# postqe Documentation

***Release 0.1***

**M. Palumbo, D. Brunato, P. D. Delugas**

October 18, 2017

**1 Introduction** **3**
1.1 Installation . . . 3
1.2 General notes . . . 4
1.2.1 Version . . . 4
1.2.2 Plotting . . . 4

**2 Tutorial** **5**
2.1 Fitting the total energy using Murnaghan EOS (examples 1) . . . 5
2.2 Calculate and plot the band structure of silicon (examples 2) . . . 6
2.3 Calculate and plot the density of states (DOS) of silicon (examples 3) . . . 7
2.4 Plotting a 1D section of the charge density (examples 4) . . . 9
2.5 Plotting a 2D section of the charge density (examples 5) . . . 10
2.6 Plotting 1D sections of different potentials (examples 6) . . . 11
2.7 Export formats (examples 7) . . . 15

**3 postqe package** **17**
3.1 Submodules . . . 17
3.2 postqe.api module . . . 17
3.3 postqe.eos module . . . 22
3.4 postqe.dos module . . . 22
3.5 postqe.charge module . . . 22

**4 Indices and tables** **25**

**Python Module Index** **27**

**Index** **29**

Contents:

CHAPTER

ONE

# INTRODUCTION

`postqe` is a Python package for postprocessing of results obtained with the Quantum Espresso (*QE*) code [1]. The package provides Python API functions for the most common tasks, such as plotting the charge density or fitting the total energy with an equation of state (EOS). It also makes available in Python some QE functionalities using the F2PY code [2] and wrappers to generate Python modules from QE dynamically linked libraries. Finally, it also includes an interface with the popular Atomic Simulation Environment (*ASE*) [3], from which in fact leverages for some functionalities.

The package implents some Python classes for handling the most important quantities, such as the charge or a potential. Some classes are derived from *ASE* and adapted to the postprocessing needs of QE.

It is meant to be imported into your own Python scripts or used from the command line interface (see the Tutorial for some examples). It is also meant for people who want to tinker with the code and adapt it to their own needs. The package is based on *numpy*, *scipy*, *matplotlib* and *ASE* libraries.

Finally it is a software framework where Quantum Espresso developers or advanced users may implement new functionalities which are needed by the community. In this respect, it offers the possibility to develop code in different languages (Python, C/C++, Fortran) and then use Python to "glue" everything together.

Current features of the package include:

- Fit the total energy $E_{tot}(V)$ with an equation of state (Murnaghan, Vinet, Birch, etc.)
- Calculate and plot the electronic band structure
- Calculate and plot the electronic density of states (DOS)
- Plot 1D, 2D or 3D sections of the charge density
- Plot 1D, 2D or 3D sections of different potentials (Hartree, exchange-correlation, etc.)

## Installation

You can download all package files from GitHub and then install it with the command:

```
sudo python setup.py install
```

The most useful functions for the user are directly accessible. You can import all of them as:

```
from postqe import *
```

[1] http://www.quantum-espresso.org/
[2] https://docs.scipy.org/doc/numpy-dev/f2py/
[3] https://wiki.fysik.dtu.dk/ase/

or you can import only the ones you need. The above command also makes available a number of useful constants that you can use for unit conversions.

More functions are available as submodules. See the related documentation for more details. Note, however, that most of these functions are less well documented and are meant for advanced users or if you want to tinker with the code.

# General notes

## Version

The package is still under development. Hence, it may contain bugs and some features may not be fully implemented. Use at your own risk.

## Plotting

`postqe` uses the *matplotlib* library for Plotting. Some functions in the package are simply useful wrappers for *matplotlib* functionalities of common uses. They return a *matplotlib* object which can be further adapted to specific needs and personal taste. Alternatively, you can of course manipulate and plot the post-procecessed data with any other Python tool of your choice.

It is also possible to export the charge (and various potentials) into text files according to different available formats (XSF XCrySDen format, cube Gaussian format, Gnuplot formats, contour.x and plotrho.x formats).

CHAPTER

TWO

# TUTORIAL

This is a simple tutorial demonstrating the main functionalities of `postqe`. The examples below show how to use the package to perform the most common tasks. The code examples can be found in the directory *examples* of the package and can be run either as interactive sessions in your Python interpreter or as scripts. It is also possible to obtain the same results using the command line and the available commands and options of `postqe`, as detailed in the following. You should run the example scripts or command-line commands from the corresponding example directory, which contains the proper input files.

The tutorial is based on the following examples:

| Example n. | Description |
|---|---|
| 1 | Fitting $E_{tot}(V)$ for a cubic (isotropic) system using Murnaghan EOS |
| 2 | Calculate and plot the band structure of silicon |
| 3 | Calculate and plot the density of states (DOS) of silicon |
| 4 | Plotting a 1D section of the charge density |
| 5 | Plotting a 2D section of the charge density |
| 6 | Plotting 1D sections of different potentials |

Several simplified plotting functions are available in `postqe` and are used in the following tutorial to show what you can plot. Note however that all plotting functions need the matplotlib library, which must be available on your system and can be used to further tailor your plot.

## Fitting the total energy using Murnaghan EOS (examples 1)

The simplest task you can do with `postqe` is to fit the total energy as a function of volume $E_{tot}(V)$. You can use an equation of state (EOS) such as Murnaghan's or similar. Currently you can use Murnaghan, Vinet, Birch, Birch-Murnaghan, Pourier-Tarantola and Anton-schmidt EOS and 3rd order (direct and inverse) polynomials in `postqe`. See the documentation of `get_eos()` for details.

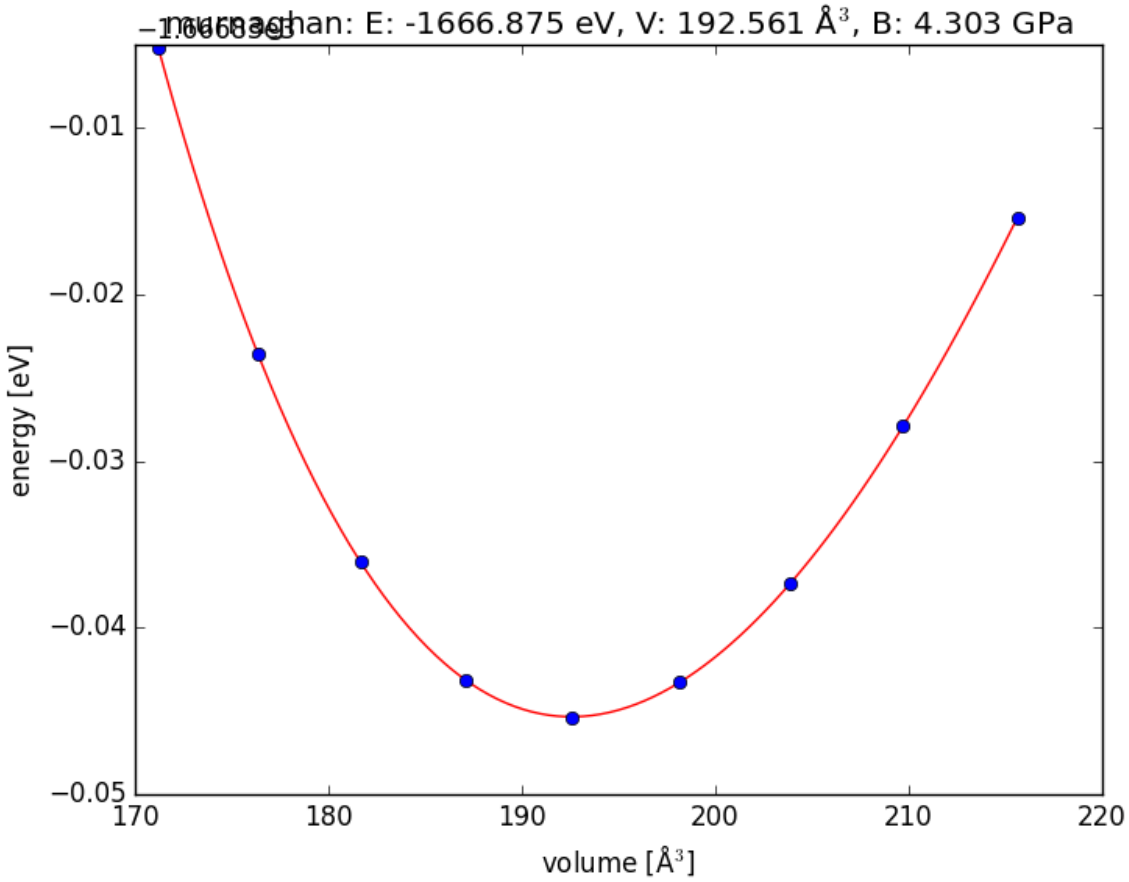
Let's see how to fit $E_{tot}(V)$. This is the case of isotropic cubic systems (simple cubic, body centered cubic, face centered cubic) or systems which can be approximated as isotropic (for example an hexagonal system with nearly constant $c/a$ ratio).

```
from postqe import units, get_eos

eos = get_eos(prefix='Nienergies.dat', eos_type='murnaghan')
v0, e0, B = eos.fit()
# Print some data and plot
print('Equilibrium volume = '+str(v0)+' Ang^3')
print('Equilibrium energy = '+str(e0)+' eV')
print('Equilibrium Bulk modulus = '+str(B / units.kJ * 1.0e24)+' GPa')
eos.write()
fig = eos.plot('Ni-eos.png', show=True)
```

```
# Save the plot in a different format (pdf) with Matplotlib if you like
fig.figure.savefig('figure.pdf', format='pdf')
```

