pyqha Documentation

Release 0.1

Mauro Palumbo

Oct 26, 2016

CONTENTS:

CHAPTER

ONE

INTRODUCTION

`pyqha` is a Python package to perform quasi-harmonic and related calculations from total energies at 0 K, elastic constants at 0 K and phonon densities of states. The package provides Python functions to postprocess the results of your favourite DFT code, such as Quantum Espresso [1] or VASP [2], to obtain quasi-harmonic properties. It is meant to be imported in your own code or used to produce quasi-harmonic results (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. Finally note that you may couple the package with some other available calculation Python tools, such as ASE or AiiDA. The package is based on numpy, scipy and matplotlib libraries.

A non-exhaustive list of properties which can be obtained using `pyqha` is:

- quasi-harmonic Helmholtz energy for isotropic and anisotropic unit cells
- quasi-harmonic thermal expansions for isotropic and anisotropic unit cells
- quasi-harmonic bulk modulus for isotropic unit cells
- quasi-harmonic heat capacity for isotropic unit cells
- quasi-static elastic constants for anisotropic unit cells

Current features of the package include:

- Fit the total energy $E_{tot}(V)$ with Murnaghan's equation of state
- Fit the total energy $E_{tot}(a, b, c)$, where (a, b, c) are the lattice parameters of hexagonal, tetragonal, orthorombic cells, using a quadratic or quartic polynomial
- Minimize the energy $E_{tot}(V) + F_{vib}(V, T)$ as a function of temperature with Murnaghan's equation of state
- Minimize the energy $E_{tot}(a, b, c) + F_{vib}(a, b, c, T)$ as a function of temperature using a quadratic or quartic polynomial
- Calculate the quasi-static elastic constant tensor as a function of temperature

The equations to obtain these properties are relatively simple, for an introduction on the quasi-harmonic approximation you can see Baroni et al., available online at https://arxiv.org/abs/1112.4977 or [3]. For an introduction on quasi-static elastic constants see [4] Have a look at the very good documentation of the *thermo_pw* fortran package available at http://qeforge.qe-forge.org/gf/project/thermo_pw/.

[1] http://www.quantum-espresso.org/

[2] https://www.vasp.at/

[3]

13. Palumbo, B. Burton, A. Costa e silva, B. Fultz, B. Grabowski, G. Grimvall, B. Hallstedt, O. Hellman, B. Lindahl, A. Schneider, P.E.A. Turchi, and W. Xiong. Physica Status Solidi (B) Basic Research, 251(1):14–32, 2014

[4]

25. Wang, J. J. Wang, H. Zhang, V. R. Manga, S. L. Shang, L.-Q. Chen, and Z.-K. Liu. Journal of Physics Condensed Matter, 22:225404, 2010.

1.1 Installation

You can download all package files from GitHub (https://github.com/mauropalumbo75/pyqha) 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 `pyqha`. You can import all of them as:

```
from pyqha 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.

CHAPTER TWO

TUTORIAL

This is a simple tutorial demonstrating the main functionalities of `pyqha`. 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	Fit $E_{tot}(V)$ for a cubic (isotropic) system using Murnaghan EOS
2	Fit $E_{tot}(a, c)$ for an hexagonal (anisotropic) system using a polynomial
3	Calculate the harmonic thermodynamic properties (ZPE, vibrational energy, Helmholtz energy, entropy and heat capacity from a phonon DOS
4	Calculate the harmonic thermodynamic properties as in the previous examples from several phonon DOS
5	A quasi-harmonic calculation for a cubic (isotropic) system using Murnaghan EOS
6	A quasi-harmonic calculation for an hexagonal (anisotropic) system using a quadratic polynomial
7	A quasi-static calculation for the elastic tensor of an hexagonal (anisotropic) system using a quadratic polynomial
8	Numerical issues in quasi-harmonic calculations

Several simplified plotting functions are available in `pyqha` 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.

2.1 Fitting the total energy

The simplest task you can do with `pyqha` is to fit the total energy as a function of volume $E_{tot}(V)$ (example1) or lattice parameters values $E_{tot}(a, b, c, \alpha, \beta, \gamma)$ (example2). In the former case, you can use an equation of state (EOS) such as Murnaghan's or similar. In the latter case, you must use polynomials. Currently the Murnaghan EOS and quadratic and quartic polynomials are implemented in `pyqha`. Besides, only a, b, c lattice parameters can be handled. This includes cubic, hexagonal, tetragonal and orthorombic systems.

Let's start with the simpler case where we want 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 approximate as isotropic (for example an hexagonal system with nearly constant c/a ratio).

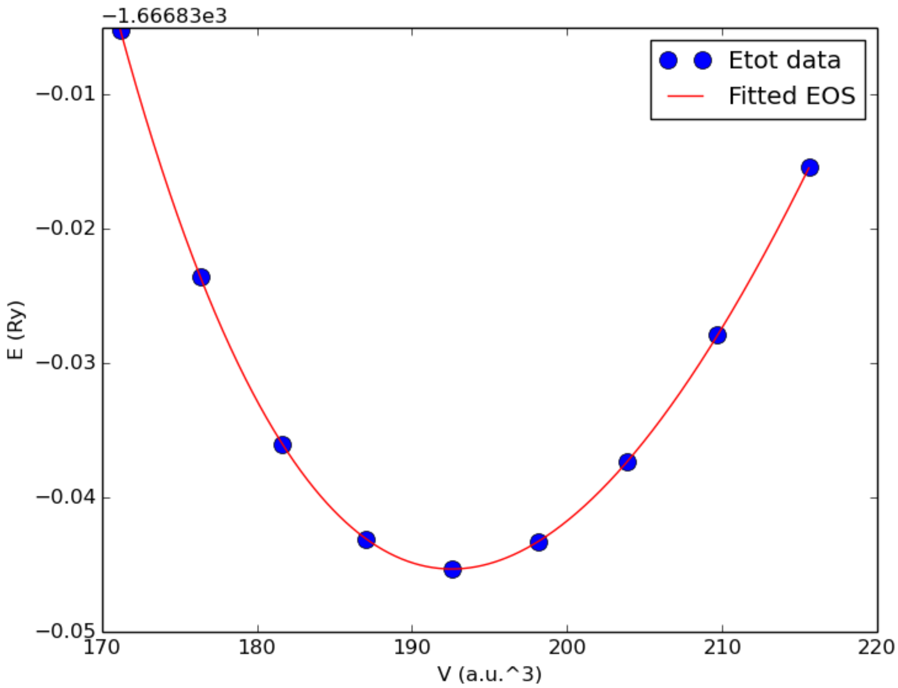
```
from pyqha import fitEtotV, plot_EV

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 χ^2 *chi*. The fitting results are also written in details on the *stdout*:

```
# Murnaghan EOS                    chi squared= 6.3052568895e-09
# Etotmin= -1.6668753460e+03 Ry          Vmin= 1.9256061524e+02 a.u.^3         B0= 3.
↪9507615923e+03 kbar         dB0/dV= 4.7879823925e+00
##############################################################################
('# V *a.u.^3)', '\t\t', 'Etot', ' (Ry)\t\t', 'Etotfit', ' (Ry)\t\t', 'Etot-Etotfit␣
↪(Ry)\tP (kbar)')
('1.7119697047e+02', '\t', '-1.6668351807e+03\t -1.6668351587e+03\t -2.2057946126e-
↪05\t 6.2382144794e+02')
('1.7637989181e+02', '\t', '-1.6668536038e+03\t -1.6668536431e+03\t 3.9279193061e-
↪05\t 4.3100002530e+02')
('1.8166637877e+02', '\t', '-1.6668660570e+03\t -1.6668660710e+03\t 1.4066826679e-
↪05\t 2.6537032641e+02')
('1.8705745588e+02', '\t', '-1.6668731355e+03\t -1.6668731118e+03\t -2.3774691499e-
↪05\t 1.2288570223e+02')
('1.9255414767e+02', '\t', '-1.6668753764e+03\t -1.6668753460e+03\t -3.0400133255e-
↪05\t 1.3270797876e-01')
('1.9815747866e+02', '\t', '-1.6668732871e+03\t -1.6668732783e+03\t -8.8363487976e-
↪06\t -1.0577273936e+02')
('2.0386847338e+02', '\t', '-1.6668673220e+03\t -1.6668673472e+03\t 2.5137771445e-
↪05\t -1.9727140701e+02')
('2.0968815635e+02', '\t', '-1.6668579007e+03\t -1.6668579345e+03\t 3.3763105193e-
↪05\t -2.7643217490e+02')
('2.1561755211e+02', '\t', '-1.6668454001e+03\t -1.6668453730e+03\t -2.7177809670e-
↪05\t -3.4501143525e+02')
```

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:



The second example shows how to fit the total energy of an hexagonal system, i.e. as a function of the lattice parameters (a, c). The input file (*fin*) contains three columns, the first two with the (a, c) (in $a.u.$) and the third one with the energies (in $Ryd/cell$). Note that if the original data are as $(a, c/a)$, as often reported, you must convert the c/a values into c values in the input file. The `fitEtot()` reads the input file and perform the fit using either a quadratic or quartic polynomial (as specified by the parameter *fitype*).

```
fin = "./Etot.dat"          # contains the input energies

# fits the energies and returns the coeffients a0 and the chi squared chia0
# the fit is done with a quartic polynomial
celldmsx, Ex, a0, chia0, mincelldms, fmin = fitEtot(fin,fittype="quartic",guess=[5.
↪12374914,0.0,8.19314311,0.0,0.0,0.0])

# 3D plot only with fitted energy
fig1 = plot_Etot(celldmsx,Ex=None,n=(5,0,5),nmesh=(50,0,50),fittype="quartic",ibrav=4,
↪a=a0)
fig1.savefig("figure_1.png")
# 3D plot fitted energy and points
fig2 = plot_Etot(celldmsx,Ex,n=(5,0,5),nmesh=(50,0,50),fittype="quartic",ibrav=4,a=a0)
fig2.savefig("figure_2.png")
# 2D contour plot with fitted energy
fig3 = plot_Etot_contour(celldmsx,nmesh=(50,0,50),fittype="quartic",ibrav=4,a=a0)
fig3.savefig("figure_3.png")
```

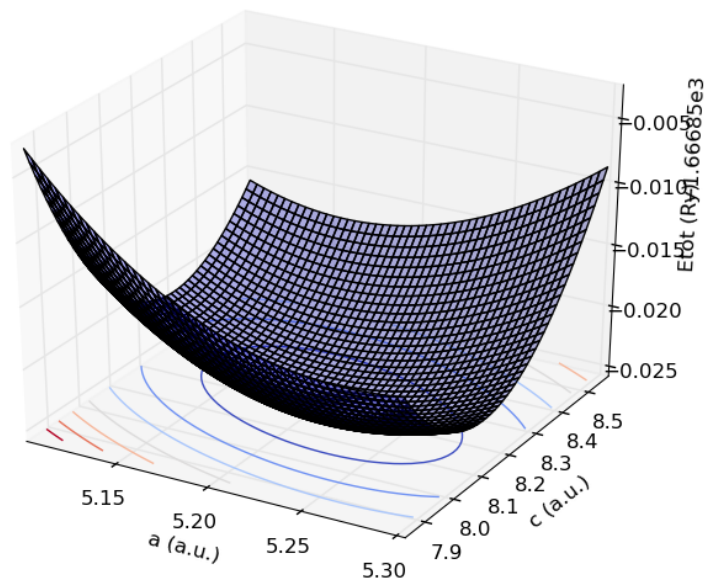
The output of the fitting function is:

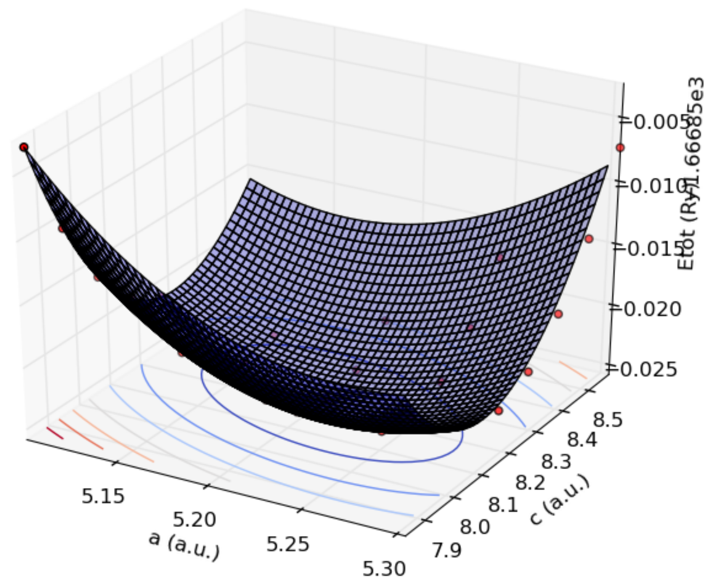
```
quartic fit
('a', '\t\t\t', 'c', '\t\t\t', 'Etot', '\t\t\t', 'Etotfit', '\t\t\t', 'Etot-Etotfit')
('5.1043155930e+00', '\t', '7.8471807981e+00', '\t', '-1.6668528744e+03', '\t', '-1.
↪6668528745e+03', '\t', '7.0725036494e-08')
```

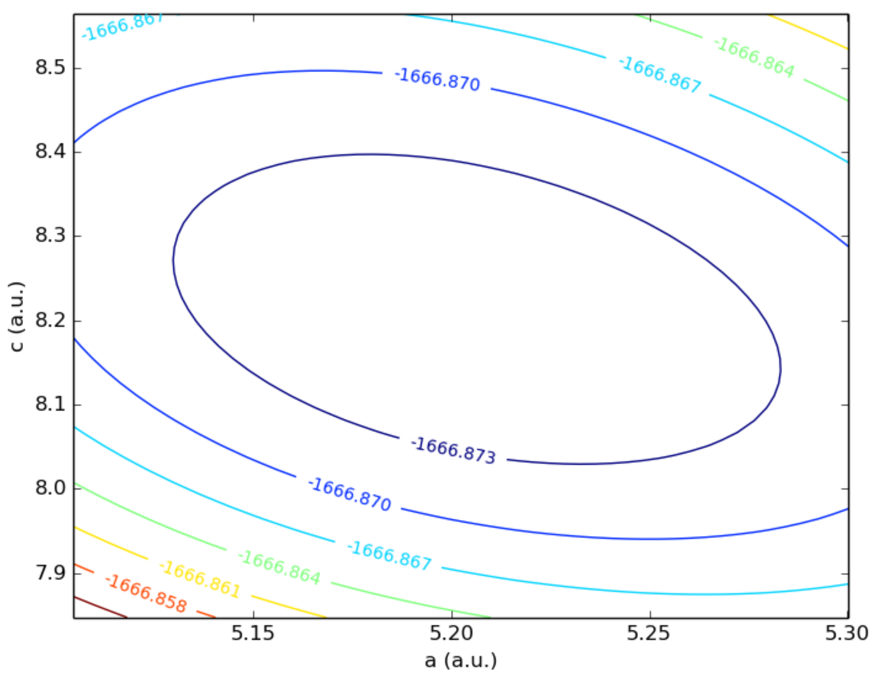
```
('5.1543155930e+00', '\t', '7.9240488978e+00', '\t', '-1.6668649001e+03', '\t', '-1.
↪6668649002e+03', '\t', '9.8750206234e-08')
('5.2043155930e+00', '\t', '8.0009169975e+00', '\t', '-1.6668716783e+03', '\t', '-1.
↪6668716776e+03', '\t', '-7.7131721810e-07')
('5.2543155930e+00', '\t', '8.0777850972e+00', '\t', '-1.6668737355e+03', '\t', '-1.
↪6668737365e+03', '\t', '9.2776986094e-07')
('5.3043155930e+00', '\t', '8.1546531969e+00', '\t', '-1.6668715550e+03', '\t', '-1.
↪6668715547e+03', '\t', '-3.2629236557e-07')
('5.1043155930e+00', '\t', '7.9492671099e+00', '\t', '-1.6668606004e+03', '\t', '-1.
↪6668606001e+03', '\t', '-2.1225582714e-07')
('5.1543155930e+00', '\t', '8.0271352096e+00', '\t', '-1.6668700587e+03', '\t', '-1.
↪6668700591e+03', '\t', '3.0630963010e-07')
('5.2043155930e+00', '\t', '8.1050033093e+00', '\t', '-1.6668744794e+03', '\t', '-1.
↪6668744800e+03', '\t', '6.7416499405e-07')
('5.2543155930e+00', '\t', '8.1828714090e+00', '\t', '-1.6668743826e+03', '\t', '-1.
↪6668743814e+03', '\t', '-1.2613072613e-06')
('5.3043155930e+00', '\t', '8.2607395087e+00', '\t', '-1.6668702280e+03', '\t', '-1.
↪6668702285e+03', '\t', '4.9947925618e-07')
('5.1043155930e+00', '\t', '8.0513534218e+00', '\t', '-1.6668660570e+03', '\t', '-1.
↪6668660572e+03', '\t', '2.5535950954e-07')
('5.1543155930e+00', '\t', '8.1302215215e+00', '\t', '-1.6668731355e+03', '\t', '-1.
↪6668731348e+03', '\t', '-7.0765509008e-07')
('5.2043155930e+00', '\t', '8.2090896212e+00', '\t', '-1.6668753764e+03', '\t', '-1.
↪6668753765e+03', '\t', '1.0862777344e-08')
('5.2543155930e+00', '\t', '8.2879577209e+00', '\t', '-1.6668732871e+03', '\t', '-1.
↪6668732880e+03', '\t', '8.4404246081e-07')
('5.3043155930e+00', '\t', '8.3668258206e+00', '\t', '-1.6668673220e+03', '\t', '-1.
↪6668673216e+03', '\t', '-4.0985673877e-07')
('5.1043155930e+00', '\t', '8.1534397336e+00', '\t', '-1.6668694343e+03', '\t', '-1.
↪6668694339e+03', '\t', '-4.5153024075e-07')
('5.1543155930e+00', '\t', '8.2333078333e+00', '\t', '-1.6668743061e+03', '\t', '-1.
↪6668743073e+03', '\t', '1.2780792531e-06')
('5.2043155930e+00', '\t', '8.3131759330e+00', '\t', '-1.6668745384e+03', '\t', '-1.
↪6668745377e+03', '\t', '-7.0226747084e-07')
('5.2543155930e+00', '\t', '8.3930440327e+00', '\t', '-1.6668706178e+03', '\t', '-1.
↪6668706173e+03', '\t', '-4.6083209782e-07')
('5.3043155930e+00', '\t', '8.4729121324e+00', '\t', '-1.6668629842e+03', '\t', '-1.
↪6668629845e+03', '\t', '3.4305685404e-07')
('5.1043155930e+00', '\t', '8.2555260455e+00', '\t', '-1.6668709188e+03', '\t', '-1.
↪6668709191e+03', '\t', '3.3864898796e-07')
('5.1543155930e+00', '\t', '8.3363941452e+00', '\t', '-1.6668737584e+03', '\t', '-1.
↪6668737574e+03', '\t', '-9.7452812042e-07')
('5.2043155930e+00', '\t', '8.4172622449e+00', '\t', '-1.6668721346e+03', '\t', '-1.
↪6668721354e+03', '\t', '7.8953144111e-07')
('5.2543155930e+00', '\t', '8.4981303446e+00', '\t', '-1.6668665315e+03', '\t', '-1.
↪6668665315e+03', '\t', '-4.8681386033e-08')
('5.3043155930e+00', '\t', '8.5789984443e+00', '\t', '-1.6668573689e+03', '\t', '-1.
↪6668573688e+03', '\t', '-1.0538019524e-07')

Fitted polynomial is:

p(x1,x2) = -1291.10429456 + -184.969221276 * x1 + 42.2676103527 * x1^2 + -4.
↪81052197937 * x1^3 + 0.188178329491 * x1^4 +
-49.1590620388 *x2 + 5.34145510286 *x2^2 + -0.245662970361 *x2^3 + -0.000328650634003␣
↪*x2^4 +
8.48734670683 *x1*x2 + -0.67806386411 *x1*x2^2 + 0.0444127765503 *x1*x2^3 + -0.
↪393401452901 *x1^2*x2 + -0.0458527834065 *x1^2*x2^2 +
0.0705006802514 *x1^3*x2
```

```
('Chi squared: ', 9.8189738184091843e-12, '\n')
('Minimun quartic: ', array([ 5.20422334,  0.        ,  8.20918742,  0.        ,  0. ␣
↪      ,  0.        ]), '\tEnergy at the minimum: -1.66687537646402097380e+03\n')
```

