postqe Documentation

Release 0.1

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

July 24, 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:

- Fit the total energy $E_{tot}(V)$ with Murnaghan's equation of state
- Calculate and plot the electronic band structure
- Calculate and plot the electronic density of states (DOS)
- Plot 1D or 2D sections of the charge density
- Plot 1D or 2D 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 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 for *matplotlib* functionalities of 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	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 and nickel
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 taylor 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 $E_{tot}(V)$ (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`.

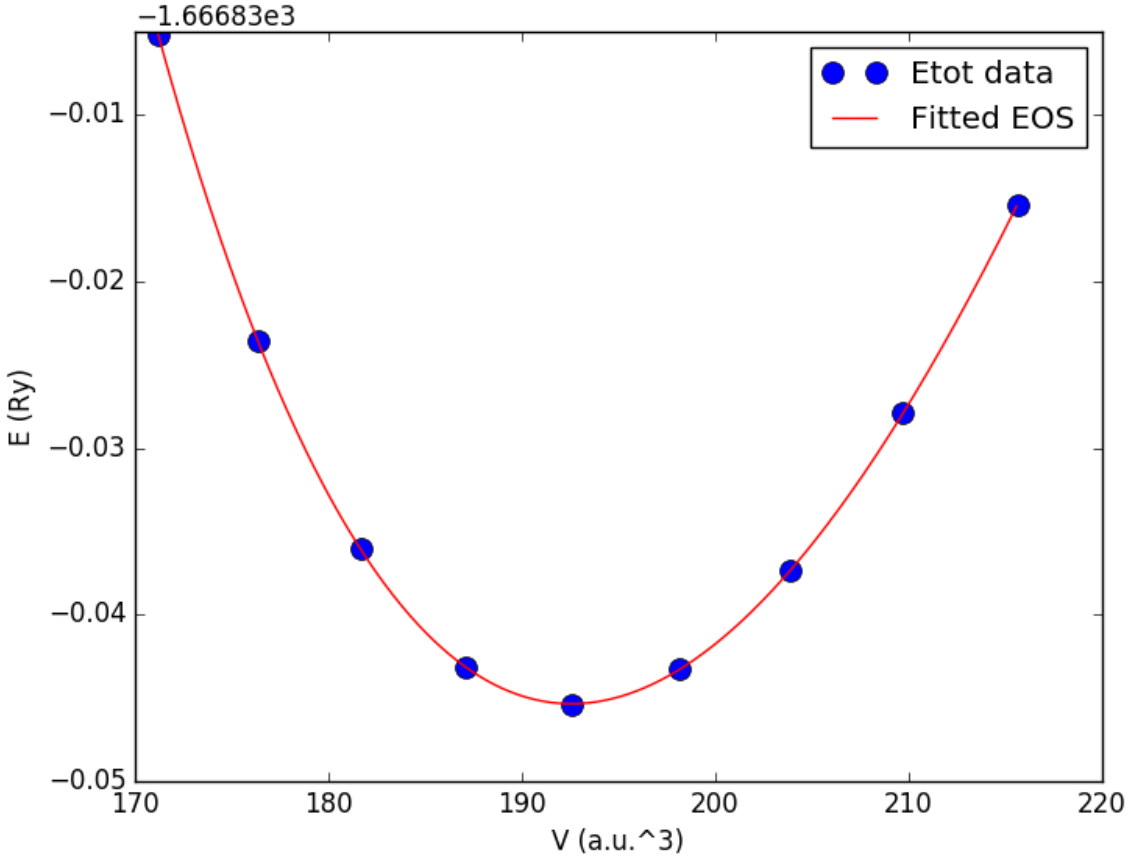
Let's see how to fit $E_{tot}(V)$. This is the case of isotropic cubic systems (simple cubic, body centered cubic, face centered cubic) or systems which can be approximated as isotropic (for example an hexagonal system with nearly constant c/a ratio).

```
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")                     # save the matplotlib figure
```

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 χ^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:



