The Alloy-Theoretic Automated Toolkit (ATAT): A User Guide

Axel van de Walle

January 27, 2005

Chapter 1

Introduction

The Alloy-Theoretic Automated Toolkit (ATAT) is a generic name that refers to a collection of alloy theory tools:

- A code to construct cluster expansions from first-principles (MAPS). A cluster expansion is a very compact and efficient expression giving the energy of an substitutional alloy as a function of its configuration (i.e. which type of atom sits where on the lattice).
- A code to perform Monte Carlo simulation of lattice models in order to compute thermodynamic properties of alloys starting from a cluster expansion.
- Extension of the two above tools that allow the construction of so-called reciprocal-space cluster expansion, which are useful to model the energetics of alloys exhibiting a large atomic size mismatch.
- Utilities to interface the above tools with first-principles codes (such as VASP).
- Job control utilities that enable the efficient use of a cluster of workstations to run the first-principles codes that provide the input to the above codes.

Chapter 2

Credits and Licence

The Alloy-Theoretic Automated Toolkit (ATAT)[1] is a generic name that refers to a collection of alloy theory tools developped by Axel van de Walle[2], in collaboration with various research groups.

2.1 Collaborators

The MAPS[3] (MIT Ab-initio Phase Stability) code, which automatically constructs a cluster expansion from the result of first-principles calculations, was developped by Axel van de Walle in collaboration with Prof. Gerd Ceder's group[4] from the Department of Materials Science and Engineering at the Massachusetts Institute of Technology. MAPS consists of the following codes: maps, corrdump, genstr, checkcell, kmesh, cv.

The EMC2 (Easy Monte Carlo Code), which automate the calculation of alloy thermodynamic properties via Monte Carlo simulations of lattice models, were developped by Axel van de Walle in collaboration with Prof. Mark Asta's group[5] from the Department of Materials Science and Engineering at Northwestern University. EMC2 consists of the following codes: emc2, phb.

The CSE (Constituent Strain Extension) to both the MAPS and EMC2 codes, which implement the constituent strain formalism based on a reciprocal-space cluster expansion, was developped by Axel van de Walle in collaboration with Alex Zunger's Solid State Theory Group[6] at the National Renewable Energy Laboratory in Golden, Colorado and in collaboration with Gus Hart[7] from the Department of Physics and Astronomy at Northern Arizona University. CSE conssists of the following files: csfit predcs.cc, predrs.cc, kspacecs.cc.

Volker Blum at NREL has contributed to improve the portability of the package by providing a `perl` version of the `chl` utility.

Dongwon Shin at Penn State has converted a large number of common lattices (found at the NRL navy web site[8]) into the `atat` format. See directory `data/str`.

2.2 Financial Support

The developpement of MAPS was supported by the U.S. Department of Energy, Office of Basic Energy Sciences, under contract no. DE-F502-96ER 45571. Gerbrand Ceder acknowledges support of Union Minière through a Faculty Development Chair. Axel van de Walle acknowledges support of the National Science Foundation under program DMR-0080766 and DMR-0076097 during his stay at Northwestern University.

[1] http://cms.northwestern.edu/atat/
[2] http://www.mit.edu/~avdw/
[3] http://www.mit.edu/~avdw/maps
[4] http://burgaz.mit.edu/
[5] http://cms.northwestern.edu/
[6] http://www.sst.nrel.gov/
[7] http://www.phy.nau.edu/~hart/
[8] http://cst-www.nrl.navy.mil/lattice/

The developpement of EMC2 was supported by the NSF under program DMR-0080766.
The developpement of the CSE is supported by the NSF under program DMR-0080766.

2.3 Licence and Agreements

Any scientific article or book whose results were obtained with the codes described above must properly acknowledge their use by citing the following papers:

1. A. van de Walle and G. Ceder, "Automating First-Principles Phase Diagram Calculations[9]", *Journal of Phase Equilibria*, **23**, 348, (2002).
2. A. van de Walle and M. Asta, "Self-driven lattice-model Monte Carlo simulations of alloy thermodynamic properties and phase diagrams[10]", *Modelling Simul. Mater. Sci. Eng.* **10**, 521, (2002).
3. A. van de Walle, M. Asta and G. Ceder, "The Alloy Theoretic Automated Toolkit: A User Guide[11]" *CALPHAD Journal*, **26**, 539, (2002).
4. If the constituent strain extension is used: D. B. Laks and L. G. Ferreira and S. Froyen and A. Zunger, *Phys. Rev. B* **46**, p. 12587 (1992).

The files included in this distribution cannot be further distributed either in their original or in a modified form without consent of the author, Axel van de Walle (avdw@alum.mit.edu).

Users are free to modify the code solely for their personal use and are encouraged to share their improvements with the author at (avdw@alum.mit.edu[12]). Their contributions will be acknowledged in the present section, in future versions of this manual.

2.4 Beta testers

The following researchers have provided numerous constructive comments that have proven extremely useful to improve the quality of the program.

1. Dane Morgan (MIT, Gerd Ceder's group)
2. Dinesh Balachandran (MIT, Gerd Ceder's group)
3. Ben Burton (National Institute of Standards and Technology)
4. Gautam Ghosh (Northwestern University)
5. Nikolai Andreevich Zarkevich (University of Illinois at Urbana-Champaign, Duane Johson's group)
6. Volker Blum (NREL, Alex Zunger's group)
7. Chris Woodward (Northwestern University/Air Force)
8. Zhe Liu (Northwestern University, Mark Asta's group)
9. Yi Wang and Raymundo Arroyave (Pennsylvania State University, Long-Qing Chen and Zhe-Kui Liu's group)
10. Elif Ertekin (University of California at Berkeley, Daryl Chrzan's group)
11. Rodrigo Barbosa Capaz (University of California at Berkeley)
12. Sundar Amancherla (General Electric)

[9] http://arXiv.org/abs/cond-mat/0201511
[10] http://arXiv.org/abs/cond-mat/0201473
[11] http://arxiv.org/abs/cond-mat/0212159
[12] mailto:avdw@alum.mit.edu

Chapter 3

Getting started

3.0.1 Requirements

You need the following utilities installed:

- `g++` version 2.7.2 or later. Type `g++ --version` to verify this. This package can be downloaded from http://www.gnu.org/.
- GNU `make` (any version). Type `make --version` to verify this. On some systems this command may be called `gmake` or `gnumake`. This package can be downloaded from http://www.gnu.org/.
- A first-principle electronic structure calculation code, such as VASP[1]
- You may want to use `gnuplot` to plot the output of the code. Type `gnuplot` and check the program starts (Type `q` to quit). If not, it can be downloaded from http://www.gnuplot.info/.
- You may need `ssh` if you have multiple machines and they are connnected through an unsecure network (e.g. the internet). This package can be downloaded from http://www.openssh.com/.

3.0.2 Installation

Type

```
gunzip atat.tar.gz

tar -xvf atat.tar.gz
```

These commands create a directory called atat in the current directory. It contains the whole package. For future reference, I'll call the whole access path to this directory `ATAT`.

Type

```
cd atat
```

and open the file `makefile` with a text editor and look for the line `BINDIR=$(HOME)/bin/`. Change `$(HOME)/bin/` to point wherever you want to put the executables. Type

```
make
```

If no error message appears, proceed with the next steps, otherwise consult Chapter 6.

```
make install

rehash (not necessary with bash shell)
```

[1] http://cms.mpi.univie.ac.at/vasp/

3.0.3 Test MAPS with a simple example

Change to a directory of your choice (preferably empty) and type

```
cp ATAT/examples/cuau.in lat.in
```

```
maps -d &
```

Maps is running and waiting for a signal. Type

```
touch ready
```

to indicate that you are ready to for maps to generate a structure. Maps replies `Finding best structure...` To find the structure just created, wait for `done` to appear and type:

```
ls */wait
```

to observe that directory `0` has been created. This directory contains a file `str.out` which describes the structure whose energy needs to be calculated. The file `wait` is just a flag that allows you to find the newly created directory. Let's pretend that we have computed the energy of that structure. We need to let maps know about it. Type, for instance:

`echo 1.1 > 0/energy` (If 1.1 is the energy of the structure.)

```
rm 0/wait
```

Maps responds by `Finding best cluster expansion...`, followed by `done`. You can repeat the process (`touch ready`, etc.) to add more structures. Maps will update the current cluster expansion every time a new energy becomes known. (By default, maps checks every 10 sec.). For a description of the output files, type:

```
maps -h | more
```

or refer to section 5 A nice utility called `mapsrep` allows you to plot the results using gnuplot. To stop maps cleanly, type: `touch stop`

Suggestion: to clarify the output of the program, it is recommended that you run `maps` in one terminal window and type all other command in another terminal window.

3.1 Install the interface between MAPS and VASP

Type

```
ezvasp
```

and follow the instructions posted on the screen to configure this command.

To test this interface, change to a directory of your choice and type

```
maps -d &
```

(unless maps is running already in that same directory).

While MAPS automatically create files that describe the geometry of the structures (called`n/str.out`, where `n` is the structure name), we need to provide a file containing all the other parameters needed by the first-principles code. Type:

```
cp ATAT/glue/vasp/vasp.wrap .
```

to copy an example of such file in the current directory. For a description of these parameters, type:

```
ezvasp -h | more
```

Let's say you have a new structure in directory `0` (created by typing `touch ready`). Type:

```
cd 0

runstruct_vasp
```

When the command has terminated, the directory `0` will contain a file `energy` giving the energy of the structure. If error messages appear consult Chapter 6.

If no error messages appear, you can proceed another level up in automation. Type

`cd ..` (to go back into the main directory)

```
pollmach runstruct_vasp &
```

This script will automatically call the above command repeatedly. To stop it cleanly, type:

```
touch stoppoll
```

(Disregard the warning message.) If you only have access to one machine, this is as good as it gets, if you have more than one machine, read the next section. If you want to use another code than VASP, read section 3.3.

3.2 Install and test the job control utilities

This section requires some knowledge of UNIX, but it is worth it! Networking problems can be tricky and we will test various thing as we go along. You only need to perform this installation on your "master" machine that will run maps. All other machine (which we be called the "remote" machines) only need to have VASP (or any other ab-initio code).

Before you start, you must first make sure that it is possible to login from the master machine to the remote machines without entering a password. This is essential for the program to be able to run on its own, without your intervention. Don't worry it is generally possible to do this without compromising the security of your system. Two commands allow you to run a command on a remote machine. If the master and remote machines are connected through a secure network (e.g. a beowolf cluster having its own local network) or if you don't care about security (for now), you can use `rsh`. Otherwise, `ssh` provides a secure way to command a machine remotely.

To set up `rsh` so that you can login without typing a password, you must have the appropriate `.rhosts` file on the remote machine. For more information, consult the `rsh` man page. (One important issue, often not mentioned in the man pages, its that you need to set the file permissions of the `.rhosts` file so that noone else but you has "write" permission: `chmod og-w ~/.rhosts`).

To set up `ssh` so that you can login without typing a password, consult the `ssh` man page, especially the section on "RSA-based host authentication". (This is the feature that makes the login secure even if no password is needed.) In general, setting this up involves creating a `.shosts` file and generating a public key files to be copied onto the remote machines.

If your username is different on the remote machine and if you use `ssh`, use the syntax `node username@host` instead. If you use `rsh`, use the syntax `node -l username host`.

Once you are able to run either `rsh node2 ls` or `ssh node2 ls` and get the content of your home directory on the remote machine (assuming that you have a remote machine called `node2`), you can proceed to the next step. Do you have the same home directory on the master and remote machines and does it have the same access path? To check this, `cd` to some arbitrary subdirectory and type:

```
node node2 ls
```

where `node` is a command provided with ATAT and node2 is the name of the remote machine. If you want to use `ssh` instead, type

```
node -s node2 ls
```

This should print the content of the *current* directory on the master machine (not your home directory). If you get an error message or if you get the content of another directory, you will need to check if the following works. Make sure you are in a directory that does not contain too many files. If you want to use `rsh`, type:

```
node -r node2 ls
```

while if you want to use `ssh`

```
node -s -r node2 ls
```

In either case, you should get the content of the current directory before continuing on. The `node -r` command works by copying the content of the current directory on the remote machine in a temporary directory. Once the command has terminated, the new content is copied back and the remote temporary directory is deleted.

We are now ready to automate the calculations. Type

```
chl
```

and, as indicated on the screen, open the file `~/.machines.rc` with a text editor. This file contains numerous comment lines explaining the format of the file and a few examples.

The commands in the first column (before the `+`) must print a single number indicating the load of the machine. It is a good check to copy and paste each of these command, one at the time, into a shell window to see if the output is a single number. In order to extract a single number out of a complicated output, the command `getvalue` is provided. It extracts the single number following the token given as an argument. The first entry, with the `none` keyword after the `+` indicates the threshold load above which a machine is considered too busy to be usable. Note that the load checking commands may quite elaborate if, for instance, you need to "rescale" the load of some machines because they have a different reporting scheme or if you want to tweak the priority given to each machine.

The second column (after the `+`) give the command prefix needed to run on each remote machine. These prefix will usually consist of the command `node` described above. It is very important that the command prefix be such that the current directory in the remote machine when the command is run is the same as on the local machine. The best way to test that is to try the prefix in front of the `ls` command and verify that what is printed is indeed the content of the current local directory.

Once you are done with entering the information for each of your machines (you can also enter only a few and come back later to add the remaining ones), make sure to comment out the examples provided (placing a `#` at the beginning of the unwanted line). Do not comment out the first line which contains the `none` keyword.

Once you have edited the `~/.machines.rc` file, type `chl`. This should give a list of the load of all machines in the first column and a list of command prefix in the second. Next, try the command `minload`. It should give the command prefix that let you access the machine with the minimum load or `none` if no machine is available. To check, once again, that the command prefix are correct, type `` `minload` ls `` (make sure you use backward quotes!). This should print the content of the current directory (unless there are no machines available).

