# **berry** suite of programs

Documentation

v.0.3

Ricardo Mendes Ribeiro

July 7, 2021

# 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.

All programs of this package are in python 3.

The package contains the python scripts:

- preprocessing.py
- generatewfc.py
- dotproduct.py
- compara.py
- interpolation.py
- basisrotation.py
- r2k.py
- berryConnection.py
- berryCurvature.py
- condutivity.py
- shg.py

and the auxiliary scripts:

- draw2D_machine.py
- draw2D_corrected.py
- drawberry.py

- drawcurvature.py
- emailSender.py

Also with some subroutines (depend on the other scripts):

- headerfooter.py
- contatempo.py
- parser.py
- loaddata.py
- dft.py
- comutator.py
- write_k_points.py

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

## 1.1 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

- Ícaro Jael Moura (Fortaleza, Brasil) for the exploratory work and testing;
- Irving Leander Reascos Valencia (Braga, Portugal) for improving the performance and AI work;
- 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.

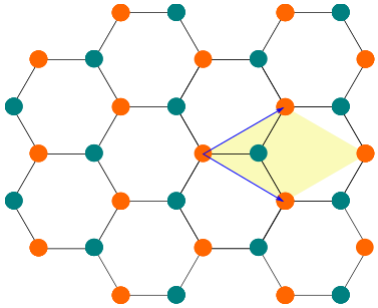

Figure 1.1: Example of a unit cell in a two dimensional material, with hexagonal symmetry. The yellow shadow represents the unit cell and the two blue vectors represent the lattice vectors.

## 1.2 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 still no mailling list for reporting bugs or other questions, but an email can be sent to ricardo.ribeiro@physics.org

## 1.3 Electronic structure calculations basics

### Crystal

A crystal is defined by its translational symmetry. A set of atoms, a pattern, 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

$$\boldsymbol{R} = n_1\boldsymbol{a}_1 + n_2\boldsymbol{a}_2 + n_3\boldsymbol{a}_3 \tag{1.1}$$

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:

$$H_{\boldsymbol{k}}\psi_{\boldsymbol{k},s}(\boldsymbol{r}) = E_{\boldsymbol{k},s}\psi_{\boldsymbol{k},s}(\boldsymbol{r}) \tag{1.2}$$

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 1.2 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 1.2 are given by Bloch's theorem:

$$\psi_{\boldsymbol{k}s}(\boldsymbol{r}) = e^{i\boldsymbol{k}\cdot\boldsymbol{r}}u_{\boldsymbol{k}s}(\boldsymbol{r}) \tag{1.3}$$

with

$$u_{\boldsymbol{k}s}(\boldsymbol{r}) = u_{\boldsymbol{k}s}(\boldsymbol{r} + \boldsymbol{R}) \tag{1.4}$$

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 1.2, $E_{\boldsymbol{k},1}$ 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:

$$\boldsymbol{\xi}_{\boldsymbol{k}ss'} = i\langle u_{\boldsymbol{k}s}|\nabla_{\boldsymbol{k}}u_{\boldsymbol{k}s'}\rangle = \frac{i}{v_C}\int_{uc} d^3\boldsymbol{r}\, u^*_{\boldsymbol{k}s}(\boldsymbol{r})\nabla_{\boldsymbol{k}}u_{\boldsymbol{k}s'}(\boldsymbol{r}) \tag{1.5}$$

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

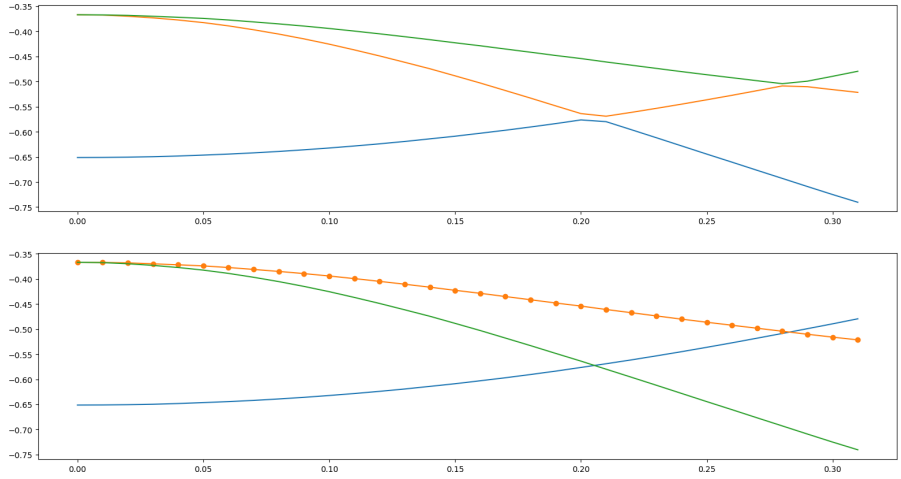


Figure 1.2: Example of electronic bands in one dimension. Different colors indicate different bands. Top: the bands according to the DFT output, that is, ordered by energy. Bottom: the bands according to continuity of the Bloch factors. The dots on the orange curve in the bottom figure indicate the positions 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 1.2 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 only 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 $\psi_A$ would be the wavefunction at $k$ that would have continuity in band $A$ and $\psi_B$ for band $B$. Of course, $\psi_A$ and $\psi_B$ have the same eigenvalue.

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

can give any result of the form

$$\psi_1 = a\psi_A + b\psi_B \tag{1.6}$$
$$\psi_2 = a'\psi_A + b'\psi_B \tag{1.7}$$

as long as $\psi_1$ and $\psi_2$ 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.

