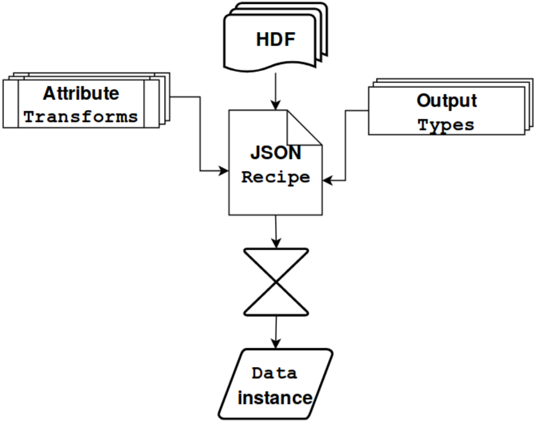
Preprocessor Module

Input: Fleur calculation results stored in Hierarchical Data Format (HDF). Combining *Python ABC*[1] *and type introspection* enables concise **Recipes** for different applications:

- ➔ attribute dependency resolution
- ➔ modular output types

[1] Abstract Base Class

Preprocessor Module

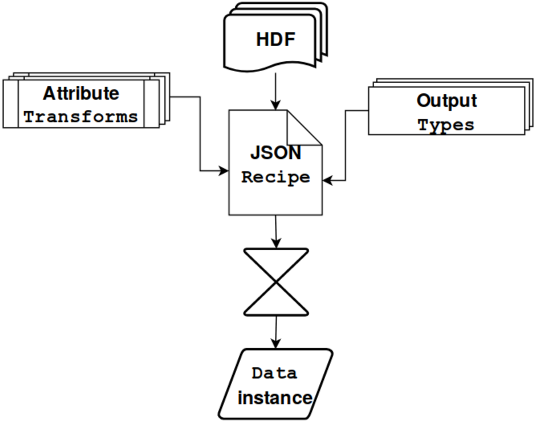


Input: Fleur calculation results stored in Hierarchical Data Format (HDF). Combining *Python ABC*[1] *and type introspection* enables concise **Recipes** for different applications:

- ➔ attribute dependency resolution
- ➔ modular output types

[1] Abstract Base Class

Preprocessor Module

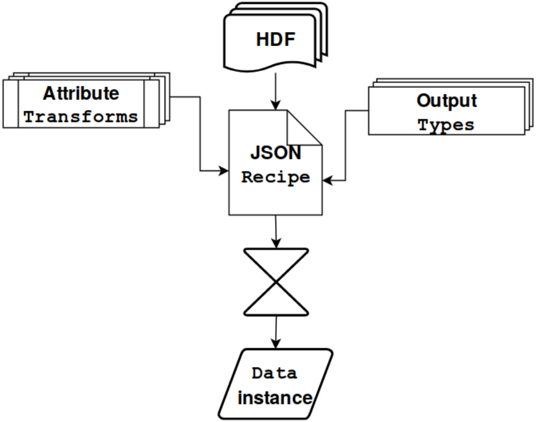


Input: Fleur calculation results stored in Hierarchical Data Format (HDF). Combining *Python ABC*[1] *and type introspection* enables concise **Recipes** for different applications:

- ➔ attribute dependency resolution
- ➔ modular output types

[1] Abstract Base Class

Preprocessor Module



Input: Fleur calculation results stored in Hierarchical Data Format (HDF). Combining *Python ABC*[1] *and type introspection* enables concise **Recipes** for different applications:

- ➔ attribute dependency resolution
- ➔ modular output types

[1] Abstract Base Class

Data Selection for Visualization

Application Band Structure Viz.: from 4D discrete weighted points to 2D bands.

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

- $W^{\text{eff}}_{s,k,\nu}$: effective weight
- s spin, k point on k-path, ν band, g group, l character
- $n_{s,k,\nu,g,l}$: State-specific l-like charge
- N_g: no. of atoms in group
- $W^{\text{unf}}_{s,k,\nu}$: unfolding weight; $\alpha = 0 \implies$ no unfolding

Data Selection for Visualization

Application Band Structure Viz.: from 4D discrete weighted points to 2D bands.

