calphy

Sarath Menon

Sep 15, 2022

CONTENTS

`calphy`(pronounced *cal-phee*) is a Python library and command line tool for calculation of free energies. It uses LAMMPS as the molecular dynamics driver to enable calculation of free energies using thermodynamic integration in a completely automated and efficient manner. The various methods that calphy can perform are:

- $F(V_i, T_i)$ and $G(P_i, T_i)$ for both solid and liquid phases at the given thermodynamic conditions using non-equilibrium Hamiltonian interpolation.
- $F(T_i \rightarrow T_f)$ and $G(T_i \rightarrow T_f)$, temperature dependence of Gibbs and Helmholtz free energies using the reversible scaling approach.
- Calculation of solid-solid or solid-liquid phase transition temperatures.
- Calculation of coexistence lines using dynamic Clausius-Clapeyron integration.
- Calculation of specific heat $c_P(T)$ as a function of temperature.
- Calculation of $F(x, T)$ and $G(x, T)$ for multicomponent systems using alchemical transformations.
- Upsampling calculations.

Calphy works with all interatomic potentials implemented in LAMMPS and can obtain reliable results with low error bars in as low as 50 ps of simulation time with less than 3000 atoms. More information about the methods in calphy can be found in the associated publication.

CHAPTER

ONE

DOCUMENTATION

1.1 Installation

1.1.1 Supported operating systems

`calphy` can be installed on Linux and Mac OS based systems. On Windows systems, it is recommended to use Windows subsystem for Linux.

1.1.2 Installation using Conda

calphy can be installed directly using Conda from the conda-forge channel by the following statement:

```
conda install -c conda-forge calphy
```

1.1.3 Installation from the repository

calphy can be built from the repository by-

```
git clone https://github.com/ICAMS/calphy.git
cd calphy
python setup.py install --user
```

Please note that the list of dependencies mentioned below has to installed manually by the user.

1.1.4 Using a conda environment

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://github.com/ICAMS/calphy.git
```

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

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

Note that the conda-forge distribution of LAMMPS will be automatically installed. Alternatively, you can use an existing version of LAMMPS (compiled as library). If a LAMMPS distribution need not be installed, use the `calphy/environment-nolammps.yml` file instead to create the environment. 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
```

The environment is now set up to run calphy.

1.1.5 Dependencies

- lammps `conda install -c conda-forge lammps`
- 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`

Optional

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

1.1.6 About LAMMPS for `calphy`

calphy uses LAMMPS as the driver for molecular dynamics simulations. For calphy to work, LAMMPS needs to be compiled as a library along with the Python interface. The easiest way to do this is to install LAMMPS through the conda-forge channel using:

```
conda install -c conda-forge lammps
```

Alternatively, when interatomic potentials with special compilation needs are to be used, LAMMPS (Stable release 29 Sept 2021 and above) can be compiled manually using the following set of instructions.

In order to help with installing all the prerequisites, an environment file which does not include the LAMMPS distribution is also provided. **This is only required if you want use a conda environment.** This environment can be installed using:

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

Activate the environment using:

```
conda activate calphy2
```

Obtain the stable version from here and extract the archive. From the extracted archive, the following steps, used in the conda-forge recipe can be run:

```
mkdir build_lib
cd build_lib
cmake -D BUILD_LIB=ON -D BUILD_SHARED_LIBS=ON -D BUILD_MPI=ON -D PKG_MPIIO=ON -D LAMMPS_
↪EXCEPTIONS=yes -D PKG_MANYBODY=ON -D PKG_MISC=ON -D PKG_MISC=ON -D PKG_EXTRA-
↪COMPUTE=ON -D PKG_EXTRA-DUMP=ON -D PKG_EXTRA-FIX=ON -D PKG_EXTRA-PAIR=ON ../cmake
make
cp liblammps${SHLIB_EXT}* ../src
cd ../src
make install-python
mkdir -p $PREFIX/include/lammps
cp library.h $PREFIX/include/lammps
cp liblammps${SHLIB_EXT}* "${PREFIX}"/lib/
cd ..
```

(**Optional**) The above commands only builds the MANYBODY package. To use some of the other potentials, the following commands could be added to the `cmake` call.

- `-D PKG_ML-PACE=ON` for performant Atomic Cluster Expansion potential (from October 2021 version).
- `-D PKG_ML-SNAP=ON`for SNAP potential.
- `-D PKG_MEAM=ON` for MEAM potential.
- `-D PKG_KIM=ON` for KIM support.

Install the python wrapper:

```
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 (sometimes `PREFIX` needs to be used instead of `CONDA_PREFIX`):

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

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:

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

Now, we install pylammpsmpi using,

```
pip install pylammpsmpi
```

And finally calphy:

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

1.2 Documentation

1.2.1 calphy 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 table below gives a quick overview of all available keywords in calphy.

main keys

element

type: string/list of strings *default*: None *example*:

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

Chemical symbol(s) of the element(s) in the simulation.

mass

type: float/list of floats *default*: 1.0 *example*:

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

Mass of the element(s) in the simulation. It should follow the same order as that of element, and should be of the same length.

calculations 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 calphy 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]
  reference_phase: solid
  n_iterations: 1
```

The various keys are-

`mode`

type: string, `fe` or `ts` or `mts` or `alchemy` or `melting_temperature` or `tscale` or `pscale` *default*: None *example*:

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

Calculation mode. A small description of the different modes are given below.

- `fe` performs a direct free energy calculation
- `ts` performs a direct free energy calculation followed by reversible scaling to find temperature dependence.
- `mts` performs only reversible scaling part and can be used for dynamic Clausius-Clapeyron integration.
- `alchemy` is used for switching between two different interatomic potentials, or for integration over concentration.
- `melting_temperature` can be used for automated calculation of melting temperature.
- `tscale` is used for similar purpose as `ts`, but scales the temperature directly.
- `pscale` calculates the free energy as a function of the pressure.

`temperature`

type: float or list of floats *default*: None *example*:

```
temperature: 1200
temperature: [1200, 1300]
```

Temperatures for the simulation in Kelvin. The way temperature is used in `calphy` depends on the selected mode of calculation.

- mode `ts` or `tscale` or `mts`, a temperature sweep is carried out. In that case, only two values of temperature should be specified.
- mode `melting_temperature`, if provided it is used as an initial guess temperature. If not, the experimental melting temperature is used as a guess value.
- all other modes, a calculation is launched for each temperature on the list.

`pressure`

type: None or float or list of floats *default*: None *example*:

```
pressure: None
pressure: 0
pressure: [0,0,0]
pressure: [0, 10000]
pressure: [[1,1,1], [1000, 1000, 1000]]
```

Pressure for the simulation in bars. Depending on the pressure input, other options are automatically set. These options are `iso` and `fix_lattice`. `iso` decides if the barostat relaxes the system isotropically or anisotropically. `fix_lattice` does not relax the lattice at all. A table consistning of possible pressure input options and their effect is below:

`lattice`

type: string or list of strings *default*: None *example*:

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

Lattice to be used for the calculations. 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.

