WANNIER BERRI : an advanced tool for interpolation of Berry curvature and related quantities. (aka Wannier19)

Stepan S. Tsirkin

Department of Physics, University of Zürich, Switzerland

Universität Zürich UZH

Anomalous Hall and Nernst effects

Tee AHC tensor is given by:

$$\sigma^{\rm AHE}_{\alpha\beta}(0) = -\frac{e^2}{\hbar}\int\frac{d\mathbf{k}}{(2\pi)^3}\sum_n f_{n\mathbf{k}}\Omega_{n,\alpha\beta}(\mathbf{k}) \qquad (1)$$

in terms of Berry curvature:

$$\Omega_{n,\alpha\beta}(\mathbf{k}) = -2{\rm Im}\langle\nabla_{k_\alpha}u_{n\mathbf{k}}|\nabla_{k_\beta}u_{n\mathbf{k}}\rangle. \qquad (2)$$

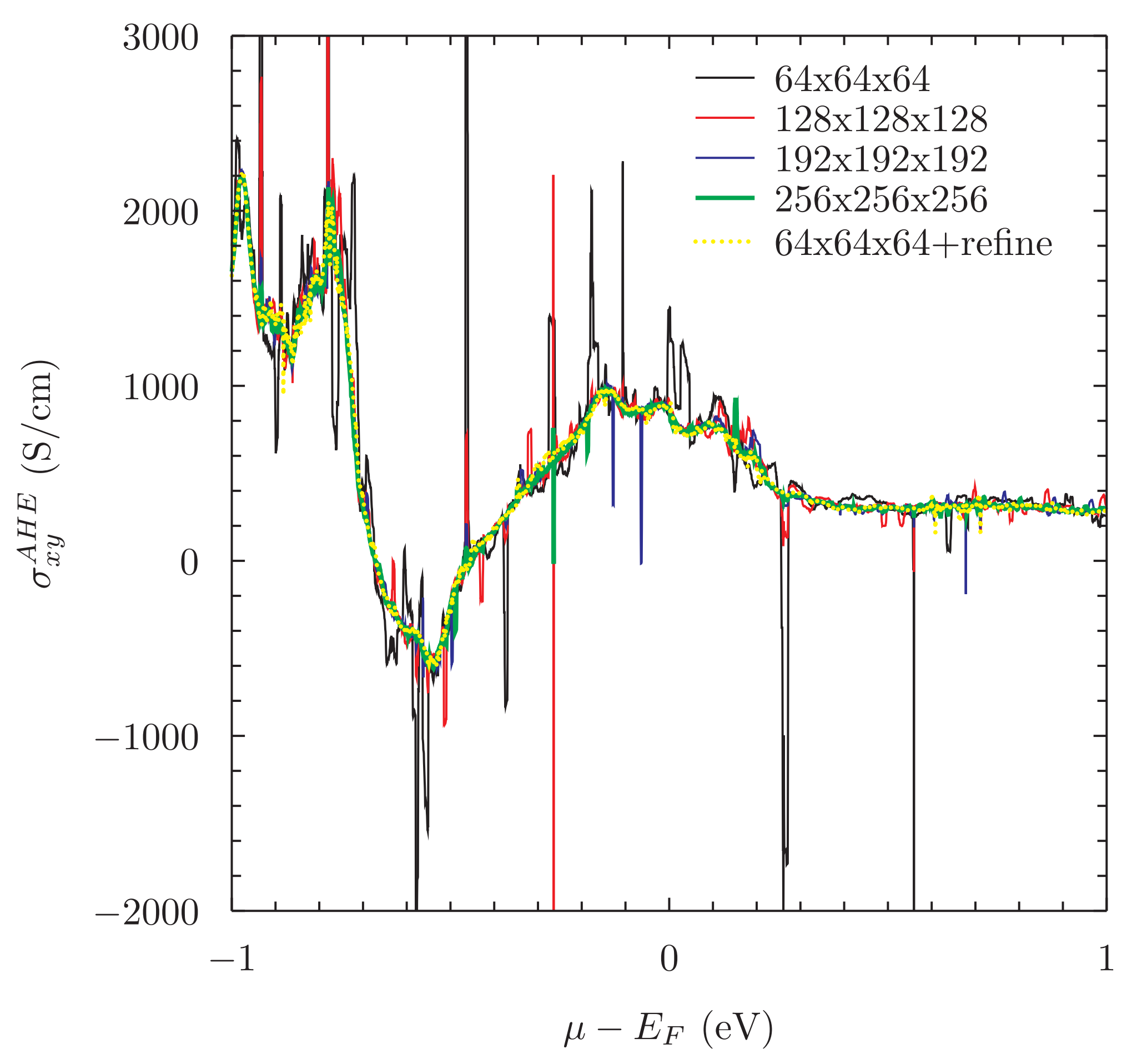
where $|u_{n\mathbf{k}}\rangle = e^{-i\mathbf{k}\cdot\mathbf{r}}|\psi_{n\mathbf{k}}\rangle$ is the cell-periodic Bloch function. to evaluate it one needs:

- a smooth gauge
- dense grid of **k**-points

That is where Wannier interpolation helps a lot.
Anomalous Nernst Effect is even harder

$$\alpha^{\rm ANE}_{xy} = -\frac{1}{e}\int d\varepsilon\frac{\partial f}{\partial\varepsilon}\sigma^{\rm AHE}_{xy}(\varepsilon)\frac{\varepsilon-\mu}{T} \qquad (3)$$

because at low T it is essentially the derivative of σ^{AHE}



Irudia 1: Example: AHC of bcc Fe as a function of chemical potential μ, for different interpolation grids, with and without recursive refinement

Overview

W-BERRI is an improved python implementation of postw90.x part of **Wannier90**. ("19-- because it was started in 2019.)

W-BERRI accepts the Wannier functions generated by Wannier90, or any tight-binding model. Than a BZ integral of any combination of velocities, Berry curvatures and other k-space quantities can be evaluated.

Wannier90 is **good**, because it

- is a well-established code
- known for ages
- has a broad community of developers
- has many implemented features
- ..... (many more)

W-BERRI is **better**, because it

- is faster
- Employs a mixture of fast and slow Fourier transforms
- uses recursive adaptive refinement for accurate evaluation around special points in BZ
- accounts for the symmetries, to integrate only the irreducible part of BZ and make the result symmetric.
- is written in object-oriented Python, which allows easier further development.
- has some more efficient algorithms, in particular
  - for multiple Fermi levels
  - minimal-distance replica selection method