3.3 Interfacing MAPS with other first-principles codes

You need to provide a command (e.g. a shell script) called `runstruct_xxx`, where xxx is any name of your choice. This script should read, from the current directory , the file `str.out` describing the geometry of the structure and create the appropriate input files for the first-principles code. It should then execute the command(s) needed to run the code. If a multiple machine environment is used, the script should use the first argument passed to the script (`$1`) as command prefix to put in front of any command in order for them to be run on a remote machine. That is, if the first-principle code is called "myfp" the script should execute

```
$1 myfp
```

Once the first-principles code has terminated, the script should

- Create a file called `energy` containing the energy of the structure per unit cell of the structure (not the lattice) (this is what first-principles code usually give anyways).
- If the calculation fails, no `energy` file should not be created. Instead, an empty file called `error` should be created.

The above files must all reside in the current directory (from where the script was invoked). To follow the philosophy of the package, the additional input parameters (besides the structure geometry) needed by the first-principles code should be contained in a file called `xxx.wrap` located one (or two) levels up in the directory hierarchy, relative to the current directory.

As a starting point to write this script, have a look at the file `ATAT/glue/vasp/runstruct_vasp`.

Chapter 4

User guide

4.1 Introduction

First-principles calculations of alloy thermodynamic properties have been successfully employed in a variety of contexts for metallic, semi-conductor and ceramic systems, including the computation of: composition-temperature phase diagrams, thermodynamic properties of stable and metastable phases, short-range order in solid solutions, thermodynamic properties of planar defects (including surfaces or antiphase and interphase boundaries), and the morphology of precipitate microstructures [1–7].

Although the formalism that allows the calculation of thermodynamic properties from first principles has been known for decades [1–3], its practical implementation remains tedious. These practical issues limit the accuracy researchers are able to obtain without spending an unreasonable amount of their time writing input files for various computer codes, monitoring their execution and processing their output. These practical difficulties also limit the community of researchers that use these methods solely to those that possess the necessary expertise to carry out such calculations.

The Alloy Theoretic Automated Toolkit (ATAT) [8] drastically simplifies the practical use of these methods by implementing decision rules based on formal statistical analysis that free the researchers from a constant monitoring during the calculation process and automatically "glues" together the input and the output of various codes, in order to provide a high-level interface to the calculation of thermodynamic properties from first principles. In order the make this powerful toolkit available to the wide community of researchers who could benefit from it, this article present a concise user guide to this toolkit.

4.2 Theoretical Background

While there exist numerous methodologies that enable the calculation of thermodynamic properties from first principles, we will focus on the following two-step approach (see Figure 4.1). First, a compact representation of the energetics of an alloy, known as the cluster expansion [1–3, 9], is constructed using first-principles calculations of the formation energies of various atomic arrangements. Second, the cluster expansion is used as a Hamiltonian for Monte Carlo simulations [10–13] that can provide the thermodynamic properties of interest, such as the free energy of a phase or short-range-order parameters as a function of temperature and concentration. This two-step approach is essential, because the calculation of thermodynamic quantities through Monte Carlo involves averaging the property of interest over many different atomic configurations and it would be infeasible to calculate the energy of each of these configurations from first principles. The cluster expansion enables the prediction of the energy of any configuration from the knowledge of the energies of a small number of configurations (typically between 30 and 50), thus making the procedure amenable to the use of first-principles methods.

Formally, the cluster expansion is defined by first assigning occupation variables σ_i to each site i of the *parent lattice*, which is defined as the set of all the atomic sites that can be occupied by one of a few possible atomic species. In the common case of a binary alloy system, the occupation variables σ_i take the value -1 or $+1$ depending on the type of atom occupying the site. A particular arrangement of these "spins" on

the parent lattice is called a *configuration* and can be represented by a vector σ containing the value of the occupation variable for each site in the parent lattice. Although we focus here on the case of binary alloys, this framework can be extended to arbitrary multicomponent alloys (the appropriate formalism is presented in [9]).

The cluster expansion then parametrizes the energy (per atom) of the alloy as a polynomial in the occupation variables:

$$E(\sigma) = \sum_{\alpha} \mathbf{m}_{\alpha} \mathbf{J}_{\alpha} \left\langle \prod_{\mathbf{i} \in \alpha'} \sigma_{\mathbf{i}} \right\rangle \tag{4.1}$$

where α is a cluster (a set of sites i). The sum is taken over all clusters α that are not equivalent by a symmetry operation of the space group of the parent lattice, while the average is taken over all clusters α' that are equivalent to α by symmetry. The coefficients J_α in this expansion embody the information regarding the energetics of the alloy and are called the effective cluster interaction (ECI). The *multiplicities* m_α indicate the number of clusters that are equivalent by symmetry to α (divided by the number of lattice sites).

It can be shown that when *all* clusters α are considered in the sum, the cluster expansion is able to represent any function $E(\sigma)$ of configuration σ by an appropriate selection of the values of J_α. However, the real advantage of the cluster expansion is that, in practice, it is found to converge rapidly. An accuracy that is sufficient for phase diagram calculations can be achieved by keeping only clusters α that are relatively compact (*e.g.* short-range pairs or small triplets). The unknown parameters of the cluster expansion (the ECI) can then be determined by fitting them to the energy of a relatively small number of configurations obtained through first-principles computations. This approach is known as the Structure Inversion Method (SIM) or the Collony-Williams [14] method.

The cluster expansion thus presents an extremely concise and practical way to model the configurational dependence of an alloy's energy. A typical well-converged cluster expansion of the energy of an alloy consists of about 10 to 20 ECI and necessitates the calculation of the energy of around 30 to 50 ordered structures (see, for instance, [15–17]). Once the cluster expansion has been constructed, the energy of any configuration can be calculated using Equation 4.1 at a very small computational cost. This enables the use of various statistical mechanical techniques such as Monte Carlo simulations [11], the low-temperature expansion (LTE) [1, 18], the high-temperature expansion (HTE) [1], or the cluster variation method (CVM) [1, 19] to calculate thermodynamic properties and phase diagrams. The `atat` software implements Monte Carlo simulations, the LTE and the HTE.

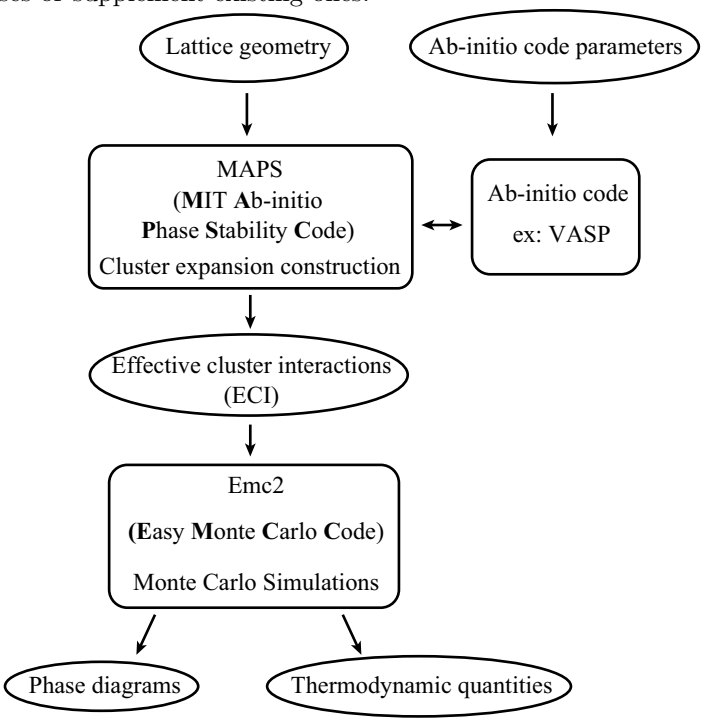
Paralleling the two-step approach described in the previous section, `atat` consists of two main computer programs (see Figure 4.1). The cluster expansion construction is performed by the MIT *Ab initio* Phase Stability (MAPS) code [20], while the Monte Carlo simulations are driven by the Easy Monte Carlo Code (EMC2), developed at Northwestern University [21]. Each of these codes will be discussed in turn.

While the present user guide describes how the `atat` software can be used to carry out all the steps necessary for the calculation of thermodynamic properties from first principles, it must be emphasized that each part of the toolkit can be used as a stand-alone code. For instance, many users may have access to an existing cluster expansion obtained through the SIM or other popular methods, such as concentration-wave-based methods (see, for instance, [1, 22, 23]). It is then staightforward to setup the appropriate input files to run the `emc2` Monte Carlo code. Alternatively, after obtaining a cluster expansion using the `maps` code, users could choose to calculate thermodynamic properties with the cluster variation method (CVM) [1, 19], as implemented in the `IMR-CVM` code [24]. The modularity of the toolkit actually extends below the level of the `maps` and `emc2` codes — many of the subroutines underlying these codes can be accessed through stand-alone utilities [8].

4.3 Cluster expansion construction using the MAPS code

The `maps` code implements the so-called Structure Inversion Method (SIM), also known as the Connolly-Williams method [14]. While the algorithms underlying the `maps` code are described in [20], the present section focuses on its practical use.

Figure 4.1: Methodology implemented in `atat` for the computation of thermodynamic properties from first principles. The automated construction of the cluster expansion is performed by the `maps` code. Whenever needed, `maps` requests the calculation of the formation energy of various atomic configurations by a first-principles code (such as `vasp`). The necessary input files are created and the resulting output files are parsed without requiring user intervention. The output of `maps` is a set of effective cluster interactions that define a computationally efficient Hamiltonian that can be used to perform Monte Carlo simulations with the `emc2` code. These simulations provide thermodynamic properties and phase diagrams that can be used to create thermodynamic databases or supplement existing ones.



4.3.1 Input files

The `maps` code needs two input files: one that specifies the geometry of the parent lattice (`lat.in`) and one that provides the parameters of the first-principles calculations (`xxxx.wrap`, where `xxxx` is the name of the first-principles code used). The clear separation between the thermodynamic and first-principles calculations is a distinguishing feature of `atat` that enables the package to be easily interfaced with any first-principles code. Table 4.1 gives two annotated examples of a lattice geometry input file. The package includes ready-made lattice files for the common lattice types (*e.g.* bcc, fcc, hcp). It also includes an utility that automatically constructs multiple lattice geometry input files for common lattices. For instance,

```
makelat Al,Ti fcc,bcc,hcp
```

creates 3 subdirectories containing the appropriate input files for each specified lattice.

The first-principles input file is usually less than 10 lines long, thanks to the dramatic improvements in the user-friendliness of most modern first-principles codes. For instance, in the case of the widely used VASP code [25, 26], a typical input file is given in Table 4.2. Examples of such input files are provided with the package. Note that `atat` contains a utility that enables the automatic construction of k-point meshes from a single parameter defining the desired target k-point density, the number of k-point per reciprocal atom (`KPPRA`).

4.3.2 Running the code

The `maps` code is started using the command

Table 4.1: Examples of lattice geometry input file `lat.in`. Typically, the coordinate system entry is used to define the conventional unit cell so that all other entries can be specified in the normalized coordinates that are the most natural for the symmetry of the lattice. The input lattice parameters do not need to be exact, as the first-principles code will optimize them.

Example 1: hcp Ti-Al system

`3.1 3.1 5.062 90 90 120`	(Coordinate system: a b c α β γ notation)
`1 0 0`	(Primitive unit cell: one vector per line
`0 1 0`	expressed in multiples of the above coordinate
`0 0 1`	system vectors)
`0 0 0 Al,Ti`	(Atoms in the lattice)
`0.6666666 0.3333333 0.5 Al,Ti`	

Example 2: rocksalt CaO-MgO pseudobinary system

`4.1 4.1 4.1 90 90 90`	
`0 0.5 0.5`	
`0.5 0 0.5`	
`0.5 0.5 0`	
`0 0 0 Ca,Mg`	("Active" atoms in the lattice)
`0.5 0.5 0.5 O`	("spectator" ion)

Table 4.2: Examples of first-principles code input file (example given for the `vasp` code). It is especially important to verify that the `KPPRA` parameter is set sufficiently large for the system under study.

`[INCAR]`	
`PREC = high`	
`ENMAX = 200`	
`ISMEAR = -1`	
`SIGMA = 0.1`	
`NSW=41`	
`IBRION = 2`	
`ISIF = 3`	(See `vasp` manual for a description of the above 6 parameters.)
`KPPRA = 1000`	(Sets the k-point density (K Point Per Reciprocal Atom))
`DOSTATIC`	(Performs a "static run" — see `vasp` manual)

```
maps -d &
```

where the option `-d` indicates that all default values of the input parameters should be used, which is what most users will ever need. (The optional parameters can be displayed by typing `maps` by itself and further help is available via the command `maps -h`.) The trailing `&` character cause the command to execute in "background" mode. In this fashion, `maps` can continuously be on the lookout, responding to various "signals", while the user performs other tasks. (The ongoing discussion assumes that the code is run under a UNIX environment within a shell such as `sh`, `csh`, `tcsh` or `bash`.)

The process of constructing a cluster expansion from first-principles calculations can be summarized as follows.

1. Determine the parameters of the first-principles code that provide the desired accuracy.
2. Let `maps` refine the cluster expansion.
3. Decide when the cluster expansion is sufficiently accurate.

Typically, one calibrates the accuracy of the first-principles calculations using the "pure"[1] structures of the alloy system of interest. To generate the two "pure" structures, type

```
touch ready
```

This creates a file called `ready` which tells `maps` that you are ready to calculate the energy of a structure. Within 10 seconds, `maps` replies with

```
Finding best structure...

done!
```

`maps` has just created a directory called `0` and, within it, a file called `str.out` that contains the geometry of one of the two "pure" structures. If you type `touch ready` once more, the other "pure" structure is written to `1/str.out`. You now need to launch the first-principles code to calculate the energy of each structure. Type

```
cd 0
```

to go into the directory of the first structure. Assuming that your first-principles code is called `xxxx`, type

```
runstruct_xxxx &
```

After this command has successfully terminated, display the energy of that structure and go back to the initial directory

```
cat energy

cd ..
```

and edit the file defining the first-principles code parameters

```
emacs xxxx.wrap &
```

so that the precision of the calculation is increased (*e.g.* increase the k-point density or the cut-off of the plane-wave energy). Then you rerun the calculations to check by how much the calculated energy has changed:

```
cd 0

runstruct_xxxx &

cat energy (After the calculations are completed)

cd ..
```

This process is repeated until the user is satisfied with the precision of the calculation (that is, if the energy has become insensitive to changes in the input parameters within the desired accuracy).[2] A similar study should also be performed for the other "pure" structure (labeled structure `1`) and, if one is really concerned with precision, for a few structures with intermediate concentrations.

