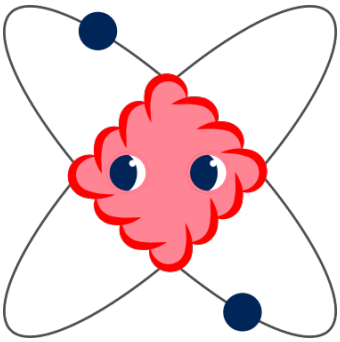


berry suite of programs

Documentation

v.2.0

October 23, 2023

Contents

Introduction

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 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
- Linear optical conductivity
- Second harmonic generation (SHG) optical conductivity

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;
- 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;
- Daniel Sousa (Braga, Portugal) for implementing the noncolinear code;
- Ícaro Jael Moura (Fortaleza, Brasil) for the exploratory work and testing;
- Afonso Duarte Ribeiro, author of the logo.

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 maillling list yet for reporting bugs or other questions, but an email can be sent to ricardo.ribeiro@physics.org or you can create an issue in the github page.

4 Aknowledgement

We aknowledge the Fundação para a Ciência e a Tecnologia (FCT) under project QUEST2D *Excitations in quantum 2D materials* PTDC/FIS-MAC/2045/2021 and in the framework of the Strategic Funding UIDB/04650/2020.

Instalation and running

1 PIP install

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

```
pip install berry-suite
```

Alternatively, it can be downloaded from the github
(https://ricardoribeiro-2020.github.io/berry/)
extracted to a directory and from that directory run the command:

```
pip install -e .
```

For developers, the developer dependencies can be installed using:

```
pip install berry-suite[dev]
```

or

```
pip install -e .[dev]
```

1.1 Patch for noncolinear calculations

As for the time of this version, the Quantum ESPRESSO suite of programs does not include a script to correctly extracts the spinors of the noncolinear calculation in real space.

The script that should do it is `wfck2r.x`, but it only writes the first function of the spinor. In order to use the **berry** correctly in noncolinear calculations, a modified version of `wfck2r.x` has to be used.

In appendix A, we explain how to do it.

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.

This version was tested with python 3.8 and 3.10, but likely works with other versions of python3.

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.

Scheme of working directory:

```
working_directory|
                 |input_file
                 |dft|
                     |scf.in
                     |pseudopotentials.upf
```

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

```
berry [package options] script parameter [script options]
```

where the `script` is one of the following

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

and the parameter is dependent on which script is beeing run. The command `berry` is a command line interface (CLI) of the package.

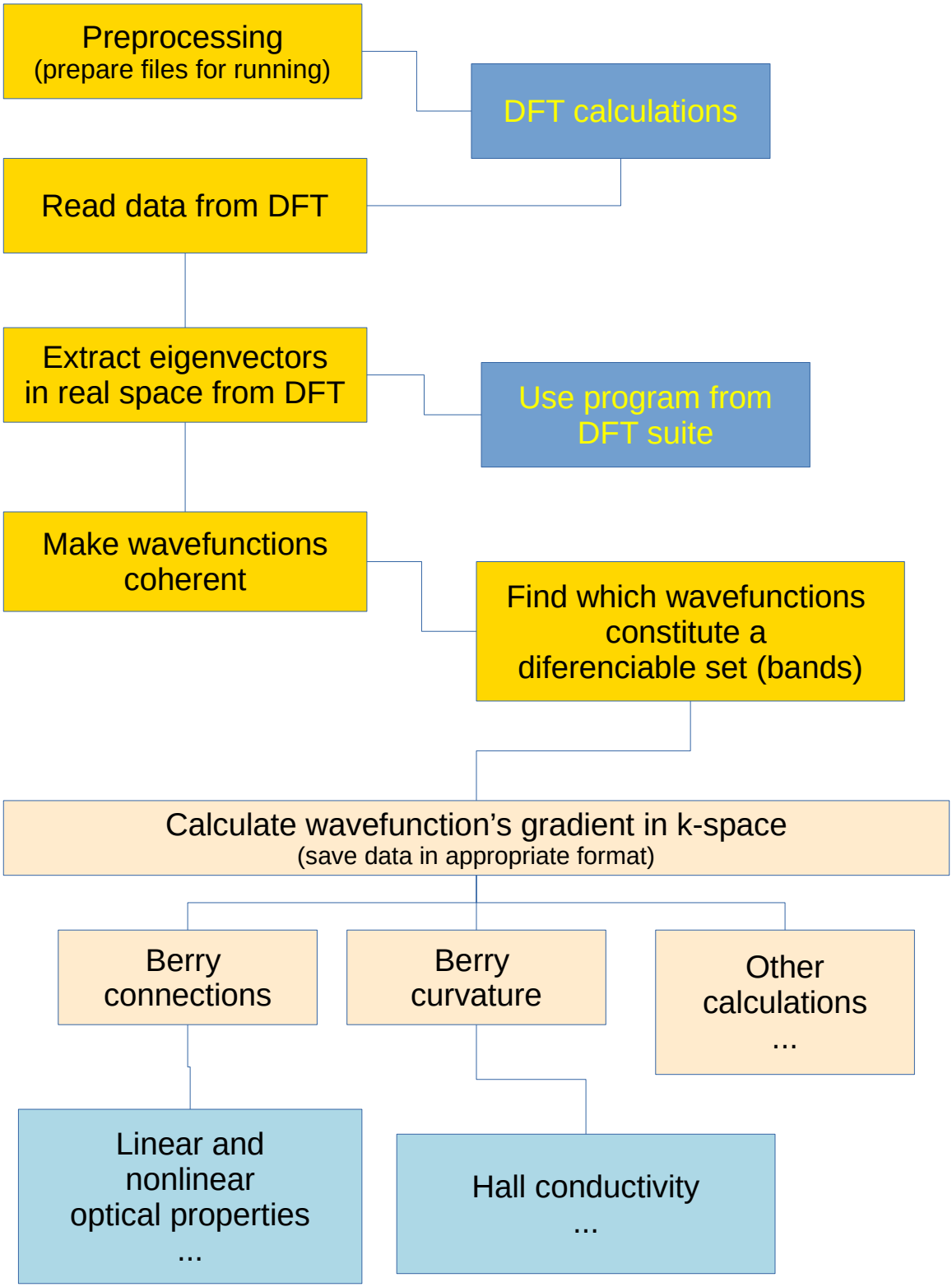
These scripts should be run in sequence, in the order shown, and the `berry` command assures that all the needed files are present for each run. The package options are shown in table 2.1.

