panther Documentation

Release 0.5.2

Lukasz Mentel

Nov 23, 2017

Contents

CHAPTER 1

Contents:

1.1 Installation

1.1.1 Dependencies

- numpy
- scipy
- pandas
- matplotlib
- seaborn
- future
- six
- writeBmat based on the approach presented in[1]
- lxml

The recommended installation method is with pip. The latest version can be installed directly from bitbucket repository:

```
pip install https://bitbucket.org/lukaszmentel/panther/get/tip.tar.gz
```

or cloned first

```
hg clone https://lukaszmentel@bitbucket.org/lukaszmentel/panther
```

and installed via

[1] Bucko, T., Hafner, J., & Angyan, J. G. (2005). Geometry optimization of periodic systems using internal coordinates. The Journal of Chemical Physics, 122(12), 124508. doi:10.1063/1.1864932

```
pip install -U [--user] ./panther
```

1.2 User guide

Available CLI programs:

- *panther* format conversion, calculation of harmonic and anharmonic frequencies
- *plotmode* visualiztion of vibrational potential per mode
- *writemodes* conversion of geometry files to collections of modes

1.2.1 panther

The panther script takes two command line argument

```
$ panther

usage: panther [-h] {convert,harmonic,anharmonic} config

positional arguments:
  {convert,harmonic,anharmonic}
                        choose what to do
  config                file with the configuration parameters for thermo

optional arguments:
  -h, --help            show this help message and exit
```

The input file is in the standard condif file format and contains three sections `conditions`, `job` and `system` defining the parameters.

conditions

pressure [float] Pressure in MPa

Tinitial [float] Smallest Temperature in K

Tfinal [float] Largest temperatue in K

Tstep [float] Temperature step for temperature grid (in K)

```
[conditions]
Tinitial = 303.15
Tfinal = 403.15
Tstep = 10.0
pressure = 0.1
```

job

translations [bool] If True the translational degrees of freedom will be projected out from the hessian

rotations [bool] If True the translational degrees of freedom will be projected out from the hessian

code [str] Program to use for single point calcualtions

```
[job]
translations = true
rotations = false
code = vasp
```

system

pointgroup [str] Point group symbol of the system

phase [str] Phase of the system, either `gas` or `solid`

```
[system]
pointgroup = Dooh
phase = gas
```

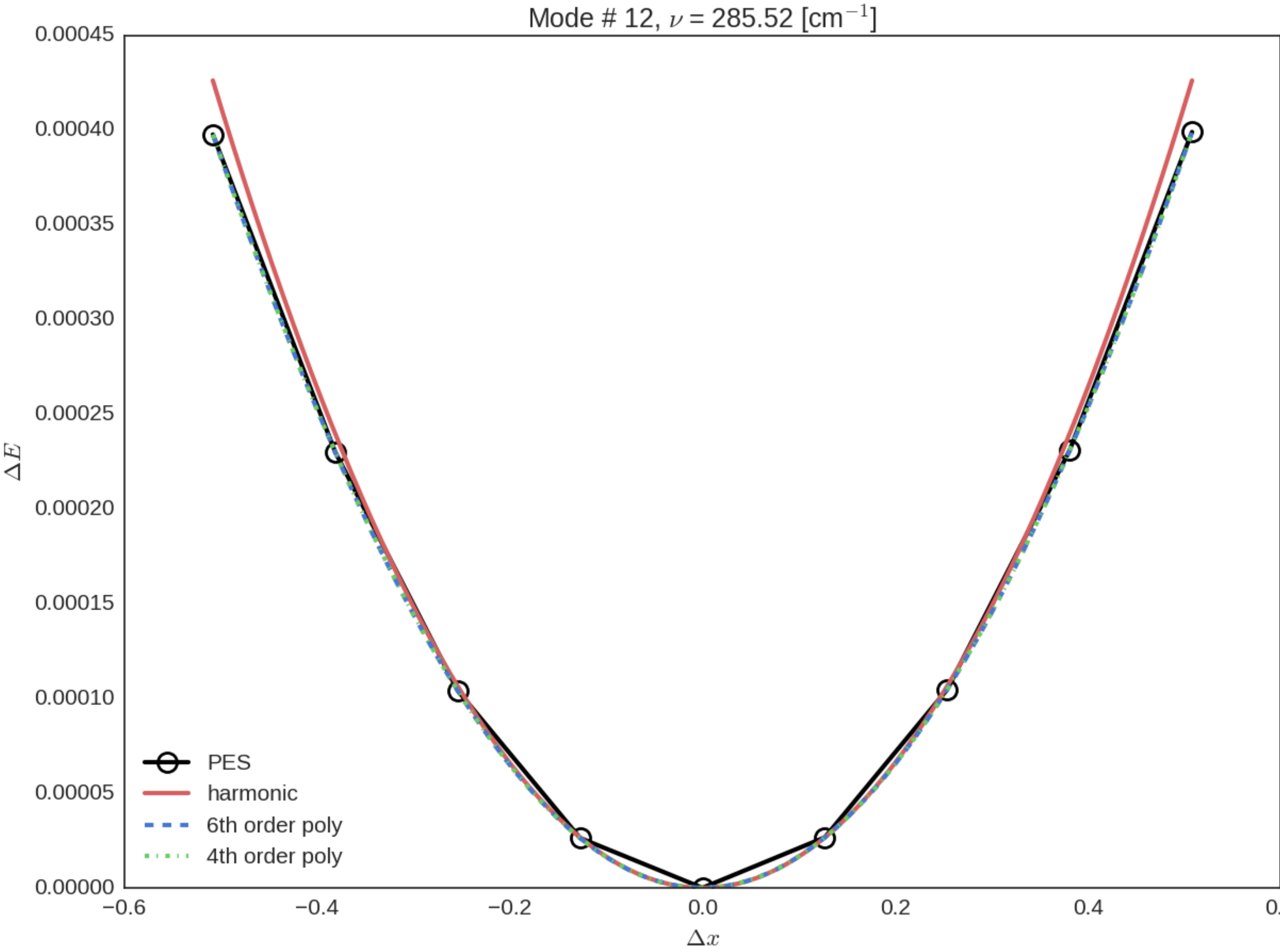
1.2.2 plotmode

```
$ plotmode -h

usage: plotmode [-h] [-s SIXTH] [-f FOURTH] [-p PES] [-o OUTPUT] mode

positional arguments:
  mode                  number of the mode to be printed

optional arguments:
  -h, --help            show this help message and exit
  -s SIXTH, --sixth SIXTH
                        file with sixth order polynomial fit,
                        default="em_freq"
  -f FOURTH, --fourth FOURTH
                        file with fourth order polynomial fit,
                        default="em_freq_4th"
  -p PES, --pes PES     file with the potential energy surface (PES),
                        default="test_anharm"
  -o OUTPUT, --output OUTPUT
                        name of the output file
```