`reference_phase`

type: string or list of strings *default*: None *example*:

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

The protocol to be used for the calculation. The `reference_phase` 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, `reference_phase` command specifies which reference state to use.

`lattice_constant`

type: float or list of floats *default*: Experimental lattice constant *example*:

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

Lattice constant for input structures. 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.

`repeat`

type: list of ints of length 3 *default*: [1, 1, 1] *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.

`n_iterations`

type: int *example*:

```
n_iterations: 3
```

The number of backward and forward integrations to be carried out in all modes. If more than one integration cycle is used, the errors can also be evaluated.

`temperature_high`

type: float *default*: 2*temperature *example*:

```
temperature_high: 1600
```

The temperature used to melt a solid structure to create a liquid. If `reference_phase` is chosen as `liquid`, calphy performs a melting cycle to create an equilibrated liquid structure. calphy starts from the given input structure, and heats it using 2 times the highest temperature provided in the `temperature` option. If the structure is not melted, the temperature is increased progressively. `temperature_high` keyword can be used to set the temperature to overheat the structure and melt it.

---

npt

type: bool *default*: True *example*:

```
npt: True
```

`npt` determines if calculations are carried out in the NPT ensemble. This option is used with modes `alchemy`, `ts` and `mts`. The effect is described below:

- for mode `ts` and `mts`: the reversible scaling approach is carried out in NPT if `npt` is True, otherwise the `NVT` ensemble is used.
- for mode `alchemy`: If `npt` is False, the NVT ensemble is used, meaning that the calculated work during alchemical transformation is calculated on the equilibrated volume of the first pair style.

---

pair_style

type: string or list of strings *example*:

```
pair_style: eam/alloy
pair_style: [eam/alloy, eam/alloy]
pair_style: eam/fs
pair_style: pace
```

The pair style command for LAMMPS. All styles supported in LAMMPS can be used with calphy. Except for mode `alchemy`, only the first value in the list will be used. For mode `alchemy`, there should be two pair styles specified, and the alchemical transformation is carried out between the two.

---

pair_coeff

type: string or list of strings *default*: None *example*:

```
pair_coeff: "* * Cu_EAM/Cu01.eam.alloy Cu"
pair_coeff: ["* * Cu_EAM/Cu01.eam.alloy Cu", "* * Cu_EAM/Cu02.eam.alloy Cu"]
pair_coeff: "* * CuZr_EAM/CuZr.eam.fs Cu Zr"
```

The pair coeff command for LAMMPS. It should be the same length as `pair_style`. Except for mode `alchemy`, only the first value in the list will be used. For mode `alchemy`, there should be two pair styles specified, and the alchemical transformation is carried out between the two.

`potential_file`

type: string *default*: None *example*:

```
pair_coeff: "/home/calc/potential.inp"
```

If specified, the `pair_style` and `pair_coeff` commands are not used, but rather the potential is read in from the provided input file using `include` command in LAMMPS. This allows the use of more complex or multiple potential files. Due to the `hybrid/scaled` styles employed in calphy, **this option only works with mode ``fe`` and ``reference_phase`` solid.**

`n_equilibration_steps`

type: int *default*: 25000 *example*:

```
n_equilibration_steps: 10000
```

The number of time steps for equilibrating the system.

`n_switching_steps`

type: int or list of ints *default*: 50000 *example*:

```
n_switching_steps: 10000
n_switching_steps: [10000, 20000]
```

The number of switching steps. If a list of two integers is provided, the first value is used for mode `fe` while the second value will be used for all other modes.

`n_print_steps`

type: int *default*: 0 *example*:

```
n_print_steps: 100
```

Record MD trajectory during temperature sweep runs in the given interval of time steps. Default 0, trajectories are never recorded.

---

spring_constants

type: list of floats *default*: None *example*:

```
spring_constants: 1.2
spring_constants: [1.2, 1.3]
```

Spring constants for Einstein crystal. If specified, the automatic calculation is not performed. Should be equal to the number of species in the system.

---

equilibration_control

type: str *default*: None *example*:

```
equilibration_control: berendsen
equilibration_control: nose-hoover
```

The barostat and thermostat combination to be used for the equilibration stage. By default, Berendsen will be used for solid reference and Nose-Hoover will be used for liquid. The damping parameters can be tuned using the *nose_hoover* block or the *berendsen* block. Used only for equilibration stage.

---

melting_cycle

type: bool *default*: True *example*:

```
melting_cycle: True
melting_cycle: False
```

If True, a melting cycle is carried out to melt the given input structure. Only used if the `reference_phase` is `"liquid "`.

---

folder_prefix

type: string *default*: None *example*:

```
folder_prefix: set1
```

Prefix string to be added to folder names for calculation. Folders for calculations in calphy are named as `mode-lattice-temperature-pressure`. Therefore, if more than one calculation is run with the same parameters, they will be overwritten. To prevent this, `folder_prefix` can be used. If `folder_prefix` is provided, the folders will be named as `folder_prefix-mode-lattice-temperature-pressure`.

`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:
  timestep: 0.001
  thermostat_damping: 0.1
  barostat_damping: 0.1
```

`timestep`

type: float *default*: 0.001 *example*:

```
timestep: 0.001
```

The timestep for MD in picoseconds.

`thermostat_damping`

type: float *default*: 0.1 *example*:

```
thermostat_damping: 0.1
```

The thermostat damping constant for MD. For damping during equilibration stage see ``*equilibration_control*` <#equilibration_control>`_.

`barostat_damping`

type: float *default*: 0.1 *example*:

```
barostat_damping: 0.1
```

Pressure damping for MD. For damping during equilibration stage see ``*equilibration_control*` <#equilibration_control>`_.

`n_small_steps`

type: int *default*: 10000 *example*:

```
n_small_steps: 10000
```

The number of time steps for equilibration cycles to calculate spring constant and average volume.

`n_every_steps`

type: int *default*: 10 *example*:

```
n_every_steps: 100
```

Keywords to tune how often average values are recorded in LAMMPS. Please see here for more details.

`n_repeat_steps`

type: int *default*: 10 *example*:

```
n_repeat_steps: 10
```

Keywords to tune how often average values are recorded in LAMMPS. Please see here for more details.

`n_cycles`

type: int *default*: 100 *example*:

```
n_cycles: 100
```

Number of cycles to try converging the pressure of the system. If the pressure is not converged after `n_cycles`, an error will be raised. In each `n_cycle`, `n_small_steps` MD steps will be run.

---

queue block

This block consists 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`

type: string *example*:

```
scheduler: slurm
```

The scheduler to be used for the job. Can be `local`, `slurm` or `sge`. The code has been tested only on local and slurm.

`cores`

type: int *example*:

```
cores: 4
```

The number of cores to be used for the job.

`jobname`

type: string *example*:

```
jobname: cu
```

Name of the job. Not used for `local`.

`walltime`

type: string *example*:

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

Walltime for the job. Not used for `local`.

`queuename`

type: string *example*:

```
queuename: "shorttime"
```

Name of the queue. Not used for `local`.

---

memory

type: string *example*:

```
memory: 3GB
```

Memory to be used per core. Not used for `local`.

---

commands

type: list of strings *example*:

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

Command that will be run **before** the actual calculations are carried out. 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.

---

modules

type: list of strings *example*:

```
modules:
  - anaconda
  - lammps
```

List of modules to be loaded before running the calculations. The given module names will be prefixed by `module load`.

---

options

type: string *example*:

```
options:
  - memory: 3GB
```

Extra options to be added to the submission script.

`tolerance` block

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

```
tolerance:
   spring_constant: 0.01
   solid_fraction: 0.7
   liquid_fraction: 0.05
   pressure: 0.5
```

`spring_constant`

type: float *default*: 0.01 *example*:

```
spring_constant: 0.01
```

tolerance for the convergence of spring constant calculation.

`solid_fraction`

type: float *default*: 0.7 *example*:

```
solid_fraction: 0.7
```

The minimum amount of solid particles that should be there in solid.

`liquid_fraction`

type: float *default*: 0.05 *example*:

```
liquid_fraction: 0.05
```

Maximum fraction of solid atoms allowed in liquid after melting.

`pressure`

type: float *default*: 0.5 *example*:

```
pressure: 0.5
```

tolerance for the convergence of pressure.

`melting_temperature` block

This block contains keywords that are used only for the mode `melting_temperature`.

```
melting_temperature:
   step: 200
   attempts: 5
```

`step`

type: int *example*:

```
step: 200
```

Temperature interval for search of melting temperature. Only used if mode is `melting_temperature`.

`attempts`

type: int *example*:

```
attempts: 5
```

The number of maximum attempts to try find the melting temperature in a automated manner. Only used if mode is `melting_temperature`.

`composition_scaling` block

This block contains keywords that are used only for the mode `composition_scaling`.

```
composition_scaling:
  input_chemical_composition:
     - Cu: 512
     - Zr: 512
  output_chemical_composition:
     - Cu: 513
     - Zr: 511
```

`input_chemical_composition`

type: list *example*:

```
input_chemical_composition:
  - Cu: 512
  - Zr: 512
```

The input chemical composition in number of atoms. It should be identical to the input structure provided.

`output_chemical_composition`

type: list *example*:

```
output_chemical_composition:
  - Cu: 513
  - Zr: 511
```

The output chemical composition in number of atoms. The total number of atoms should be equal to the input provided.

`nose_hoover` block

This block contains keywords that are used only for the equilibration stage if ``*equilibration_control*` <#equilibration_control>`_ is `nose_hoover`.

```
nose_hoover:
  thermostat_damping: 0.1
  barostat_damping: 0.1
```

`thermostat_damping`

type: float *default*: 0.1 *example*:

```
thermostat_damping: 0.1
```

The thermostat damping constant for equilibration MD.

---

barostat_damping

type: float *default*: 0.1 *example*:

```
barostat_damping: 0.1
```

Pressure damping for equilibration MD.

---

berendsen block

This block contains keywords that are used only for the equilibration stage if ``*equilibration_control*` <#equilibration_control>`_ is `berendsen`.

```
berendsen:
   thermostat_damping: 100.0
   barostat_damping: 100.0
```

---

thermostat_damping

type: float *default*: 100.0 *example*:

```
thermostat_damping: 100.0
```

The thermostat damping constant for equilibration MD.

---

barostat_damping

type: float *default*: 100.0 *example*:

```
barostat_damping: 100.0
```

Pressure damping for equilibration MD.

1.2.2 Running calphy

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

```
calphy -i inputfilename
```

The `calphy` command will wrap each calculation defined in the `calculations` block into a script and submit it based on the scheduler specified.

Alternatively, if you want to run a calculation directly from the terminal, the `calphy_kernel` command can be used.

```
calphy_kernel -i input.yaml -k 0
```

An extra argument `-k` or `--kernel` is required, which takes an integer argument. This argument refers to the which calculation needs to be run from the input file. For example if the `calculations`block of the input file is as follows:

```
calculations:
- mode: ts
  temperature: [1200, 1400]
  pressure: [0]
  lattice: [FCC, LQD]
  repeat: [5, 5, 5]
  reference_phase: [solid, liquid]
  n_iterations: 1
```

Specify `-k 0` will run a calculation block equivalent to:

```
- mode: ts
  temperature: [1200, 1400]
  pressure: [0]
  lattice: [FCC]
  repeat: [5, 5, 5]
  reference_phase: [solid]
  n_iterations: 1
```

and `-k 1` will run:

```
- mode: ts
  temperature: [1200, 1400]
  pressure: [0]
  lattice: [LQD]
  repeat: [5, 5, 5]
  reference_phase: [liquid]
  n_iterations: 1
```

1.2.3 Running `calphy` on computing clusters

The easiest way to run `calphy` on computing clusters is by using `queue` block in the input file. The `queue` block looks like this:

```
queue:
  scheduler: slurm
  cores: 20
  jobname: fec
  walltime: "23:50:00"
  queuename: shorttime
  memory: 3GB
  commands:
    - source ~/.bashrc
    - conda activate calphy
```

The default `scheduler` is local, which means that the calculations are run on the local machine. Instead `slurm` or `sge` can be specified to run on computing clusters. The number of cores, walltime, queue name etc can be set during the respective keywords. Note that if you are using a conda environment, it needs to be activated. This can be done using the `commands` argument. Anything listed within the `commands` argument is copied directly to the submission script.

Adding a new scheduler

In the current version only `slurm` and `sge` schedulers are supported. There are two ways in which a new scheduler can be added, they are described below:

Editing `scheduler.py`

The scheduler are implemented in `calphy` within `calphy/scheduler.py` file. An existing class can be copied and modified to support the new submission script.

Adding calphy to a submission script

An easier method would be to add `calphy` manually within a submission script. `calphy` offers a `calphy_kernel` command line interface which can be used for direct submission. The above `queue` block can be translated to an equivalent submission script as follows:

```
#!/bin/bash
#SBATCH --job-name=fec
#SBATCH --time=23:50:00
#SBATCH --partition=shorttime
#SBATCH --ntasks=20
#SBATCH --mem-per-cpu=3GB

source ~/.bashrc
conda activate calphy

calphy_kernel -i input.yaml -k 0
```

Note that `calphy_kernel` is called instead of `calphy`. Additionally an extra argument `-k` or `--kernel` is required, which takes an integer argument. This argument refers to the which calculation needs to be run from the input file. For example if the `calculations`block of the input file is as follows:

```
calculations:
- mode: ts
  temperature: [1200, 1400]
  pressure: [0]
  lattice: [FCC, LQD]
  repeat: [5, 5, 5]
  reference_phase: [solid, liquid]
  n_iterations: 1
```

Specify `-k 0` will run a calculation block equivalent to:

```
- mode: ts
  temperature: [1200, 1400]
  pressure: [0]
  lattice: [FCC]
  repeat: [5, 5, 5]
```

(continues on next page)

(continued from previous page)

```
  reference_phase: [solid]
  n_iterations: 1
```

and `-k 1` will run:

```
- mode: ts
  temperature: [1200, 1400]
  pressure: [0]
  lattice: [LQD]
  repeat: [5, 5, 5]
  reference_phase: [liquid]
  n_iterations: 1
```

CHAPTER

TWO

EXAMPLES

2.1 Examples

2.1.1 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.

2.1.2 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]
  reference_phase: [solid]
  n_iterations: 1
  pair_style: eam
  pair_coeff: "* * ../potentials/Fe.eam"
  n_equilibration_steps: 10000
  n_switching_steps: 25000
- mode: ts
  temperature: [100, 1400]
  pressure: [0]
  lattice: [FCC]
  repeat: [5, 5, 5]
  reference_phase: [solid]
  n_iterations: 1
  lattice_constant: [6.00]
  pair_style: eam
  pair_coeff: "* * ../potentials/Fe.eam"
  n_equilibration_steps: 10000
  n_switching_steps: 25000
```

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:

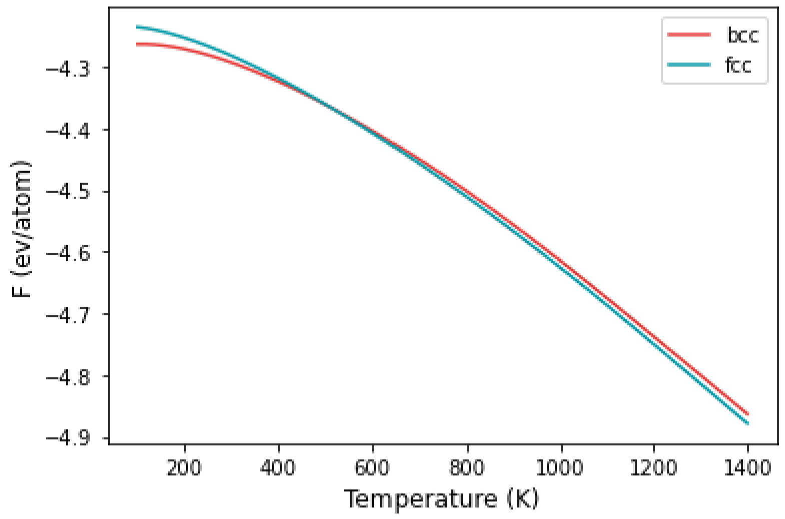
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
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)

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

```
508.07011498303046 K
```

[3]:
```
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()
plt.savefig("fe_transition.png", dpi=300, bbox_inches="tight")
```



nbsphinx-code-borderwhite

[ ]:

2.1.3 Example 03: Manual 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
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
```

(continues on next page)

(continued from previous page)

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

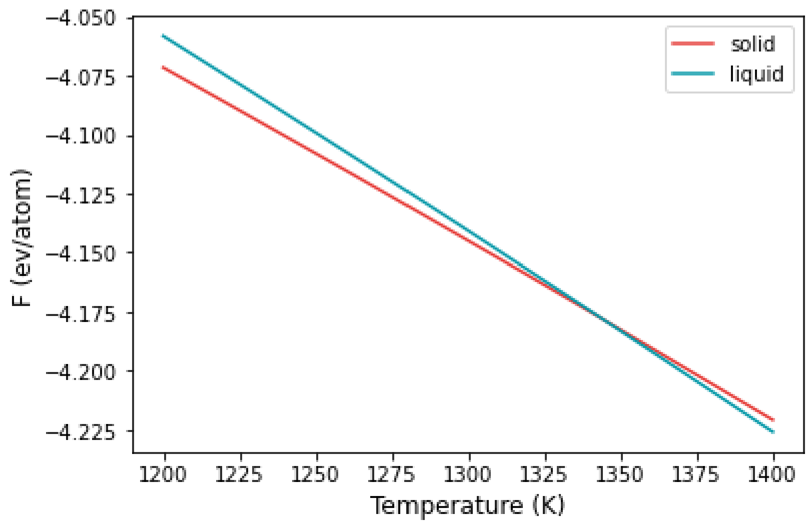
The main difference here is that under the `average` block, the `density` is reported instead of `spring_constant`.

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.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[3]:
```
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)
```

[4]:
```
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()
plt.savefig("tm.png", dpi=300, bbox_inches="tight")
```



nbsphinx-code-borderwhite

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.

[5]:
```
args = np.argsort(np.abs(sfe-lfe))
print(st[args[0]], "K")
```

```
1344.3095672974282 K
```

2.1.4 Example 04: Automated calculation of melting temperature

In Example 03, we calculated the melting temperature for Cu. The same can be done in a fully automated way using Calphy.

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 `melting_temperature`. This mode is special, and needs much less input information than the other common modes. For example, options such as `lattice`, `state` and `temperature` are (generally; see below for specific cases) not needed.

The input file in this case is shown below:

```
element: Cu
mass: 63.546
calculations:
- mode: melting_temperature
  pressure: [0]
  repeat: [5, 5, 5]
  nsims: 1

md:
  pair_style: eam/alloy
  pair_coeff: "* * ../potentials/Cu01.eam.alloy Cu"
```

(continues on next page)

(continued from previous page)

```
  timestep: 0.001
  nsmall: 10000
  tdamp: 0.1
  pdamp: 0.1
  te: 10000
  ts: 15000