The `get_eos()` needs in input a file with two columns: the first with the volumes (in $a.u.^3$), the second with energies (in $Ryd/cell$). You also define here what EOS to use, in this case Murnaghan's. This function returns an *eos* object. The method `fit()` performs the fitting and returns the equilibrium volume *v0*, energy *e0* and bulk modulus *B*. The fitting results can then be printed or further processed.

Optionally, you can plot the results with the `plot_EV()`. The original data are represented as points.




It is possible to obtain the same results using the command line and the available commands and options of `postqe`. For this example, you must type:

```
$ postqe eos -prefix Nienergies.dat -eos_type murnaghan -fileout eos -fileplot Ni-eos.png
```

Note that you can get help from the command line with the -h or –help options. Try for example:

```
$ postqe -h
$ postqe eos -h
```

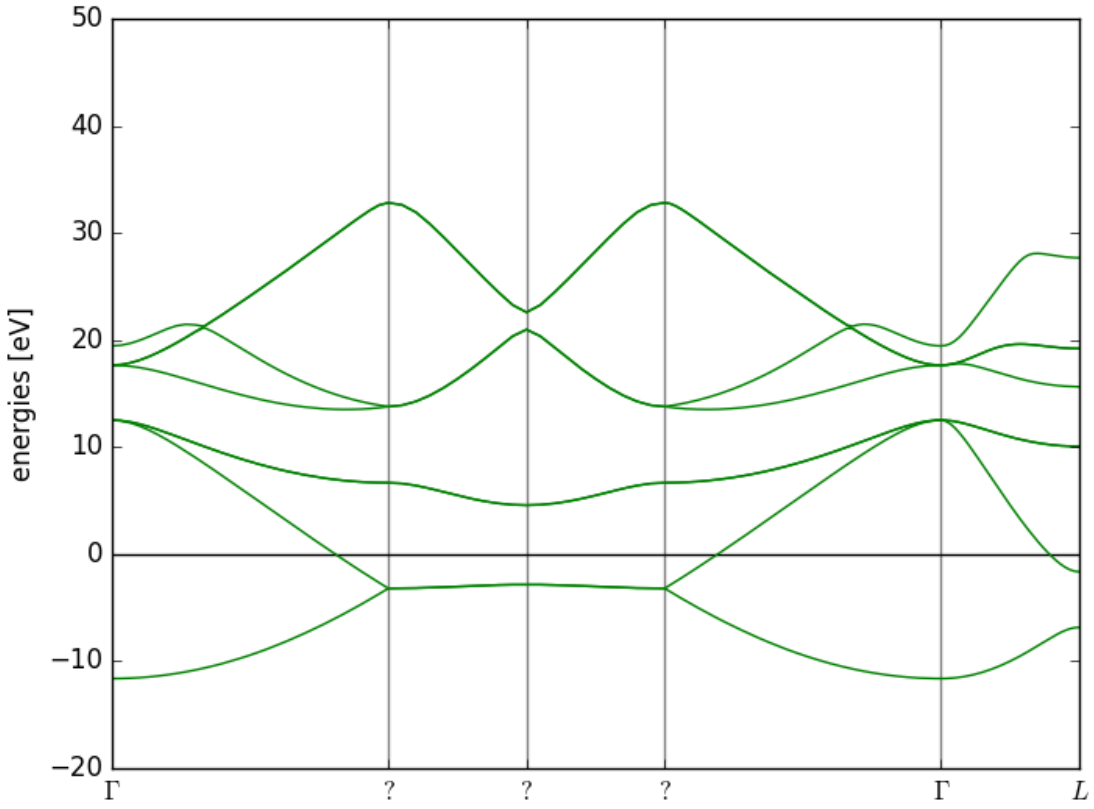
## Calculate and plot the band structure of silicon (examples 2)

This example shows how to calculate the electronic band structure of silicon with `postqe`. All necessary information is extracted from the standard xml output file of Quantum Espresso, produced by a proper calculation along the desired

path in the Brillouin zone.

```
bs = get_band_structure(prefix='Si', schema='../../schemas/qes.xsd', reference_energy=0)
fig = bs.plot(emin=-20, emax=50, show=True, filename='Sibands.png')
```

The `get_band_structure()` needs a parameter *prefix* which identies the system and the corresponding xml file (prefix.xml). As customary in Quantum Espresso, you can optionally provide an *outdir* contain the full path to the file (and other output files). If not provided, `postqe` tries to get the ESPRESSO_TMPDIR environment variable. If this fails too, it assumes the output file are in the current working directory. The *schema* (optional) parameter allows the code to properly parse and validate the xml file. The necessary xml schema is fetched from the source specified in the xml files itself, but the user can override this default schema by setting the *schema* parameter. The parameter *reference_energy* (usually the Fermi level) allows you to shift the plot accordingly. `get_band_structure()` returns a band structure object which can be further processed. For example, the method `plot()` creates a figure and save it in a png file.




The above results can be obtained from the command line typing:

```
$ postqe bands -prefix Si -schema ../../schemas/qes.xsd
```