Example

Provided that the default files `em_freq`, `em_freq_4th` and `test_anharm` are present to plot the last mode only requires the argument `12`

```
plotmode 12
```



1.2.3 writemodes

This program takes the single file with continuous geometries in VASP POSCAR format as input and writes separate file in ASE trajectory format per node to a specified directory.

```
$ writemodes -h

usage: writemodes [-h] [-d DIR] filename

positional arguments:
  filename           name of the file with geometries, default="POSCARs"

optional arguments:
  -h, --help         show this help message and exit
  -d DIR, --dir DIR  directory to put the modes, default="modes"
```

Example

Provided that the POSCARs file exists we can generate trajectory files with the modes with:

```
writemodes POSCARs
```

which produces the mode.X.traj files in the modes directory where X is the mode number.

We can now generate a set of PNG files representing the snapshots of the mode by:

```
from ase.io import read, write
modes = read('mode.1.traj', index=':')

for i, mode in enumerate(modes):
    write('{0:0>3d}.pov'.format(i), mode, run_povray=True, rotation='90x', canvas_
↪width=800)
```

To see the animation we can create an GIF file from the previously generated PNG files using the convert program from the ImageMagick package:

```
convert -delay 15 -loop 0 *.png mode1-animation.gif
```

1.3 Tutorials

1.3.1 Harmonic and Anharmonic Thermochemistry

This tutorial explains step by step how to use the panther package to calculate the thermodynamic functions of molecules and solids making use of the standard harmonic vibrational analysis and anharmonic vibrations in the independent mode approximation as explained in detail in references[1],[2],[3].

For each step some of the data will be explicitly printed to illustrate the underlying data structures, however in production runs such level of verbosity is not necessary.

The methanol molecule will be used as and example and VASP code will be used to perform the calculations, however since panther is interfaced with ASE any of the supported calculators can be used instead with appropriate modifications.

Structure relaxation

First of all the molecule needs to be relaxed and it is recommended to converge the forces below 1.0e-5 eV/A. Here the initial structure is read from the methanol.xyz file and the structure relaxation is performed using the LBFGS method implemented in ASE instead of the internal VASP optimizers.

```
import ase.io
from ase.calculators.vasp import Vasp
from ase.optimizers import LBFGS
```

[1] Piccini, G., Alessio, M., Sauer, J., Zhi, Y., Liu, Y., Kolvenbach, R., Jentys, A., Lercher, J. A. (2015). Accurate Adsorption Thermodynamics of Small Alkanes in Zeolites. Ab initio Theory and Experiment for H-Chabazite. The Journal of Physical Chemistry C, 119(11), 6128–6137. doi:10.1021/acs.jpcc.5b01739

[2] Piccini, G., & Sauer, J. (2014). Effect of anharmonicity on adsorption thermodynamics. Journal of Chemical Theory and Computation, 10, 2479–2487. doi:10.1021/ct500291x

[3] Piccini, G., & Sauer, J. (2013). Quantum Chemical Free Energies: Structure Optimization and Vibrational Frequencies in Normal Modes. Journal of Chemical Theory and Computation, 9(11), 5038–5045. doi:10.1021/ct4005504

```
meoh = ase.io.read('methanol.xyz')

calc = Vasp(
        prec='Accurate',
        gga='PE',
        lreal=False,
        ediff=1.0e-8,
        encut=600.0,
        nelmin=5,
        nsw=1,
        nelm=100,
        ediffg=-0.001,
        ismear=0,
        ibrion=-1,
        nfree=2,
        isym=0,
        lvdw=True,
        lcharg=False,
        lwave=False,
        istart=0,
        npar=2,
        ialgo=48,
        lplane=True,
        ispin=1,
)

meoh.set_calculator(calc)

optimizer = LBFGS(meoh, trajectory='relaxed.traj',
                  restart='lbfgs.pkl', logfile='optimizer.log')

optimizer.run(fmax=0.00001)
```

Hessian matrix

Having optimized the structure we will also need the hessian matrix. We will use internal VASP mode (*IBRION=5*) to generate the hessian using cartesian coordinate displacements, therefore we need to update the calculator's parameters. After the hessian is calculated it is read from the OUTCAR symmetrized, converted to atomic units and saved as a numpy array for convenience

```
from panther.io import read_vasp_hessian

# adjust the calcualtor argument for hessian calculation
calc.set(ibrion=5, potim=0.02)

calc.calculate(meoh)

hessian = read_vasp_hessian('OUTCAR', symmetrize=True, convert2au=True, negative=True)
np.save('hessian', hessian)
```

1.3.2 Harmonic vibrations and thermochemistry

We can now use the hessian to calculate thermochemical functions in the harmonic oscillator approximation, starting by calcualting the frequencies and normal modes

```
from panther.vibrations import harmonic_vibrational_analysis

frequencies, normal_modes = harmonic_vibrational_analysis(hessian, meoh,
            proj_translations=True, proj_rotations=True, ascomplex=False)
```

The resulting frequencies are in atomic units and need to be converted to Joules and passed to `Thermochemistry` to calculate thermochemical functions

```
from scipy.constants import value, Planck
from panther.thermochemistry import Thermochemistry

vibenergies = Planck * frequencies.real * value('hartree-hertz relationship')
vibenergies = vibenergies[vibenergies > 0.0]

thermo = Thermochemistry(vibenergies, meoh, phase='gas', pointgroup='Cs')
thermo.summary(T=273.15, p=0.1)
```

```
================ THERMOCHEMISTRY =================

   @ T = 273.15 K  p =   0.10 MPa

--------------------------------------------------
Partition functions:
ln q                    :          23.802
    ln q_translational  :          15.574
    ln q_rotational     :           7.949
    ln q_vibrational    :           0.280
--------------------------------------------------
Enthalpy (H)            :         140.014  kJ/mol
    H translational     :           3.407  kJ/mol
    H rotational        :           3.407  kJ/mol
    H vibrational       :         130.930  kJ/mol
        @ 0 K (ZPVE)    :         129.733  kJ/mol
        @ 273.15 K      :           1.197  kJ/mol
        pV              :           2.271  kJ/mol
--------------------------------------------------------------------------
                                                             *T
Entropy (S)             :           0.2355 kJ/mol*K        64.3395 kJ/mol
    S translational     :           0.1503 kJ/mol*K        41.0476 kJ/mol
    S rotational        :           0.0786 kJ/mol*K        21.4591 kJ/mol
    S vibrational       :           0.0067 kJ/mol*K         1.8328 kJ/mol
--------------------------------------------------------------------------
U - T*S                 :          75.6749 kJ/mol
--------------------------------------------------
Electronic energy       :       -2918.9516 kJ/mol
```

