SiScLab Project 8

Katta, Partmann, Wasmer

January 24, 2019

Problem Statement

- Solid state physics: electronic structure computation
  - $\rightarrow$ Fleur: electronic structure of crystals using DFT
  - 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 $\rightarrow$ data exploration $\rightarrow$ visualization

Motivation

- Physicists problem with the simulation data
- fast computation time
- code modularization
- intuitive usage
- high-quality export features

Requirements

- process Fleur output files
- fast computation
- code: modularization, easy maintainability
- frontend: no installation required, intuitive usage
- plots: publication-quality export

Steps

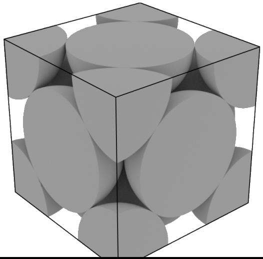
- Understanding physics and problem
- preprocessing the data
- exploring the data(implementation)
- visualization(GUI)
- Results

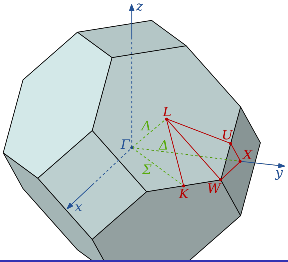
How is the data generated?

- Fluer computes electron density in crystals
- Density functional theory (DFT) approach:
  - Hohenberg Kohn theorem: use electron density
  - Kohn Sham system: Solve one particle Schrdinger equations in effective potential (self consistent)
  - State of the art method for electronic structure computations in solids

What data is generated?

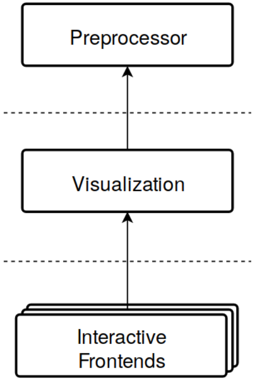
- Bandstructure $E_\nu(k)$:
  - Eigenenergies of (Bloch-) Eigenfunctions of the Hamiltonian for each (crystal-) momentum k
  - Disperion relation: Relation between crystal momentum and Energies of the Bloch electrons
  - Sampled along a 1d Path between high symmetry points in 3d reciprocal space





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

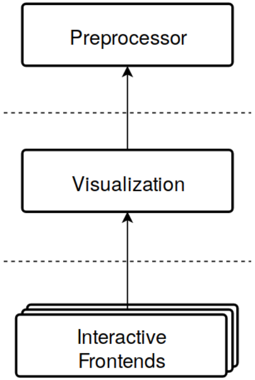
Module Design Goals



Multifunctionality:

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

Module Design Goals

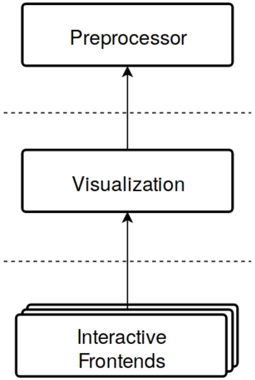


Multifunctionality:

- 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).

- modular output types for application domain (e.g. viz)
- dependency resolution

Preprocessor Module

Input: Fleur calculation results stored in Hierarchical Data Format (HDF).
Module uses *type introspection* to enable features:

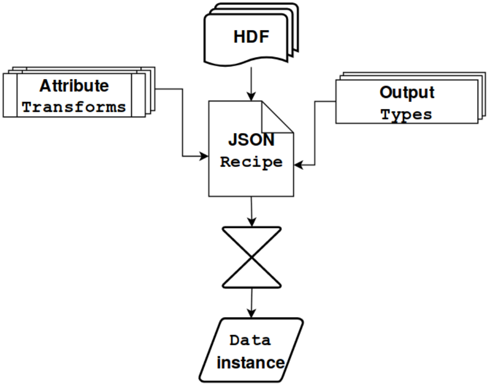
- modular output types for application domain (e.g. viz)
- dependency resolution

Preprocessor Module

Input: Fleur calculation results stored in Hierarchical Data Format (HDF).
Module uses *type introspection* to enable features:

- modular output types for application domain (e.g. viz)
- dependency resolution

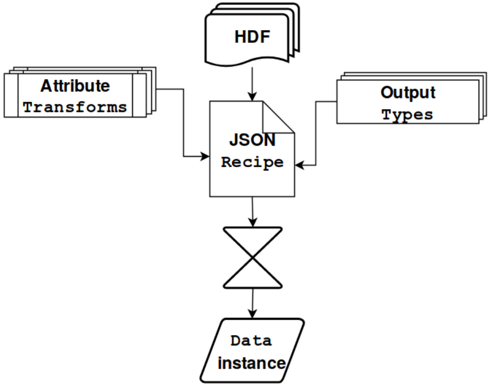
Preprocessor Module



Input: Fleur calculation results stored in Hierarchical Data Format (HDF).
Module uses *type introspection* to enable features:

- modular output types for application domain (e.g. viz)
- dependency resolution

Preprocessor Module



Input: Fleur calculation results stored in Hierarchical Data Format (HDF).
Module uses *type introspection* to enable features:

- modular output types for application domain (e.g. viz)
- dependency resolution

Data Selection for Viz

The main compute-intensive routine:

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

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

Data Selection for Viz

The main compute-intensive routine:

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

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

Data Selection for Viz

The main compute-intensive routine:

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

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

Data Selection for Viz

The main compute-intensive routine:

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

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

Data Selection for Viz

The main compute-intensive routine:

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

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

Data Selection for Viz

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

Optimizations:

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

➔ Speedup $\sim 10^2$

Visualization Module

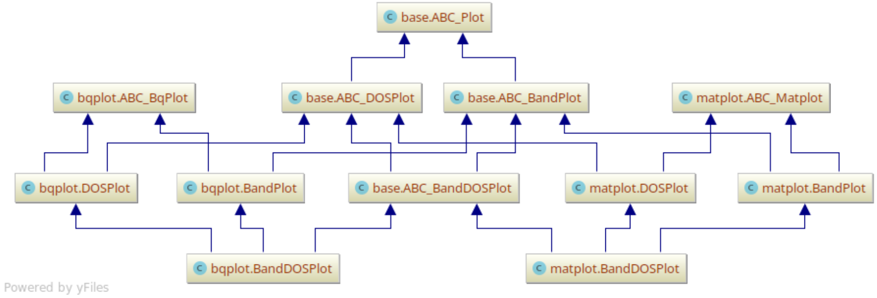
- Abstract interfaces for different viz. libs and applications
- `InteractiveControlDisplay` as frontend contracts

Visualization Module

- Abstract interfaces for different viz. libs and applications
- `InteractiveControlDisplay` as frontend contracts

Visualization Module

- Abstract interfaces for different viz. libs and applications
- `InteractiveControlDisplay` as frontend contracts



Desktop Frontend

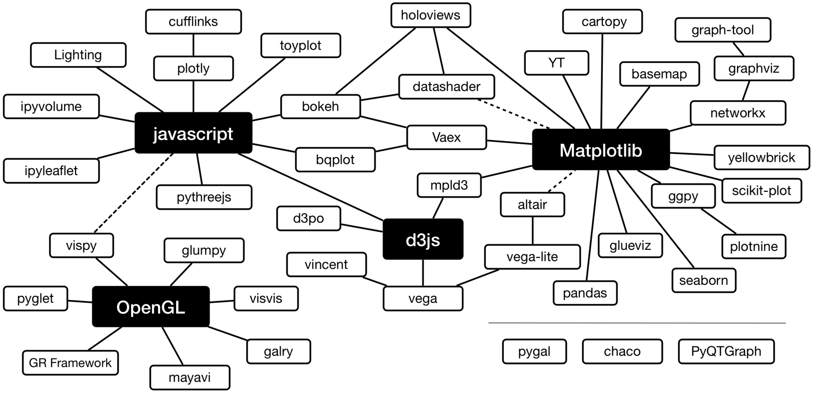
Choice of GUI Toolkit: **TKinter**, Kivy, PySide/PyQt, ...
Choice of Plotting tool: **matplotlib**

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: an OSS **Tool Selection Process** for building a Web Dashboard using **only** .
- Decision Priority Order: *support...*
  - I. ... *interactive graphical control elements ('widgets')*
  - II. ... *easy deployment*
  - III. ... *some actual plotting libraries*

Web Frontend

- Needed: an OSS **Tool Selection Process** for building a Web Dashboard using **only** .
- Decision Priority Order: *support...*
  - I. ... *interactive graphical control elements ('widgets')*
  - II. ... *easy deployment*
  - III. ... *some actual plotting libraries*

Web Frontend

- Needed: an OSS **Tool Selection Process** for building a Web Dashboard using **only** .
- Decision Priority Order: *support...*
  - I. ... *interactive graphical control elements ('widgets')*
  - II. ... *easy deployment*
  - III. ... *some actual plotting libraries*

Web Frontend

- Needed: an OSS **Tool Selection Process** for building a Web Dashboard using **only** .
- Decision Priority Order: *support...*
  - I. ... *interactive graphical control elements ('widgets')*
  - II. ... *easy deployment*
  - III. ... *some actual plotting libraries*

Web Frontend

- Needed: an OSS **Tool Selection Process** for building a Web Dashboard using **only** .
- Decision Priority Order: *support...*
  - I. ... *interactive graphical control elements ('widgets')*
  - 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 jupyter	pyviz Panel panel	Bokeh bokeh	dash
Languages	🐍	🐍	🐍 / JS	🐍 / JS
II. **Deployment**				
- Jupyter	✓	✓	✗	✗
- Standalone[1]	(binder, 🐳)[2]	Bokeh	Bokeh	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	Bokeh	plotly plotly
III. **Plots**[3]				
- 2D	mpl, bqplot, ...	hvPlot hvplot, Bokeh	Bokeh	plotly
- 3D	ipyvolume	✖	Bokeh	plotly

[1]Excluded: writing from scratch using Flask
[2]workaround. See also: appmode, voila, thebelab
[3]interactive only

Effective mass and Fermi velocity:

- Derived Quantities:
- mass, that an electon in a crystal appears to have compared to a free electron (due to interactions in the solid)
- $m^* = \hbar^2 \frac{d^2E(k)}{dk^2}$
- Group velocity at the Fermi energy
- $v_{Fermi} = \frac{dE(k)}{dk}$ at $E = E_F$
- Bandstructure periodic: Using FFT to compute accurate Derivates:
- $\Leftrightarrow$ Differentiate finite Fourier Series $f^{(n)}(x) = \mathcal{F}^{-1}\left((ik)^n \mathcal{F}(f(x))\right)$