berry suite of programs

Documentation

v.1.0

November 17, 2022

Contents

Getting Started

This guide gives a general introduction to the **berry** suite of programs, including instalation and running. An introduction to the basics of electronic structure calculations is given, in the context of this suite.

1.1 What berry does

The **berry** suite of programs extracts the Bloch wavefunctions from DFT calculations in an ordered way so they can be directly used to make calculations.

In practice it retrieves the wavefunctions and their gradients in reciprocal space totally ordered by unentangled bands, where continuity (analyticity) applies.

With the Bloch wavefunctions, berry can calculate:

- Berry connections

- Berry curvatures

- Optical conductivity

- Second harmonic generation (SHG) optical conductivity

1.2 People

The **berry** suite is coordinated by Ricardo Mendes Ribeiro (University of Minho and INL, Braga, Portugal), which is also the main developer.

Other contributors include

- Irving Leander Reascos Valencia (Braga, Portugal) for improving the performance and AI work;

- Fábio Carneiro (Braga, Portugal) for the parallelization of the python code and its performance;

- André Mendes Marinho (Braga, Portugal) for the development of new features

- Nuno Castro (LIP-Minho, Braga, Portugal) for coordinating AI work;

- Gonçalo Ventura (Porto, Portugal) for the equations for non-linear optical properties calculated from the Berry connections;

- Ícaro Jael Moura (Fortaleza, Brasil) for the exploratory work and testing;

- Afonso Duarte Ribeiro, author of the logo.

1.3 Contacts

The web site for the **berry** suite is

https://ricardoribeiro-2020.github.io/berry/

All files for instalation can be found there.

There is no mailling list yet for reporting bugs or other questions, but an email can be sent to ricardo.ribeiro@physics.org

1.4 Aknowledgement

Instalation

2.1   PIP install

To install the **berry** suite a pip install command is enough:

```
pip install berry_suite
```

2.2   Requirements

This version of **berry** only supports the DFT package Quantum Espresso, version v.6.6 or higher.

So a working Quantum Espresso instalation has to be in the system, and must be in the $PATH.

2.3   Running

To run, first one has to create a working directory where the results will be saved, and inside it create a directory called *dft*.

Inside *dft* should go the pseudopotential files for the atomic species of the dft calculation and the file *scf.in* with the details of the scf run. You can use another name for the file, but then you have to add a line in the input file of the script **preprocess** to change the default (see section Preprocessing).

This *scf.in* file has to be a Quantum Espresso scf run file; this is the only one implemented so far.

Create an input file (as described in chapter Workflow, section Preprocessing) in the working directory.

All the scripts should be run from the working directory. To run, a command of the form:

```
berry [-h] script parameter [options]
```

where the `script` is one of the following

```
preprocess
wfcgen
dot
cluster
basis
r2k band
geometry
condutivity
shg
```

and the parameter is dependent on which script is beeing run.

These scripts should be run in sequence, and the *berry* command assures that all the needed files are present for each run.

Chapter Workflow explains what each script does, what parameters are required and what options they have.

Theoretical background to *berry*

3.1 Electronic structure calculations basics

Crystal

A crystal is a set of atoms, forming a pattern, that is replicated all over space in one, two or three dimensions. It forms a crystal *lattice*.

Here we will deal only with 2D materials, so the pattern, which is called *unit cell*, is repeated in two dimensions, in a plane.

The unit cell is defined by three vectors (two in 2D) $\boldsymbol{a}_1$, $\boldsymbol{a}_2$ and $\boldsymbol{a}_3$, in position space (the real space). One can go from one unit cell to any other one by aplying the translation

$$\boldsymbol{R} = n_1\boldsymbol{a}_1 + n_2\boldsymbol{a}_2 + n_3\boldsymbol{a}_3 \tag{3.1}$$

where n_1, n_2 and n_3 are integers, and $\boldsymbol{R}$ is called the lattice vector.

