DebyeTools

Javier Jofre

May 14, 2024

CONTENT:

`debyetools` is a set of tools written in Python for the calculation of thermodynamic and thermophysical properties. It's a library in The Python Package Index (PyPI). The software presented here is based in the Debye approximation of the quasiharmonic approximation (QHA) using the crystal internal energetics parametrized at ground-state, (go to *input file formats* to see how DFT calculations results can be used as inputs) to project the *thermodynamics properties* at high temperatures. We present here how each contribution to the free energy are considered and a description of the architecture of the calculation engine and of the *GUI*.

The code is freely available under the GNU Affero General Public License.

CHAPTER

ONE

HOW TO CITE:

If you use `debyetools` in a publication, please refer to the source code. If you use the implemented method for the calculation of the thermodynamic properties, please cite the following publication:

Jofre, J., Gheribi, A. E., & Harvey, J.-P. Development of a flexible quasi-harmonic-based approach for fast generation of self-consistent thermodynamic properties used in computational thermochemistry. Calphad 83 (2023) 102624. doi: 10.1016/j.calphad.2023.102624.

```
@article{,
   author = {Javier Jofré and Aïmen E. Gheribi and Jean-Philippe Harvey},
   doi = {10.1016/j.calphad.2023.102624},
   issn = {03645916},
   journal = {Calphad},
   month = {12},
   pages = {102624},
   title = {Development of a flexible quasi-harmonic-based approach for fast generation␣
↪of self-consistent thermodynamic properties used in computational thermochemistry},
   volume = {83},
   year = {2023},
}
```

1.1 Calculate quality thermodynamic properties in a flexible and fast manner:

It's possible to couple the Debye model to other algorithms, to *fit experimental data* and in this way use data available to calculate other properties like thermal expansion, free energy, bulk modulus among many others.

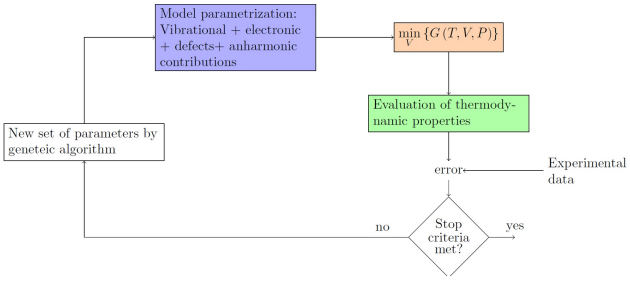


Fig. 1: `debyetools` coupled to a Genetic Algorithm.

The prediciton of *thermodynamic phase equilibria at high pressure* can be performed by simultaneous parameter adjusting to experimental heat capacity and thermal expansion at P = 0.

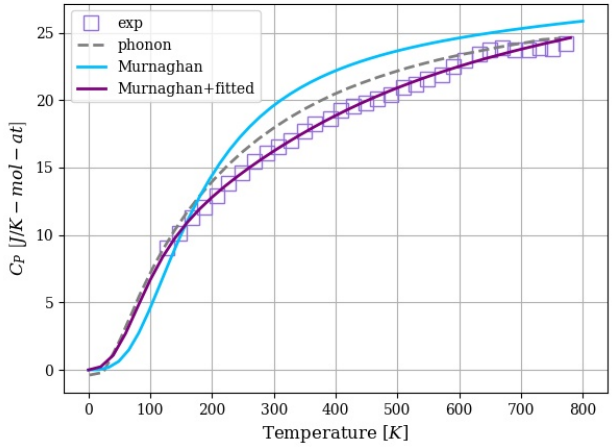


Fig. 2: Heat capacity of LiFePO4 calculated with `debyetools` and compared to other methods.

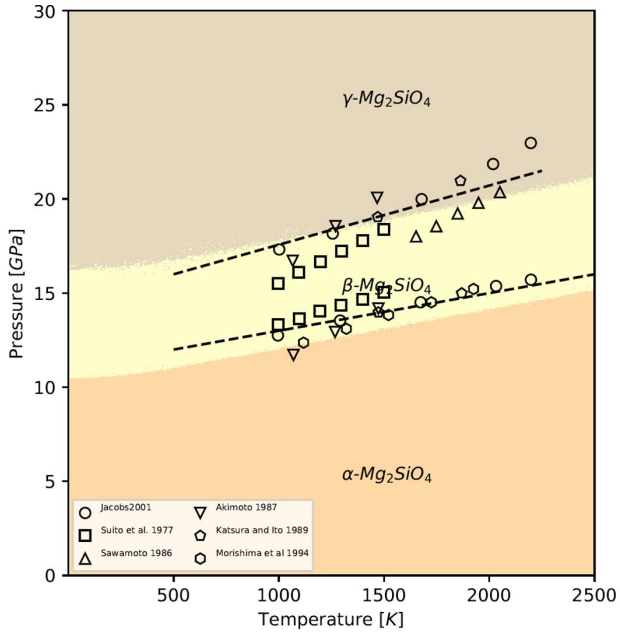


Fig. 3: Phase diagram P versus T for the , and forms of Mg2SiO4. Symbols are literature data for the phase stability regions boundaries.

1.2 Using debyetools through the GUI:

debyetools is a Python library that also comes with a graphical user interface to help perform quick calculations without the need to code scripts.

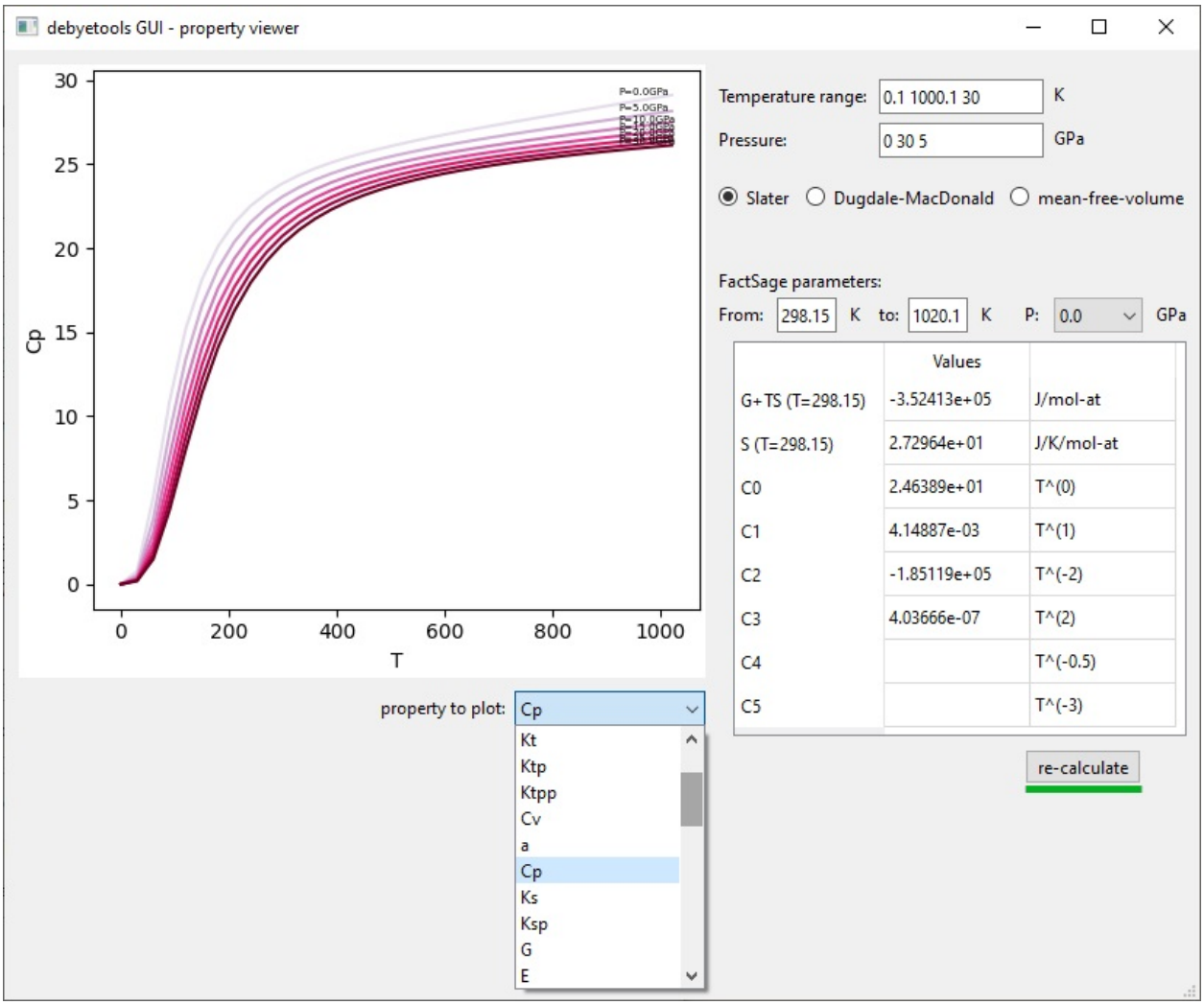


Fig. 4: debyetools property viewer.

1.3 Using debyetools as a Python library. Example: Al fcc using Morse Potential:

Using debyetools as a Python library adds versatility and expands its usability.

EOS parametrization:

```
>>> import debyetools.potentials as potentials
>>> from debyetools.aux_functions import load_V_E
>>> V_data, E_data = load_V_E('/path/to/SUMMARY', '/path/to/CONTCAR')
```

(continues on next page)

(continued from previous page)

```
>>> params_initial_guess = [-3e5, 1e-5, 7e10, 4]
>>> formula = 'AlAlAlLi'
>>> cell = np.array([[4.025,0,0],[0,4.025488,0],[0,0,4.025488]])
>>> basis = np.array([[0,0,0],[.5,.5,0],[.5,0,.5],[0,.5,.5]])
>>> cutoff, number_of_neighbor_levels = 5, 3
>>> Morse = potentials.MP(formula, cell, basis, cutoff,
...                       number_of_neighbor_levels)
>>> Morse.fitEOS(V_data, E_data, params_initial_guess)
array([-3.26551e+05,9.82096e-06,6.31727e+10,4.31057e+00])
```

Calculation of the electronic contribution:

```
>>> from debyetools.aux_functions import load_doscar
>>> from debyetools.electronic import fit_electronic
>>> p_el_inittial = [3.8e-01,-1.9e-02,5.3e-04,-7.0e-06]
>>> E, N, Ef = load_doscar('/path/to/DOSCAR.EvV.',list_filetags=range(21))
>>> fit_electronic(V_data, p_el_inittial, E, N, Ef)
array([1.73273079e-01,-6.87351153e+03,5.3e-04,-7.0e-06])
```

Poisson's ratio:

```
>>> import numpy as np
>>> from debyetools . poisson import poisson_ratio
>>> from debyetools.aux_functions import load_EM
>>> EM = load_EM( 'path/to/OUTCAR')
>>> poisson_ratio ( EM )
0.2 2 9 41 5 49 8 67148558
```

Free energy minimization:

```
>>> from debyetools.ndeb import nDeb
>>> from debyetools import potentials
>>> from debyetools.aux_functions import gen_Ts,load_V_E
>>> m = 0.021971375
>>> nu = poisson_ratio (EM)
>>> p_electronic = fit_electronic(V_data, p_el_inittial, E, N, Ef)
>>> p_defects = [8.46, 1.69, 933, 0.1]
>>> p_anh, p_intanh = [0,0,0], [0, 1]
>>> V_data, E_data = load_V_E('/path/to/SUMMARY', '/path/to/CONTCAR')
>>> eos = potentials.BM()
>>> peos = eos.fitEOS(V_data, E_data, params_initial_guess)
>>> ndeb = nDeb (nu , m, p_intanh , eos , p_electronic , p_defects , p_anh )
>>> T = gen_Ts ( T_initial , T_final , 10 )
>>> T, V = ndeb.min_G (T,  1e-5, P=0)
>>> V
array([9.98852539e-06, 9.99974297e-06, 1.00578469e-05, 1.01135875e-05,
       1.01419825e-05, 1.02392921e-05, 1.03467847e-05, 1.04650048e-05,
       1.05953063e-05, 1.07396467e-05, 1.09045695e-05, 1.10973163e-05])
```

Evaluation of the thermodynamic properties:

```
>>> trprops_dict=ndeb.eval_props(T,V)
>>> tprops_dict['Cp']
```

(continues on next page)

(continued from previous page)

```
array([4.02097531e-05, 9.68739597e+00, 1.96115210e+01, 2.25070513e+01,
       2.34086394e+01, 2.54037595e+01, 2.68478029e+01, 2.82106379e+01,
       2.98214145e+01, 3.20143195e+01, 3.51848547e+01, 3.98791392e+01])
```

FS compound database parameters:

```
>>> from debyetools.fs_compound_db import fit_FS
>>> T_from = 298.15
>>> T_to = 1000.1
>>> FS_db_params = fit_FS(tprops_dict, T_from,T_to)
>>> Fs_db_params['Cp']
array([ 3.48569519e+01, -2.56558596e-02, -6.35562885e+05,  2.65035585e-05])
```

1.3.1 Installation

Source Code

All the code is in `debyetools` repository on GitHub.

Requirements

- Python 3.6 or newer
- NumPy (base N-dimensional array package)
- SciPy (fundamental algorithms for scientific computing in Python)
- mpmath (real and complex floating-point arithmetic with arbitrary precision)

Other recommended packages:

- Matplotlib (for plotting, a comprehensive library for visualizations in Python)
- PySide6 (for the GUI, PySide6 is the official Python module from the Qt for Python project)

Installation using pip

The simplest way to install `debyetools` is to use pip which will automatically get the source code from PyPI:

```
$ pip install --upgrade debyetools
```

1.3.2 GUI

Table of contents

General Overview

The interface is a GUI that allows to easily parametrize the Free energy and calculate the thermodynamic properties. It is organized in two main level, (1) GUI, and (2) calculation engine. The GUI will receive the user defined options and/or parameter values and will launch the calculations and display the results.The user level part of the software is divided in four modules: (1) parametrization, (2) free energy minimization,(3) evaluation of thermodynamic properties, and (4) calculation of the database parameters.

Note that in this version, most *input file* must be in VASP format, i.e., CONTCAR for the crystal structure, and DOSCAR for the calculation of the electronic contribution. Also, the elastic moduli matrix is read from an OUTCAR file.

How to launch it:

To start getting familiar with the interface you can download examples input files. The GUI can be launched by executing the interface script from the debyetools repository main folder:

```
$ python interface.py
```

Or you can launch inside python:

```
>>> from debyetools.tpropsgui.gui import interface
>>> interface()
```

The interface main window:

The model parametrization is done in the main window.

The properties viewer

The calculation results are shown in this window.

Parametrization

The parameters of all contributions are entered and/or calculated in this module. The mass is entered as $kg/mol - at$. EOS parameters can be fitted for the internal energy if the energy curve and initial guess for the parameters are given. The EOS to be used can be selected from a drop-down list. For the selected EOS/potential, the fitting of the parameters is carried out if the 'fit' option is selected, otherwise, the parameters entered manually are used.