Once the appropriate *ab initio* code parameters have been determined, the fully automated process can begin. From within the directory where `maps` was started, type

```
pollmach runstruct_xxxx
```

to start the job manager that will monitor the load on your local network of workstations and ask `maps` to generate new structures (*i.e.* atomic arrangements) whenever a processor becomes available. Note that the first time the command is run, instructions will appear on screen that explain how to configure the job dispatching system in accordance to your local computing environment. Once this configuration is complete, the above command should be invoked in the background by appending a "`&`" to it.

[1] In the case of pseudobinary alloys with spectator ions (*e.g.* the MgO-CaO system), the "pure" structures would correspond to the structures where the sublattice of interest is entirely filled with a single type of atom.

[2] A key number to keep in mind is that an error of 25meV corresponds to 300K on a temperature scale.

4.3.3 Output of MAPS

While the calculations are running, you can check on the status of the best cluster expansion obtained so far. The file `log.out` contains a brief description of the status of the calculations, such as the accuracy of the cluster expansion and various warning messages. Most of the messages pertains to the accurate prediction of the so-called ground states of the alloy system. The ground states, which are the structures that have the lowest energy for each given concentration, are extremely important to predict accurately because they determine which phases will appear on the phase diagram. The four possible messages are described below.

- `Not enough known energies to fit CE.` Before displaying any results, `maps` waits until enough structural energies are known to fit a minimal cluster expansion containing only nearest-neighbor pair interactions and test its accuracy. Thus, the first cluster expansion is typically constructed after at least 4 structural energies have been computed (this number may vary as a function of the symmetry of the lattice).
- `Among structures of known energy, true ground states differ from fitted ground states.` The current cluster expansion incorrectly predicts which structures have the lowest energy for given concentrations, among structures whose first-principles energy is known. The code has built-in checks to avoid this. However, in rare instances, it may be mathematically impossible to satisfy all the constraints that the code imposes for a cluster expansion to be acceptable. This problem becomes increasingly unlikely as the number of calculated structural energies increases, so the user should just wait until the problem fixes itself.
- `Among structures of known energy, true and predicted ground states agree.` Opposite of the previous message. When this message is displayed, `maps` also displays either one of the following two messages.
- `New ground states of volume less or equal to` n `predicted, see predstr.out.` This indicates that the cluster expansion predicts that, at some concentration, there exist other structures that should have an energy even lower than the one of the structures whose energy has been calculated from first principles. In this case, `maps` will investigate the matter by generating those structures and requesting that their energy be calculated. Once again, the user should just wait for the problem to fix itself. The predicted ground states are flagged by a `g` in the `predstr.out` file, so that you can display their energy by typing

  ```
  grep g predstr.out
  ```

- `No other ground states of` n `atoms/unit cell or less exist.` The energies of all ground states predicted by the cluster expansion have been confirmed by first-principles calculations. Because it can be computationally intensive to perform a full ground state search when interactions extend beyond the nearest-neighbor shell [1], `maps` uses a search algorithm that merely enumerates every possible structures having n atoms or less per unit cell and uses the cluster expansion to predict their energies. The upper limit n increases automatically as calculations progress.

The `log.out` file also contains two other pieces of information:

- `Concentration range used for ground state checking:` [a,b] This displays the user-selected range of concentration over which ground state checking is performed (which can be specified as a command-line option of the `maps` command: `-c0=`a `-c1=`b). It may be useful to relax the constraints that ground states be correctly reproduced over the whole concentration range when it is known that other parent lattices are stable in some concentration range. In this fashion, the code can focus on providing a higher accuracy in the concentration range where the user needs it.
- `Crossvalidation score:` s. This provides the predictive power of the cluster expansion. It is analogous to the root mean square error, except that it is specifically designed to estimate the error made in predicting the energy for structures not included in the least-squares fit [20]. It is defined as

$$CV = \left(\frac{1}{n} \sum_{i=1}^{n} \left(E_i - \hat{E}_{(i)} \right)^2 \right)^{1/2}$$

where E_i is the calculated energy of structure i, while $\hat{E}_{(i)}$ is the predicted value of the energy of structure i obtained from a least-squares fit to the $(n-1)$ other structural energies.

The `maps` code also outputs quantitative data in various output files. The simplest way to analyze this data is by typing

```
mapsrep
```

As illustrated in Figure 4.2, this command displays, in turn

- The `log.out` file described earlier.
- The formation energy of all structures whose energy is known from first-principles calculations, as well as the predicted energy of all structures `maps` has in memory. The convex hull of the ground states among structures of known energy is overlaid while the new predicted ground states (if any) are marked by an "×". (Note that this ground state line is only meaningful if the `log.out` file contains "`Among structures of known energy, true and predicted ground states agree.`")
- The formation energy of all structures calculated from first principles and associated ground state line.
- A plot of the magnitude of the Effective Cluster Interactions (ECI) as a function of the diameter of their associated cluster (defined as the maximum distance between any two sites in the cluster). Pairs, triplets, etc. are plotted consecutively. This plot is useful to assess the convergence of the cluster expansion. When the magnitude of the ECI for the larger clusters has clearly decayed to a negligible value (relative to the nearest-neighbor pair ECI), this is indicative of a well-converged cluster expansion.
- A plot of the residuals of the fit (*i.e.* the fitting error) for each structure. This information is useful to locate potential problems in the first-principles calculations. Indeed, when first-principles calculations exhibit numerical problems, this typically results in calculated energies that are poorly reproduced by the cluster expansion.

When the user is satisfied with the results (which are constantly updated), `maps` can be stopped by creating a file called `stop` in the current directory using the command:

```
touch stop
```

while the job dispatching system can be stopped by typing:

```
touch stoppoll
```

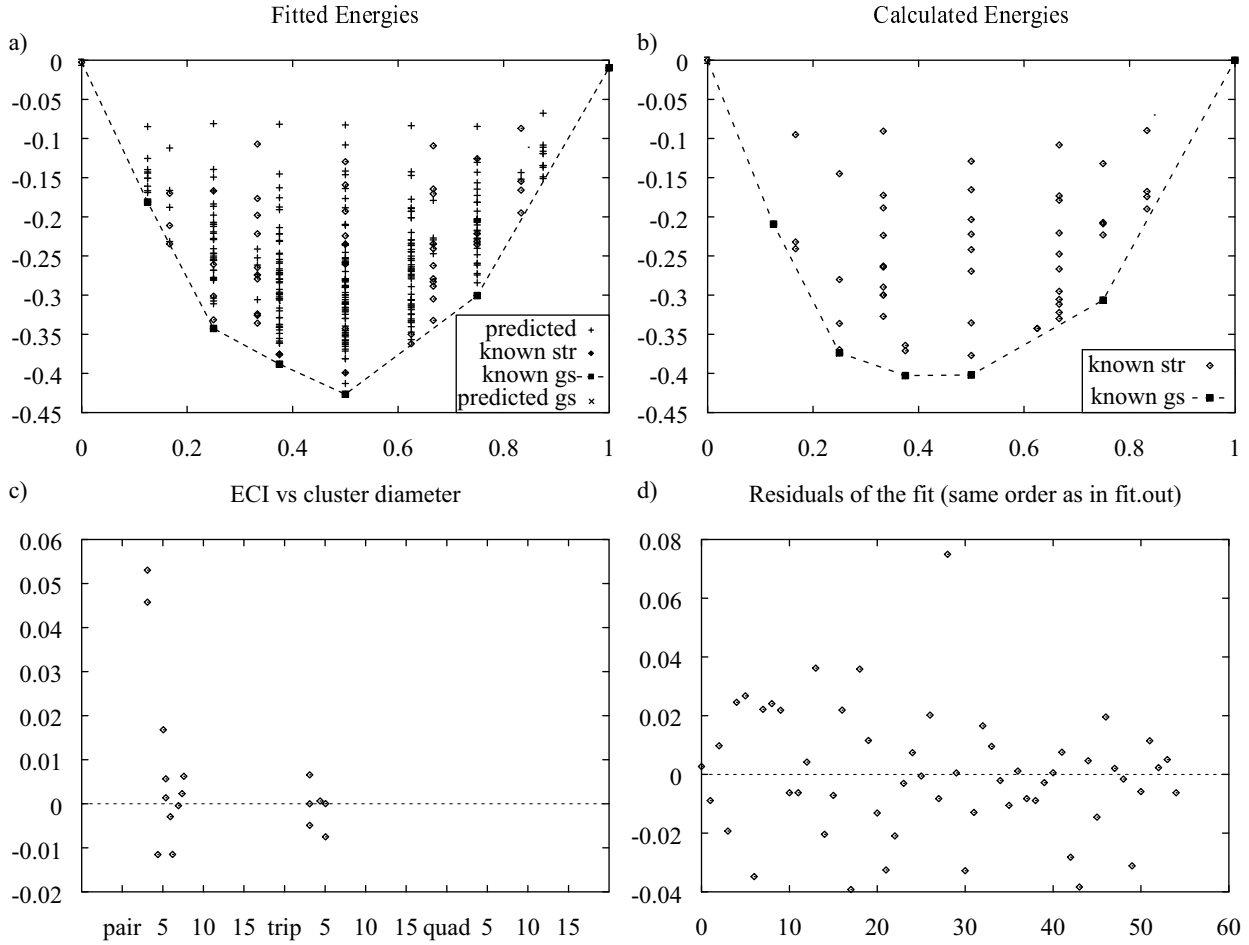
A cluster expansion can be considered satisfactory when

1. All ground states are correctly reproduced and no new ground states are predicted. (The `log.out` file would then indicate that `Among structures of known energy, true and predicted ground states agree. No other ground states of` n `atoms/unit cell or less exist.`)
2. The crossvalidation score, as given in the `log.out` file, is small (typically less than 0.025 eV).
3. Optionally, it is instructive to verify that the magnitude of the ECI decays as a function of the diameter of the corresponding cluster and as a function of the number of sites it contains.

4.4 Monte Carlo simulations

The `emc2` code implements semi-grand canonical Monte Carlo simulations [10–13], where the total number of atoms is kept fixed, while the concentration is allowed to adapt to an externally imposed difference in the chemical potential of the two types of atoms. The chemical potential difference will be simply referred to as the "chemical potential" in what follows. This ensemble offers the advantage that, for any imposed

Figure 4.2: Output of the `maps` Code, as reported by the `mapsrep` command. a) Energies predicted from the cluster expansion as a function of composition for each structure generated. "`known str`" denotes structures whose energy has been calculated from first principles. "`known gs`" indicate the ground states that have so far been confirmed by first-principles calculations and the dashed line outlines the convex hull of the ground states, which serves as a threshold to detect other candidate ground states. "`predicted`" denotes structures whose energy has *not yet* been calculated from first principles. "`predicted gs`" are structures that are predicted by the cluster expansion to be ground states, although this prediction has not yet been confirmed by first-principles calculations. b) Energies calculated from first principles. "`known str`" and "`known gs`" are as in a), except that the energy calculated from first principles is reported. c) Effective Cluster Interaction (ECI) as a function of the diameter of the associated cluster and as a function of the number of sites in the cluster (*i.e.* pair, triplet, etc.). d) Residuals of the fit, that is, the difference between the first-principles energies and the energies predicted from the cluster expansion. (The abscissa refers to the line number within the output file `fit.out` listing all the structures with known energies.)



chemical potential, the equilibrium state of the system is a single phase equilibrium, free of interfaces.[3] It also simplifies the process of calculating free energies through thermodynamic integration. While a detailed description of the algorithm underlying this code can be found in [21], the current section focuses on the practical usage of the code.

[3]If the simulation cell is commensurate with the unit cell of the phase under study, a requirement that the code automatically ensures.

4.4.1 General input parameters

The Monte Carlo code needs the following files as an input

1. A lattice geometry file (`lat.in`), which is the same as the input for `maps` (see Table 4.1).
2. Files providing the cluster expansion (the clusters used are listed in the `clusters.out` file while the corresponding ECI are listed in the `eci.out` file.). These files are automatically generated by `maps`, although users can supply their own cluster expansion, if desired. A description of the format of these files is available by typing `maps -h`.
3. A list of ground states (`gs_str.out`), which merely provide convenient starting configurations for the simulations. `maps` also automatically creates this file.

The parameters controlling the simulation are specified as command-line options. The first input parameter(s) needed by the code are the phase(s) whose thermodynamic properties are to be determined. There are two ways to invoke the Monte Carlo simulation code. When the command `emc2` is used, a single Monte Carlo simulation is run to allow the calculation of thermodynamic properties of a single phase for the whole region of chemical potential and temperature where that phase is stable. The phase of interest is specified by a command-line option of the form

`-gs=`n,

where n is a number between -1 and $G-1$ (inclusive), where G is the number of ground states. The value -1 indicates the disordered phase while values ranging from 0 to $G-1$ indicate the phases associated with each ground states (0 denoting the ground state with the smallest composition). When the command `phb` is used, two Monte Carlo simulations are run simultaneously to enable the determination of the temperature-composition phase boundary associated with a given two-phase equilibrium. The two phases are specified by

`-gs1=`n_1 `-gs2=`n_2.