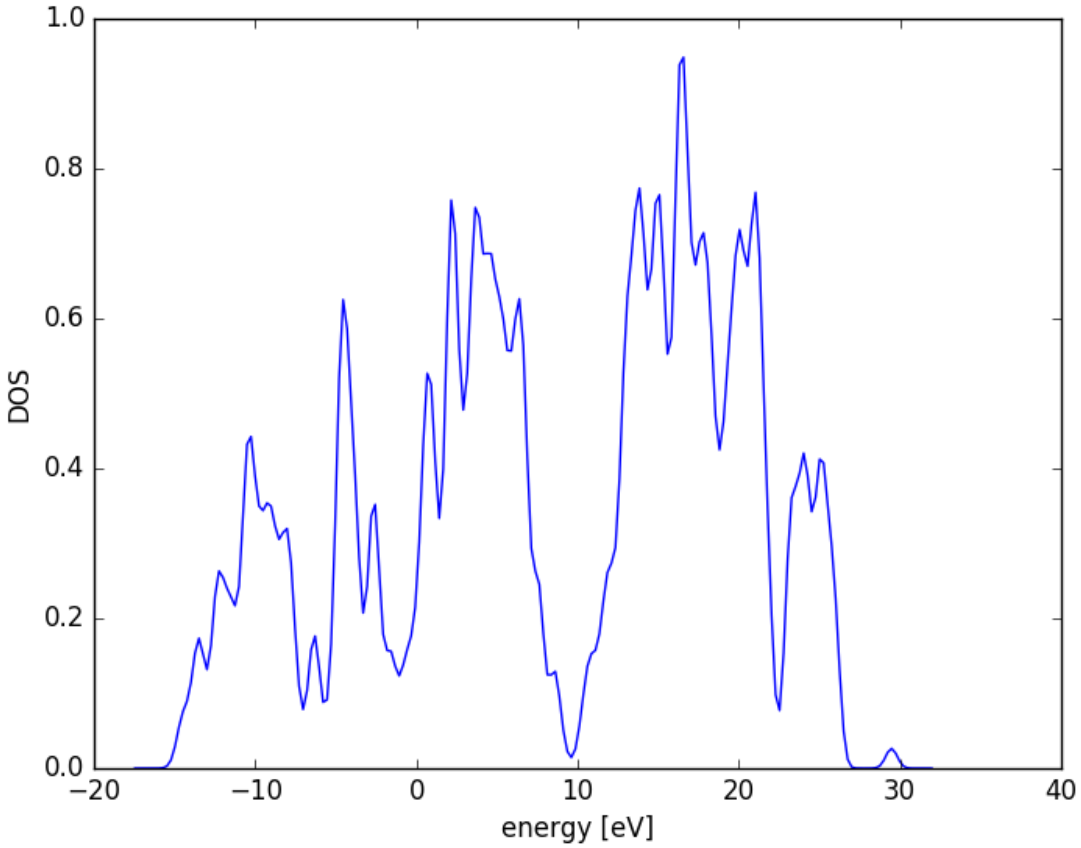
## Calculate and plot the density of states (DOS) of silicon (examples 3)

This example shows how to calculate the electronic density of states (DOS) with `postqe`. All necessary information is extracted from the standard xml output file. The following code shows how to do it for silicon (xml output file: Si.xml)

```
from postqe import get_dos

# get a DOS object
dos = get_dos(prefix='Si', schema='../../schemas/qes.xsd', width=0.5, npts=200)

# get the dos and energies for further processing
d = dos.get_dos()
e = dos.get_energies()

# Plot the DOS with Matplotlib...
import matplotlib.pyplot as plt
plt.plot(e, d)
plt.xlabel('energy [eV]')
plt.ylabel('DOS')
plt.savefig('figure.png')
plt.show()

# save DOS in a file
dos.write('DOS.out')
```

The `get_dos()` needs in input the xml file produced by pw.x, after a proper DOS calculation. This is identified using *label* which may contain the full path to the file (.xml is automatically added). The *schema* (optional) parameter allows the code to properly parse and validate the xml file. You must also specify the number of energy steps in the DOS (*npts*), plus the Gaussian broadening (*width*). `get_dos()` then returns a DOS object.

The DOS values and the corresponding energies can be obtained from the DOS object using the methods `get_dos()` and `get_energies()`. If you want you can further manipulate these values. For example you can make a plot with the Python library *Matplotlib*. The output plot looks like the following:




You can of course continue to calculate other quantities in your script.

The above results can be obtained from the command line typing:

```
$ postqe dos -prefix Si -schema ../../schemas/qes.xsd -npts 200 -width 0.5 -fileplot figure.png
```

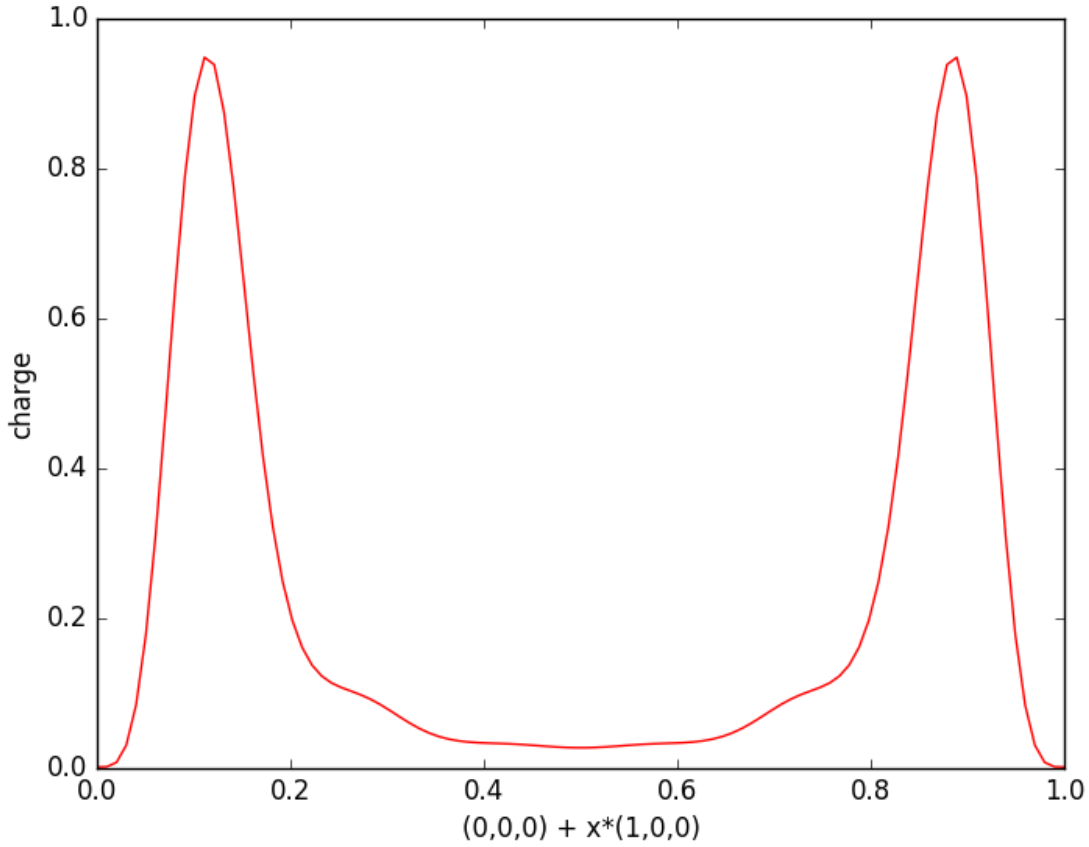
## Plotting a 1D section of the charge density (examples 4)

A common task you can perform with `postqe` is to plot the electronic charge density along one direction. The charge is read from the HDF5 output file create by the Quantum Espresso calculation in *outdir*. Additional information are extracted from the standard xml output file, identified by the *prefix* parameter in the `get_charge()` function. The *schema* (optional) parameter allows the code to properly parse and validate the xml file. The full code to do this is shown below:

```
from postqe import get_charge

charge = get_charge(prefix='Ni', schema='../../schemas/qes.xsd')
charge.write('outputcharge.dat')

figure = charge.plot(x0=(0, 0, 0), e1=(1, 0, 0), nx=100)
figure.savefig('figure_1.png')
```

The call to `get_charge()` creates a *charge* object. The charge can be written in a text file using the method `write()`. The call to the method `plot()` returns a *Matplotlib* figure object containing a 1D section plot of the

charge from the point *x0* along the direction *e1*. By default, the charge is plotted from the point (0,0,0) along the direction (1,0,0). The *Matplotlib* figure object can be further modified with the standard methods of this library. For example, the plot can be save in a png file using the method `savefig()`. The result is shown below.