**Interpolation** Assume that $\psi_A^-$ and $\psi_A^+$ are the wavefunctions immediatly at left and right of $k$ that belong to band $A$, and $\psi_B^-$ and $\psi_B^+$ are the same for band $B$. Wavefunctions of bands $A$ and $B$ in the same point in momentum space are orthogonal, so $\psi_A^- \perp \psi_B^-$ and $\psi_A^+ \perp \psi_B^+$. Even a simple interpolation

$$\psi_A = \frac{1}{2}\left(\psi_A^- + \psi_A^+\right) \tag{1.8}$$
$$\psi_B = \frac{1}{2}\left(\psi_B^- + \psi_B^+\right) \tag{1.9}$$

will lead to $\psi_A \perp \psi_B$ and restore analycity within the numerical precision, since $\psi_A^-$ and $\psi_A^+$ are very close and continuous, and the same for $\psi_B^-$ and $\psi_B^+$. This results from the continuity of the hamiltonean in momentum space.

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

$$\begin{bmatrix}\psi_A \\ \psi_B\end{bmatrix} = U \begin{bmatrix}\psi_1 \\ \psi_2\end{bmatrix} \tag{1.10}$$

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 $\psi_A$ and both $\psi_A^-$ and $\psi_A^+$, and the same for $\psi_B$ relative to $\psi_B^-$ and $\psi_B^+$.

# Instalation

## 2.1 Download

The github packages should be downloaded from

https://github.com/ricardoribeiro-2020/berry/archive/master.zip

unziped in the directory where the instalation is to be done and the directory structure kept.

With this, each python script of the suite should run, giving the full path to the script.

## 2.2 Requirements

This software uses python3 with the following libraries, that have to be installed (most of these should already be there by default):

- numpy
- sys
- time
- joblib
- math
- os
- findiff
- matplotlib.pyplot
- mpl_toolkits.mplot3d

- xml.etree.ElementTree

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 **preprocessing.py** 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, using the full path to the python scripts.

**Sequence** of programs to run should be the following:

```
preprocessing.py   inputfile
generatewfc.py
dotproduct.py
compara.py [k-point]
interpolation.py [band] or basisrotation.py [band]
r2k.py  band1 [band2]
berryConnection.py  band1 [band2]
berryCurvature.py   band1 [band2]
condutivity.py   band1 band2 [inputfile]
shg.py   band1 band2 [inputfile]
```

In brackets are optional parameters.

# Workflow

## 3.1 Preprocessing

After creating the files and directories mentioned in section Running, the next thing is to run the script **preprocessing.py**, which will do the following:

1. Reads file 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 3.1. A full list of the parameters that are accepted with the respective defaults (if any) is shown in table 3.1. There is a unique reference number that is attributed when running **preprocessing.py**, 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.

Figure 3.1: Example of input file for **preprocessing.py**.

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

Table 3.1: List of variables for the input file of **preprocessing.py**.

| **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 **preprocessing.py** 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.

## 3.2 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 **generatewfc.py**.

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 3.1), away from points of high symmetry, and all wavefunctions are multiplied by a phase such that at that point all wavefunctions have zero phase.

Then the wavefunctions are saved in the directory choosen to save them (see table 3.1; default is *./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.

## 3.3 Dot product

**dotproduct.py** 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 wavefunctions
- *dp.npy* contains the modulus of *dpc.npy*

## 3.4 Establishing the bands

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

This program reads the dot product file *dp.dat* and uses it to check continuity.

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] equals -1 if there is an inconsistency at that k-point or a number $\geq 0$ giving the number of times it found a continuity. The larger the value, the better.

**compara.py** may need to be run several times until the results are good. Not that it improves as the number of trials increases, but the starting point (which is choosen randomly, but can be specified explicitly as an input parameter) may simply be a better one to find the continuities.

## 3.5 Interpolation

As discussed in subsection 1.3 of the Introduction, there may be a few k points where there are actual discontinuities and so need to restore analitycity.

For this, **interpolation.py** should be run, which uses the interpolation method described in that subsection.

The interpolation done by **interpolation.py** is not a simple mean value but a polinomial interpolation that uses up to six surrounding k points to do the job.

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

Since you probably just want to interpolate in the bands that are well determined, and argument with the last band number is needed.

```
python3 interpolate.py [band number]
```

## 3.6 Basis rotation

Instead of the interpolation suggested in the last section, it is more frequent that a basis rotation is applied to the wavefunctions in such a way as to maximize continuity.

For this, **basisrotation.py** 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.

Since you probably just want to interpolate in the bands that are well determined, and argument with the last band number is needed.

```
python3 basisrotation.py [band number]
```

## 3.7 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.py** 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#.gz* where # = number of band. This will be a dictionary with a label for each position in real space (from 0 to nr) and for each label a numpy array with the complex value of the wavefunction for each k point in momentum space.

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

## 3.8 Calculation of the Berry connections and curvatures

The Berry connections (equation 1.5) are calculated using the script **berryConnection.py** and the Berry curvature with **berryCurvature.py**.

These scrips must have one or two integers as input.

If it is just one, the scripts 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 (or curvature) for that pair of bands.

## 3.9 Optical conductivity and second harmonic generation

Calculate the linear condutivity by running **condutivity.py** and the second harmonic condutivity by running **shg.py**.

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)

# 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 ($2\pi$/bohr) |
| b2 | Second lattice vector in reciprocal space ($2\pi$/bohr) |
| b3 | Third lattice vector in reciprocal space ($2\pi$/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 ($2\pi$/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 |