Optionally, you can use the functions `plot_Etot()`, `plot_Etot_contour()` to create 3D or contour plots of the fitted energy over the grid (a, c), including or not the original energy points:







2.2 Computing thermal properties from phonon DOS

`pyqha` can calculate the vibrational properties of your system from the phonon DOS in the harmonic approximation as shown in *example3*. The DOS file must be a two columns one, the first column being the energy (in $Ryd/cell$) and the second column being the density of states (in $(Ryd/cell)^{-1}$).

```
fin = "./dos.dat"
fout = "./thermo"

TT = gen_TT(1,1000,0.5)         # create a numpy array of temperatures from 1 to 1000␣
↪step 0.5

E, dos = read_dos(fin)         # read the dos file. It returns the energies and dos␣
↪values.


T, Evib, Svib, Cvib, Fvib, ZPE, modes = compute_thermo(E/RY_TO_CMM1,dos*RY_TO_CMM1,TT)
write_thermo(fout,T, Evib, Fvib, Svib, Cvib, ZPE, modes)

from pyqha import simple_plot_xy, multiple_plot_xy
# plot the original phonon DOS
fig1 = simple_plot_xy(E,dos,xlabel="E (Ryd/cell)",ylabel="phonon DOS (Ry/cell)^{-1}")
fig1.savefig("figure_1.png")
# create several plots for the thermodynamic quantities computed
fig2 = simple_plot_xy(T,Evib,xlabel="T (K)",ylabel="Evib (Ry/cell)")
fig2.savefig("figure_2.png")
fig3 = simple_plot_xy(T,Fvib,xlabel="T (K)",ylabel="Fvib (Ry/cell)")
fig3.savefig("figure_3.png")
fig4 = simple_plot_xy(T,Svib,xlabel="T (K)",ylabel="Svib (Ry/cell/K)")
fig4.savefig("figure_4.png")
fig5 = simple_plot_xy(T,Cvib,xlabel="T (K)",ylabel="Cvib (Ry/cell/K)")
fig5.savefig("figure_5.png")
```

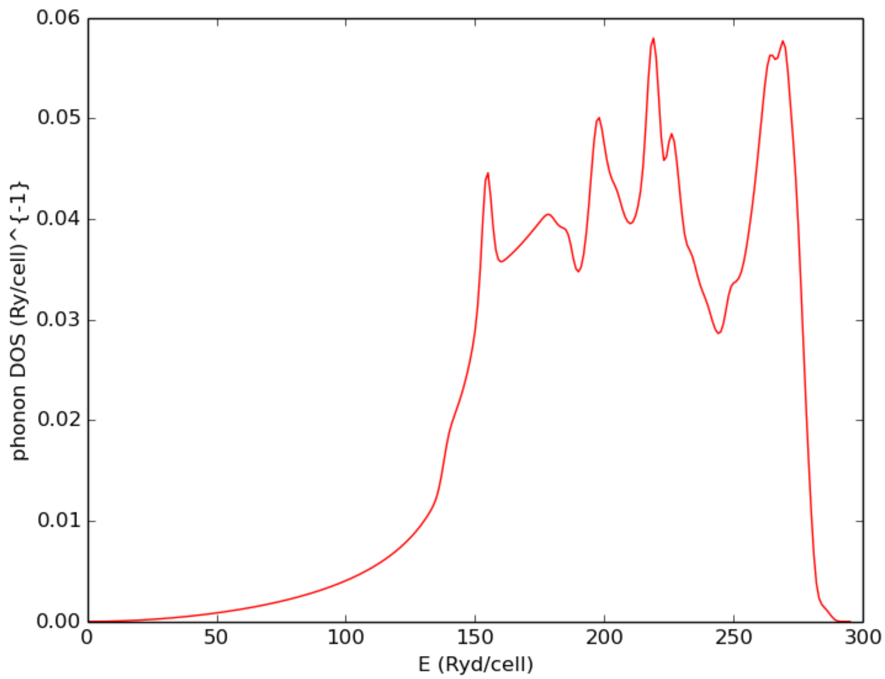
The output produced by the function `compute_thermo()` is stored in the variables *T, Evib, Svib, Cvib, Fvib, ZPE, modes* and can be written in a file using the function `write_thermo()`. This output file is as:

```
# total modes from dos = 5.9999726114e+00
# ZPE = 5.6214272319e-03 Ry/cell
# Multiply by 13.6058 to have energies in eV/cell etc..
# Multiply by 13.6058 x 23060.35 = 313 754.5 to have energies in cal/(N mol).
# Multiply by 13.6058 x 96526.0 = 1 313 313 to have energies in J/(N mol).
# N is the number of formula units per cell.
#
# T (K)          Evib (Ry/cell)        Fvib (Ry/cell)        Svib (Ry/cell/
↪K)          Cvib (Ry/cell/K)
1.0000000000e+00         5.6214272416e-03        5.6214271698e-03        7.1803604634e-
↪11          2.7457378670e-11
1.5000000000e+00         5.6214272660e-03        5.6214271296e-03        9.0971071082e-
↪11          7.6156598111e-11
2.0000000000e+00         5.6214273247e-03        5.6214270765e-03        1.2411743622e-
↪10          1.6670823649e-10
2.5000000000e+00         5.6214274420e-03        5.6214270023e-03        1.7586528823e-
↪10          3.1294495785e-10
3.0000000000e+00         5.6214276492e-03        5.6214268967e-03        2.5083680991e-
↪10          5.2875068522e-10
3.5000000000e+00         5.6214279847e-03        5.6214267469e-03        3.5365843213e-
↪10          8.2803005242e-10
4.0000000000e+00         5.6214284935e-03        5.6214265377e-03        4.8896152188e-
↪10          1.2247123119e-09
```

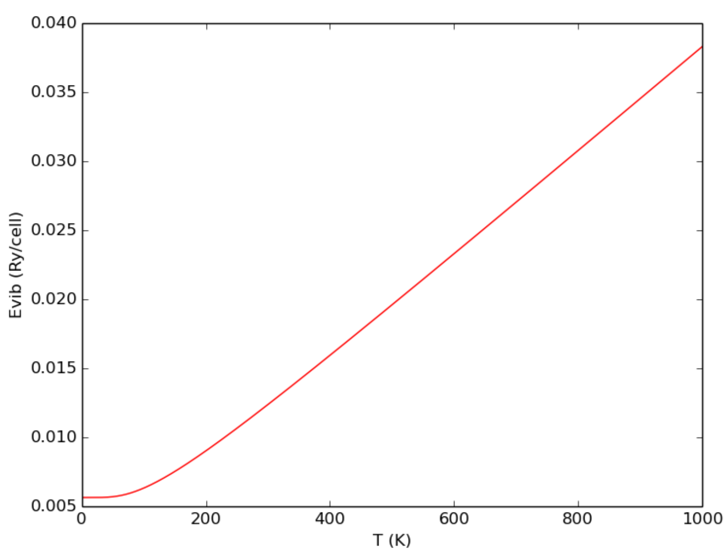
```
4.5000000000e+00        5.6214292279e-03        5.6214262517e-03        6.6138346588e-
↪10       1.7327611278e-09
5.0000000000e+00        5.6214302472e-03        5.6214258693e-03        8.7556952905e-
↪10       2.3661903162e-09
5.5000000000e+00        5.6214316174e-03        5.6214253684e-03        1.1361755892e-
↪09       3.1390850039e-09
6.0000000000e+00        5.6214334118e-03        5.6214247246e-03        1.4478717445e-
↪09       4.0656278651e-09
6.5000000000e+00        5.6214357110e-03        5.6214239112e-03        1.8153467817e-
↪09       5.1601305330e-09
7.0000000000e+00        5.6214386024e-03        5.6214228992e-03        2.2433135214e-
↪09       6.4370706688e-09
7.5000000000e+00        5.6214421809e-03        5.6214216571e-03        2.7365150966e-
↪09       7.9111356177e-09
8.0000000000e+00        5.6214465489e-03        5.6214201510e-03        3.2997322753e-
↪09       9.5972742802e-09
8.5000000000e+00        5.6214518161e-03        5.6214183448e-03        3.9377920199e-
↪09       1.1510759902e-08
9.0000000000e+00        5.6214581001e-03        5.6214161999e-03        4.6555775947e-
↪09       1.3667267917e-08
9.5000000000e+00        5.6214655265e-03        5.6214136752e-03        5.4580406777e-
↪09       1.6082974484e-08
1.0000000000e+01        5.6214742291e-03        5.6214107269e-03        6.3502160901e-
↪09       1.8774682469e-08
1.0500000000e+01        5.6214843502e-03        5.6214073091e-03        7.3372398855e-
↪09       2.1759981704e-08
1.1000000000e+01        5.6214960411e-03        5.6214033730e-03        8.4243715943e-
↪09       2.5057449144e-08
1.1500000000e+01        5.6215094629e-03        5.6213988672e-03        9.6170213626e-
↪09       2.8686891747e-08
1.2000000000e+01        5.6215247869e-03        5.6213937375e-03        1.0920782549e-
↪08       3.2669631041e-08
1.2500000000e+01        5.6215421953e-03        5.6213879269e-03        1.2341470051e-
↪08       3.7028823823e-08
1.3000000000e+01        5.6215618826e-03        5.6213813755e-03        1.3885164240e-
↪08       4.1789809033e-08
1.3500000000e+01        5.6215840567e-03        5.6213740202e-03        1.5558259988e-
↪08       4.6980467173e-08
1.4000000000e+01        5.6216089398e-03        5.6213657946e-03        1.7367519841e-
↪08       5.2631576208e-08
1.4500000000e+01        5.6216367707e-03        5.6213566288e-03        1.9320130039e-
↪08       5.8777146866e-08
1.5000000000e+01        5.6216678056e-03        5.6213464493e-03        2.1423757831e-
↪08       6.5454720686e-08
1.5500000000e+01        5.6217023209e-03        5.6213351785e-03        2.3686608350e-
↪08       7.2705615875e-08
1.6000000000e+01        5.6217406143e-03        5.6213227346e-03        2.6117479280e-
↪08       8.0575108690e-08
1.6500000000e+01        5.6217830073e-03        5.6213090314e-03        2.8725811574e-
↪08       8.9112541358e-08
```

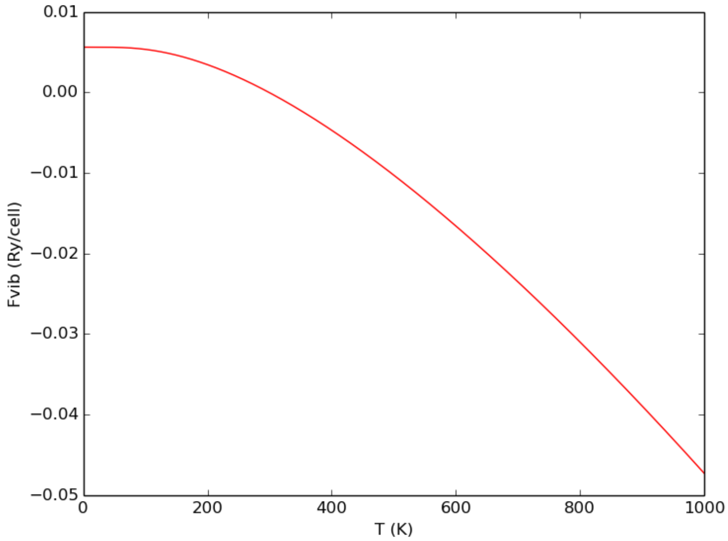
The first line is the simple integral of the input dos. It must be approximately equal to $3N$, where N is the number of atoms in the cell. In the present case (hex Os) it is equal to 6. The second line shows the Zero Point Energy (ZPE). After a few comments lines, the vibrational energy (Evib), Helmholtz energy (Fvib), entropy (Svib) and heat capacity (Cvib) are written as a function of temperature. All quantities are calculated in the harmonic approximation, i.e. for fixed volume (and lattice parameters).

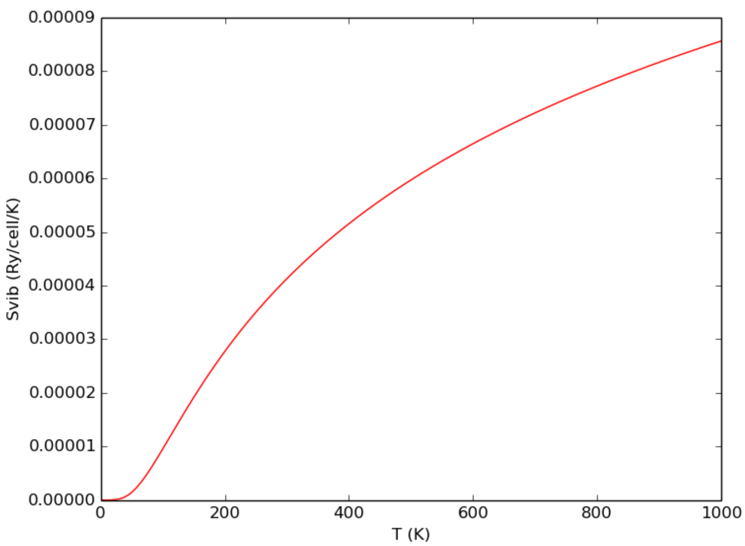
The original dos is plotted as:

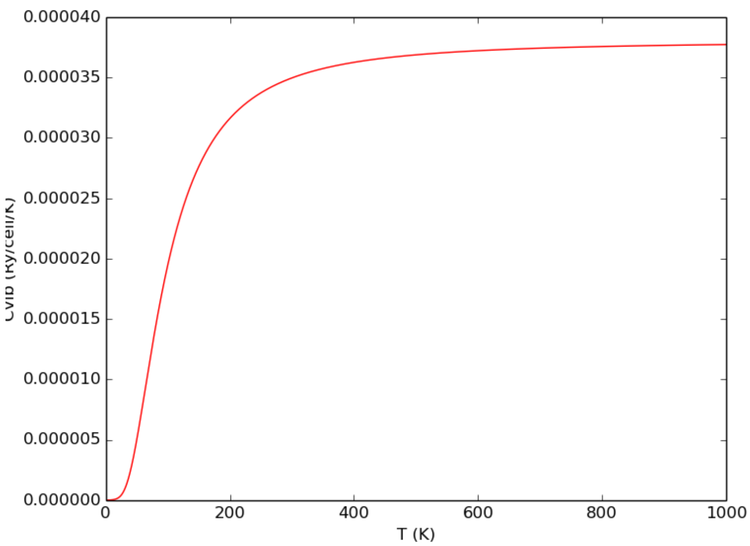


The calculated thermodynaminc functions are plotted as:




Fvib (Ry/cell)
T (K)


Svib (Ry/cell/K)
T (K)

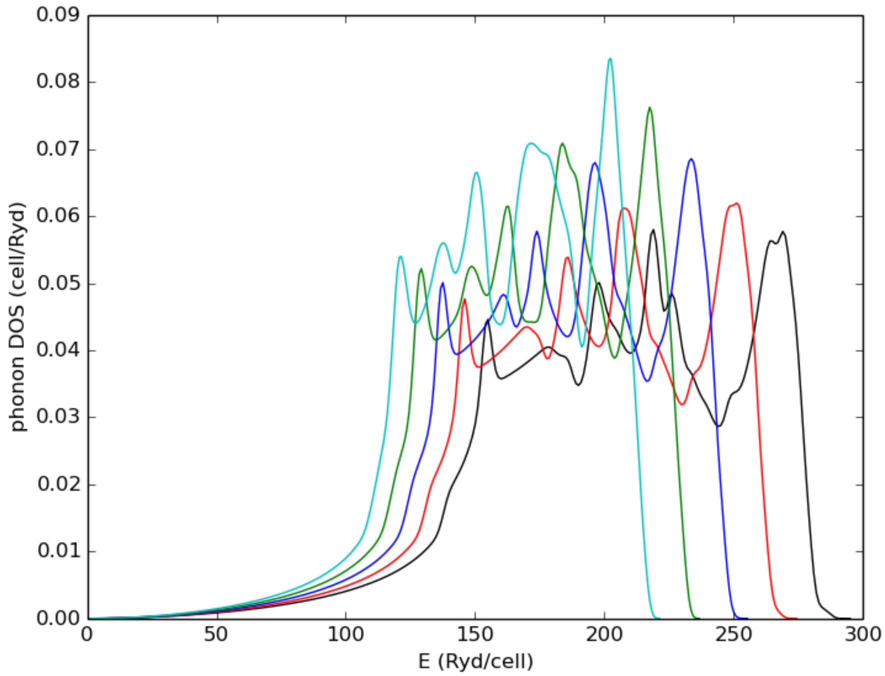


The following code (example4) shows how multiple dos files can be handled, a step which is preliminary to a quasi-harmonic calculation. The dos are for different volumes (for hexagonal Os).