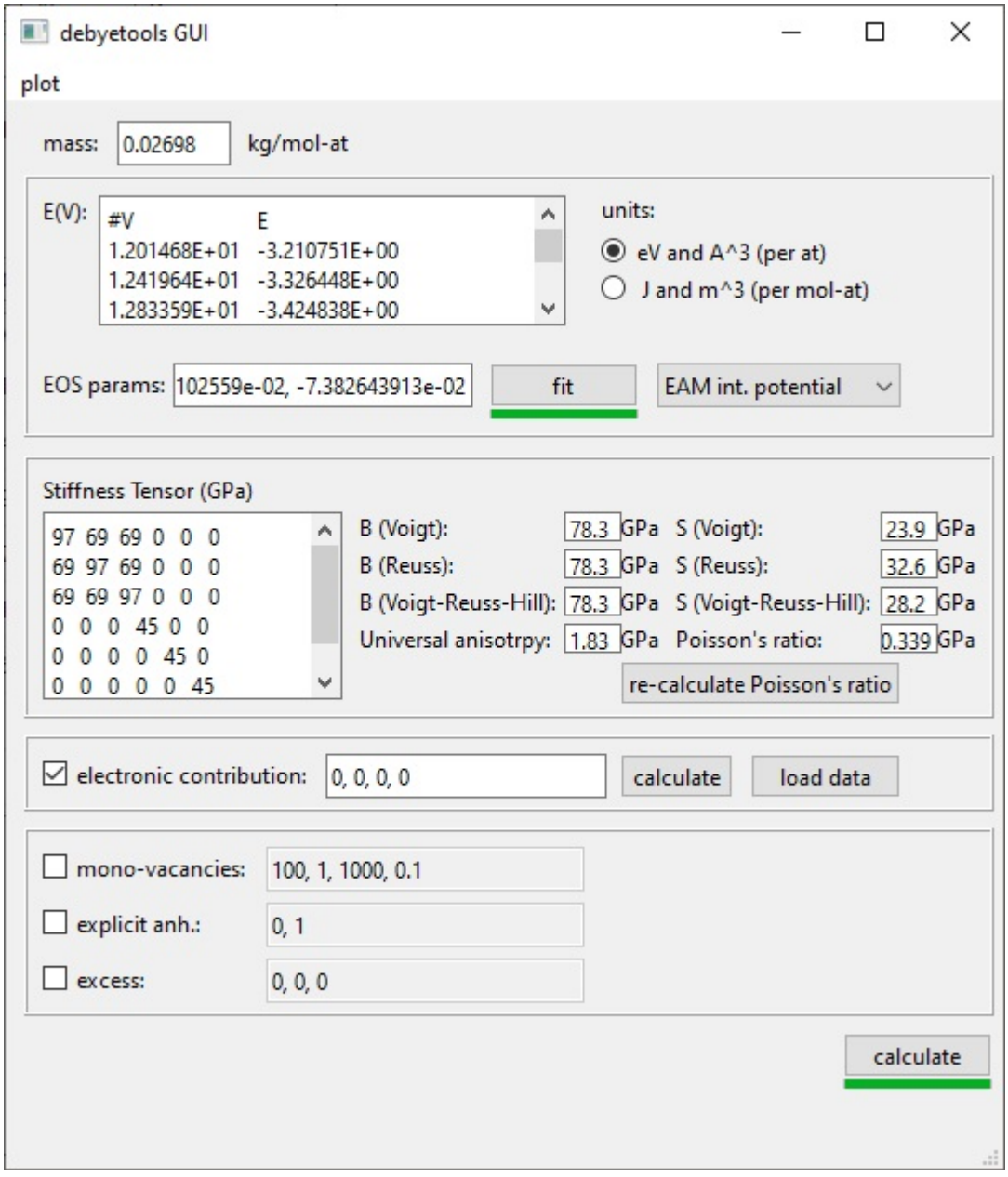


Fig. 5: `debyetools` interface main window.

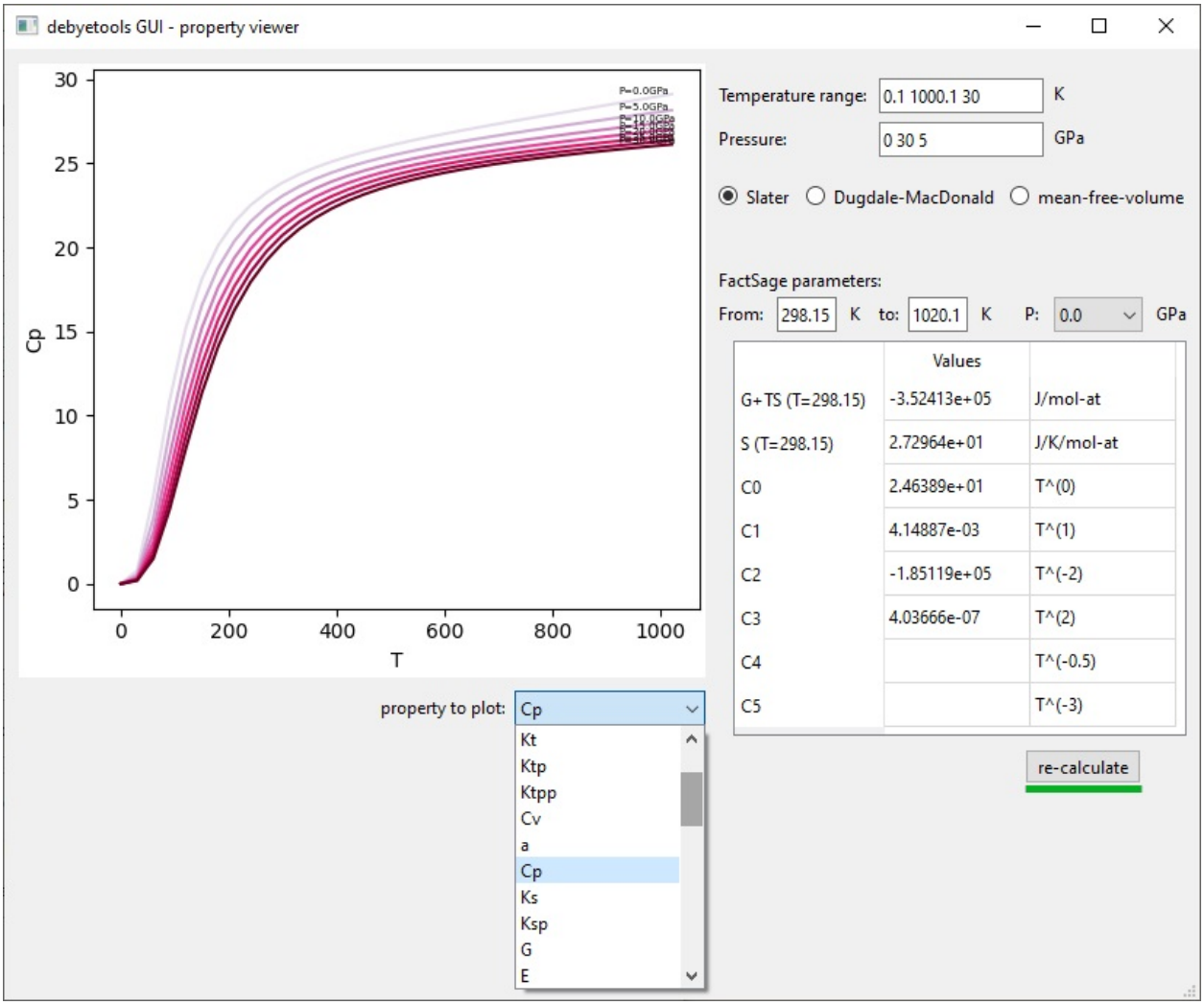


Fig. 6: `debyetools` properties viewer.

V(T)

After the parametrization is complete, the temperature dependence of the equilibrium volume is carried out when clicking the button 'calculate'. Once the input parameters have been entered and/or calculated, it is possible to proceed to the calculation of the volume as a function of temperature through free energy minimization. Default values for pressure and temperature are used the first time the calculation is carried out, with pressures from 0 to $30GPa$ and temperatures from 0.1 to $1000.1K$. An energy minimization will be performed at each temperature and each pressure. The calculated volume as a function of temperature and pressure is stored for the evaluation of the thermodynamic properties.

Thermodynamic Properties

The evaluation of the thermodynamic properties can be performed in this module for the list of triplets (T, V, P) from the previous part. Each individual property can be selected from a drop-down menu to be plotted.

In the results visualization window properties are evaluated and shown as a function of temperature and pressure. By default is the heat capacity that is presented but the available properties can be selected from a drop-down menu. Note that right-clocking in the figure allows to copy the data to the clipboard. The defined pressure and temperature range can be modified from default values. The default setting performs calculations using the Slater approximation for the Debye temperature but the calculations can be performed using also the Dugdale-Macdonald or mean-free volume theory approximations. The FactSace compound database parameters are also presented in this view for each pressure, for the chosen range of temperature. If the setting are changed, the calculation can be re-run clicking 're-calculate'.

FS compound database parameters.

The calculated thermodynamic properties for each EOS selected are used to fit the models for heat capacity, thermal expansion, bulk modulus and pressure derivative of the bulk modulus. The resulting parameters are printed in the GUI to be used in FactSage as a compound database.

1.3.3 Examples

Table of contents

Al_3Li $L1_2$ thermodynamic properties

In order to calculate the thermodynamic properties of an element or compound, we need first to parametrize the function that will describe the internal energy of the system. In this case we have chosen the Birch-Murnaghan equation of state and fitted it against DFT data loaded using the `load_V_E` function. The `BM` object instantiates the representation og the EOS and its derivatives. In the module `potentials` there are all the implemented EOS. The method `fitEOS` with the option `fit=True` will fit the EOS parameters to (Volume, Energy) data using `initial_parameters` as initial guess. The following is an example for Al_3Li $L1_2$.

The following code allows to access the example files:

```
>>> import os
>>> import debyetools
>>> file_path = debyetools.__file__
>>> dir_path = os.path.dirname(file_path)
```

Loading the energy curve and fitting the Birch-Murnaghan EOS:

```
>>> from debyetools.aux_functions import load_V_E
>>> import debyetools.potentials as potentials
>>> import numpy as np
>>> V_DFT, E_DFT = load_V_E(dir_path+'/examples/Al3Li_L12/SUMMARY.fcc', dir_path+'/
↪examples/Al3Li_L12/CONTCAR.5', units='J/mol')
>>> initial_parameters =  np.array([-4e+05, 1e-05, 7e+10, 4])
>>> eos_BM = potentials.BM()
>>> eos_BM.fitEOS(V_DFT, E_DFT, initial_parameters=initial_parameters, fit=True)
>>> p_EOS = eos_BM.pEOS
>>> p_EOS
array([-3.26544606e+05,  9.82088168e-06,  6.31181335e+10,  4.32032416e+00])
```

To fit the electronic contribution to eDOS data we can load them as *VASP* format *DOSCAR* files using the function `load_doscar`. Then, at each `V_DFT` volume, the parameters of the electronic contribution will be fitted with the `fit_electronic` function from the `electronic` module, using `p_el_initial` as initial parameters.

```
>>> from debyetools.aux_functions import load_doscar
>>> from debyetools.electronic import fit_electronic
>>> p_el_inittial = [3.8027342892e-01, -1.8875015171e-02, 5.3071034596e-04, -7.
↪0100707467e-06]
>>> E, N, Ef = load_doscar(dir_path+'/examples/Al3Li_L12/DOSCAR.EvV.')
>>> p_electronic = fit_electronic(V_DFT, p_el_inittial,E,N,Ef)
>>> p_electronic
array([ 1.73372534e-01, -6.87754210e+03,  5.30710346e-04, -7.01007075e-06])
```

The Poisson's ratio and elastic constants can be calculated using the `poisson_ratio` method and the *elastic moduli matrix* in the *VASP* format *OUTCAR* obtained when using `IBRION = 6` in the *INCAR* file, loaded using `load_EM`.

```
>>> from debyetools.aux_functions import load_EM
>>> from debyetools.poisson import poisson_ratio
>>> EM = load_EM(dir_path+'/examples/Al_fcc/OUTCAR.eps')
>>> nu = poisson_ratio(EM)
>>> nu
0.33702122500881493
```

For this example, all other contributions are set to zero.

```
>>> Tmelting = 933
>>> p_defects = 1e10, 0, Tmelting, 0.1
>>> p_intanh = 0, 1
>>> p_anh = 0, 0, 0
```

The temperature dependence of the equilibrium volume is calculated by minimizing *G*. In this example is done at *P=0*. We need to instantiate first a `nDeb` object and define the arbitrary temperatures (this can be done using `gen_Ts`, for example). The minimization og the Gibbs free energy is done by calling the method `nDeb.minG`.

```
>>> from debyetools.ndeb import nDeb
>>> from debyetools.aux_functions import gen_Ts
>>> m = 0.026981500000000002
>>> ndeb_BM = nDeb(nu, m, p_intanh, eos_BM, p_electronic, p_defects, p_anh)
>>> T_initial, T_final, number_Temps = 0.1, 1000, 10
>>> T = gen_Ts(T_initial, T_final, number_Temps)
>>> T, V = ndeb_BM.min_G(T,p_EOS[1],P=0)
>>> T, V
(array([1.0000e-01, 1.1120e+02, 2.2230e+02, 2.9815e+02, 3.3340e+02,
        4.4450e+02, 5.5560e+02, 6.6670e+02, 7.7780e+02, 8.8890e+02,
        1.0000e+03]),
 array([9.93477130e-06, 9.95708573e-06, 1.00309860e-05, 1.00924551e-05,
        1.01230085e-05, 1.02253260e-05, 1.03361669e-05, 1.04567892e-05,
        1.05882649e-05, 1.07335434e-05, 1.08954899e-05]))
```

To plot the volume as function of temperature:

```
>>> from matplotlib import pyplot as plt
>>> plt.figure()
>>> plt.plot(T,V, label='Volume')
>>> plt.legend()
>>> plt.show()
```

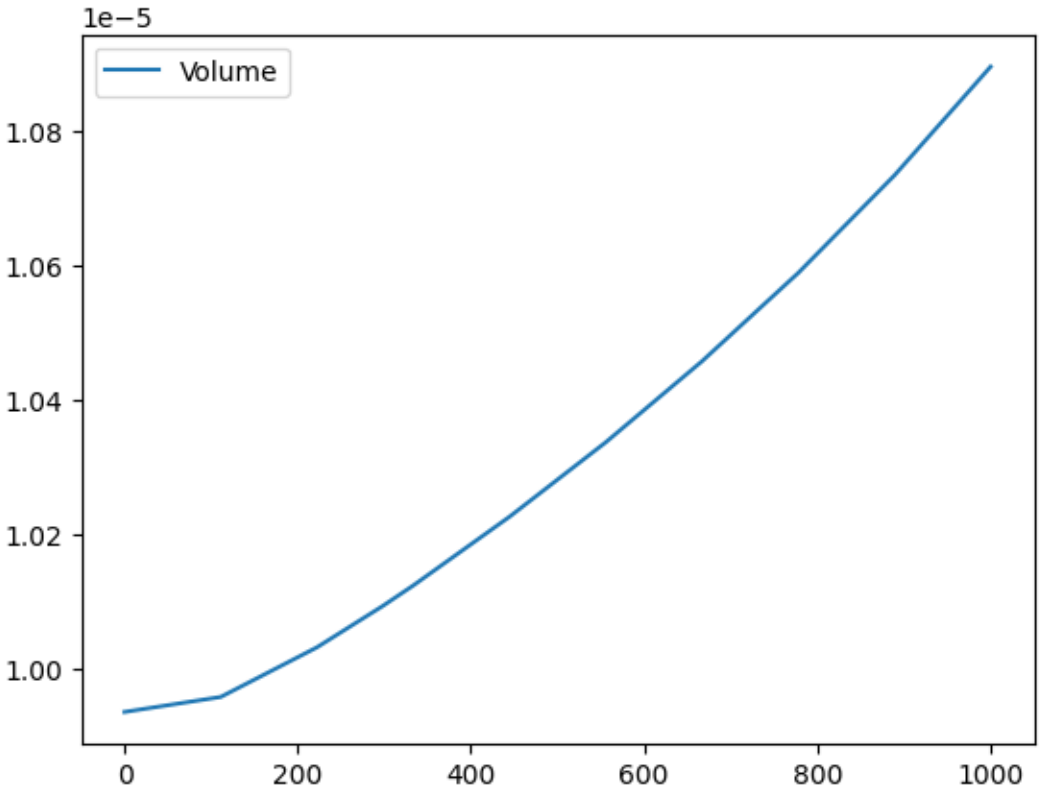
The thermodynamic properties are calculated by just evaluating the thermodynamic functions with `nDeb.eval_props`. This will return a dictionary with the values of the different thermodynamic properties.