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

TODO

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

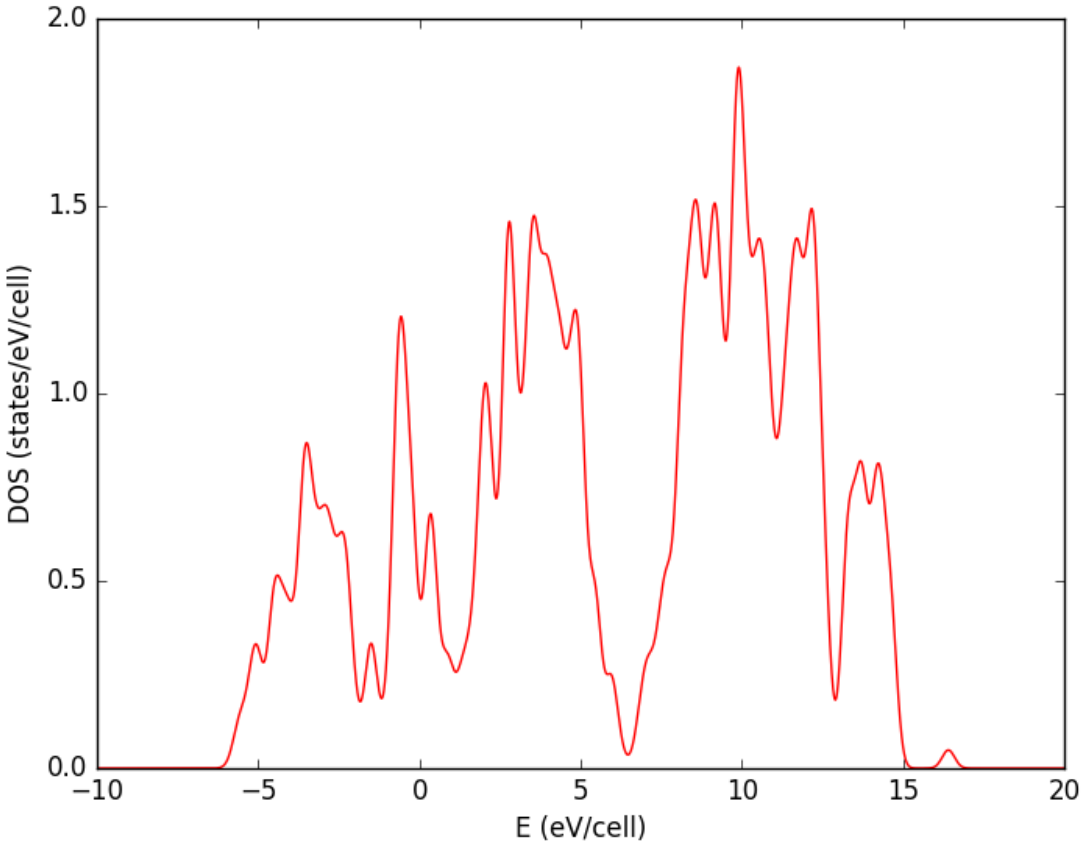
This example shows how to calculate the electronic density of states (DOS) with `postqe`. All necessary information is extracted from the standard xml output file. The following code shows how to do it for silicon (xml output file: Si.xml)

```
from postqe import compute_dos, simple_plot_xy, multiple_plot_xy

e, dos_up, dos_down = compute_dos('Si.xml', filedos='filedosSi', e_min=-10, e_max=20,
                                  e_step=0.01, degauss=0.02, ngauss=0)

# plot the DOS
fig1 = simple_plot_xy(e,dos_up,xlabel="E (eV/cell)",ylabel="DOS (states/eV/cell)")
fig1.savefig("figure_Sidos.png")

e2, dos_up2, dos_down2 = compute_dos('Ni.xml', filedos='filedosNi', e_min=0, e_max=50,
                                  e_step=0.01, degauss=0.02, ngauss=0)
```

The `compute_dos()` needs in input the xml file produced by pw.x. You must also specify the range of energies (and step) for which the DOS will be calculate (*e_min*, *e_max*, *e_step*), plus the type of Gaussian broadening (*ngauss*) and the value (*degauss*). The DOS values (and corresponding energies) are returned by the function in *e, dos_up, dos_down*. If you want to write the DOS values on a file, you must give it in the parameter *filedos* and the file will be like:

```
#  E                  dos up              dos down
#  (eV)               (states/eV/cell)    (states/eV/cell)
  -1.000000000E+01    4.590900107E-86     0.000000000E+00
  -9.990000000E+00    4.590900107E-86     0.000000000E+00
  -9.980000000E+00    4.590900107E-86     0.000000000E+00
  -9.970000000E+00    4.590900107E-86     0.000000000E+00
  -9.960000000E+00    4.590900107E-86     0.000000000E+00
  -9.950000000E+00    4.590900107E-86     0.000000000E+00
  -9.940000000E+00    4.590900107E-86     0.000000000E+00
  -9.930000000E+00    4.590900107E-86     0.000000000E+00
```