queue:
  scheduler: local
  cores: 4
  commands:
    - conda activate calphy
```

Once the input file is set up, the calculation can be run using:

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

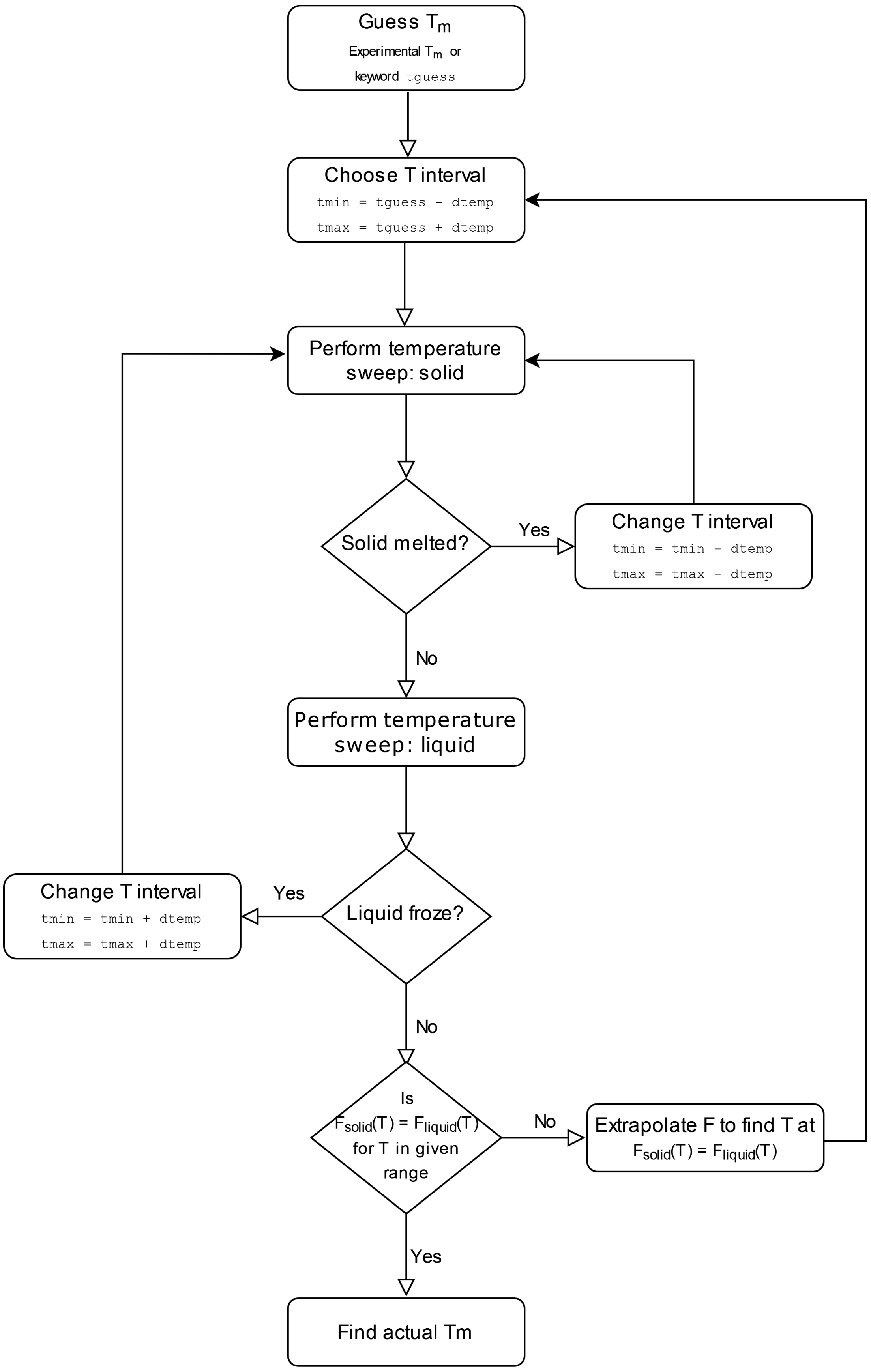
A log file called `calphy.log` is also be produced. The log file contains a lot of information about the calculation. Here the most important ones are discussed. The important log file contents are prefixed with `STATE`. On running `grep STATE calphy.log`, the following output is produced.

```
calphy.helpers INFO     STATE: Temperature range of 1156.600000-1556.600000 K
calphy.helpers INFO     STATE: Tm = 1340.39 K +/- 0.00 K, Exp. Tm = 1356.60 K
```

The calculated melting temperature for this interatomic potential is 1340 K.

How does it work?

A flowchart of the automated melting temperature calculation is shown below:


Guess T_m
Experimental T_m or keyword tguess
Choose T interval
tmin = tguess - dtemp
tmax = tguess + dtemp
Perform temperature sweep: solid
Solid melted?
Yes
Change T interval
tmin = tmin - dtemp
tmax = tmax - dtemp
No
Perform temperature sweep: liquid
Liquid froze?
Yes
Change T interval
tmin = tmin + dtemp
tmax = tmax + dtemp
No
Is $F_{solid}(T) = F_{liquid}(T)$ for T in given range
No
Extrapolate F to find T at $F_{solid}(T) = F_{liquid}(T)$
Yes
Find actual Tm

FAQs

- **How can I tune the initial guess temperature?**
  The initial guess temperature can be tuned using the keyword `tguess` in the `md` block.
- **How can I tune the width of the temperature range?**
  The width of the temperature scan range can be tuned using the keyword `dtemp` in the `md` block.
- **What if the system undergoes a solid-solid phase transition before melting?**
  The lattice that calphy automatically chooses for the solid is the **ground state**. For some elements, for example Ti, a solid-solid phase transition occurs before melting. HCP Ti transforms to BCC Ti and then to liquid. Therefore to use the automated method, the solid lattice has to be specified using the `lattice` keyword.
- **How can I calculate melting temperature at non zero pressure?**
  The required pressure can be specified using the `pressure` keyword. Note that specifying `tguess` could help speed up the calculation.

2.1.5 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.

[3]:
```
import numpy as np
import matplotlib.pyplot as plt
```

The starting temperatures and pressures for our calculations:

[4]:
```
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:

[5]:
```
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))
```

(continues on next page)

(continued from previous page)

```
    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.

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

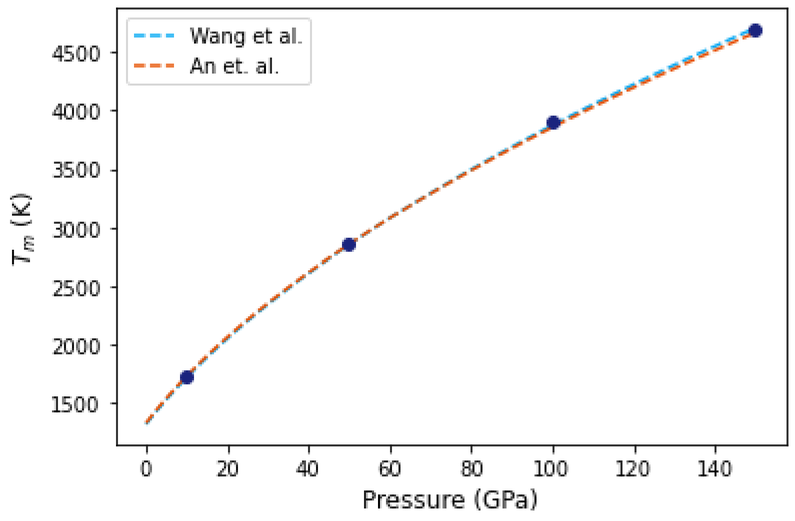
- An, Qi, Sheng-Nian Luo, Li-Bo Han, Lianqing Zheng, and Oliver Tschauner. "Melting of Cu under Hydrostatic and Shock Wave Loading to High Pressures." Journal of Physics: Condensed Matter 20, no. 9 (March 5, 2008): 095220.

```
[8]: 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.