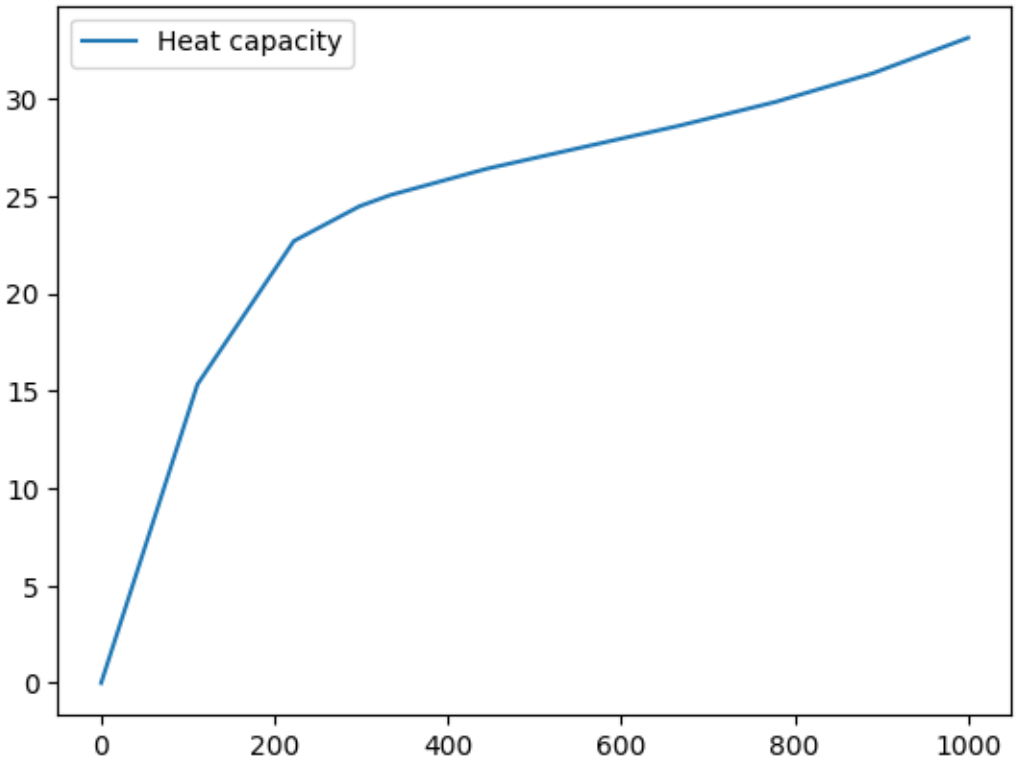
```
>>> tprops_dict = ndeb_BM.eval_props(T,V,P=0)
>>> Cp = tprops_dict['Cp']
>>> Cp
array([4.03108486e-05, 1.53280407e+01, 2.26806532e+01, 2.44706878e+01,
       2.50389680e+01, 2.63913291e+01, 2.75000371e+01, 2.86033148e+01,
       2.98237204e+01, 3.12758030e+01, 3.31133279e+01])
>>> plt.figure()
>>> plt.plot(T,Cp, label='Heat capacity')
>>> plt.legend()
>>> plt.show()
```

The FactSage Cp polynomial is fitted to the previous calculation:

```
>>> from debyetools.fs_compound_db import fit_FS
>>> T_from = 298.15
>>> T_to = 1000
>>> FS_db_params = fit_FS(tprops_dict, T_from, T_to)
>>> FS_db_params
```

(continues on next page)

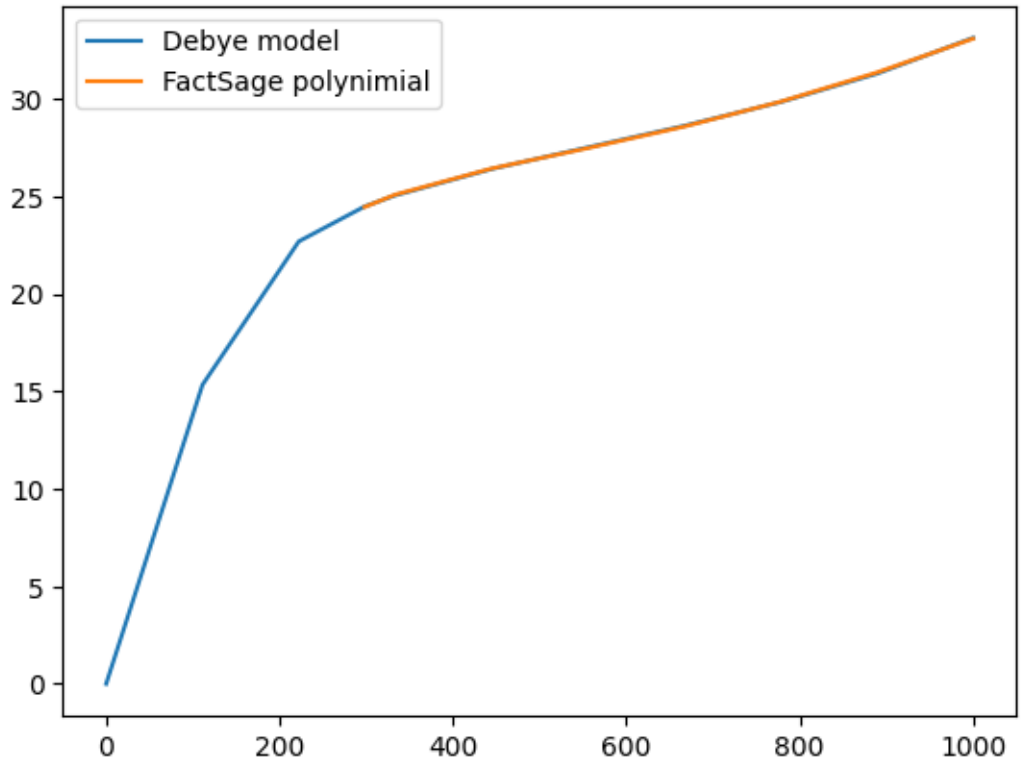

1e−5
Volume
1.08
1.06
1.04
1.02
1.00
0
200
400
600
800
1000


Heat capacity
30
25
20
15
10
5
0
0
200
400
600
800
1000

(continued from previous page)

```
{'Cp': array([ 2.82760954e+01, -6.12271903e-03, -2.66975291e+05,  1.11891931e-05]),
 'a': array([-8.00942545e-05,  1.65169216e-07,  6.62935957e-02, -9.59227812e+00]),
 '1/Ks': array([ 1.58260299e-11,  3.89418226e-15, -1.26886122e-18,  2.36654487e-21]),
 'Ksp': array([4.50472269e+00, 1.16376200e-03])}
```

Plot the parameterized heat capacity:



Thermodynamic properties with the `debyetools` interface

The same calculations as the previous example were carried out using *debyetools* GUI.

The calculated results can be plotted in the viewer window that will pop-up after clicking the button 'calculate'. Note that the number of calculations where modified from default settings to show smoother curves.

Genetic algorithm to fit Cp to experimental data.

To show how flexible `debyetools` is we shoe next a way to fit a thermodynamic property like the heat capacity to experimental data using a genetic algorithm.

First we set the initial input values and experimental values:

```
>>> import numpy as np
>>> import debyetools.potentials as potentials
>>> eos_MU = potentials.MU()
>>> V0, K0, K0p = 6.405559904e-06, 1.555283892e+11, 4.095209375e+00
```

(continues on next page)