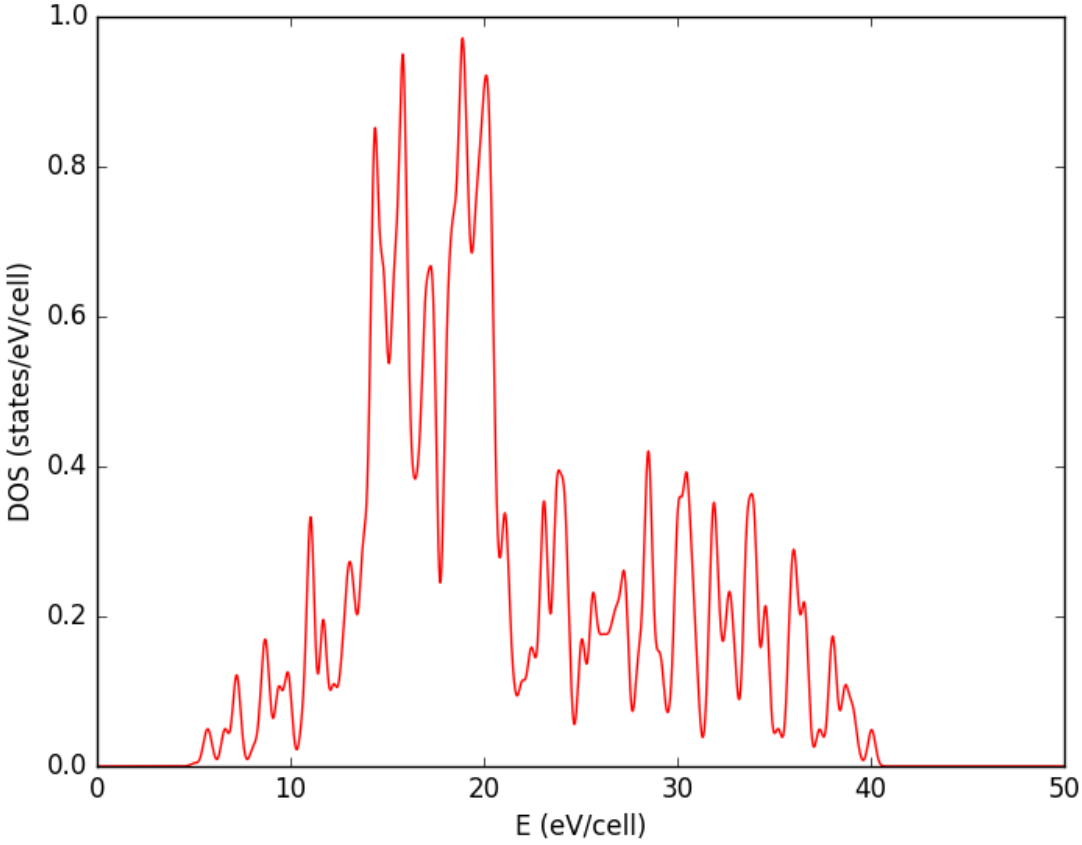
The first column contains the energy values, the second one contains the DOS values. Since silicon is non magnetic, the third column contains zero values. Optionally, you can plot the results with the `simple_plot_xy()`, which is simply a wrapper to matplotlib. The output plot looks like the following:

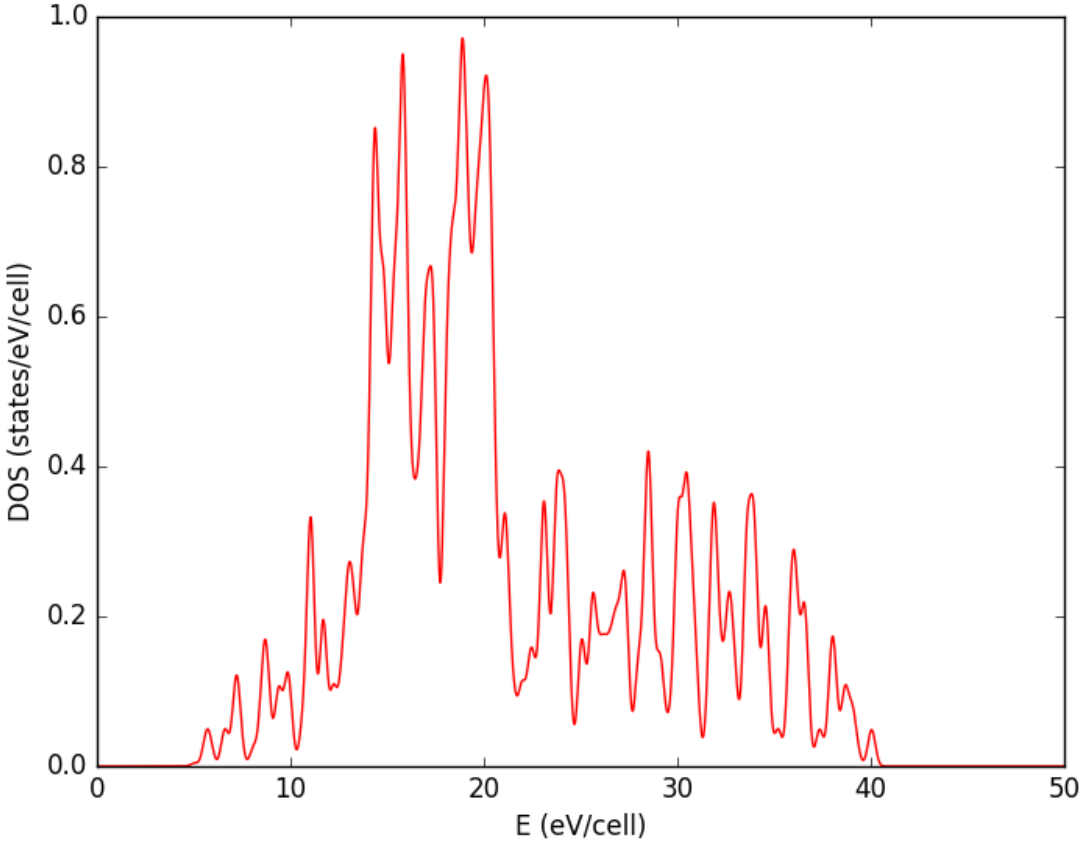


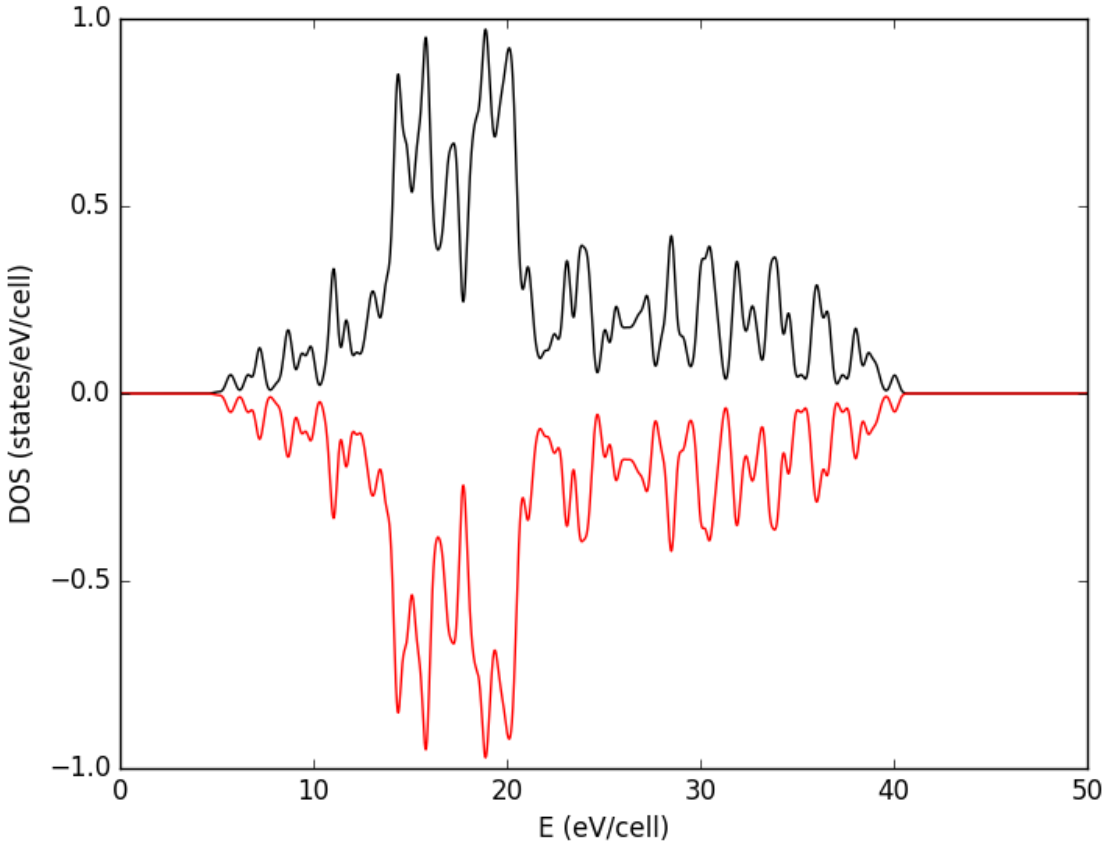
You can of course continue to calculate other quantities in your script. For example, the following lines show how to calculate the DOS for (magnetic) nickel.

```
# plot the DOS
fig2 = simple_plot_xy(e2,dos_up2,xlabel="E (eV/cell)",ylabel="DOS (states/eV/cell)")
fig2.savefig("figure_Nidosup.png")
fig3 = simple_plot_xy(e2,dos_down2,xlabel="E (eV/cell)",ylabel="DOS (states/eV/cell)")
fig3.savefig("figure_Nidosdown.png")

# create a numpy matrix *doss* for plotting both spin up and down on the same plot
import numpy as np
doss = np.zeros((len(e2),2))
doss[:,0] = dos_up2
doss[:,1] = -dos_down2
fig4 = multiple_plot_xy(e2,doss,xlabel="E (eV/cell)",ylabel="DOS (states/eV/cell)")
fig4.savefig("figure_Nidosupanddown.png")
```

The above lines compute the electronic DOS for nickel with spin up and down and are rather selfexplaining. The last lines put together in a numpy matrix the dos for spin up and down to show it in the same plot using the *matplotlib* wrapper `multiple_plot_xy()`. The output plots look like the following:


1.0
0.8
0.6
0.4
0.2
0.0
DOS (states/eV/cell)
0
10
20
30
40
50
E (eV/cell)


1.0
0.8
0.6
0.4
0.2
0.0
DOS (states/eV/cell)
0
10
20
30
40
50
E (eV/cell)