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

- $W^{\text{eff}}_{s,k,\nu}$: effective weight
- s spin, k point on k-path, ν band, g group, l character
- $n_{s,k,\nu,g,l}$: State-specific l-like charge
- N_g: no. of atoms in group
- $W^{\text{unf}}_{s,k,\nu}$: unfolding weight; $\alpha = 0 \implies$ no unfolding

Data Selection for Visualization

Application Band Structure Viz.: from 4D discrete weighted points to 2D bands.

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

- $W^{\text{eff}}_{s,k,\nu}$: effective weight
- s spin, k point on k-path, ν band, g group, l character
- $n_{s,k,\nu,g,l}$: State-specific l-like charge
- N_g: no. of atoms in group
- $W^{\text{unf}}_{s,k,\nu}$: unfolding weight; $\alpha = 0 \implies$ no unfolding

Data Selection for Visualization

Application Band Structure Viz.: from 4D discrete weighted points to 2D bands.

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

- $W^{\text{eff}}_{s,k,\nu}$: effective weight
- s spin, k point on k-path, ν band, g group, l character
- $n_{s,k,\nu,g,l}$: State-specific l-like charge
- N_g: no. of atoms in group
- $W^{\text{unf}}_{s,k,\nu}$: unfolding weight; $\alpha = 0 \implies$ no unfolding

Data Selection for Visualization

Application Band Structure Viz.: from 4D discrete weighted points to 2D bands.

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

- $W^{\text{eff}}_{s,k,\nu}$: effective weight
- s spin, k point on k-path, ν band, g group, l character
- $n_{s,k,\nu,g,l}$: State-specific l-like charge
- N_g: no. of atoms in group
- $W^{\text{unf}}_{s,k,\nu}$: unfolding weight; $\alpha = 0 \implies$ no unfolding

Data Selection for Visualization

Application Band Structure Viz.: from 4D discrete weighted points to 2D bands.

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

- $W^{\text{eff}}_{s,k,\nu}$: effective weight
- s spin, k point on k-path, ν band, g group, l character
- $n_{s,k,\nu,g,l}$: State-specific l-like charge
- N_g: no. of atoms in group
- $W^{\text{unf}}_{s,k,\nu}$: unfolding weight; $\alpha = 0 \implies$ no unfolding

Data Selection for Viz

Typically, $\sim 10^7$ data points are accessed.

Optimizations:

- reshaping $(k, \nu) \rightarrow (k \cdot \nu)$
- weight filter t: $W^{\text{eff}}_{s,k,\nu} > t$
- using optimized `numpy` functions for tensor product
- buffering on selection change

➔ Speedup $\sim 10^2$

Data Selection for Viz

Typically, $\sim 10^7$ data points are accessed.

Optimizations:

- reshaping $(k, \nu) \rightarrow (k \cdot \nu)$
- weight filter t: $W^{\text{eff}}_{s,k,\nu} > t$
- using optimized `numpy` functions for tensor product
- buffering on selection change

➔ Speedup $\sim 10^2$

Data Selection for Viz

Typically, $\sim 10^7$ data points are accessed.

Optimizations:

- reshaping $(k, \nu) \rightarrow (k \cdot \nu)$
- weight filter t: $W^{\text{eff}}_{s,k,\nu} > t$
- using optimized `numpy` functions for tensor product
- buffering on selection change

➔ Speedup $\sim 10^2$

Data Selection for Viz

Typically, $\sim 10^7$ data points are accessed.

Optimizations:

- reshaping $(k, \nu) \rightarrow (k \cdot \nu)$
- weight filter t: $W^{\text{eff}}_{s,k,\nu} > t$
- using optimized `numpy` functions for tensor product
- buffering on selection change

➔ Speedup $\sim 10^2$

Data Selection for Viz

Typically, $\sim 10^7$ data points are accessed.

Optimizations:

- reshaping $(k, \nu) \rightarrow (k \cdot \nu)$
- weight filter t: $W^{\text{eff}}_{s,k,\nu} > t$
- using optimized `numpy` functions for tensor product
- buffering on selection change

➔ Speedup $\sim 10^2$

Data Selection for Viz

Typically, $\sim 10^7$ data points are accessed.

Optimizations:

- reshaping $(k, \nu) \rightarrow (k \cdot \nu)$
- weight filter t: $W^{\text{eff}}_{s,k,\nu} > t$
- using optimized `numpy` functions for tensor product
- buffering on selection change

➔ Speedup $\sim 10^2$

Data Selection for Viz

Typically, $\sim 10^7$ data points are accessed.