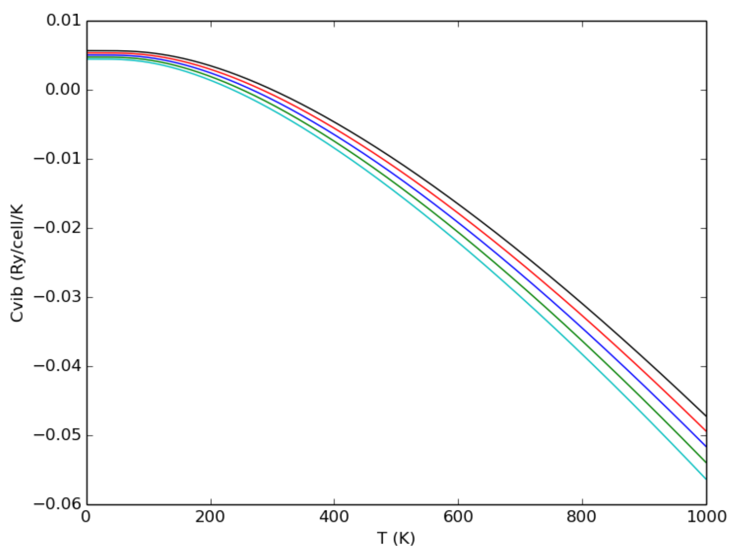
```
from pyqha import gen_TT, read_dos_geo, compute_thermo_geo
from pyqha import simple_plot_xy, multiple_plot_xy

fin = "dos_files/output_dos.dat.g"        # base name for the dos files (numbers will␣
↪be added as postfix)
fout = "thermo"                  # base name for the output files (numbers will be␣
↪added as postfix)

ngeo = 9

gE, gdos = read_dos_geo(fin,ngeo)         # read ngeo=9 dos files


# plot the first 5 phonon dos
fig1 = multiple_plot_xy(gE[:,0:5],gdos[:,0:5],xlabel="E (Ryd/cell)",ylabel="phonon␣
↪DOS (cell/Ryd)")
fig1.savefig("figure_1.png")

TT =gen_TT(1,1000)         # generate the numpy array of temperatures for which the␣
↪properties will be calculated

# compute the thermodynamic properties for all ngeo dos files and write them in fout+
↪"i" files, where is an int from 1 to ngeo
T, ggEvib, ggFvib, ggSvib, ggCvib, ggZPE, ggmodes = compute_thermo_geo(fin,fout,ngeo,
↪TT)

# plot the vibrational Helmholtz energy for the first 5 phonon dos
fig2 = multiple_plot_xy(T,ggFvib[:,0:5],xlabel="T (K)",ylabel="Cvib (Ry/cell/K")
fig2.savefig("figure_2.png")

# plot the vibrational entropy for the first 5 phonon dos
fig3 = multiple_plot_xy(T,ggSvib[:,0:5],xlabel="T (K)",ylabel="Cvib (Ry/cell/K")
fig3.savefig("figure_3.png")
```

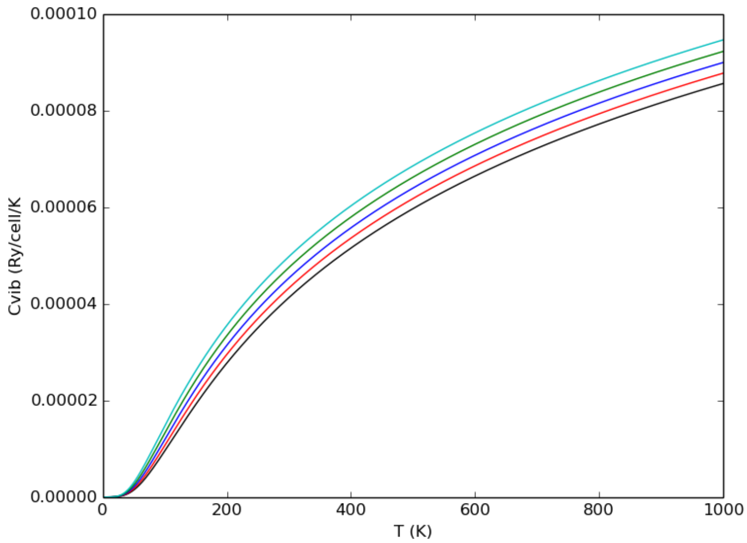
```
# plot the vibrational heat capacity for the first 5 phonon dos
fig4 = multiple_plot_xy(T,ggCvib[:,0:5],xlabel="T (K)",ylabel="Cvib (Ry/cell/K")
fig4.savefig("figure_4.png")
```

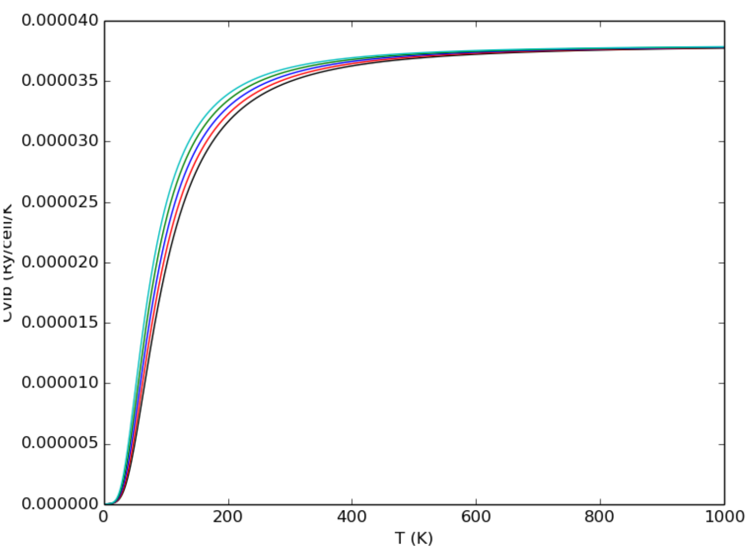
The first 5 phonon dos are plotted as (color order is: black, red, blue, green, cyan for increasing volumes):



The corresponding vibrational Helmoltz energies, entropies and heat capacity are plotted as:







2.3 Computing quasi-harmonic properties

Here we show how to do a full quasi-harmonic calculation starting from the E_{tot} and phonon DOS. First, we show an example using the Murnaghan EOS, having $E_{tot}(V)$ and the corresponding DOS, then using a quartic polynomial on the full grid (a, c) for an hexagonal cell.

Here is the code in the Murnaghan case:

```
from pyqha import RY_KBAR
from pyqha import gen_TT, read_dos_geo, compute_thermo_geo, read_thermo, rearrange_
↪thermo, fitFvibV, write_xy
from pyqha import simple_plot_xy, multiple_plot_xy

# this part is for calculating the thermodynamic properties from the dos
```

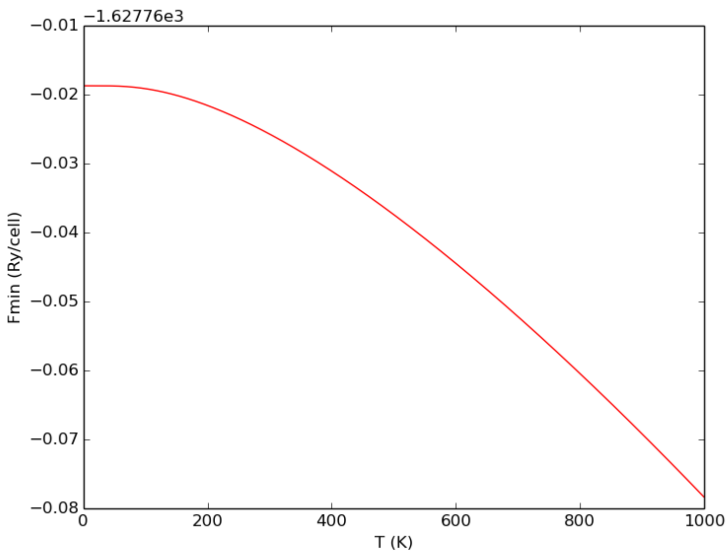
```
fdos="dos_files/output_dos.dat.g"        # base name for the dos files (numbers will␣
↪be added as postfix)
fthermo = "thermo"              # base name for the output files (numbers will be␣
↪added as postfix)

ngeo = 9        # this is the number of volumes for which a dos has been␣
↪calculated

TT =gen_TT(1,1000)        # generate the numpy array of temperatures for which the␣
↪properties will be calculated
T, Evib, Fvib, Svib, Cvib, ZPE, modes = compute_thermo_geo(fdos,fthermo,ngeo,TT)
nT = len(T)


# Alternatively, read the thermodynamic data from files, if you have already
# done the calculations. Uncomment the following 2 lines and delete the previous 3␣
↪lines
#T1, Evib1, Fvib1, Svib1, Cvib1 = read_thermo( fthermo, ngeo )
#T, T, Evib, Fvib, Svib, Cvib = rearrange_thermo( T1, Evib1, Fvib1, Svib1, Cvib1,␣
↪ngeo )


fEtot = "./Etot.dat"
thermodata = nT, T, Evib, Fvib, Svib, Cvib
TT, Fmin, Vmin, B0, betaT, Cv, Cp, aT, chi = fitFvibV(fEtot,thermodata)

fig1 = simple_plot_xy(TT,Fmin,xlabel="T (K)",ylabel="Fmin (Ry/cell)")
fig2 = simple_plot_xy(TT,Vmin,xlabel="T (K)",ylabel="Vmin (a.u.^3)")
fig3 = simple_plot_xy(TT,B0,xlabel="T (K)",ylabel="B0 (kbar)")
fig4 = simple_plot_xy(TT,betaT,xlabel="T (K)",ylabel="beta")
fig5 = simple_plot_xy(TT,Cp,xlabel="T (K)",ylabel="Cp (Ry/cell/K")
fig1.savefig("figure_1.png")
fig2.savefig("figure_2.png")
fig3.savefig("figure_3.png")
fig4.savefig("figure_4.png")
fig5.savefig("figure_5.png")

# save the results in a file if you want...
write_xy("Fmin.dat",T,Fmin,"T (K)","Fmin (Ryd/cell)")
write_xy("Vmin.dat",T,Vmin,"T (K)","Vmin (a.u.^3)")
write_xy("B0.dat",T,B0*RY_KBAR,"T (K)","B0 (kbar)")
write_xy("beta.dat",T, betaT,"T (K)","Beta=1/V dV/dT (1/K)")

import numpy as np
CvCp = np.zeros((len(T),2))
CvCp[:,0] = Cv
CvCp[:,1] = Cp
fig6 = multiple_plot_xy(TT,CvCp,xlabel="T (K)",ylabel="Cv/Cp (Ry/cell/K")
fig6.savefig("figure_6.png")

print_eos_data(V,E+Fvib[i],a,chi,"E")  # print full detail at each T
```

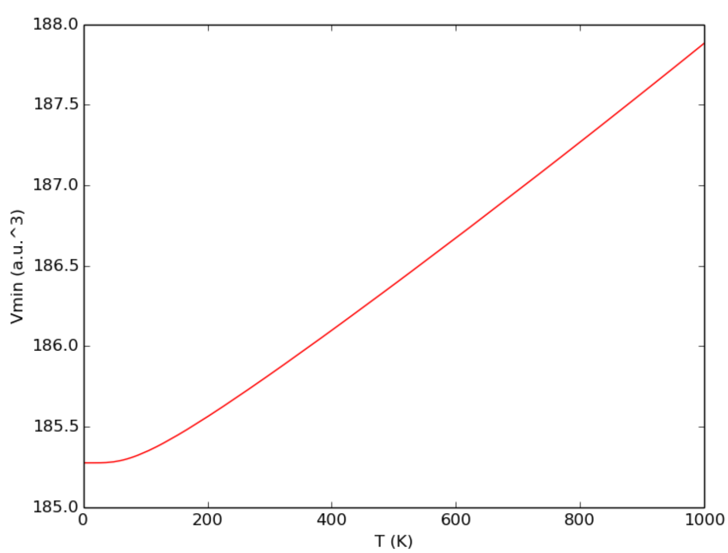
Note from the first line that there are some constants you can import from the module and use for unit conversions. See the documentation for more details on which ones are available. In this example, 9 volumes are used (ngeo=9). First the harmonic thermodynamic properties are computed as in the previous example. You store these quantities in a list called *thermodata*. You also need to read the total energies as in example 1 from the file *Etot.dat*, which is taken care inside the function `fitEtotV()`. This is the function which is really doing the quasi-harmonic calculations, i.e.

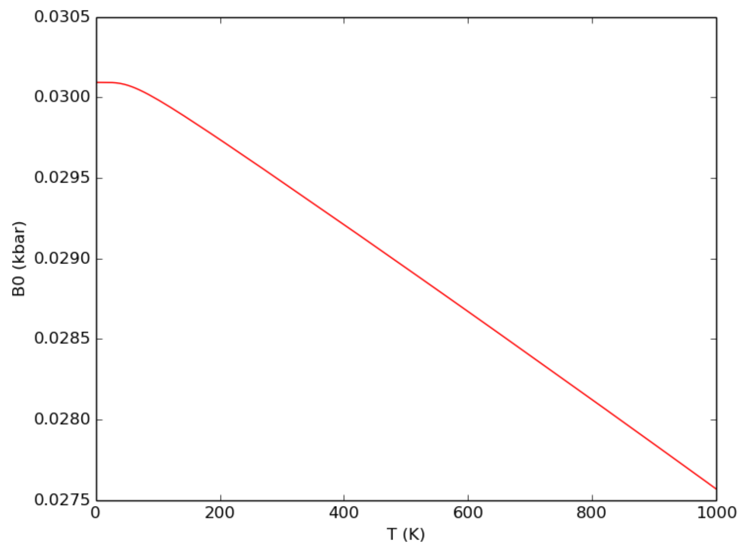
it fits a Murnaghan EOS at each T using $E_{tot}(V) + F_{vib}(V,T)$. It returns *TT, Fmin, Vmin, B0, betaT, Cv, Cp*, which are all numpy 1D arrays containing the temperatures where the calculations were done and the resulting minimun Helmholtz energy (at each T), minimun volume, isobaric bulk modulus, volume thermal expansion, constant volume and constant pressure heat capacities, respectively. These quantities correspond to $P = 0$.

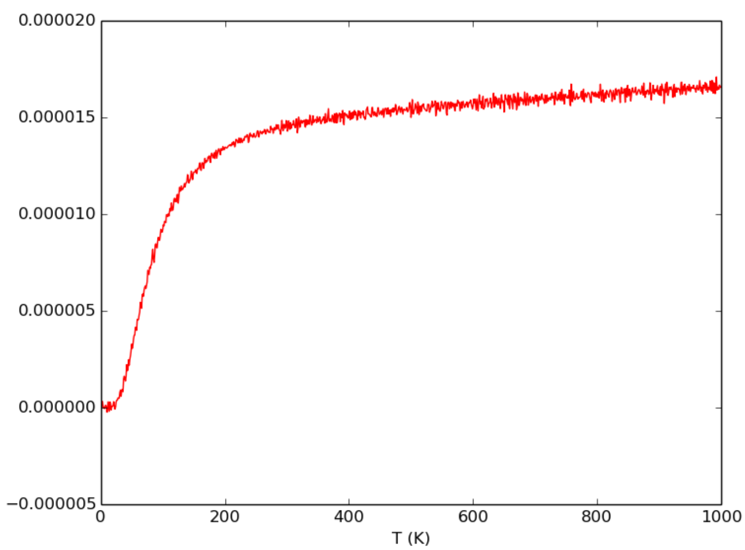
The following lines show how to plot each quantity on a single plot (using the function `simple_plot_xy()`), write the results in files (using the `write_xy()`) and plot both *Cv* and *Cp* in a single plot (using the function `multiple_plot_xy()`).

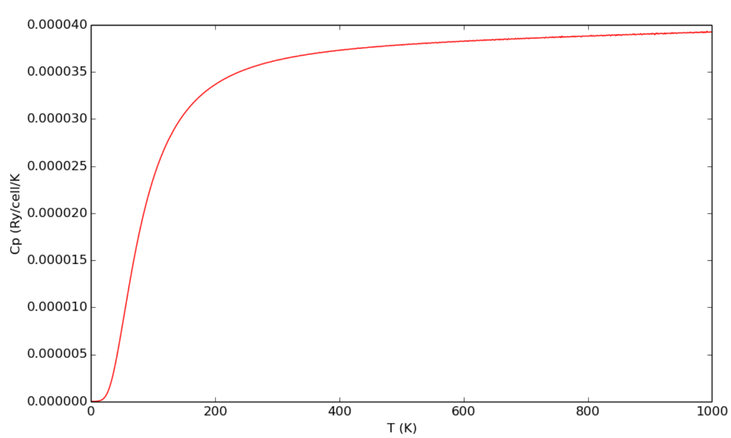
If everything went well, you should get the following plots:

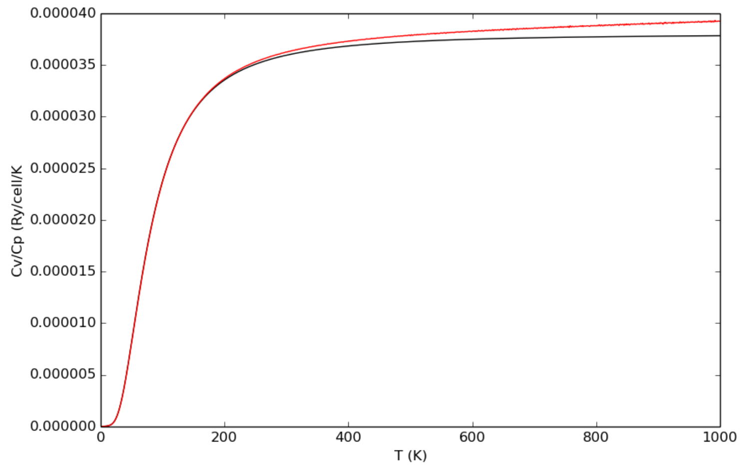





B0 (kbar)
0.0305
0.0300
0.0295
0.0290
0.0285
0.0280
0.0275
0
200
400
600
800
1000
T (K)


0.000020
0.000015
0.000010
0.000005
0.000000
−0.000005
0
200
400
600
800
1000
T (K)


Cp (Ry/cell/K
0.000040
0.000035
0.000030
0.000025
0.000020
0.000015
0.000010
0.000005
0.000000
0
200
400
600
800
1000
T (K)

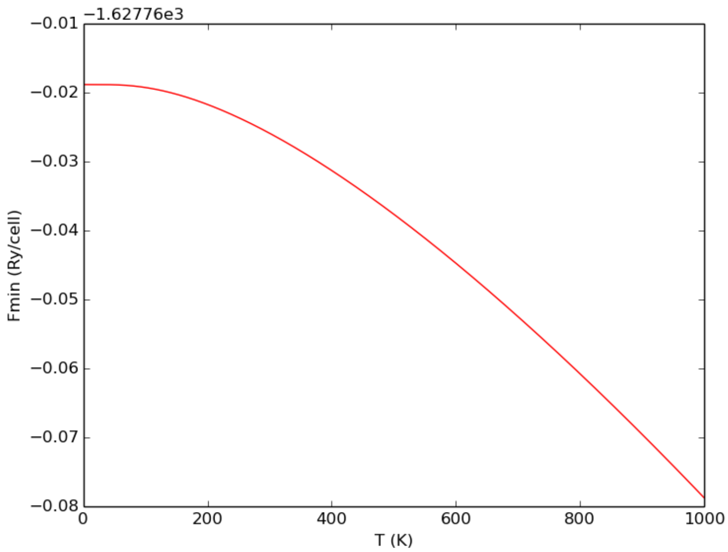


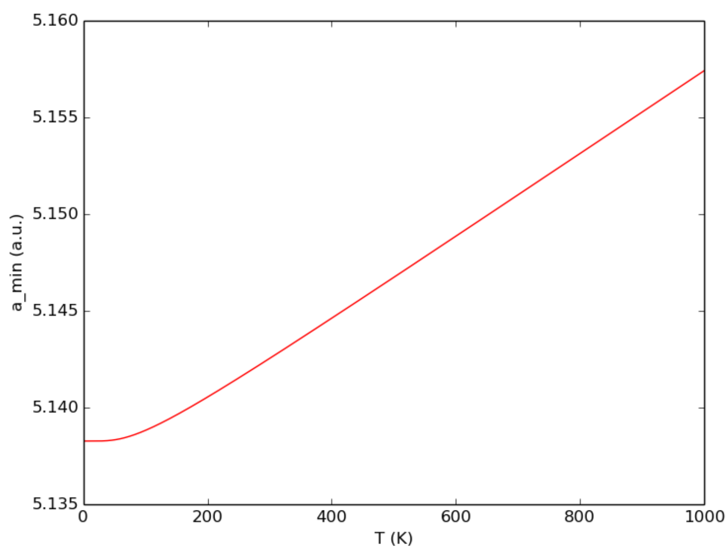
In the following we show the code for a similar example of an hexagonal (anisotropic) system. The code is similar to the previous examples with a few important differences.