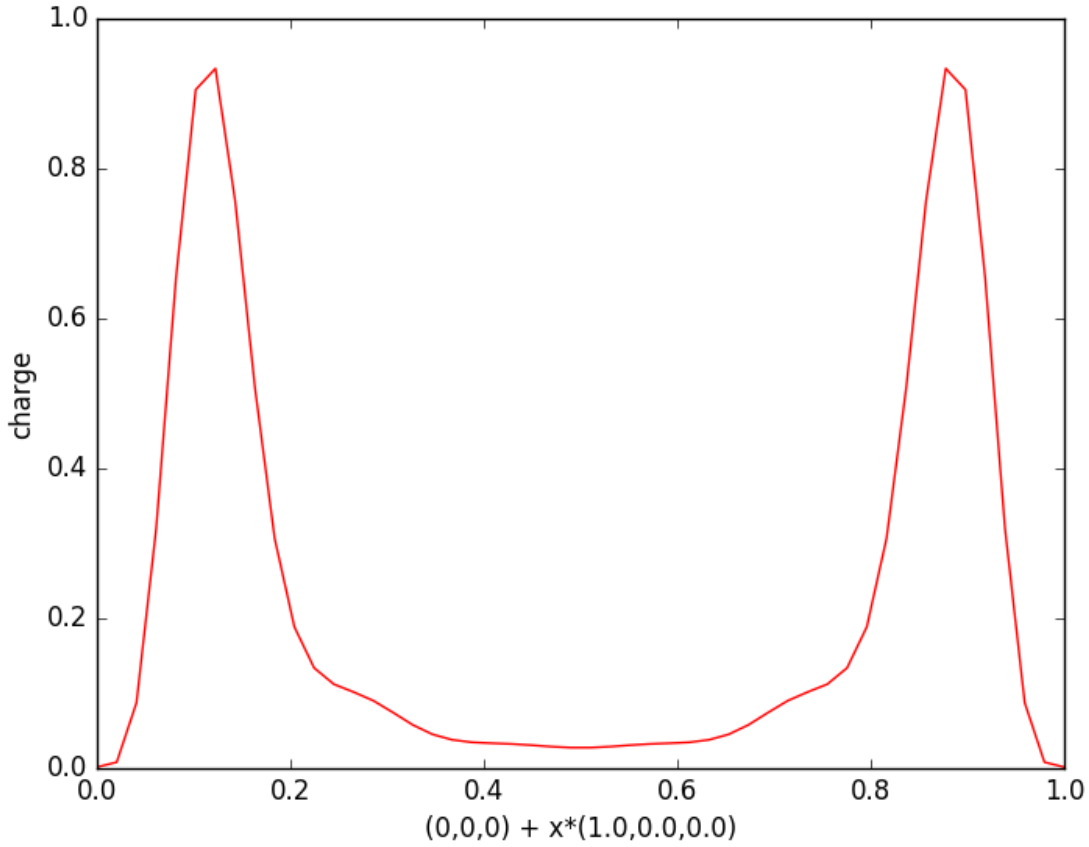


Plotting a 1D section of the charge density (examples 4)

A common task you can do 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. The code to do this is shown below:

```
from postqe import get_charge, get_cell_data, compute_G, plot1D_FFTinterp

fin = "./Ni.xml"                                              # file xml produce by QE
ibrav, alat, a, b, nat, ntyp,\
atomic_positions, atomic_species = get_cell_data(fin)  # get some data on the unit cell

charge, chargediff = get_charge(fin)              # get the charge (and charge diff)

G = compute_G(b, charge.shape)
fig1 = plot1D_FFTinterp(charge, G, a, x0=(0, 0, 0), e1=(1, 0, 0), nx=50, plot_file='plotfile')
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).

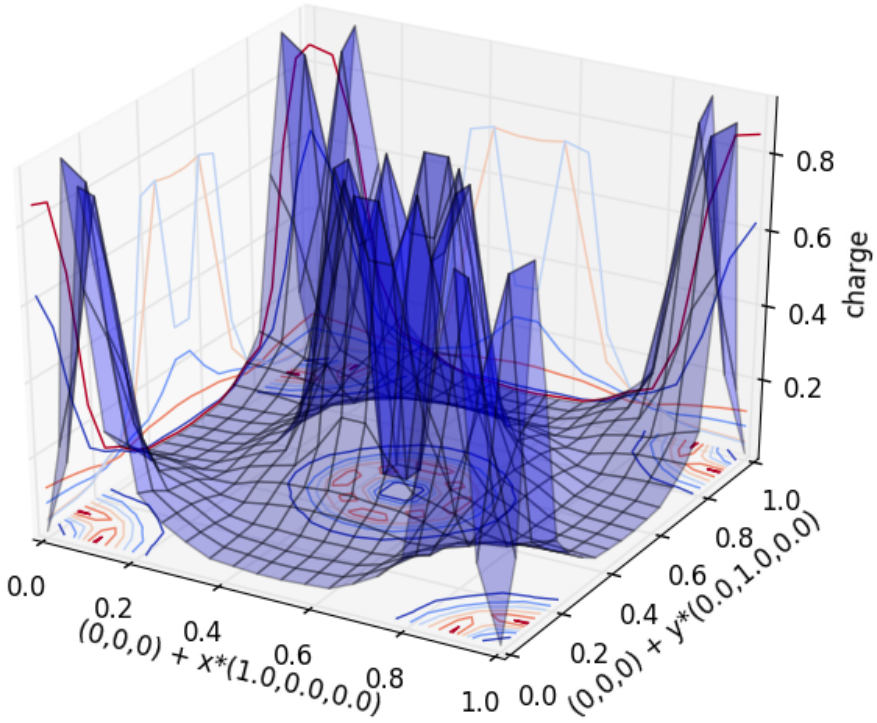


Plotting a 2D section of the charge density (examples 5)

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 and two 3D vectors which define the plane.