A Fourier transform of this periodic pattern gives another periodic pattern in the so-called *reciprocal space* or momentum space. This new periodicity
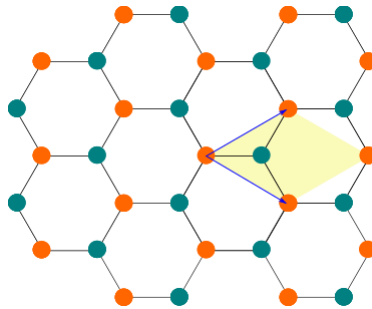

Figure 3.1: Example of a unit cell in a two dimensional material, with hexagonal symmetry. The yellow shadow represents the unit cell and the two blue vectors represent the lattice vectors.

is defined by the reciprocal vectors:

$$\boldsymbol{b}_1 = 2\pi\frac{\boldsymbol{a}_2 \times \boldsymbol{a}_3}{\boldsymbol{a}_1 \cdot (\boldsymbol{a}_2 \times \boldsymbol{a}_3)}; \qquad \boldsymbol{b}_2 = 2\pi\frac{\boldsymbol{a}_3 \times \boldsymbol{a}_1}{\boldsymbol{a}_1 \cdot (\boldsymbol{a}_2 \times \boldsymbol{a}_3)}; \qquad \boldsymbol{b}_3 = 2\pi\frac{\boldsymbol{a}_1 \times \boldsymbol{a}_2}{\boldsymbol{a}_1 \cdot (\boldsymbol{a}_2 \times \boldsymbol{a}_3)}$$

Electrons in a crystal

Electrons in a crystal can be described by the solutions to time independent Schrödinger equation:

$$H_{\boldsymbol{k}}\psi_{\boldsymbol{k},s}(\boldsymbol{r}) = E_{\boldsymbol{k},s}\psi_{\boldsymbol{k},s}(\boldsymbol{r}) \tag{3.2}$$

where $H_{\boldsymbol{k}}$ is the hamiltonean of the system, which has the periodicity of the crystal lattice, and is a function of the momentum $\boldsymbol{k}$. For each $\boldsymbol{k}$ we have an equation 3.2 that gives a (infinite) set of solutions, which are labeled by the letter s.

The eigenvalues are the energies $E_{\boldsymbol{k},s}$ and the respective eigenvectors are the wavefunctions $\psi_{\boldsymbol{k},s}(\boldsymbol{r})$.

For a periodic material, the solutions to equation 3.2 are given by Bloch's theorem:

$$\psi_{\boldsymbol{k}s}(\boldsymbol{r}) = e^{i\boldsymbol{k}\cdot\boldsymbol{r}}u_{\boldsymbol{k}s}(\boldsymbol{r}) \tag{3.3}$$

with

$$u_{\boldsymbol{k}s}(\boldsymbol{r}) = u_{\boldsymbol{k}s}(\boldsymbol{r} + \boldsymbol{R}) \tag{3.4}$$

and $\boldsymbol{k}$ is a point within the Brillouin Zone (BZ) and s numbers the bands. $\boldsymbol{R}$ is the lattice vector.

$u_{\boldsymbol{k}s}(\boldsymbol{r})$ is called the Bloch factor and $e^{i\boldsymbol{k}\cdot\boldsymbol{r}}$ is called the Bloch phase factor.

Bands

In the previous subsection the solutions of the Schrödinger equation were ordered by a *band* index s. In practice, the eigenvectors and eigenvalues are ordered in increasing energy, so that $E_{\boldsymbol{k},0}$ is the lowest energy solution of equation 3.2, $E_{\boldsymbol{k},1}$ is the second lowest energy solution, and so on.

Actually, there are a number of property calculations where it is necessary to use the set of eigenvectors and eigenvalues that form a band as an entity in which mathematical operators must be applied (the gradient in momentum space, for instance). The Berry connection is an example. The Berry connection is defined as:

$$\boldsymbol{\xi}_{\boldsymbol{k}ss'} = i\langle u_{\boldsymbol{k}s}|\nabla_{\boldsymbol{k}}u_{\boldsymbol{k}s'}\rangle = \frac{i}{v_C}\int_{uc} d^3\boldsymbol{r}\, u_{\boldsymbol{k}s}^*(\boldsymbol{r})\nabla_{\boldsymbol{k}}u_{\boldsymbol{k}s'}(\boldsymbol{r}) \tag{3.5}$$