Table 2.1: The package options.

`-h, --help`	Shows this help message and exit
`--version`	Displays current Berry version

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

berry flow chart



The flowchart of the previous page explains how **berry** works.

First a preprocessing script is run which runs the DFT calculations, reads data from those calculations, and saves it in files ready to be read by the folllowing scripts.

Then another script reads the wavefunctions from the DFT calculations and make them coherent. This is done with the help of a script from the DFT suite.

The main part is the one that classifies each wavefunction into a band, assuring continuity. Once this classification is done, the gradient in reciprocal space of the Bloch factors of the wavefunctions can be calculated.

After that, Berry connections and curvatures, as well as other calculations can be performed.

Theoretical background to *berry*

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 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.

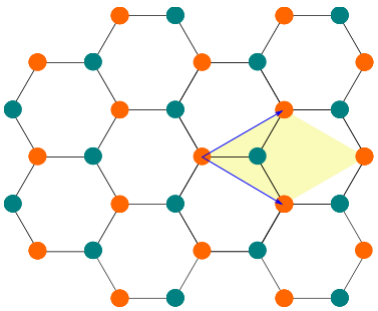


Figure 3.1: Example of a unit cell in a two dimensional material, with hexagonal symmetry. The yellow shadow represents the unit cell and the two blue vectors represent the lattice vectors.

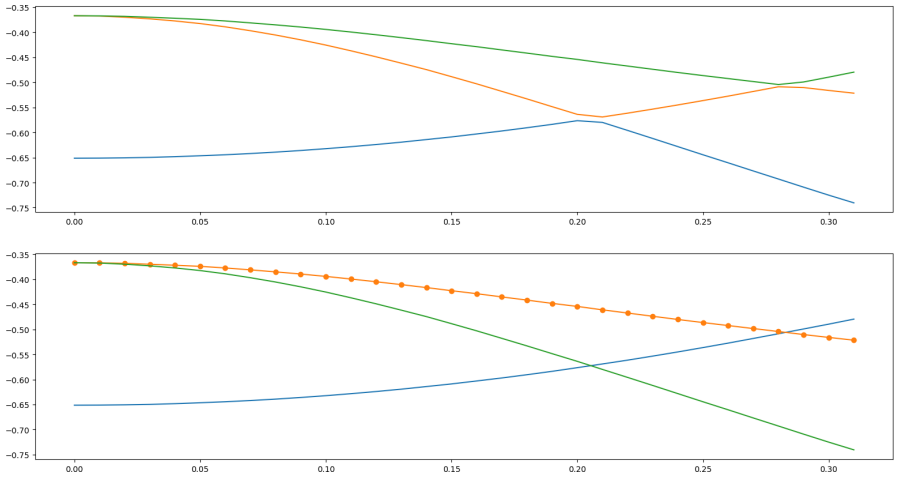


Figure 3.2: Example of electronic bands in one dimension. Different colors indicate different bands. Top: the bands according to the DFT output, that is, ordered by energy. Bottom: the bands according to continuity of the Bloch factors. The dots on the orange curve in the bottom figure indicate the positions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

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.

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^+.

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

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

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 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

1 Preprocessing

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

```
berry preprocess input_file [script options]
```

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 file 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 is 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.