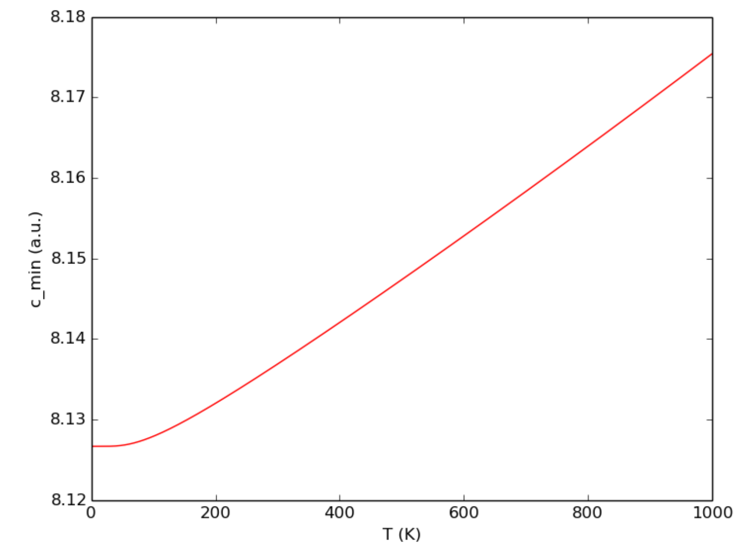
```
from pyqha import RY_KBAR
from pyqha import gen_TT, read_Etot, read_dos_geo, compute_thermo_geo, read_thermo, rearrange_thermo, fitFvib, write_celldmsT, write_alphaT
from pyqha import simple_plot_xy, plot_Etot, plot_Etot_contour

# this part is for calculating the thermodynamic properties from the dos
fdos="dos_files/output_dos.dat.g"        # base name for the dos files (numbers will be added as postfix)
fthermo = "thermo"                # base name for the output files (numbers will be added as postfix)

ngeo = 25         # this is the number of volumes for which a dos has been calculated

#TT =gen_TT(1,1000)        # generate the numpy array of temperatures for which the properties will be calculated
#T, Evib, Fvib, Svib, Cvib, ZPE, modes = compute_thermo_geo(fdos,fthermo,ngeo,TT)
#nT = len(T)

# Alternatively, read the thermodynamic data from files if you have already
# done the calculations
T1, Evib1, Fvib1, Svib1, Cvib1 = read_thermo( fthermo, ngeo )
nT, T, Evib, Fvib, Svib, Cvib = rearrange_thermo( T1, Evib1, Fvib1, Svib1, Cvib1, ngeo )


fEtot = "./Etot.dat"
thermodata = nT, T, Evib, Fvib, Svib, Cvib
TT, Fmin, celldmsminT, alphaT, a0, chi, aT, chi = fitFvib(fEtot,thermodata,minoptions={'gtol': 1e-7})

fig1 = simple_plot_xy(TT,Fmin,xlabel="T (K)",ylabel="Fmin (Ry/cell)")
fig2 = simple_plot_xy(TT,celldmsminT[:,0],xlabel="T (K)",ylabel="a_min (a.u.)")
fig3 = simple_plot_xy(TT,celldmsminT[:,2],xlabel="T (K)",ylabel="c_min (a.u.)")
fig4 = simple_plot_xy(TT,celldmsminT[:,2]/celldmsminT[:,0],xlabel="T (K)",ylabel="c/a")
fig5 = simple_plot_xy(TT,alphaT[:,0],xlabel="T (K)",ylabel="alpha_xx (1/K)")
```

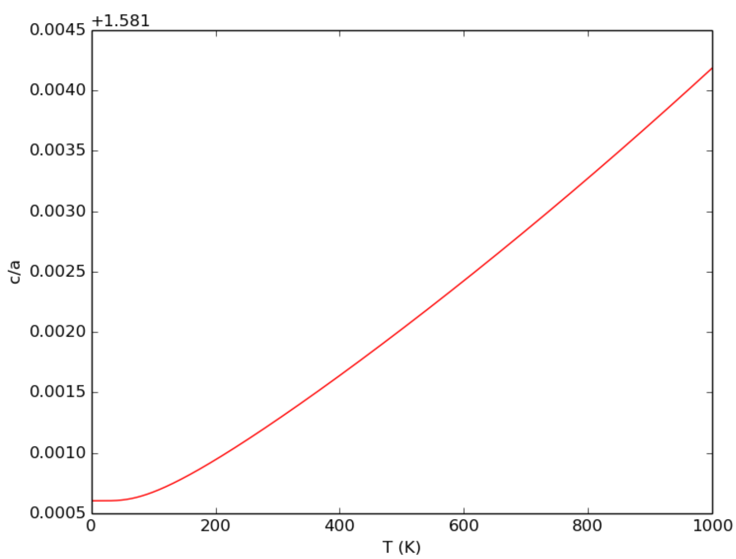
```
fig6 = simple_plot_xy(TT,alphaT[:,2],xlabel="T (K)",ylabel="alpha_zz (1/K)")

# write a(T) and c(T) on a file
write_celldmsT("celldmminT",T,celldmsminT,ibrav=4)
# write alpha_xx(T) and alpha_zz(T) on a file
write_alphaT("alphaT",T,alphaT,ibrav=4)

# Plot several quantities at T=998+1 K as an example
celldmsx, Ex = read_Etot(fEtot)  # since the fitFvib does not return Etot data, you␣
↪must read them from the original file
iT=998                     # this is the index of the temperatures array, not the␣
↪temperature itself
print("T= ",TT[iT]," (K)")
# 3D plot only with fitted energy (Etot+Fvib)
fig7 = plot_Etot(celldmsx,Ex=None,n=(5,0,5),nmesh=(50,0,50),fittype="quadratic",
↪ibrav=4,a=a0+aT[iT])
# 3D plot fitted energy and points
fig8 = plot_Etot(celldmsx,Ex+Fvib[iT],n=(5,0,5),nmesh=(50,0,50),fittype="quadratic",
↪ibrav=4,a=a0+aT[iT])
# 3D plot with fitted energy Fvib only
fig9 = plot_Etot(celldmsx,Ex=None,n=(5,0,5),nmesh=(50,0,50),fittype="quadratic",
↪ibrav=4,a=aT[iT])
# 2D contour plot with fitted energy (Etot+Fvib)
fig10 = plot_Etot_contour(celldmsx,nmesh=(50,0,50),fittype="quadratic",ibrav=4,
↪a=a0+aT[iT])
# 2D contour plot with fitted energy Fvib only
fig11 = plot_Etot_contour(celldmsx,nmesh=(50,0,50),fittype="quadratic",ibrav=4,
↪a=aT[iT])

# Save all plots
fig1.savefig("figure_1.png")
fig2.savefig("figure_2.png")
fig3.savefig("figure_3.png")
fig4.savefig("figure_4.png")
fig5.savefig("figure_5.png")
fig6.savefig("figure_6.png")
fig7.savefig("figure_7.png")
fig8.savefig("figure_8.png")
fig9.savefig("figure_9.png")
fig10.savefig("figure_10.png")
fig11.savefig("figure_11.png")
```

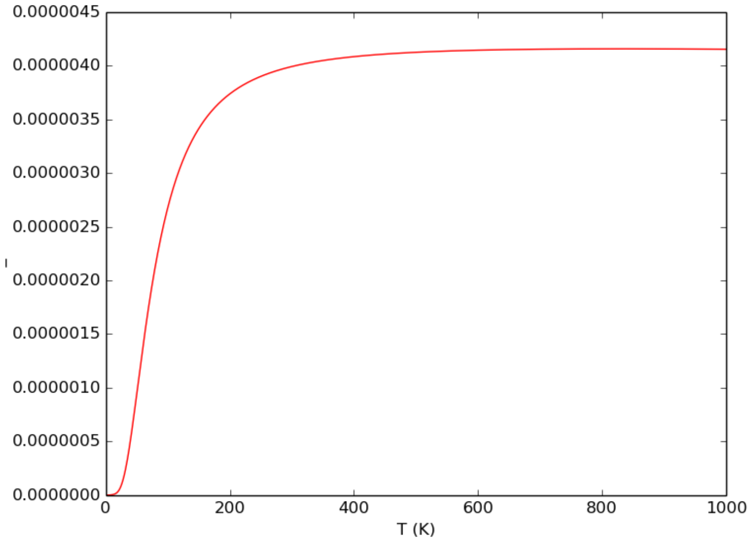
If everything went well, you should get the following plots:

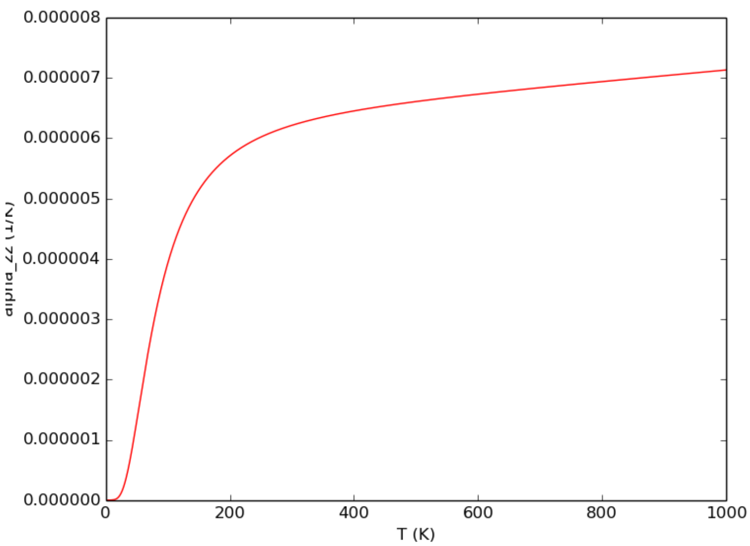

−1.62776e3
−0.01
−0.02
−0.03
−0.04
−0.05
−0.06
−0.07
−0.08
Fmin (Ry/cell)
0
200
400
600
800
1000
T (K)


5.160
5.155
5.150
5.145
5.140
5.135
a_min (a.u.)
0
200
400
600
800
1000
T (K)


8.18
8.17
8.16
8.15
8.14
8.13
8.12
c_min (a.u.)
0
200
400
600
800
1000
T (K)


+1.581
0.0045
0.0040
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
c/a
0
200
400
600
800
1000
T (K)





2.4 Computing quasi-static elastic constants

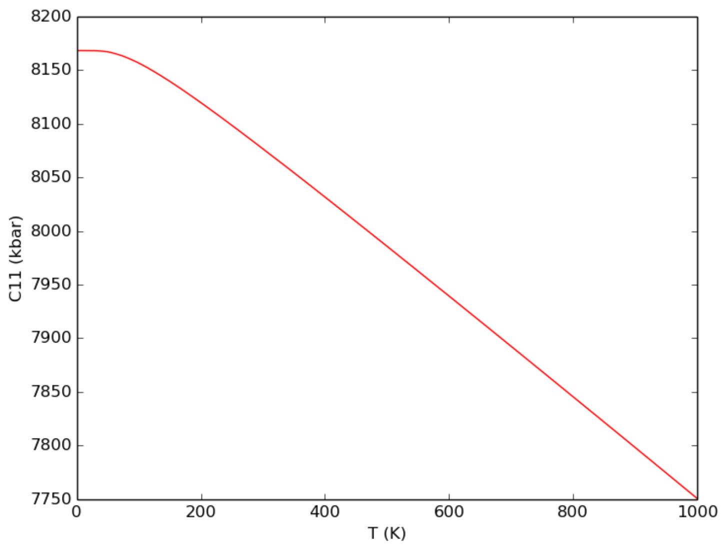
The following code example shows how to do a calculation of a quasi-static elastic tensor as a function of temperature for an hexagonal system. This kind of calculation requires that a quasi-harmonic calculation has already be done (as in example 6). Besides, the elastic constants for different (a, c) values must be available. To compute these elastic constants you can use for example the thermo_pw code [1].

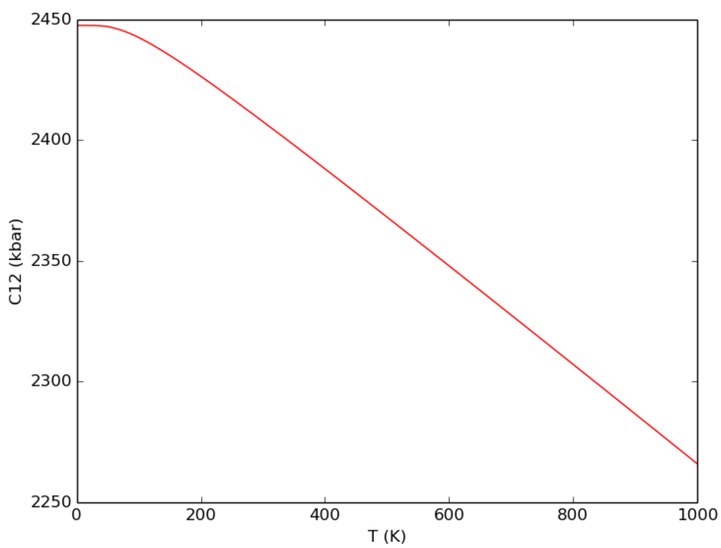
```
from pyqha import RY_KBAR
from pyqha import gen_TT, read_Etot, read_dos_geo, compute_thermo_geo, read_thermo,␣
↪rearrange_thermo, fitFvib, write_celldmsT, write_alphaT
from pyqha import simple_plot_xy, multiple_plot_xy
from pyqha import read_elastic_constants_geo, write_C_geo, write_CT, rearrange_Cx,␣
↪fitCxx, fitCT
```

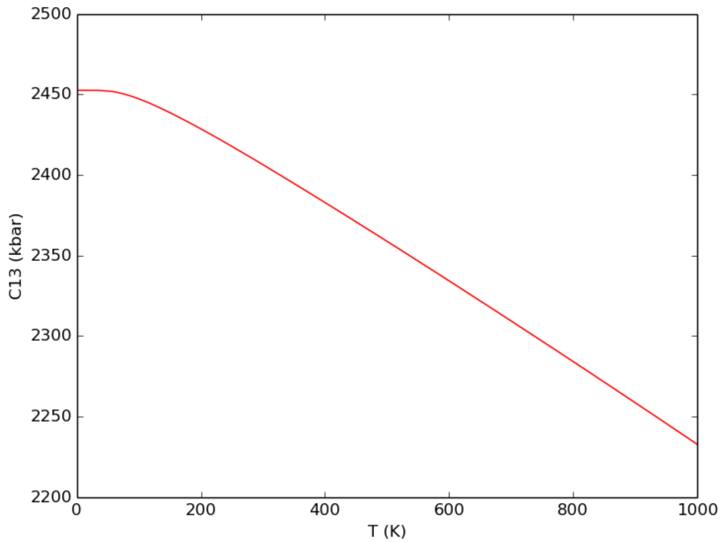
[1] http://qeforge.qe-forge.org/gf/project/thermo_pw/

```
fEtot = "./Etot.dat"
celldmsx, Ex = read_Etot(fEtot)  # since the fitFvib does not return Etot data, you␣
↪must read them from the original file

# this part is for calculating the thermodynamic properties from the dos
fdos="dos_files/output_dos.dat.g"        # base name for the dos files (numbers will␣
↪be added as postfix)
fthermo = "thermo"                   # base name for the output files (numbers will be␣
↪added as postfix)

ngeo = 25        # this is the number of volumes for which a dos has been calculated

#TT =gen_TT(1,1000)        # generate the numpy array of temperatures for which the␣
↪properties will be calculated
#T, Evib, Fvib, Svib, Cvib, ZPE, modes = compute_thermo_geo(fdos,fthermo,ngeo,TT)
#nT = len(T)

# Alternatively, read the thermodynamic data from files if you have already
# done the calculations
T1, Evib1, Fvib1, Svib1, Cvib1 = read_thermo( fthermo, ngeo )
nT, T, Evib, Fvib, Svib, Cvib = rearrange_thermo( T1, Evib1, Fvib1, Svib1, Cvib1,␣
↪ngeo )

fEtot = "./Etot.dat"
thermodata = nT, T, Evib, Fvib, Svib, Cvib
TT, Fmin, celldmsminT, alphaT, a0, chi, aT, chi = fitFvib(fEtot,thermodata,typeEtot=
↪"quartic",typeFvib="quartic",defaultguess=[5.12374914,0.0,8.19314311,0.0,0.0,0.0])


# Now start the quasi-static calculation
fC = "./elastic_constants/output_el_cons.g"

# Read the elastic constants and compliances from files
Cx, Sx = read_elastic_constants_geo(fC, ngeo)
Cxx = rearrange_Cx(Cx,ngeo)      # rearrange them in the proper order for fitting

# Optionally save them
write_C_geo(celldmsx, Cxx, ibrav=4, fCout="./elastic_constants/")

# Fit the elastic constants as a function of celldmsx
aC, chiC = fitCxx(celldmsx, Cxx, ibrav=4,typeC="quadratic")

T, CT = fitCT(aC, chiC, TT, celldmsminT, ibrav=4, typeC="quadratic")

write_CT(TT,CT,fCout="./elastic_constants/")

fig1 = simple_plot_xy(TT,CT[:,0,0],xlabel="T (K)",ylabel="C11 (kbar)")
fig2 = simple_plot_xy(TT,CT[:,0,1],xlabel="T (K)",ylabel="C12 (kbar)")
fig3 = simple_plot_xy(TT,CT[:,0,2],xlabel="T (K)",ylabel="C13 (kbar)")
fig4 = simple_plot_xy(TT,CT[:,2,2],xlabel="T (K)",ylabel="C33 (kbar)")
fig1.savefig("figure_1.png")
fig2.savefig("figure_2.png")
fig3.savefig("figure_3.png")
fig4.savefig("figure_4.png")


# plot now 4 elastic constants in the same plot
```

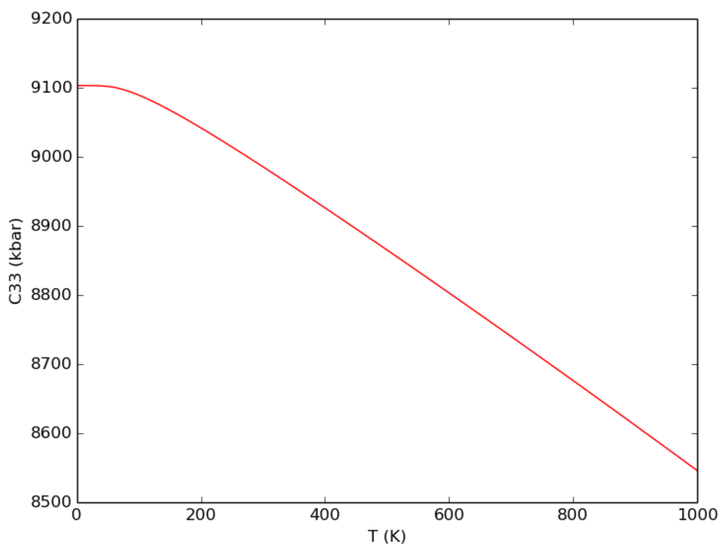
```
import numpy as np
pCxx = np.zeros((len(T),4))
pCxx[:,0] = CT[:,0,0]
pCxx[:,1] = CT[:,0,1]
pCxx[:,2] = CT[:,0,2]
pCxx[:,3] = CT[:,2,2]
Clabels = ["C11","C12","C13","C33"]
fig5 = multiple_plot_xy(T,pCxx,xlabel="T (K)",ylabel="Cxx (kbar)",labels=Clabels)
fig5.savefig("figure_5.png")
```



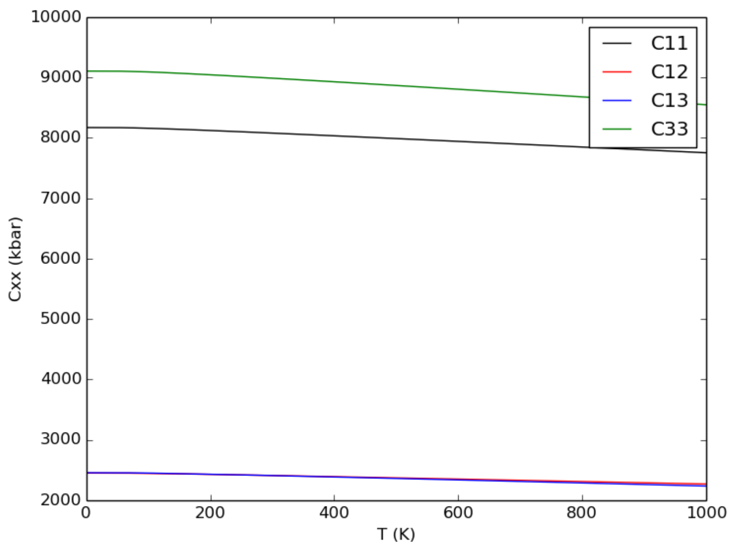




C13 (kbar)
2500
2450
2400
2350
2300
2250
2200
0
200
400
600
800
1000
T (K)



C33 (kbar)
9200
9100
9000
8900
8800
8700
8600
8500
0
200
400
600
800
1000
T (K)



2.5 Numerical issues

It is important to realize that the practical application of the quasi-harmonic approximation relies on fitting and minimizing the free energy as a function of volume, lattice parameters and temperature, ultimately on numerical methods. `pyqha` uses numpy and scipy functions to this aim. The user must select the best methods/options for the specific system under investigation and it is always better to test, test, test...