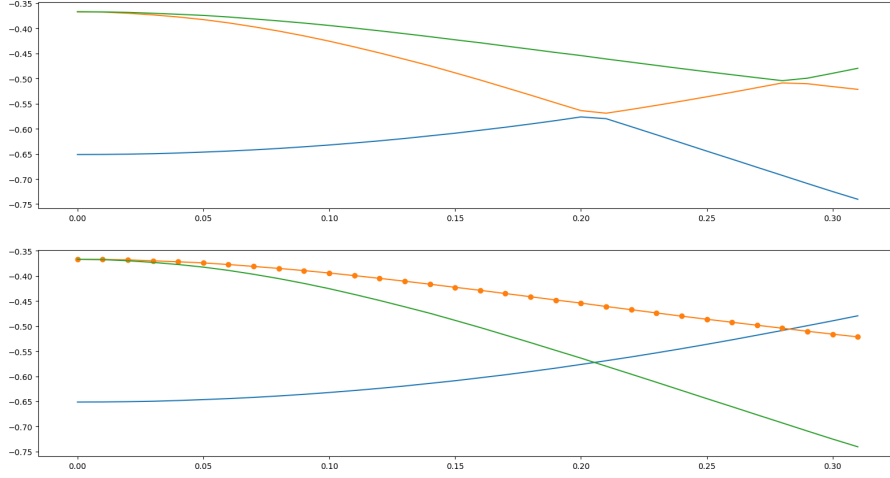where v_C is the volume of the unit cell and uc stands for unit cell.

Figure 3.2: Example of electronic bands in one dimension. Different colors indicate different bands. Top: the bands according to the DFT output, that is, ordered by energy. Bottom: the bands according to continuity of the Bloch factors. The dots on the orange curve in the bottom figure indicate the positions of the k point sampling.

But the ordering (and grouping) of eigenstates by energy fails to give an usable entity for a gradient calculation, because it frequently has discontinuities, i.e. is non-analytic. Figure 3.2 ilustrates the problem. Where actual bands cross, the simple energy ordering cannot keep with the right band and jumps to another, leading in fact to a discontinuity.

In order to be able to apply a gradient in momentum space we need to have the eigenstates ordered and grouped in such a way that analycity is guaranteed. That is the first goal of this suite of programs.

Band crossings

It is clear that the problem of classifying the eigenstates by bands where analycity applies is problematic at degeneracies i.e. band crossings.

There, any linear combination of eigenstates is also an eigenstate.

Lets consider two bands A and B, crossing at point k. The eigenstates of A are continuous at the left of k and at the right, and the same is true for eigenstates of B. But not necessarily at k. Supppose that ψ_A would be the wavefunction at k that would have continuity in band A and ψ_B for band B. Of course, ψ_A and ψ_B have the same eigenvalue.

But the numerical process of solving the Schrödinger equation at point k

can give any result of the form

$$\psi_1 = a\psi_A + b\psi_B \tag{3.6}$$

$$\psi_2 = a'\psi_A + b'\psi_B \tag{3.7}$$

as long as ψ_1 and ψ_2 are orthogonal (the coeficcients have to be such that they assure orthogonality). Of course, these wavefunctions are not continuous with A or B, in general.

There are two ways to workaround this problem and restore analycity. One is to interpolate the wavefunction at k using the closest eigenstates of band A and then the ones of band B. The other is to apply an unitary transformation.

Unitary transformation   The most frequent way of restoring analycity is to apply an unitary transformation

$$\begin{bmatrix} \psi_A \\ \psi_B \end{bmatrix} = U \begin{bmatrix} \psi_1 \\ \psi_2 \end{bmatrix} \tag{3.8}$$

where U is choosen in such a way as to 'iron out' the nonanalytic behaviour at k. In practice, this is done by minimizing the difference betwenn ψ_A and both ψ_A^- and ψ_A^+, and the same for ψ_B relative to ψ_B^- and ψ_B^+.

3.2   Graph Theory (GT)

Graph Theory (GT) studies the properties and the applications of a mathematical structure called a Graph. In a strict sense, a set of vertices (nodes) V and a set of edges E describes a Graph, $G(V, E)$. A vertex (node), $v \in V$, is a terminal point, and an edge, is a link between two nodes. This link must follow some relation, $R(v_1, v_2)$, between these nodes, for example, a co-authorship (edge) between two authors (nodes).

The edges are the set defined by

$$E = \{(v_1, v_2) : v_1, v_2 \in V \land \exists\, R(v_1, v_2)\} \tag{3.9}$$

The edges are oriented when the relationship between two nodes are diferent in each direction ($R(v_1, v_2) \neq R(v_2, v_1)$). Thus, the graph is a directed graph, for example, the papers' citations (one paper cites another, but the opposite is not true). However, if all pairs of nodes have the same edge in both directions ($R(v_1, v_2) = R(v_2, v_1)$) or the orientation is irrelevant, its graph is undirected.