1.3.3 Normal Mode Relaxation

In some cases it is advantegeous to refine the structure using displacements along normal modes of vibrations, such a functionality is provided through the `NormalModeBFGS` which is based on the Optimizer class from the ASE pakckage. The method requires an initial guess for the hessian matrix for which we'll reuse the hessian calculated in one of the previous steps, however in this case the hessian should be in eV/Angstrom^2 units, therefore it needs to be converted.

```
from panther.nmrelaxation import NormalModeBFGS

from scipy.constants import angstrom, value

ang2bohr = angstrom / value('atomic unit of length')
ev2hartree = value('electron volt-hartree relationship')

hessian = hessian * (ang2bohr**2) /ev2hartree

# create the optimizer
optimizer = NormalModeBFGS(meoh, 'gas', hessian, logfile='optimizer.log',
                           trajectory='relaxed.traj', proj_translations=True,
                           proj_rotations=True)

# start the relaxation
optimizer.run(fmax=0.001)
```

1.3.4 Anharmonic Thermochemistry

Internal coordinate displacements

With frequencies and normal modes we can further generate a grid of displacements along each normal mode using internal coordinates to improve the sampling of the potential energy surface. This is done using the `calculate_displacements` function. The function returns a nested `OrderedDict` of structures as ase.Atoms objects with mode number and displacement sample number as keys. For example if `npoints=4` is given as an argument there will be 8 structures per mode labeled with numbers 1, 2, 3, 4, -1, -2, -3, -4 signifying the direction and the magnitude of the displacement.

```
from panther.displacements import calculate_displacements

images, modeinfo = calculate_displacements(meoh, hessian, frequencies, normal_modes,␣
↪npoints=4)

print(modeinfo.to_string())

           HOfreq  effective_mass displacement is_stretch vibration     P_stretch     ␣
↪    P_bend     P_torsion  P_longrange
mode
0     3748.362703     1944.298846      3.05268       True      True  1.000538e+00  9.
↪633947e-07  3.581071e-08          0.0
1     3033.988514     2003.111476      3.39309       True      True  1.000487e+00  1.
↪444002e-03  1.358614e-08          0.0
2     2956.839029     2015.939983      3.43707       True      True  1.000163e+00  2.
↪112104e-03  0.000000e+00          0.0
3     2897.901987     1886.235715      3.47185       True      True  1.002597e+00  4.
↪553844e-05  0.000000e+00          0.0
4     1445.646111     1896.588719      2.45777      False      True  2.423247e-04  8.
↪382427e-01  1.620089e-01          0.0
5     1430.743791     1910.050531      2.47054      False      True  5.828069e-07  7.
↪696362e-01  2.314594e-01          0.0
6     1413.372913     2064.662087      2.48567      False      True  1.101886e-05  1.
↪013066e+00  1.246737e-02          0.0
7     1320.779390     2344.971044      2.57133      False      True  2.648486e-03  8.
↪819130e-01  1.163030e-01          0.0
8     1122.078049     2310.188376      2.78972      False      True  8.856000e-04  8.
↪562025e-01  1.423684e-01          0.0
```

```
9     1043.856152    2998.235955      2.89236      False      True  4.590826e-01  4.
↪698373e-01  7.957428e-02        0.0
10     999.428001    4154.928137      2.95595      False      True  5.647864e-01  3.
↪944914e-01  4.924659e-02        0.0
11     276.606858    1950.430919      5.61876      False      True  5.101834e-06  1.
↪465507e-04  1.002540e+00        0.0
12       0.000000    8792.819312          inf      False     False           NaN  ␣
↪      inf           NaN          0.0
13       0.000000    4750.377947          inf       True     False           inf  ␣
↪      NaN           NaN          0.0
14       0.000000    6312.514911          inf       True     False           inf  ␣
↪      NaN           NaN          0.0
15       0.000000    5243.620927          inf       True     False           inf  ␣
↪      NaN           NaN          0.0
16       0.000000    2177.259022          inf      False     False           NaN  ␣
↪      inf           NaN          0.0
17       0.000000    8532.108968          inf       True     False           inf  ␣
↪      NaN           NaN          0.0
```

The function also returns `modeinfo` DataFrame with additional characteristics of the mode such as `displacement`, `is_stretch` and `effective_mass` and components of the vibrational population analysis.

Calculating energies for the displaced structures

Per each displaced structure we can calculate the energy, in this example using the VASP calculator again in the single point calculation mode

```
from panther.pes import calculate_energies

# set the calculator in single point mode
calc.set(ibrion=-1)

energies = calculate_energies(images, calc, modes='all')
```

This will return a DataFrame with `npoints * 2` energies per mode.

The `energies` are missing the equilibrium structure energy which can be easily set through

```
energies['E_0'] = meoh.get_potential_energy()

print(energies.to_string())

           E_-4       E_-3       E_-2       E_-1        E_0        E_1        E_2    ␣
↪ E_3        E_4
0  -29.838210 -29.999174 -30.129845 -30.219158 -30.252801 -30.212111 -30.072575 -29.
↪801815 -29.357050
1  -29.887274 -30.033740 -30.148793 -30.224943 -30.252801 -30.220603 -30.113614 -29.
↪913317 -29.596341
2  -29.765209 -29.983697 -30.134831 -30.223555 -30.252801 -30.223573 -30.135003 -29.
↪984312 -29.766713
3  -29.880739 -30.032824 -30.149891 -30.225671 -30.252801 -30.222657 -30.125177 -29.
↪948635 -29.679366
4  -30.194580 -30.220226 -30.238397 -30.249217 -30.252801 -30.249248 -30.238656 -30.
↪221115 -30.196709
5  -30.196107 -30.220941 -30.238652 -30.249266 -30.252801 -30.249268 -30.238667 -30.
↪220985 -30.196211
6  -30.199391 -30.222460 -30.239171 -30.249354 -30.252801 -30.249264 -30.238446 -30.
↪219995 -30.193484
```

```
7  -30.202508 -30.224242 -30.239987 -30.249567 -30.252801 -30.249516 -30.239548 -30.
↪222731 -30.198903
8  -30.208386 -30.227840 -30.241715 -30.250030 -30.252801 -30.250030 -30.241714 -30.
↪227847 -30.208412
9  -30.209316 -30.228681 -30.242227 -30.250194 -30.252801 -30.250259 -30.242766 -30.
↪230511 -30.213676
10 -30.210619 -30.229477 -30.242607 -30.250294 -30.252801 -30.250378 -30.243261 -30.
↪231673 -30.215822
11 -30.241961 -30.246534 -30.249960 -30.252081 -30.252801 -30.252072 -30.249935 -30.
↪246486 -30.241889
```