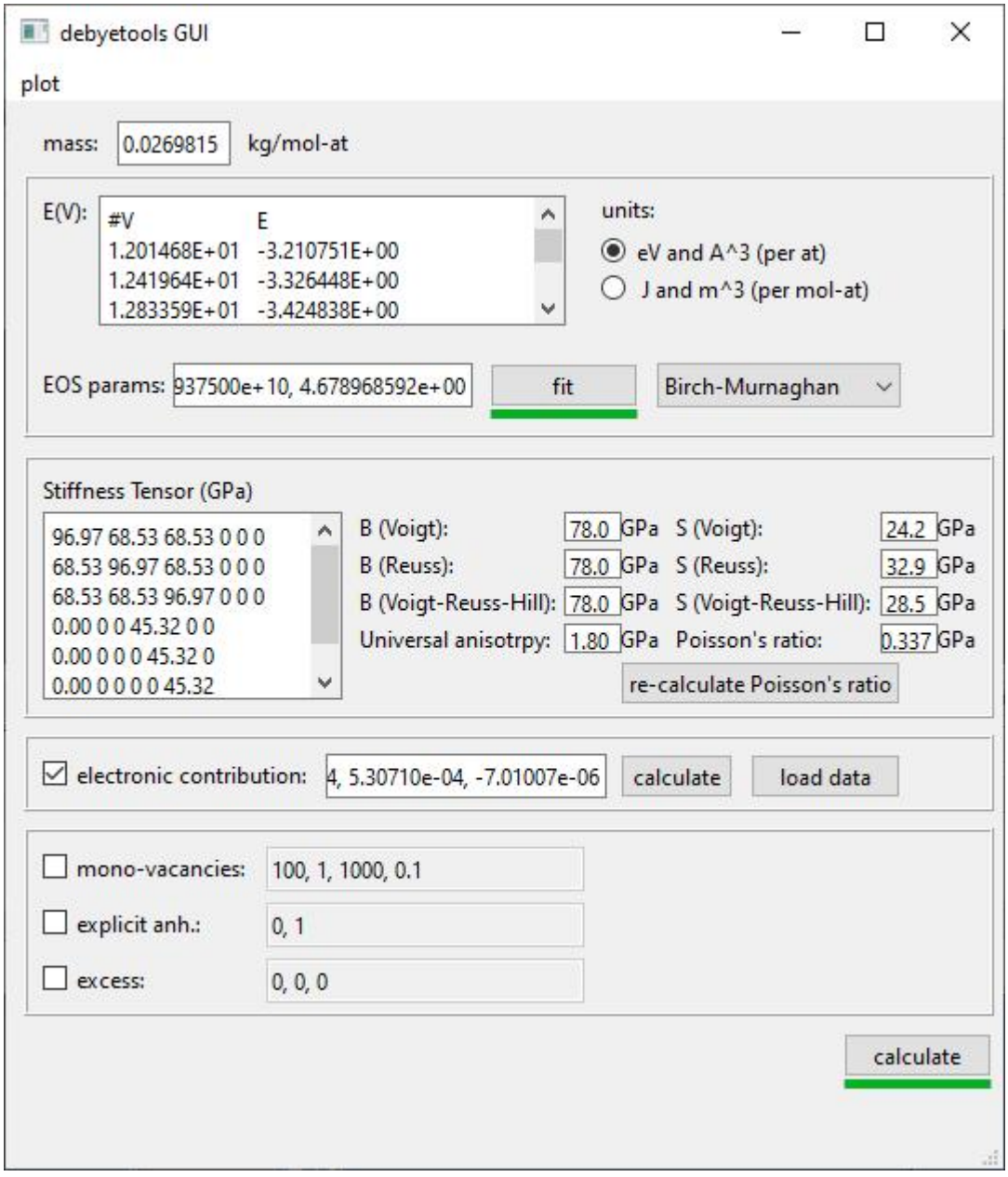


Fig. 7: *debyetools main interface*

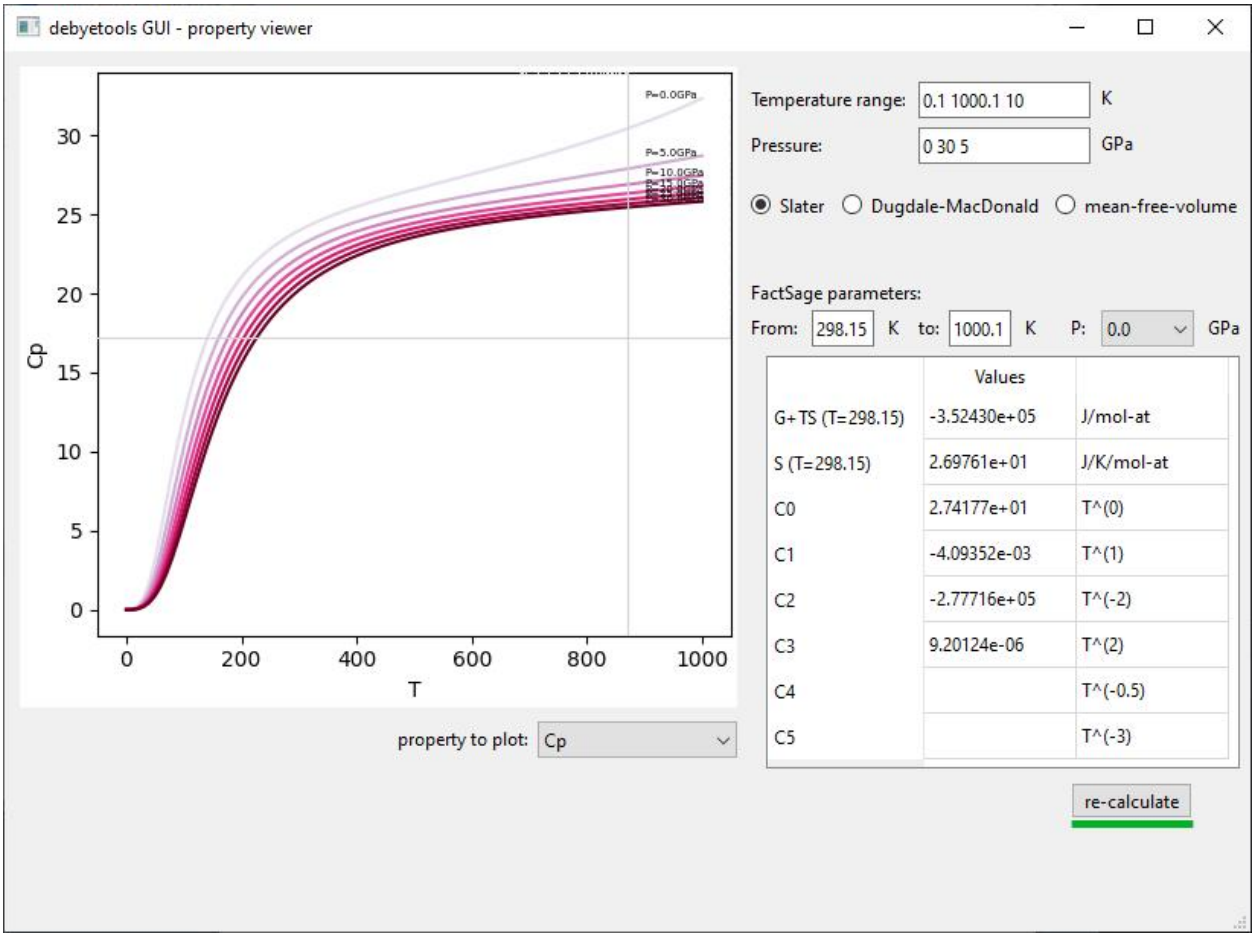


Fig. 8: *debyetools viewer window*

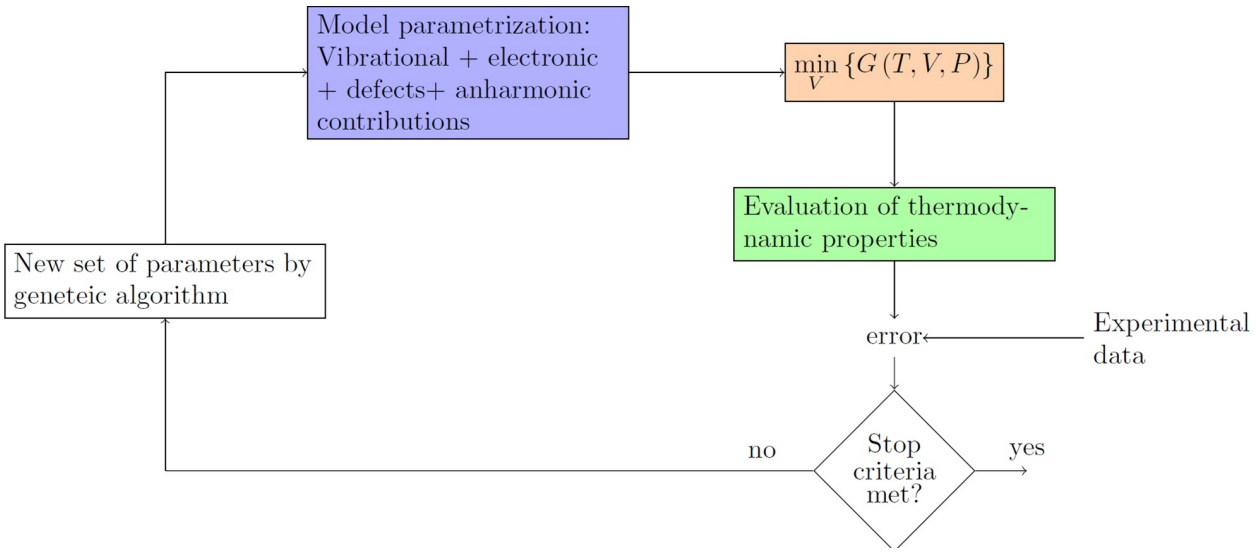


Fig. 9: Schematics for data fitting o the heat capacity to experimental data.

(continued from previous page)

```
>>> nu = 0.2747222272342077
>>> a0, m0 = 0, 1
>>> s0, s1, s2 = 0, 0, 0
>>> edef, sdef = 20,0
>>> T = np.array([126.9565217,147.826087,167.826087,186.9565217,207.826087,226.9565217,
↪248.6956522,267.826087,288.6956522,306.9565217,326.9565217,349.5652174,366.9565217,391.
↪3043478,408.6956522,428.6956522,449.5652174,467.826087,488.6956522,510.4347826,530.
↪4347826,548.6956522,571.3043478,590.4347826,608.6956522,633.0434783,649.5652174,670.
↪4347826,689.5652174,711.3043478,730.4347826,750.4347826,772.173913])
>>> C_exp = np.array([9.049180328,10.14519906,11.29742389,12.05620609,12.92740047,13.
↪82669789,14.61358314,15.45667447,16.07494145,16.55269321,17.00234192,17.73302108,18.
↪21077283,18.60421546,19.25058548,19.53161593,19.78454333,20.12177986,20.4028103,20.
↪90866511,21.18969555,21.52693208,21.89227166,22.4824356,22.96018735,23.40983607,23.
↪69086651,23.88758782,23.71896956,23.7470726,23.85948478,23.83138173,24.19672131])
```

Then we run a genetic algorithm to fit the heat capacity to the experimental data.

```
>>> import numpy.random as rnd
>>> from debyetools.ndeb import nDeb
>>> ix = 0
>>> max_iter = 500
>>> mvar=[(V0,V0*0.01), (K0,K0*0.05), (K0p,K0p*0.01), (nu,nu*0.01), (a0,5e-6), (m0,5e-3),
↪ (s0,5e-5), (s1,5e-5), (s2,5e-5), (edef,0.5), (sdef, 0.1)]
>>> parents_params = mutate(params = [V0, K0, K0p, nu, a0, m0, s0, s1, s2, edef, sdef],␣
↪n_chidren = 2, mrate=0.7, mvar=mvar)
>>> counter_change = 0
>>> errs_old = 1
>>> while ix <= max_iter:
...     children_params = mate(parents_params, 10, mvar)
...     parents_params, errs_new = select_bests(Cp_LiFePO4, T, children_params,2, C_exp)
...     V0, K0, K0p, nu, a0, m0, s0, s1, s2, edef, sdef = parents_params[0]
...     mvar=[(V0,V0*0.05), (K0,K0*0.05), (K0p,K0p*0.05), (nu,nu*0.05), (a0,5e-6), (m0,5e-
↪3), (s0,5e-5), (s1,5e-5), (s2,5e-5), (edef,0.5), (sdef, 0.1)]
...     if errs_old == errs_new[0]:
...         counter_change+=1
...     else:
...         counter_change=0
...     ix+=1
...     errs_old = errs_new[0]
...     if counter_change>=20: break
>>> T = np.arange(0.1,800.1,20)
>>> Cp1 = Cp_LiFePO4(T, parents_params[0])
>>> best_params = parents_params[0]
```

The algorithm consists in first generating the *parent* set of parameters by running `mutate` function with the option `n_children = 2` to generate two variation of the initial set. Then the iterations goes by (1) *mating* the parents using the function `mate`, (2) evaluating and (3) selecting the best 2 sets that will be the new *parents*. This will go until stop conditions are met. The `mate`, `mutate`, `select_bests` and `evaluate` are as follows:

```
def mutate(params, n_chidren, mrate, mvar):
    res = []
    for i in range(n_chidren):
```

(continues on next page)

(continued from previous page)

```
        new_params = []
        for pi, mvars in zip(params, mvar):
            if rnd.randint(0,100)/100.<=mrate:
                step = mvars[1]/10
                lst1 = np.arange(mvars[0]-mvars[1], mvars[0]+mvars[1]+step, step )
                var = lst1[rnd.randint(0,len(lst1))]
                new_params.append(var)
            else:
                new_params.append(pi)

        res.append(new_params)
    return res

def evaluate(fc, T, pi, yexp):
    return np.sqrt(np.sum((fc(T, pi)/T - yexp/T)**2))
    try:
        return np.sqrt(np.sum((fc(T, pi)/T - yexp/T)**2))
    except:
        print('these parameters are not working:',pi)
        return 1

def select_bests(fn, T, params, ngen, yexp):
    arr = []
    for ix, pi in enumerate(params):
        arr.append([ix, evaluate(fn, T, pi, yexp)])

    arr = np.array(arr)
    sorted_arr = arr[np.argsort(arr[:, 1])]
    tops_ix = sorted_arr[:ngen,0]

    return [params[int(j)] for j in tops_ix], [arr[int(j),1] for j in tops_ix]

def mate(params, ngen,mvar):
    res = [params[0],params[1]]
    ns = int(max(2,ngen-2)/2)

    for i in range(ns):
        cutsite = rnd.randint(0,len(params[0]))
        param1 = mutate(params[0][:cutsite]+params[1][cutsite:], 1, 0.5, mvar)[0]
        param2 = mutate(params[1][:cutsite]+params[0][cutsite:], 1, 0.5, mvar)[0]

        res.append(param1)
        res.append(param2)

    return res
```

The function to evaluate, the heat capacity, is as follows:

```
def Cp_LiFePO4(T, params):
    V0, K0, K0p, nu, a0, m0, s0, s1, s2, edef, sdef = params
    p_intanh = a0, m0
    p_anh = s0, s1, s2
```

(continues on next page)

(continued from previous page)

```
    # EOS parametrization
    #=========================
    initial_parameters =  [-6.745375544e+05, V0, K0, K0p]
    eos_MU.fitEOS([V0], 0, initial_parameters=initial_parameters, fit=False)
    p_EOS = eos_MU.pEOS
    #=========================

    # Electronic Contributions
    #=========================
    p_electronic = [0,0,0,0]
    #=========================

    # Other Contributions parametrization
    #=========================
    Tmelting = 800
    p_defects = edef, sdef, Tmelting, 0.1
    #=========================

    # F minimization
    #=========================
    m = 0.02253677142857143
    ndeb_MU = nDeb(nu, m, p_intanh, eos_MU, p_electronic,
                   p_defects, p_anh, mode='jj)
    T, V = ndeb_MU.min_G(T, p_EOS[1], P=0)
    #=========================

    # Evaluations
    #=========================
    tprops_dict = ndeb_MU.eval_props(T, V, P=0)
    #=========================

    return tprops_dict['Cp']
```

The result of this fitting can be plotted using the `plotter` module:

```
import debyetools.tpropsgui.plotter as plot

T_exp = np.array([126.9565217,147.826087,167.826087,186.9565217,207.826087,226.9565217,
↪248.6956522,267.826087,288.6956522,306.9565217,326.9565217,349.5652174,366.9565217,391.
↪3043478,408.6956522,428.6956522,449.5652174,467.826087,488.6956522,510.4347826,530.
↪4347826,548.6956522,571.3043478,590.4347826,608.6956522,633.0434783,649.5652174,670.
↪4347826,689.5652174,711.3043478,730.4347826,750.4347826,772.173913])
Cp_exp = np.array([9.049180328,10.14519906,11.29742389,12.05620609,12.92740047,13.
↪82669789,14.61358314,15.45667447,16.07494145,16.55269321,17.00234192,17.73302108,18.
↪21077283,18.60421546,19.25058548,19.53161593,19.78454333,20.12177986,20.4028103,20.
↪90866511,21.18969555,21.52693208,21.89227166,22.4824356,22.96018735,23.40983607,23.
↪69086651,23.88758782,23.71896956,23.7470726,23.85948478,23.83138173,24.19672131])
T_ph = [1.967263911, 24.08773869, 40.16838464, 51.99817063, 62.61346532, 71.62728127, 82.
↪14182721, 95.16347545, 108.6874128, 123.7174904, 140.2528445, 158.7958422, 179.3467704,
↪ 202.4077519, 226.4743683, 250.5441451, 274.6162229, 299.1922033, 323.2681948, 347.
↪8476048, 371.9269543, 396.0073777, 420.0891204, 444.171937, 468.7572464, 492.8416261,␣
```

(continues on next page)

(continued from previous page)

```
↪516.9264916, 541.5140562, 565.6001558, 589.6869304, 613.7740731, 638.3634207, 662.
↪4510066, 686.0373117, 711.1294163, 734.2134743, 764.3270346]
Cp_ph =[-0.375850956, -0.178378686, 1.227397939, 2.313383473, 3.431619848, 4.344789455,␣
↪5.478898585, 6.723965937, 7.953256737, 9.166990283, 10.40292814, 11.64187702, 12.
↪87129914, 14.08268875, 15.21632722, 16.2118242, 17.10673273, 17.9153379, 18.63917154,␣
↪19.29786266, 19.87491167, 20.4050194, 20.87745642, 21.30295216, 21.70376428, 22.
↪06093438, 22.39686914, 22.69910457, 22.98109412, 23.23357771, 23.46996716, 23.69426517,
↪ 23.91128202, 24.1000059, 24.28807125, 24.49073617, 24.58375529]

T_JJ = [1.00000E-01,1.64245E+01,3.27490E+01,4.90735E+01,6.53980E+01,8.17224E+01,9.
↪80469E+01,1.14371E+02,1.30696E+02,1.47020E+02,1.63345E+02,1.79669E+02,1.95994E+02,2.
↪12318E+02,2.28643E+02,2.44967E+02,2.61292E+02,2.77616E+02,2.93941E+02,2.98150E+02,3.
↪10265E+02,3.26590E+02,3.42914E+02,3.59239E+02,3.75563E+02,3.91888E+02,4.08212E+02,4.
↪24537E+02,4.40861E+02,4.57186E+02,4.73510E+02,4.89835E+02,5.06159E+02,5.22484E+02,5.
↪38808E+02,5.55133E+02,5.71457E+02,5.87782E+02,6.04106E+02,6.20431E+02,6.36755E+02,6.
↪53080E+02,6.69404E+02,6.85729E+02,7.02053E+02,7.18378E+02,7.34702E+02,7.51027E+02,7.
↪67351E+02,7.83676E+02,8.00000E+02]
Cp_JJ = [Cp_LiFePO4(T, params_Murnaghan) fir T in T_JJ]
Cp_JJ_fitted = [Cp_LiFePO4(T, best_params) fir T in T_JJ]

fig = plot.fig(r'Temperature$~\left[K\right]$', r'$C_P~\left[J/K-mol-at\right]$')

fig.add_set(T_exp, Cp_exp, label = 'exp', type='dots')
fig.add_set(T_ph, Cp_ph, label = 'phonon', type='dash')
fig.add_set(T_JJ, Cp_JJ, label = 'Murnaghan', type='line')
fig.add_set(T_JJ_fit, Cp_JJ_fit, label = 'Murnaghan+fitted', type='line')
fig.plot(show=True)
```

The resulting figure is:

Simultaneous parameter adjusting to experimental heat capacity and thermal expansion at P = 0 and prediction of thermodynamic phase equilibria at high pressure

Similarly to the previous example, a genetic algorithm was implemented to adjust model parameters fitting experimental data. The compound studied was Mg$_2$SiO$_4$ in the $alpha$, $beta$, and $gamma$ phases (forsterite, wadsleyite, and ringwoodite) with structures Pnma, Imma, and Fd3m, respectively, for temperatures from 0 to $2500~K$ and pressures from 0 to $30~GPa$. In this usage example, the isobaric heat capacity and the thermal expansion were fitted simultaneously at 0 pressure. For that, the objective function should simultaneously evaluate the thermal expansion and heat capacity as:

```
def Cp_alpha_Mg2SiO4(T, params):
    V0, K0, K0p, nu, a0, m0, s0, s1, s2, edef, sdef = params
    p_intanh = a0, m0
    p_anh = s0, s1, s2
    initial_parameters =  [-6.745375544e+05, V0, K0, K0p]
    eos_MU.fitEOS([V0], 0, initial_parameters=initial_parameters, fit=False)
    p_EOS = eos_MU.pEOS
    p_electronic = [0,0,0,0]
    Tmelting = 800
    p_defects = edef, sdef, Tmelting, 0.1
    m = 0.02253677142857143
```

(continues on next page)

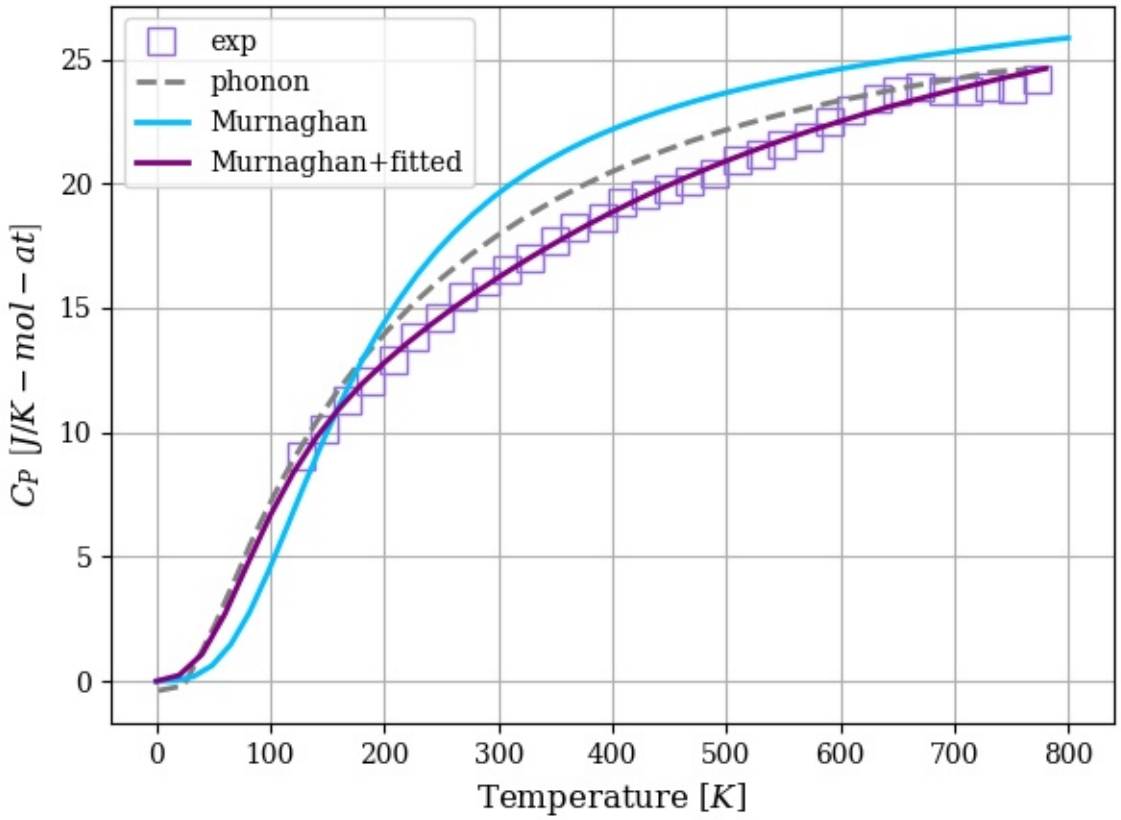


Fig. 10: LiFePO4 heat capacity.

(continued from previous page)

```
    ndeb_MU = nDeb(nu, m, p_intanh, eos_MU, p_electronic,
                   p_defects, p_anh, mode='jj)
    T, V = ndeb_MU.min_G(T, p_EOS[1], P=0)
    tprops_dict = ndeb_MU.eval_props(T, V, P=0)
    return [tprops_dict['a'], tprops_dict['Cp']]
```

The genetic algorithms remains the same as the previous example except for the evaluation function which now takes target data for both thermal expansion and heat capacity.

```
def evaluate(fc, T_set1, T_set2, pi, yexp, yexp2):
    evalfunc1 = fc(T_set1, pi, eval='min')
    evalfunc2 = fc(T_set2, pi, eval='min')
    try:
        errtotal1 = np.sqrt(np.sum(((evalfunc1[0] - yexp) / yexp) ** 2)) / len(T_set1)
        errtotal2 = np.sqrt(np.sum(((evalfunc2[1] - yexp2) / yexp2) ** 2)) / len(T_set2)
        return errtotal1 + errtotal2
    except:
        return 1e10
```

Once the optimal parameters for the three phases are obtained, the calculation of the thermodynamic properties can be calculated as function of the temperature and pressure as:

```
Ps = gen_Ps(0, 30e9, n_vals)

tprops_dict = []

# Pressure loop:
for P in Ps:
    # minimization of the free energy:
    T, V = ndeb.min_G(Ts, V0, P=P)
    # evaluation of the thermodynamic properties:
    tprops_dict.append(ndeb.eval_props(T, V, P=P))
```

In order to access the Gibbs free energy of each phase we use the key `G` in the `tprops_dict` list. Note that this list stores, for each pressure, a dictionary with all the thermodynamic properties.

```
G_alpha = np.zeros((len(Ts), len(Ps)))
G_beta = np.zeros((len(Ts), len(Ps)))
G_gamma = np.zeros((len(Ts), len(Ps)))
for i in range(len(Ts)):
    for j in range(len(Ps)):
        G_alpha[i, j] = tprops_dict_alpha[j]['G'][i]
        G_beta[i, j] = tprops_dict_beta[j]['G'][i]
        G_gamma[i, j] = tprops_dict_gamma[j]['G'][i]
```

To evaluate the stability relative to these three phases, the Gibbs free energy of each of them is compared:

```
G_z = np.zeros((len(Ts), len(Ps)))
for i in range(len(Ts)):
    for j in range(len(Ps)):
        G_list = [tprops_dict_alpha[j]['G'][i], tprops_dict_beta[j]['G'][i], tprops_dict_
↪gamma[j]['G'][i]]
```

(continues on next page)

(continued from previous page)

```
        print(G_list)
        G_z[j,i] = G_list.index(min(G_list)) +1
```

This can be plotted in a P vs T predominance diagram:

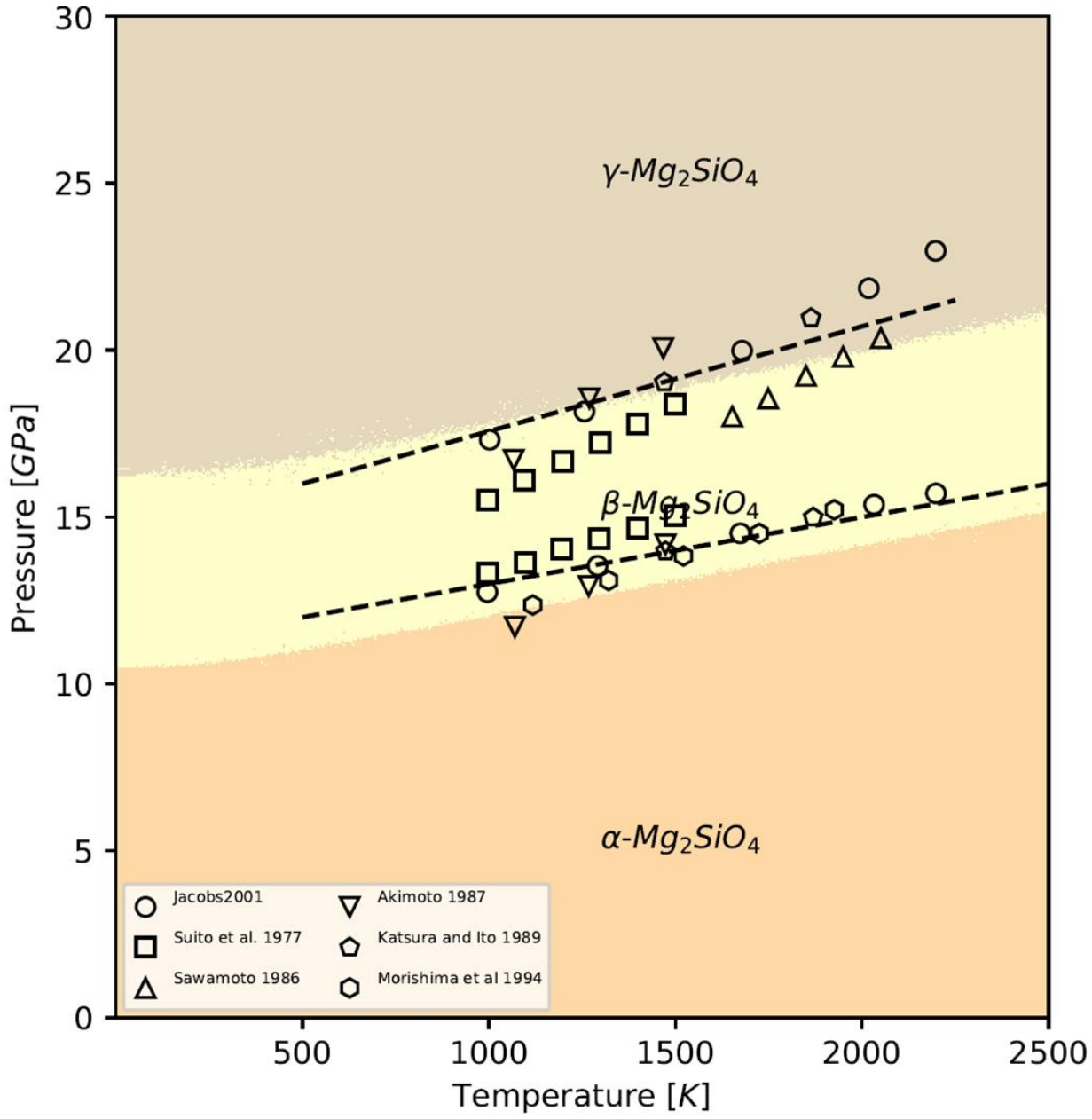


Fig. 11: Phase diagram P versus T for the , and forms of Mg2SiO4. Symbols are literature data for the phase stability regions boundaries.

1.3.4 Thermodynamic Properties

Table of contents

EOS parametrization

The EOS's

The EOS implemented are:

- Rose-Vinet (RV)
- Tight-binding second-moment-approximation (TB-SMA)
- Third order Birch-Murnaghan (BM3)
- Mie-Gruneisen (MG)
- Murnaghan (Mu1)
- Poirier-Tarantola (PT)
- Fourth order Birch-Murnaghan (BM4)
- Second order Murnaghan (Mu2)

Two description of the internal energy through inter-atomic potentials has been included as well:

- Morse
- EAM

The parameters can be entered by the user or fitted if there is data available.

Example

```
>>> import numpy as np
>>> import debyetools.potentials as potentials
>>> V_data = np.array([11.89,12.29,12.70,13.12,13.55,13.98,14.43,14.88,15.35,15.82,16.31,
↪16.80,17.31,17.82,18.34,18.88,19.42,19.98,20.54,21.12,21.71])*(1E-30*6.02E+23)
>>> E_data = np.array([-2.97,-3.06,-3.14,-3.20,-3.26,-3.30,-3.33,-3.36,-3.37,-3.38,-3.38,
↪-3.38,-3.37,-3.36,-3.34,-3.32,-3.30,-3.27,-3.24,-3.21,-3.17])*(1.60218E-19 * 6.
↪02214E+23)
>>> params_initial_guess = [-3e5, 1e-5, 7e10, 4]
>>> Birch_Murnaghan = potentials.BM()
>>> Birch_Murnaghan.fitEOS(V_data, E_data, params_initial_guess)
array([-3.26551e+05,9.82096e-06,6.31727e+10,4.31057e+00])
```

Source code

The following is the source for the description of the third order Birch-Murnaghan EOS:

class debyetools.potentials.**BM**(**args*, *units='J/mol'*, *parameters=''*)

Third order Birch-Murnaghan EOS and derivatives.

E0(*V: float | numpy.ndarray*) → float | numpy.ndarray

Internal energy.

Parameters
V (*float | np.ndarray*) – Volume.

Returns
E0(V)

Return type
float|np.ndarray

E04min(*V: float*, *pEOS: ndarray*) → float

Energy for minimization.

Parameters

- **V** (*float*) – Volume.
- **pEOS** (*np.ndarray*) – parameters.

Returns
E0.

Return type
float

d2E0dV2_T(*V: float | numpy.ndarray*) → float | numpy.ndarray

Internal energy second volume derivative.

Parameters
V (*float | np.ndarray*) – Volume.

Returns
d2E0dV2_T(V)

Return type
float|np.ndarray

d3E0dV3_T(*V: float | numpy.ndarray*) → float | numpy.ndarray

Internal energy third volume derivative.

Parameters
V (*float | np.ndarray*) – Volume.

Returns
d3E0dV3_T(V)

Return type
float|np.ndarray

d4E0dV4_T(*V: float | numpy.ndarray*) → float | numpy.ndarray

Internal energy fourth volume derivative.

Parameters
V (*float | np.ndarray*) – Volume.

Returns
d4E0dV4_T(V)

Return type
float|np.ndarray

d5E0dV5_T(*V: float | numpy.ndarray*) → float | numpy.ndarray

Internal energy fifth volume derivative.

Parameters
V (*float | np.ndarray*) – Volume.

Returns
d5E0dV5_T(V)

Return type
float|np.ndarray

d6E0dV6_T(*V: float | numpy.ndarray*) → float | numpy.ndarray

Internal energy sixth volume derivative.

Parameters
V (*float | np.ndarray*) – Volume.

Returns
d6E0dV6_T(V)

Return type
float|np.ndarray

dE0dV_T(*V: float | numpy.ndarray*) → float | numpy.ndarray

Internal energy volume derivative.

Parameters
V (*float | np.ndarray*) – Volume.

Returns
dE0dV_T(V)

Return type
float|np.ndarray

error2min(*P: ndarray*, *Vdata: ndarray*, *Edata: ndarray*) → ndarray

Error for minimization.