Furthermore, all relations need not have the same weight. For example, getting back to the co-authorship graph, some authors may collaborate more times with a few than others. Thus, the edge between these two authors may include this weight $\omega(v_1, v_2)$ (it is a weighted graph). Nonetheless, if the weights ω_e of all edges e are the same ($\forall\, e \in E\, :\, \omega_e = \omega$) or the edges expresse a binary relation (may exist or not), the graph is unweighted. In this work, we will use unweighted and undirected graphs.

Another essential concept is that of the subgraph. A subgraph $g(V', E')$ is a graph formed by a subset of vertices $V' \subset V$ and edges $E' \subset E$ of a bigger graph $G(V, E)$.

A subgraph g, whose nodes do not have any edge between nodes of another subgraph $g', g \neq g'$ is called a component.

Graph theory is very well studied, and there are many implemented algorithms to study graphs' properties. We will use the *networkx* python module for the implementation and analysis of graphs.

3.3   Unsupervised Machine Learning (UML)

AI solves problems that normally require a human to make decisions. The cornerstone of Artificial Intelligence is designing agents that learn how to make informed decisions in possibly ever-changing environments without human supervision. Machine learning is a subset of AI where models learn patterns from datasets. How an artificial agent learns these patterns classifies them into Supervised or Unsupervised machine learning (UML) models.

Unlike supervised machine learning, which uses data labeled by an expert and learned patterns from comparison, UML algorithms are used for clustering or association because they identify patterns from uncategorized data. These algorithms recognize patterns in the datasets using some metrics to evaluate differences or similarities between data.

UML algorithms for clustering use a dataset of uncategorized points, and a problem dependent metric. The algorithm processes the points to form groups (clusters) represented by some pattern or structure determined by the metric. The algorithm is called exclusive when a datapoint belongs exclusively to a single cluster. The band-crossing problem lies in this class of algorithms due to its structure. Thus, we will work with this type of clustering algorithm.

Workflow

4.1 Preprocessing

After creating the files and directories mentioned in section Running, the next thing is to run the script **preprocess**,

```
berry preprocess inputfile
```

which will do the following:

1. Reads file `inputfile` with the description of the run wanted.

2. Runs the scf calculation using QUANTUM ESPRESSO, if it has not ran before.

3. Generates an nscf input file according to the input data of the run and runs an nscf calculation, using QUANTUM ESPRESSO, if it has not ran before.

4. Reads data from the nscf calculation and saves to a file.

5. Makes several simple calculations that will be used afterwards in other scripts, and save them to several files.

An input file for the berry run has to be created in the working directory with several parameters. The minimum parameters needed are:

- origin of k-points (k0)

- number of k-points in each direction (nkx, nky, nkz)

- step of the k-points (step)

- number of bands to be calculated (nbnd)

An example of the minimum input file is shown in figure 4.1. A full list of the parameters that are accepted with the respective defaults (if any) is shown in table 4.1. There is a unique reference number that is attributed when running **preprocess**, that can be changed in the input file, but it not recomendable, because it is too easy to forget to change that variable in the input file and attribute the same value for different runs.

```
k0 0.00 0.00 0.00
nkx 100
nky 100
nkz 1
step 0.005
nbnd 8
```

Figure 4.1: Example of input file for **preprocess**.

Table 4.1: List of variables for the input file of **preprocess**.

Role of keyword	Variable (and defaults)
origin of k-points	k0
number of k-points in each direction	nkx, nky, nkz
step of the k-points	step
number of bands to be calculated	nbnd
number of processors	npr = 1
dft directory	dftdirectory = 'dft/'
scf file name	name_scf = 'scf.in'
directory where wavefunctions will be saved	wfcdirectory = 'wfc/'
point in real space where all phases match	point = 1.178097
software for DFT	program = 'QE'
Unique reference of run	refname = date and time

The script **preprocess** creates the files:

- *phase.npy*

- *neighbors.npy*

- *eigenvalues.npy*

- *occupations.npy*

- *positions.npy*

- *kpoints.npy*

- *nktoijl.npy*

- *ijltonk.npy*

and *datafile.npy* which is the main data file.

File *neighbors.npy* lists the neighbors of each k-point (which neighbors to consider may be an input parameter in the future).