Hosted on github

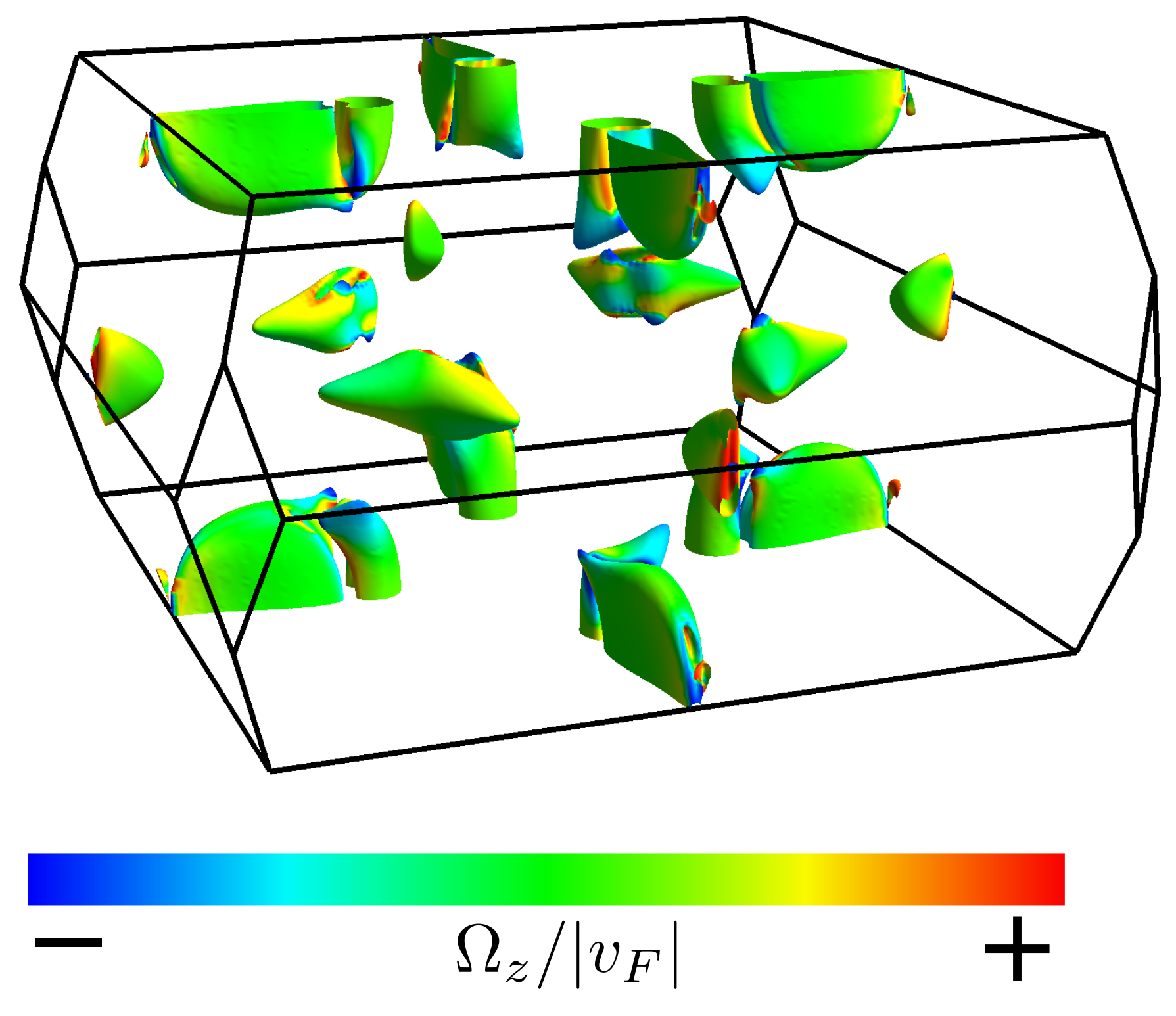
https://github.com/stepan-tsirkin/wannier-berri