Parameters

- **P** (*np.ndarray*) – E0 parameters.
- **Vdata** (*np.ndarray*) – Volume data.
- **Edata** (*np.ndarray*) – Energy data.

Returns
Error.

Return type
np.ndarray

fitEOS(*Vdata: ndarray*, *Edata: ndarray*, *initial_parameters: ndarray = None*, *fit: bool = True*) → None

Parameters fitting.

Parameters

- **Vdata** (*np.ndarray*) – Input volume data.
- **Edata** (*np.ndarray*) – Target energy data.
- **initial_parameters** (*np.ndarray*) – Initial guess.
- **fit** (*bool.*) – True to run the fitting. False to just use the input parameters.

Returns
Optimal parameters.

Return type
np.ndarray

The other potentials are:

class debyetools.potentials.**RV**(**args*, *units='J/mol'*, *parameters=''*)

Rose-Vinet EOS and derivatives.

class debyetools.potentials.**TB**(**args*, *units='J/mol'*, *parameters=''*)

Thight-Binding second-order-approximation EOS and derivatives.

class debyetools.potentials.**MG**(**args*, *units='J/mol'*, *parameters=''*)

Mie-Gruneisen EOS and derivatives.

class debyetools.potentials.**MU**(**args*, *units='J/mol'*, *parameters=''*)

Murnaghan EOS and derivatives.

class debyetools.potentials.**PT**(**args*, *units='J/mol'*, *parameters=''*)

Poirier-Tarantola EOS and derivatives.

class debyetools.potentials.**BM4**(**args*, *units='J/mol'*, *parameters=''*)

Fourth order Birch-Murnaghan EOS and derivatives.

class debyetools.potentials.**MU2**(**args*, *units='J/mol'*, *parameters=''*)

Second order Murnaghan EOS and derivatives.

class debyetools.potentials.**MP**(**args*, *units='J/mol'*, *parameters=''*, *prec=10*)

Morse potential and derivatives.

Parameters

- **args** (*list*) – formula, primitive_cell, basis_vectors, cutoff, number_of_neighbor_levels.
- **parameters** (*list_of_floats*) – Morse potential parameters.

class debyetools.potentials.**EAM**(**args*, *units='J/mol'*, *parameters=''*)

EAM potential and derivatives.

Parameters

- **args** (*Tuple[str,np.ndarray,np.ndarray, float, int]*) – formula, primitive_cell, basis_vectors, cutoff, number_of_neighbor_levels.
- **parameters** (*np.ndarray*) – EAM potential parameters.

Poisson's ratio

The Poisson's ratio used in the calculation of the Debye temperature can entered manually by the user or calculated from the Elastic moduli matrix.

Example

```
>>> from debyetools.poisson import poisson_ratio
>>> EM = EM = load_EM(folder_name+'/OUTCAR.eps')
>>> nu = poisson_ratio(EM)
```

Source Code

debyetools.poisson.**poisson_ratio**(*EM: ndarray*, *quiet: bool = False*) → Union[float, Tuple[float, float, float, float, float, float, float, float]]

Calculation of the Poisson's ratio from elastic moduli matrix.

Parameters

- **EM** (*np.ndarray*) – Elastic moduli matrix.
- **quiet** (*bool*) – (optional) If verbose.

Returns
Poisson's ratio.

Return type
float

debyetools.poisson.**quiet_pa**(*EM: ndarray*) → Tuple[float, float, float, float, float, float, float, float]

Calculation of the Poisson's ratio from elastic moduli matrix.

Parameters
EM (*np.ndarray*) – Elastic moduli matrix.

Returns
BR, BV, B, GR, GV, S, AU, nu

Return type
Tuple[float,float,float,float,float,float,float,float]

Thermodynamic Properties

The thermodynamic properties are calculated by first creating the instance of a `nDeb` object. Once the parametrization is complete and the temperature and volumes are know (this can be done using the `nDeb.min_F` method), there is just an evaluation of the thermodynamic properties left using `nDeb.eval_tprops`.

Example: Minimization of the free energy.

```
>>> from debyetools.ndeb import nDeb
>>> from debyetools import potentials
>>> from debyetools.aux_functions import gen_Ts,load_V_E
>>> m = 0.021971375
>>> nu = poisson_ratio (EM)
>>> p_electronic = fit_electronic(V_data, p_el_inittial, E, N, Ef)
>>> p_defects = [8.46, 1.69, 933, 0.1]
>>> p_anh, p_intanh = [0,0,0], [0, 1]
>>> V_data, E_data = load_V_E('/path/to/SUMMARY', '/path/to/CONTCAR')
>>> eos = potentials.BM()
>>> peos = eos.fitEOS(V_data, E_data, params_initial_guess)
>>> ndeb = nDeb (nu , m, p_intanh , eos , p_electronic , p_defects , p_anh )
>>> T = gen_Ts ( T_initial , T_final , 10 )
>>> T, V = ndeb.min_G (T,  1e-5, P=0)
>>> V
array([9.98852539e-06, 9.99974297e-06, 1.00578469e-05, 1.01135875e-05,
       1.01419825e-05, 1.02392921e-05, 1.03467847e-05, 1.04650048e-05,
       1.05953063e-05, 1.07396467e-05, 1.09045695e-05, 1.10973163e-05])
```

Example: Evaluation of the thermodynamic properties:

```
>>> trprops_dict=ndeb.eval_props(T,V)
>>> tprops_dict['Cp']
array([4.02097531e-05, 9.68739597e+00, 1.96115210e+01, 2.25070513e+01,
       2.34086394e+01, 2.54037595e+01, 2.68478029e+01, 2.82106379e+01,
       2.98214145e+01, 3.20143195e+01, 3.51848547e+01, 3.98791392e+01])
```

Source code

class debyetools.ndeb.**nDeb**(*nu: float*, *m: float*, *p_intanh: ndarray*, *EOS: object*, *p_electronic: ndarray*, *p_defects: ndarray*, *p_anh: ndarray*, **args: object*, *units: str = 'J/mol'*, *mode: str = 'jjsl'*)

Instantiate an object that contains all the parameters for the evaluation of the thermodynamic properties of a certain element or compound. Also contains the method that implements an original Debye formalism for the calculation of the thermodynamic properties.

Parameters

- **nu** (*float*) – Poisson's ratio.
- **m** (*float*) – mass in Kg/mol-at
- **p_intanh** (*np.ndarray*) – Intrinsic anharmonicity parameters: a0, m0, V0.

- **EOS** (*object*) – Equation of state instance.
- **p_electronic** (*np.ndarray*) – Electronic contribution parameters.
- **p_defects** (*np.ndarray*) – Mono-vacancies defects contribution parameters: Evac00,Svac00,Tm,a,P2,V0.
- **p_anh** (*np.ndarray*) – Excess contribution parameters.
- **mode** (*str*) – Type of approximation of the Debye temperature (see vibrational contribution).

eval_Cp(*T: ndarray*, *V: ndarray*, *P=None*) → dict

Evaluates the Heat capacity of a given compound/element at (T,V).

Parameters

- **T** (*np.ndarray*) – The temperature in Kelvin.
- **V** (*np.ndarray*) – The volume in "units".

Returns

A dictionary with the following keys: 'T': temperature, 'V': volume, 'tD': Debye temperature, 'g': Gruneisen parameter, 'Kt': isothermal bulk modulus, 'Ktp': pressure derivative of the isothermal bulk modulus, 'Ktpp': second order pressure derivative of the isothermal bulk modulus, 'Cv': constant-volume heat capacity, 'a': thermal expansion, 'Cp': constant-pressure heat capacity, 'Ks': adiabatic bulk modulus , 'Ksp': pressure derivative of the adiabatic bulk modulus, 'G': Gibbs free energy, 'E': total internal energy, 'S': entropy, 'E0': 'cold' internal energy defined by the EOS, 'Fvib': vibrational free energy, 'Evib': vibrational internal energy, 'Svib': vibrational entropy, 'Cvvib': vibrational heat capacity, 'Pcold': 'cold' pressure, 'dPdT_V': (dP/dT)_V, 'G^2': Ktp**2-2*Kt*Ktpp, 'dSdP_T': (dS/dP)_T, 'dKtdT_P': (dKt/dT)_P, 'dadP_T': (da/dP)_T, 'dCpdP_T': (dCp/dP)_T, 'ddSdT_PdP_T': (d2S/dTdP).

Return type

dict

eval_props(*T: ndarray*, *V: ndarray*, *P=None*) → dict

Evaluates the thermodynamic properties of a given compound/element at (T,V).

Parameters

- **T** (*np.ndarray*) – The temperature in Kelvin.
- **V** (*np.ndarray*) – The volume in "units".

Returns

A dictionary with the following keys: 'T': temperature, 'V': volume, 'tD': Debye temperature, 'g': Gruneisen parameter, 'Kt': isothermal bulk modulus, 'Ktp': pressure derivative of the isothermal bulk modulus, 'Ktpp': second order pressure derivative of the isothermal bulk modulus, 'Cv': constant-volume heat capacity, 'a': thermal expansion, 'Cp': constant-pressure heat capacity, 'Ks': adiabatic bulk modulus , 'Ksp': pressure derivative of the adiabatic bulk modulus, 'G': Gibbs free energy, 'E': total internal energy, 'S': entropy, 'E0': 'cold' internal energy defined by the EOS, 'Fvib': vibrational free energy, 'Evib': vibrational internal energy, 'Svib': vibrational entropy, 'Cvvib': vibrational heat capacity, 'Pcold': 'cold' pressure, 'dPdT_V': (dP/dT)_V, 'G^2': Ktp**2-2*Kt*Ktpp, 'dSdP_T': (dS/dP)_T, 'dKtdT_P': (dKt/dT)_P, 'dadP_T': (da/dP)_T, 'dCpdP_T': (dCp/dP)_T, 'ddSdT_PdP_T': (d2S/dTdP).

Return type

dict

f2min(*T: float*, *V: float*, *P: float*) → float

free energy for minimization.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.
- **P** (*float*) – Pressure.

Returns
Free energy.

Return type
float

min_G(*T: ndarray*, *initial_V: float*, *P: float*) → Tuple[ndarray, ndarray]

Procedure for the calculation of the volume as function of temperature.

Parameters

- **T** (*list_of_floats*) – Temperature.
- **initial_V** (*float*) – initial guess.
- **P** (*float*) – Pressure.

Returns
Temperature and Volume

Return type
Tuple[np.ndarray,np.ndarray]

1.3.5 Contributions to the free energy

Table of contents

Anharmonicity

The anharmonicity can be included in the calculations as an excess contribution which is called just 'anharmonicity'. The temperature dependence of the phonon frequencies con be introduced using what its called 'intrinsic anharmonicity'.

Source code

class debyetools.anharmonicity.**Anharmonicity**(*s0: float*, *s1: float*, *s2: float*)

Instance for the excess contribution to the free energy.

Parameters
s0,s1,s2 (*float*) – Parameters of the A(V) term.

A(*V: float*) → float

A(V) = s0+s0*V+s1*V**2, where A is the polynomial model for the excess contribution to the free energy, A(V)*T.

Parameters
V (*float*) – Volume

Returns
s0+s0*V+s1*V**2

Return type
float

E(*T*, *V: float*) → float

Internal energy due the excess term.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
A(V)*T**2/2

Return type
float

F(*T*, *V: float*) → float

Free energy due the excess term.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
-A(V)*T**2/2

Return type
float

S(*T*, *V: float*) → float

Entropy due the excess term.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
A(V)*T

Return type
float

d2AdV2_T(*V: float*) → float

Second order volume derivative of A at fixed T.

Parameters
V (*float*) – Volume

Returns
2*s2

Return type
float

d2FdT2_V(*T*, *V: float*) → float

Second order Temperature derivative of the free energy due the excess term, at fixed V.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
-A(V)

Return type
float

d2FdV2_T(*T*, *V: float*) → float

Second order volume derivative of the free energy due the excess term, at fixed T.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
-(d2A(V)/dV2)_T*T**2/2

Return type
float

d2FdVdT(*T*, *V: float*) → float

Second order derivative of the free energy due the excess term, with respect to T and V.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
-(dA(V)/dV)_T*T

Return type
float

d3AdV3_T(*V: float*) → float

Third order volume derivative of A at fixed T.

Parameters
V (*float*) – Volume

Returns
0

Return type
float

d3FdV2dT(*T*, *V: float*) → float

Third order derivative of the free energy due the excess term, with respect to T, V, and V.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
-(d2A(V)/dV2)_T*T

Return type
float

d3FdV3_T(*T*, *V: float*) → float

Third order volume derivative of the free energy due the excess term, at fixed T.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
-(d3A(V)/dV3)_T*T**2/2

Return type
float

d3FdVdT2(*T*, *V: float*) → float

Third order derivative of the free energy due the excess term, with respect to T, T, and V.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
-(dA(V)/dV)_T

Return type
float

d4AdV4_T(*V: float*) → float

Fourth order volume derivative of A at fixed T.

Parameters
V (*float*) – Volume.

Returns
0

Return type
float.

d4FdV4_T(*T*, *V: float*) → float

Fourth order volume derivative of the free energy due the excess term, at fixed T.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
-(d4A(V)/dV4)_T*T**2/2

Return type
float

dAdV_T(*V: float*) → float

Volume derivative of A at fixed T.

Parameters
V (*float*) – Volume

Returns
s1+2*V*s2

Return type
float

dFdT_V(*T*, *V: float*) → float

Temperature derivative of the free energy due the excess term, at fixed V.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
-A(V)

Return type
float

dFdV_T(*T*, *V: float*) → float

Volume derivative of the free energy due the excess term, at fixed T.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
-(dA(V)/dV)_T*T**2/2

Return type
float

Defects

The defects due to mono-vacancies can be taken into account if the parameters are provided.

Source code

class debyetools.defects.**Defects**(*Evac00: float*, *Svac00: float*, *Tm: float*, *a: float*, *P2: float*, *V0: float*)

Implementation of the defects contribution due to monovancies to the free energy.

Parameters

- **Evac00** (*float*) – Fomration energy of vacancies.
- **Svac00** (*float*) – Fomration entropy of vacancies.
- **Tm** (*float*) – Melting temperature.
- **a** (*float*) – Volume ratio of a mono vacancie relative to the equilibrium volume.
- **P2** (*float*) – Bulk modulus.
- **V0** (*float*) – Equilibrium volume.

E(*T: float*, *V: float*) → float

Defects energy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume

Returns
E_def

Return type
float

Evac(*V: float*) → float

Enthalpy of formation of vacancies.

Parameters
V (*float*) – Volume.

Returns
Ef(V)

Return type
float

F(*T: float*, *V: float*) → float

Implementation of the defects contribution to the free energy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
energy

Return type
float

S(*T: float*, *V: float*) → float

Defects entropy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume

Returns
S_def

Return type
float

Svac(*V: float*) → float

Entropy of formation of vacancies.

Parameters
V (*float*) – Volume.

Returns
Svac0

Return type
float

d2EvacdV2_T(*V: float*) → float

Volume-derivative of the enthalpy of formation of vacancies.