Calculating the frequencies

Frequencies can now be calculated using the finite difference method implemented in *panther.pes.differentiate()* function and appended as a `frequency` column to the `modeinfo`. The returned `vibs` matrix contains four columns corresponding to derivatives calculated with the central formula using 2, 4, 6 and 8 points

```
from panther.pes import differentiate
from scipy.constants import value

dsp = modeinfo.loc[modeinfo['vibration'], 'displacement'].astype(float).values
vibs = differentiate(dsp, energies, order=2)

au2invcm = 0.01 * value('hartree-inverse meter relationship')
np.sqrt(vibs) * au2invcm

array([[ 3757.6949986 ,  3745.36173164,  3745.5117494 ,  3745.51978786],
       [ 3038.75202112,  3032.49074659,  3032.55620542,  3032.56159197],
       [ 2960.0679129 ,  2956.11388526,  2956.13869496,  2956.14205087],
       [ 2900.20373811,  2897.17093192,  2897.18620368,  2897.18888617],
       [ 1446.18374932,  1446.16517443,  1446.1824322 ,  1446.19041632],
       [ 1431.77027217,  1431.68206976,  1431.68116942,  1431.68294377],
       [ 1414.58204596,  1414.17987052,  1414.182009  ,  1414.18039006],
       [ 1321.14267911,  1321.22424463,  1321.24026442,  1321.2470148 ],
       [ 1122.70558461,  1122.64210276,  1122.63752157,  1122.63384929],
       [ 1043.87393741,  1043.78448965,  1043.78296923,  1043.78025961],
       [  999.41759187,   999.30803727,   999.31024481,   999.30971807],
       [  285.06040099,   285.79248818,   285.87505212,   285.9193547 ]])

# assign the frequencies fitted with 8 points to a frequency column
# in the modeinfo
modeinfo.loc[modeinfo['vibration'], 'frequency'] = (np.sqrt(vibs)*au2invcm)[:, 3]
```

Fitting the potentials

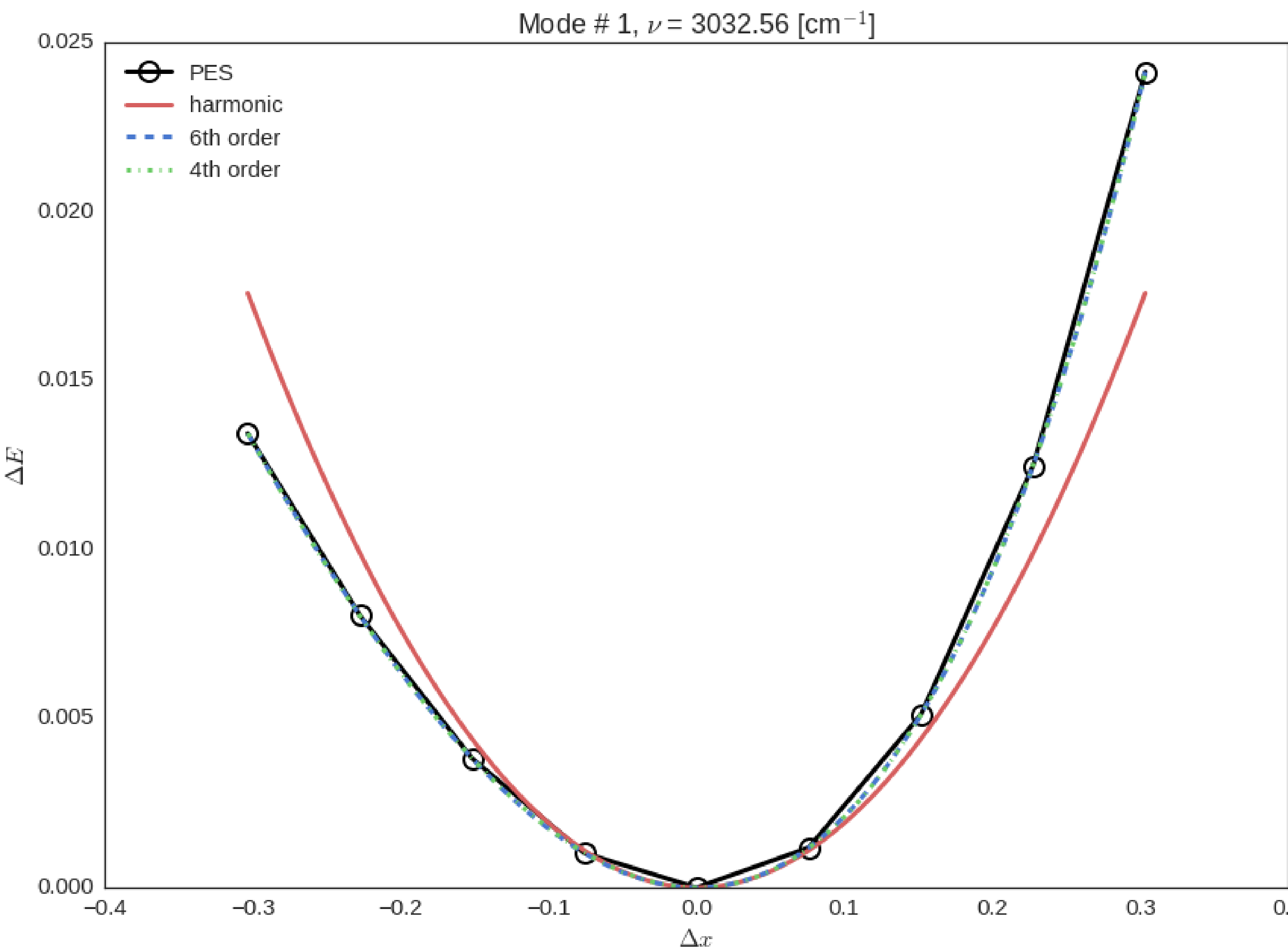
The last this is to fit the potential energy surfaces as 6th and 4th order polynomials

```
from panther.pes import fit_potentials

# fit the potentials on 6th and 4th order polynomials
c6o, c4o = fit_potentials(modeinfo, energies)
```