The script `preprocess` creates the files:

- *phase.npy*
- *neighbors.npy*

```
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 (3 real numbers)	k0
number of k-points in each direction (integer)	nkx, nky, nkz
step of the k-points (real number)	step
number of bands to be calculated (integer)	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

- *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. The options for `preprocess` are shown in table 4.2.

Table 4.2: Options for script `preprocess`

`-h`, `--help`	Show a help message and exit
`-o file_path`	Name of output log file Extension will be ignored
`-flush`	Flush the ouput to stdout
`-v`	Increase output verbosity

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`:

```
berry wfcgen [script options]
```

It uses the package wfck2r.x of the QUANTUM ESPRESSO suite to extract and save the wavefunctions from the original format to a numpy file, in position space, one for each k-point and band. It uses the same number of processes as used before in the DFT calculations.

After extracting the wavefunctions, 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 have zero phase (are real).

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.

The options for `wfcgen` are shown in table 4.3. The default is to generate all formated wavefunctions, but for debugging it may be useful to generate just for a singke k-point or a single wavefunction, for which we have the options `-nk` and `-band` (has to be used in conjunction with `-nk`).

Table 4.3: Options for script `wfcgen`

`-nk`	Generates wavefunctions for a single k-point (default: All)
`-band`	Generates wavefunctions for a single k-point and band (default: All)
`-h`, `--help`	Show a help message and exit
`-o file_path`	Name of output log file Extension will be ignored
`-flush`	Flush the ouput to stdout
`-v`	Increase output verbosity

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.

```
berry dot [script options]
```

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*

The options for `dot` are shown in table 4.4

Table 4.4: Options for script `dot`

`-h, --help`	Show a help message and exit
`-np`	Number of processes to use (default: 1)
`-o file_path`	Name of output log file. Extension will be ignored.
`-flush`	Flush the ouput to stdout
`-v`	Increase output verbosity

4 Find the bands

To find which eigenvalues/eigenfunctions have continuity, one runs script `cluster`.

```
berry cluster [script options]
```

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.

A detailed explanation of the algorithm is given in the appendix Algorithm of `cluster` script.

The options for `cluster` are shown in table 4.5.

Table 4.5: Options for script `cluster`

`-h, --help`	Show a help message and exit
`-mb`	Minimum band to consider (default: 0)
`-np`	Number of processes to use (default: 1)
`-t [0.0-1.0]`	Tolerance used for graph construction (default: 0.95)
`-o file_path`	Name of output log file Extension will be ignored
`-flush`	Flush the ouput to stdout
`-v`	Increase output verbosity

5 Basis rotation

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

For this, `basis` can be run.

```
berry basis [script options]
```

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. The options for `basis` are shown in table 4.6

Table 4.6: Options for script `basis`

`-h, --help`	Show a help message and exit
`-np`	Number of processes to use (default: 1)
`-o file_path`	Name of output log file Extension will be ignored
`-flush`	Flush the ouput to stdout
`-v`	Increase output verbosity

6 Convert wavefunctions to k space

With the bands well defined, it is now necessary to proceed to calculate the wavefunctions in k-space and their gradients.

For that, run `r2k`:

```
berry r2k highest_band [script options]
```

It will calculate from band 0 to the value of the band given (`highest_band`), unless a minimum band is given. The highest band has to be given because in principle not all bands that were initialy calculated will be complete, and it doesn't make sense to use them in further calculations.

The wave functions originaly are saved for each k-point in a file, and the file contains the value of the wavefunction in each point of the real space. This program converts this format to one where for each point in real space there is a file with all the values of wavefunctions of the same band as a function of k-points, which is suitable to diferentiate.

$$\Psi_{\boldsymbol{k}}(\boldsymbol{r}) \rightarrow \Psi_{\boldsymbol{r}}(\boldsymbol{k})$$

Will save the wavefunctions of each band in a compressed file *wfcpos#.npy* where # = number of band and the gradients of each band in a compressed file *wfcgra#.npy* where # = number of band. The options for `r2k` are shown in table 4.7.

Table 4.7: Options for script `r2k`

`-h, --help`	Show a help message and exit
`-np`	Number of processes to use (default: 1)
`-mb`	Minimum band to consider (default: 0)
`-o file_path`	Name of output log file Extension will be ignored
`-flush`	Flush the ouput to stdout
`-v`	Increase output verbosity

7 Calculation of the Berry connections and curvatures

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

```
berry geometry highest_band [script options]
```

The highest band is the number of the last band to be used in the calculations. In principle, it will be the same as the one used in `r2k`. The options for `geometry` are shown in table 4.8.

Table 4.8: Options for script `geometry`

`-h, --help`	Show a help message and exit
`-np`	Number of processes to use (default: 1)
`-mb`	Minimum band to consider (default: 0)
`-prop` [both,connection,curvature]	Specify which property to calculate. (default: both)
`-o file_path`	Name of output log file Extension will be ignored
`-flush`	Flush the ouput to stdout
`-v`	Increase output verbosity

8 Optical conductivity and second harmonic generation

To calculate the linear condutivity, the script `condutivity` should be used and for the second harmonic condutivity, the script `shg`.