Parameters
V (*float*) – Volume.

Returns
Volume-derivative of the enthalpy of formation of vacancies.

Return type
float

d2FdT2_V(*T: float*, *V: float*) → float

Derivative of defects contribution to the free energy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
Derivative of F_def

Return type
float

d2FdV2_T(*T: float*, *V: float*) → float

Derivative of defects contribution to the free energy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
Derivative of F_def

Return type
float

d2FdVdT(*T: float*, *V: float*) → float

Derivative of defects contribution to the free energy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
Derivative of F_def

Return type
float

d2SvacdV2_T(*V: float*) → float

Volume-derivative of the entropy of formation of vacancies.

Parameters
V (*float*) – Volume.

Returns
0

d3EvacdV3_T(*V: float*) → float

Volume-derivative of the enthalpy of formation of vacancies.

Parameters
V (*float*) – Volume.

Returns
Volume-derivative of the enthalpy of formation of vacancies.

Return type
float

d3FdV2dT(*T: float*, *V: float*) → float

Derivative of defects contribution to the free energy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
Derivative of F_def

Return type
float

d3FdV3_T(*T: float*, *V: float*) → float

Derivative of defects contribution to the free energy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
Derivative of F_def

Return type
float

`d3FdVdT2`(*T: float*, *V: float*) → float

Derivative of defects contribution to the free energy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
Derivative of F_def

Return type
float

`d3SvacdV3_T`(*V: float*) → float

Volume-derivative of the entropy of formation of vacancies.

Parameters
V (*float*) – Volume.

Returns
0

`d4EvacdV4_T`(*V: float*) → float

Volume-derivative of the enthalpy of formation of vacancies.

Parameters
V (*float*) – Volume.

Returns
Volume-derivative of the enthalpy of formation of vacancies.

Return type
float

`d4FdV4_T`(*T: float*, *V: float*) → float

Derivative of defects contribution to the free energy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
Derivative of F_def

Return type
float

`d4SvacdV4_T`(*V: float*) → float

Volume-derivative of the entropy of formation of vacancies.

Parameters
V (*float*) – Volume.

Returns
0

dEvacdV_T(*V: float*) → float

Volume-derivative of the enthalpy of formation of vacancies.

Parameters
V (*float*) – Volume.

Returns
Volume-derivative of the enthalpy of formation of vacancies.

Return type
float

dFdT_V(*T: float*, *V: float*) → float

Derivative of defects contribution to the free energy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
Derivative of F_def

Return type
float

dFdV_T(*T: float*, *V: float*) → float

Derivative of defects contribution to the free energy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
Derivative of F_def

Return type
float

dSvacdV_T(*V: float*) → float

Volume-derivative of the entropy of formation of vacancies.

Parameters
V (*float*) – Volume.

Returns
0

Electronic Contribution

In order to take the electronic contribution intro account an approximation of the electronic DOS evaluated at the volume dependent Fermi level is implemented. The parameters can be entered manually or fitted to DOS data from DFT calculations.

Source code

class debyetools.electronic.**Electronic**(**params: ndarray*)

Implementation of the electronic contribution to the free energy.

Parameters
params (*float*) – N(Ef)(V) function parameters.

E(*T: float | numpy.ndarray*, *V: float | numpy.ndarray*) → float | numpy.ndarray

Electronic energy

Parameters

- **T** (*float | np.ndarray*) – Temperature.
- **V** (*float | np.ndarray*) – Volume.

Returns
E_el

Return type
float|np.ndarray

F(*T: float | numpy.ndarray*, *V: float | numpy.ndarray*) → float | numpy.ndarray

Electronic contribution to the free energy.

Parameters

- **T** (*float | np.ndarray*) – Temperature.
- **V** (*float | np.ndarray*) – Volume.

Returns
F_el

Return type
float|np.ndarray

NfV(*V: float*) → float

N(Ef)(V)

Parameters
V (*float*) – Volume.

Returns
N(Ef)(V)

Return type
float

S(*T: float | numpy.ndarray*, *V: float | numpy.ndarray*) → float | numpy.ndarray

Electronic entropy.

Parameters

- **T** (*float | np.ndarray*) – Temperature.
- **V** (*float | np.ndarray*) – Volume.

Returns
S_el

Return type
float|np.ndarray

d2FdT2_V(*T: float | numpy.ndarray*, *V: float | numpy.ndarray*) → float | numpy.ndarray

Derivative of the electronic contribution to the free energy.

Parameters

- **T** (*float|np.ndarray*) – Temperature.
- **V** (*float|np.ndarray*) – Volume.

Returns
F_el

Return type
float|np.ndarray

d2FdV2_T(*T: float | numpy.ndarray*, *V: float | numpy.ndarray*) → float | numpy.ndarray

Derivative of the electronic contribution to the free energy.

Parameters

- **T** (*float|np.ndarray*) – Temperature.
- **V** (*float|np.ndarray*) – Volume.

Returns
F_el

Return type
float|np.ndarray

d2FdVdT(*T: float | numpy.ndarray*, *V: float | numpy.ndarray*) → float | numpy.ndarray

Derivative of the electronic contribution to the free energy.

Parameters

- **T** (*float|np.ndarray*) – Temperature.
- **V** (*float|np.ndarray*) – Volume.

Returns
F_el

Return type
float|np.ndarray

d2NfVdV2_T(*V: float*) → float

derivative of N(Ef)(V)

Parameters
V (*float*) – Volume.

Returns
derivative of N(Ef)(V)

Return type
float

d3FdV2dT(*T: float | numpy.ndarray*, *V: float | numpy.ndarray*) → float | numpy.ndarray

Derivative of the electronic contribution to the free energy.

Parameters

- **T** (*float|np.ndarray*) – Temperature.
- **V** (*float|np.ndarray*) – Volume.

Returns
F_el

Return type
float|np.ndarray

d3FdV3_T(*T: float | numpy.ndarray*, *V: float | numpy.ndarray*) → float | numpy.ndarray

Derivative of the electronic contribution to the free energy.

Parameters

- **T** (*float|np.ndarray*) – Temperature.
- **V** (*float|np.ndarray*) – Volume.

Returns
F_el

Return type
float|np.ndarray

d3FdVdT2(*T: float | numpy.ndarray*, *V: float | numpy.ndarray*) → float | numpy.ndarray

Derivative of the electronic contribution to the free energy.

Parameters

- **T** (*float|np.ndarray*) – Temperature.
- **V** (*float|np.ndarray*) – Volume.

Returns
F_el

Return type
float|np.ndarray

d3NfVdV3_T(*V: float*) → float

derivative of N(Ef)(V)

Parameters
V (*float*) – Volume.

Returns
derivative of N(Ef)(V)

Return type
float

d4FdV4_T(*T: float | numpy.ndarray*, *V: float | numpy.ndarray*) → float | numpy.ndarray

Derivative of the electronic contribution to the free energy.

Parameters

- **T** (*float|np.ndarray*) – Temperature.
- **V** (*float|np.ndarray*) – Volume.

Returns
F_el

Return type
float|np.ndarray

d4NfVdV4_T(*V: float*) → float

derivative of N(Ef)(V)

Parameters
V (*float*) – Volume.

Returns
derivative of N(Ef)(V)

Return type
float

dFdT_V(*T: float | numpy.ndarray*, *V: float | numpy.ndarray*) → float | numpy.ndarray

Derivative of the electronic contribution to the free energy.

Parameters

- **T** (*float | np.ndarray*) – Temperature.
- **V** (*float | np.ndarray*) – Volume.

Returns
F_el

Return type
float|np.ndarray

dFdV_T(*T: float | numpy.ndarray*, *V: float | numpy.ndarray*) → float | numpy.ndarray

Derivative of the electronic contribution to the free energy.

Parameters

- **T** (*float | np.ndarray*) – Temperature.
- **V** (*float | np.ndarray*) – Volume.

Returns
F_el

Return type
float|np.ndarray

dNfVdV_T(*V: float*) → float

derivative of N(Ef)(V)

Parameters
V (*float*) – Volume.

Returns
derivative of N(Ef)(V)

Return type
float

debyetools.electronic.**NfV2m**(*P: ndarray*, *Vdata: ndarray*, *NfVdata: ndarray*) → ndarray

Error function for minimizaiton.

Parameters

- **P** (*np.ndarray*) – Parameters.
- **Vdata** (*np.ndarray*) – Volume data.
- **NfVdata** (*np.ndarray*) – N(Ef)(V) data.

Returns
err.

Return type
np.ndarray

debyetools.electronic.**NfV_poly_fun**(*V: float*, *_A: float*, *_B: float*, *_C: float*, *_D: float*) → float

Polynomial model for N(Ef)(V) for min.

Parameters

- **V** (*float*) – Volume/
- **_A** (*float*) – param.
- **_B** (*float*) – param.
- **_C** (*float*) – param.
- **_D** (*float*) – param.

Returns
polynimial for minimization.

Return type
float

debyetools.electronic.**fit_electronic**(*Vs: ndarray*, *p_el: ndarray*, *E: ndarray*, *N: ndarray*, *Ef: ndarray*, *ixss: int = 8*, *ixse: int = -1*) → ndarray

Fitting procedure for the N(Ef)(V) function.

Parameters

- **Vs** (*np.ndarray*) – Volumes.
- **p_el** (*np.ndarray*) – Initial parameters.
- **E** (*np.ndarray*) – Matrix with energies at each level for each volume.
- **N** (*np.ndarray*) – Matrix with densities of state at each level for each volume.
- **Ef** (*np.ndarray*) – Fermi levels as function of temperature.
- **ixse** (*int*) – (optional) eDOS subset index.
- **intixss** – (optional) eDOS subset index.

Retun np.ndarray
optimized parameters.

Vibrational

The evaluation of the thermal behavior of compounds are calculating using the Debye approximation. The mass of the compound and the Poisson's ration must be entered as input parameters. The information about the internal energy is passed as an `potential.EOS` object.

Source code

class debyetools.vibrational.**Vibrational**(*nu: float*, *EOS_obj: object*, *m: float*, *intanh: ndarray*, *mode: str*)

Instantiate the vibrational contribution to the free energy and its derivatives for the calculation of the thermodynamic properties.

Parameters

- **nu** (*float*) – Poisson's ratio.
- **EOS_obj** (*potential_instance*) – Equation of state object.
- **m** (*float*) – Mass in Kg/mol-at.
- **intanh** (*intAnharmonicity_instance*) – Intrinsic anharmonicity object.

F(*T: float*, *V: float*) → float

Vibration Helmholtz free energy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
F_vib.

Return type
float

d2FdT2_V(*T: float*, *V: float*) → float

Derivative of vibrational Helmholtz free energy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
d2FdT2_V.

Return type
float

d2FdV2_T(*T: float*, *V: float*) → float

Derivative of vibrational Helmholtz free energy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
d2FdV2_T.

Return type
float

d2FdVdT(*T: float*, *V: float*) → float

Derivative of vibrational Helmholtz free energy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
d2FdVdT.

Return type
float

d3FdV2dT(*T: float*, *V: float*) → float

Derivative of vibrational Helmholtz free energy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
d3FdV2dT.

Return type
float

d3FdV3_T(*T: float*, *V: float*) → float

Derivative of vibrational Helmholtz free energy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
d3FdV3_T.

Return type
float

d3FdVdT2(*T: float*, *V: float*) → float

Derivative of vibrational Helmholtz free energy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
d3FdVdT2.

Return type
float

d4FdV4_T(*T: float*, *V: float*) → float

Derivative of vibrational Helmholtz free energy.

Parameters

- **T** (*float*) – Temperature.

- **V** (*float*) – Volume.

Returns
d4FdV4_T.

Return type
float

dFdT_V(*T: float*, *V: float*) → float

Derivative of vibrational Helmholtz free energy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
dFdT_V.

Return type
float

dFdV_T(*T: float*, *V: float*) → float

Derivative of vibrational Helmholtz free energy.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

Returns
dFdV_T.

Return type
float

set_int_anh(*T: float*, *V: float*) → None

Calculates intrinsic anharmonicity correction to the Debye temperature and its derivatives.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

set_int_anh_4minF(*T: float*, *V: float*) → None

Calculates intrinsic anharmonicity correction to the Debye temperature and its derivatives.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

set_theta(*T: float*, *V: float*) → None

Calculates the Debye Temperature and its derivatives.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

set_theta_4minF(*T: float*, *V: float*) → None

Calculates the Debye Temperature and its derivatives.

Parameters

- **T** (*float*) – Temperature.
- **V** (*float*) – Volume.

1.3.6 FactSage compound database parametrization

The calculated thermodynamic properties for each EOS selected are used to fit the models for heat capacity, thermal expansion, bulk modulus and its pressure derivative. The resulting parameters can be used in FactSage as a compound database.

Example

```
>>> from debyetools.fs_compound_db import fit_FS
>>>
>>> T_from = 298.15
>>> T_to = 1000
>>> FS_db_params = fit_FS(tprops_dict, T_from, T_to)
>>> print(FS_db_params)
[ 1.11898466e+02 -8.11995443e-02  7.22119591e+05  4.29282477e-05
 -1.31482568e+03  1.00000000e+00]
```

Source code

debyetools.fs_compound_db.**fit_FS**(*tprops: dict*, *T_from: float*, *T_to: float*) → dict

Procedure for the fitting of FS compound database parametes to thermodynamic properties.

Parameters

- **tprops** (*dict*) – Dictionary with the thermodynamic properties.
- **T_from** (*float*) – Initial temperature.
- **T_to** (*float*) – Final temperature.

Returns

Dictionary with the optimal parameters.

Return type

dict

1.3.7 Pair Analysis Calculation

A pair analysis calculator was implemented where the atom position can be read from a POSCAR file or entered manually. The neighbor list and binning are calculated to build up the pairs list.

Example

```
>>> from debyetools.pairanalysis import pair_analysis
>>> formula = 'AaAaBbAa'
>>> cutoff = 10
>>> basis_vectors = np.array([[0,0,0], [.5,.5,0], [.5,0,.5], [0,.5,.5]])
>>> primitive_cell = np.array([[4,0,0] ,[0,4,0] ,[0,0,4]])
>>> pa_result = pair_analysis(formula, cutoff, basis_vectors, primitive_cell)
>>> distances , num_pairs_per_formula , combs_types = pa_result
>>> print('distances | # of pairs per type')
>>> print('          | '+  ' '.join(['%s' for _ in combs_types])%tuple(combs_types))
>>> for d, n in zip( distances , num_pairs_per_formula ):
...     print(' %.6f ' % (d) + '| ' + ' '.join([' %.2f' for _ in n])%tuple(n))
...
distances | # of pairs per type
          | Aa-Aa  Aa-Bb  Bb-Bb
 2.828427 |  6.00  6.00  0.00
 4.000000 |  4.50  0.00  1.50
 4.898979 |  12.00  12.00  0.00
 5.656854 |  9.00  0.00  3.00
 6.324555 |  12.00  12.00  0.00
 6.928203 |  6.00  0.00  2.00
 7.483315 |  24.00  24.00  0.00
 8.000000 |  4.50  0.00  1.50
 8.485281 |  18.00  18.00  0.00
 8.944272 |  18.00  0.00  6.00
 9.380832 |  12.00  12.00  0.00
 9.797959 |  18.00  0.00  6.00
```

Source code

debyetools.pairanalysis.**neighbor_list**(*size: ndarray*, *basis: ndarray*, *cell: ndarray*, *cutoff: float*) → Tuple[ndarray, ndarray, ndarray, ndarray, ndarray]

calculate a list i, j, dij where i and j are a pair of atoms of indexes i and j, respectively, and dij is the distance between them.