Install with pip

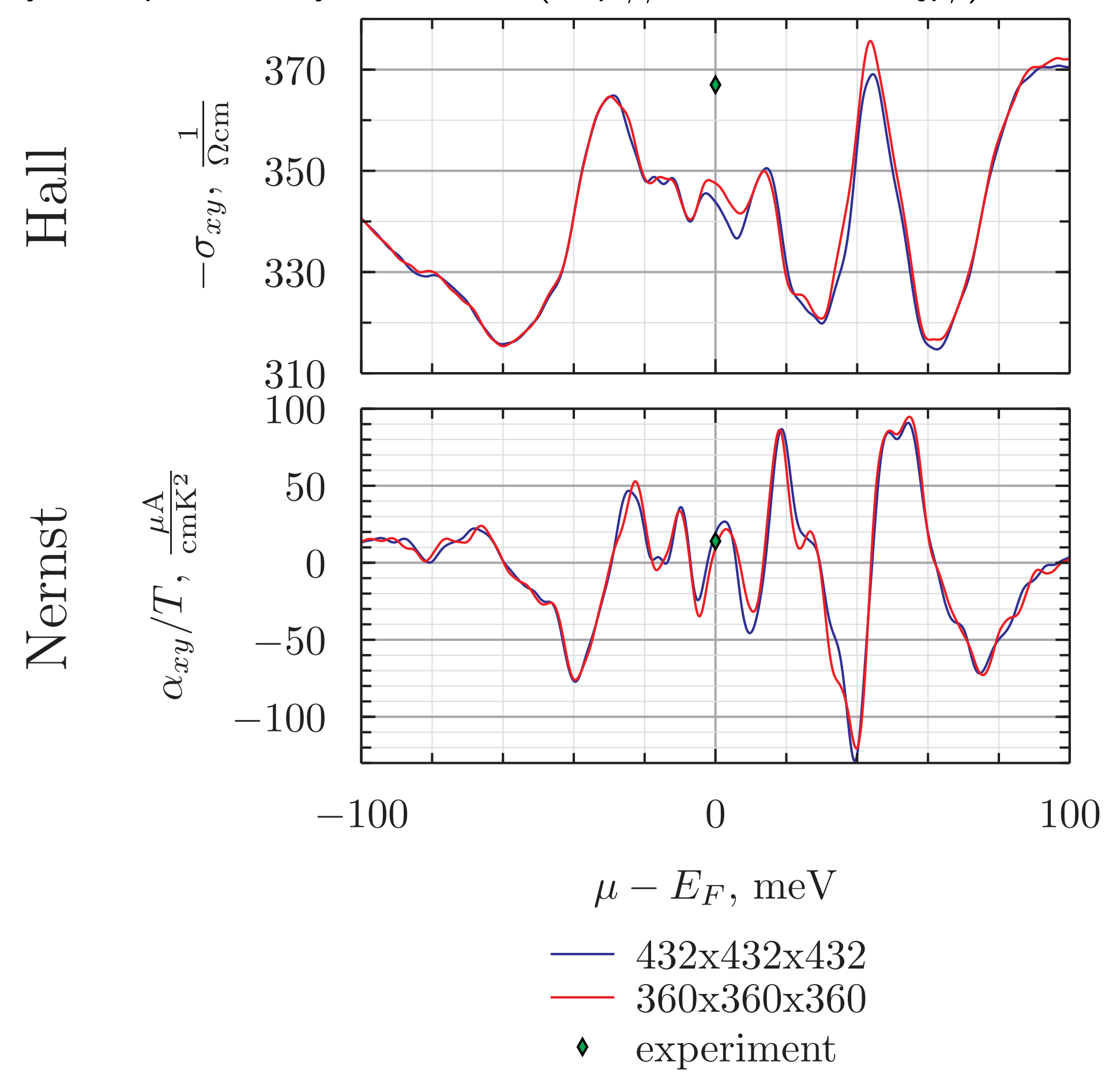
```
pip3 install wannierberri
```

First real-life use: AHE and ANE in magnetic Weyl Semimetal PrAlGe

"Magnetism and anomalous transport in the Weyl semimetal PrAlGe: possible route to axial gauge fields". D. Destraz, L. Das, S.S. Tsirkin, Y. Xu, *et al.* Accepted in **npj Quantum Materials**.



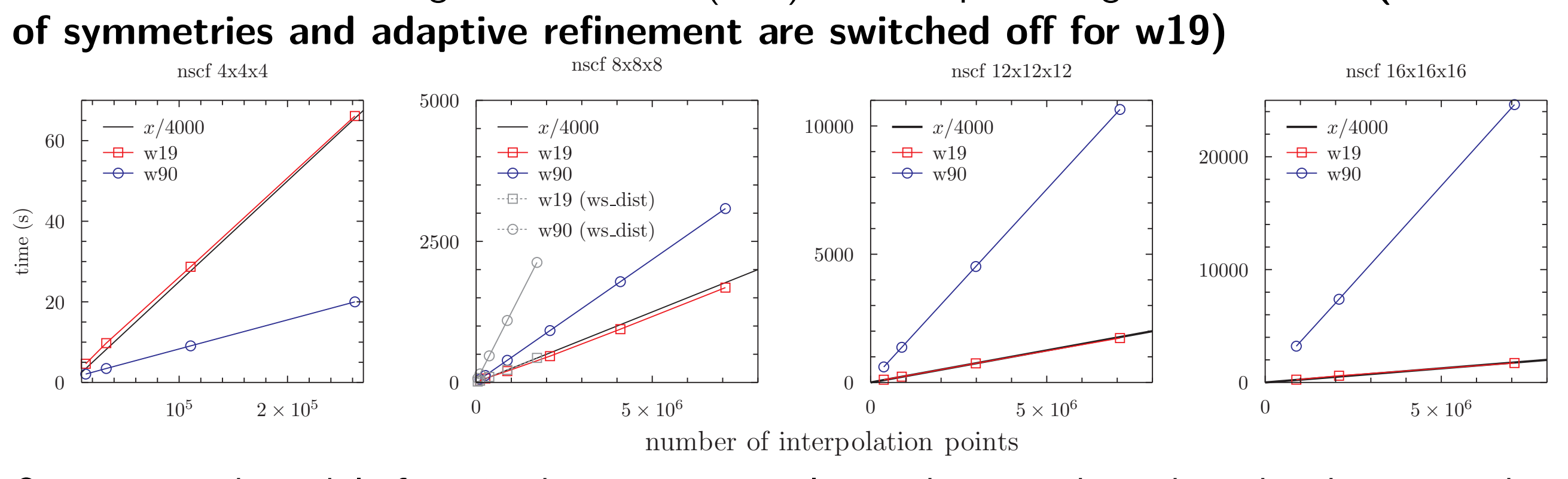
Irudia 2: Fermi surface of ReAlGe coloured by Berry curvature divided by Fermi velocity. Plot produced by FermiSurfer (http://fermisurfer.osdn.jp/)