```
from postqe import get_charge, get_cell_data, compute_G, plot2D_FFTinterp

fin = "./Ni.xml"                                                    # file xml produce by QE
ibrav, alat, a, b, nat, ntyp,\
atomic_positions, atomic_species = get_cell_data(fin)  # get some data on the unit cell
charge, chargediff = get_charge(fin)              # get the charge (and charge diff)

G = compute_G(b, charge.shape)
fig1 = plot2D_FFTinterp(charge, G, a, x0=(0, 0, 0), e1=(1, 0, 0), e2=(0, 1, 0),
                        nx=20, ny=20, plot_file='plotfile')
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:



Plotting 1D sections of different potentials (examples 6)

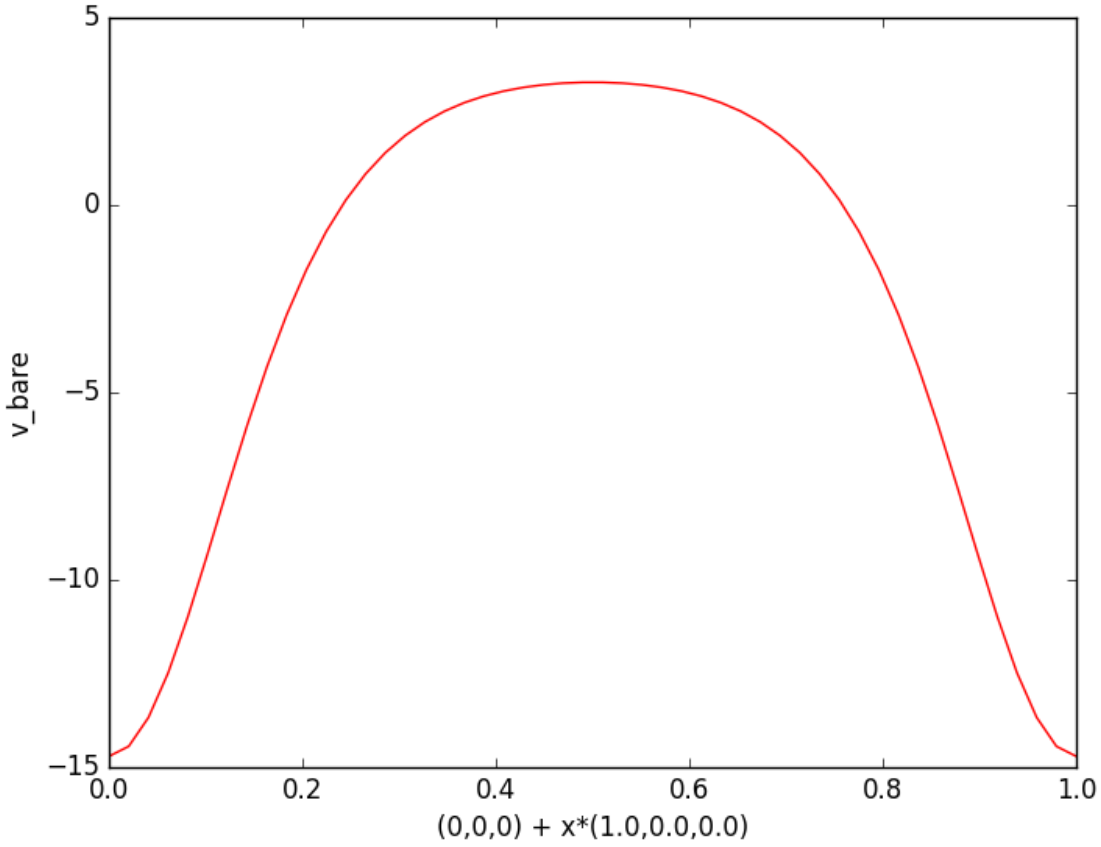
This example computes all the different potentials available, i.e. the bare potential V_{bare}, the Hartree potential V_H, the exchange-correlation potential V_{xc} and the total potential $V_{tot} = V_{bare} + V_H + V_{xc}$. All necessary information is taken from the xml output file of QE and the HDF5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