```
berry conductivity highest_band [script options]
```

and

```
berry shg highest_band [script options]
```

The highest band is the number of the last band to be used in the calculations, and has to be higher than the valence band. There are several options for the calculation that can be changed and are listed in table 4.9.

Table 4.9: Options for scripts `conductivity` and `shg`.

`-h, --help`	Show a help message and exit
`-np`	Number of processes to use (default: 1)
`-eM`	Maximum energy in Ry units (default: 2.5)
`-eS`	Energy step in Ry units (default: 0.001)
`-brd`	Energy broading in Ry units (default: 0.01)
`-o file_path`	Name of output log file Extension will be ignored
`-flush`	Flush the ouput to stdout
`-v`	Increase output verbosity

These scrips calculate the conductivity between 0 and the maximum energy (in Ry), with a step (energy step) and a broadning.

The details of these calculations can be found in

`https://arxiv.org/abs/2006.02744`

Appendices

Installing for noncolinear calculations

For noncolinear calculations, the fortran script `wfck2r.f90` has to be edited, because currently it only prints the first function of the spinor, and for **berry** calculations we need the full spinor in real space.

The steps for editing and compiling the fortran scipt are the following:

First, create a copy of `wfck2r.f90` located in PP/src of the QUANTUM ESPRESSO's home directory and name it `wfck2rFR.f90` in the same directory.

In the wfck2rFR.f90 file, on line 252, between `enddo` and `endif`, add in new lines:

```
if (noncolin) then
  do i3 = 1, dffts%nr3x, nevery(3)
  do i2 = 1, dffts%nr2x, nevery(2)
  do i1 = 1, dffts%nr1x, nevery(1)
    !val = local_average()
    val=dist_evc_r(i1+(i2-1)*dffts%nr1x+(i3-1)*dffts%nr1x*dffts%nr2x,2)
    write(iuwfcr+1,'("(",E20.12,",",E20.12,")")') val
  enddo
  enddo
  enddo
endif
```

So this is immediatly after writing the wavefunction (first of the spinor), and simply replicates that writing but for the second wavefunction of the spinor.

The first wavefunction of the spinor is on array `dist_evc_r(:,1)` and the second is on `dist_evc_r(:,2)`.

Now the file needs to be compiled using the same configuration as the rest of QUANTUM ESPRESSO. For this, one needs to edit the `Makefile` that is in the same directory PP/src of the QUANTUM ESPRESSO's home directory.

First we add `wfck2rFR.x` to the list of executables listed under the tag `all :`

Then add the following lines (for instance, bellow the equivalent lines for `wfck2r.x`):

```
wfck2rFR.x : wfck2rFR.o libpp.a $(MODULES)
        $(LD) $(LDFLAGS) -o $@ \
                wfck2rFR.o libpp.a $(MODULES)  $(QELIBS)
        - ( cd ../../bin ; ln -fs ../PP/src/$@ . )
```

(beware that tabs must be used on second to fourth line, and not spaces.)

After this, to compile, just execute the command on the directory PP/src:

```
make wfck2rFR.x
```

and it's done. You shuld see a link to the executable `wfck2rFR` on the `bin` directory of QUANTUM ESPRESSO.

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 (2π/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 'vis':

- _debug.py
- _geometry.py
- _wave.py

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

- headerfooter.py
- contatempo.py
- parserQE.py

- loaddata.py
- loadmeta.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

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

Algorithm of `cluster` script

1 Algorithm

This solution employs different techniques to solve the bands crossing problem as graph theory and unsupervised machine learning methods. The bands clustering algorithm uses the information of k-points' wavefunctions, the dot products of their Bloch factors, and their corresponding band energy. Thus, the solver algorithm iterates between two main pieces: the component identification based on graph theory and the samples assignment using unsupervised machine learning methods.

The solver's objective is to get closer to the best solution in each iteration by maximizing the number of solved bands.

The bands clustering algorithm uses the wave functions, their dot products, and their corresponding band energy for these calculations.

The output are five files that will be used by other programs, and will be described below.

In the following sections we will discuss in more detail the purpose of each band clustering algorithm's part and how it works.

1.1 Problem configuration and notation definition

We first need to organize the information problem in a convenient data structure. The essential elements we have are the energies $E_{k,n}$ for each k-point $\mathbf{k} \equiv (k_x, k_y)$ and band n, the k-points' neighbor identification, and the dot product between the neighboring Bloch factors. The *loaddata* module gives the algorithm most of this information; however, this module does not contain the dot product data; it is in the *dp.npy* file.

First, we create a mixed entity using a k-point and its band energy as if it were a vector on a "bands-energy" space. In the following, a variable $\mathbf{p}$ identifies a point in this space. Thus, this variable $\mathbf{p}$ contains the k-point coordinates and an eigenenergy of the k-point (which is a band designation).

$$\mathbf{p} \equiv (k_x, k_y, E_{k,n}) \tag{D.1}$$

The set of all k-points identifiers (k indices) is K, where $K \equiv [k = 0, \ldots, N-1]$, and $\mathbf{K} \equiv \{\mathbf{k}_0, \mathbf{k}_1, \ldots, \mathbf{k}_{N-1}\}$ is the set of all k-points vectors. N is the total number of k-points. It is possible to define an index p which uniquely determines a wavefunction' Bloch factors:

$$p = n\,N + k \tag{D.2}$$

We call P to the set of all p's. n is the original band as attributed by the DFT software; we define the set of n's: $B \equiv [0, n_B - 1]$. n_B is the total number of bands.

Then

$$P \equiv [0, n_B N - 1] \tag{D.3}$$

It leads to

$$E_{k,n} = E_p, \qquad k \in K,\ n \in B,\ p \in P \tag{D.4}$$

Let us consider the points p and $q \in P$, with q is neighbor of p (by neighbor we mean that the k-points that are implicit in the varaibles are adjacent in x or y directions in reciprocal space); the dot product between their Bloch factors is given by $0 \leq |\langle p|q\rangle| < 1$. If the Bloch factors p and q belong to the same band, and the band is analytic, since they are close together, their dot product should be close to one. On the other hand, this dot product should be close to zero if p and q belong to different bands, since different bands are orthogonal.

Thus, this data seems to give all the information necessary to solve the band-crossing problem. However, from a computational point of view, more is needed; during the implementation of this solution, several problems were found, and just a few of them were solved successfully.

Regardless, the dot product information is not wholly needless for values close to one $|\langle p|q\rangle| \approx 1$; it is possible to form groups of well-constructed groups of points. The existence of points and a way of measuring relations between them reminds the graph concept. This is the motivation for one of the algorithm's parts: use graph theory methods to identify these groups and, after that, use unsupervised machine learning techniques to classify them.

Consequently, this information makes it possible to construct the first data structure. It is a graph $G(V, E)$ (V is a set of vertices or nodes, and E is a set of edges (connections)) where each point p is a node, so that $V = P$, and if the connection (dot-product) is more significant than an adequate tolerance TOL, there is an edge between these nodes. For simplicity, this graph is undirected and does not have weights on its edges.

$$E = \{(p, q) \mid p, q \in P \ \ \wedge \ \ |\langle p|q\rangle| > \text{TOL}\} \tag{D.5}$$