The above results can be obtained from the command line typing:

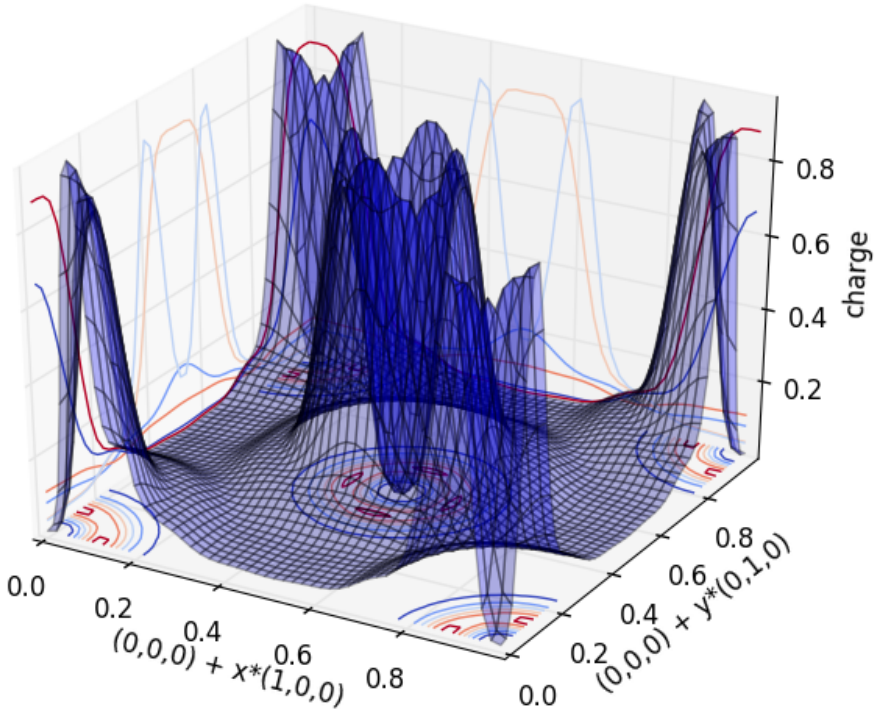
```
$ postqe charge -prefix Ni -schema ../../schemas/qes.xsd -fileout outputcharge.dat -x0 0,0,0 -e1 1,0,
```

## Plotting a 2D section of the charge density (examples 5)

This example is similar to the previous one except for producing a 2D plot of a planar section of the electronic charge density. The plane is defined by an initial point *x0* and two vectors, *e1* and *e2* which define the plane, which are given as parameters to the method `plot()` together with *dim=2* to define a 2D plot.

```
from postqe import get_charge

charge = get_charge(prefix='Ni', schema='../../schemas/qes.xsd')

figure = charge.plot(x0=(0, 0, 0), e1=(1, 0, 0), e2=(0, 1, 0), nx=50, ny=50, dim=2)
figure.savefig("figure_1.png")
figure.savefig("figure_1.pdf", format='pdf')
```

The resulting *Matplotlib* plot is




The above results can be obtained from the command line typing:

```
$ postqe charge -prefix Ni -schema ../../schemas/qes.xsd -fileout outputcharge.dat -dim 2 -x0 0,0,0 -
```

## Plotting 1D sections of different potentials (examples 6)

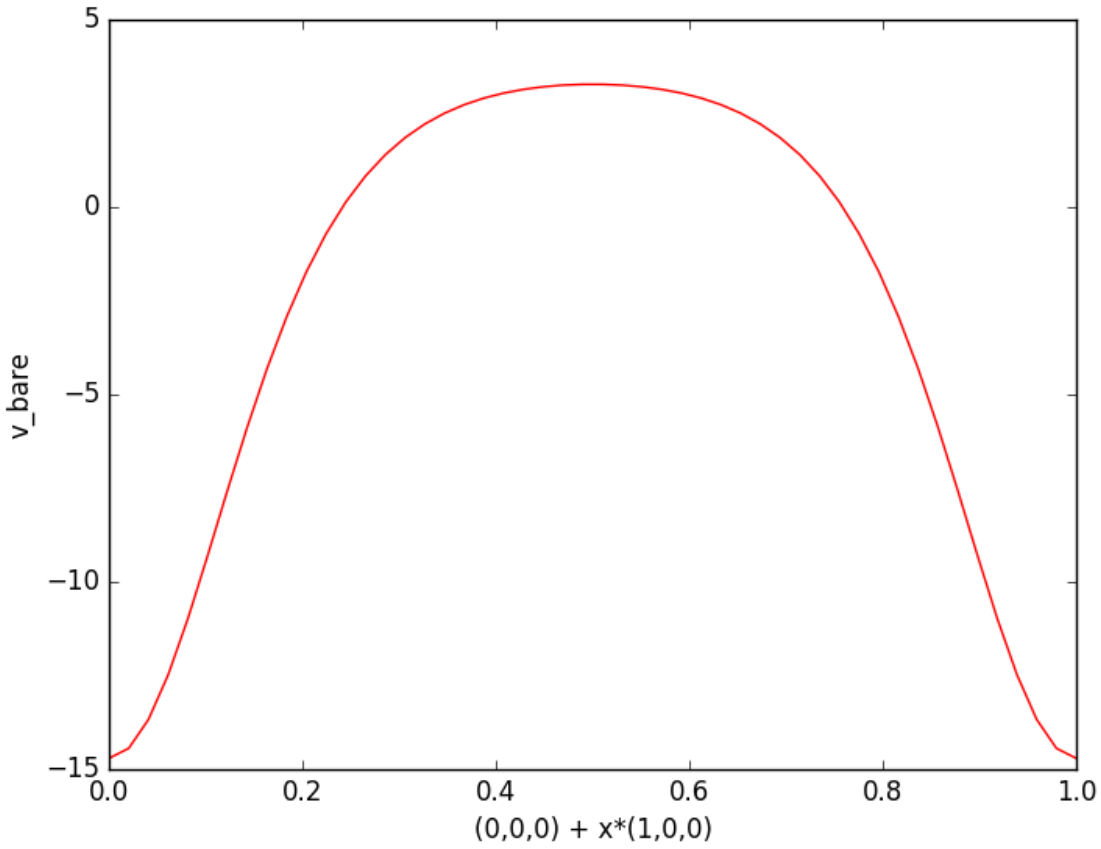
This example computes all the different potentials available, i.e. the bare potential $V_{bare}$, the Hartree potential $V_H$, the exchange-correlation potential $V_{xc}$ and the total potential $V_{tot} = V_{bare} + V_H + V_{xc}$. All necessary information is taken from the xml output file of QE and the HDF5 charge file. The pseudopotential file is also necessary to compute $V_{bare}$.

