calphy

Sarath Menon

Jul 21, 2021

CONTENTS

CHAPTER

ONE

CALPHY

Python library and command line tool for calculation of free energies.

1.1 Dependencies

- lammps 2021.05.27
- mendeleev 0.7.0 `pip install mendeleev`
- pylammpsmpi 0.0.8 `pip install pylammpsmpi`
- pyscal 2.10.14 `pip install git+https://github.com/srmnitc/pyscal`
- pyyaml 5.4.1 `pip install pyyaml`
- scipy 1.7.0 `pip install scipy`
- tqdm 4.61.2 `pip install tqdm`

1.1.1 Optional

- matplotlib 3.4.2 `pip install matplotlib`
- pytest 6.2.4 `pip install pytest`

1.2 Installation

> **NOTE**: If you are planning to use a custom version of LAMMPS, read the 'Notes on the LAMMPS installation' section first.

It is **strongly** recommended to install and use `calphy` within a conda environment. To see how you can install conda see here.

Once a conda distribution is available, the following steps will help set up an environment to use `calphy`. First step is to clone the repository.

```
git clone https://git.noc.ruhr-uni-bochum.de/atomicclusterexpansion/calphy.git
```

After cloning, an environment can be created from the included file-

```
cd calphy
conda env create -f environment.yml
```

This will install the necessary packages and create an environment called calphy. It can be activated by,

```
conda activate calphy
```

then, install `calphy` using,

```
python setup.py install --user
```

The environment is now set up to run calphy.

1.3 Notes on the LAMMPS installation

NOTE: 08 July 2021: The upcoming release of LAMMPS will significantly change the compilation procedure and render the below information outdated.

- The above commands will install LAMMPS from the conda-forge channel. With this version of LAMMPS, only `eam` pair style is supported!
- For other pair styles such as Stillinger-Weber, MEAM, ADP, and SNAP, a custom version of LAMMPS is available. Please contact here.
- For using with pair style PACE, the LAMMPS distribution has to be compiled manually:

```
wget https://github.com/lammps/lammps/archive/refs/tags/stable_29Oct2020.tar.gz
tar -xvf stable_29Oct2020.tar.gz
cd lammps-stable_29Oct2020
mkdir build_lib
cd build_lib
cmake -D BUILD_LIB=ON -D BUILD_SHARED_LIBS=ON -D BUILD_MPI=ON -D PKG_MANYBODY=ON -D PKG_
↪USER-MISC=ON -D PKG_USER-PACE=ON ../cmake
make # -j${NUM_CPUS}
cp liblammps${SHLIB_EXT}* ../src
```

If the commands are run within a conda environment and errors during compilation are observed, try installing the following packages:

```
conda install -c conda-forge cmake gcc_linux-64 gxx_linux-64 gfortran_linux-64
```

Now the python wrapper can be installed:

```
cd ../src
make install-python
```

In the case of a conda environment, the following commands can be used to copy the compiled libraries to an accessible path:

```
mkdir -p $CONDA_PREFIX/include/lammps
cp library.h $CONDA_PREFIX/include/lammps
cp liblammps${SHLIB_EXT}* $CONDA_PREFIX/lib/
```

The combined libraries **should be available** on the system or the environment paths. Please see here for more details.

Once LAMMPS is compiled and the libraries are available in an accessible location, the following commands can be used within python to test the installation:

```
from lammps import lammps
lmp = lammps()
```

CHAPTER

TWO

DOCUMENTATION

2.1 Documentation

2.1.1 `pytint` input file

The inputfile is `yaml` formatted. In this section the possible keys in the inputfile is discussed. The input file consists of two main keys, and four separate blocks. For a sample of the inputfile see the end of this document. The two main keys are as shown below-

main keys

- **`element`** [Chemical symbol(s) of the element(s) in the simulation.] *type*: string/list of strings *example*:

```
element: 'Cu'
element: ['Cu', 'Zr']
```

- **`mass`** [Mass of the element(s) in the simulation. It should follow the same order as that of `element`.] *type*: float/list of floats *example*:

```
mass: 63.546
mass: [63.546, 91.224]
```

`calculation` block

Other than these two main keys, all other options are specified in blocks. The first block is the `calculations` block. This block can list all the calculations that `pytint` should perform and it can have more than one entry. A sample calculation block is shown below.

```
calculations:
- mode: ts
  temperature: [1300, 1400]
  pressure: [0]
  lattice: [FCC]
  repeat: [2, 2, 2]
  state: solid
  nsims: 1
```