The following example shows how different methods/options may lead to different sets of results. Sometimes the differences are within the target numerical precision. Sometimes the results are simply wrong because of an improper choice of the methods/options.

First, lets compute some example results for a hypotetical hexagonal system, using total energy and phonon DOS for different values of (a, c) lattice parameters. We use all default values of the function `fitFvib()`, i.e. a quadratic polynomial for fitting the total energies, a quadratic polynomial for fitting the vibrational energies, BFGS algorithm for minimization with default options (see the documentation of scipy.optimize.minimize for more details).

```
from pyqha import RY_KBAR
from pyqha import gen_TT, read_Etot, read_dos_geo, compute_thermo_geo, read_thermo,␣
↪rearrange_thermo, fitFvib, write_celldmsT, write_alphaT
from pyqha import simple_plot_xy, plot_Etot, plot_Etot_contour, multiple_plot_xy
import numpy as np

# this part is for calculating the thermodynamic properties from the dos
fdos="dos_files/output_dos.dat.g"        # base name for the dos files (numbers will␣
↪be added as postfix)
fthermo = "thermo"                  # base name for the output files (numbers will be␣
↪added as postfix)

ngeo = 25         # this is the number of volumes for which a dos has been␣
↪calculated

#TT =gen_TT(1,2000)        # generate the numpy array of temperatures for which the␣
↪properties will be calculated
#T, Evib, Fvib, Svib, Cvib, ZPE, modes = compute_thermo_geo(fdos,fthermo,ngeo,TT)
#nT = len(T)
```

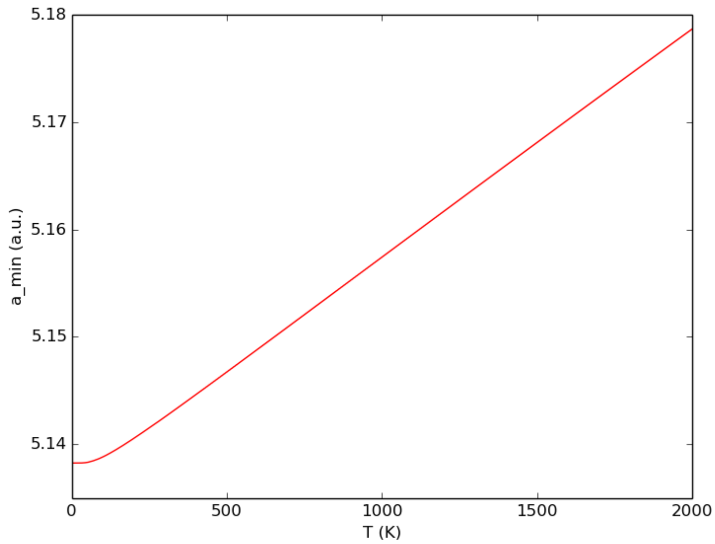
```
# Alternatively, read the thermodynamic data from files if you have already
# done the calculations
T1, Evib1, Fvib1, Svib1, Cvib1 = read_thermo( fthermo, ngeo )
nT, T, Evib, Fvib, Svib, Cvib = rearrange_thermo( T1, Evib1, Fvib1, Svib1, Cvib1,␣
↪ngeo )


fEtot = "./Etot.dat"
thermodata = nT, T, Evib, Fvib, Svib, Cvib

# Fit and minimize with default options, quadratic polynomials for both Etot and Fvib,
↪ minimization method="BFGS", minoptions={'gtol': 1e-5}
#TT, Fmin, celldmsminT, alphaT, a0, chi, aT, chiT = fitFvib(fEtot,thermodata)
res1 = fitFvib(fEtot,thermodata)
fig1 = simple_plot_xy(res1[0],res1[2][:,0],xlabel="T (K)",ylabel="a_min (a.u.)")
fig2 = simple_plot_xy(res1[0],res1[2][:,2],xlabel="T (K)",ylabel="c_min (a.u.)")
fig3 = simple_plot_xy(res1[0],res1[3][:,0],xlabel="T (K)",ylabel="alpha_xx (1/K)")
fig4 = simple_plot_xy(res1[0],res1[3][:,2],xlabel="T (K)",ylabel="alpha_zz (1/K)")
```

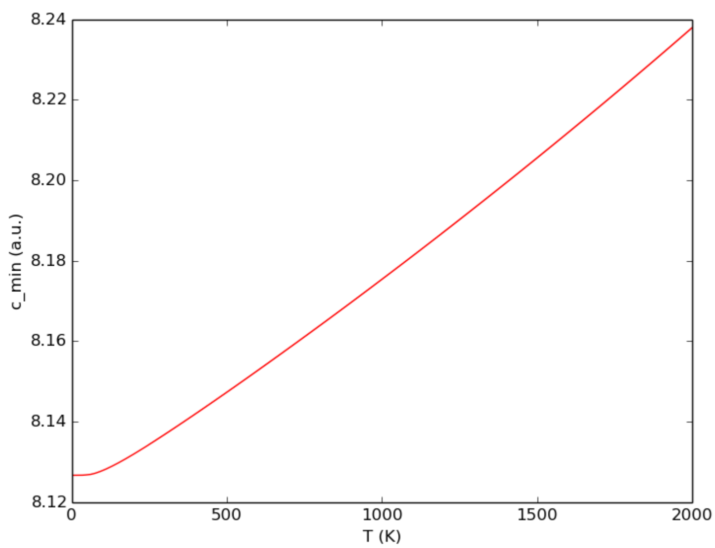
Running the above code and observing the results (not shown here), you should notice that the thermal expansions present some spikes. These quantities are obtained as numerical derivatives of the lattice parameters and are thus more sensitive to any numerical noise.

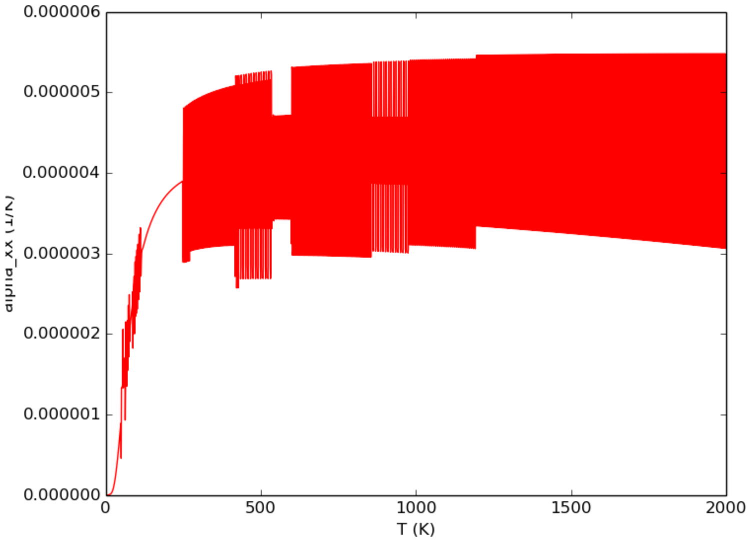
Let's see what happens if we use different polynomial forms for fitting, but the same minimization method.

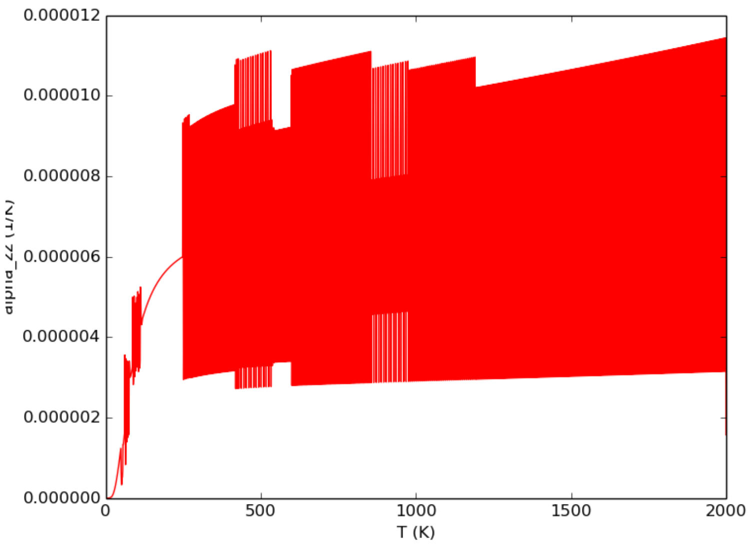
```
res2 = fitFvib(fEtot,thermodata,minoptions={'gtol': 1e-7})

# Fit the quartic polynomial for Etot and quadratic for Fvib, default minimization␣
↪method="BFGS", minoptions={'gtol': 1e-5}
res3 = fitFvib(fEtot,thermodata,typeEtot="quartic",minoptions={'gtol': 1e-7})

# Fit the quartic polynomial for Etot and quartic for Fvib, default minimization␣
↪method="BFGS", minoptions={'gtol': 1e-5}
res4 = fitFvib(fEtot,thermodata,typeEtot="quartic",typeFvib="quartic",minoptions={
↪'gtol': 1e-7})

# plot together the 3 resulting lattice parameters and thermal expansions
y = np.zeros((len(res1[0]),3))
y[:,0] = res2[2][:,0]
y[:,1] = res3[2][:,0]
y[:,2] = res4[2][:,0]
fig5 = multiple_plot_xy(res1[0],y,xlabel="T (K)",ylabel="a_min (a.u.)",labels=[
↪"quad+quad fit","quart+quad fit","quart+quart fit"])
y[:,0] = res2[2][:,2]
y[:,1] = res3[2][:,2]
y[:,2] = res4[2][:,2]
fig6 = multiple_plot_xy(res1[0],y,xlabel="T (K)",ylabel="c_min (a.u.)",labels=[
↪"quad+quad fit","quart+quad fit","quart+quart fit"])
y[:,0] = res2[3][:,0]
y[:,1] = res3[3][:,0]
y[:,2] = res4[3][:,0]
fig7 = multiple_plot_xy(res1[0],y,xlabel="T (K)",ylabel="alpha_xx (1/K)",labels=[
↪"quad+quad fit","quart+quad fit","quart+quart fit"])
y[:,0] = res2[3][:,2]
y[:,1] = res3[3][:,2]
y[:,2] = res4[3][:,2]
```

The results of the above code are shown here:







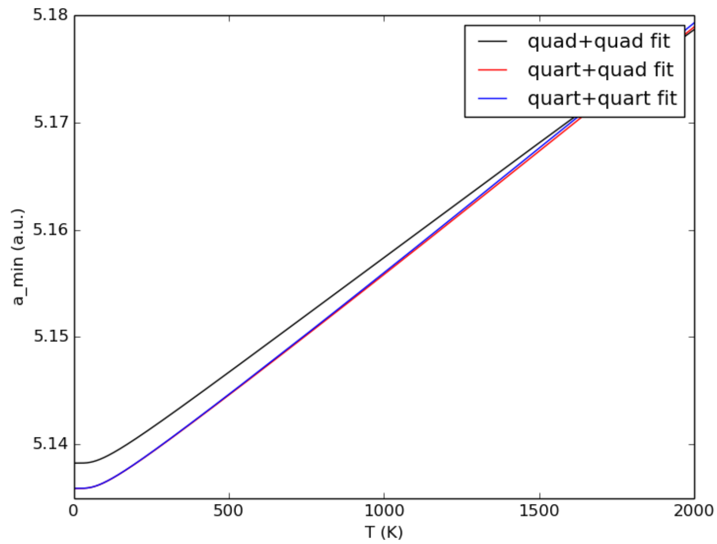


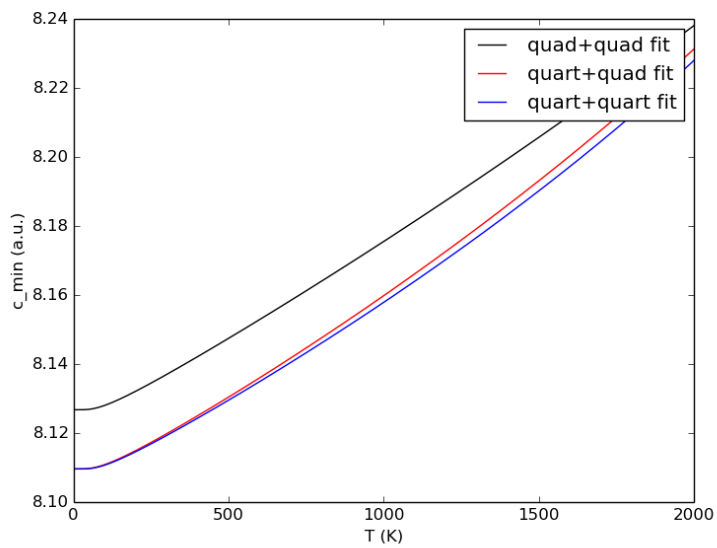
You can see that the fit with quadratic polynomials for both Etot and Fvib (quad+quad) gives a slightly different result at 0 K, with a difference of the order of 0.2%. Some differences remain even when using quadratic/quartic or quartic/quartic polynomials. In general, a quartic polynomial is expected to provide a better fit, but care must be paid to avoid overfitting and the best choice also depends on the shape of your energy surface.

Next, you try to minimize with gtol=1e-3 (gradient norm must be less than gtol before successful termination, see the documentation of scipy.optimize.minimize), same method.

```
fig8 = multiple_plot_xy(res1[0],y,xlabel="T (K)",ylabel="alpha_zz (1/K)",labels=[
↪"quad+quad fit","quart+quad fit","quart+quart fit"])


res5 = fitFvib(fEtot,thermodata,method="BFGS",minoptions={'gtol': 1e-7})

res6 = fitFvib(fEtot,thermodata,method="BFGS",minoptions={'gtol': 1e-8})
```

As you can see in the following figures, this leads to wrong results:

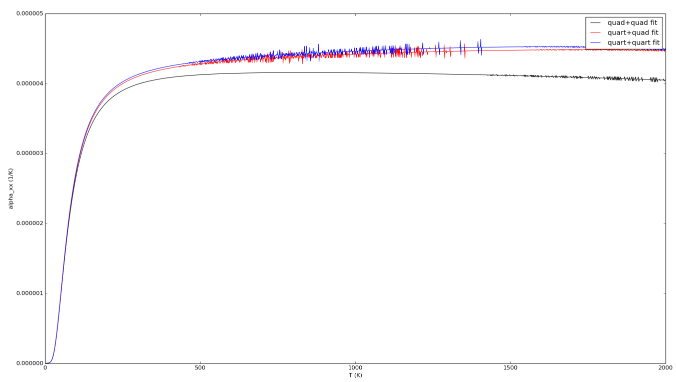


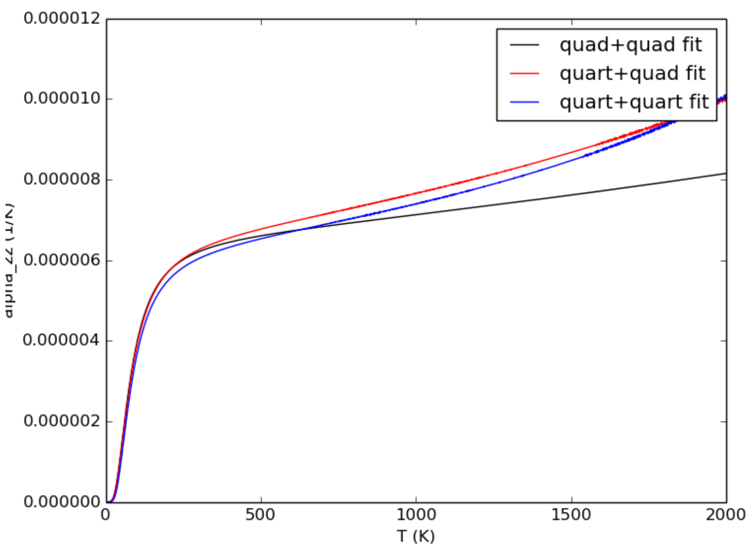


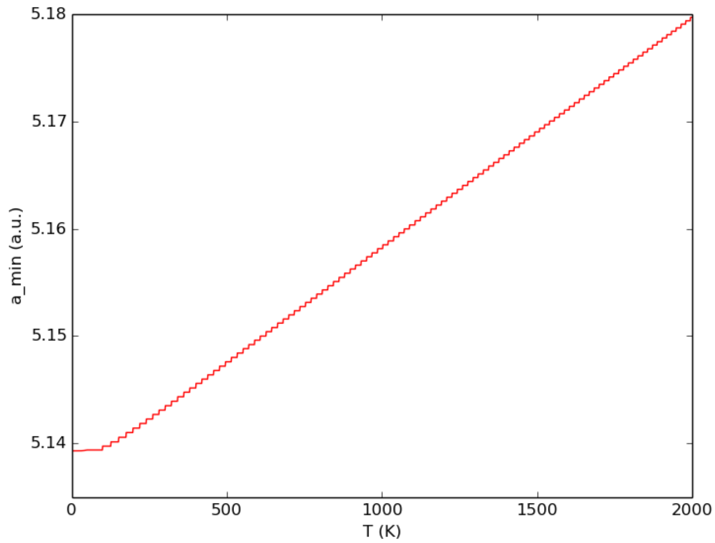
If on the contrary you increase the default gtol=1e-5 value, you can the following results:

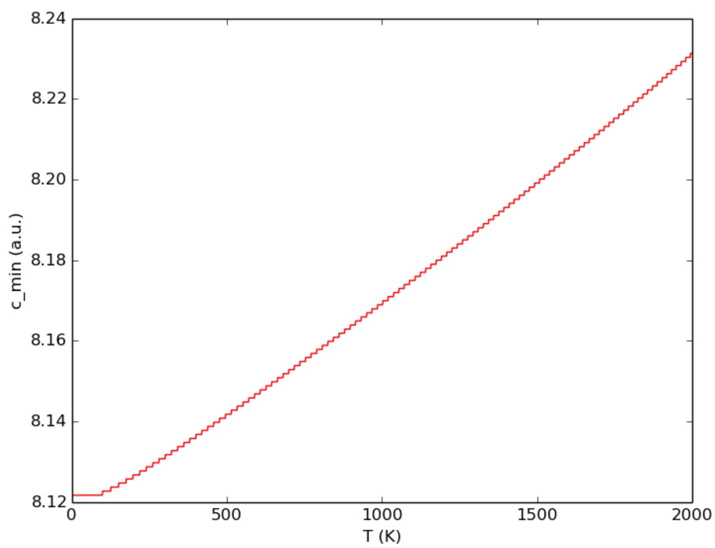
```
# plot together the 3 resulting thermal expansions
y[:,0] = res2[2][:,0]
y[:,1] = res5[2][:,0]
y[:,2] = res6[2][:,0]
fig13 = multiple_plot_xy(res1[0],y,xlabel="T (K)",ylabel="a_min (a.u.)",labels=["1e-6
↪","1e-7","1e-8"])
y[:,0] = res2[2][:,2]
y[:,1] = res5[2][:,2]
y[:,2] = res6[2][:,2]
fig14 = multiple_plot_xy(res1[0],y,xlabel="T (K)",ylabel="c_min (a.u.)",labels=["1e-6
↪","1e-7","1e-8"])
y[:,0] = res2[3][:,0]
```

```
y[:,1] = res5[3][:,0]
y[:,2] = res6[3][:,0]
fig15 = multiple_plot_xy(res1[0],y,xlabel="T (K)",ylabel="alpha_xx (1/K)",labels=["1e-
↪6","1e-7","1e-8"])
y[:,0] = res2[3][:,2]
y[:,1] = res5[3][:,2]
y[:,2] = res6[3][:,2]
fig16 = multiple_plot_xy(res1[0],y,xlabel="T (K)",ylabel="alpha_zz (1/K)",labels=["1e-
↪6","1e-8","1e-8"])


# The minimum shifts with temperatue, so does the quality of the fit (for example the␣
↪chi^2)
celldmsx, Ex = read_Etot(fEtot)  # since the fitFvib does not return Etot data, you␣
↪must read them from the original file
```





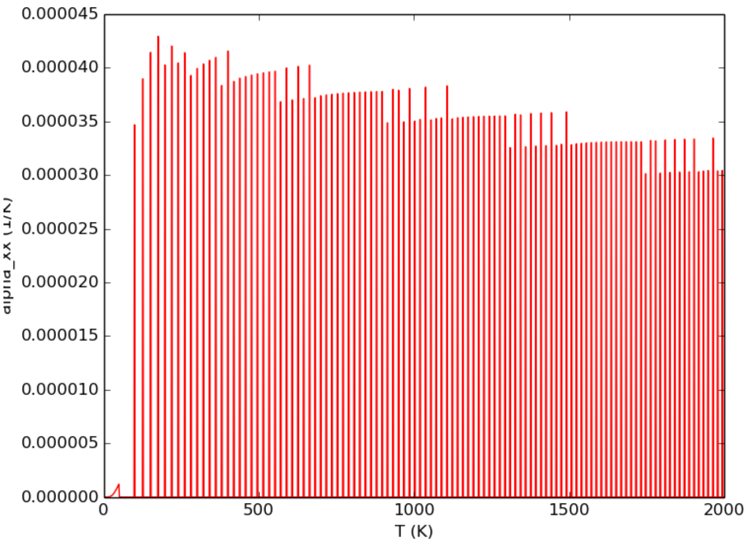




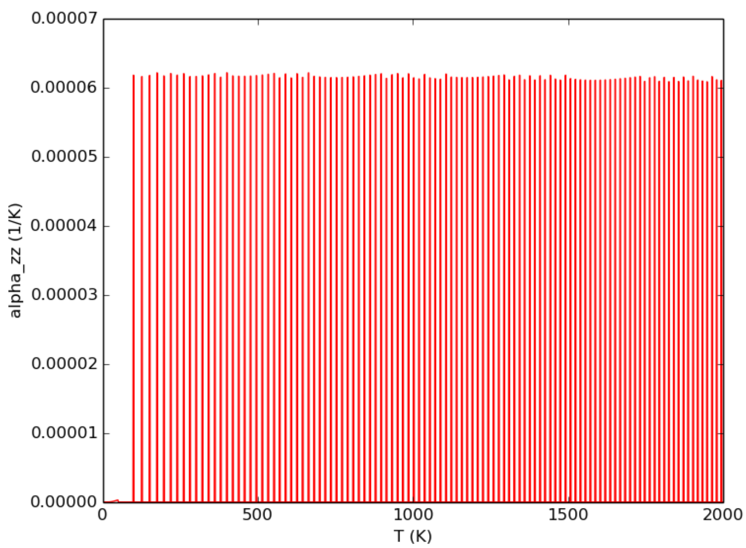
The difference in the lattice parameters is very limited and can hardly be seen, but it is more evident in the thermal expansions. In the latters, notice how the spikes are reduced when decreasing gtol.

Let's try now a different minimization algorithm, the common coniugate-gradient method (CG) as implemented in scipy.optimize.minimize, with default options and quadratic polynomial for fitting.

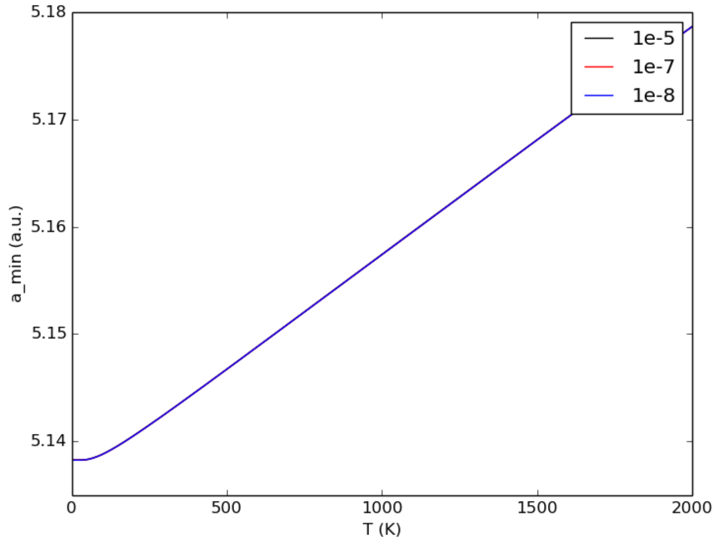
```
# 2D contour plot with fitted energy (Etot+Fvib)
fig21 = plot_Etot_contour(celldmsx,nmesh=(50,0,50),fittype="quartic",ibrav=4,
↪a=res3[4]+res3[6][iT])
iT=1998                   # this is the index of the temperatures array, not the␣
↪temperature itself
print("T= ",res3[0][iT]," (K)")
# 2D contour plot with fitted energy (Etot+Fvib)
fig22 = plot_Etot_contour(celldmsx,nmesh=(50,0,50),fittype="quartic",ibrav=4,
↪a=res3[4]+res3[6][iT])
```

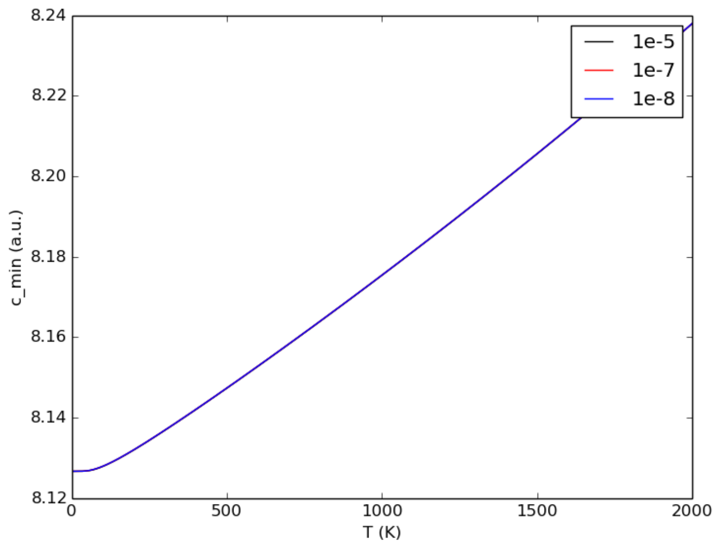


alpha_xx (1/K)
0.000045
0.000040
0.000035
0.000030
0.000025
0.000020
0.000015
0.000010
0.000005
0.000000
0
500
1000
1500
2000
T (K)



alpha_zz (1/K)
0.00007
0.00006
0.00005
0.00004
0.00003
0.00002
0.00001
0.00000
0
500
1000
1500
2000
T (K)



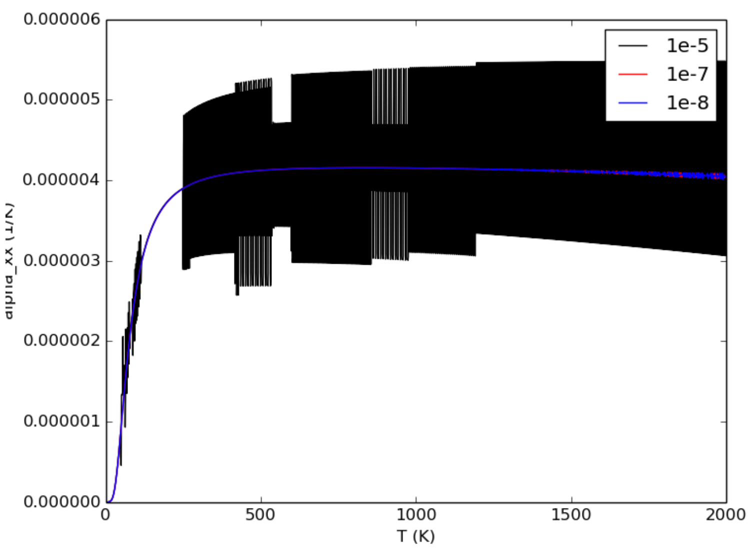


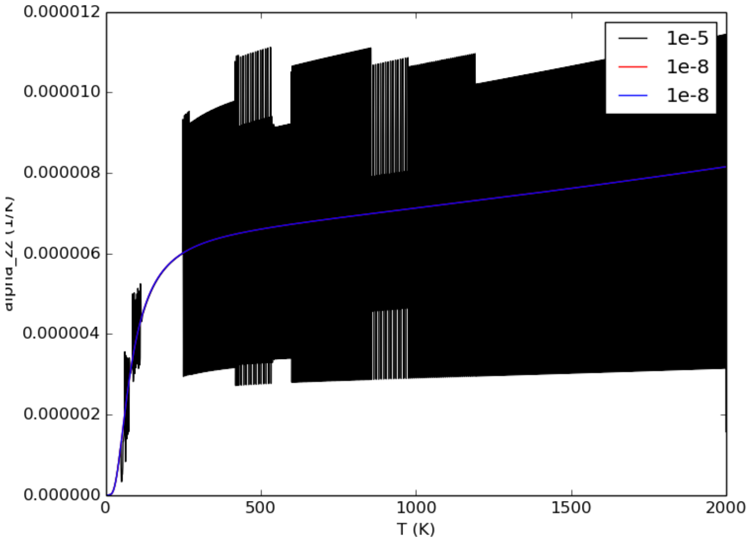
In this case the thermal expansions show much more noise than previously. It is thus clear that care must be paid to properly choose the fitting/minimizations methods and options.

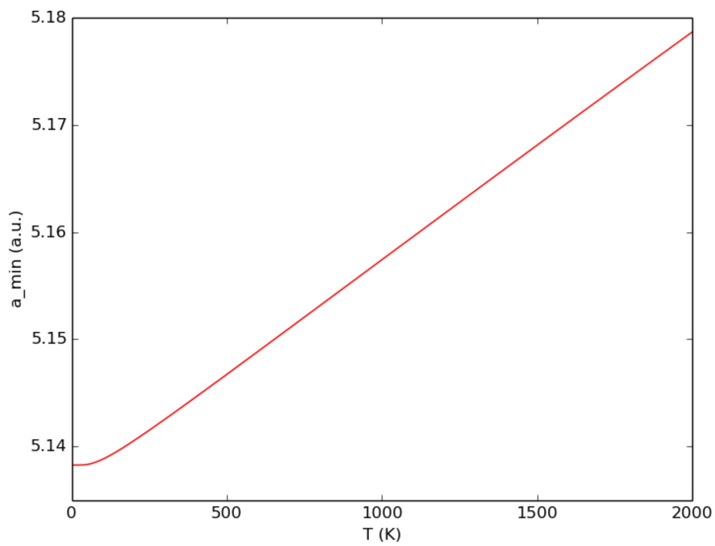
Finally, let's see another numerical issue in quasi-harmonic calculations which is illustrated in the following code and figure:

```
# Save all plots
fig1.savefig("figure_1.png")
fig2.savefig("figure_2.png")
fig3.savefig("figure_3.png")
fig4.savefig("figure_4.png")
fig5.savefig("figure_5.png")
fig6.savefig("figure_6.png")
fig7.savefig("figure_7.png")
```

```
fig8.savefig("figure_8.png")
fig9.savefig("figure_9.png")
fig10.savefig("figure_10.png")
fig11.savefig("figure_11.png")
fig12.savefig("figure_12.png")
fig13.savefig("figure_13.png")
fig14.savefig("figure_14.png")
fig15.savefig("figure_15.png")
fig16.savefig("figure_16.png")
fig17.savefig("figure_17.png")
fig18.savefig("figure_18.png")
fig19.savefig("figure_19.png")
fig20.savefig("figure_20.png")
fig21.savefig("figure_21.png")
fig22.savefig("figure_22.png")
fig23.savefig("figure_23.png")
```







The first two figures above show iso-contour lines for the $E_{tot}(a,c) + F_{vib}(a,c)$ surface at T=1 K and T=1999 K. You can see that the minimum is shifting as expected because of thermal expansion (usually positive) and as a consequence it becomes closer to the boundary of the chosen (a, c) grid. It is important to check that the minimun does not get too close to the boundary in order to avoid a serious decrease of the fit accuracy. In any case, the χ^2 of the fitting procedure is always slightly changing (usually increasing) with temperature, as shown in the last figure.

CHAPTER

THREE

PYQHA PACKAGE

3.1 Module contents

The following functions are available from `pyqha` module and are the most common ones for the end user.

`pyqha.fitEtot.`**`fitEtot`**(*fin*, *out=True*, *ibrav=4*, *fittype='quadratic'*, *guess=None*, *method='BFGS'*, *minoptions={}*)

This function reads the file *fin* containing the energies as a function of the lattice parameters $E(a, b, c)$ and fits them with a quartic (*fittype="quartic"*) or quadratic (*fittype="quadratic"*) polynomial. Then it finds the minimun energy and the corresponding lattice parameters. ibrav is the Bravais lattice, guess is an initial guess for the minimization. Depending on ibrav, a different number of lattice parameters is considered. It prints fitting results on the screen (which can be redirected to *stdout*) if *out=True*. It returns the lattice parameters and energies as in the input file *fin*, the fitted coefficients of the polynomial, the corresponding χ^2, the lattice parameters at the minimum and the minimun energy.

Note: for cubic systems use fitEtotV instead.

Advanced input parameters:

guess, an initial guess for the minimization. It is a 6 elements list [a,b,c,0,0,0].

method, the method to be used in the minimization procedure, as in the scipy.optimize.minimize. See its documentation for details.

minoptions, a dictionary with additional options for the minimization procedure, as in the scipy.optimize.minimize. See its documentation for details.

`pyqha.fitEtot.`**`fitEtotV`**(*fin*, *fout=None*)

This function reads $E(V)$ data from the input file *fin*, fits them with a Murnaghan EOS, prints the results on the *stdout* and write them in the file "fout". It returns the volumes and energies read from the input file, the fitted coefficients of the EOS and the corresponding χ^2.

`pyqha.thermo.`**`compute_thermo`**(*E*, *dos*, *TT*)

This function computes the vibrational energy, Helmholtz energy, entropy and heat capacity in the harmonic approximation from the input numpy arrays *E* and *dos* containing the phonon DOS(E). The calculation is done over a set of temperatures given in input as a numpy array *TT*. It also computes the number of phonon modes obtained from the input DOS (which must be approximately equal to $3*N$, with *N* the number of atoms per cell) and the ZPE. The input energy and dos are expected to be in 1/cm-1. It returns numpy arrays for the following quantities (in this order): temperatures, vibrational energy, Helmholtz energy, entropy, heat capacity. Plus it returns the ZPE and number of phonon modes obtained from the input DOS.

`pyqha.thermo.`**`compute_thermo_geo`**(*fin*, *fout=None*, *ngeo=1*, *TT=array([1])*)

This function reads the input dos file(s) from *fin+i*, with *i* a number from 1 to *ngeo* + 1 and computes vibrational energy, Helmholtz energy, entropy and heat capacity in the harmonic approximation. Then writes the output on file(s) if fout!=None. Output file(s) have the following format:

T	E_{vib}	F_{vib}	S_{vib}	C_{vib}
1	...	...	...	...

and are names *fout* +1, *fout* +2,... for each geometry.

Returning values are (len(TT),ngeo) numpy matrices (T,gEvib,gFvib,gSvib,gCvib,gZPE,gmodes) containing the temperatures and the above mentioned thermodynamic functions as for example: Fvib[T,geo] -> Fvib at the temperature "T" for the geometry "geo"

`pyqha.thermo.`**`dos_integral`**(*E*, *dos*, *m=0*)

A function to compute the integral of an input phonon DOS (*dos*) with the 3/8 Simpson method. *m* is the moment of the integral, if $m > 0$ different moments can be calculated. For example, with $m = 0$ (default) it returns the number of modes from the dos, with $m = 1$ it returns the ZPE. The input energy (*E*) and phonon DOS (*dos*) are expected to be in cm^{-1}.

`pyqha.thermo.`**`gen_TT`**(*Tstart=1*, *Tend=1000*, *Tstep=1*)

A simple function to generate a numpy array of temperatures, starting from *Tstart* and ending to *Tend* (or the closest *T<Tend* accorinding to the *Tstep*) with step *Tstep* .

`pyqha.thermo.`**`rearrange_thermo`**(*T*, *Evib*, *Fvib*, *Svib*, *Cvib*, *ngeo=1*)

This function just rearranges the order of the elements in the input matrices The first index of the returning matrices *X* now gives all geometries at a given *T*, i.e. *X[0]* is the vector of the property *X* a *T=T[0,0]* . *X[0,0]* for the first geometry, *X[0,1]* the second geometry and so on.

`pyqha.fitFvib.`**`fitFvib`**(*fEtot, thermodata, ibrav=4, typeEtot='quadratic', typeFvib='quadratic', defaultguess=[0.0, 0.0, 0.0, 0.0, 0.0, 0.0], method='BFGS', minoptions={}, splinesoptions=None*)

This function computes quasi-harmonic quantities from the $E_{tot}(a, b, c) + F_{vib}(a, b, c, T)$ as a function of temperature with Murnaghan's EOS. $E_{tot}(a, b, c)$ is read from the *fin* file. $F_{vib}(a, b, c, T)$ are given in *thermodata* which is a list containing the number of temperatures (*nT*) for which the calculations are done and the numpy matrices for temperatures, vibrational energy, Helmholtz energy, entropy and heat capacity. All these quantities are for each (a,b,c) as in *fin* file. The real number of lattice parameters depends on *ibrav*, for example for hexagonal systems (*ibrav=4*) you have only (a,c) values. *ibrav* identifies the Bravais lattice, as in Quantum Espresso.

The function fits $E_{tot}(a, b, c) + F_{vib}(a, b, c, T)$ with a quadratic or quartic polynomial (as defined by *typeEtot* and *typeFvib*) at each temperature in *thermodata* and then stores the fitted coefficients. Note that you can chose a different polynomial type for fitting $E_{tot}(a, b, c)$ and $F_{vib}(a, b, c)$. Then it computes the minimun energy $E_{tot} + F_{vib}$ and the corresponding lattice parameters $(a_{min}, b_{min}, c_{min})$ at each temperature by miniimizing the energy.