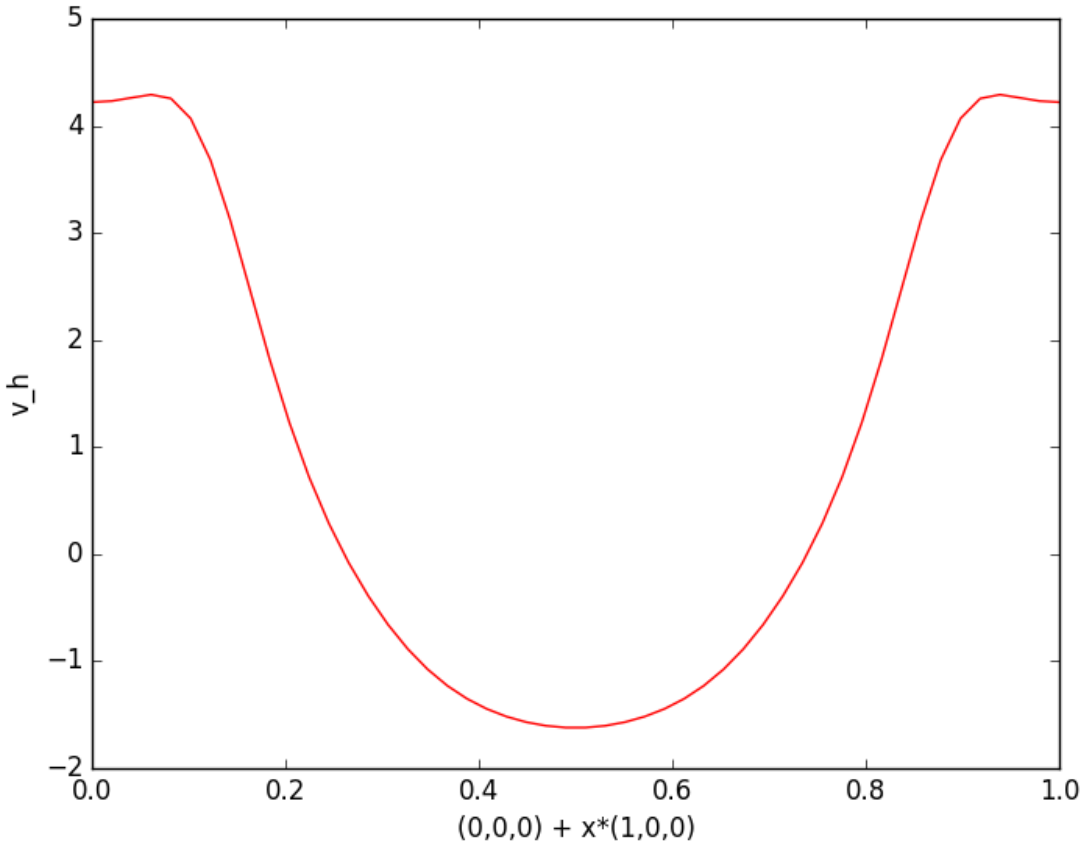
```
from postqe import get_potential

v_bare = get_potential(prefix='Ni', schema='../../schemas/qes.xsd', pot_type='v_bare')
v_bare.write('v_bare.dat')
fig1 = v_bare.plot(x0=(0, 0, 0), e1=(1, 0, 0), nx=50)
fig1.savefig("figure_v_bare.png")

v_h = get_potential(prefix='Ni', schema='../../schemas/qes.xsd', pot_type='v_h')
v_h.write('v_h.dat')
fig2 = v_h.plot(x0=(0, 0, 0), e1=(1, 0, 0), nx=50)
fig2.savefig("figure_v_h.png")

v_xc = get_potential(prefix='Ni', schema='../../schemas/qes.xsd', pot_type='v_xc')
```

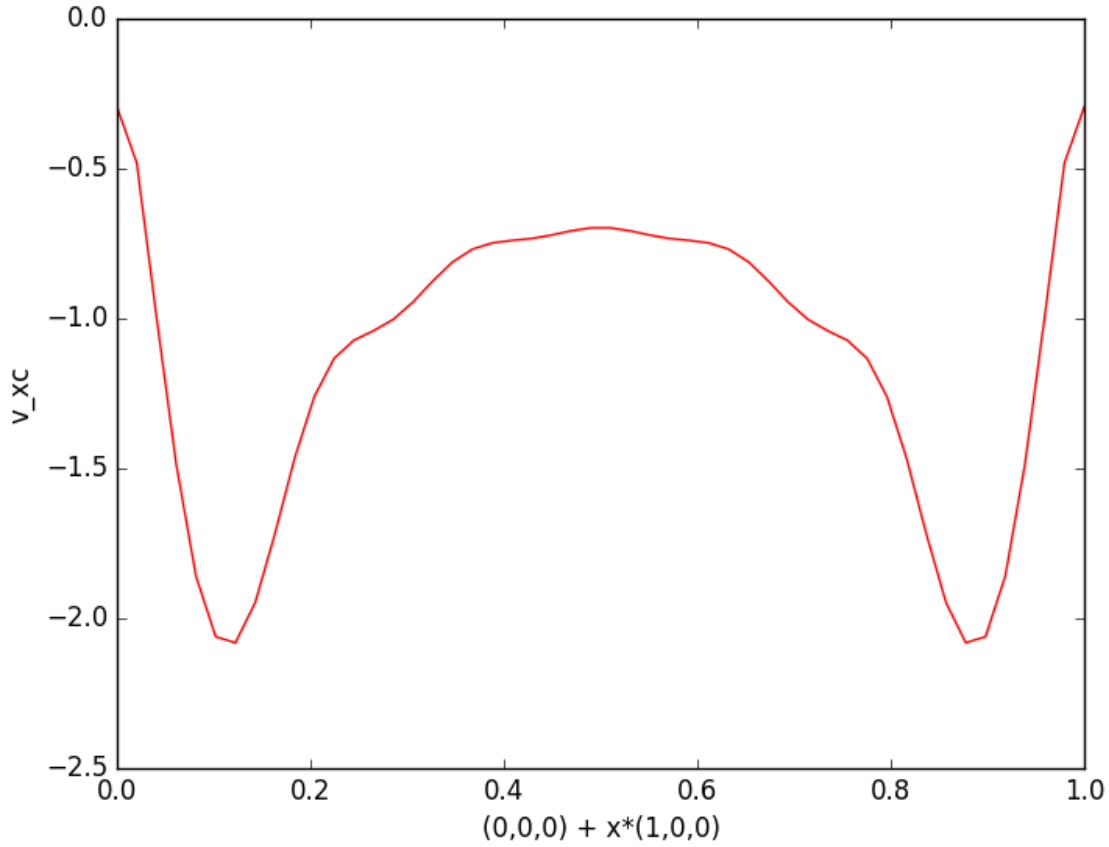
```
v_xc.write('v_xc.dat')
fig3 = v_xc.plot(x0=(0, 0, 0), e1=(1, 0, 0), nx=50)
fig3.savefig("figure_v_xc.png")

v_tot = get_potential(prefix='Ni', schema='../../schemas/qes.xsd', pot_type='v_tot')
v_tot.write('v_tot.dat')
fig4 = v_tot.plot(x0=(0, 0, 0), e1=(1, 0, 0), nx=50)
fig4.savefig("figure_v_tot.png")
```

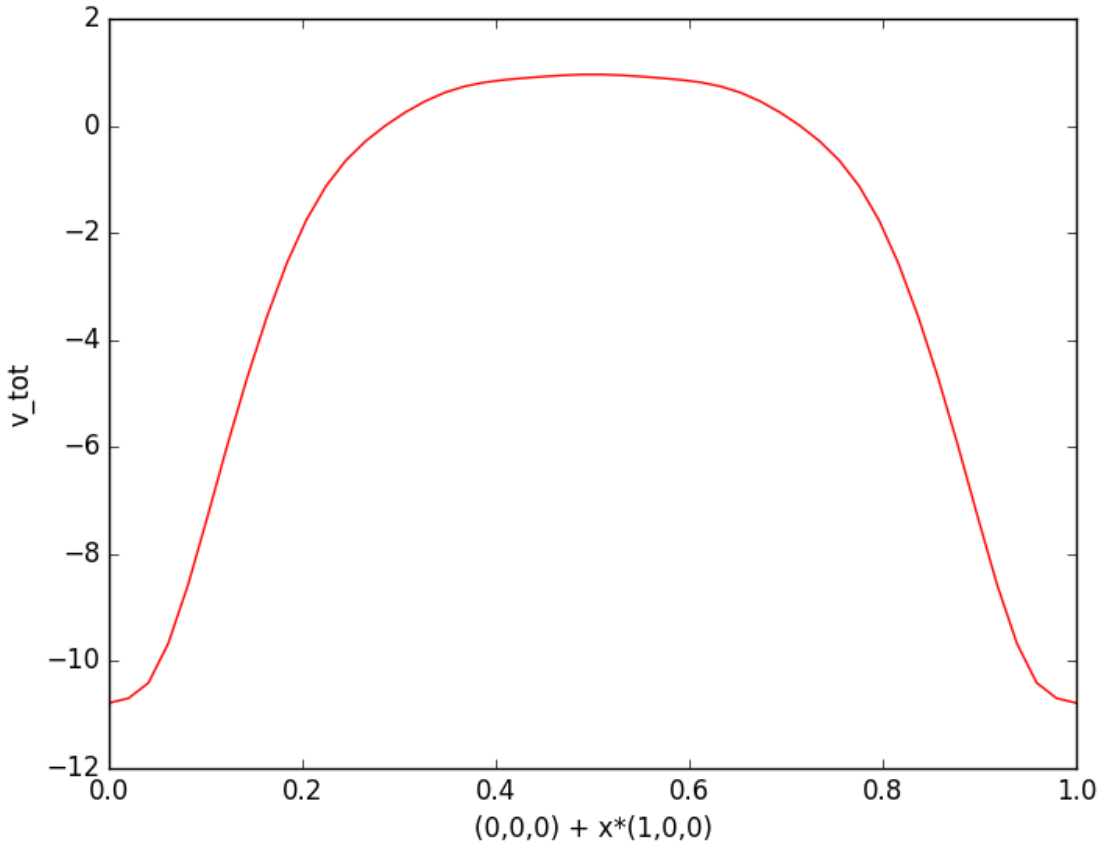
The code essentialy call the function `get_potential()`, which returns a potential object of the type defined in *pot_typ*. The `write()` and `plot()` methods are then used to write the output in a text file and produce the plots as in the previous example (in fact they accept the same parameters).