The two DataFrame objects `c6o` and `c4o` contain fitted polynomial coefficients for each mode. We can use the energies and the polynomial coefficients to plot the PES and the fitted potentials, here as an example, second mode

(*mode=1* since the modes are indexed from 0) is plotted

```
from panther.plotting import plotmode

plotmode(1, energies, modeinfo, c6o, c4o)
```



1.3.5 Anharmonic frequencies from 1-D Schrodinger Equation

Anharmonic frequencies are calculated first by solving the 1-D Schrodinger equation per mode as exaplained in reference[4] and then those frequencies are used to calculate the thermodynamic functions

```
from panther.anharmonicity import anharmonic_frequencies, harmonic_df, merge_vibs
from panther.thermochemistry import AnharmonicThermo

anh6o = anharmonic_frequencies(meoh, 273.15, c6o, modeinfo)
```

[4] Beste, A. (2010). One-dimensional anharmonic oscillator: Quantum versus classical vibrational partition functions. Chemical Physics Letters, 493(1-3), 200–205. doi:10.1016/j.cplett.2010.05.036

```
anh4o = anharmonic_frequencies(meoh, 273.15, c4o, modeinfo)

harmonicdf = harmonic_df(modeinfo, 273.15)
finaldf = merge_vibs(df6, df4, hdf, verbose=False)

at = AnharmonicThermo(fdf, meoh, phase='gas', pointgroup='Cs')
at.summary(T=273.15, p=0.1)
```

```
================ THERMOCHEMISTRY =================

  @ T = 273.15 K  p =   0.10 MPa

--------------------------------------------------
Partition functions:
ln q                      :         23.667
    ln qtranslational     :         15.574
    ln qrotational        :          7.949
    ln qvibrational       :          0.145
--------------------------------------------------
Enthalpy (H)              :        138.890  kJ/mol
    H translational       :          3.407  kJ/mol
    H rotational          :          3.407  kJ/mol
    H vibrational         :        129.806  kJ/mol
        @ 0 K (ZPVE)      :        129.675  kJ/mol
        @ 273.15 K        :          0.131  kJ/mol
        pV                :          2.271  kJ/mol
--------------------------------------------------------------------------
                                                               *T
Entropy (S)               :          0.2375 kJ/mol*K       64.8694 kJ/mol
    S translational       :          0.1503 kJ/mol*K       41.0476 kJ/mol
    S rotational          :          0.0786 kJ/mol*K       21.4591 kJ/mol
    S vibrational         :          0.0086 kJ/mol*K        2.3627 kJ/mol
--------------------------------------------------------------------------
H - T*S                   :         74.0209 kJ/mol
--------------------------------------------------
Electronic energy         :       -2918.9516 kJ/mol
```

1.4 API Reference

1.4.1 Thermochemistry

class panther.thermochemistry.**Thermochemistry**(*vibenergies*, *atoms*, **args*, ***kwargs*)

Calculate thermochemistry in harmonic approximation

Parameters **vibenergies** : numpy.array

Vibrational energies in Joules

atoms : ase.Atoms

Atoms obect

phase : str

Phase, should be either *gas* or *solid*

pointgroup : str

symmetrynumber : str

If *pointgroup* is specified *symmetrynumber* is obsolete, since it will be inferred from the *pointgroup*

get_enthalpy(*T=273.15*)
Return the enthalpy H

Parameters T : float

Temperature in *K*

get_entropy(*T=273.15*)
Return the entropy S

Parameters T : float

Temperature in *K*

get_heat_capacity(*T=273.15*)
Heat capacity at constant pressure

get_internal_energy(*T=273.15*)
Return the internal energy U

Parameters T : float

Temperature in *K*

get_qvibrational(*T=273.15*, *uselog=True*)
Calculate the vibrational partition function at temperature *T* in kJ/mol

Parameters T : float

Temperature in *K*

uselog : bool

When *True* return the natural logarithm of the partition function

Notes

$$q_{vib}(T) = \prod_{i=1}^{3N-6} \frac{1}{1 - \exp(-h\omega_i / k_B T)}$$

get_vibrational_energy(*T=273.15*)
Calculate the vibational energy correction at temperature *T* in kJ/mol

Parameters T : float

Temperature in *K*

Notes

$$U_{vib}(T) = \frac{R}{k_B} \sum_{i=1}^{3N-6} \frac{h\omega_i}{\exp(h\omega_i / k_B T) - 1}$$

get_vibrational_entropy(*T=273.15*)

Calculate the vibrational entropy at temperature *T* in kJ/mol

Parameters **T** : float

Temperature in *K*

Notes

$$S_{vib}(T) = R \sum_{i=1}^{3N-6} \left[\frac{h\omega_i}{k_B T (\exp(h\omega_i/k_B T) - 1)} - \ln(1 - \exp(-h\omega_i/k_B T)) \right]$$

get_vibrational_heat_capacity(*T=273.15*)

Return the heat capacity

Parameters **T** : float

Temperature in *K*

Notes

$$C_{p,vib}(T) = R \sum_{i=1}^{3N-6} \left(\frac{h\omega_i}{k_B T} \right)^2 \frac{\exp(-h\omega_i/k_B T)}{\left[1 - \exp(-h\omega_i/k_B T)\right]^2}$$

get_zpve()

Calculate the Zero Point Vibrational Energy (ZPVE) in kJ/mol

Notes

$$E_{\text{ZPV}} = \frac{1}{2} \sum_{i=1}^{3N-6} h\omega_i$$

summary(*T=273.15*, *p=0.1*)

Print summary with the thermochemical data at temperature *T* in kJ/mol

Parameters **T** : float

Temperature in *K*

p : float

Pressure in MPa

1.4.2 NormalModeBFGS

class `panther.nmrelaxation.`**`NormalModeBFGS`**(*atoms*, *phase*, *hessian=None*, *hessian_update='BFGS'*, *logfile='-'*, *trajectory=None*, *restart=None*, *proj_translations=True*, *proj_rotations=True*, *master=None*, *verbose=False*)

Normal mode optimizer with approximate hessian update

Parameters **atoms** : ase.Atoms

Atoms object with the structure to optimize

phase : str

Phase, should be either *gas* or *solid*

hessian : array_like (N, N)

Initial hessian matrix in eV/Angstrom^2

hessian_update : str

Name of the approximate formula to udpate hessian, one of: *BFGS*, *SR1*, *DFP*

proj_translations : bool

If `True` translational degrees of freedom will be projected from the hessian

proj_rotations : bool

If `True` rotational degrees of freedom will be projected from the hessian

logfile : str

Name log the log file

trajectory : str

Name of the trajectory file

`log`(*grad*, *grad_nm*, *step_nm*)

Print a line with convergence information

`log_header`()

Header for the log with convergence information

`run`(*fmax=0.05*, *steps=100000000*)

Run structure optimization algorithm.