It also computes the linear thermal expansion tensor (as a numerical derivative of the minimum lattice parameters as a function of temperature (`compute_alpha()`).

It returns the numpy arrays and matrices containing the temperatures (as in input), the minimun energy, minimun lattice parameters, linear thermal expansions. It also returns the fitted coefficients and the χ^2 for $E_{tot}(a, b, c)$ only (at T=0 K) and the fitted coefficients and the χ^2 for $E_{tot}(a, b, c) + F_{vib}(a, b, c, T)$ at each temperature.

> **Warning:** The quantities in *thermodata* are usually obtained from `compute_thermo_geo()` or from `read_thermo()` and `rearrange_thermo()`. It is important that the order in the total energy file *fin* and the order of the thermodynamic data in *thermodata* is the same! See also *example6* and the tutorial.

Advanced input parameters:

guess, an initial guess for the minimization. It is a 6 elements list [a,b,c,0,0,0].

method, the method to be used in the minimization procedure, as in the scipy.optimize.minimize. See its documentation for details. Note that the methods which usually gives better results for quasi-harmonic calculations are the "BFGS" or Newton-CG". Default is "BFGS".

minoptions, a dictionary with additional options for the minimization procedure, as in the scipy.optimize.minimize. See its documentation for details. Note the the options are different for different methods.

splinesoptions, determines whether to use or not splines to reduce the noise on numerical derivatives (thermal expansions). If *splinesoptions*==None, use finete differences for derivatives, else use splines as implemented in scipy.interpolate (see documentation). In the latter case, *splinesoptions* must be a dictionary. This dictionary must contains the keywords *k0*, *s0*, *k1*, *s1*, *k2*, *s2* which are passed to `scipy.interpolate.splrep()`, one couple for each set of thermal expansions (alpha_xx, alpha_yy, alpha_zz). *k* is the order of the spline (default=3), s a smoothing condition (default=None). If *splinesoptions=={}* use the default options of `scipy.interpolate.splrep()` Note: use this option with care

`pyqha.fitFvib.`**`fitFvibV`**(*fin*, *thermodata*, *verbosity='low'*)

This function computes quasi-harmonic quantities from the $E_{tot}(V) + F_{vib}(V,T)$ as a function of temperature with Murnaghan's EOS. $E_{tot}(V)$ is read from the *fin* file. $F_{vib}(V,T)$ are given in *thermodata* which is a list containing the number of temperatures (*nT*) for which the calculations are done and the numpy matrices for temperatures, vibrational energy, Helmholtz energy, entropy and heat capacity. All these quantities are for each volume as in *fin* file.

The function fits $E_{tot}(V) + F_{vib}(V,T)$ with a Murnaghan's EOS at each temperature in *thermodata* and then stores the fitted coefficients. It also computes the volume thermal expansion as a numerical derivative of the minimum volume as a function of temperature (`compute_beta()`), the constant volume heat capacity at the minimum volume at each T (`compute_Cv()`) and the constant pression heat capacity (`compute_Cp()`).

It returns the numpy 1D arrays containing the temperatures (as in input), the minimun energy, minimun volume, bulk modulus, volume thermal expansion, constant volume and constant pressure heat capacities, one matrix with all fitted coefficients at each T and finally an array with the χ^2 at each T.

> **Warning:** The quantities in *thermodata* are usually obtained from `compute_thermo_geo()` or from `read_thermo()` and `rearrange_thermo()`. It is important that the order in the total energy file *fin* and the order of the thermodynamic data in *thermodata* is the same! See also *example5* and the tutorial.

`pyqha.fitC.`**`fitCT`**(*aC*, *chiC*, *T*, *minT*, *ibrav=4*, *typeC='quadratic'*)

This function calculates the elastic constants tensor *CT* as a function of temperatature in the quasi-static approximation. It takes in input *aC* and *chiC*, the fitted coefficients of the elastic constants as a function of (a, b, c) and the corresponding χ^2. It also takes in input an array of temperatures *T* and the corresponding lattice parameters *minT*, i.e. $(a_{min}, b_{min}, c_{min})$ from a previous quasi-harmonic calculations (as in example6). It also needs in input the Bravais lattice (*ibrav*) and the type of polynomial (*typeC*) used for fitting the input *aC*.

The function uses the coefficients *aC* to compute the elastic tensor at each temperature in the array *T* from the corresponding lattice parameters $(a_{min}, b_{min}, c_{min})$ in *minT*.

It returns the temperature array and the a matrix *CT* with all the elastic tensors at each T (*CT[i]* is the elastic constants matrix for the temperature *T[i]*)

> **Warning:** The coefficients *aC* must be the result of fitting the elastic constants over the same (a, b, c) grid used in the quasi-harmonic calculations corresponding to *minT* values! (See example7)

`pyqha.fitC.`**`fitCxx`**(*celldmsx*, *Cxx*, *ibrav=4*, *typeC='quadratic'*)

This function fits the elastic constant elements of *Cxx* as a function of the grid of lattice parameters (a, b, c). The real number of lattice parameters depends on *ibrav*, for example for hexagonal systems (*ibrav=4*) you have only (a,c) values. *ibrav* identifies the Bravais lattice, as in Quantum Espresso.

It returns a 6*6 matrix, each element *[i,j]* being the set of coefficients of the polynomial fit and another 6*6 matrix, each element *[i,j]* being the corresponding χ^2. If the chi squared is zero, the fitting procedure was NOT succesful

`pyqha.fitC.`**`rearrange_Cx`**(*Cx*, *ngeo*)

This function rearrange the input numpy matrix *Cx* into an equivalent matrix *Cxx* for fitting it. *Cx* is a $ngeo*6*6$ matrix, each *Cx[i]* is the 6*6 *C* matrix for a given geometry (*i*) *Cxx* is a Lmath:*6*6*ngeo* matrix, each *Cxx[i][j]* is a vector with all values for different geometries of the *Cij* elastic constant matrix element. For example, *Cxx[0,0]* is the vector with ngeo values of the *C11* elastic constant and so on.

3.2 Submodules

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

3.3 pyqha.constants module

Some useful standard constants for conversions and calculations.

3.4 pyqha.eos module

`pyqha.eos.`**`E_Murn`**(*V*, *a*)

As `E_MurnV()` but input parameters are given as a single list *a=[a0,a1,a2,a3]*.

`pyqha.eos.`**`E_MurnV`**(*V*, *a0*, *a1*, *a2*, *a3*)

This function implements the Murnaghan EOS (in a form which is best for fitting). Returns the energy at the volume *V* using the coefficients *a0,a1,a2,a3* from the equation:

$$a_0 - (a_2 * a_1)/(a_3 - 1.0)V a_2/a_3(a_1/V^{a_3})/(a_3 - 1.0) + 1.0)$$

`pyqha.eos.`**`H_Murn`**(*V*, *a*)

This function return

As `E_MurnV()` but input parameters are given as a single list *a=[a0,a1,a2,a3]* and it returns the pressure not the energy from the EOS.

`pyqha.eos.`**`P_Murn`**(*V*, *a*)

As `E_MurnV()` but input parameters are given as a single list *a=[a0,a1,a2,a3]* and it returns the pressure not the energy from the EOS.

`pyqha.eos.`**`calculate_fitted_points`**(*V*, *a*)

Calculates a denser mesh of E(V) points for plotting...

`pyqha.eos.`**`compute_Cp`**(*T*, *Cv*, *V*, *B0*, *beta*)

This function computes the isobaric heat capacity from the eq. Cp-Cv=... Not implemented yet.

`pyqha.eos.`**`compute_Cv`**(*T*, *Vmin*, *V*, *Cvib*)

This function computes the isocoric heat capacity as a function of temperature. From *Cvib*, which is a matrix with *Cvib(T,V)* as from the harmonic calculations determines the *Cv* at each temperature by linear interpolation between the values at the two volumes closest to Vmin(T). Vmin(T) is from the minimization of F(V,T) and *V* is the array of volumes used for it. Returns *Cv(T)*.

Not implemented yet.

`pyqha.eos.`**`compute_beta`**(*minT*)

This function computes the volumetric thermal expansion as a numerical derivative of the volume as a function of temperature V(T). This is obtained from the free energy minimization which should be done before.

`pyqha.eos.`**`fit_Murn`**(*V*, *E*)

This is the function for fitting with the Murnaghan EOS as a function of volume only.

The input variable V is an 1D array of volumes, E are the corresponding energies (or other analogous quantity to be fitted with the Murnaghan EOS.

`pyqha.eos.`**`print_eos_data`**(*x*, *y*, *a*, *chi*, *ylabel='Etot'*)

Print the data and the fitted results using the EOS. It can be used for different fitted quantities using the proper ylabel. ylabel can be "Etot", "Fvib", etc.

`pyqha.eos.`**`write_Etotfitted`**(*filename*, *x*, *y*, *a*, *chi*, *ylabel='E'*)

Write in filename the data and the fitted results using the EOS. It can be used for different fitted quantities using the proper ylabel. ylabel can be "Etot", "Fvib", etc.

3.5 pyqha.fitC module

`pyqha.fitC.`**`fitCT`**(*aC*, *chiC*, *T*, *minT*, *ibrav=4*, *typeC='quadratic'*)

This function calculates the elastic constants tensor *CT* as a function of temperatature in the quasi-static approximation. It takes in input *aC* and *chiC*, the fitted coefficients of the elastic constants as a function of (a, b, c) and the corresponding χ^2. It also takes in input an array of temperatures *T* and the corresponding lattice parameters *minT*, i.e. $(a_{min}, b_{min}, c_{min})$ from a previous quasi-harmonic calculations (as in example6). It also needs in input the Bravais lattice (*ibrav*) and the type of polynomial (*typeC*) used for fitting the input *aC*.

The function uses the coefficients *aC* to compute the elastic tensor at each temperature in the array *T* from the corresponding lattice parameters $(a_{min}, b_{min}, c_{min})$ in *minT*.

It returns the temperature array and the a matrix *CT* with all the elastic tensors at each T (*CT[i]* is the elastic constants matrix for the temperature *T[i]*)

> **Warning:** The coefficients *aC* must be the result of fitting the elastic constants over the same (a, b, c) grid used in the quasi-harmonic calculations corresponding to *minT* values! (See example7)

`pyqha.fitC.`**`fitCxx`**(*celldmsx*, *Cxx*, *ibrav=4*, *typeC='quadratic'*)

This function fits the elastic constant elements of *Cxx* as a function of the grid of lattice parameters (a, b, c). The real number of lattice parameters depends on *ibrav*, for example for hexagonal systems (*ibrav=4*) you have only (a,c) values. *ibrav* identifies the Bravais lattice, as in Quantum Espresso.

It returns a 6*6 matrix, each element *[i,j]* being the set of coefficients of the polynomial fit and another 6*6 matrix, each element *[i,j]* being the corresponding χ^2. If the chi squared is zero, the fitting procedure was NOT succesful

`pyqha.fitC.`**`rearrange_Cx`**(*Cx*, *ngeo*)

This function rearrange the input numpy matrix *Cx* into an equivalent matrix *Cxx* for fitting it. *Cx* is a $ngeo{*}6{*}6$ matrix, each *Cx[i]* is the 6*6 *C* matrix for a given geometry (*i*) *Cxx* is a Lmath:*6*6*ngeo* matrix, each *Cxx[i][j]* is a vector with all values for different geometries of the *Cij* elastic constant matrix element. For example, *Cxx[0,0]* is the vector with ngeo values of the *C11* elastic constant and so on.

3.6 pyqha.fitEtot module

`pyqha.fitEtot.`**`fitEtot`**(*fin*, *out=True*, *ibrav=4*, *fittype='quadratic'*, *guess=None*, *method='BFGS'*, *minoptions={}*)

This function reads the file *fin* containing the energies as a function of the lattice parameters $E(a, b, c)$ and fits them with a quartic (*fittype="quartic"*) or quadratic (*fittype="quadratic"*) polynomial. Then it finds the minimun energy and the corresponding lattice parameters. ibrav is the Bravais lattice, guess is an initial guess for the minimization. Depending on ibrav, a different number of lattice parameters is considered. It prints fitting results on the screen (which can be redirected to *stdout*) if *out=True*. It returns the lattice parameters and energies as in the input file *fin*, the fitted coefficients of the polynomial, the corresponding χ^2, the lattice parameters at the minimum and the minimun energy.

Note: for cubic systems use fitEtotV instead.

Advanced input parameters:

guess, an initial guess for the minimization. It is a 6 elements list [a,b,c,0,0,0].

method, the method to be used in the minimization procedure, as in the scipy.optimize.minimize. See its documentation for details.

minoptions, a dictionary with additional options for the minimization procedure, as in the scipy.optimize.minimize. See its documentation for details.

`pyqha.fitEtot.`**`fitEtotV`**(*fin*, *fout=None*)

This function reads $E(V)$ data from the input file *fin*, fits them with a Murnaghan EOS, prints the results on the *stdout* and write them in the file "fout". It returns the volumes and energies read from the input file, the fitted coefficients of the EOS and the corresponding χ^2.

3.7 pyqha.fitFvib module

`pyqha.fitFvib.`**`fitFvib`**(*fEtot, thermodata, ibrav=4, typeEtot='quadratic', typeFvib='quadratic', defaultguess=[0.0, 0.0, 0.0, 0.0, 0.0, 0.0], method='BFGS', minoptions={}, splinesoptions=None*)

This function computes quasi-harmonic quantities from the $E_{tot}(a, b, c) + F_{vib}(a, b, c, T)$ as a function of temperature with Murnaghan's EOS. $E_{tot}(a, b, c)$ is read from the *fin* file. $F_{vib}(a, b, c, T)$ are given in *thermodata* which is a list containing the number of temperatures (*nT*) for which the calculations are done and the numpy matrices for temperatures, vibrational energy, Helmholtz energy, entropy and heat capacity. All these quantities are for each (a,b,c) as in *fin* file. The real number of lattice parameters depends on *ibrav*, for example for hexagonal systems (*ibrav=4*) you have only (a,c) values. *ibrav* identifies the Bravais lattice, as in Quantum Espresso.

The function fits $E_{tot}(a, b, c) + F_{vib}(a, b, c, T)$ with a quadratic or quartic polynomial (as defined by *typeEtot* and *typeFvib*) at each temperature in *thermodata* and then stores the fitted coefficients. Note that you can chose a different polynomial type for fitting $E_{tot}(a, b, c)$ and $F_{vib}(a, b, c)$. Then it computes the minimun energy $E_{tot} + F_{vib}$ and the corresponding lattice parameters $(a_{min}, b_{min}, c_{min})$ at each temperature by miniimizing the energy.

It also computes the linear thermal expansion tensor (as a numerical derivative of the minimum lattice parameters as a function of temperature (`compute_alpha()`).

It returns the numpy arrays and matrices containing the temperatures (as in input), the minimun energy, minimun lattice parameters, linear thermal expansions. It also returns the fitted coefficients and the χ^2 for $E_{tot}(a, b, c)$ only (at T=0 K) and the fitted coefficients and the χ^2 for $E_{tot}(a, b, c) + F_{vib}(a, b, c, T)$ at each temperature.

> **Warning:** The quantities in *thermodata* are usually obtained from `compute_thermo_geo()` or from `read_thermo()` and `rearrange_thermo()`. It is important that the order in the total energy file *fin* and the order of the thermodynamic data in *thermodata* is the same! See also *example6* and the tutorial.

Advanced input parameters:

guess, an initial guess for the minimization. It is a 6 elements list [a,b,c,0,0,0].

method, the method to be used in the minimization procedure, as in the scipy.optimize.minimize. See its documentation for details. Note that the methods which usually gives better results for quasi-harmonic calculations are the "BFGS" or Newton-CG". Default is "BFGS".

minoptions, a dictionary with additional options for the minimization procedure, as in the scipy.optimize.minimize. See its documentation for details. Note the the options are different for different methods.

splinesoptions, determines whether to use or not splines to reduce the noise on numerical derivatives (thermal expansions). If *splinesoptions*==None, use finete differences for derivatives, else use splines as implemented in scipy.interpolate (see documentation). In the latter case, *splinesoptions* must be a dictionary. This dictionary must contains the keywords *k0*, *s0*, *k1*, *s1*, *k2*, *s2* which are passed to `scipy.interpolate.splrep()`, one couple for each set of thermal expansions (alpha_xx, alpha_yy, alpha_zz). *k* is the order of the spline (default=3), s a smoothing condition (default=None). If *splinesoptions==\{\}* use the default options of `scipy.interpolate.splrep()` Note: use this option with care

`pyqha.fitFvib.`**`fitFvibV`**`(`*fin*, *thermodata*, *verbosity='low'*`)`

This function computes quasi-harmonic quantities from the $E_{tot}(V) + F_{vib}(V,T)$ as a function of temperature with Murnaghan's EOS. $E_{tot}(V)$ is read from the *fin* file. $F_{vib}(V,T)$ are given in *thermodata* which is a list containing the number of temperatures (*nT*) for which the calculations are done and the numpy matrices for temperatures, vibrational energy, Helmholtz energy, entropy and heat capacity. All these quantities are for each volume as in *fin* file.

The function fits $E_{tot}(V) + F_{vib}(V,T)$ with a Murnaghan's EOS at each temperature in *thermodata* and then stores the fitted coefficients. It also computes the volume thermal expansion as a numerical derivative of the minimum volume as a function of temperature (`compute_beta()`), the constant volume heat capacity at the minimum volume at each T (`compute_Cv()`) and the constant pression heat capacity (`compute_Cp()`).

It returns the numpy 1D arrays containing the temperatures (as in input), the minimun energy, minimun volume, bulk modulus, volume thermal expansion, constant volume and constant pressure heat capacities, one matrix with all fitted coefficients at each T and finally an array with the χ^2 at each T.

> **Warning:** The quantities in *thermodata* are usually obtained from `compute_thermo_geo()` or from `read_thermo()` and `rearrange_thermo()`. It is important that the order in the total energy file *fin* and the order of the thermodynamic data in *thermodata* is the same! See also *example5* and the tutorial.

3.8 pyqha.fitfreqgrun module

`pyqha.fitfreqgrun.`**`fitfreq`**`(`*celldmsx*, *min0*, *inputfilefreq*, *ibrav=4*, *typefreq='quadratic'*, *compute_grun=False*`)`

An auxiliary function for fitting the frequencies. It returns a matrix of nq*modes frequencies obtained for the fitted polynomial (quadratic or quartic) at the minimun point min0. It also returns the weigths of each q point where the frequencies are available.

`pyqha.fitfreqgrun.`**`fitfreqxx`**`(`*celldmsx*, *freqxx*, *ibrav*, *out*, *typefreq*`)`

This function fits the frequencies in freqxx as a function of the grid of lattice parameters.

It returns a nq*modes matrix, whose element [i,j] is the set of coefficients of the polynomial fit and another nq*modes matrix, whose element [i,j] is the corresponding chi squared. If the chi squared is zero, the fitting procedure was NOT succesful

`pyqha.fitfreqgrun.`**`freqmin`**(*afreq*, *min0*, *nq*, *modes*, *ibrav*, *typefreq*)

This function calculate the frequencies from the fitted polynomials at the minimun point min0. afreq contains the fitted polynomial coefficients.

It returns a nq*modes matrix, whose element [i,j] is the fitted frequency

`pyqha.fitfreqgrun.`**`freqmingrun`**(*afreq*, *min0*, *nq*, *modes*, *ibrav*, *typefreq*)

This function calculate the frequencies and the gruneisen parameters from the fitted polynomials at the minimun point min0. afreq contains the fitted polynomial coefficients.