Optimizations:

- reshaping $(k, \nu) \rightarrow (k \cdot \nu)$
- weight filter t: $W^{\text{eff}}_{s,k,\nu} > t$
- using optimized `numpy` functions for tensor product
- buffering on selection change

➔ Speedup $\sim 10^2$

Visualization Module

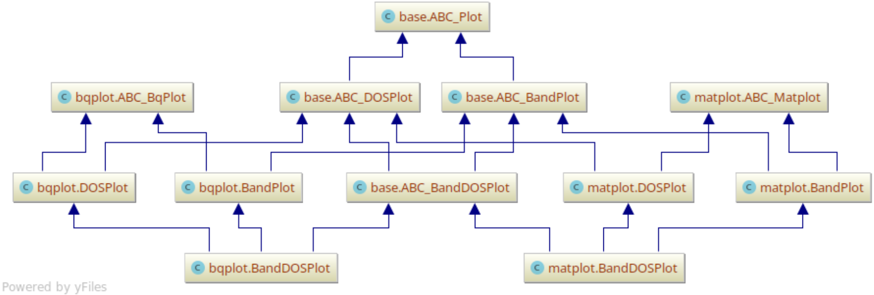
- Combine *Python ABC* and *multiple inheritance*
- ➔ maximizes code reuse for different applications and viz. libraries employed

Visualization Module

- Combine *Python ABC* and *multiple inheritance*
- ➔ maximizes code reuse for different applications and viz. libraries employed

Visualization Module

- Combine *Python ABC* and *multiple inheritance*
- ➔ maximizes code reuse for different applications and viz. libraries employed



Desktop Frontend

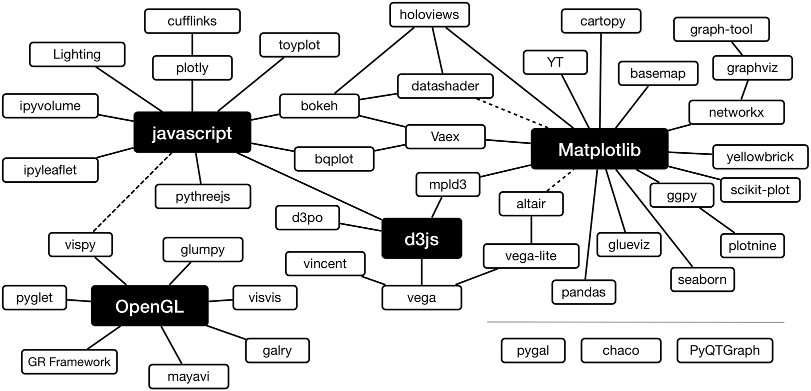
Choice of GUI Toolkit: **TKinter** over (Kivy, PySide/PyQt, ...)
Choice of Plotting tool: **matplotlib**

Web Frontend

The Python Visualization Landscape as of 2017...

Web Frontend

The Python Visualization Landscape as of 2017...



Python Visualization Landscape by rougier / BSD-2

Web Frontend

- Needed: a **survey** of OSS Frameworks for building a Web Dashboard using **only** 🐍.
- Selection Process: *Framework supports...*
  - I. ... *interactive graphical control elements ('widgets') to control plots*
  - II. ... *easy deployment*
  - III. ... *some actual plotting libraries*

Web Frontend

- Needed: a **survey** of OSS Frameworks for building a Web Dashboard using **only** 🐍.
- Selection Process: *Framework supports...*
  - I. ... *interactive graphical control elements ('widgets') to control plots*
  - II. ... *easy deployment*
  - III. ... *some actual plotting libraries*

Web Frontend

- Needed: a **survey** of OSS Frameworks for building a Web Dashboard using **only** 🐍.
- Selection Process: *Framework supports...*
  - I. ... *interactive graphical control elements ('widgets') to control plots*
  - II. ... *easy deployment*
  - III. ... *some actual plotting libraries*

Web Frontend

- Needed: a **survey** of OSS Frameworks for building a Web Dashboard using **only** 🐍.
- Selection Process: *Framework supports...*
  - I. ... *interactive graphical control elements ('widgets') to control plots*
  - II. ... *easy deployment*
  - III. ... *some actual plotting libraries*