The output figures are as follows:




5
4
3
2
1
0
−1
−2
v_h
0.0
0.2
0.4
0.6
0.8
1.0
(0,0,0) + x*(1,0,0)


0.0
−0.5
−1.0
−1.5
−2.0
−2.5
v_xc
0.0
0.2
0.4
0.6
0.8
1.0
(0,0,0) + x*(1,0,0)




The above results can be obtained from the command line typing:

```
$ postqe potential -pot_type v_bare -prefix Ni -schema ../../schemas/qes.xsd -fileout v_bare.dat -x0
$ postqe potential -pot_type v_h -prefix Ni -schema ../../schemas/qes.xsd -fileout v_h.dat -x0 0,0,0
$ postqe potential -pot_type v_xc -prefix Ni -schema ../../schemas/qes.xsd -fileout v_xc.dat -x0 0,0,
$ postqe potential -pot_type v_tot -prefix Ni -schema ../../schemas/qes.xsd -fileout v_tot.dat -x0 0,
```

## Export formats (examples 7)

This is a (rather boring) example to illustrate the different formats available to export in a text file the charge extracted with `postqe`. The following code computes different 1, 2 and 3D sections of the charge and export the results in some available formats. These exported files can then be viewed using the corresponding package. Note that for 3D sections, no plot is generated by `postqe` and you can only export the charge in XSF (XCrySDen) and cube (Gaussian) formats to visualize it with these packages. Note that the number of points for the grids is small to make the calculations faster. For 2D and especially 3D sections, a significant time may be necessary for the calculations.

CHAPTER

THREE

# POSTQE PACKAGE

## Submodules

Additional functions are available as submodules. Please note the documentation of these functions is still ongoing and can be incomplete or wrong.

## postqe.api module

A collection of functions defining postqe API and exposed to the user.

postqe.api.**comput_dos**(*prefix*, *outdir=None*, *schema=None*, *width=0.01*, *window=None*, *npts=100*, *fileout=''*, *fileplot='dosplot.png'*, *show=True*)

This function returns an DOS object from an output xml Espresso file containing the results of a DOS calculation.

**Parameters**

- **prefix** – prefix of saved output files
- **outdir** – directory containing the input data. Default to the value of ESPRESSO_TMPDIR environment variable if set or current directory ('.') otherwise
- **schema** – the XML schema to be used to read and validate the XML output file
- **width** – width of the gaussian to be used for the DOS (in eV)
- **= emin, emax** (*window*) – defines the minimun and maximun energies for the DOS
- **npts** – number of points of the DOS
- **fileout** – output file with DOS results (default='', not written).
- **fileplot** – output plot file (default='dosplot') in png format.
- **show** – True -> plot results with Matplotlib; None or False -> do nothing. Default = True

**Returns** a DOS object and a Matplotlib figure object

postqe.api.**compute_band_structure**(*prefix*, *outdir=None*, *schema=None*, *reference_energy=0*, *emin=-50*, *emax=50*, *fileplot='bandsplot.png'*, *show=True*)

This function returns a "band structure" object from an output xml Espresso file containing the results of a proper calculation along a path in the Brilluoin zone.

**Parameters**

- **prefix** – prefix of saved output file

- **outdir** – directory containing the input data. Default to the value of ESPRESSO_TMPDIR environment variable if set or current directory ('.') otherwise
- **schema** – the XML schema to be used to read and validate the XML output file
- **reference_energy** – the Fermi level, defines the zero of the plot along y axis
- **emin** – the minimum energy for the band plot (default=-50)
- **emax** – the maximum energy for the band plot (default=50)
- **fileplot** – output plot file (default='bandsplot.png') in png format.
- **show** – True -> plot results with Matplotlib; None or False -> do nothing. Default = True

**Returns** an ASE band structure object and a Matplotlib figure object

postqe.api.**compute_charge**(*prefix*, *outdir=None*, *schema=None*, *fileout=''*, *x0=(0.0*, *0.0*, *0.0)*, *e1=(1.0*, *0.0*, *0.0)*, *nx=50*, *e2=(0.0*, *1.0*, *0.0)*, *ny=50*, *e3=(0.0*, *0.0*, *1.0)*, *nz=50*, *radius=1*, *dim=1*, *ifmagn='total'*, *plot_file=''*, *method='FFT'*, *format='gnuplot'*, *show=True*)

Returns an Charge object from an output xml Espresso file and the corresponding HDF5 charge file containing the results of a calculation. Returns also a Matplotlib figure object from a 1D or 2D section of the charge. It also (optionally) exports the charge (1, 2 or 3D section) in a text file according to different formats (XSF, cube, Gnuplot, etc.).

**Parameters**

- **prefix** – prefix of saved output file
- **outdir** – directory containing the input data. Default to the value of ESPRESSO_TMPDIR environment variable if set or current directory ('.') otherwise
- **schema** – the XML schema to be used to read and validate the XML output file
- **fileout** – text file with the full charge data as in the HDF5 file. Default='', nothing is written.
- **x0** – 3D vector (a tuple), origin of the line
- **e2, e3** (*e1,*) – 3D vectors (tuples) which determines the plotting lines
- **ny, nz** (*nx,*) – number of points along e1, e2, e3
- **radius** – radious of the sphere in the polar average method
- **dim** – 1, 2, 3 for a 1D, 2D or 3D section respectively
- **ifmagn** – for a magnetic calculation, 'total' plot the total charge, 'up' plot the charge with spin up, 'down' for spin down
- **plotfile** – file where plot data are exported in the chosen format (Gnuplot, XSF, cube Gaussian, etc.)
- **method** – interpolation method. Available choices are:

  'FFT' -> Fourier interpolation (default)

  'polar' -> 2D polar plot on a sphere

  'spherical' -> 1D plot of the spherical average

  'splines' -> not implemented

- **format** – format of the (optional) exported file. Available choices are:

  'gnuplot' -> plain text format for Gnuplot (default). Available for 1D and 2D sections.

'xsf' -> XSF format for the XCrySDen program. Available for 2D and 3D sections.

'cube' -> cube Gaussian format. Available for 3D sections.

'contour' -> format for the contour.x code of Quantum Espresso.

'plotrho' -> format for the plotrho.x code of Quantum Espresso.

- **show** – if True, show the Matplotlib plot (only for 1D and 2D sections)

**Returns** a Charge object and a Matplotlib figure object for 1D and 2D sections, a Charge object and None for 3D sections

postqe.api.**compute_eos**(*prefix*, *outdir=None*, *eos_type='murnaghan'*, *fileout=''*, *fileplot='EOSplot'*, *show=True*, *ax=None*)

Fits an Equation of state of type *eos_type*, writes the results into *fileout* (optionally) and creates a Matplotlib figure. Different equation of states are available (see below).

