# postqe Documentation

***Release 0.1***

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

July 19, 2017

Contents:

# CHAPTER ONE

# INTRODUCTION

`postqe` is a Python package to perform postprocessing calculations for results obtained with the Quantum Espresso (QE) code [1]. The package provides Python functions to post-process the results, such as plotting the charge density (or the bare/Hartree/total potentials) on 1D or 2D sections, fitting the total energy with Murnaghan Equation of State (EOS), etc. The package also exposes some QE functionalities in Python using the F2PY code [2] and wrappers to generate Python modules from QE dynamically linked libraries.

It is meant to be imported in your own Python code or used from the command line (see the Tutorial part of this documentation). 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 and matplotlib libraries.

Current features of the package include:

- Plot 1D or 2D sections of the charge density
- Plot 1D or 2D sections of the bare, Hartree or total potential
- Fit the total energy $E_{tot}(V)$ with Murnaghan's equation of state

## 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 common user are directly accessible from the `postqe`. You can import all of them as:

```
from postqe import *
```

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.

[1] http://www.quantum-espresso.org/
[2] https://docs.scipy.org/doc/numpy-dev/f2py/

## General notes

### Plotting

`postqe` uses the *matplotlib* library for Plotting. Some functions in the package are simply useful wrappers of *matplotlib* functions for common uses. They return a *matplotlib* object which can be further adapted to specific needs and personal taste.

# 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 intepreter or as scripts. The tutorial is based on the following examples:

| Example n. | Description |
|---|---|
| 1 | Plotting a 1D section of the charge density |
| 2 | Plotting a 2D section of the charge density |
| 3 | Fitting $E_{tot}(V)$ for a cubic (isotropic) system using Murnaghan EOS |

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 taylor your plot.

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

The most common task you can do with `postqe` is probably to plot the electronic charge density along one direction. The charge is read from the HSF5 output file create by the Quantum Espresso calculation in *outdir*. The code to do this is shown below:

```
from postqe import plot_charge1D

fin = "./Ni.xml"                                  # file xml produce by QE
plot_charge1D(fin)                            # plot a 1D section of the charge

#fig1 = plot_EV(V,E,a)                          # plot the E(V) data and the fitting line
#fig1.savefig("figure_1.png")
```

and it is essentially a call to the function `plot_charge1D()`, which needs in input the xml file create by Quantum Espresso. All other values are optional and taken either from the xml file or from

default values. By default, the charge is plotted from the point (0,0,0) along the direction (1,0,0).

../examples/example2/figure_1.png

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

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 and two 3D vectors which define the plane.

```
from postqe import plot_charge2D

fin = "./Ni.xml"                                   # file xml produce by QE
plot_charge2D(fin)                             # plot a 1D section of the charge

#fig1 = plot_EV(V,E,a)                           # plot the E(V) data and the fitting line
#fig1.savefig("figure_1.png")
```

As in the previous example, it is essentially a call to a single function, which is in this case `plot_charge1D()`. The output figure is:

../examples/example2/figure_1.png

## Fitting the total energy using Murnaghan EOS (examples 3)

The simplest task you can do with `postqe` is to fit the total energy as a function of volume $E_{tot}(V)$ (example3). You can use an equation of state (EOS) such as Murnaghan's or similar. Currently the Murnaghan EOS and quadratic and quartic polynomials are implemented in `postqe`.

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).