The various keys are-

- **mode** [Calculation mode. Either `fe` for free energy calculations or `ts` for temperature sweep.] *type*: string, `fe` or `ts` *example*:

```
mode: fe
mode: ts
```

- **temperature** [Temperatures for the simulation in Kelvin.] *type*: list of floats *example*:

```
   temperature: [1200, 1300]

The way temperature is used ``pytint`` depends on the selected mode of calculation.␣
↪If the ``mode`` is ``fe``\ , a free energy calculation is launched for each␣
↪temperature on the list. If the mode is ``ts``\ , a temperature sweep is carried␣
↪out. In that case, only two values of temperature should be specified.
```

- **pressure** [Pressure for the simulation in bars.] *type*: list of floats *example*:

```
   pressure: [0, 10000]

For each pressure specified in the list, one calculation will be started. For␣
↪example, in the following combination-
```

```
   mode: fe
   temperature: [1000, 1200]
   pressure: [0, 10000]

A total of 2 temperatures * 2 pressures, 4 calculations will be started. If the␣
↪mode is ``ts`` for the same configuration above, 2 calculations will be started.
```

- **lattice** [Lattice to be used for the calculations.] *type*: string of list of strings *example*:

```
   lattice: FCC
   lattice: [FCC, LQD]
   lattice: [FCC, conf.data]

The ``lattice`` option can use either LAMMPS for creation of input structure or use␣
↪an input file in the LAMMPS data format. To use LAMMPS to create the structure,␣
↪the keyword specified should be from the following: ``BCC``\ , ``FCC``\ , ``HCP``\
↪ , ``DIA``\ , ``SC`` and ``LQD``. LAMMPS lattice creation can **only be used for␣
↪single species**. If ``LQD`` is specified, a solid structure will be first␣
↪created and melted within the MD run. Alternatively, a LAMMPS data file can be␣
↪specified which contains the configuration.
```

- **state** [The protocol to be used for the calculation.] *type*: string of list of strings *example*:

```
   state: solid
   state: [solid, liquid]

The ``state`` input is closely related to the ``lattice`` command. It should be of␣
↪the same length as the ``lattice`` input. For each of the lattice specified,␣
↪``state`` command specifies which reference state to use.
```

- **lattice_constant** [lattice constant for input structures] *type*: list of floats *example*:

```
   lattice_constant: 3.68
   lattice_constant: [3.68, 5.43]

Lattice constant values to be used for initial structure creation. Only required if␣
↪the structure is created through LAMMPS. If not specified, the experimental␣
↪lattice constant will be used.
```

- **iso** [Specify if the barostat is isotropic or anisotropic.] *type*: list of bools *example*:

```
   iso: True
   iso: [True, False]

Specify whether the barostat will control the pressure isotropically or␣
↪anisotropically.
```

- **repeat** [The number of unit cells to be replicated in each direction.] *type*: list of ints of length 3 *example*:

```
   repeat: [3,3,3]

``repeat`` command specifies the number of unit cells required in each x, y and z␣
↪directions. This is only used if ``lattice`` command uses one of the LAMMPS␣
↪structure keywords.
```

- **nsims** [The number of backward and forward interactions to be carried out.] *type*: int *example*:

```
nsims: 3
```

md block

MD block consists of the various options required for the MD runs. An example block is given below and the keys are discussed.

```
md:
  pair_style: eam/alloy
  pair_coeff: "* * Cu_EAM/Cu01.eam.alloy Cu"
  timestep: 0.001
  tdamp: 0.1
  pdamp: 0.1
  te: 10000
  ts: 15000
```

- **pair_style** [The pair style command for LAMMPS.] *type*: string *example*:

```
pair_style: eam/alloy
pair_style: eam/fs
pair_style: pace
```

- **pair_coeff** [The pair coeff command for LAMMPS.] *type*: string *example*:

```
pair_coeff: "* * Cu_EAM/Cu01.eam.alloy Cu"
pair_coeff: "* * CuZr_EAM/CuZr.eam.fs Cu Zr"
```

- **timestep** [The timestep for MD in picoseconds.] *type*: float *example*:

```
timestep: 0.001
```

- **tdamp** [The thermostat damping for MD in time units.] *type*: float *example*:

```
tdamp: 0.1
```