**Parameters**

- **prefix** – name of the input file with volumes and energies
- **outdir** – directory containing the input data. Default to the value of ESPRESSO_TMPDIR environment variable if set, or current directory ('.') otherwise
- **eos_type** – type of equation of state (EOS) for fitting. Available types are:

  'murnaghan' (default) -> Murnaghan EOS, PRB 28, 5480 (1983)

  'sjeos' -> A third order inverse polynomial fit, PhysRevB.67.026103

  E(V) = c_0 + c_1 t + c_2 t^2 + c_3 t^3 , t = V^(-1/3)

  'taylor' -> A third order Taylor series expansion around the minimum volume

  'vinet' -> Vinet EOS, PRB 70, 224107

  'birch' -> Birch EOS, Intermetallic compounds: Principles and Practice, Vol I: Principles, p. 195

  'birchmurnaghan' -> Birch-Murnaghan EOS, PRB 70, 224107

  'pouriertarantola' -> Pourier-Tarantola EOS, PRB 70, 224107

  'antonschmidt' -> Anton-Schmidt EOS, Intermetallics 11, 23 - 32(2003)

  'p3' -> A third order inverse polynomial fit
- **fileout** – output file with fitting data and results (default='', not written).
- **fileplot** – output plot file (default='EOSplot') in png format.
- **show** – True -> plot results with Matplotlib; None or False -> do nothing. Default = True
- **ax** – a Matplotlib "Axes" instance (see Matplotlib documentation for details). If ax=None (default), creates a new one

**Returns** an QEEquationOfState object and a Matplotlib figure object

postqe.api.**compute_potential**(*prefix*, *outdir=None*, *schema=None*, *pot_type='v_tot'*, *fileout=''*, *x0=(0.0, 0.0, 0.0)*, *e1=(1.0, 0.0, 0.0)*, *nx=50*, *e2=(0.0, 1.0, 0.0)*, *ny=50*, *e3=(0.0, 0.0, 1.0)*, *nz=50*, *radius=1*, *dim=1*, *plot_file=''*, *method='FFT'*, *format='gnuplot'*, *show=True*)

Returns an Potential object from an output xml Espresso file and the corresponding HDF5 charge file containing the results of a calculation. Returns also a Matplotlib figure object from a 1D or 2D section of the charge. It also (optionally) exports the charge (1, 2 or 3D section) in a text file according to different formats (XSF, cube, Gnuplot, etc.).

**Parameters**

- **prefix** – prefix of saved output file
- **outdir** – directory containing the input data. Default to the value of ESPRESSO_TMPDIR environment variable if set or current directory ('.') otherwise
- **schema** – the XML schema to be used to read and validate the XML output file
- **fileout** – text file with the calculate potential data. Default='', nothing is written.
- **x0** – 3D vector (a tuple), origin of the line
- **e2, e3** (*e1,*) – 3D vectors (tuples) which determines the plotting lines
- **ny, nz** (*nx,*) – number of points along e1, e2, e3
- **radius** – radious of the sphere in the polar average method
- **dim** – 1, 2, 3 for a 1D, 2D or 3D section respectively
- **plotfile** – file where plot data are exported in the chosen format (Gnuplot, XSF, cube Gaussian, etc.)
- **method** – interpolation method. Available choices are:

  'FFT' -> Fourier interpolation (default)

  'polar' -> 2D polar plot on a sphere

  'spherical' -> 1D plot of the spherical average

  'splines' -> not implemented
- **format** – format of the (optional) exported file. Available choices are:

  'gnuplot' -> plain text format for Gnuplot (default). Available for 1D and 2D sections.

  'xsf' -> XSF format for the XCrySDen program. Available for 2D and 3D sections.

  'cube' -> cube Gaussian format. Available for 3D sections.

  'contour' -> format for the contour.x code of Quantum Espresso.

  'plotrho' -> format for the plotrho.x code of Quantum Espresso.
- **show** – if True, show the Matplotlib plot (only for 1D and 2D sections)

**Returns** a Potential object and a Matplotlib figure object for 1D and 2D sections, a Potential object and None for 3D sections

postqe.api.**get_band_structure**(*prefix*, *outdir=None*, *schema=None*, *reference_energy=0*)

This function returns a "band structure" object from an output xml Espresso file containing the results of a proper calculation along a path in the Brilluoin zone.

**Parameters**

- **prefix** – prefix of saved output file
- **outdir** – directory containing the input data. Default to the value of ESPRESSO_TMPDIR environment variable if set or current directory ('.') otherwise
- **schema** – the XML schema to be used to read and validate the XML output file
- **reference_energy** – the Fermi level, defines the zero of the plot along y axis

**Returns** an ASE band structure object

postqe.api.**get_charge**(*prefix*, *outdir=None*, *schema=None*)

Returns an Charge object from an output xml Espresso file and the corresponding HDF5 charge file containing the results of a calculation.

**Parameters**

- **prefix** – prefix of saved output file
- **outdir** – directory containing the input data. Default to the value of ESPRESSO_TMPDIR environment variable if set or current directory ('.') otherwise
- **schema** – the XML schema to be used to read and validate the XML output file

**Returns** a Charge object

postqe.api.**get_dos**(*prefix*, *outdir=None*, *schema=None*, *width=0.01*, *window=None*, *npts=100*)

This function returns an DOS object from an output xml Espresso file containing the results of a DOS calculation.

**Parameters**

- **prefix** – prefix of saved output file
- **outdir** – directory containing the input data. Default to the value of ESPRESSO_TMPDIR environment variable if set or current directory ('.') otherwise
- **schema** – the XML schema to be used to read and validate the XML output file
- **width** – width of the gaussian to be used for the DOS (in eV)
- **= emin, emax** (*window*) – defines the minimun and maximun energies for the DOS
- **npts** – number of points of the DOS

**Returns** a DOS object

postqe.api.**get_eos**(*prefix*, *outdir=None*, *eos_type='murnaghan'*)

Fits an Equation of state of type *eos* and returns an QEEquationOfState object. Different equation of states are available (see below).

**Parameters**

- **prefix** – name of the input file with volumes and energies
- **outdir** – directory containing the input data. Default to the value of ESPRESSO_TMPDIR environment variable if set, or current directory ('.') otherwise
- **eos_type** – type of equation of state (EOS) for fitting. Available types are:

  'murnaghan' (default) -> Murnaghan EOS, PRB 28, 5480 (1983)

  'sjeos' -> A third order inverse polynomial fit, PhysRevB.67.026103

  E(V) = c_0 + c_1 t + c_2 t^2 + c_3 t^3 , t = V^(-1/3)

  'taylor' -> A third order Taylor series expansion around the minimum volume

  'vinet' -> Vinet EOS, PRB 70, 224107

  'birch' -> Birch EOS, Intermetallic compounds: Principles and Practice, Vol I: Principles, p. 195

  'birchmurnaghan' -> Birch-Murnaghan EOS, PRB 70, 224107

  'pouriertarantola' -> Pourier-Tarantola EOS, PRB 70, 224107

  'antonschmidt' -> Anton-Schmidt EOS, Intermetallics 11, 23 - 32(2003)

'p3' -> A third order inverse polynomial fit

**Returns** an QEEquationOfState object

`postqe.api.`**`get_label`**(*prefix*, *outdir=None*)

`postqe.api.`**`get_potential`**(*prefix*, *outdir=None*, *schema=None*, *pot_type='v_tot'*)

This function returns an Potential object from an output xml Espresso file and the corresponding HDF5 charge file containing the results of a calculation. The available potentials are the bare (pot_type='v_bare'), Hartree (pot_type='v_h'), exchange-correlation (pot_type='v_xc') and total (pot_type='v_tot').

**Parameters**

- **`prefix`** – prefix of saved output files
- **`outdir`** – directory containing the input data. Default to the value of ESPRESSO_TMPDIR environment variable if set or current directory ('.') otherwise
- **`schema`** – the XML schema to be used to read and validate the XML output file
- **`pot_type`** – type of the Potential ('v_tot', ....)

**Returns** a Potential object

## postqe.eos module

A specialization of ASE EquationOfState class with a modified default EOS type and a write method.