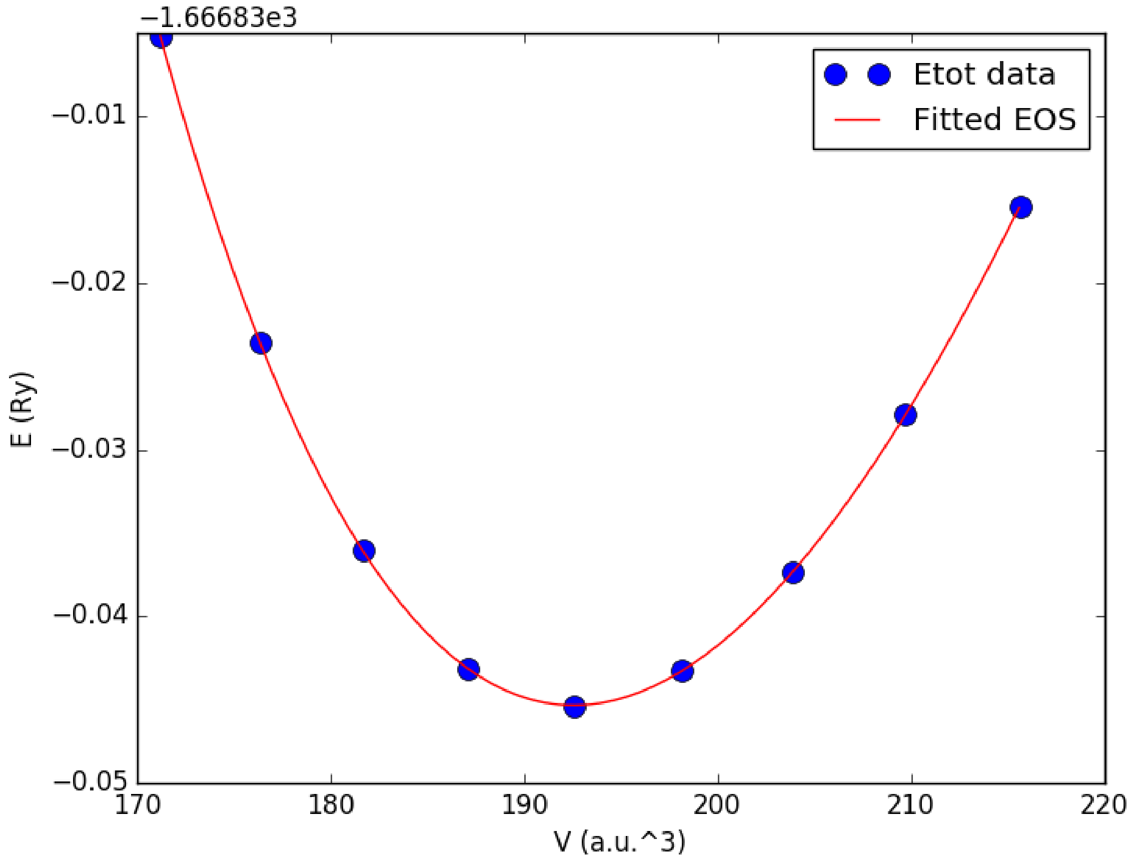
```
from postqe import fitEtotV, plot_EV

fin = "./EtotV.dat"                    # file with the total energy data E(V)
V, E, a, chi2 = fitEtotV(fin)              # fits the E(V) data, returns the coefficients a and
                                         # the chi squared chi2

fig1 = plot_EV(V,E,a)                          # plot the E(V) data and the fitting line
fig1.savefig("figure_1.png")
```

The `fitEtotV()` needs in input a file with two columns: the first with the volumes (in $a.u.^3$), the second with energies (in $Ryd/cell$). It returns the volumes $V$ and energies $E$ from the input file plus the fitting coefficients *a* and the $\chi^2$ *chi*. The fitting results are also written in details on the *stdout*:

```
# Murnaghan EOS                   chi squared= 6.3052908120e-09
# Etotmin= -1.6668753461e+03 Ry          Vmin= 1.9256068649e+02 a.u.^3          B0= 3.9507640525e+03 kb
################################################################################
# V *a.u.^3)                    Etot   (Ry)                 Etotfit   (Ry)                 Etot-Etotfit
1.7119697047e+02          -1.6668351807e+03         -1.6668351587e+03         -2.1971172146e-05
1.7637989181e+02          -1.6668536038e+03         -1.6668536431e+03         3.9227047637e-05
1.8166637877e+02          -1.6668660570e+03         -1.6668660709e+03         1.3986418708e-05
```

```
1.8705745588e+02          -1.6668731355e+03          -1.6668731117e+03          -2.3822185085e-05
1.9255414767e+02          -1.6668753764e+03          -1.6668753461e+03          -3.0392647432e-05
1.9815747866e+02          -1.6668732871e+03          -1.6668732784e+03          -8.7825126229e-06
2.0386847338e+02          -1.6668673220e+03          -1.6668673472e+03          2.5205460133e-05
2.0968815635e+02          -1.6668579007e+03          -1.6668579345e+03          3.3793803595e-05
2.1561755211e+02          -1.6668454001e+03          -1.6668453729e+03          -2.7248831202e-05
```

Optionally, you can plot the results with the `plot_EV()`. The original data are represented as points. If *a!=None*, a line with the fitting EOS will also be plotted. The output plot looks like the following:



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.constants module

## postqe.eos module

## postqe.plot module

## postqe.readutils module

CHAPTER

# FOUR

# INDICES AND TABLES

- genindex
- modindex
- search