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

```
[11]: 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()
      plt.savefig("tp-diagram.png", dpi=300, bbox_inches="tight")
```



nbsphinx-code-borderwhite

2.1.6 Example 06: Running calphy from jupyter notebooks

In this example, calphy will be used as a library to run Example 01 directly from a jupyter notebook. Please check example 01 before completing this example. We start by import a function to read the input file.

```
[1]: from calphy.input import read_inputfile
```

```
[2]: options = read_inputfile("input.yaml")
```

We can check the keys of the options dictionary.

```
[3]: options.keys()
```

```
[3]: dict_keys(['calculations', 'md', 'queue', 'conv', 'element', 'mass', 'nelements'])
```

The individual methods that are required to run the calculation can be imported from the queuekernel module.

```
[4]: import calphy.queuekernel as cq
```

First, we set up a class which prepares everything for the calculation. It takes options as the first argument, followed by an integer argument. The second argument indicates which calculation from the calculation block is run.

```
[5]: job = cq.setup_calculation(options, 0)
```

```
[6]: job
```

```
[6]: solid system with T=100.000000, P=0.000000 in bcc lattice for mode fe
```

The specifics of the calculation can be obtained by,

```
[7]: job.calc
```

```
[7]: {'mode': 'fe',
      'temperature': 100,
      'pressure': 0,
      'lattice': 'BCC',
      'state': 'solid',
      'temperature_stop': 100,
      'nelements': 1,
      'element': ['Fe'],
      'lattice_constant': 2.8842,
      'iso': True,
      'fix_lattice': False,
      'repeat': [5, 5, 5],
      'nsims': 1,
      'thigh': 200.0,
      'directory': 'fe-BCC-100-0'}
```

These properties can also be accessed individually.

```
[9]: job.t, job.p, job.l
```

```
[9]: (100, 0, 'bcc')
```

Now finally the calculation can be run

```
[8]: job = cq.run_calculation(job)
```

The results can be obtained through the `report` variable

```
[10]: job.report
```

```
[10]: {'input': {'temperature': 100,
        'pressure': 0.0,
        'lattice': 'bcc',
        'element': 'Fe',
        'concentration': '1'},
       'average': {'vol/atom': 11.994752673986264, 'spring_constant': '3.35'},
       'results': {'free_energy': -4.263447380763835,
        'error': 0.0,
        'reference_system': 0.015145685052402l6,
        'work': -4.278593065816237,
        'pv': 0.0}}
```

or individually as class attributes

```
[11]: job.fe, job.w, job.pv
```

```
[11]: (-4.263447380763835, -4.278593065816237, 0)
```

If more than one calculation is present, they should be run individually. For example, we use the `input2.yaml` file.

```
[13]: options = read_inputfile("input2.yaml")
```

This input file has two structures: BCC and liquid. We can check the list of calculations

```
[17]: len(options["calculations"])
```

```
[17]: 2
```

If you want to run the second calculation in the list, we have to set up the job as follows.

```
[ ]: job = cq.setup_calculation(options, 1)
```

2.1.7 Example 07: Upsampling calculations

In this example, upsampling calculations which can be used to switch a system between two different interatomic potentials is illustrated.

The major change in the input file is that both `pair_style` and `pair_coeff` keywords have two arguments. These are the two potentials between which the transformation will be carried out.

The first potential is a Finnis-Sinclair (FS) potential for copper:

M.I. Mendelev, M.J. Kramer, C.A. Becker, and M. Asta (2008), "Analysis of semi-empirical interatomic potentials appropriate for simulation of crystalline and liquid Al and Cu", Philosophical Magazine, 88(12), 1723-1750.

The second potential is an EAM:

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.

If we know the free energy of the FS potential at a given temperature, we can calculate the free energy of the EAM through upsampling calculations. Upsampling calculations generally need only much less switching time, thus is quite useful in the case of expensive interatomic potentials, such as machine learning potentials.

We start by calculating the free energy of the FS potential. The input file is available at `pot1/input-fe.yaml`. As usual, the calculation can be run using `calphy -i input-fe.yaml`. We can now load the `report.yaml` file and check the free energy.

[1]:
```
import yaml
import numpy as np
```

[2]:
```
with open('pot1/fe-FCC-600-0/report.yaml', 'r') as fin:
    pot1 = yaml.safe_load(fin)
```

[3]:
```
pot1['results']['free_energy']
```

[3]:
```
-3.4389380251304913
```

Now we can transform FS to EAM potential (see input file above). After running the calculation, we can check the free energy of this alchemical transformation.

[4]:
```
with open('alchemy-FCC-600-0/report.yaml', 'r') as fin:
    alchemy = yaml.safe_load(fin)
```

[5]:
```
alchemy['results']['free_energy']
```

[5]:
```
-0.25451608590151586
```

The free energy of the EAM potential, $F_{\mathrm{EAM}} = F_{\mathrm{FS}} + F_{\mathrm{upsampling}}$

[6]:
```
pot1['results']['free_energy']+alchemy['results']['free_energy']
```

[6]:
```
-3.693454111032007
```

We can verify this calculation by directly computing the free energy of the EAM potential. The input file is available at `pot2/input-fe.yaml`.

[7]:
```
with open('pot2/fe-FCC-600-0/report.yaml', 'r') as fin:
    pot2 = yaml.safe_load(fin)
```

[8]:
```
pot2['results']['free_energy']
```

[8]:
```
-3.6923377327832667
```

We can see that the directly calculation is in meV agreement with the upsampling calculations.

2.1.8 Example 08: Temperature scaling two ways

Finding the free energy as a function of the temperature can be calculated using reversible scaling methodology (mode `ts`) as discussed in previous examples.

However, calphy provides another mode `tscale` that can be used for the same purpose. In this example, `ts` and `tscale` modes are compared. Before that:

- *What is the difference between ``ts`` and ``tscale``?*

`ts` performs reversible scaling method. Here, the simulation is performed at a constant temperature; but the potential energy is scaled. In `tscale`, the temperature is directly scaled.

- *Why do we need ``tscale``?*

  If `ts` mode works for the given potential, it should be used. `ts`, however, needs the `` `pair hybrid/scaled `` <https://docs.lammps.org/pair_hybrid.html>`__. If this is not possible due to some restriction of the potential, `tscale` can be used.

For this example, we will use the following EAM potential: 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 given in `input.yaml`. There are two sets of calculations, one with mode `tscale`, and another one with mode `ts` that can be used for comparison.