- **pdamp** [The pressure damping for MD in time units.] *type*: float *example*:

```
pdamp: 0.1
```

- **te** [The number of time steps for equilibrating the system.] *type*: int *example*:

```
te: 10000
```

- **ts** [The number of time steps for switching between the system of interest and reference system.] *type*: int *example*:

```
ts: 10000
```

queue block

This block consints of the options for specifying the scheduler for carrying out the calculations. An example block is given below-

```
queue:
  scheduler: local
  cores: 2
  jobname: ti
  walltime: "23:50:00"
  queuename: shorttime
  memory: 3GB
  modules:
    - anaconda/4
  commands:
    - conda activate env
```

- **scheduler** [The scheduler to be used for the job. Can be `local`, `slurm` or `sge`.] *type*: string *example*:

```
   scheduler: slurm

The code has been tested only on local and slurm.
```

- **cores** [The number of cores to be used for the job.] *type*: int *example*:

```
cores: 4
```

- **jobname** [Name of the job.] *type*: string *example*:

```
   jobname: cu

Not used for ``local``.
```

- **walltime** [Walltime for the job.] *type*: string *example*:

```
   walltime: "23:50:00"

Not used for ``local``.
```

- **queuename** [Name of the queue.] *type*: string *example*:

```
   queuename: "shorttime"

Not used for ``local``.
```

- **memory** [memory to be used per core.] *type*: string *example*:

```
   memory: 3GB

Not used for ``local``.
```

- **commands** [Command that will be run **before** the actual calculations are carried out.] *type*: list of strings *example*:

```
   command:
     - source .bashrc
     - conda activate ace
     - module load lammps

This section can be used to specify commands that need to be run before the actual␣
↪calculation. If the calculations are run within a conda environment, the activate␣
↪command for conda should be specified here. If additional modules need to be␣
↪loaded, that can also be specified here.
```

conv block

This block helps to tune the internal convergence parameters that `pytint` uses. Generally, tuning these parameters are not required.

```
conv:
   k_tol: 0.01
   solid_frac: 0.7
   liquid_frac: 0.05
   p_tol: 0.5
```

- **k_tol** [tolerance for the convergence of spring constant calculation.] *type*: float *example*:

```
ktol: 0.01
```

- **solid_frac** [The minimum amount of solid particles that should be there in solid.] *type*: float *example*:

```
solid_frac: 0.7
```

- **liquid_frac** [Maximum fraction of solid atoms allowed in liquid after melting.] *type*: float *example*:

```
liquid_frac: 0.05
```

- **p_tol** [tolerance for the convergence of pressure.] *type*: float *example*:

```
p_tol: 0.5
```

2.1.2 Running `calphy`

After the calculations are specified in the input file, `calphy` can be run by:

```
calphy -i inputfilename
```

CHAPTER

THREE

EXAMPLES

3.1 Examples

3.1.1 Example 01: Calculation of free energy

In this example, a simple calculation of the Helmholtz free energy $F(NVT)$ is illustrated. We use a Fe BCC structure at 100K.

The EAM potential that will be used: Meyer, R, and P Entel. "Martensite-austenite transition and phonon dispersion curves of Fe1xNix studied by molecular-dynamics simulations." Phys. Rev. B 57, 5140.

The reference data is from: Freitas, Rodrigo, Mark Asta, and Maurice de Koning. "Nonequilibrium Free-Energy Calculation of Solids Using LAMMPS." Computational Materials Science 112 (February 2016): 333–41.

Solid free energy

The input file for calculation of solid BCC Fe at 100 K is provided in the folder. A detailed description of the input file is available here. The calculation can be started from the terminal using:

```
calphy -i input.yaml
```

This should give a message:

```
Total number of 1 calculations found
```

`calphy` is running in the background executing the calculation. A new folder called `fe-BCC-100-0`, and files called `fe-BCC-100-0.sub.err` and `fe-BCC-100-0.sub.out` will also be created.

After the calculation is over, the results are available in the `report.yaml` file. The file is shown below:

```
average:
  spring_constant: '3.32'
  vol/atom: 11.994539429692749
input:
  concentration: '1'
  element: Fe
  lattice: bcc
  pressure: 0.0
  temperature: 100
results:
  error: 0.0
```

(continues on next page)

(continued from previous page)

```
  free_energy: -4.263568357143783
  pv: 0.0
  reference_system: 0.015029873513789175
  work: -4.278598230657573
```

The calculated free energy is -4.2636 eV/atom, the value reported in the publication is $-4.2631147(1)$ eV/atom. The calculated value can be further improved by increasing the system size, and by increasing the switching time.