Web Frontend

- Needed: a **survey** of OSS Frameworks for building a Web Dashboard using **only** 🐍.
- Selection Process: *Framework supports...*
  - I. ... *interactive graphical control elements ('widgets') to control plots*
  - II. ... *easy deployment*
  - III. ... *some actual plotting libraries*

Web Frontend

I. **Widgets**	jupyter	pyviz panel	bokeh	dash
Languages	🐍	🐍	🐍 / JS	🐍 / JS

[1] Excluded: writing from scratch using Flask

[2] workaround. See also: appmode, voila, thebelab

[3] interactive only

Web Frontend

I. **Widgets**	jupyter	pyviz Panel panel	Bokeh bokeh	dash
Languages	🐍	🐍	🐍 / JS	🐍 / JS
II. **Deployment**				
- Jupyter	✓	✓	✓	✓
- Standalone[1]	(**binder**, 🐳)[2]	Bokeh server	Bokeh server	plotly plotly

[1] Excluded: writing from scratch using Flask

[2] workaround. See also: appmode, voila, thebelab

[3] interactive only

Web Frontend

I. **Widgets**	jupyter jupyter	pyviz Panel panel	Bokeh bokeh	dash
Languages	🐍	🐍	🐍 / JS	🐍 / JS
II. **Deployment**				
- Jupyter	✓	✓	✓	✓
- Standalone[1]	(binder, 🐳)[2]	Bokeh server	Bokeh server	plotly plotly
III. **Plots**[3]				
- 2D	mpl, bqplot, **all** ←	hvPlot hvplot, → **most** ←	Bokeh	plotly
- 3D	ipyvolume, **all** ←	✗	Bokeh	plotly

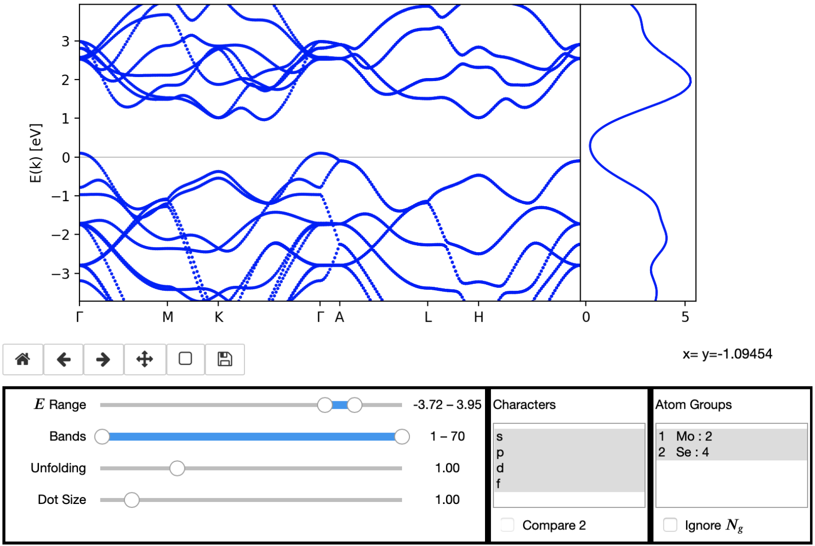
[1] Excluded: writing from scratch using Flask