Irudia 3: AHE and ANE calculated in PrAlGe in comparison with experiment

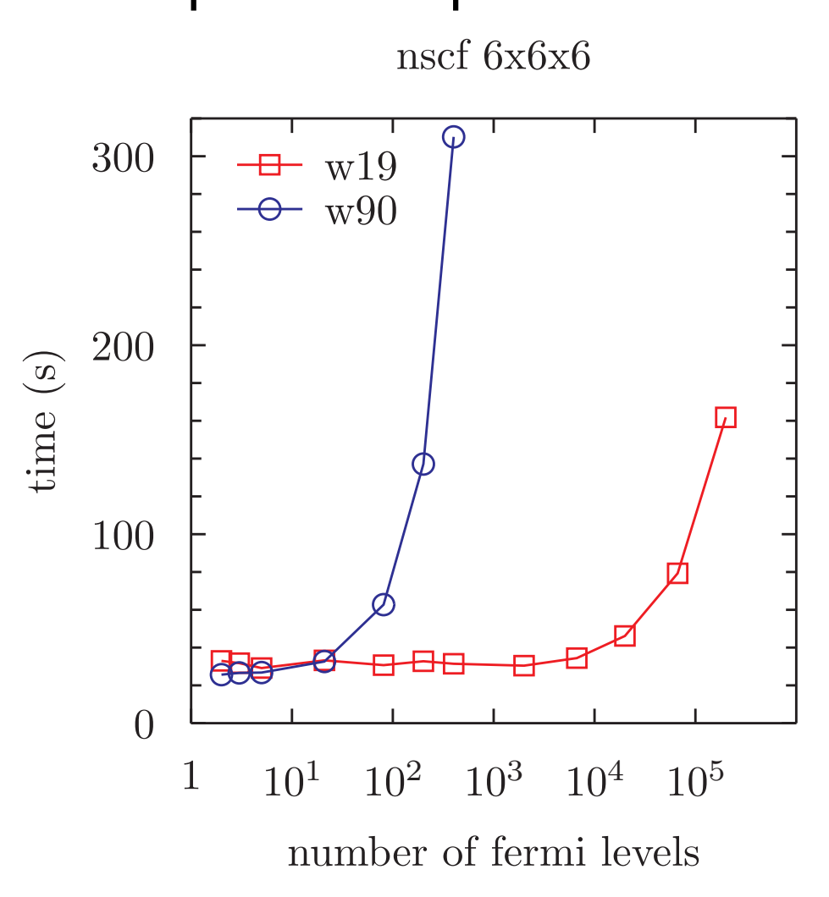
Computational time

Calculations were performed with WannierBerri and wannier90 on identical 40-core nodes of a cluster at UZH. Different grids of ab-initio (nscf) and interpolation grids were used. **(account of symmetries and adaptive refinement are switched off for w19)**



One can see that while for w19 the computational time does not depend on the ab-initio grid and the use of parameter "use_ws_distance". While for w90 they increase time a lot.