Therefore, the initial state of the problem is prepared in a convenient data structures. The next step is the solver algorithm that will use that data and the graph previously constructed as the initial solution point. The solver algorithm will iterate over this information.

1.2 Solver Algorithm

The solver algorithm (SA) is an optimization process based on iteration for solving the band-crossing problem. As discussed before, this algorithm iterates between two parts. One piece is based on graph theory (GT), and the other is based on unsupervised machine-learning (UML) techniques. The initial conditions for this algorithm were described in the previous section. This SA follows the flux chart in figure D.1.

The SA can be simply described as various iterations starting in an initial state, and as a result some points are not attributed to their initial band and others are noted as having a strong connection between them. We call components to the sets of points with connections and if there are no connections, the single point is also considered a component. In the first iteration, these components were computed by constructing the graph using the dot-product information only.

The initial state of an iteration is the best result of the previous iterations, since SA is an optimization algorithm.

One SA iteration involves identifying the components, given the initial state in the form of a graph. Then, clustering these components gives larger components, that will

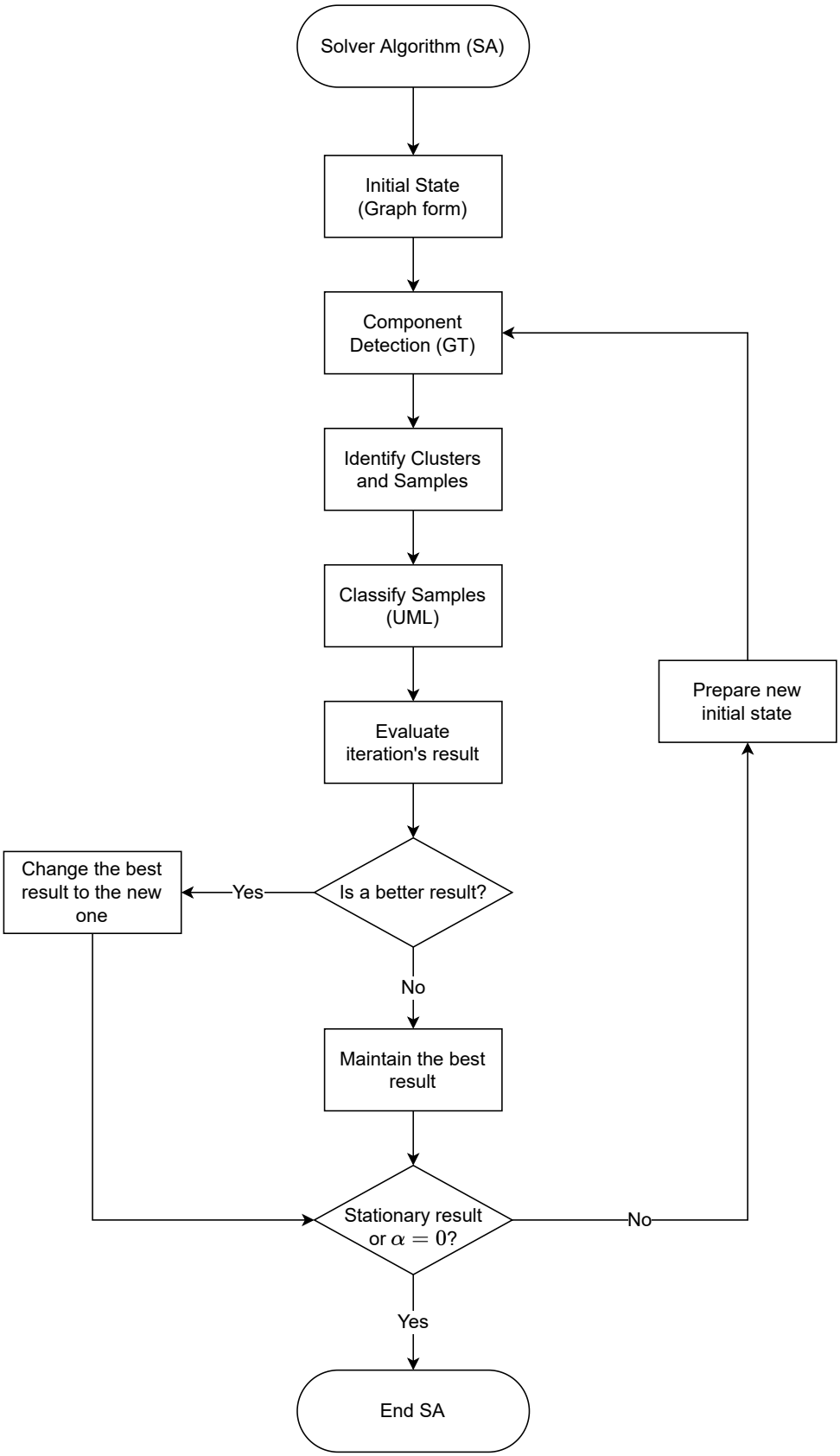


Figure D.1: Flowchart with a simplified description of the solver algorithm.

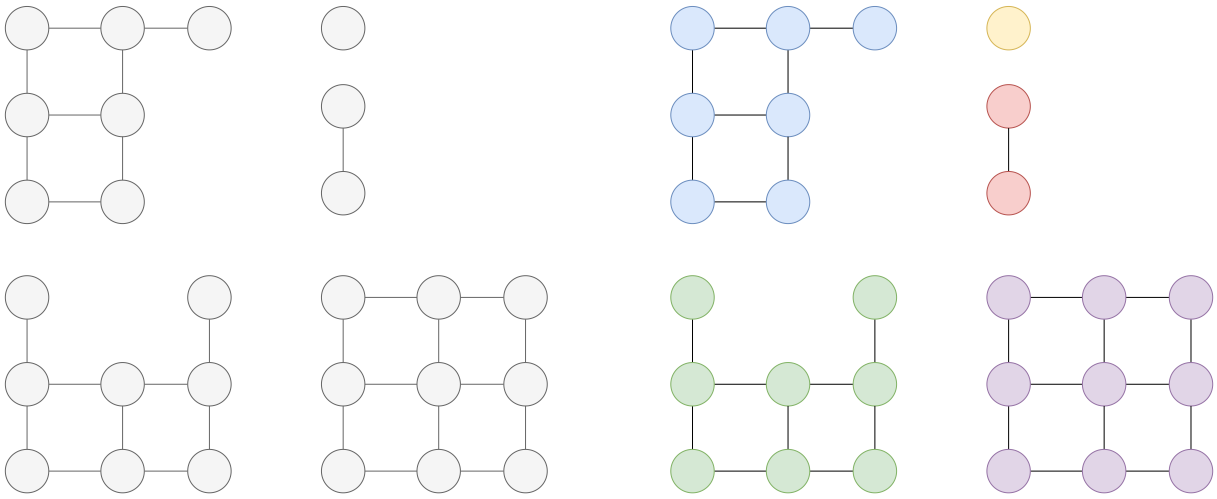


Figure D.2: Figure D.2a represents a general graph $G(V, E)$ where the vertices V are the gray circles, and the edges E are the line that connects them. Figure D.2b shows the G's components. Each component is identified with a different color.

form a new state that will be used as a initial state of the next iteration. The optimization process ends when the result does not change between iterations, reaching a stationary solution or the relaxation parameter reaches the minimum value. This relaxation value is a parameter for the second part of the algorithm, and will be discussed in more detail later.

1.3 Identification of problems and component detection

The concept of a component in GT is necessary to understand this part of the SA. A component is a sub-graph (i.e., a smaller graph g belonging to a bigger one G where $g \subset G$), with edges (connections) between its nodes, but no connections with the nodes of other components, as shown in the example in figure D.2.

These components are straightforward to identify using existing GT techniques, but as expected, they are not the final result are just portions of a band. For this reason, it is essential to classify these components to form completed bands. Hence, the SA uses UML techniques to perform the assignment.

This algorithm gives very good results for some materials; however, there are problems. Namely, a few points create "loops" between prohibited points. It is a prohibited path because, there should be no connection between two points in the same k-point (that is two wavefunctions with the same k-point must not belong to the same band). This results in the algorithm assuming two components that should be separated as a single component.

Two types of points create these "loops." One of them is straightforward to identify; they are degenerate points up to a numerical error (i.e., for p and $q \in P$ and the same k-point, their energy is numerically the same, $E_p = E_q$). Thus, it is only necessary to identify them and see if they create these forbidden paths. When found, the connection is broken.

The second type of points are not strictly degenerate but have the same behavior, complicating their detection. These more peculiar points are in high energy bands and, besides other inconveniences, limit the algorithm to obtain better results. Nonetheless,

for both types of problems, we need a more substantial justification for this behavior.

1.4 Unsupervised Machine Learning Techniques to Band Clustering

Since the SA uses UML techniques to classify the components, it is necessary to establish a clear notation. We call clusters the components with many points that can not join to another large component, because they are geometricaly incompatibles. These clusters are electronic bands of eingenenergies with part of their points attributed. We call samples to the rest of components, that can be assigned to clusters. When a cluster has the number of nodes (points p) equal to the total number of k-points, N, this cluster is solved (the band is solved). Otherwise, it is an incomplete cluster, and the sample assignment tries to complete it.