Parameters

- **size** (*np.ndarray*) – Number of times we are replicating the primitive cel
- **basis** (*np.ndarray*) – atoms position within a single primitive cell
- **cell** (*np.ndarray*) – the primitive cell
- **cutoff** (*float*) – cut-off distance

Returns

D, I , J

Return type
Tuple[np.ndarray, np.ndarray, np.ndarray]

debyetools.pairanalysis.**pair_analysis**(*atom_types*, *cutoff*, *basis*, *cell*, *prec=10*, *full=False*)

run a pair analysis of a crystal structure of almost any type of symmetry.

Parameters

- **atom_types** (*str*) – the types of each atom in the primitive cell in the same order as the basis vectors.
- **cutoff** (*float*) – cut-off distance
- **basis** (*np.ndarray*) – atoms position within a single primitive cell
- **cell** (*np.ndarray*) – the primitive cell
- **prec** (*int*) – precision.
- **full** (*boolean*) – if True, returns also data of ghost cells.

Returns
pair distance, pair number, pair types

Return type
Tuple[np.ndarray,np.ndarray,np.ndarray]

1.3.8 Input file formats

Table of contents

Folder structure

The input files must be located in the same folder of a given compound and that folder have to be ideally be named by the formula followed by the space group, for example for Al2O3 R3c the folder name would be *Al2O3_R3c*. There is also a naming rule for the input files. *SUMMARY.fcc* contains the output total energies for the NVT calculations. *CONTCAR.5* contain the cell coordinates and basis vectors of the crystal structure. *OUTCAR.eps* is the output of the calculations of the elastic properties and have the Elastic moduli tensor. 'DOSCAR.EvV.x' are a set of files containing the electronic DOS, and are outputs of the NVT calculations.

```
-- Al_fcc
        \
         |-- CONTCAR.5
         |-- DOSCAR.EvE.0.01
         |-- DOSCAR.EvE.0.02
         |-- DOSCAR.EvE.0.03
         |-- DOSCAR.EvE.0.04
```

(continues on next page)

(continued from previous page)

```
          |-- DOSCAR.EvE.0.05
          |-- DOSCAR.EvE.-0.00
          |-- DOSCAR.EvE.-0.01
          |-- DOSCAR.EvE.-0.02
          |-- DOSCAR.EvE.-0.03
          |-- DOSCAR.EvE.-0.04
          |-- DOSCAR.EvE.-0.05
          |-- OUTCAR.eps
          |-- SUMMARY.fcc
```

NPT calculations

CONTCAR.5

```
Al4
   1.00000000000000
     4.0396918604376202    0.0000000000000000    0.0000000000000000
     0.0000000000000000    4.0396918604376202    0.0000000000000000
     0.0000000000000000    0.0000000000000000    4.0396918604376202
   Al
     4
Direct
  0.0000000000000000  0.0000000000000000  0.0000000000000000
  0.0000000000000000  0.5000000000000000  0.5000000000000000
  0.5000000000000000  0.0000000000000000  0.5000000000000000
  0.5000000000000000  0.5000000000000000  0.0000000000000000

  0.00000000E+00  0.00000000E+00  0.00000000E+00
  0.00000000E+00  0.00000000E+00  0.00000000E+00
  0.00000000E+00  0.00000000E+00  0.00000000E+00
  0.00000000E+00  0.00000000E+00  0.00000000E+00
```

NVT calculations

SUMMARY.fcc:

```
-0.10 1 F= -.12910797E+02 E0= -.12906918E+02 d E =-.775954E-02 mag= -0.0007
-0.09 1 F= -.13370262E+02 E0= -.13366237E+02 d E =-.804932E-02 mag= -0.0011
-0.08 1 F= -.13758391E+02 E0= -.13754102E+02 d E =-.857638E-02 mag= -0.0010
-0.07 1 F= -.14083100E+02 E0= -.14078383E+02 d E =-.943565E-02 mag= -0.0003
-0.06 1 F= -.14349565E+02 E0= -.14344316E+02 d E =-.104987E-01 mag= 0.0009
-0.05 1 F= -.14563748E+02 E0= -.14558028E+02 d E =-.114391E-01 mag= 0.0019
-0.04 1 F= -.14729909E+02 E0= -.14723867E+02 d E =-.120830E-01 mag= 0.0026
-0.03 1 F= -.14853318E+02 E0= -.14847102E+02 d E =-.124328E-01 mag= 0.0029
-0.02 1 F= -.14936018E+02 E0= -.14929721E+02 d E =-.125950E-01 mag= 0.0028
-0.01 1 F= -.14983118E+02 E0= -.14976823E+02 d E =-.125893E-01 mag= 0.0022
-0.00 1 F= -.14997956E+02 E0= -.14991733E+02 d E =-.124452E-01 mag= 0.0010
0.01 1 F= -.14984456E+02 E0= -.14978268E+02 d E =-.123746E-01 mag= -0.0002
0.02 1 F= -.14945044E+02 E0= -.14938772E+02 d E =-.125429E-01 mag= -0.0007
0.03 1 F= -.14882846E+02 E0= -.14876412E+02 d E =-.128695E-01 mag= -0.0007
```

(continues on next page)

(continued from previous page)

```
0.04 1 F= -.14799945E+02 E0= -.14793346E+02 d E =-.131984E-01 mag= -0.0006
0.05 1 F= -.14698295E+02 E0= -.14691567E+02 d E =-.134551E-01 mag= -0.0005
0.06 1 F= -.14581670E+02 E0= -.14574842E+02 d E =-.136561E-01 mag= -0.0004
0.07 1 F= -.14451232E+02 E0= -.14444380E+02 d E =-.137049E-01 mag= -0.0006
0.08 1 F= -.14310295E+02 E0= -.14303519E+02 d E =-.135523E-01 mag= -0.0012
0.09 1 F= -.14158570E+02 E0= -.14151879E+02 d E =-.133818E-01 mag= -0.0015
0.10 1 F= -.13997863E+02 E0= -.13991236E+02 d E =-.132555E-01 mag= -0.0016
```

Extract of *DOCAR.EvV.-0.01*.

```
  4   4   1   0
 0.1599154E+02  0.3999295E-09  0.3999295E-09  0.3999295E-09  0.5000000E-15
 1.00000000000000E-004
 CAR
unknown system
    20.23257380     -4.13078501  301      8.31659488      1.00000000
    -4.131  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    -4.050  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    -3.968  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    -3.887  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    -3.806  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    -3.725  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    -3.644  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    -3.562  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    -3.481  0.5925E-06  0.5691E-06  0.4812E-07  0.4622E-07
    .
    .
    .
    19.339  0.3717E-02  0.3257E-02  0.1200E+02  0.1200E+02
    19.420  0.6633E-03  0.5821E-03  0.1200E+02  0.1200E+02
    19.502  0.9200E-04  0.9769E-04  0.1200E+02  0.1200E+02
    19.583  0.0000E+00  0.0000E+00  0.1200E+02  0.1200E+02
    19.664  0.0000E+00  0.0000E+00  0.1200E+02  0.1200E+02
    19.745  0.0000E+00  0.0000E+00  0.1200E+02  0.1200E+02
    19.827  0.0000E+00  0.0000E+00  0.1200E+02  0.1200E+02
    19.908  0.0000E+00  0.0000E+00  0.1200E+02  0.1200E+02
    19.989  0.0000E+00  0.0000E+00  0.1200E+02  0.1200E+02
    20.070  0.0000E+00  0.0000E+00  0.1200E+02  0.1200E+02
    20.151  0.0000E+00  0.0000E+00  0.1200E+02  0.1200E+02
    20.233  0.0000E+00  0.0000E+00  0.1200E+02  0.1200E+02
    20.23257380     -4.13078501  301      8.31659488      1.00000000
    -4.131  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    -4.050  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    -3.968  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    -3.887  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
```

(continues on next page)

(continued from previous page)

```
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
   -3.806  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
   -3.725  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
   -3.644  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
   -3.562  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
   -3.481  0.4086E-07  0.3921E-07  0.8085E-25  0.1185E-24  0.8085E-25  0.1185E-24  0.
↪8085E-25  0.1185E-24  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
   -3.400  0.5808E-06  0.5573E-06  0.5778E-09  0.5548E-09  0.5778E-09  0.5548E-09  0.
↪5778E-09  0.5548E-09  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
   -3.319  0.4242E-05  0.4071E-05  0.7578E-08  0.7277E-08  0.7578E-08  0.7277E-08  0.
↪7578E-08  0.7277E-08  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    .
    .
    .
   18.852  0.9848E-05  0.1001E-04  0.7738E-04  0.7627E-04  0.7738E-04  0.7627E-04  0.
↪7738E-04  0.7627E-04  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
   18.933  0.2834E-04  0.3565E-04  0.2534E-03  0.2897E-03  0.2534E-03  0.2897E-03  0.
↪2534E-03  0.2897E-03  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
   19.014  0.7157E-04  0.9188E-04  0.7132E-03  0.7921E-03  0.7132E-03  0.7921E-03  0.
↪7132E-03  0.7921E-03  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
   19.096  0.7540E-04  0.9274E-04  0.1027E-02  0.9899E-03  0.1027E-02  0.9899E-03  0.
↪1027E-02  0.9899E-03  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
   19.177  0.3376E-04  0.3686E-04  0.7680E-03  0.6530E-03  0.7680E-03  0.6530E-03  0.
↪7680E-03  0.6530E-03  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
   19.258  0.8935E-05  0.8866E-05  0.2940E-03  0.2522E-03  0.2940E-03  0.2522E-03  0.
↪2940E-03  0.2522E-03  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
   19.339  0.1897E-05  0.1825E-05  0.7109E-04  0.6212E-04  0.7109E-04  0.6212E-04  0.
↪7109E-04  0.6212E-04  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
   19.420  0.1630E-06  0.1508E-06  0.1297E-04  0.1138E-04  0.1297E-04  0.1138E-04  0.
↪1297E-04  0.1138E-04  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
   19.502  0.6965E-11  0.2036E-11  0.1834E-05  0.1949E-05  0.1834E-05  0.1949E-05  0.
↪1834E-05  0.1949E-05  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
```

(continues on next page)

(continued from previous page)

```
    19.583  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    19.664  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    19.745  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    19.827  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    19.908  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    19.989  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    20.070  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    20.151  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    20.233  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    20.23257380     -4.13078501  301       8.31659488      1.00000000
    -4.131  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    -4.050  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    -3.968  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    -3.887  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    -3.806  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    -3.725  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    -3.644  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    -3.562  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
```

(continues on next page)

(continued from previous page)

```
    -3.481  0.4086E-07  0.3921E-07  0.8085E-25  0.1185E-24  0.8085E-25  0.1185E-24  0.
↪8085E-25  0.1185E-24  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    -3.400  0.5808E-06  0.5573E-06  0.5778E-09  0.5548E-09  0.5778E-09  0.5548E-09  0.
↪5778E-09  0.5548E-09  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    .
    .
    .
    19.502  0.6965E-11  0.2036E-11  0.1834E-05  0.1949E-05  0.1834E-05  0.1949E-05  0.
↪1834E-05  0.1949E-05  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    19.583  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    19.664  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    19.745  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    19.827  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    19.908  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    19.989  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    20.070  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    20.151  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    20.233  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.
↪0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
```

Elastic properties calculations

Extract of *OUTCAR.eps*.

```
ELASTIC MODULI CONTR FROM IONIC RELAXATION (kBar)
Direction    XX          YY          ZZ          XY          YZ          ZX
--------------------------------------------------------------------------------
XX           0.0000      0.0000      0.0000      0.0000      0.0000     -0.0000
YY           0.0000      0.0000      0.0000     -0.0000      0.0000     -0.0000
ZZ           0.0000      0.0000     -0.0000      0.0000      0.0000     -0.0000
XY           0.0000     -0.0000      0.0000      0.0000     -0.0000     -0.0000
```

(continues on next page)

(continued from previous page)

```
YZ           0.0000      0.0000      0.0000     -0.0000      0.0000     -0.0000
ZX          -0.0000     -0.0000     -0.0000     -0.0000     -0.0000      0.0000
-------------------------------------------------------------------------------


TOTAL ELASTIC MODULI (kBar)
Direction     XX          YY          ZZ          XY          YZ          ZX
-------------------------------------------------------------------------------
XX         4520     1500     1070      200.0000     -0.0000      0.0000
YY         1500     4520     1070     -200.0000     -0.0000      0.0000
ZZ         1070     1070     4540     -0.0000       0.0000       0.0000
XY         200      -200    -0        1320         -0.0000     -0.0000
YZ         -0      -0        0       -0            1320      200
ZX          0       0        0       -0            200     1510
-------------------------------------------------------------------------------
```

Direct inputs

Instead of files, the input data can be entered directly as *numpy.arrays* or, in the case if the GUI, as plain text. For example, the energy curve for Al_3Li $L1_2$ in eV/A^3 per atom:

```
#V  E
1.188839e+01        -2.971644e+00
1.228909e+01        -3.061733e+00
1.269870e+01        -3.138113e+00
1.311730e+01        -3.202274e+00
1.354501e+01        -3.255126e+00
1.398191e+01        -3.297945e+00
1.442811e+01        -3.331133e+00
1.488370e+01        -3.355852e+00
1.534878e+01        -3.372588e+00
1.582345e+01        -3.382024e+00
1.630781e+01        -3.384997e+00
1.680195e+01        -3.382070e+00
1.730598e+01        -3.373913e+00
1.781999e+01        -3.360960e+00
1.834407e+01        -3.343770e+00
1.887833e+01        -3.322563e+00
1.942286e+01        -3.298003e+00
1.997777e+01        -3.270464e+00
2.054315e+01        -3.240472e+00
2.111909e+01        -3.208295e+00
2.170570e+01        -3.174068e+00
```

The elastic constants for Al_3Li $L1_2$ in GPa:

```
122.26  35.00  35.00 -0.00  0.00  0.00
 35.00 122.22  34.95 -0.00  0.00 -0.00
 35.00  34.95 122.22  0.00  0.00  0.00
 -0.00  -0.00   0.00 41.47  0.00  0.00
 0.00    0.00   0.00  0.00 41.39  0.00
```

(continues on next page)

(continued from previous page)

```
 0.00   -0.00   0.00  0.00  0.00 41.47
```

CHAPTER

TWO

INDICES

- genindex
- modindex

PYTHON MODULE INDEX

d

INDEX

E

F

M

N

P

Q

R

S

T

V