3.1.2 Example 02: Phase transformation in Fe

In this example, we will make use of the temperature sweep algorithm in `calphy` to calculate the transformation temperature for BCC to FCC transition in Fe.

The EAM potential that will be used: Meyer, R, and P Entel. "Martensite-austenite transition and phonon dispersion curves of Fe1xNix studied by molecular-dynamics simulations." Phys. Rev. B 57, 5140.

The reference data is from: Freitas, Rodrigo, Mark Asta, and Maurice de Koning. "Nonequilibrium Free-Energy Calculation of Solids Using LAMMPS." Computational Materials Science 112 (February 2016): 333–41.

The input file is provided in the folder. The calculation can be started from the terminal using:

```
calphy -i input.yaml
```

In the input file, the `calculations` block is as shown below:

```
- mode: ts
  temperature: [100, 1400]
  pressure: [0]
  lattice: [BCC]
  repeat: [5, 5, 5]
  state: [solid]
  nsims: 1
- mode: ts
  temperature: [100, 1400]
  pressure: [0]
  lattice: [FCC]
  repeat: [5, 5, 5]
  state: [solid]
  nsims: 1
  lattice_constant: [6.00]
```

The mode is listed as `ts`, which stands for temperature sweep. The sweep starts from the first value in the `temperature` option, which is 100 K. The free energy is integrated until 1400 K, which is the second value listed. Furthermore, there are also two calculation blocks. You can see that the `lattice` mentioned is different; one set is for BCC structure, while the other is FCC.

Once the calculation is over, there will a file called `temperature_sweep.dat` in each of the folders. This file indicates the variation of free energy with the temperature. We can read in the files and calculate the transition temperature as follows:

```
import numpy as np

bt, bfe, bferr = np.loadtxt("ts-BCC-100-0/temperature_sweep.dat", unpack=True)
ft, ffe, fferr = np.loadtxt("ts-FCC-100-0/temperature_sweep.dat", unpack=True)
```

(continues on next page)

(continued from previous page)

```
args = np.argsort(np.abs(bfe-ffe))
print(bt[args[0]], "K")

plt.plot(bt, bfe, color="#E53935", label="bcc")
plt.plot(ft, ffe, color="#0097A7", label="fcc")
plt.xlabel("Temperature (K)", fontsize=12)
plt.ylabel("F (ev/atom)", fontsize=12)
plt.legend()
```

A jupyter notebook that calculates the transition temperature is shown here.

3.1.3 Example 03: Calculation of Cu melting temperature

In this example, the melting temperature for Cu is calculated using a combination of free energy calculation and temperature sweep.

The EAM potential we will use is : Mishin, Y., M. J. Mehl, D. A. Papaconstantopoulos, A. F. Voter, and J. D. Kress. "Structural Stability and Lattice Defects in Copper: Ab Initio , Tight-Binding, and Embedded-Atom Calculations." Physical Review B 63, no. 22 (May 21, 2001): 224106.

The calculation block gives the input conditions at which the calculation is carried out. First of all, the `mode` is `ts`, a temperature sweep over the temperature range given in the `temperature` keyword will be done. `FCC` and `LQD` lattice are chosen, the former for `solid` and the latter for `liquid`. The potential file is specified in `pair_coeff` command in the `md` block.