Also scanning multiple fermi levels has almost no cost in w19 up to 10^4 points:



Recursive adaptive refinement

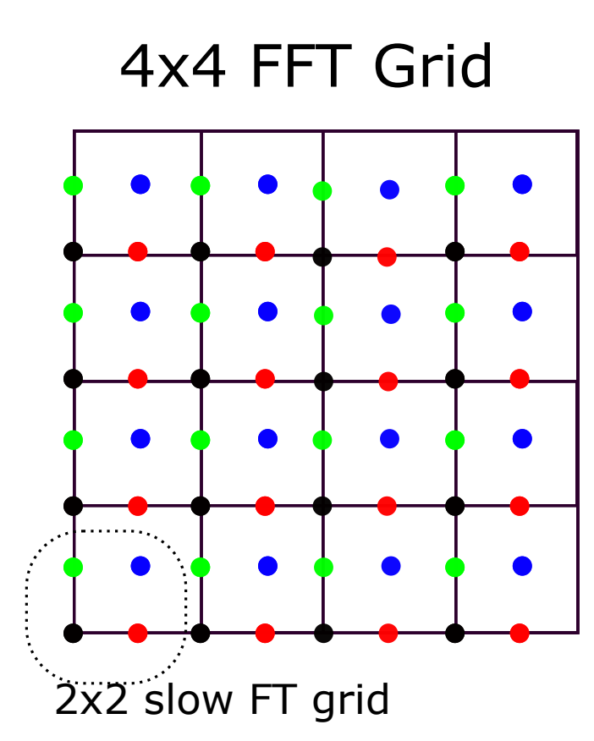
A few k-points are chosen that give the largest contribution to the integral, or to the jittering of the curve. — typically the points near (avoided) degeneracies. They are replaced by grids of 2x2x2 points surrounding it. The procedure is repeated recursively (new points can be refined again) to get a smoother curve.

Mixed Fourier Transform

n — size of *ab initio* grid
$N = m\times n$ — size of interpolation grid (m-integer)
we want to Fourier Transform a quantity A:
$A(R_1),\ldots,A(R_n)\rightarrow A(k_0),\ldots A(k_{N-1})$, where $k_i = 2\pi i/N$

For each $j = 0,\ldots,m-1$ we "slow FT"
$A(R_l)\rightarrow\tilde{A}^j(R_l) = A(R_l)\times e^{2\pi l\frac{j}{N}}$
and then do a FFT of $\tilde{A}^j(R_l)$, thus getting k-points $k_{i,j} = 2\pi(j+im)/N$

time scaling:
slow FT $\propto N\times n$
fast FT $\propto N\log N$
mixed FT $\propto N\log n$



Thus, FFT is better then SFT for large ab initio grids, while MFT is better in any case.

Advantages of mixed FT over FFT:

- easier parallelization
- allows to remove symmetry-equivalent "slow"k-point
- no need to store the whole interpolation grid (less memory needed)
- allows adaptive recursive refinement of most "important"points.

tutorial.py

```python
#!/usr/bin/env python3
import wannier19 as w19
import numpy as np

name="Fe"
NK=(96,96,96)  # a grid 96x96x96
Efermi=np.linspace(12.,13.,1001)  # range of Fermi levels
system=w19.System(tb_file='Fe_tb.dat',getAA=True)

# symmetries of the system to be accounted for :
SYM=w19.symmetry
generators=[SYM.Inversion,SYM.C4z,SYM.TimeReversal*SYM.C2x]

# tabulate quntities to be plotted by FermiSurfer
w19.tabulate(system,
             NK=NK,
             quantities=["V","berry"],
             symmetry_gen=generators,
             fout_name=name,
             numproc=4,
             Ef0=0,
             restart=False  )

# integrated quantities
w19.integrate(system,
            NK=NK,
            Efermi=Efermi,
            smearEf=10,
            quantities=["ahc","dos"],
            numproc=4,
            adpt_num_iter=10,
            fout_name=name,
            symmetry_gen=generators,
            restart=False  )
```

Acknowledgements

I thank Ivo Souza and Miguel Angel Jiménez Herrera for fruitfull discussion