**class** `postqe.eos.`**`QEEquationOfState`**(*volumes*, *energies*, *eos='murnaghan'*)

Bases: `ase.eos.EquationOfState`

**`write`**(*filename='eos.out'*)

`postqe.eos.`**`create_header`**(*eos*, *v0*, *e0*, *B*)

## postqe.dos module

A specialization of ASE DOS class with a new get_dos_int method for computing the integral of the DOS and a modified write method to write it properly.

**class** `postqe.dos.`**`QEDOS`**(*calc*, *width=0.1*, *window=None*, *npts=201*)

Bases: `ase.dft.dos.DOS`

**`get_dos_int`**()

Get array of integral DOS values (always for the total DOS).

**`write`**(*filename='dos.out'*)

## postqe.charge module

**class** `postqe.charge.`**`Charge`**(**args*, ***kwargs*)

Bases: `object`

A class for charge density.

**plot**(*x0=(0.0, 0.0, 0.0)*, *e1=(1.0, 0.0, 0.0)*, *nx=50*, *e2=(0.0, 1.0, 0.0)*, *ny=50*, *e3=(0.0, 0.0, 1.0)*, *nz=50*, *radius=1*, *dim=1*, *ifmagn='total'*, *plot_file=''*, *method='FFT'*, *format='gnuplot'*, *show=True*)

Plot a 1D, 2D or 3D section of the charge from x0 along e1 (e2, e3) direction(s) using Fourier interpolation or another method (see below). For 1D or 2D sections, the code produce a Matplotlib plot. For a 3D plot, the charge must be exported in 'plotfile' with a suitable format ('xsf' or 'cube') and can be visualized with the corresponding external codes.

**Parameters**

- **x0** – 3D vector (a tuple), origin of the line
- **e2, e3** (*e1,*) – 3D vectors (tuples) which determines the plotting lines
- **ny, nz** (*nx,*) – number of points along e1, e2, e3
- **radius** – radius of the sphere in the polar average method
- **dim** – 1, 2, 3 for a 1D, 2D or 3D section respectively
- **ifmagn** – for a magnetic calculation, 'total' plot the total charge, 'up' plot the charge with spin up, 'down' for spin down
- **plotfile** – file where plot data are exported in the chosen format (Gnuplot, XSF, cube Gaussian, etc.)
- **method** – interpolation method. Available choices are:

  'FFT' -> Fourier interpolation (default) 'polar' -> 2D polar plot on a sphere 'spherical' -> 1D plot of the spherical average 'splines' -> not implemented
- **format** – format of the (optional) exported file. Available choices are:

  'gnuplot' -> plain text format for Gnuplot (default). Available for 1D and 2D sections. 'xsf' -> XSF format for the XCrySDen program. Available for 2D and 3D sections. 'cube' -> cube Gaussian format. Available for 3D sections. 'contour' -> format for the contour.x code of Quantum Espresso 'plotrho' -> format for the plotrho.x code of Quantum Espresso
- **show** – if True, show the Matplotlib plot (only for 1D and 2D sections)

**Returns** a Matplotlib figure object for 1D and 2D sections, None for 3D sections

**read**(*filename*, *nr=None*)

Read the charge from a HDF5 file.

**Parameters**

- **filename** – HDF5 file with charge data
- **nr** – a numpy array or list of length 3 containing the grid dimensions

**set_calculator**(*calculator*)

**setvars**(*nr_temp*, *charge=None*, *charge_diff=None*)

**write**(*filename*)

**class** postqe.charge.**Potential**(**args*, *pot_type='v_tot'*, ***kwargs*)

Bases: *postqe.charge.Charge*

A class for a potential. This is derived from a Charge class and additionally contains the potential.

**compute_potential**()

Compute the potential from the electronic charge. The type of potential is defined in self.pot_type when an instance of the class Potential is create (default 'v_tot').

**plot** (*x0=(0.0, 0.0, 0.0), e1=(1.0, 0.0, 0.0), nx=50, e2=(0.0, 1.0, 0.0), ny=50, e3=(0.0, 0.0, 1.0), nz=50, radius=1, dim=1, plot_file='', method='FFT', format='gnuplot', show=True*)

Plot a 1D, 2D or 3D section of the potential from x0 along e1 (e2, e3) direction(s) using Fourier interpolation or another method (see below). For 1D or 2D sections, the code produce a Matplotlib plot. For a 3D plot, the charge must be exported in 'plotfile' with a suitable format ('xsf' or 'cube') and can be visualized with the corresponding external codes.

**Parameters**

- **x0** – 3D vector (a tuple), origin of the line
- **e2, e3** (*e1,*) – 3D vectors (tuples) which determines the plotting lines
- **ny, nz** (*nx,*) – number of points along e1, e2, e3
- **radius** – radius of the sphere in the polar average method
- **dim** – 1, 2, 3 for a 1D, 2D or 3D section respectively
- **plotfile** – file where plot data are exported in the chosen format (Gnuplot, XSF, cube Gaussian, etc.)
- **method** – interpolation method. Available choices are:

  'FFT' -> Fourier interpolation (default) 'polar' -> 2D polar plot on a sphere 'spherical' -> 1D plot of the spherical average 'splines' -> not implemented
- **format** – format of the (optional) exported file. Available choices are:

  'gnuplot' -> plain text format for Gnuplot (default). Available for 1D and 2D sections. 'xsf' -> XSF format for the XCrySDen program. Available for 2D and 3D sections. 'cube' -> cube Gaussian format. Available for 3D sections. 'contour' -> format for the contour.x code of Quantum Espresso 'plotrho' -> format for the plotrho.x code of Quantum Espresso
- **show** – if True, show the Matplotlib plot (only for 1D and 2D sections)

**Returns** a Matplotlib figure object for 1D and 2D sections, None for 3D sections

**write** (*filename*)

Write the potential in a text file. The potential must have been calculated before. :param filename: name of the output file

postqe.charge.**read_charge_file_hdf5** (*filename*, *nr=None*)

Reads a charge file written with QE in HDF5 format. *nr = [nr1,nr2,nr3]* (the dimensions of the charge k-points grid) are given as parameter (taken for the xml output file by the caller).

Notes: In the new format, the values of the charge in the reciprocal space are stored. Besides, only the values of the charge > cutoff are stored, together with the Miller indexes. Hence

postqe.charge.**write_charge** (*filename*, *charge*, *header*)

Write the charge or another quantity calculated by postqe into a text file *filename*.

CHAPTER

FOUR

# INDICES AND TABLES

- genindex
- modindex
- search

## p

postqe.api, 17
postqe.charge, 22
postqe.dos, 22
postqe.eos, 22

## C

Charge (class in postqe.charge), 22
comput_dos() (in module postqe.api), 17
compute_band_structure() (in module postqe.api), 17
compute_charge() (in module postqe.api), 18
compute_eos() (in module postqe.api), 19
compute_potential() (in module postqe.api), 19
compute_potential() (postqe.charge.Potential method), 23
create_header() (in module postqe.eos), 22

## G

get_band_structure() (in module postqe.api), 20
get_charge() (in module postqe.api), 20
get_dos() (in module postqe.api), 21
get_dos_int() (postqe.dos.QEDOS method), 22
get_eos() (in module postqe.api), 21
get_label() (in module postqe.api), 22
get_potential() (in module postqe.api), 22

## P

plot() (postqe.charge.Charge method), 22
plot() (postqe.charge.Potential method), 23
postqe.api (module), 17
postqe.charge (module), 22
postqe.dos (module), 22
postqe.eos (module), 22
Potential (class in postqe.charge), 23

## Q

QEDOS (class in postqe.dos), 22
QEEquationOfState (class in postqe.eos), 22

## R

read() (postqe.charge.Charge method), 23
read_charge_file_hdf5() (in module postqe.charge), 24

## S

set_calculator() (postqe.charge.Charge method), 23
setvars() (postqe.charge.Charge method), 23

## W

write() (postqe.charge.Charge method), 23
write() (postqe.charge.Potential method), 24
write() (postqe.dos.QEDOS method), 22
write() (postqe.eos.QEEquationOfState method), 22
write_charge() (in module postqe.charge), 24