[2] workaround. See also: appmode, voila, thebelab

[3] interactive only

Live Demonstration

Web GUI Demonstration



x= y=-1.09454

E Range -3.72 – 3.95

Bands 1 – 70

Unfolding 1.00

Dot Size 1.00

Characters

s
p
d
f

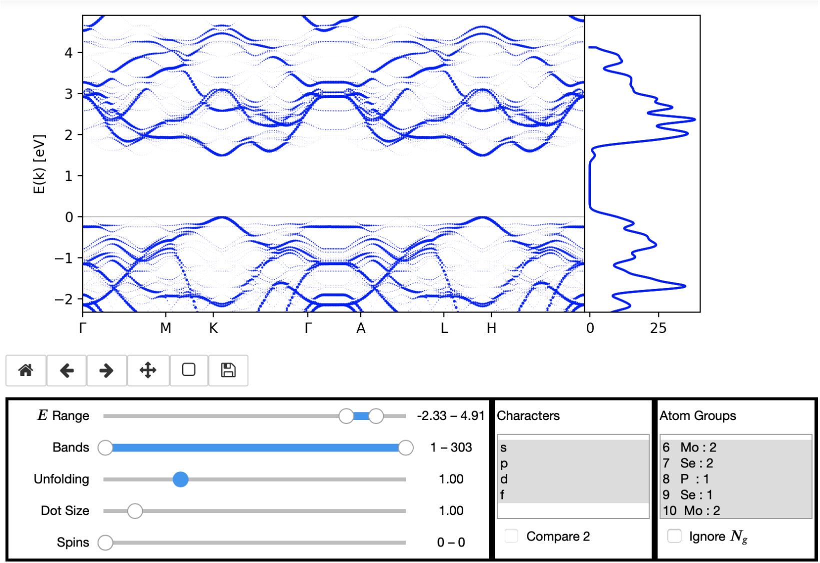
□ Compare 2

Atom Groups

1 Mo : 2
2 Se : 4

□ Ignore N_g

Web GUI Demonstration



E Range -2.33 – 4.91

Bands 1 – 303

Unfolding 1.00

Dot Size 1.00

Spins 0 – 0

Characters

s
p
d
f

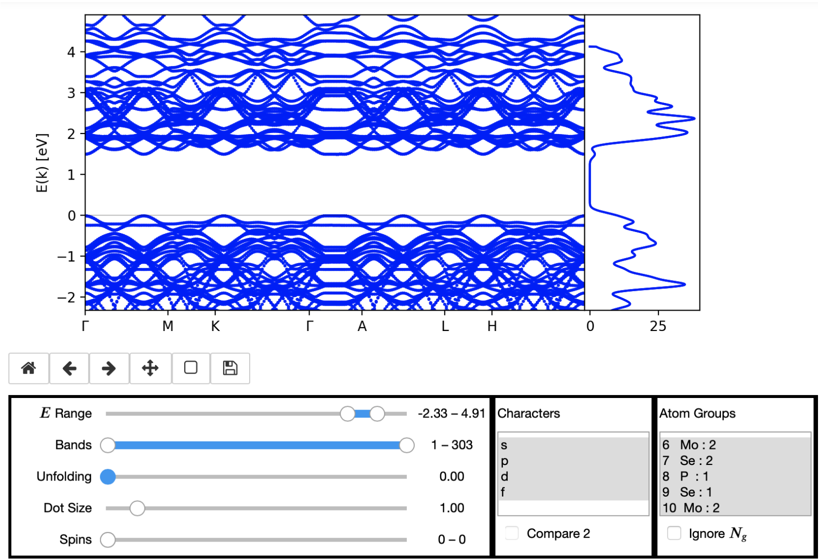
□ Compare 2

Atom Groups

6 Mo : 2
7 Se : 2
8 P : 1
9 Se : 1
10 Mo : 2

□ Ignore N_g

Web GUI Demonstration



E Range -2.33 – 4.91

Bands 1 – 303

Unfolding 0.00

Dot Size 1.00

Spins 0 – 0

Characters

s
p
d
f

□ Compare 2

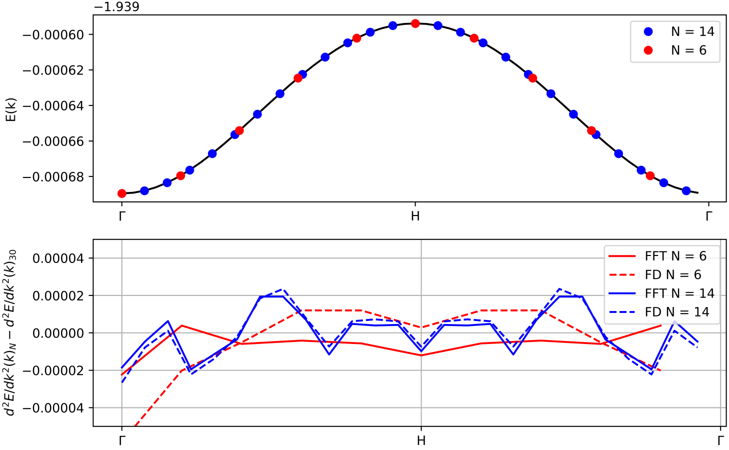
Atom Groups

6 Mo : 2
7 Se : 2
8 P : 1
9 Se : 1
10 Mo : 2

□ Ignore N_g

Effective mass and group velocity:

- Derived Quantities:
- Effective mass, that an electon in a crystal appears to have compared to a free electron (due to interactions in the solid)
  $m^* = \hbar^2 \left(\frac{\partial^2 E(k)}{\partial k^2} \right)^{-1}$
- Group velocity: $v_G(\vec{k}) = \frac{1}{\hbar} \frac{\partial E(\vec{k})}{\partial \vec{k}}$
- Problem: sparse k-Point mesh, but periodic bandstructure
- Idea: Using FFT to compute accurate derivates:
  $\Leftrightarrow$ Differentiate a finite Fourier series
  $f^{(n)}(x) = \mathcal{F}^{-1}\left((ik)^n \mathcal{F}(f(x))\right)$


−1.939
−0.00060
−0.00062
−0.00064
−0.00066
−0.00068
E(k)
● N = 14
● N = 6
Γ
H
Γ
0.00004
0.00002
0.00000
−0.00002
−0.00004
$d^2E/dk^2(k)_N - d^2E/dk^2(k)_{30}$
FFT N = 6
FD N = 6
FFT N = 14
FD N = 14
Γ
H
Γ

Conclusion

- Developed a package to visualize DFT data in a physically correct way
- Easy to use API and graphical interface
- studied data to extract pyhsical features (m^*, v_{fermi}) using numerical differentiation techniques
- Useful in physics research:
  - Make Fleur output easily accessible to non-experts
  - facilitate output ➔ input conversion to other simulation codes while HDF format develops
  - create small apps to calculate certain physical properties

Conclusion

- Developed a package to visualize DFT data in a physically correct way
- Easy to use API and graphical interface
- studied data to extract pyhsical features (m^*, v_{fermi}) using numerical differentiation techniques
- Useful in physics research:
  - Make Fleur output easily accessible to non-experts
  - facilitate output ➔ input conversion to other simulation codes while HDF format develops
  - create small apps to calculate certain physical properties

Conclusion

- Developed a package to visualize DFT data in a physically correct way
- Easy to use API and graphical interface
- studied data to extract pyhsical features (m^*, v_{fermi}) using numerical differentiation techniques
- Useful in physics research:
  - Make Fleur output easily accessible to non-experts
  - facilitate output ➔ input conversion to other simulation codes while HDF format develops
  - create small apps to calculate certain physical properties

Conclusion

- Developed a package to visualize DFT data in a physically correct way
- Easy to use API and graphical interface
- studied data to extract pyhsical features (m^*, v_{fermi}) using numerical differentiation techniques
- Useful in physics research:
  - Make Fleur output easily accessible to non-experts
  - facilitate output ➔ input conversion to other simulation codes while HDF format develops
  - create small apps to calculate certain physical properties

Conclusion

- Developed a package to visualize DFT data in a physically correct way
- Easy to use API and graphical interface
- studied data to extract pyhsical features (m^*, v_{fermi}) using numerical differentiation techniques
- Useful in physics research:
  - Make Fleur output easily accessible to non-experts
  - facilitate output ➔ input conversion to other simulation codes while HDF format develops
  - create small apps to calculate certain physical properties

Conclusion

- Developed a package to visualize DFT data in a physically correct way
- Easy to use API and graphical interface
- studied data to extract pyhsical features (m^*, v_{fermi}) using numerical differentiation techniques
- Useful in physics research:
  - Make Fleur output easily accessible to non-experts
  - facilitate output ➔ input conversion to other simulation codes while HDF format develops

  - create small apps to calculate certain physical properties

Conclusion

- Developed a package to visualize DFT data in a physically correct way
- Easy to use API and graphical interface
- studied data to extract pyhsical features (m^*, v_{fermi}) using numerical differentiation techniques
- Useful in physics research:
    - Make Fleur output easily accessible to non-experts
    - facilitate output ➔ input conversion to other simulation codes while HDF format develops
    - possible to integrate into AiiDA workflow
    - create small apps to calculate certain physical properties

Conclusion

- Developed a package to visualize DFT data in a physically correct way
- Easy to use API and graphical interface
- studied data to extract pyhsical features (m^{*}, v_{fermi}) using numerical differentiation techniques
- Useful in physics research:
  - Make Fleur output easily accessible to non-experts
  - facilitate output ➔ input conversion to other simulation codes while HDF format develops
  - possible to integrate into AiiDA workflow
  - create small apps to calculate certain physical properties