File *phase.npy* saves for each (k,r) the phase factor of the Bloch functions.

This finishes the preparatory phase.

4.2 Extraction of wavefunctions

The next step is to extract the wavefunctions from the format used in the DFT package (in this case it is QUANTUM ESPRESSO), make them coherent and save them in a format adequate for the **berry** suite.

The script that does this job is **wfcgen**.

It uses the package wfck2r.x of the QUANTUM ESPRESSO suite to extract and save the wavefunctions from the original format to a text file, in position space, one for each k-point and band.

Then a point in position space is choosen (see table 4.1), away from points of high symmetry, and all wavefunctions are multiplied by a phase such that at that point all wavefunctions have zero phase.

The wavefunctions are then saved in the directory choosen to save them (see table 4.1; default is `working_directory/`*wfc/*), and the files will be named in the form *k0**b0XX.wfc* where ** is replaced by the number of the k point and *XX* is replaced by the number of the band.

Bands are as given by the DFT software i.e. ordered by energy, and the lowest energy band is numbered zero and so on.

4.3 Dot product

dot reads the wavefunctions saved in directory *wfc*, removes the phase factor of the Bloch functions, and calculates the dot product of the Bloch factors of the neighboring k points.

The neighboring k points are the four nearest neighbors, as determined in the preprocessing and saved in file *neighbors.npy*.

In the end we get two files *dpc.npy* and *dp.npy*:

- *dpc.npy* contains for each pair of k-points and bands the dot product of the Bloch factors

- *dp.npy* contains the modulus of *dpc.npy*

4.4 Establishing the bands

Find which eigenvalues/eigenfunctions have continuity, by running program **cluster**.

This script uses graph theory and unsupervised machine learning to establish electronic bands that are analytic in k-space.

It uses data from files *dp.npy* and *eigenvalues.npy* to establish continuity between wavefunctions of neighboring k-points.

It retrieves two files:

bandsfinal.npy stores an array bandsfinal[k,b] = number of band that is continuous to band b.

signalfinal.npy signalfinal[k,b] is a measure of the quality of the attribution to a band.

4.5 Basis rotation

It is frequent that a basis rotation is applied to the wavefunctions in such a way as to maximize continuity, where there is a degeneracy.

For this, **basis** can be run. It works like this: Consider a point in k-space and two wavefunctions $|\Psi_1\rangle$ and $|\Psi_2\rangle$ that have been signaled as not continuous to the neighboring wavefunctions $|\Psi_A\rangle$ and $|\Psi_B\rangle$ that belong to band A and band B, respectively.

We will apply an unitary transformation

$$|\Psi'_A\rangle = a_1|\Psi_1\rangle + a2|\Psi_2\rangle$$
$$|\Psi'_B\rangle = b_1|\Psi_1\rangle + b2|\Psi_2\rangle$$

where we have the restrictions due to orthonormalization:

$$a_1 a_1^* + a_2 a_2^* = 1$$
$$b_1 b_1^* + b_2 b_2^* = 1$$
$$a_1^* b_1 + a_2^* b_2 = 0$$

Then we want this transformation to be such that maximizes continuity to bands A and B, so we want the dot products $\langle\Psi_A|\Psi'_A\rangle$ and $\langle\Psi_B|\Psi'_B\rangle$ to be close to 1, which is the maximum they can be.

We look for the set a_1, a_2, b_1, b_2 that give

$$\max\langle\Psi_A|\Psi'_A\rangle = \max\left(\langle\Psi_A|\Psi_1\rangle a_1 + \langle\Psi_A|\Psi_2\rangle a_2\right)$$
$$\max\langle\Psi_B|\Psi'_B\rangle = \max\left(\langle\Psi_B|\Psi_1\rangle b_1 + \langle\Psi_B|\Psi_2\rangle b_2\right)$$

It will create new wavefunctions with extension *.wfc1*, that will be used in subsequent calculations.

4.6 Convert wavefunctions to k space

With the bands right, now proceeds to calculate the wavefunctions in k-space and their gradients.

For that, run **r2k** with one or two arguments:

- If one, will calculate from band 0 to the value inputed.

- If two, will calculate for the set of bands between the two values, inclusive.

Will save the wavefunctions of each band in a compressed file *wfcpos#.npy* where # = number of band.

Will save the gradients of each band in a compressed file *wfcgra#.npy* where # = number of band.