It is possible to compute a two-phase equilibrium between phases defined on a different parent lattice. In this case, the user needs to specify the directories where the cluster expansions of each lattice resides using the options of the form

`-d1=`*directory 1* `-d2=`*directory 2*

The accuracy of the thermodynamic properties obtained from Monte-Carlo simulations is determined by two parameters: The size of the simulation cell and the duration of the simulation.

The size of the simulation cell is specified by providing the radius r of a sphere through the command-line option

`-er=`r.

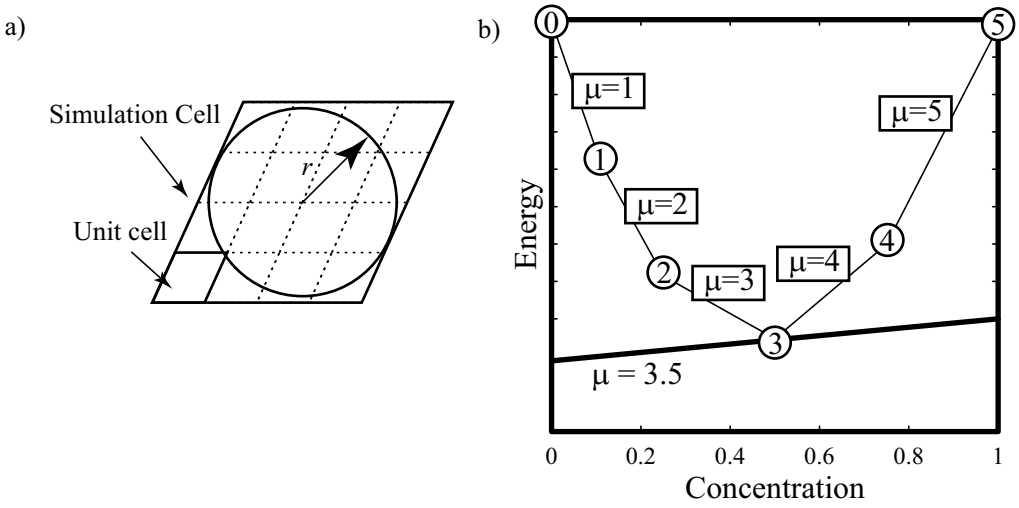
As illustrated in Figure 4.3a, the simulation cell size will be the smallest supercell that both contains that sphere and that is commensurate with the unit cell of the ground state of interest. This way of specifying the simulation cell size ensures that the system size is comparable along every direction, regardless of the crystal structure of the ground state of interest. It also frees the user from manually checking the complicated requirement of commensurability. It is important that the user check that the simulation cell size is sufficiently large for the thermodynamic properties of interest to be close to their infinite-system-size limiting value. This can be done by gradually increasing the system size until the calculated quantities become insensitive to the further increases in system size, within the desired accuracy.

The duration of the simulations is automatically determined by the code from a user-specified target precision on the atomic composition of the phase, indicated by a command-line option of the form

`-dx=`Δx.

Alternatively, the user may also manually set the number n_{eq} of Monte Carlo steps the system is allowed to equilibrate before thermodynamic averages are computed over a certain number n_{avg} of Monte Carlo steps using the options

Figure 4.3: Definitions of the quantities used to specify a) the simulation cell size and b) the chemical potential.



`-eq=`n_{eq} `-n=`n_{avg}.

The Monte Carlo code also needs additional parameters that specify which portion of a phase's free energy surface needs to be computed. With `emc2`, the range of temperatures to be scanned are specified in either one of the following two ways:

`-T0=`T_0 `-T1=`T_1 `-dT=`ΔT (for steps in direct temperature)

or `-T0=`T_0 `-T1=`T_1 `-db=`$\Delta(1/T)$ (for steps in reciprocal temperature).

The temperature steps in reciprocal temperature ($\Delta(1/T)$) can be useful when calculations are started from infinite temperatures down to a finite temperature. The `-T1` and `-dT` (or `-db`) options can be omitted if calculations at a single temperature are desired. Since the program automatically stops when a phase transition is detected, it is not necessary to know in advance the temperature range of stability of the phase. The user only needs to ensure that the initial temperature lies within the region of stability of the phase of interest. An obvious starting point is $T_0 = 0$, since the ground state is then stable, by definition. With the `phb` code, the syntax is

`-T=`T `-dT=`ΔT

If the `-T` option is omitted, calculations start at absolute zero.[4] The energy and temperature units used are set by specifying the Boltzman's constant with the command-line option

`-k=`k_B.

A value of 8.617×10^{-5} corresponds to energies in eV and temperatures in Kelvin.

With `emc2`, the range of chemical potentials to be scanned needs to be specified. Once again, only the starting point really matters, because the code will stop when a phase transition is reached. By default, chemical potentials are given in a dimensionless form, so as to facilitate the link between the value of the chemical potential and the phase that it stabilizes. For instance, a chemical potential equal to 3.0 is such that it would stabilize a two phase equilibrium between phase number 2 and phase number 3 at absolute zero (see Figure 4.3b). A chemical potential between 3.0 and 4.0 stabilizes phase number 3 at absolute zero. While these ranges of stability are no longer exact at finite temperature, this dimensionless chemical potential still provides easy-to-interpret input parameters. The syntax is

[4]Temperature steps in reciprocal temperature are not needed, because a two-phase equilibrium never extents up to infinite temperature.

`-mu0=`μ_0 `-mu1=`μ_1 `-dmu=`$\Delta\mu$

where $\Delta\mu$ is the chemical potential step between each new simulation. Chemical potentials can also be entered in absolute value (say in eV, if the energies are in eV) by specifying the `-abs` option. Note that the output files always give the absolute chemical potentials, so that thermodynamic quantities can be computed from them. With `phb`, the initial chemical potential is optional when starting from absolute zero because the code can determine the required value from the ground state energies. It can nevertheless be specified (in absolute value) with the `-mu=`μ option, if a finite temperature starting point is desired.

A list of the command line options of either the `emc2` or `phb` codes can be displayed by simply typing either command by itself. More detailed help is displayed using the `-h` option.

4.4.2 Examples

We now give simple examples of the usage of these commands. Consider the calculation of the free energy of the phase associated with ground state number 1 as a function of concentration and temperature. Then, the required commands could, for instance, be

```
emc2 -gs=1 -mu0=1.5 -mu1=0.5 -dmu=0.04 -T0=300 -T1=5000 -dT=50 -k=8.617e-5 -dx=1e-3 -er=50
-innerT -o=mc10.out
```

```
emc2 -gs=1 -mu0=1.5 -mu1=2.5 -dmu=0.04 -T0=300 -T1=5000 -dT=50 -k=8.617e-5 -dx=1e-3 -er=50
-innerT -o=mc12.out
```

(The only difference in the two command lines is the value of `-mu1` and the output file name, specified by the `-o` option.) These commands separately compute the two "halfs" of the free energy surface, corresponding to the values of the chemical potential below and above the "middle" value of 1.5 which stabilizes ground state 1 at absolute zero. This natural separation allows you to run each half calculation on a separate processor and obtain the results in half the time. The values of `-dmu`, `-dT`, `-dx` and `-er` given here are typical values. The user should ensure that these values are such that the results are converged. Note that, thanks to the way these precisions parameters are input, if satisfying values have been found for one simulation, the same values will provide a comparable accuracy for other simulations of the same system. The option `-innerT` indicates that the inner loop of the sequence of simulations scans the temperature axis while the outer loop scans the chemical potential. In this fashion, the point of highest temperature in the region of stability of the phase will be known early during the calculations. If the user is more interested in obtaining solubility limits early on, this option can be omitted and the inner loop with scan the chemical potential axis. In either cases, the code exits the inner loop (and the outer loop, if appropriate) when it encounters a phase transition.

The `emc2` code thus enables the automated calculation of the whole free energy surface of a given phase, as illustrated in Figure 4.4a. Such free energy surfaces can be used as an input to construct thermodynamic databases or supplement existing ones. To facilitate this process, a utility that converts the output of `emc2` into input files for the fitting module of ThermoCalc is provided.

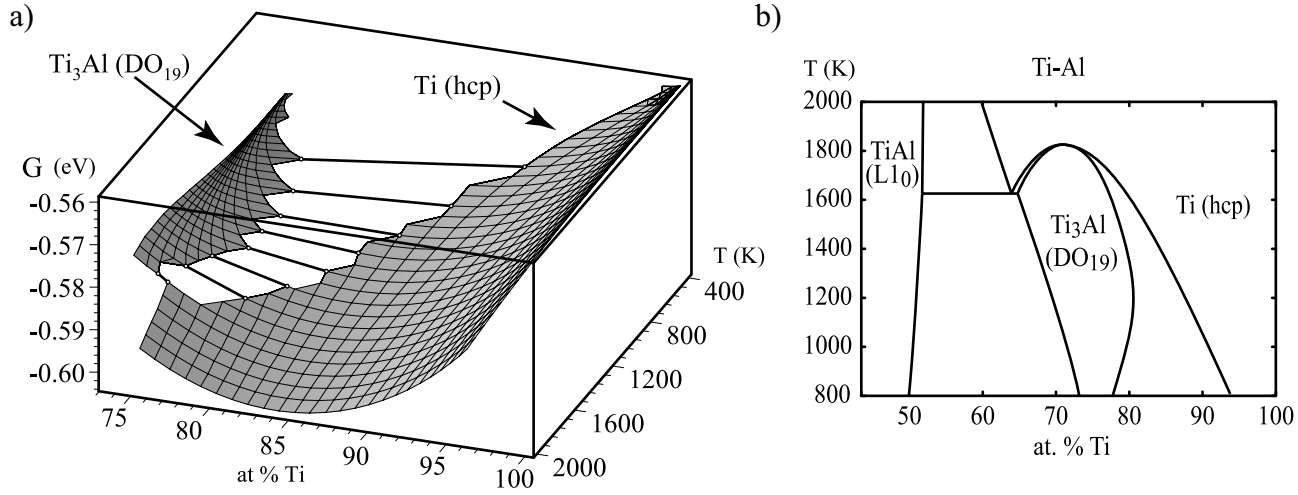
While the above examples focus on the calculation of a phase's thermodynamic properties over its whole region of stability, one may be interested in directly computing the temperature-composition phase boundary without first constructing a full free energy surface. To accomplish this task, a typical command-line for the `phb` program would be

```
phb -gs1=0 -gs2=1 -dT=25 -dx=1e-3 -er=50 -k=8.617e-5 -ltep=5e-3 -o=ph01.out
```

This command computes the two phase equilibrium between phase 0 and phase 1, starting at absolute zero and incrementing temperature in steps of 25 K. (The `-ltep` option indicates that a Low Temperature Expansion (LTE) should be used instead of Monte Carlo simulation whenever its precision is better than 5×10^{-3} eV.) The output file `ph01.out` contains the temperature-composition phase boundary of interest, as well as the chemical potential stabilizing the two-phase equilibrium as a function of temperature. This output can be used to generate phase diagrams, as illustrated in Figure 4.4b.

The program automatically terminates when the "end" of the two-phase equilibrium has been reached. If the two-phase equilibrium disappears because of the appearance of a third phase, two new two-phase equilibria have to be separately calculated. To do so, one uses the final temperature T and chemical potential μ given in the output file as a starting point for two new `phb` runs:

Figure 4.4: Output of Monte Carlo codes. a) The `emc2` provides free energy surfaces as a function of temperature T and composition x. (For clarity, the common tangent construction (thick lines) is drawn over the calculated free energy.) b) The `phb` command generates temperature-composition phase diagrams. The calculational details underlying these results can be found in [20, 21]




```
phb -T=T -mu=μ -gs1=0 -gs2=-1 -dT=25 -dx=1e-3 -er=50 -k=8.617e-5 -o=ph0d.out
```

```
phb -T=T -mu=μ -gs1=-1 -gs2=1 -dT=25 -dx=1e-3 -er=50 -k=8.617e-5 -o=phd1.out
```

In the above example, it is assumed that the new phase appearing is the disordered phase (indicated by the number -1), which will usually be the case. Of course, it is also possible that a given two-phase equilibrium terminates because one of the two phases disappears. In this case, only one new calculation needs to be started, as in the following example:

```
phb -T=T-mu=μ -gs1=0 -gs2=2 -dT=25 -dx=1e-3 -er=50 -k=8.617e-5 -o=phd1.out
```

Note that phase 1 has been replaced by phase 2. Finally, it is also possible that the two-phase equilibrium terminates because the concentration of each phase converges to the same value, a situation which requires no further calculations. The user can easily distinguish these three cases by merely comparing the final composition of each phase.

4.4.3 Interpreting the output files

The output file of `emc2` reports the value of all calculated thermodynamic functions for each value of temperature and chemical potential scanned. The quantities reported include

- Statistical averages over Monte Carlo steps, such as energy, concentration, short-range and long-range order parameters.[5]
- Integrated statistical averages, such as the Gibbs free energy G or the semi-grand-canonical potential $\phi = G - \mu x$.
- The result of common approximations, namely, the low temperature expansion (LTE) [1, 18, 27], the mean-field (MF) approximation and the high temperature expansion (HTE) (see, for instance, [1]).

While quantities obtained from statistical averages over Monte Carlo steps are valid for all temperatures and chemical potentials, caution must be exercised when interpreting the result of the various approximations or when looking at the integrated quantities. The LTE, MF and HTE approximations are only accurate in a limited range of temperature and it is the responsibility of the user to assess this range of validity.

[5] For efficiency reasons, the long range order parameters are only calculated when starting from an ordered phase.

Also, the free energy or the semi-grand-canonical potential are obtained from thermodynamic integration and are thus only valid if the starting point of the integration is chosen appropriately. By default, the low temperature expansion value is used as a starting point whenever the phase of interest is a ground state, while the high temperature expansion is used when the phase of interest is the disordered state. Hence, to obtain absolute values of the semi-grand-canonical potential, one must ensure that the calculations are started at a sufficiently low temperature (or sufficiently high temperature, in the case of the disordered phase). This can be checked by comparing the Monte Carlo estimates with the LTE (or HTE) estimates and verifying that they agree for the first few steps of the thermodynamic integration. A user-specified starting point for ϕ (*e.g.* obtained from an earlier Monte Carlo simulation) can be indicated using the option

`-phi0=`ϕ

Note that, unlike `emc2`, the `phb` code automatically makes use of the low temperature expansion whenever it is sufficiently accurate in order to save a considerable amount of computational time.