Supervised machine learning methods require large quantities of labeled examples, which we do not have. Thus, this problem lies in the unsupervised machine learning (UML) problems category. Consequently, this SA's part aims to identify the samples that complete the clusters using these UML methods.

The UML method needs a way to compare samples to clusters and evaluates the attribution. Therefore, let us define the similarity between two objects, X and Y, $\mathcal{S}(\mathrm{X},\mathrm{Y})$. The $\mathcal{S}(\mathrm{X},\mathrm{Y})$ value measures how similar these objects are based on one chosen metric. When the similarity is higher, the objects are more similar. We will use a normalized similarity. So, the $\mathcal{S}$ value must satisfy the following properties:

1. It is positive, $\mathcal{S}(\mathrm{X},\mathrm{Y}) > 0$
2. It is symetric, $\mathcal{S}(\mathrm{X},\mathrm{Y}) = \mathcal{S}(\mathrm{Y},\mathrm{X})$
3. It is maximum when $\mathrm{X} = \mathrm{Y}$, $\mathcal{S}(\mathrm{X},\mathrm{X}) = 1$

On the other hand, we use a similarity metric between samples and clusters that only needs their representative points. A cluster has many well-connected points, but only a few are required for comparison with a candidate sample to join the cluster. Only the cluster and sample boundary points may establish relations between them. Then, these are the representative points.

Identifying the representative points is an undersampling technique. We use the convolution of the Prewitt operator, (G_x, G_y), with the cluster's k-space projection A to determine their edges. The matrix A has the shape of k-space; each matrix's entry, A_{k_x,k_y}, is one if the cluster includes a point $\mathbf{p}(p)$ with that k-point and zero otherwise.

This operator is a simple discrete gradient that identifies high-frequency variations. These variations define the cluster's edges.

$$A_{k_x,k_y} = \begin{cases} 1, & \mathbf{k} \in \mathbf{K}_{\text{Cluster}} \subset \mathbf{K} \\ 0, & \text{otherwise} \end{cases} \tag{D.6}$$

The Prewitt operator G_x and G_y is defined as:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{D.7}$$

$$G_y = G_x^T = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{D.8}$$

Then, the gradient on the $\alpha = \{x, y\}$ direction of A is

$$A_\alpha = G_\alpha * A, \qquad \alpha = \{x, y\} \tag{D.9}$$

where $*$ is the convolution operator and the representative cluster edges are given by.

$$\text{edges} = \sqrt{A_x^2 + A_y^2} > 0 \tag{D.10}$$

Now, we can explicitly choose a metric to define the similarity that we will use. A similarity score between neighboring representative points is a weighted mean between the modulus of the dot product $|\langle p|q\rangle|$, and a function $f_p(E_q)$ of the energies E_q of the neighbor point q.

The similarity of a cluster with a sample is defined as the mean of the individual similarity score between neighboring representative points.

Thus, the similarity score, s_p^q, between the cluster's $p \in P$ point and its sample's $q \in P$ neighbor is:

$$s_p^q = \alpha\, |\langle p|q\rangle| + (1 - \alpha)\, f_p(E_q) \tag{D.11}$$

The value of α is a relaxation parameter in the optimization process. Its initial value is $1/2$, and it diminishes on each iteration. The α's minimum value is 0 due to normalization. The function $f_p(E_q)$ computes the normalized difference between the best-expected energy E_{best} with the actual q's energy, E_q.

$$f_p(E_q) = \frac{\min\{\Delta E_{\text{best}}^i\}_i}{\Delta E_{\text{best}}^q}, \qquad \Delta E_a^b = |E_a - E_b| \tag{D.12}$$

$$i \in \{nN + k : \forall_n \;\; n \in B\} \tag{D.13}$$

We consider a smooth variation of bands in bands-energy space. Thus, the energy profile in each band's $\mathbf{p}$ point can be approximated by a second-order polinomial. This approximation allows us to calculate the best energy value E_{best} by extrapolation from the neighboring points that belong to the cluster. If there are not enough points to fit the curve, the p's energy is used, $E_{\text{best}} = E_p$.

Comparing all possible energy values E_i for the corresponding q's k-point k, the value that minimizes the difference with the best energy is obtained.

Then, the similarity $\mathcal{S}(\mathrm{C}, \mathrm{S})$ between the cluster C and sample S is

$$\mathcal{S}(\mathrm{C}, \mathrm{S}) = \frac{1}{4\tilde{N}} \sum_{p \in \mathrm{C}} \sum_{q \in \mathrm{S}} s_p^q, \qquad \mathrm{C}, \mathrm{S} \subset P \tag{D.14}$$

Where $\tilde{N}$ is the total number of points in the sample's edge. The factor of 4 appears because each q point generally contains four neighbors.

Assigning samples to clusters uses similarity values, trying to maximize it between them. Hence, all the samples are compared to all clusters to obtain their similarities. The classification is made iteratively. For each iteration, one sample is assigned to the best cluster. After that, the similarity scores are recalculated between the samples and the modified cluster.