It returns a nq*modes matrix, whose element [i,j] is the fitted frequency In addition, it returns a nq*modes*6 with the Gruneisein parameters. Each element [i,j,k] is the the Gruneisein parameter at nq=i, mode=j and direction k (for example, in hex systems k=0 is a direction, k=2 is c direction, other are zero)

Note that the Gruneisein parameters are not multiplied for the lattice parameters

`pyqha.fitfreqgrun.`**`rearrange_freqx`**(*freqx*)

This function rearrange the input numpy matrix freqx into an equivalent matrix freqxx for the subsequent fitting. freqx is a ngeo*nq*modes matrix, each freqx[i] is the nq*modes freq matrix for a given geometry (i) freqxx is a nq*modes*ngeo matrix, each freqxx[i][j] is a vector with all values for different geometries of the frequencies at point q=i and mode=j. For example, freqxx[0][0] is the vector with ngeo values of the frequencies at the first q-point and first mode so on.

3.9 pyqha.fitutils module

`pyqha.fitutils.`**`expand_quadratic_to_quartic`**(*a*)

This function gets a vector of coefficients from a quadratic fit and turns it into a vector of coeffients as from a quartic fit (extra coeffients are set to zero).

`pyqha.fitutils.`**`fit_anis`**(*celldmsx*, *Ex*, *ibrav=1*, *out=False*, *type='quadratic'*, *ylabel='Etot'*)

An auxiliary function for handling fitting in the anisotropic case

`pyqha.fitutils.`**`fit_quadratic`**(*x*, *y*, *ibrav=4*, *out=False*, *ylabel='E'*)

This is the function for fitting with a quadratic polynomial

The most general fitting multidimensional quadratic polynomial for a triclinic system is: a1 + a2 x1 + a3 x1^2 + a4 x2 + a5 x2^2 + a6 x1*x2 + + a7 x3 + a8 x3^2 + a9 x1*x3 + a10 x2*x3 + + a11 x4 + a12 x4^2 + a13 x1*x4 + a14 x2*x4 + a15 x3*x4 + + a16 x5 + a17 x5^2 + a18 x1*x5 + a19 x2*x5 + a20 x3*x5 + a21 x4*x5 + a22 x6 + a23 x6^2 + a24 x1*x6 + a25 x2*x6 + a26 x3*x6 + a27 x4*x6 + a28 x5*x6

ONLY THE HEXAGONAL AND GENERAL CASE ARE IMPLEMENTED, more to be done

The input variable x is a matrix ngeo*6, where x[:,0] is the set of a values x[:,1] is the set of b values x[:,2] is the set of c values x[:,3] is the set of alpha values x[:,4] is the set of beta values x[:,5] is the set of gamma values

`pyqha.fitutils.`**`fit_quartic`**(*x*, *y*, *ibrav=4*, *out=False*, *ylabel='E'*)

This is the function for fitting with a quartic polynomial

The most general fitting multidimensional quadratic polynomial for a triclinic system is:

ONLY THE HEXAGONAL CASE IS IMPLEMENTED, more to be done

The input variable x is a matrix ngeo*6, where x[:,0] is the set of a values x[:,1] is the set of b values x[:,2] is the set of c or c/a values x[:,3] is the set of alpha values x[:,4] is the set of beta values x[:,5] is the set of gamma values

`pyqha.fitutils.`**`print_data`**(*x*, *y*, *results*, *A*, *ibrav*, *ylabel='E'*)
This function prints the data and the fitted results ylabel can be "E", "Fvib", "Cxx", etc. so that can be used for different fitted quantities

`pyqha.fitutils.`**`print_polynomial`**(*a*, *ibrav=4*)
This function prints the fitted polynomial, either quartic or quadratic

3.10 pyqha.gruneisen1D module

`pyqha.gruneisen1D.`**`compute_grun`**(*ngeo*, *celldmsx*, *inputfilefreq*, *ibrav=4*, *ext=False*)
Read the frequencies for all geometries where the gruneisen parameters must be calculated. This depends on the direction (along a, along c, etc.) According to the direction chosen, start,stop,step must be given to loop over all geometries as listed in the file containing the energies

More work to do: entend to other ibrav types, etc.

`pyqha.gruneisen1D.`**`compute_grun_along_one_direction`**(*nq*, *modes*, *ngeo*, *cgeo*, *celldmsx*, *freqgeo*, *rangegeo*, *xindex=0*)
Compute the Gruneisen parameters along one direction. This function uses a 1-dimensional polynomial of fourth degree to fit the frequencies along a certain direction (along a and c axis in hexagonal systems for example).

`pyqha.gruneisen1D.`**`find_geocenters`**(*ngeo*)
Find the center geometries. Remember indexex in lists starts from 0...

3.11 pyqha.minutils module

`pyqha.minutils.`**`calculate_fitted_points_anis`**(*celldmsx*, *nmesh*, *fittype='quadratic'*, *ibrav=4*, *a=None*)
Calculates a denser mesh of Efitted(celldmsx) points for plotting. nmesh = (nx,ny,nz) gives the dimensions of the mesh.

`pyqha.minutils.`**`contract_vector`**(*x*, *ibrav=4*)
Utility function: contract a vector x, len(x)=6, into a x-dim vector (x<6) according to ibrav Note: not all ibrav are implemented yet

`pyqha.minutils.`**`expand_vector`**(*x*, *ibrav=4*)
Utility function: expands a vector x, len(x)<6, into a 6-dim vector according to ibrav Note: not all ibrav are implemented yet

`pyqha.minutils.`**`find_min`**(*a*, *ibrav*, *type*, *guess=None*, *method='BFGS'*, *minoptions={}*)
An auxiliary function for handling the minimum search

`pyqha.minutils.`**`find_min_quadratic`**(*a*, *ibrav*, *guess*, *method*, *minoptions*)
This is the function for finding the minimum of the quadratic polynomial

`pyqha.minutils.`**`find_min_quartic`**(*a*, *ibrav*, *guess*, *method*, *minoptions*)
This is the function for finding the minimum of the quartic polynomial

`pyqha.minutils.`**`fquadratic`**(*x*, *a*, *ibrav=4*)
Implemented polynomials for fitting and miminizing

only ibrav=4 and the most general case are implemented for now

`pyqha.minutils.`**`fquadratic_der`**(*x*, *a*, *ibrav=4*)

`pyqha.minutils.`**`fquartic`**(*x*, *a*, *ibrav=4*)

`pyqha.minutils.`**`fquartic_der`**(*x*, *a*, *ibrav=4*)

3.12 pyqha.plotutils module

pyqha.plotutils.**multiple_plot_xy**(*x*, *y*, *xlabel=''*, *ylabel=''*, *labels=''*)
This function generates a simple xy plot with matplotlib overlapping several lines as in the matrix y. y second index refers to a line in the plot, the first index is for the array to be plotted.

pyqha.plotutils.**plot_EV**(*V*, *E*, *a=None*, *labely='Etot'*)
This function plots with matplotlib E(V) data and if a is given it also plot the fitted results

pyqha.plotutils.**plot_Etot**(*celldmsx*, *Ex*, *n*, *nmesh=(50, 50, 50)*, *fittype='quadratic'*, *ibrav=4*, *a=None*)
This function makes a 3D plot with matplotlib Ex(celldmsx) data and if a is given it also plot the fitted results. The plot type depends on ibrav.

pyqha.plotutils.**plot_Etot_contour**(*celldmsx*, *nmesh=(50, 50, 50)*, *fittype='quadratic'*, *ibrav=4*, *a=None*)
This function makes a countour plot with matplotlib of Ex(celldmsx) fitted results. The plot type depends on ibrav.

pyqha.plotutils.**simple_plot_xy**(*x*, *y*, *xlabel=''*, *ylabel=''*)
This function generates a simple xy plot with matplotlib.

3.13 pyqha.properties_anis module

pyqha.properties_anis.**compute_alpha**(*minT*, *ibrav*)
This function calculate the thermal expansion alphaT at different temperatures from the input minT matrix by computing the numerical derivatives with numpy. The input matrix minT has shape nT*6, where the first index is the temperature and the second the lattice parameter. For example, minT[i,0] and minT[i,2] are the lattice parameters a and c at the temperature i.

More ibrav types must be implemented

pyqha.properties_anis.**compute_alpha_splines**(*TT*, *minT*, *ibrav*, *splinesoptions*)
This function calculates the thermal expansions alphaT at different temperatures as the previous function but using spline interpolation as implemented in scipy.interpolate.

pyqha.properties_anis.**compute_heat_capacity**(*TT*, *minT*, *alphaT*, *C*, *ibrav=4*)
This function calculate the difference between the constant stress heat capacity C_sigma and the constant strain heat capacity C_epsilon from the V, the thermal expansions and the elastic constant tensor C

pyqha.properties_anis.**compute_volume**(*celldms*, *ibrav=4*)
Compute the volume given the celldms, only for ibrav=4 for now

3.14 pyqha.read module

pyqha.read.**read_Etot**(*fname*, *ibrav=4*, *bc_as_a_ratio=True*)
Read cell parameters *(a,b,c)* and the corresponding energies from input file *fname*. Each set of cell parameters is stored in a numpy array of lenght 6 for *(a,b,c,alpha,beta,gamma)* respectively. This is done for a future possible extension but for now only the first 3 elements are used (the others are always 0). All sets are stored in *celldmsx* and *Ex*, the former is a nE*6 matrix, the latter is a nE array.

ibrav identifies the Bravais lattice as in Quantum Espresso and is needed in input (default is 4, i.e. hexagonal cell). The input file format depends on *ibrav*, for example in the hex case, the first two columns are for *a* and *c* and the third is for the energies.

If *bc_as_a_ratio=True*, the input data are assumed to be given as $(a, b/a, c/a)$ in the input file and hence converted into (a, b, c) which is how they are always stored internally in `pyqha`.

Units must be $a.u.$ and $Ryd/cell$

`pyqha.read.`**`read_EtotV`**(*fname*)
Read cell volumes and the corresponding energies from input file *fname* (1st col, volumes, 2nd col energies). Units must be $a.u.^3$ and $Ryd/cell$

`pyqha.read.`**`read_alpha`**(*fname*)

`pyqha.read.`**`read_celldmt_hex`**(*filename*)

`pyqha.read.`**`read_dos`**(*filename*)
Read the phonon density of states (y axis) and the corresponding energies (x axis) from the input file *filename* (1st col energies, 2nd col DOS) and store it in two numpy arrays which are returned.

`pyqha.read.`**`read_dos_geo`**(*fin*, *ngeo*)
Read the phonon density of states and energies as in `read_dos()` from *ngeo* input files *fin1*, *fin2*, etc. and store it in two numpy matrices which are returned.

`pyqha.read.`**`read_elastic_constants`**(*fname*)
This function reads and returns the elastic constants and compliances from the file *fname* . Elastic constants (and elastic compliances) are stored in Voigt notation They are then 6x6 matrices, stored as numpy matrices of shape [6,6] So, the elastic constant C11 is in C[0,0], C12 in C[0,1] and so on. Same for the elastic compliances.

`pyqha.read.`**`read_elastic_constants_geo`**(*fC*, *ngeo*)
Read elastic constants calculated on a multidimensional grid of lattice parameters *ngeo* defines the total number of geometries evaluated Note: the order must be the same as for the total energies!

`pyqha.read.`**`read_freq`**(*filename*)
This funcstion reads the phonon frequencies at each *q* point from a frequency file. Input file has the following format (to be done).

Returning values are a nq*3 matrix q, each q[i] being a q point (vector of 3 elements) and a nq*modes matrix freq, each element freq[i] being the phonon frequencies (vector of modes elements)

`pyqha.read.`**`read_freq_ext`**(*filename*)
Read the phonon frequencies at each q point from a frequency file. The format of this file is different from the one read by the function read_freq and contains usually more frequencies, each with a weight, but no qpoint coordinates. Input file has the following format:

First line contains n. atoms, nqx, nqy, nqz, nq total. Second line not read. Third line: weight of the first qpoint Following lines: phonon frequencies (their number is modes=3*n. atoms), one per line then again: weight of the next qpoint, phonon frequencies (3*modes), one per line, etc.

Weights are diffent because of simmetry

Returning values are a nq vector weights, each weights[i] being the weight of a q point and a nq*modes matrix freq, each element freq[i] being the phonon frequencies (vector of modes elements) at the qpoint i

`pyqha.read.`**`read_freq_ext_geo`**(*inputfilefreq*, *rangegeo*)
Read the frequencies for all geometries where the gruneisen parameters must be calculated.

Notes: nq = qgeo.shape[1] -> total number of q points read modes = freqgeo.shape[2] -> number of frequency modes

`pyqha.read.`**`read_freq_geo`**(*inputfilefreq*, *rangegeo*)
Read the frequencies for all geometries where the gruneisen parameters must be calculated. Start, stop, step must be given accordingly. It can be used to read the frequencies only at some geometries from a larger set, if necessary, providing the proper start, stop and step values.

Notes: nq = qgeo.shape[1] -> total number of q points read modes = freqgeo.shape[2] -> number of frequency modes

`pyqha.read.`**`read_thermo`**(*fname*, *ngeo=1*)

Read vibrational thermodynamic functions (Evib, Fvib, Svib, Cvib) as a function of temperature from the input file *fname*. *ngeo* is the number of input files to read, corresponding for example to different geometries in a quasi-harmonic calculation. If *ngeo>1* reads from the files *fname1*, *fname2*, etc. up to *ngeo* Input file(s) have the following format:

T	E_{vib}	F_{vib}	S_{vib}	C_{vib}
1	...	...	...	...

Lines starting with "#" are not read (comments).

Returning values are $nT * ngeo$ numpy matrices (T,Evib,Fvib,Svib,Cvib) containing the temperatures and the above mentioned thermodynamic functions as for example: Fvib[T,geo] -> Fvib at the temperature *T* for the geometry *geo*

Units must be *K* for temperature, *Ryd/cell* for energies, *Ryd/cell/K* for entropy and heat capacity.

3.15 pyqha.thermo module

`pyqha.thermo.`**`compute_thermo`**(*E*, *dos*, *TT*)

This function computes the vibrational energy, Helmholtz energy, entropy and heat capacity in the harmonic approximation from the input numpy arrays *E* and *dos* containing the phonon DOS(E). The calculation is done over a set of temperatures given in input as a numpy array *TT*. It also computes the number of phonon modes obtained from the input DOS (which must be approximately equal to $3*N$, with *N* the number of atoms per cell) and the ZPE. The input energy and dos are expected to be in 1/cm-1. It returns numpy arrays for the following quantities (in this order): temperatures, vibrational energy, Helmholtz energy, entropy, heat capacity. Plus it returns the ZPE and number of phonon modes obtained from the input DOS.

`pyqha.thermo.`**`compute_thermo_geo`**(*fin*, *fout=None*, *ngeo=1*, *TT=array([1])*)

This function reads the input dos file(s) from *fin+i*, with *i* a number from 1 to *ngeo* + 1 and computes vibrational energy, Helmholtz energy, entropy and heat capacity in the harmonic approximation. Then writes the output on file(s) if fout!=None. Output file(s) have the following format:

T	E_{vib}	F_{vib}	S_{vib}	C_{vib}
1	...	...	...	...

and are names *fout* +1, *fout* +2,... for each geometry.

Returning values are (len(TT),ngeo) numpy matrices (T,gEvib,gFvib,gSvib,gCvib,gZPE,gmodes) containing the temperatures and the above mentioned thermodynamic functions as for example: Fvib[T,geo] -> Fvib at the temperature "T" for the geometry "geo"

`pyqha.thermo.`**`dos_integral`**(*E*, *dos*, *m=0*)

A function to compute the integral of an input phonon DOS (*dos*) with the 3/8 Simpson method. *m* is the moment of the integral, if $m > 0$ different moments can be calculated. For example, with $m = 0$ (default) it returns the number of modes from the dos, with $m = 1$ it returns the ZPE. The input energy (*E*) and phonon DOS (*dos*) are expected to be in cm^{-1}.

`pyqha.thermo.`**`gen_TT`**(*Tstart=1*, *Tend=1000*, *Tstep=1*)

A simple function to generate a numpy array of temperatures, starting from *Tstart* and ending to *Tend* (or the closest *T<Tend* accorinding to the *Tstep*) with step *Tstep* .

`pyqha.thermo.`**`rearrange_thermo`**(*T*, *Evib*, *Fvib*, *Svib*, *Cvib*, *ngeo=1*)

This function just rearranges the order of the elements in the input matrices The first index of the returning matrices *X* now gives all geometries at a given *T*, i.e. *X[0]* is the vector of the property *X* a *T=T[0,0]* . *X[0,0]* for the first geometry, *X[0,1]* the second geometry and so on.

3.16 pyqha.write module

pyqha.write.**write_CT**(*Ts*, *CT*, *fCout=''*)
Write elastic constants calculated on a multidimensional grid of lattice parameters ngeo defines the total number of geometries evaluated Note: the order must be the same as for the total energies!

pyqha.write.**write_C_geo**(*celldmsx*, *C*, *ibrav=4*, *fCout=''*)
Write elastic constants calculated on a multidimensional grid of lattice parameters ngeo defines the total number of geometries evaluated Note: the order must be the same as for the total energies in the quasi-harmonic calculations!

pyqha.write.**write_Etot**(*celldmsx*, *Ex*, *fname*, *ibrav=4*)
Read cell parameters (a,b,c,alpha,beta,gamma) and energies for a grid of cell parameters values from file output_energy1. Each celldms is a vector of lenght 6 containing a,b,c,alpha,beta,gamma respectively celldmsx and Ex contains the grid of values of celldms and E so that: celldmsx[0] = celldms0 Ex[0] = E0 celldmsx[1] = celldms1 Ex[1] = E1 celldmsx[2] = celldms2 Ex[2] = E2 .... values are taken from the file "fname" ibrav is the Bravais lattice as in Quantum Espresso and is needed in input (default is cubic)

pyqha.write.**write_alphaT**(*fname*, *T*, *alphaT*, *ibrav=4*)

pyqha.write.**write_celldmsT**(*fname*, *T*, *x*, *ibrav=4*)

pyqha.write.**write_elastic_constants**(*C*, *S*, *fname*)
Elastic constants (and elastic compliances) are stored in Voigt notation They are then 6x6 matrices, stored as numpy matrices of shape [6,6] So, the elastic constant C11 is in C[0][0], C12 in C[0][1] and so on. Same for the elastic compliances

pyqha.write.**write_freq**(*qgeo*, *freq*, *filename*)
Write frequencies (or Gruneisen parameters) in a file. In this format also q points coordinates are written but not the weight of each point. It can be used to write the Gruneisen mode parameters, giving them in input as freq

pyqha.write.**write_freq_ext**(*weights*, *freq*, *filename*)
Write frequencies (or Gruneisen parameters) on an extended mesh in a file. In this format, q points coordinates are NOT written but the weight of each point yes. It can be used to write the Gruneisen mode parameters, giving them in input as freq Write the gruneisen parameters

pyqha.write.**write_thermo**(*fname*, *T*, *Evib*, *Fvib*, *Svib*, *Cvib*, *ZPE*, *modes*)

pyqha.write.**write_xy**(*fname*, *x*, *y*, *labelx*, *labely*)
This function writes a quantity y versus quantity x into the file fname. y and x are arrays and should have the same lenght. labelx and labely are the axis labels (possibly with units), written in the header of the file (first line).

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

p

C

D

E

F

G

H

M

P

R

S

W