The calculation can be run by,

```
calphy -i input.yaml
```

Once submitted, it should give a message `Total number of 2 calculations found`. It will also create a set of folders with the names `mode-lattice-temperature-pressure`. In this case, there will be `ts-FCC-1200-0` and `ts-LQD-1200-0`. If there are any errors in the calculation, it will be recoreded in `ts-FCC-1200-0.sub.err` and `ts-LQD-1200-0.sub.err`. Once the calculation starts, a log file called `tint.log` will be created in the aforementioned folders. For example, the `tint.log` file in `ts-FCC-1200-0` is shown below:

```
2021-07-08 15:12:02,873 calphy.helpers INFO     At count 1 mean pressure is -5.337803␣
↪with 12.625936 vol/atom
2021-07-08 15:12:09,592 calphy.helpers INFO     At count 2 mean pressure is -5.977502␣
↪with 12.625639 vol/atom
2021-07-08 15:12:15,790 calphy.helpers INFO     At count 3 mean pressure is -6.557647␣
↪with 12.623906 vol/atom
2021-07-08 15:12:22,059 calphy.helpers INFO     At count 4 mean pressure is -2.755662␣
↪with 12.624017 vol/atom
2021-07-08 15:12:28,157 calphy.helpers INFO     At count 5 mean pressure is -1.306381␣
↪with 12.625061 vol/atom
2021-07-08 15:12:34,405 calphy.helpers INFO     At count 6 mean pressure is 0.718567␣
↪with 12.625150 vol/atom
2021-07-08 15:12:40,595 calphy.helpers INFO     At count 7 mean pressure is 2.185729␣
↪with 12.625295 vol/atom
2021-07-08 15:12:47,020 calphy.helpers INFO     At count 8 mean pressure is 2.162312␣
↪with 12.625414 vol/atom
```

(continues on next page)

(continued from previous page)

```
2021-07-08 15:12:53,323 calphy.helpers INFO     At count 9 mean pressure is 1.137092␣
↪with 12.624953 vol/atom
2021-07-08 15:12:59,692 calphy.helpers INFO     At count 10 mean pressure is 0.377612␣
↪with 12.625385 vol/atom
2021-07-08 15:12:59,693 calphy.helpers INFO     finalized vol/atom 12.625385 at pressure␣
↪0.377612
2021-07-08 15:12:59,693 calphy.helpers INFO     Avg box dimensions x: 18.483000, y: 18.
↪483000, z:18.483000
2021-07-08 15:13:05,878 calphy.helpers INFO     At count 1 mean k is 1.413201 std is 0.
↪072135
2021-07-08 15:13:12,088 calphy.helpers INFO     At count 2 mean k is 1.398951 std is 0.
↪065801
2021-07-08 15:13:12,090 calphy.helpers INFO     finalized sprint constants
2021-07-08 15:13:12,090 calphy.helpers INFO     [1.4]
```

The file gives some information about the preparation stage. It can be seen that at loop 10, the pressure is converged and very close to the 0 value we specified in the input file. After the pressure is converged, box dimensions are fixed, and the spring constants for the Einstein crystal are calculated.

The `ts` mode consists of two stages, the first step is the calculation of free energy at 1200 K, followed by a sweep until 1400 K. The results of the free energy calculation is recorded in `report.yaml` file. The file is shown below:

```
average:
  spring_constant: '1.4'
  vol/atom: 12.625384980886036
input:
  concentration: '1'
  element: Cu
  lattice: fcc
  pressure: 0.0
  temperature: 1200
results:
  error: 0.0
  free_energy: -4.071606995367944
  pv: 0.0
  reference_system: -0.7401785806445611
  work: -3.331428414723382
```

In the file, the average and input quantities are recorded. The more interesting block is the `results` block. Here the calculated free energy value in eV/atom is given in the `free_energy` key. The free energy of the reference system is given in `reference_system` and the work done in switching is under `work`. The `error` key gives the error in the calculation. In this case, its 0 as we ran only a single loop (see `nsims`). The `report.yaml` file for liquid looks somewhat similar.

```
average:
  density: 0.075597559457131
  vol/atom: 13.227983329718013
input:
  concentration: '1'
  element: Cu
  lattice: fcc
  pressure: 0.0
  temperature: 1200
```

(continues on next page)

(continued from previous page)

```
results:
  error: 0.0
  free_energy: -4.058226884054426
  pv: 0.0
  reference_system: 0.6852066332000204
  work: 4.743433517254447
```

The main difference here is that under the `average` block, the `density` is reported instead of `spring_constant`.

The variation of the free energy within the temperature range is given in the `temperature_sweep.dat` files instead of each of the folders. The file contains three columns, temperature, free energy and the error in free energy. The files are read in and plotted below.

```
import numpy as np
import matplotlib.pyplot as plt

st, sfe, sferr = np.loadtxt("ts-FCC-1200-0/temperature_sweep.dat", unpack=True)
lt, lfe, lferr = np.loadtxt("ts-LQD-1200-0/temperature_sweep.dat", unpack=True)

plt.plot(st, sfe, color="#E53935", label="solid")
plt.plot(lt, lfe, color="#0097A7", label="liquid")
plt.xlabel("Temperature (K)", fontsize=12)
plt.ylabel("F (ev/atom)", fontsize=12)
plt.legend()
```

From the plot, at temperatures below 1300 K, solid is the more stable structure with lower free energy. Around 1340 K, liquid becomes the more stable structure. We can find the temperature at which the free energy of both phases are equal, which is the melting temperature.

```
args = np.argsort(np.abs(sfe-lfe))
print(st[args[0]], "K")
```

3.1.4 Example 04: Pressure-temperature phase diagram of Cu

In this example, the pressure-temperature phase diagram of Cu will be calculated.