4.7 Calculation of the Berry connections and curvatures

The Berry connections (equation 3.5) and curvature are calculated using the script **geometry**.

This scrip must have one or two integers as input.

If it is just one, the script will calculate for all combinations of bands from the first up to the number given as input.

If two integers are given, they will only calculate the Berry connection and curvature for that pair of bands.

4.8 Optical conductivity and second harmonic generation

Calculate the linear condutivity with the script **condutivity** and the second harmonic condutivity with the script **shg**.

Both need two or three arguments.

If there are two: the first is the last filled band, the second is last empty band. If the first argument is negative, it will only consider transitions from one band to the other, while if it is positive it will consider all bands below the first value and all the empty bands up to the second value. Real part has to be always positive and zero below the gap.

The third argument, if it exists, is the name of an input file that should have the values wanted for

enermax 2.5	Maximum energy (Ry)
enerstep 0.001	Energy step (Ry)
broadning 0.01j	energy broadning (Ry)

(these are the default values, if there is no input file)

Appendices

Glossary

This glossary indicates the meaning of the variables used in the suite **berry**.

npr	Number of processors for the run
nkx	Number of k-points in the x direction
nky	Number of k-points in the y direction
nkz	Number of k-points in the z direction
nks	Total number of k-points
nr1	Number of points of wfc in real space x direction
nr2	Number of points of wfc in real space y direction
nr3	Number of points of wfc in real space z direction
nr	Total number of points of wfc in real space
k0	Initial k-point (coordinates) (2pi/bohr)
step	Step between k-points (2pi/bohr)

dftdirectory	Directory of DFT files
name_scf	Name of scf file (without suffix)
name_nscf	Name of nscf file (without suffix)
wfcdirectory	Directory for the wfc files
prefix	Prefix of the DFT QE calculations
outdir	Directory for DFT saved files
dftdatafile	Path to DFT file with data of the run
berrypath	Path of BERRY files

a1	First lattice vector in real space (bohr)
a2	Second lattice vector in real space (bohr)
a3	Third lattice vector in real space (bohr)
b1	First lattice vector in reciprocal space (2π/bohr)
b2	Second lattice vector in reciprocal space (2π/bohr)
b3	Third lattice vector in reciprocal space (2π/bohr)

nbnd	Number of bands
eigenvalues	Array with all eigenvalues of the calculation (Ry)
occupations	Array with all occupations of the calculation
phase[i,nk]	Array with exp(1j*(kpoints[nk,0]*r[i,0] + kpoints[nk,1]*r[i,1] + kpoints[nk,2]*r[i,2]))
kpoints[nk,2]	Array with coordinates of all k-points $(2\pi/\text{bohr})$
r[i,2]	Array with coordinates of all points of space (bohr)
neig[nk,3]	Array with number of the four k-points around nk

List of files in the suite

All programs of this package are in python 3.

The package contains the python scripts in the main berry directory:

- preprocessing.py

- generatewfc.py

- dotproduct.py

- clustering_bands.py

- basisrotation.py

- r2k.py

- berry_geometry.py

- condutivity.py

- shg.py

and the auxiliary files:

- cli.py

- __init__.py

- _version.py

Visualization scripts are in directory 'viz':

- draw2D_machine.py

- draw2D_corrected.py

- viz_geometry.py

- viz_wfc.py

Also the subroutines (depend on the other scripts) are in directory '_subroutines':

- headerfooter.py

- contatempo.py

- parserQE.py

- loaddata.py

- dft.py

- comutator.py

- write_k_points.py

- clustering_libs.py

- clustering_signaling.py

Some utilities are in directory 'utils':

- emailSender.py

- jit.py

- _logger.py

- NestablePool.py

It is likely that some of these scripts will be merged with others while new ones will appear due to new functionalities.

Flowchart

berry flow chart

Preprocessing
(prepare files for running)

DFT calculations

Read data from DFT

**Extract eigenvectors
in real space from DFT**

**Use program from
DFT suite**

**Make wavefunctions
coherent**

**Find which wavefunctions
constitute a
diferenciable set (bands)**

Calculate wavefunction's gradient in k-space
(save data in appropriate format)

**Berry
connections**

**Berry
curvature**

**Other
calculations
...**

**Linear and
nonlinear
optical properties**

**Hall conductivity
Anomalous velocity
...**