This method will return when the forces on all individual atoms are less than *fmax* or when the number of steps exceeds *steps*.

`step`(*grad*)

Calculate the step in cartesian coordinates based on the step in normal modes in the rational function approximation (RFO)

Args:

grad [array_like (N,)] Current gradient

`update_hessian`(*coords*, *grad*)

Perform hessian update

Parameters **coords** : array_like (N,)

Current coordinates as vector

grad : array_like (N,)

Current gradient

1.4.3 panther.anharmonicity

Methods for solving the one dimentional vibrational eigenproblem

`panther.anharmonicity.`**`anharmonic_frequencies`**(*atoms*, *T*, *coeffs*, *modeinfo*)

Calculate the anharmonic frequencies

Parameters **atoms** : ase.Atoms

Atoms object

T : float

Temperature in *K*

coeffs : pandas.DataFrame

modeinfo : pandas.DataFrame

`panther.anharmonicity.`**`factsqrt`**(*m*, *n*)

Return a factorial like constant

Parameters **m** : int

Argument of the series

n : int

Length of the series

Notes

$$f(m,n) = \prod_{i=0}^{n-1} \sqrt{m-i}$$

`panther.anharmonicity.`**`get_anh_state_functions`**(*eigenvals*, *T*)

Calculate the internal energy `U` and entropy `S` for an anharmonic vibrational mode with eigenvalues `eigvals` at temperature `T` in kJ/mol

$$U = N_A \frac{\sum_{i=1}^{n} \epsilon_i \exp(\epsilon_i/k_B T)}{\sum_{i=1}^{n} \exp(\epsilon_i/k_B T)}$$

$$S = N_A k_B \log(\sum_{i=1}^{n} \exp(\epsilon_i/k_B T)) + \frac{N_A}{T} \frac{\sum_{i=1}^{n} \epsilon_i \exp(\epsilon_i/k_B T)}{\sum_{i=1}^{n} \exp(\epsilon_i/k_B T)}$$

Parameters **eigenvals** : numpy.array

Eigenvalues of the anharmonic 1D Hamiltonian in Joules

T : float

Temperature in *K*

Returns **(U, S)** : tuple of floats

Tuple with the internal energy and entropy in kJ/mol `panther.anharmonicity.`**`get_hamiltonian`**(*rank*, *freq*, *mass*, *coeffs*)

Compose the Hamiltonian matrix for the anharmonic oscillator with the potential described by the sixth order polynomial.

Parameters **rank** : int

Rank of the Hamiltonian matrix

freq : float

Fundamental frequency in hartrees

mass : float

Reduced mass of the mode

coeffs : array

A one dimensional array with polynomial coeffients

Notes

$$H_{ij} = \left\langle \Psi_i \middle| \hat{H} \middle| \Psi_j \right\rangle$$

where

$$\hat{H} = -\frac{\hbar^2}{2}\frac{\partial^2}{\partial \boldsymbol{Q}^2} + \sum_{\mu=0}^{6} c_\mu \boldsymbol{Q}^\mu$$

and Ψ_i are the standard harmonic oscillator functions.

`panther.anharmonicity.`**`harmonic_df`**(*modeinfo*, *T*)

Calculate per mode contributions to the thermodynamic functions in the harmonic approximation

Parameters **modeinfo** : pandas.DataFrame

T : float

Temperature in *K*

Returns **df** : pandas.DataFrame

`panther.anharmonicity.`**`merge_vibs`**(*anh6*, *anh4*, *harmonic*, *verbose=False*)

Form a DataFrame with the per mode thermochemical contributions from three separate dataframes with sixth order polynomial fitted potentia, fourth order fitted potential and harmonic frequencies.

Parameters **anh6** : pandas.DataFrame

anh4 : pandas.DataFrame

harmonic : pandas.DataFrame

Returns **df** : pandas.DataFrame

1.4.4 panther.displacements module

`panther.displacements.`**`calculate_displacements`**(*atoms*, *hessian*, *freqs*, *normal_modes*, *npoints=4*, *modes='all'*)

Calculate displacements in internal coordinates

Parameters **atoms** : ase.Atoms

Atoms object with the equilibrium structure

hessian : array_like

Hessian matrix in atomic units

freqs : array_like

Frequencies (square roots of the hessian eigenvalues) in atomic units

normal_modes : array_like

Normal modes in atomic units

npoints : int

Number of points to displace structure, the code will calculate `2*npoints` displacements since + and - directions are taken

modes : str or list/tuple of ints, default 'all'

Range of the modes for which the displacements will be calculated

Returns **images** : dict of dicts

A nested (ordred) dictionary with the structures with mode, point as keys, where point is a number from -4, -3, -2, -1, 1, 2, 3, 4

mi : pandas.DataFrame

DataFrame with per mode characteristics, displacements, masses and vibrational population analysis

`panther.displacements.`**`get_internals_and_bmatrix`**(*atoms*)

internals is a numpy record array with 'type' and 'value' records bmatrix is a numpy array n_int x n_cart

Parameters **atoms** : ase.Atoms

Atoms object

`panther.displacements.`**`get_modeinfo`**(*hessian*, *freqs*, *ndof*, *Bmatrix_inv*, *Dmatrix*, *mwevecs*, *npoints*, *internals*)

Compose a DataFrame with information about the vibrations, each mode corresponds to a separate row

`panther.displacements.`**`get_nvibdof`**(*atoms*, *proj_rotations*, *proj_translations*, *phase*, *include_constr=False*)

Calculate the number of vibrational degrees of freedom

Parameters **atoms** : ase.Atoms

proj_translations : bool

If `True` translational degrees of freedom will be projected from the hessian

proj_rotations : bool

If `True` rotational degrees of freedom will be projected from the hessian

include_constr : bool

If `True` the constraints will be included

Returns **nvibdof** : float

Number of vibrational degrees of freedom `panther.displacements.`**`vib_population`**(*hessian*, *freqs*, *Bmatrix_inv*, *Dmatrix*, *internals*, *mi*)

Calculate the vibrational population analysis

Parameters **hessian** : array_like

Hessian matrix

freqs : array_like

A vector of frequencies (square roots fof hessian eigenvalues)

Bmatrix_inv : array_like

Inverse of the B matrix

Dmatrix : array_like

D matrix

internals : array_like

Structured array with internal coordinates

mi : pandas.DataFrame

Modeinfo

Returns **mi** : pandas.DataFrame

Modeinfo DataFrame updated with columns with vibrational population analysis results

1.4.5 panther.io module

Module providing functions for reading the input and other related files

`panther.io.`**`get_symmetry_number`**(*pointgroup*)

Return the symmetry number for a given point group

See also:

C. J. Cramer, *Essentials of Computational Chemistry, Theories and Models*, 2nd Edition, p. 363

Parameters **pointgroup** : str

Symbol of the point group

`panther.io.`**`parse_arguments`**()

Parse the input/config file name from the command line, parse the config and return the parameters.