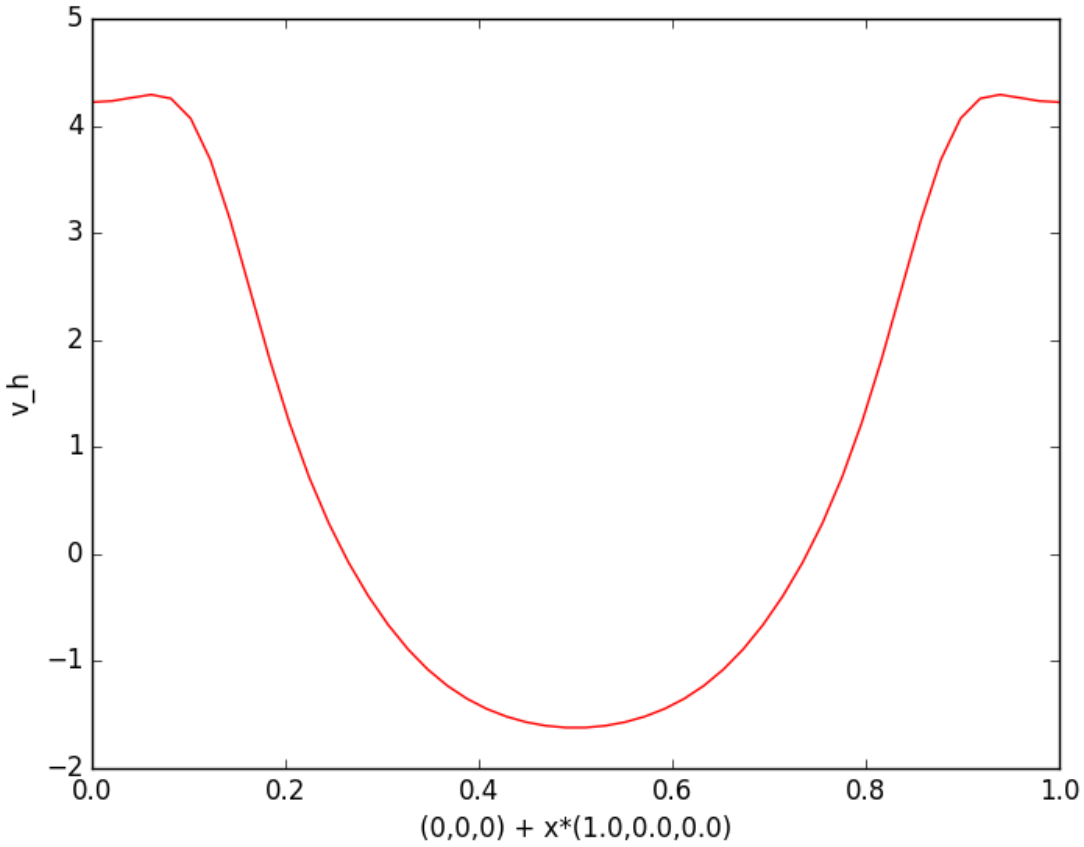
```
import numpy as np
from postqe import get_charge, get_cell_data, get_calculation_data, \
    compute_v_bare, compute_v_h, compute_v_xc, compute_G, plot1D_FFTinterp

fin = "./Ni.xml"                                          # file xml produce by QE
ibrav, alat, a, b, nat, ntyp,\
atomic_positions, atomic_species = get_cell_data(fin)  # get some data on the unit cell
prefix, outdir, ecutwfc, ecutrho, functional, lsda, noncolin, pseudodir, nr, nr_smooth =\
    get_calculation_data(fin)               # get some data on the QE calculation

charge, chargediff = get_charge(fin)            # get the charge (and charge diff) from the HDF5 file

# Compute the bare potential (v_bare), the Hartree potential (v_h) and the exhange-correlation potent
v_bare = compute_v_bare(ecutrho, alat, a[0], a[1], a[2], nr, atomic_positions, atomic_species, pseudo
v_h = compute_v_h(charge, ecutrho, alat, b)
charge_core = np.zeros(charge.shape)
```

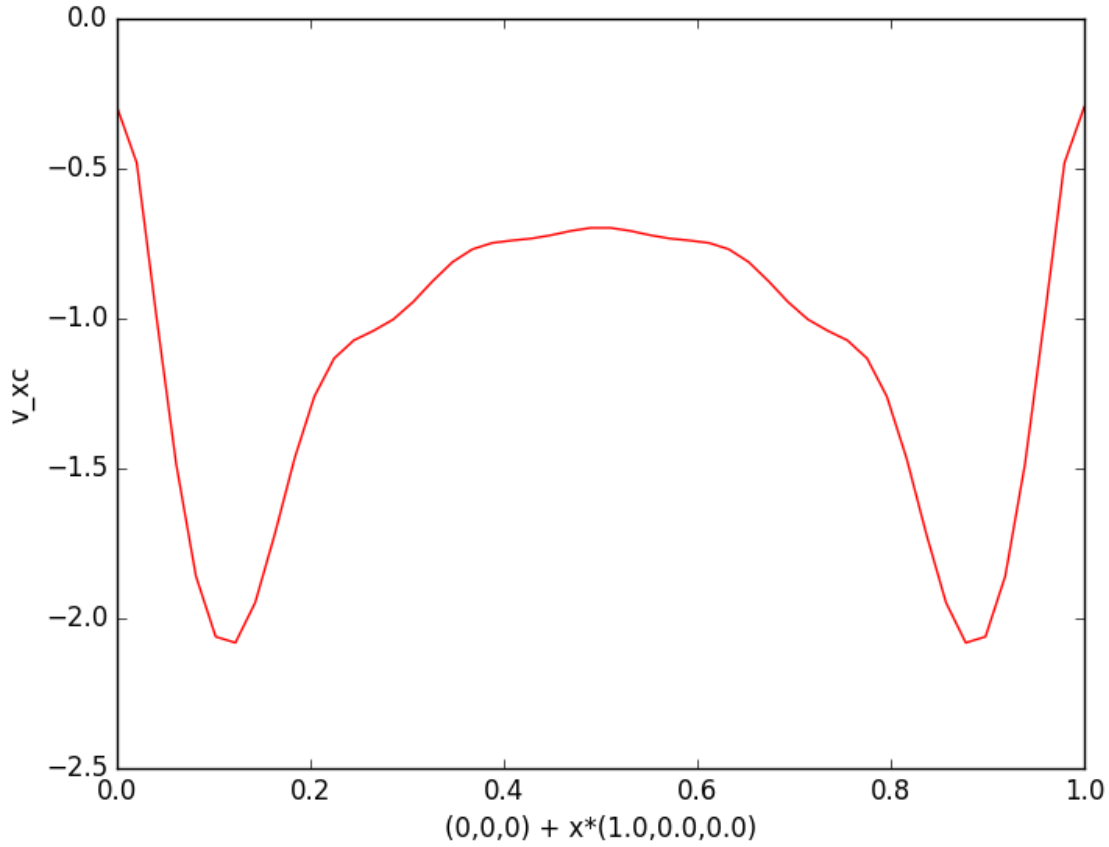
```
v_xc = compute_v_xc(charge, charge_core, str(functional))
# Add them up to get the total potential
v_tot = v_bare + v_h + v_xc

G = compute_G(b, charge.shape)       # calculate the G vectors for plotting
# Plot the potentials as 1D sections from (0,0,0) along (1,0,0)
fig1 = plot1D_FFTinterp(v_bare, G, a, x0=(0, 0, 0), e1=(1, 0, 0), nx=50, ylab='v_bare', plot_file='pl
fig1.savefig("figure_v_bare.png")

fig2 = plot1D_FFTinterp(v_h, G, a, x0=(0, 0, 0), e1=(1, 0, 0), nx=50, ylab='v_h', plot_file='plot_v_h
fig2.savefig("figure_v_h.png")

fig3 = plot1D_FFTinterp(v_xc, G, a, x0=(0, 0, 0), e1=(1, 0, 0), nx=50, ylab='v_xc', plot_file='plot_v
fig3.savefig("figure_v_xc.png")

fig4 = plot1D_FFTinterp(v_tot, G, a, x0=(0, 0, 0), e1=(1, 0, 0), nx=50, ylab='v_tot', plot_file='plot
fig4.savefig("figure_v_tot.png")
```