By default, the code reports the thermodynamic quantities associated with the semi-grand-canonical ensemble, such as the semi-grand-canonical potential ϕ. The command-line option `-can`, instructs the code to add μx to all appropriate thermodynamic quantities, so that the code outputs the more commonly used canonical quantities, such as the Gibbs free energy G and the internal energy E.

4.5 Future developments

At the present time, some of the features of `atat` are only available for binary systems. The multicomponent version of `maps` is `mmaps` and the multicomponent version of `emc2` is `memc2`. The `phb` code does not have a multicomponent version. All other utilities can handle multicomponent systems.

Although the present tutorial does not discuss the topic, `atat` also implements reciprocal space cluster expansions and, in particular, the constituent strain formalism [28], see the command reference for `csfit`. `atat` can also calculate nonconfigurational contributions to the free energy, such as lattice vibrations and electronic excitations, see the command reference for `fitsvsl`, `svsl`, `fitfc` and `felec`.

4.6 Conclusion

The Alloy Theoretic Automated Toolkit (ATAT) drastically simplifies the practical implementation of the Connolly-Williams method, in combination with semi-grand-canonical Monte Carlo simulations, thus providing a high-level interface to the calculation of thermodynamic properties from first principles. This toolkit enables researcher to focus on higher-level aspects of first-principles thermodynamic calculations by encapsulating the intricate details of the calculations under an easy-to-use interface. It also makes these powerful methodologies readily available to the wide community of researchers who could benefit from it.

Bibliography

[1] F. Ducastelle, *Order and Phase Stability in Alloys* (Elsevier Science, New York, 1991).

[2] D. de Fontaine, Solid State Phys. **47**, 33 (1994).

[3] A. Zunger, in *NATO ASI on Statics and Dynamics of Alloy Phase Transformation*, edited by P. E. Turchi and A. Gonis (Plenum Press, New York, 1994), Vol. 319, p. 361.

[4] A. Zunger, MRS Bull. **22**, 20 (1997).

[5] C. Wolverton, V. Ozoliņš, and A. Zunger, J. Phys.: Condens. Matter **12**, 2749 (2000).

[6] G. Ceder, A. van der Ven, C. Marianetti, and D. Morgan, Modelling Simul. Mater Sci Eng. **8**, 311 (2000).

[7] M. Asta, V. Ozolins, and C. Woodward, JOM - Journal of the Minerals Metals & Materials Society **53**, 16 (2001).

[8] A. van de Walle, M. Asta, and G. Ceder, CALPHAD Journal **26**, 539 (2002).

[9] J. M. Sanchez, F. Ducastelle, and D. Gratias, Physica **128A**, 334 (1984).

[10] M. E. J. Newman and G. T. Barkema, *Monte Carlo Methods in Statistical Physics* (Clarendon Press, Oxford, 1999).

[11] K. Binder and D. W. Heermann, *Monte Carlo Simulation in Statistical Physics* (Springer-Verlag, New York, 1988).

[12] B. Dünweg and D. P. Landau, Phys. Rev. B **48**, 14182 (1993).

[13] M. Laradji, D. P. Landau, and B. Dünweg, Phys. Rev. B **51**, 4894 (1995).

[14] J. W. Connolly and A. R. Williams, Phys. Rev. B **27**, 5169 (1983).

[15] A. van der Ven *et al.*, Phys. Rev. B **58**, 2975 (1998).

[16] G. D. Garbulksy and G. Ceder, Phys. Rev. B **51**, 67 (1995).

[17] V. Ozoliņš, C. Wolverton, and A. Zunger, Phys. Rev. B **57**, 6427 (1998).

[18] A. F. Kohan, P. D. Tepesch, G. Ceder, and C. Wolverton, Comput. Mater. Sci. **9**, 389 (1998).

[19] R. Kikuchi, Phys. Rev. **81**, 988 (1951).

[20] A. van de Walle and G. Ceder, Journal of Phase Equilibria **23**, 348 (2002).

[21] A. van de Walle and M. Asta, Modelling Simul. Mater. Sci. Eng. **10**, 521 (2002).

[22] F. Ducastelle and F. Gautier, Journal of Physics F — Metal Physics **6**, 2039 (1976).

[23] P. E. A. Turchi, in *Intermetallic Compounds: Principles and Practice*, edited by J. H. Westbrook and R. L. Fleisher (John Wiley, New York, 1995), Vol. 1, p. 21.

[24] M. H. Sluiter, IMR-CVM code, 2000, http://www-lab.imr.tohoku.ac.jp/~marcel/cvm/cvm.html.

[25] G. Kresse and J. Furthmüller, Phys. Rev. B **54**, 11169 (1996).

[26] G. Kresse and J. Furthmüller, Comput. Mater. Sci. **6**, 15 (1996).

[27] C. Woodward, M. Asta, G. Kresse, and J. Hafner, Phys. Rev. B **63**, 094103 (2001).

[28] D. B. Laks, L. G. Ferreira, S. Froyen, and A. Zunger, Physical Review B **46**, 12587 (1992).

Chapter 5

Command Reference

5.1 maps

```
-> What does this program do?

It gradually constructs a increasingly more accurate cluster expansion.
A user-provided script running concurrently is responsible for notifying
maps when computer time is available. maps creates files describing
structures whose energy should be calculated. The user-provided script
sets up the runs needed to calculate the energy of these structures.
As maps becomes aware of more and more structural energies, it gradually
improves the precision of the cluster expansion, which is continously
written to an output file.
The code terminates when a stop file is created by typing, for instance,
touch stop

NOTE: Fully functional scripts are included with the package:
pollmach and runstruct_vasp.
For for more information type
  pollmach
  runstruct_vasp -h

-> Format of the input file defining the lattice (specified by the -l option)

First, the coordinate system is specified, either as
[a] [b] [c] [alpha] [beta] [gamma]
or as:
[ax] [ay] [az]
[bx] [by] [bz]
[cx] [cy] [cz]
Then the lattice vectors are listed, expressed in the coordinate system just defined:
[ua] [ub] [uc]
[va] [vb] [vc]
[wa] [wb] [wc]
Finally, atom positions and types are given, expressed in the same coordinate system
as the lattice vectors:
[atom1a] [atom1b] [atom1c] [atom1types]
[atom2a] [atom2b] [atom2c] [atom2types]
etc.
```

```
-The atom type is a comma-separated list of the atomic
 symbols of the atoms that can sit the lattice site.
-The first symbol listed is assigned a spin of -1 and the second, a spin of 1.
-When only one symbol is listed, this site is ignored for the purpose
 of calculating correlations, but not for determining symmetry.

Examples:

The fcc lattice of the Cu-Au system:
3.8 3.8 3.8 90 90 90
0   0.5 0.5
0.5 0   0.5
0.5 0.5 0
0 0 0 Cu,Au

A lattice for the Li_x Co_y Al_(1-y) O_2 system:
 0.707 0.707 6.928 90 90 120
 0.3333  0.6667 0.3333
-0.6667 -0.3333 0.3333
 0.3333 -0.3333 0.3333
 0       0      0       Li,Vac
 0.3333  0.6667 0.0833  O
 0.6667  0.3333 0.1667  Co,Al
 0       0      0.25    O
Running the above example requires the multicomponent version of maps,
called mmaps.

Optional input file: ref_energy.in

  Contains the reference energy (per site) to be subtracted to get formation energies.
  The first line is the c=0 energy, the second line is the c=1 energy.
  If this file is omitted, the energies of leftmost and rightmost structures
  (on the concentration axis) are taken.

Optional input file: nbclusters.in
  Allows the user to manually select which clusters to include in the fit.
  This file should contains:
    number of pairs to include
    number of triplets to include
    etc.
  This file can be changed while maps is running. However, you must type
   touch refresh
  to tell maps to reread it.

-> Output files

maps.log
Contains possible warnings:
  'Not enough known energies to fit CE'
  'True ground states not = fitted ground states'
  'New ground states predicted, see predstr.out'

These warning should disappear as more structural energies become available
and the following messages should be displayed:
```

```
  'Among structures of known energy, true and predicted ground states agree.'
  'No other ground states of xx atoms/unit cell or less exist.'

This file also gives the crossvalidation score of the current fit
(before the weighting is turned on in order to get the correct ground states).

fit.out
Contains the results of the fit, one structure per line and each line
has the following information:
  concentration energy fitted_energy (energy-fitted_energy) weight index
'concentration' lies between 0 and 1.
'energy' is per site (a site is a place where more than one atom type can lie)
'weight' is the weight of this structure in the fit.
'index' is the name of the directory associated with this structure.

predstr.out
Contains the predicted energy (per site) of all structures maps has in memory but
whose true energy is unknown or has been flagged with error.
Format: one structure per line, and each line has the following information:
  concentration energy predicted_energy index status
index is the structure number (or -1 if not written to disk yet).
energy is the calculated energy (or 0 if unknown).
status is either b for busy (being calculated), e for error or u for unknown
(not yet calculated). A g is appended to status if that structure is predicted to be
a ground state. To list all predicted ground states, type
grep 'g' predstr.out

gs.out
Lists the ground state energies, one structure per line and each line
has the following information:
  concentration energy fitted_energy index

gs_str.out
Lists the ground state structures, in the same format as the n/str.out files
(see below). Each structure is terminated by the word 'end' on a line by itself,
followed by a blank line.

eci.out
Lists the eci. (They have already been divided by multiplicity.)
The corresponding clusters are in clusters.out

clusters.out
For each cluster, the first line is the multiplicity, the second line is the
cluster diameter, and the third line is the number of points in the cluster.
The remaining lines are the coordinates of the points in the cluster
(in the coordinate system specified in the input file defining the lattice).
A blank line separates each cluster.

n/str.out
Same format as the lattice file, except that
-The coordinate system is always written as 3x3 matrix
-Only one atom is listed for each site.

ref_energy.out
```

```
Reference energies used to calculate formation energies.
(Usually: energy of the pure end members OR values given in
ref_energy.in if provided.)

-> Communication protocol between maps and the script driving the
   energy method code (e.g. ab initio code)
   (Only those who want to customize the code need to read this section.
   The scripts described in this section are provided with the atat
   distribution in the glue/ subdirectory.)

Unless otherwise specified all files mentioned reside in the directory where
maps was started. All paths are relative to the startup directory.

+The script should first wait for computer time to be available before creating
 a file called 'ready'.
-Upon detecting that the 'ready' file has been created,
 maps responds by creating a subdirectory 'n' (where 'n' is a number) and a file
 'n/str.out' containing a description of a structure whose energy needs to be
 calculated.
-maps creates a file called 'n/wait' to distinguish this directory
 from other ones created earlier.
-maps deletes the 'ready' file.
+Upon detecting that the 'ready' file has disappeared,
 the script should now look for the 'n/wait' file, start the calculations
 in the directory 'n' and delete file 'n/wait'.
+If anything goes wrong in the calculations, the script should create a file
 'n/error'.
+When the calculations terminate successfully, the energy per unit cell of the structure
 should be copied to the file 'n/energy'.
 (NOTE: use energy per unit cell of the structure NOT per unit cell of the lattice).
-maps continuously scans all the subdirectories 'n' for 'n/energy' or 'n/error'
 files and updates the cluster expansion accordingly.
-maps updates the cluster expansion whenever a file called 'refresh' is created
 (maps then deletes it).
-maps terminates when a 'stop' file is created.

Note that the script can ask maps to create new structure directories even before
the energy of the current structure has been found.
Note that human intervention is allowed: an 'n/error' file can be
manually created if an error is later found in a run.
Users can also manually step up all runs if they wish so, as long
as they follow the protocol.

Example of script
(portions in /* */ have to be filled in with the appropriate code):

#!/bin/csh

while (! -e stop)
  /* check machine load here */
  if ( /* load low enough */ ) then
    touch ready
    while (-e ready)
      sleep 30
```

```
    end
    cd `ls */wait | sed 's+/.*++g' | head -1`
    rm -f wait
    /* convert str.out to the native format of ab initio code */
    /* in background: run code and create either energy file or error file */
    cd ..
  endif
  sleep 180
end

-> Using maps with vasp

The script runstruct_vasp, when run from within directory 'n',
1) converts 'vasp.wrap' and 'n/str.out' into all the necessary files to run vasp,
2) runs vasp
3) extract all the information from the output files and writes in a format
   readable by maps.

An example of vasp.wrap is:
[INCAR]
PREC = high
ISMEAR = -1
SIGMA = 0.1
NSW=41
IBRION = 2
ISIF = 3
KPPRA = 1000
DOSTATIC

See ezvasp documentation for more information.

-> Importing structures into maps

MAPS continuously scans all the first-level subdirectories containing
a file called str.out and tries to map them onto superstructures of the
lattice provided. This lets you 'import' structures from another source
into MAPS. A word of caution: the imported structures must be
unrelaxed and no effort is made to rotate or scale them in order to
match the lattice (aside from space group symmetry operations).
```