`panther.io.`**`print_mode_thermo`**(*df*, *info=False*)

After calculating all the anharmonic modes print the per mode themochemical functions

`panther.io.`**`print_modeinfo`**(*mi*, *output=None*)

Print the vibrational population data

Parameters **mi** : pandas.DataFrame

output : str

Name of the file to store the printout, if `None` stdout will be used

`panther.io.`**`read_bmatdat`**()

Read the `bmat.dat` file with internal coordiantes and the B matrix produced by the original `writeBmat` code

Returns **internals, Bmatrix** : tuple

Internal coordiantes and B matrix

`panther.io.`**`read_em_freq`**(*fname*)

Read the file `fname` with the frequencies, reduced masses and fitted fitted coefficients for the potential into a pandas DataFrame.

Parameters fname : str

Name of the file with PES

`panther.io.`**`read_pes`**(*fname*)

Parse the file with the potential energy surface (PES) into a dict of numpy arrays with mode numbers as keys

Parameters fname : str

Name of the file with PES

`panther.io.`**`read_poscars`**(*filename*)

Read POSCARs file with the displaced structures and return an OrderedDict with the Atoms objects

`panther.io.`**`read_vasp_hessian`**(*outcar='OUTCAR'*, *symmetrize=True*, *convert_to_au=True*, *dof_labels=False*)

Parse the hessian from the VASP `OUTCAR` file into a numpy array

Parameters outcar : str

Name of the VASP output, default is `OUTCAR`

symmetrize : bool

If `True` the hessian will be symmetrized

convert_to_au : bool

If `True` convert the hessian to atomic units, in the other case hessian is returned in [eV/Angstrom**2]

dof_labels : bool, default is False

If `True` a list of labels corresponding to the degrees of freedom will also be returned

Returns hessian : numpy.array

Hessian matrix

Notes

Note: By default VASP prints negative hessian so the elements should be multiplied by -1 to restore the original hessian, this is done by default, hessian in the XML file is **NOT** symmetrized by default

`panther.io.`**`read_vasp_hessian_xml`**(*xml='vasprun.xml'*, *convert_to_au=True*, *stripmass=True*)

Parse the hessian from the VASP `vasprun.xml` file into a numpy array

Parameters xml : str

Name of the VASP output, default is `vasprun.xml`

convert_to_au : bool

If `True` convert the hessian to atomic units, in the other case hessian is returned in [eV/Angstrom**2]

dof_labels : bool, default is False

If `True` a list of labels corresponding to the degrees of freedom will also be returned

stripmass : bool

If `True` use VASP default masses to transform hessian to non-mass-weighted form

Returns **hessian** : numpy.array

Hessian matrix

Notes

Note: By default VASP prints negative hessian so the elements should be multiplied by -1 to restore the original hessian, this is done by default, hessian in the XML file is symmetrized by default

`panther.io.`**`write_modes`**(*filename='POSCARs'*)

Convert a file with multiple geometries representing vibrational modes in `POSCAR`/`CONTCAR` format into trajectory files with modes.

1.4.6 panther.nmrelaxation module

class `panther.nmrelaxation.`**`NormalModeBFGS`**(*atoms*, *phase*, *hessian=None*, *hessian_update='BFGS'*, *logfile='-'*, *trajectory=None*, *restart=None*, *proj_translations=True*, *proj_rotations=True*, *master=None*, *verbose=False*)

Bases: `ase.optimize.optimize.Optimizer`, `object`

Normal mode optimizer with approximate hessian update

Parameters **atoms** : ase.Atoms

Atoms object with the structure to optimize

phase : str

Phase, should be either *gas* or *solid*

hessian : array_like (N, N)

Initial hessian matrix in eV/Angstrom^2

hessian_update : str

Name of the approximate formula to udpate hessian, one of: *BFGS*, *SR1*, *DFP*

proj_translations : bool

If `True` translational degrees of freedom will be projected from the hessian

proj_rotations : bool

If `True` rotational degrees of freedom will be projected from the hessian

logfile : str

Name log the log file

trajectory : str

Name of the trajectory file

log(*grad*, *grad_nm*, *step_nm*)
Print a line with convergence information

log_header()
Header for the log with convergence information

read()

run(*fmax=0.05*, *steps=100000000*)
Run structure optimization algorithm.

This method will return when the forces on all individual atoms are less than *fmax* or when the number of steps exceeds *steps*.

step(*grad*)
Calculate the step in cartesian coordinates based on the step in normal modes in the rational function approximation (RFO)

Args:

grad [array_like (N,)] Current gradient

update_hessian(*coords*, *grad*)
Perform hessian update

Parameters **coords** : array_like (N,)

Current coordinates as vector

grad : array_like (N,)

Current gradient

panther.nmrelaxation.**nmoptimize**(*atoms*, *hessian*, *calc*, *phase*, *proj_translations=True*, *proj_rotations=True*, *gtol=1e-05*, *verbose=False*, *hessian_update='BFGS'*, *steps=100000*)
Relax the strcture using normal mode displacements

Parameters **atoms** : ase.Atoms

Atoms object with the structure to optimize

hessian : array_like

Hessian matrix in eV/Angstrom^2

calc : ase.Calculator

ASE Calcualtor instance to be used to calculate forces

phase : str

Phase, 'solid' or 'gas'

gtol : float, default=1.0e-5

Energy gradient threshold

hessian_update : str

Approximate formula to update hessian, possible values are 'BFGS', 'SR1' and 'DFP'

steps : int

Maximal number of iteration to be performed

verbose : bool

If `True` additional debug information will be printed

Notes

Internally eV and Angstroms are used.