The calculation can be run by,

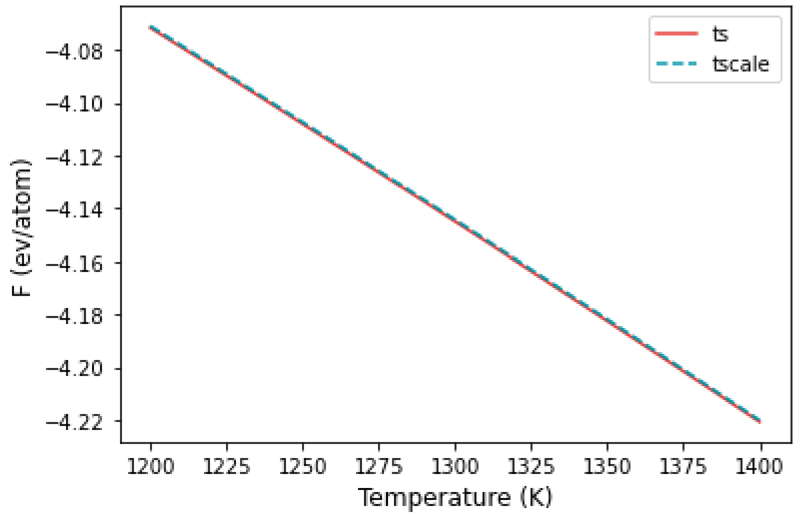
```
calphy -i input.yaml
```

After the caculation is over, we can plot the free energy as a function of temperature.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
t1, fe1, ferr1 = np.loadtxt("ts-FCC-1200-0/temperature_sweep.dat", unpack=True)
t2, fe2, ferr2 = np.loadtxt("tscale-FCC-1200-0/temperature_sweep.dat", unpack=True)
```

[4]:
```
plt.plot(t1, fe1, color="#E53935", label="ts")
plt.plot(t2, fe2, color="#0097A7", ls="dashed", label="tscale")
plt.xlabel("Temperature (K)", fontsize=12)
plt.ylabel("F (ev/atom)", fontsize=12)
plt.legend()
```



nbsphinx-code-borderwhite

As seen from the plot above, both modes agree very well with each other. Now we can compare the error in both methods.

2.1.9 Example 09: Pressure scaling

In some cases, it is useful to obtain the free energy as a function of the pressure. The calphy mode `pscale` allows to do this.

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 pressure scale input file is `input2.yaml`, in which we calculate the free energy at 1200 K from 0 to 1 GPa. To compare, we do direct free energy calculations at the same temperature and pressures of 0, 0.5 and 1 GPa. The input for this is provided in `input1.yaml`.

The calculation can be run by:

```
calphy -i input1.yaml
calphy -i input2.yaml
```

After the calculations are over, we can read in and analyse the results.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
p1 = [0, 5000, 10000]
fe1 = [-4.072, -4.032, -3.993]
```

The above values are taken from the `report.yaml` files in the corresponding folders.

Note that calphy uses bar as the unit of pressure.

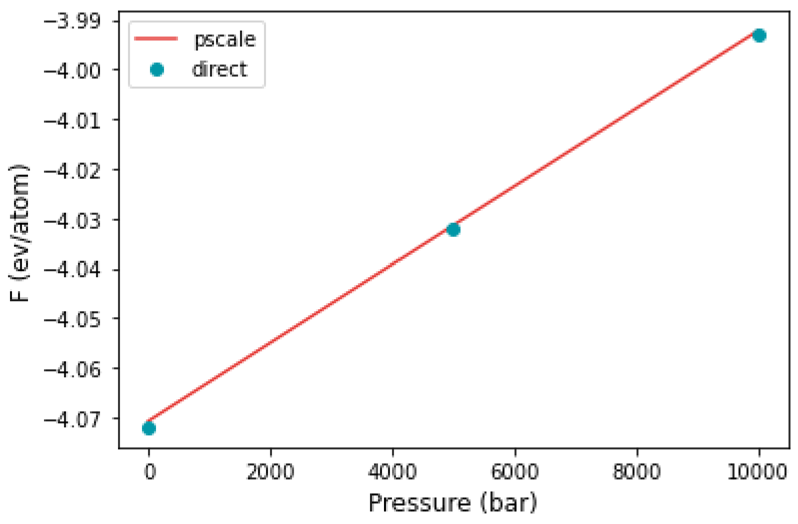
Now read in the pressure scaling results.

[5]:
```
p2, fe2, ferr2 = np.loadtxt("pscale-FCC-1200-0/pressure_sweep.dat", unpack=True)
```

Now plot

[6]:
```
plt.plot(p2, fe2, color="#E53935", label="pscale")
plt.plot(p1, fe1, 'o', color="#0097A7", label="direct")
plt.xlabel("Pressure (bar)", fontsize=12)
plt.ylabel("F (ev/atom)", fontsize=12)
plt.legend()
```

[6]:
```
<matplotlib.legend.Legend at 0x7fe37731d0c0>
```



nbsphinx-code-borderwhite

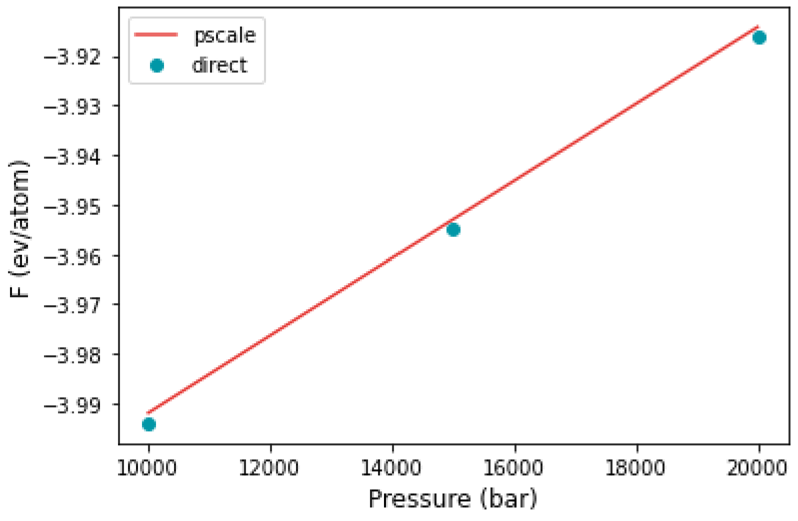
There is excellent agreement between direct calculations and pressure scaling. Now we can also do the calculations for a pressure range of 1-2 GPa. The input is given in `input3.yaml` and `input4.yaml` for the pressure scaling and direct calculations respectively.

```
[49]: p3 = [10000, 15000, 20000]
      fe3 = [-3.994, -3.955, -3.916]
```

```
[50]: p4, fe4, ferr4 = np.loadtxt("pscale-FCC-1200-10000/pressure_sweep.dat", unpack=True)
```

```
[52]: plt.plot(p4, fe4, color="#E53935", label="pscale")
      plt.plot(p3, fe3, 'o', color="#0097A7", label="direct")
      plt.xlabel("Pressure (bar)", fontsize=12)
      plt.ylabel("F (ev/atom)", fontsize=12)
      plt.legend()
```

```
[52]: <matplotlib.legend.Legend at 0x7f71512557b0>
```



nbsphinx-code-borderwhite

Once again, for the higher pressure interval, there is excellent agreement between the two methods.

2.1.10 Example 10: Composition scaling

In this notebook, we will calculate the free energy of a binary ZrCu system as a function of the composition. The structure that will be considered is ZrCu with a B2 structure. The potential that will be used is-

M.I. Mendelev , M.J. Kramer , R.T. Ott , D.J. Sordelet , D. Yagodin & P. Popel (2009) Development of suitable interatomic potentials for simulation of liquid and amorphous Cu–Zr alloys, Philosophical Magazine, 89:11, 967-987

In the first part of the example, a simple free energy difference that happens when a substitutional atom is introduced is calculated in two ways. In the second part, we directly calculate the free energy as the composition varies in the given system.

Free energy change with substitutional atom

In the file `` `ZrCu.dump `` <ZrCu.dump>`__, CuZr in B2 structure is provided. The file `` `ZrCu_substitution.dump `` <ZrCu_substitution.dump>`__ contains the same structure, but a single atom of Zr is replaced with Cu. If the free energy of both structures are calculated, the difference between the two provides the free energy change when a substitutional Cu atom is introduced.