This UML algorithm is the SA's bottleneck in executing time. This complication comes from the fact that this part is mostly computed linearly (i.e., the result on each iteration depends on the previous result). Therefore parallelizing it is challenging.

Ultimately, all the samples are assigned to one cluster leading to a band solution. However, as mentioned before, many problematic points exist. In particular, high energy

bands usually have many oscilations, and in this case the assumption of smooth variation is inaccurate, and the metric criteria has to be improved.

1.5 Evaluation and optimization of the result

The full SA is determinist because the result is always the same for the same conditions. However, after one SA iteration (GT and UML methods), it obtains one solution for the band-crossing problem. Different iterations obtain different number of bands completed and correct, which we want to maximize. Bands may have points wrongly attributed to them, and so an verification and evaluation is needed.

We use two stages to evaluate the iteration result. The first one considers only the dot-product information. It identifies and signals points acording to their assignment as indicated in table D.1. A point is correctly assigned when the mean dot-product $\overline{|\langle p|q\rangle|}$ is close to one.

Table D.1: Signals for the attribution of points to bands.

Signal	Description	Condition
0	Not solved	
1	Wrong	$\overline{\lvert\langle p\rvert q\rangle\rvert} \leq 0.2$
2	Degenerate	
3	Potential mistake	$0.2 < \overline{\lvert\langle p\rvert q\rangle\rvert} \leq 0.8$
4	Potential correct	$0.8 < \overline{\lvert\langle p\rvert q\rangle\rvert} \leq 0.9$
5	Correct	$\overline{\lvert\langle p\rvert q\rangle\rvert} > 0.9$

This stage signals some points that require more analysis, namelly the ones that are signaled as potentialy correct or mistaken. The second stage of evaluation analyzes these points to decide if they are effectively correct or not. This evaluation uses the energy continuity criteria (i.e., the bands need to be analytical in bands-energy space). Using the energy profile approximation by a second-order polynomial, it is possible to evaluate the continuity for each point's direction. Thus, each point receives a new signal value related to the number of directions D, that preserves the energy continuity.

Table D.2: Signals for the attribution of points to bands.

Signal	Description	Condition
0	Not solved	
1	Wrong	$D = 0$
2	Degenerate	
3	Other	$0 < D < 4$
4	Correct	$D = 4$

Furthermore, the total solution (all solved bands) is scored using the dot-product criteria. The score is the mean overall $\overline{|\langle p|q\rangle|}$ of the p-points of the band. This new information makes it possible to compare the final solution with the one of the previous iteration.

We consider the best result the one that reaches more solved bands and a larger score for each band. Consequently, this result will be the starting point for the next iteration.

The points not attributed, the mistakes, and the doubtful points became nodes with no connections. The points signaled as correct form well-constructed bands' portions. Thus, these points are nodes with edges between them. Accordingly, the collection of

these nodes and edges forms the new graph. Therefore, the new initial state is ready and compatible with the first step of the SA (it needs the data in a graph form).

The next iterations will try to solve the wrong attributions to obtain a better result until it reaches a stationary point or the relaxation criteria ends.

2 Bands Clustering program (BCP)

The python program *bands_clustering.py* implements the steps needed to use the algorithm explained previously. This program reads and prepares all the dependencies to perform the bands clustering algorithm in the *bands_libs.py* program.

The BCP uses the results of the previous berry suit algorithms. The needed results from the previously ran programs are:

Table D.3: Previous berry suit programs save their results in various files. This table shows the required ones.

Description	File
The material's main data	*datafile.npy*
The modulus of Bloch's factor dot products $\|\langle p\|q\rangle\|$	*dp.npy*
The eigenvalues	*eigenvalues.npy*
A list with the neighbors of each k-point	*neighbors.npy*

The berry suite must be installed in the python environment. Then, the following command line executes the BC program in the material directory.

```
$ berry cluster
```

Note that running the algorithm is only possible if all the dependencies exist.

Also, it is possible to change some of the default behaviour by passing some parameters, which are shown in table D.4.

Table D.4: Parameters that are received as optional arguments to the BCP.

Description	Flag	Default
The maximum band to solve	Mb	n_B
The minimum band to consider	-mb	0
Number of processes to use	-np	1
Dot-product tolerance used for graph construction	-t	0.95