5.2 emc2

```
Easy Monte Carlo Code (emc2)
by Axel van de Walle

Command line parameters:

-T0 : initial temperature
-T1 : final temperature
-dT : temperature step
-db : inverse temperature step

Temperatures are given in units of energy unless the -k option is set.
```

```
-dT and -db are mutually exclusive. Which option you use determines
whether T or 1/T are unformly spaced. The output file always shows T.

-k : Sets boltman'z constant (default k=1). This only affects how
     temperatures are converted in energies.  -k=8.617e-5 lets you enter
     temperatures in kelvins when energies are in eV.
     (You can also select this value by using the -keV option.)

-mu0 : initial chemical potential
-mu1 : final chemical potential

  By default, these are input as dimensionless values defined as
  follows: If x is integer, x is the chemical potential that
  stabilizes a two phase equilibrium between ground state x-1 and
  x. (Ground states are numbered starting at 0.) Fractional values
  interpolate linearly between these integer values.  Negative values
  or values larger than the number of ground states - 1 are allowed.
  The values of mu are found by linear extrapolation.  If the -abs
  option is specified, the value of mu are in units of energy per spin
  value (as in the output file).  (The ground states are read in from
  the file gs_str.out .)

  If there are only two ground states, the only correction performed is
  to shift mu so that mu=0 stabilizes a two-phase equilibrium between
  the two ground states.
  If there are less than 2 ground states (!), no correction is made.

-dmu : chemical potential step (expressed in the same units as mu0 and mu1).

NOTE: If mu1 is omitted, only a scan through T is performed keeping mu=mu0.
      If T1 is omitted, only scan through mu is performed keeping T=T0.

-gs : Gives the index of the ground state to use as an initial spin
      configuration (the leftmost pure element is numbered 0).  If the
      index is -1, the disordered state with an average spin of zero
      is used as the starting configuration.

-phi0 : value of the grand canonical potential at the initial point of
        the run.  If left unspecified, it is set to the grand canonical
        potential of given by either the 1-spin Low Temperature Expansion
        or the High Temperature Expansion, depending whether the initial
        state is an ordered or a disordered phase.

-er : enclosed radius. The Monte Carlo cell will the smallest possible
    supercell of unit cell of the initial configuration such that a
    sphere of that radius fits inside it.  This allows you to specify
    the system size in a structure-independent fashion.

-innerT : by default, the inner loop is over values mu while the outer
          loop is over values of T. This flag permutes that order.

-eq : number of equilibration passes. (Equilibration is performed at
      the beginning of each step of the outer loop.)
```

```
-n : number of Monte Carlo passes performed to evaluate thermodynamic
     quanties at each step of the inner loop.

-dx: instead of specifying -eq and -n, you can ask the code to equilibrate
     and run for a time such that the average concentration is accurate
     within the target precision specifiied by -dx=.

-tstat : Critical value of the test for discontinuity. The code is
         able to catch phase transformations (most of the time) when
         going from an ordered phase to another or to the disordered
         state. This involves a statistical test which a
         user-specified confidence level. The default, 3, corresponds
         to a 0.4% chance of finding a transition where there is
         none. (Refer to a standard normal table.)  -tstat=0 turns off
         this option.  Suggestion: if a phase transition is
         undetected, first try to reduce dT or dmu or increase n or decrease dx
         before toying around with this parameter.  Also: beware of
         hysteresis.

-sigdig : Number of significant digits printed. Default is 6.

-o:       Name of the output file (default: mc.out).

Tricks:

   To read parameters from a file, use:
    emc2 `cat inputfile`
   where inputfile contains the commands line options.

   To selectively display a few of the output quantities, use:
    emc2 [options] | cut -f n1,n2,n3,...
   where n1,n2,n3,... are the column number desired (see below).

Input files:

lat.in       : description of the lattice.
clusters.out : describes the clusters.
eci.out      : provides the ECI.
gs_str.out   : a list of ground states, in increasing order of concentration.

   These 4 files can be created by maps.
   See maps documentation (maps -h) for a description of the formats.

Optional input file:

teci.out     : if present, provides temperature-dependent eci (overrides eci.out)
               (Note that, even when the teci.out file is used, column 6 of the output file
                reflects only the configurational contribution to the heat capacity.)

The format this file is:
[maximum temperature: Tm]
[number of temperatures where eci are provided: nT]
[ECIs for temperature 0]
[ECIs for temperature Tm/(nT-1)]
```

```
[ECIs for temperature 2*Tm/(nT-1)]
...
[ECIs for temperature Tm]

Note that these numbers can be seperated by blanks or newlines, as desired.

Format of the output file:

Each contains, in order:

 1) T:  Temperature
 2) mu: chemical potential (derivative of the free energy wrt to average spin)
 3) E-mu*x:  Average energy (per spin)
 4) x:  Average Concentration [ranges from -1 to 1]
 5) phi: grand canonical potential
 6) E2: Variance of the energy (proportional to heat capacity)
 7) x2: Variance of the concentration (proportional to susceptibility)
 8) E_lte-mu*x_lte: calculated with a one spin Low Temperature Expansion
 9) x_lte
10) phi_lte
11) E_mf-mu*x_mf:  calculated with the Mean Field approximation
12) x_mf
13) phi_mf
14) E_hte-mu*x_hte: calculated with a High Temperature Expansion (ideal solution + polynomial in x)
15) x_hte
16) phi_hte
17) lro: Long Range Order parameter of the initial ordered phase (=0 if initial phase is disordered).
18-) corr: the average correlations associated with each cluster.

NOTE: to obtain 'canonical' rather than grand canonical quantities,
use the -g2c option. This has the effect of adding mu*x to columns
3,5,8,10,11,13,14,16.
```

5.3 phb

```
This is Monte Carlo code which automatically follows a given phase boundary.

Input files: (see maps documentation)
  lat.in         (describes the lattice)
  gs_str.out     (lists the ground states)
  eci.out        (ECI)
  clusters.out    (clusters)

You have to provide, on the command line, the following parameters:

-The two phases which are in equilibrium.
  For instance,
    -gs1=0 -gs2=1
  If there are n ground states, phases are numbered from 0 to n-1 .
  These ground states are read in from gs_str.out
  The disordered state is labelled by the number -1.

  If the two phases are on different lattices, you need to specify
```

```
  the path that give access to the files
    lat.in
    gs_str.out
    eci.out
    clusters.out
  for each lattice. For instance
   -d1=../fcc/ -d2=../hcp/
  If either or both options are omitted, the files are read from the
  current directory.

-The starting temperature and chemical potential
  for instance,
  -T=100 -mu=0.125
  (Make sure to set blotzmann's constant appropriately.)
  (If you tracing the phase boundary between two ordered phases, starting from
  OK, you do not need to specify a starting T and mu.)

-The temperature step
  for instance -dT=50

-The 'enclosed radius', which sets the system size
  for instance, -er=35
  (see emc2 documentation for more information)

-The precision of the calculation
 This is expressed as the desired precision of the average concentration.
  For instance, -dx=1e-3
 The code automatically finds the equilibration time and the number
 Monte Carlo steps needed to obtain the target standard deviation
 of the average concentration.

There are a number of optional parameters as well.

 -ltep: The low temperature expansion is used to find the free energy
  at low temperature. -ltep gives the maximum error allowed before
  Monte Carlo is used instead of LTE.

 -dmu: The step in chemical potential used when scanning in search
  of the phase boundary.
  for instance, -dmu=0.005
  Sometimes, the algorithm looses track of the phase boundary
  (because of statistical errors). When that happens, it scans
  a range of values of the chemical potential in search of the
  boudary of the hysteresis loop associated with the first
  order transition of interest. It then position itself in the
  middle of it. dmu is the step size used for that search.
  Note that the code will automatically shrink dmu if needed.
  By default, dmu is automaticaly set to the formation energy
  of a disordered alloy times 0.01.

  -mug lets you specify a small difference in chemical potential
  between the phases, to make the code less sensitive to accidental
  phase transition.
```

```
  -k sets boltzman's constant (see emc2 documentation)
  -keV

  -dn indicates that the boundary must be folowed downward
   (decreasing  T)

Output file:

  Column      Value
  1           temperature
  2           chemical potential
  3           concentration of phase 1 in [-1,1]
  4           concentration of phase 2 in [-1,1]
  5           energy of phase 1
  6           energy of phase 2
```

5.4 mmaps

```
-> What does this program do?

This is the multicomponent version of the maps code.

It gradually constructs a increasingly more accurate cluster expansion.
A user-provided script running concurrently is responsible for notifying
maps when computer time is available. maps creates files describing
structures whose energy should be calculated. The user-provided script
sets up the runs needed to calculate the energy of these structures.
As maps becomes aware of more and more structural energies, it gradually
improves the precision of the cluster expansion, which is continously
written to an output file.
The code terminates when a stop file is created by typing, for instance,
touch stop

NOTE: Fully functional scripts are included with the package:
pollmach and runstruct_vasp.
For for more information type
  pollmach
  runstruct_vasp -h

-> Format of the input file defining the lattice (specified by the -l option)

First, the coordinate system is specified, either as
[a] [b] [c] [alpha] [beta] [gamma]
or as:
[ax] [ay] [az]
[bx] [by] [bz]
[cx] [cy] [cz]
Then the lattice vectors are listed, expressed in the coordinate system just defined:
[ua] [ub] [uc]
[va] [vb] [vc]
[wa] [wb] [wc]
Finally, atom positions and types are given, expressed in the same coordinate system
as the lattice vectors:
```

```
[atom1a] [atom1b] [atom1c] [atom1types]
[atom2a] [atom2b] [atom2c] [atom2types]
etc.

-The atom type is a comma-separated list of the atomic
 symbols of the atoms that can sit the lattice site.
-When only one symbol is listed, this site is ignored for the purpose
 of calculating correlations, but not for determining symmetry.

Examples:

The fcc lattice of the Cu-Au system:
3.8 3.8 3.8 90 90 90
0   0.5 0.5
0.5 0   0.5
0.5 0.5 0
0 0 0 Cu,Au

A lattice for the Li_x Co_y Al_(1-y) O_2 system:
 0.707 0.707 6.928 90 90 120
 0.3333  0.6667 0.3333
-0.6667 -0.3333 0.3333
 0.3333 -0.3333 0.3333
 0       0      0       Li,Vac
 0.3333  0.6667 0.0833  O
 0.6667  0.3333 0.1667  Co,Al
 0       0      0.25    O

Optional input file: ref_energy.in

  Contains the reference energy (per site) to be subtracted to get formation energies.
  The atomic reference energies must be in the same order as in the atoms.out file.
  If this file is omitted, the energies of the structures with the most
  extreme compositions are used to determine the reference energies,
  which are then output to the ref_energy.out file.

Optional input file: nbclusters.in
  Allows the user to manually select which clusters to include in the fit.
  This file should contains:
    number of pairs to include
    number of triplets to include
    etc.
  This file can be changed while maps is running. However, you must type
   touch refresh
  to tell maps to reread it.

Optional input file: crange.in
  Selects the range of concentration over which the cluster expansion
  is to be fitted. This controls both where the correct ground states
  are inforced in the fitting process and the range of concentration
  of the generated structures. Occasionally, a structure outside that
  range is generated, to verify the ground state hull.
  Here is an example of such file:
   1.0*Al -1.0*Li +0.1*Co >= 0.5
```

```
  Multiple constraints can be listed (on separate lines).
  Make sure to include a numerical prefactor for each species even
  if it is 1.0. Do not put a space between '-' and a number.

-> Output files

maps.log
Contains possible warnings:
  'Not enough known energies to fit CE'
  'True ground states not = fitted ground states'
  'New ground states predicted, see predstr.out'

These warning should disappear as more structural energies become available
and the following messages should be displayed:
  'Among structures of known energy, true and predicted ground states agree.'
  'No other ground states of xx atoms/unit cell or less exist.'

This file also gives the crossvalidation score of the current fit
(before the weighting is turned on in order to get the correct ground states).

atoms.out
Lists all atomic species given in the input files.

fit.out
Contains the results of the fit, one structure per line and each line
has the following information:
  concentration energy fitted_energy (energy-fitted_energy) weight index
'concentration' a vector of the atom fraction of all species (in the same
               order as in atoms.out).
'energy' is per site (a site is a place where more than one atom type can sit)
'weight' is the weight of this structure in the fit.
'index' is the name of the directory associated with this structure.

predstr.out
Contains the predicted energy (per site) of all structures maps has in memory but
whose true energy is unknown.
Format: one structure per line, and each line has the following information:
  concentration predicted_energy index status
index is the structure number (or -1 if not written to disk yet).
status is either b for busy (being calculated), e for error or u for unknown
(not yet calculated). A g is appended to status if that structure is predicted to be
a ground state. To list all predicted ground states, type
grep 'g' predstr.out

gs.out
Lists the ground state energies, one structure per line and each line
has the following information:
  concentration energy fitted_energy (energy-fitted_energy) index

gs_connect.out
Indicates which ground states touch each face of the ground state convex hull.

gs_str.out
Lists the ground state structures, in the same format as the n/str.out files
```

```
(see below). Each structure is terminated by the word 'end' on a line by itself,
followed by a blank line.

eci.out
Lists the eci. (They have already been divided by multiplicity.)
The corresponding clusters are in clusters.out

clusters.out
For each cluster, the first line is the multiplicity, the second line is the
cluster diameter, and the third line is the number of points in the cluster.
The remaining lines are the coordinates of the points in the cluster
(in the coordinate system specified in the input file defining the lattice).
A blank line separates each cluster.

n/str.out
Same format as the lattice file, except that
-The coordinate system is always written as 3x3 matrix
-Only one atom is listed for each site.

ref_energy.out
Reference energies used to calculate formation energies.
(Usually: energy of the pure end members OR values given in
ref_energy.in if provided.)

-> Communication protocol between maps and the script driving the
   energy method code (e.g. ab initio code)
   (Only those who want to customize the code need to read this section.
   The scripts described in this section are provided with the atat
   distribution in the glue/ subdirectory.)

Unless otherwise specified all files mentioned reside in the directory where
maps was started. All paths are relative to the startup directory.

+The script should first wait for computer time to be available before creating
 a file called 'ready'.
-Upon detecting that the 'ready' file has been created,
 maps responds by creating a subdirectory 'n' (where 'n' is a number) and a file
 'n/str.out' containing a description of a structure whose energy needs to be
 calculated.
-maps creates a file called 'n/wait' to distinguish this directory
 from other ones created earlier.
-maps deletes the 'ready' file.
+Upon detecting that the 'ready' file has disappeared,
 the script should now look for the 'n/wait' file, start the calculations
 in the directory 'n' and delete file 'n/wait'.
+If anything goes wrong in the calculations, the script should create a file
 'n/error'.
+When the calculations terminate successfully, the energy per unit cell of the structure
 should be copied to the file 'n/energy'.
 (NOTE: use energy per unit cell of the structure NOT per unit cell of the lattice).
-maps continuously scans all the subdirectories 'n' for 'n/energy' or 'n/error'
 files and updates the cluster expansion accordingly.
-maps updates the cluster expansion whenever a file called 'refresh' is created
 (maps then deletes it).
```

```
-maps terminates when a 'stop' file is created.

Note that the script can ask maps to create new structure directories even before
the energy of the current structure has been found.
Note that human intervention is allowed: an 'n/error' file can be
manually created if an error is later found in a run.
Users can also manually step up all runs if they wish so, as long
as they follow the protocol.

Example of script
(portions in /* */ have to be filled in with the appropriate code):

#!/bin/csh

while (! -e stop)
  /* check machine load here */
  if ( /* load low enough */ ) then
    touch ready
    while (-e ready)
      sleep 30
    end
    cd `ls */wait | sed 's+/.*++g' | head -1`
    rm -f wait
    /* convert str.out to the native format of ab initio code */
    /* in background: run code and create either energy file or error file */
    cd ..
  endif
  sleep 180
end

-> Using maps with vasp

The script runstruct_vasp, when run from within directory 'n',
1) converts 'vasp.wrap' and 'n/str.out' into all the necessary files to run vasp,
2) runs vasp
3) extract all the information from the output files and writes in a format
   readable by maps.

An example of vasp.wrap is:
[INCAR]
PREC = high
ISMEAR = -1
SIGMA = 0.1
NSW=41
IBRION = 2
ISIF = 3
KPPRA = 1000
DOSTATIC

See ezvasp documentation for more information.

-> Importing structures into maps

MAPS continuously scans all the first-level subdirectories containing
```

```
a file called str.out and tries to map them onto superstructures of the
lattice provided. This lets you 'import' structures from another source
into MAPS. A word of caution: the imported structures must be
unrelaxed and no effort is made to rotate or scale them in order to
match the lattice (aside from space group symmetry operations).
```