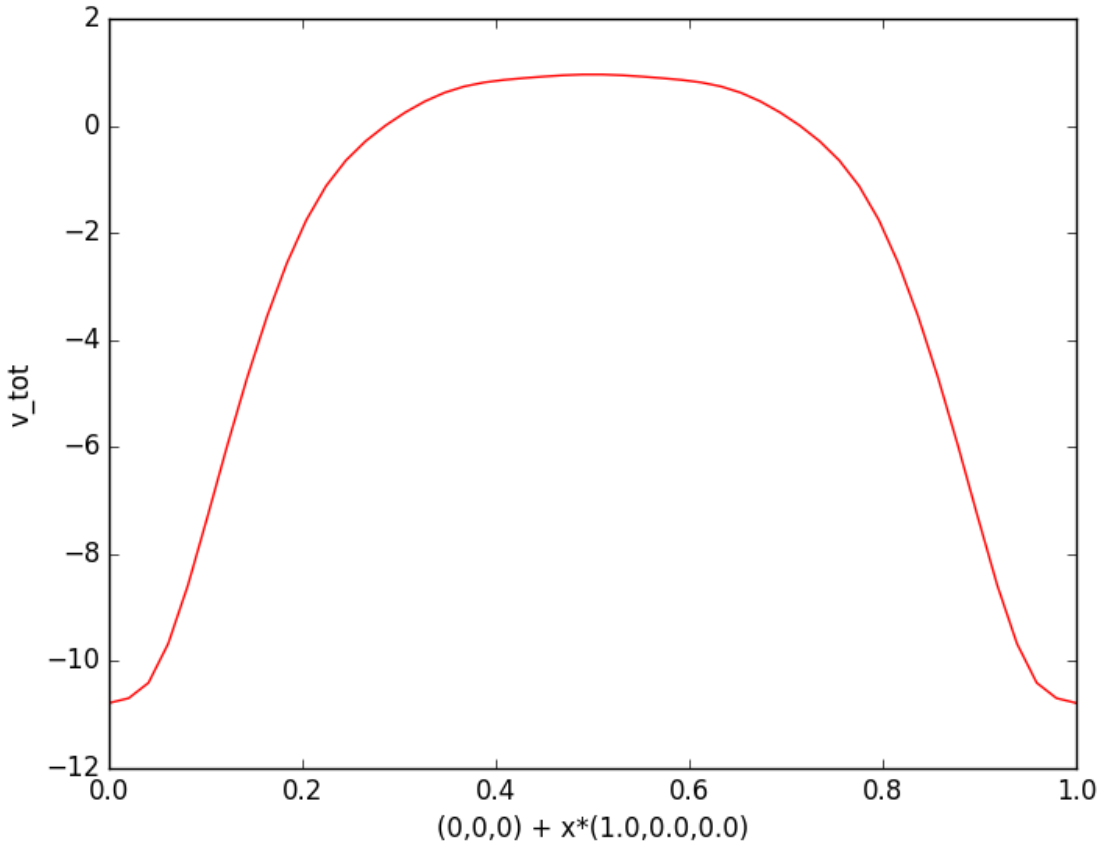
The first lines are calls to functions to extract some necessary information from the xml output file of QE. Then we need to read the charge from the HDF5 file as in example 4. We call the proper `postqe` function to compute each potential and add them up to get the total potential.

As in example 4, plotting is essentially a call to a single function, `plot_charge1D()`, passing the proper potential to be plot and changing the label on the y axis. 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)


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*(1.0,0.0,0.0)


2
0
−2
−4
−6
−8
−10
−12
v_tot
0.0
0.2
0.4
0.6
0.8
1.0
(0,0,0) + x*(1.0,0.0,0.0)

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