The input files for the direct calculations are `` `input-direct-1.yaml `` <input-direct-1.yaml>`__ and `` `input-direct-2.yaml `` <input-direct-2.yaml>`__. The calculations can be run by `calphy -i inputfilename`. Once the calculations are over, we can read in the results and analyse them.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from calphy.input import read_report
from uncertainties import ufloat
```

We import the `read_report` method as shown above. This allows for easy reading of calphy output files. The data from the pure structure is read into `fe_ref_direct` and the data with the substitutional atom is in `fe_sub_direct`. We also import the `` `uncertainties `` <https://pythonhosted.org/uncertainties/>`__ package, which will allow us to work with error bars.

[2]:
```
fe_ref_direct = read_report("fe-ZrCu.dump-800-0")
fe_sub_direct = read_report("fe-ZrCu_substitution.dump-800-0")
```

Now calculate the difference in free energy between the two structures. We create `ufloat` objects to do operations including error bars.

[4]:
```
fe_ref = ufloat(fe_ref_direct["results"]["free_energy"], fe_ref_direct["results"]["error
↪"])
fe_sub = ufloat(fe_sub_direct["results"]["free_energy"], fe_sub_direct["results"]["error
↪"])
```

Now the difference

[5]:
```
fe_sub - fe_ref
```

[5]:
```
0.0036244140356958+/-0.0005247047613173691
```

Another way to do the same calculation is through the `composition_scaling` mode in calphy. The input is given in the file `` `input-substitution.yaml `` <input-substitution.yaml>`__. The mode is `composition_scaling`. Further more, there is a new block in the input file:

```
composition_scaling:
  input_chemical_composition:
      - Cu: 512
      - Zr: 512
  output_chemical_composition:
      - Cu: 513
      - Zr: 511
```

which provides the details for composition change. The `input_chemical_composition` shows equal number of Cu and Zr atoms as expected in the B2 structure. The `output_chemical_composition` has 513 Cu and 511 Zr atoms, which means that one Cu atom is substituted for Zr over the calculation.

As usual, the calculation can be run by:

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

Once the calculation is over, we can read in the data.

```
[8]: fe_sub_comp = read_report("sub-composition_scaling-ZrCu.dump-800-0")
```

```
[9]: fe_sub_comp = ufloat(fe_sub_comp["results"]["free_energy"], fe_sub_comp["results"]["error
     ↪"])
     fe_sub_comp
```

```
[9]: 0.0037993834787761636+/-1.1368085265609911e-05
```

Both approaches we tried agree very well with each other. The second method needs only one single calculation instead of two different ones, and exhibits lower error overall. We can extend this approach to find the free energy as a function of the composition over a given range.

Free energy variation with composition

The `composition_scaling` block in the input file looks slightly different here:

```
composition_scaling:
  input_chemical_composition:
      - Cu: 512
      - Zr: 512
  output_chemical_composition:
      - Cu: 532
      - Zr: 492
```

Once again, the `input_chemical_composition` shows equal number of Cu and Zr atoms, as expected in the B2 structure. The `output_chemical_composition` has 532 Cu and 492 Zr atoms, which amounts to 48 at. % Zr. The composition integration therefore will span 48-50% of Zr.

As usual, the calculation can be run by:

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

Once the calculation is run, there is a file `composition_scaling.dat` created in the simulation folder. We can read in this file, to get the information. There are a number of columns in the file, but we will just use the first two.

```
[10]: flambda_arr, netfe = np.loadtxt("composition_scaling-ZrCu.dump-800-0/composition_sweep.
      ↪dat", unpack=True,
                                      usecols=(0,1))
```

The first column, `flambda_arr` goes from 1.00 to 0.00. This spans the composition interval. At 1.00, the composition is 50 % Zr, while at 0.00, the composition is 48 %. First, the array is converted to the actual composition.

```
[11]: comp = 0.50-flambda_arr*(0.50-0.48)
```

To compare our results, we will use the data from this publication:

Tang, C, and Peter Harrowell. "Predicting the Solid State Phase Diagram for Glass-Forming Alloys of Copper and Zirconium." Journal of Physics: Condensed Matter 24, no. 24 (June 20, 2012): 245102.

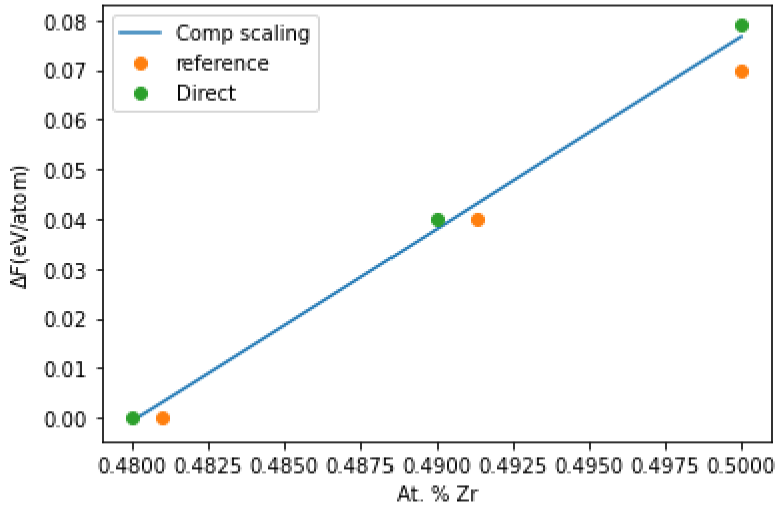
```
[12]: comp_reference = np.array([0.4810, 0.4913, 0.5])
      fe_reference = np.array([-5.27, -5.31, -5.34])
```

We will also use direct calculations done using calphy. The results from direct calculations are given here for easiness.

```
[13]: comp_direct = np.array([0.48, 0.49, 0.50])
      fe_direct = np.array([-5.271, -5.311, -5.350])
```

Note that our results, `netfe` only includes the free energy difference as composition change. We will modify the other calculations to plot the energy difference.

```
[14]: plt.plot(comp, netfe, label="Comp scaling")
      plt.plot(comp_reference, fe_reference[0]-fe_reference, "o", label="reference")
      plt.plot(comp_direct, fe_direct[0]-fe_direct, "o", label="Direct")
      plt.legend()
      plt.xlabel("At. % Zr")
      plt.ylabel(r"$ \Delta F $(eV/atom)");
```



nbsphinx-code-borderwhite

As seen from the plot, we have excellent agreement. However, we can obtain the whole range from a single calculation.

CHAPTER

THREE

API REFERENCE

3.1 calphy package

3.1.1 Submodules

3.1.2 calphy.alchemy module

3.1.3 calphy.helpers module

3.1.4 calphy.input module

3.1.5 calphy.integrators module

3.1.6 calphy.kernel module

3.1.7 calphy.lattice module

3.1.8 calphy.liquid module

3.1.9 calphy.queuekernel module

3.1.10 calphy.scheduler module

3.1.11 calphy.solid module

3.1.12 calphy.splines module

3.1.13 Module contents

CHAPTER FOUR

INDICES AND TABLES

- genindex
- modindex
- search