The EAM potential we will use is : Mishin, Y., M. J. Mehl, D. A. Papaconstantopoulos, A. F. Voter, and J. D. Kress. "Structural Stability and Lattice Defects in Copper: Ab Initio , Tight-Binding, and Embedded-Atom Calculations." Physical Review B 63, no. 22 (May 21, 2001): 224106.

The input file is provided in the folder. The calculation is very similar to the calculation of melting temperature. However, we calculate the melting temperature for various pressures to arrive at the phase-diagram.

There are five input files in the folder, from `input1.yaml` to `input5.yaml`. Each file contains the calculations for a single pressure. You can also add all of the calculations in a single file under the `calculations` block. It is split here into five files for the easiness of running the calculations on relatively small machines.

The calculation can be run by:

```
calphy -i input1.yaml
```

and so on until `input5.yaml`. After the calculations are over, we can read in the results and compare it.

```
import numpy as np
import matplotlib.pyplot as plt
```

The starting temperatures and pressures for our calculations:

```
temp = [1600, 2700, 3700, 4600]
press = [100000, 500000, 1000000, 1500000]
```

Now a small loop which goes over each folder and calculates the melting temperature value:

```
tms = []

for t, p in zip(temp, press):
    sfile =  "ts-FCC-%d-%d/temperature_sweep.dat"%(t, p)
    lfile =  "ts-LQD-%d-%d/temperature_sweep.dat"%(t, p)
    t, f, fe = np.loadtxt(sfile, unpack=True)
    t, l, fe = np.loadtxt(lfile, unpack=True)
    args = np.argsort(np.abs(f-l))

    tms.append(t[args[0]])
```

To compare our results, we will use a Simon equation, given by,

$$T_m(P) = T_{m0}(P/a + 1)^b$$

We will use reported values for parameters T_{m0}, a and b from two different publications:

- Wang, Shuaichuang, Gongmu Zhang, Haifeng Liu, and Haifeng Song. "Modified Z Method to Calculate Melting Curve by Molecular Dynamics." The Journal of Chemical Physics 138, no. 13 (April 7, 2013): 134101.

```
def get_tm(press):
    tm = 1315*(press/15.84 + 1)**0.543
    return tm
```

- An, Qi, Sheng-Nian Luo, Li-Bo Han, Lianqing Zheng, and Oliver Tschauner. "Melting of Cu under Hydrostatic and Shock Wave Loading to High Pressures." Journal of Physics: Condensed Matter 20, no. 9 (March 5, 2008): 095220.

```
def get_tm2(press):
    tm = 1325*(press/15.37 + 1)**0.53
    return tm
```

An array for pressures over which the two equations will be fit, and values of the two expressions are calculated.

```
pfit = np.arange(0, 151, 1)
tma = get_tm(pfit)
tmb = get_tm2(pfit)
```

```
plt.plot(pfit, tma, ls="dashed", label="Wang et al.", color="#03A9F4")
plt.plot(pfit, tmb, ls="dashed", label="An et. al.", color="#E65100")
plt.plot(np.array(press)/10000, tms, 'o', color="#1A237E")
plt.xlabel("Pressure (GPa)", fontsize=12)
plt.ylabel(r"$T_m$ (K)", fontsize=12)
plt.legend()
```

The analysis code can be found in this notebook.

CHAPTER

FOUR

API REFERENCE

4.1 calphy package

4.1.1 Submodules

4.1.2 calphy.alchemy module

4.1.3 calphy.helpers module

4.1.4 calphy.input module

4.1.5 calphy.integrators module

4.1.6 calphy.kernel module

4.1.7 calphy.lattice module

4.1.8 calphy.liquid module

4.1.9 calphy.queuekernel module

4.1.10 calphy.scheduler module

4.1.11 calphy.solid module

4.1.12 calphy.splines module

4.1.13 Module contents

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search