See also:

Bour, P., & Keiderling, T. A. (2002). Partial optimization of molecular geometry in normal coordinates and use as a tool for simulation of vibrational spectra. The Journal of Chemical Physics, 117(9), 4126. doi:10.1063/1.1498468

`panther.nmrelaxation.`**`update_hessian`**(*grad*, *grad_old*, *dx*, *hessian*, *update='BFGS'*)

Perform hessian update

Parameters **grad** : array_like (N,)

Current gradient

grad_old : array_like (N,)

Previous gradient

dx : array_like (N,)

Step vector x_n - x_(n-1)

hessian : array_like (N, N)

Hessian matrix

update : str

Name of the hessian update to perform, possible values are 'BFGS', 'SR1' and 'DFP'

Returns **hessian** : array_like

Update hessian matrix

1.4.7 panther.panther module

Python package for Anharmonic Thermochemistry

`panther.panther.`**`main`**()

The main Thermo program

`panther.panther.`**`temperature_range`**(*conditions*)

Calculate the temperature grid from the input values and return them as numpy array

Parameters **conditions** : dict

Variable for conditions read from the input/config

Returns **temps** : numpy.array

Array with the temperature grid

1.4.8 panther.pes module

`panther.pes.`**`calculate_energies`**(*images*, *calc*, *modes='all'*)

Given a set of images as a nested OrderedDict of Atoms objects and a calculator, calculate the energy for each displaced structure

Parameters **images** : OrderedDict

A nested OrderedDict of displaced Atoms objects

calc : calculator instance

ASE calculator

modes : str or list

Mode for which the PES will be calculated

Returns **energies** : pandas.DataFrame

DataFrame with the energies per displacement

`panther.pes.`**`differentiate`**(*displacements*, *energies*, *order=2*)

Calculate numerical detivatives using the central difference formula

Parameters **displacements** : array_like

energies : DataFrame

order : int

Order of the derivative

Notes

Central difference coefficients taken from *[R11]*

`panther.pes.`**`expandrange`**(*modestr*)

Convert a comma separated string of indices and dash separated ranges into a list of integer indices

Parameters **modestr** : str

Returns **indices** : list of ints

Examples

```
>>> from panther.pes import expandrange
>>> s = "2,3,5-10,20,25-30"
>>> expandrange(s)
[2, 3, 5, 6, 7, 8, 9, 10, 20, 25, 26, 27, 28, 29, 30]
```

`panther.pes.`**`fit_potentials`**(*modeinfo*, *energies*)

Fit the potentials with 6th and 4th order polynomials

Parameters **modeinfo** : pandas.DataFrame

DataFrame with per mode characteristics, displacements, masses and a flag to mark it a mode is a stretching mode or not

energies : pd.DataFrame

Energies per displacement

Returns **out** : (coeffs6o, coeffs4o)

DataFrames with 6th and 4th polynomial coefficients fitted to the potential

`panther.pes.`**`harmonic_potential`**(*x*, *freq*, *mu*)

Calculate the harmonic potential

Parameters **x** : float of numpy.array

Coordinate

mu : float

Reduced mass

freq : float

Frequency in cm^-1

1.4.9 panther.plotting module

Functions for plotting the each mode and PES fits

`panther.plotting.`**`plotmode`**(*mode*, *energies*, *mi*, *c6o*, *c4o*, *output=None*)

Plot a given mode

Parameters **mode** : int

Mode number (indexed from 0)

energies : pandas.DataFrame

mi : pandas.DataFrame

Modeinfo

c6o : pandas.DataFrame

c4o : pandas.DataFrame

output : str

name o file to store the plot

`panther.plotting.`**`plotmode_legacy`**(*mode*, *pes*, *coeff6*, *coeff4*, *output=None*)

Plot a given mode using legacy files

1.4.10 panther.vibrations module

`panther.vibrations.`**`get_levicivita`**()

Get the Levi_civita symemtric tensor

`panther.vibrations.`**`harmonic_vibrational_analysis`**(*hessian*, *atoms*, *proj_translations=True*, *proj_rotations=False*, *ascomplex=True*, *massau=True*)

Given a force constant matrix (hessian) perform the harmonic vibrational analysis, by calculating the eigenvalues and eigenvectors of the mass weighted hessian. Additionally projection of the translational and rotational degrees of freedom can be performed by specifying `proj_translations` and `proj_rotations` argsuments.

Parameters **hessian** : array_like

Force constant (Hessian) matrix in atomic units, should be square and symmetric

atoms : Atoms

ASE atoms object

proj_translations : bool

If `True` translational degrees of freedom will be projected from the hessian

proj_rotations : bool

If `True` rotational degrees of freedom will be projected from the hessian

massau : bool

If `True` atomic units of mass will be used

ascomplex : bool

If there are complex eigenvalues return the array as complex type otherwise make the complex values negative and return array of reals

Returns out : (w, v)

Tuple of numpy arrays with hessian square roots of the eigevalues (frequencies) and eiegenvectors in atomic units, both sorted in descending order of eigenvalues

`panther.vibrations.`**`project`**(*atoms*, *hessian*, *ndof*, *proj_translations=True*, *proj_rotations=False*, *verbose=False*)

Project out the translational and/or rotational degrees of freedom from the hessian.

Parameters atoms : ase.Atoms

Atoms object

ndof : int

Number of degrees of freedom

hessian : array_like

Hessian/force constant matrix

proj_translations : bool

If `True` translational degrees of freedom will be projected from the hessian

proj_rotations : bool

If `True` rotational degrees of freedom will be projected from the hessian

Returns proj_hessian : array_like

Hessian matrix with translational and/or rotational degrees of freedom projected out

`panther.vibrations.`**`project_massweighted`**(*args*, *atoms*, *ndof*, *hessian*, *verbose=False*)

Project translational and or rotatioanl dgrees of freedom from mass weighted hessian

CHAPTER 2

Citing

If you use *panther* in a scientific publication, please cite the software as

- 12. (a) Mentel, **PANTHER - Python Package for Anharmonic Thermochemistry**, 2016–, https://bitbucket.org/lukaszmentel/panther

An example of a BibTeX entry can look like this:

```
@misc{panther2016,
author = {Mentel, Lukasz Michal},
title = {{PANTHER}: Python Package for Anharmonic Thermochemistry},
year = {2016--},
url = {https://bitbucket.org/lukaszmentel/panther},
}
```

CHAPTER 3

Funding

This project is supported by the RCN (The Research Council of Norway) project number 239193.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Bibliography

[R11] Fornberg, B. (1988). Generation of finite difference formulas on arbitrarily spaced grids. Mathematics of Computation, 51(184), 699-699. doi:10.1090/S0025-5718-1988-0935077-0

Python Module Index

p

Index

A

C

D

E

F

G

H

L

M

N

P

R

S

T

U

V

W