5.5 csfit

```
This code fits a reciprocal-space cluster expansion of the constituent strain
energy for binary systems with cubic symmetry.

USAGE:

csfit needs, as an input, 4 files.

1) A lat.in file defining the geometry of the lattice (same format as maps).
2) Two str_relax.out residing in given directories (-pa and -pb options, or, by
   default 0/ and 1/ ) and providing the relaxed geometry of the two pure
   end members for the system.
3) A dir.in file listing the directions along which to compute epitaxial
   deformation energy in order to fit the expansion.
   Directions are specified as miller indices in the coordinate system
   defined in lat.in .

The code then generates subdirectories (in 0/ and 1/ or any other directory
you specify) with deformed structures whose energy will be used to fit the
expansion. The -np and -nl options let you control the number of structures
generated: -np provides the number of intermediate epitaxial strain
(perpendicular to specified directions) considered between the lattice parameters
of the 2 pure elements while -nl provides the number of different values of
strain along the directions specified. The range of strains spanned is
given by the -ml option (+/- that value, 0 strain being the strain that
keeps volume constant). The -ns option controls the numerical energy
minimization (it is the number of mesh points used in the search for a
minimum) and has no effect of the number of generated structures.

All default values are very reasonable. You may want to increase -ml and
-nl for systems with a large size mismatch.

The files generated are compatible with the 'pollmach'
automatic job control utility. If you use vasp, you would
typically type
  csfit -d &
  pollmach runstruct_vasp -w csvasp.wrap &

where the csvasp.wrap contains the parameters needed for a vasp run
that does not relax the structure geometry (see atat/examples directory).

WARNING: you are free to rerun csfit many times, adding new lines in dir.in
but you CANNOT rerun it with different -ml -np -nl settings without first
deleting the subdirectories generated.

The csfit programs waits until all calculations are done and then fits
```

```
the expansion and writes the results to the cs.in file.
The cs.log contains the constituent strain as a function of concentration (rows)
for long period superlattices along the specified directions (columns).

You can use the resulting expansion in maps by using the -p=cs option
and in emc2 or phb by using the -ks=cs option.

CONVENTIONS:

We expand the concentration-dependent reciprocal space ECI associated with
constituent strain as:

  J_{CS}(x,k) = sum_{l=0,2,4,... } a_{l}(x) K_{l}(k)

where |K_{l}(k)|^2 is normalized to integrate to one over the unit spherical
shell |k|=1.
By combining Equations (12) and  (19) in V. Ozolins, C. Wolverton, and A. Zunger,
Phys. Rev. B, 57, 6427 (1998), we see that the coefficients a_{l}(x) are
related to the c_{l}(x) coefficients defined in this paper through:

  a_{l}(x)=c_{l}(x)/(4x(1-x))

The configuration-dependent constituent strain energy is given by

 Delta E_{CS}(sigma) = sum_{k} Delta J_{CS}(x,k) |S(sigma,k)|^2 exp(-|k|^2/k_c^2)

(see Equation (22) in C. Wolverton, V. Ozolins, A. Zunger, J. Phys: Condems. Matter, 12, 2749 (2000)).

The S(sigma,k) is computed according to the following convention:

  S(sigma,k)=sum_{j in unit cell of structure} S_j exp(i 2 pi k . R_j) *
     * (number of atom in unit cell of parent lattice)/
       (number of atom in unit cell of structure)

Delta E_{CS}(sigma) is thus given per unit cell of the parent lattice.

1/kc is given in the same unit of length as in the lat.in file (for instance, Angstroms).

The ouput file cs.in contains:

[1/k_c, set to 0 to turn off 'attenuation']
[Number of kubic harmonic to use, e.g. 2 to use K_0 and K_4]
[Number n of mesh point in concentration grid, including x=0 and x=1]

a_{0}(0)
a_{0}(1/(n-1))
a_{0}(2/(n-1))
.
.
.
a_{0}(1)

a_{4}(0)
a_{4}(1/(n-1))
```

```
a_{4}(2/(n-1))
.
.
.
a_{4}(1)

a_{6}(0)
a_{6}(1/(n-1))
a_{6}(2/(n-1))
.
.
.
a_{6}(1)

etc.


Additional output files

cs.log

Contains the constituent strain energy (cse) of superstructures along
the specified directions as a function of concentration.
Format:
[concentration] [cse along direction 1 of dir.in] [cse along direction 2 of dir.in] etc.
repeat for each concentration

csdebug.out

Contains the raw energies for each calculation.
The outer loop is on the directions of the dir.in file.
The middle loop is on the element (which pure end member).
The inner loop is on the stretching perpendicular to the k-vector.
Each line contains the energy for various amount of stretching along the
direction parallel to the k-vector.
```

5.6 corrdump

```
->What does this program do?

1) It reads the lattice file (specified by the -l option).

2) It determines the space group of this lattice and
   writes it to the sym.out file.

3) It finds all symmetrically distinct clusters that satisfy the
   conditions specified by the options -2 through -6.
   For instance, if -2=2.1 -3=1.1 is specified,
   only pairs shorter than 2.1 units and triplets containing
   no pairs longer than 1.1 will be selected.

4) It writes all clusters found to clusters.out.
   If the -c option is specified, clusters are read from clusters.out instead.
```

```
5) It reads the structure file (specified by the -s option).

6) It determines, for that structure, the correlations associated with all
   the clusters chosen earlier.
   This information is then output on one line, in the same order as in the
   clusters.out file. See below for conventions used to calculate correlations.

7) It writes the files corrdump.log containting the list of all adjustements
   needed to map the (possibly relaxed) structure onto the ideal lattice.

->File formats

Lattice and structure files

Both the lattice and the structure files have a similar structure.
First, the coordinate system is specified, either as
[a] [b] [c] [alpha] [beta] [gamma]
or as:
[ax] [ay] [az]
[bx] [by] [bz]
[cx] [cy] [cz]
Then the lattice vectors are listed, expressed in the coordinate system
just defined:
[ua] [ub] [uc]
[va] [vb] [vc]
[wa] [wb] [wc]
Finally, atom positions and types are given, expressed in the same coordinate system
as the lattice vectors:
[atom1a] [atom1b] [atom1c] [atom1type]
[atom2a] [atom2b] [atom2c] [atom1type]
etc.

In the lattice file:
-The atom type is a comma-separated list of the atomic
 symbols of the atoms that can sit the lattice site.
-The first symbol listed is assigned a spin of -1.
-When only one symbol is listed, this site is ignored for the purpose
 of calculating correlations, but not for determining symmetry.
-The atomic symbol 'Vac' is used to indicate a vacancy.

In the structure file:
-The atom type is just a single atomic symbol
 (which, of course, has to be among the atomic symbols given in the
 lattice file).
-Vacancies do not need to be specified.

Examples

The fcc lattice of the Cu-Au system:
1 1 1 90 90 90
0   0.5 0.5
0.5 0   0.5
0.5 0.5 0
```

```
0 0 0 Cu,Au

The Cu3Au L1_2 structure:
1 1 1 90 90 90
1 0 0
0 1 0
0 0 1
0   0   0   Au
0.5 0.5 0   Cu
0.5 0   0.5 Cu
0   0.5 0.5 Cu

A lattice for the Li_x Co_y Al_(1-y) O_2 system:
0.707 0.707 6.928 90 90 120
 0.3333  0.6667 0.3333
-0.6667 -0.3333 0.3333
 0.3333 -0.3333 0.3333
 0       0      0       Li,Vac
 0.3333  0.6667 0.0833  O
 0.6667  0.3333 0.1667  Co,Al
 0       0      0.25    O

Symmetry file format (sym.out)

[number of symmetry operations]

3x3 matrix: point operation

1x3 matrix: translation

repeat, etc.

Note that if you enter more than one unit cell of the lattice,
sym.out will contain some pure translations as symmetry elements.

Cluster file format (clusters.out)

for each cluster:
[multiplicity]
[length of the longest pair within the cluster]
[number of points in cluster]
[coordinates of 1st point] [number of possible species-2] [cluster function]
[coordinates of 2nd point] [number of possible species-2] [cluster function]
etc.

repeat, etc.

(Multiplicity and length are ignored when reading in the clusters.out file.)
For each 'point' the following convention apply
-The coordinates are expressed in the coordinate system given in
 the first line (or the first 3 lines) of the lat.in file.
-The 'number of possible species' distinguishes between binaries, ternaries, etc...
 Since each site can accomodate any number of atom types,
 this is specified for each point of the cluster.
```

```
-In multicomponent system, the cluster function are numbered from 0 to number of possible species-2.
In the simple of a binary system [number of possible species-2] [cluster function] are just 0 0.
For a ternary, the possible values are 1 0 and 1 1.
All the utilities that are not yet multicomponent-ready just ignore the entries [number of possible spec

Convention used to calculate the correlations:
  The cluster functions in a m-component system are defined as
   function '0' : -cos(2*PI*  1  *s/m)
   function '1' : -sin(2*PI*  1  *s/m)
                              .
                              .
                              .
                 -cos(2*PI*[m/2]*s/m)
                 -sin(2*PI*[m/2]*s/m)   <--- the last sin( ) is omitted if m is even
  where the occupation variable s can take any values in {0,...,m-1}
  and [...] denotes the 'round down' operation.
  Note that, these functions reduce to the single function (-1)^s in the binary case.

Special options:

-sym:  Just find the space group and then abort.
-clus: Just find space group and clusters and then abort.
-z:    To find symmetry operations, atoms are considered to lie on
       top of one another when they are less than this much apart.
-sig:  Number of significant digits printed.


->Cautions

When vacancies are specified, the program may not be able to warn
you that the structure and the lattice just don't fit.
Carefully inspect the corrdump.log file!

If the structure has significant cell shape relaxations, the program
will be unable to find how it relates to the ideal lattice.
The problem gets worse as the supercell size of the structure gets
bigger.

There is no limit on how far an atom in a structure can be from
the ideal lattice site. The program first finds the atom that can
be the most unambiguously assigned to a lattice site. It then
finds the next best assignement and so on. This is actually a
pretty robust way to do this. But keep in mind that the -z option
does NOT control this process.
```

5.7 fitsvsl

```
This code determines bond stiffness vs bond length relationship for the purpose
of calculating vibrational properties (with the svsl code).

It requires the following files as an input.
1) A lattice file (by default, lat.in, but this can be overridden with the -l option) which
   allows the code to determine what chemical bonds are present in the system.
```

```
   The format is as described in the maps documentation (see maps -h).
2) A list of directories containing structures that will be used to calculate force constants
   (by default strname.in, but this can be overridden with the -dn option).
   Each of the listed directory must contain
   a) a str.out file containing an ideal unrelaxed structure that will be used to automatically
      determine the nearest neighbor shell,
   b) a str_relax.out file containing the relaxed structure that will be used to calculate
      bond lengths and that the code will perturb in various ways to determine the force constants.

The code can operate in two modes: a structure generation mode and a fitting mode (indicated by the
-f option).

In structure generation mode:

  All the above input files are needed and the option
  -er must be specified in order to indicate the size of the supercells generated.
  The -er option indicates how far from each other a displaced atom must be from
  its periodic images, the code will infer the smallest supercell satisfying this constraint.
  Typically, -er should be 3 or 4 times the nearest neighbor distance. All of these distances
  are measured using the ideal structures (*/str.out).

  The following parameters have reasonable default values which can be overridden:
   -dr specifies the displacement of the perturbed atom, which is 0.2 Angstrom by default.
   -me specifies the maximum (linear) expansion of the structures for the purpose of
       lengthing the bond lenght. For instance -me=0.01 (the default) indicates that the
       supercells will be stretched by up to 1% isotropically.
   -nv indicates the number of intermediate lattice parameters sampled (by default 2, which is
       the minimum in order to be able to determine the length dependence of bond stiffness).

After the structure generation step:

  Each of the directory specified in strname.in will contain multiple subdirectories,
  each of which contains
    a) the ideal unrelaxed supercell in a str_ideal.out file.
    b) the relaxed but unperturbed supercell in a str_unpert.out file.
    c) the actual geometry of perturbed supercell calculation in a str.out file.
  The appropriate first-principles calculations can be performed using the other utilities in the atat
  package, such as runstruct_vasp. Of course a corresponding vasp.wrap file must given
  in order to provide the input parameters for the first-principles calculations.
  Make sure that these parameters indicate a static run (no relaxations!).
  After all (or some) of these calculations are done, each subdirectory will contain
    a force.out file containing the forces acting on each atoms
    a str_relax.out file containing the atomic positions (in the same order as in force.out)

The fitting mode (-f option) of ftsvsl then needs to be used.
  The lattice file (e.g. lat.in) must be present and the code will look for all files of the form
  */force.out and */*/force.out   and the corresponding files */str*.out and */*/str*.out .
  The code will then use that information to create the length-dependent force constants
  (this may take a few minutes) and ouputs them in
     slpring.out
  Here is an example of the format of this file:
    Al Al         (gives the type of bond)
    2             (2 parameters: linear fit is used)
    50.28971      \  polynomial coeficients of the stiffness vs length relationship for stretching term
```

```
    7.88973       /  (typically, stiffness is in eV/Angstrom^2 and length is in Angstrom)
    2             \
    6.12722        | idem for bending term
    -1.01641      /
    Ti Al          (repeat for each type of chemical bond)
    3
    etc.
  The only option controling this process is -op, which specifies the order of the polynomial
  used to fit the length dependence (by default -op=1 and a linear fit is used).
  (Contact the author for information about the -sf option.)

Diagnostic files are also output:
  fitsvsl.log                     (a log file)
  fitsvsl.gnu and f_*.dat         (to plot the s vs l relationship)
```

5.8 svsl

```
This code calculates the vibrational free energy of a structure
using the Stiffness VS Length method.

Before using this code, you will probably first need to use the fitsvsl code, which
generates the length-dependent force constants that the present needs as an input.

It requires, as an input, 3 files:
1) A 'relaxed' structure (the default file name is str_relax.out,
   but it can be overridden with the -rs option).
   This provides the actual atomic positions used in the calculations.
   The format of this file is as described in the maps documentation (see maps -h).
2) An 'unrelaxed' structure (the default file name is str.out,
   but it can be overridden with the -us option).
   This provides the ideal atomic positions that are used to automatically determine which atoms
   lie in the nearest neighbor shell. This file can be the same as the relaxed structure
   but then the determination of the nn shell may be less reliable.
3) A spring constant file (the default is to lookup in slspring.out and ../slspring.out but
   this can be overridden with the -sp option).
   This file provides the bond stiffness vs bond length relationships that are
   used to determine the force constants of the springs joining neighboring atoms.
   The format of this file is described in the documentation of the fitsvsl code, which is
   a utility that fits such spring constants.

A number of optional files can be given as well.
4) An input file defining the atomic masses. By default, the code looks up in the
   ~/.atat.rc file to determine the directory where atat is installed and then
   loads the file data/masses.in. This behavior can be overridden with the -m option.
5) By default, the bulk modulus is calculated from the force constants but it
   can also be read from a file (whose name is specified with the -bf option) or
   specified on the command line with the -b option.

The parameters that govern the accuracy of the calculations are as follows.
The default values are all reasonable.
   The code uses the quasiharmonic approximation account for thermal expansion.
   For this purpose, it calculates the vibrational free for a range of lattice parameters
```

```
   from the 0K value to the (1+ms) larger, where ms is the number specified in the -ms option.
   The -ns option gives the number of volumes considered.
   Setting -ns=1 selects the harmonic (instead of the quasiharmonic) approximation.

   The code calculates the vibrational free energy from temperature T0 to T1 in steps of dT.
   a) The defaults are T0=0 T1=2000 dT=100.
   b) If a file Trange.in exists in the upper directory, it is used to set T0,T1,dT:
      Trange.in must contain two numbers on one line separated by a space: T1 (T1/dT+1).
      Note that T0=0 always.
      For phase diagram calculations, you must use this method to specify the temperature range.
   c) These defaults can be overridden by the -T0, -T1 and -dT options.

   The kpoint sampling is specified with the -kp option. The actual number of kpoints used
   is the number give divided by the number of atoms in the cell.

-> Phonon Dispersion curves
The -df=inputfile option invokes the phonon dispersion curve module.
The syntax of the input file is:
[nb of points] [kx1] [ky1] [kz1] [kx2] [ky2] [kz2]
repeat...

Each line of input defines one segment (kx1,ky1,kz1)-(kx2,ky2,kz2)
along which the dispersion curve is to be calculated.
[nb of points] specifies the number of points sampled along the segment.
The coordinates are in multiple of the reciprocal cell defined by the axes in the
file specified by the -us option (or, by default, in the str.out file).
(The k-point coordinates are appropriately strained
by the amount needed for the str.out file to be identical to the str_relax.out file.)
The phonon frequencies are output in the eigenfreq.out file.

Other parameters can be altered, if needed.
   The physical constants are set by default for
         force constants input in eV/Angstrom^2
         temperature in K
         free energy in eV
         frequencies in Hz
         masses      in a.u.
   They can be altered by the -hb, -kb, -cfk and -mu options.

   By default the code give free energies per unit cell, but the -pa option gives them per atom.
   The -sc option can provide a multiplicative factor for other conventions (e.g. per formula unit).

   By default, the code aborts whenever unstable modes are found, unless the -fn option is specified.

   Contact the author for information about the -df and -sf options.

The output files are as follows:

svsl.log :  a log file giving some of the intermediate steps of the calculations
vdos.out:   the phonon density of states for each lattice parameter considered (unstable modes appear
            as negative frequencies).
svsl.out:   gives along each row, the temperature, the free energy, and the linear thermal expansion
            (e.g. 0.01 means that the lattice has expansion by 1% at that temperature).
fvib:       gives only the free energy
```

```
-> For including vibrations in phase diagram calculations

 You are likely to use this code as follow:

  #first create the Trange.in file for up to 2000K in intervals of 100K:
  echo 2000 21 > Trange.in

  #This executes the svsl code in each subdirectory containing str_relax.out but no error file.
  foreachfile -e str_relax.out pwd \; svsl [options if desired]

  #constructs a cluster expansion of the vibrational free energy (eci are in fvib.eci)
  clusterexpand fvib

  #add the energetic eci from eci.out to the vibrational eci from fvib.eci and create the teci.out
  #file that will be read by the Monte Carlo code.
  mkteci fvib.eci
```

5.9 fitfc

```
Calculates vibrational properties by fitting a spring model to
reaction forces resulting from imposed atomic displacements.

The examples below are given assuming that one uses the vasp code,
although other ab initio codes would work as well.

The calculations proceed as follows:

1) You first need to fully relax the structure of interest.  The code
   expects the relaxed geometry in the file str_relax.out. It also
   expects a str.out file containing the unrelaxed geometry (which may be
   the same as the relaxed geometry, if you wish). The unrelaxed geometry
   will be used to determine the neighbor shells and measure distances
   between atoms.  Typically the user would specify the str.out file,
   then obtain the str_relax.out file by runing an ab initio code with a
   command of the form
       runstruct_vasp
   (making sure the vasp.wrap file indicates that all degrees of freedom
   must be relaxed).

2) Generation of the pertubations.
2a) A typical command line is as follows:
    fitfc -er=11.5 -ns=3 -ms=0.02 -dr=0.1
-er specifies how far apart the periodic images of the displaced atom must be.
    The code then finds the size of the supercell satisfying this constraint.
    Distances are measured according to the atomic positions given in str.out
    and in the same units.
-ns specifies the number of different strain at which phonon calculations
    will be performed.
    (-ns=1 implies a purely harmonic model, the default while values greater
    than 1 will invoke a quasiharmonic model)
-ms specifies the maximum strain (0.02 signifies a 2% elongation along
    every direction).
```

```
-dr the magnitude of the displacement of the perturbed atom.
The above command writes out a series of subdirectories vol_*, one for
each level of strain.
If the structure has cubic symmetry or if you are willing to assume that
thermal expansion is isotropic or if you only which to use the harmonic
approximation, the fitfc command should be invoked with the -nrr option
(do Not ReRelax) and you can now skip to step 3).

2b) Each volume subdirectory now contains a str.out file which is
    stretched version of the main str_relax.out file provided.
    You then need to run the ab initio code to rerelax the geometry at
    the various levels of imposed strain and obtain the energy as a
    function of strain. Typically, this is acheived by typing:
      pollmach runstruct_vasp &
    (make sure that the vasp.wrap file is modified so that all degrees
    of freedom except volume are allowed to relax.)
    After this command each subdirectory will contain an energy and
    a str_relax.out file.
    Type
      touch stoppoll
    after all energies have been calculated.
    The runstruct_vasp command can also be executed manually in each
    subdirectory or as follows:
       foreachfile wait runstruct_vasp \; rm wait

2c) Now you need to reinvoke fitfc to generate perturbations of the
    atomic position for each level of strain.
      fitfc -er=11.5 -ns=3 -ms=0.02 -dr=0.1
    This is exactly the same commmand as before but the code notices
    the presence on the new files and can proceed further.

3) At this point the files generated are arranged as follows.
   At the top level, there is one subdirectory per level of strain
   (vol_*, where * is the strain in percent), and in each
   subdirectory, there are a number of subsubdirectories,
   each containing a different perturbation. The pertubation
   names have the form p_<+/-><dr>_<er>_<index>, where
   <pertmag> is the number given by the -dr option,
   <er> by the -er option and <index> is a number used to distinguish
   between different pertubations. Two perturbations that differ only
   by their signs are sometimes generated and are distinguished
   by a + or - prefix.
   If you want to ensure that the third-order force constants
   cancel out exactly in the fit, you need to consider both
   perturbations.
   Otherwise, only the 'positive' perturbation will be sufficient.
   Note that whenever the third-order terms cancel out by
   symmetry, only the 'positive' perturbation will be generated.

   You then need to use the ab initio code to calculate reaction
   forces for each perturbation.
   This will typically be accomplished by typing
     pollmach runstruct_vasp &
   (make sure that the vasp.wrap file indicates that no degrees of
```

```
   freedom are allowed to relax and the smearing is used
   for Brillouin zone integration. Do not use the DOSTATIC option.)

4) Fitting the force constants and phonon calculations.
   This mode is invoked with the -f option.
   In addition, you need to specify the range of the springs included
   in the fit using the -fr=... option.
   Usually, the range specified with -fr should be not more than
   half the distance specified with the -er option ealier.
   Distances are measured according to the atomic positions given
   in str.out and in the same units.
   It is a good idea to try different values of -fr (starting with
   the nearest neighbor bond length) and check that
   the vibrational properties converge as -fr is increased.

-> Phonon Dispersion curves
The -df=inputfile option invokes the phonon dispersion curve module.
The syntax of the input file is:
[nb of points] [kx1] [ky1] [kz1] [kx2] [ky2] [kz2]
repeat...

Each line of input defines one segment (kx1,ky1,kz1)-(kx2,ky2,kz2)
along which the dispersion curve is to be calculated.
[nb of points] specifies the number of points sampled along the segment.
The coordinates are in multiple of the reciprocal cell defined by the axes in the
file specified by the -si option (or, by default, in the str.out file).
(The k-point coordinates are appropriately strained
by the amount needed for the str.out file to be identical to the str_relax.out file.)
The phonon frequencies are output in the eigenfreq.out file,
in the vol_* subdirectories.

Output files:
   fitfc.log       : A general log file.
   vol_*/vdos.out  : the phonon density of states for each volume considered
   vol_*/fc.out    : The force constants.
                     For each force constant a summary line gives:
                         (i)   the atomic species involved
                         (ii)  the 'bond length'
                         (iii) the stretching and bending terms
                     Then, each separate component of the force constant is
                     given and, finally, their sum.


  fitfc.out        : gives along each row, the temperature, the free energy,
                     and the linear thermal expansion
                      (e.g. 0.01 means that the lattice has expansion by 1%
                       at that temperature).
  fvib             : gives only the free energy
```

5.10 felec

```
This code calculates the electronic free energy within the one-electron
and temperature-independent bands approximations.
```

```
It needs an dos.out input file (whose name can be changed with -dos option) that
has the following format:
 [number of electron in unitcell]
 [energy width of each bins used to calculate the dos]
 [a multiplicative scale factor to adjust units]
 [the density in each bin, in states/unit cell/energy] <- repeated

The code calculates the electronic free energy from temperature T0 to T1 in steps of dT.
  a) The defaults are T0=0 T1=2000 dT=100.
  b) If a file Trange.in exists in the upper directory, it is used to set T0,T1,dT:
     Trange.in must contain two numbers on one line separated by a space: T1 (T1/dT+1).
     Note that T0=0 always.
     For phase diagram calculations, you must use this method to specify the temperature range.
  c) These defaults can be overridden by the -T0, -T1 and -dT options.

The output files contain the free energy per unit cell.
  felec.log contain temperature and corresponding free energy on each line.
  felec contains the free energies only.
  plotdos.out contains the dos (col 1: energy normalized so that Ef=0 , col 2: DOS)

-> For including electronic entropy in phase diagram calculations

 You are likely to use this code as follow:

  #first create the Trange.in file for up to 2000K in intervals of 100K:
  echo 2000 21 > Trange.in

  #This executes the svsl code in each subdirectory containing dos.out but no error file.
  foreachfile -e dos.out pwd \; felec [options if desired]

  #constructs a cluster expansion of the electronic free energy (eci are in felec.eci)
  clusterexpand felec

  #add the energetic eci from eci.out to the electronic eci from felec.eci and create the teci.out
  #file that will be read by the Monte Carlo code.
  mkteci felec.eci

  #you can even combine vibrational and electronic eci:
  mkteci fvib.eci felec.eci
```

5.11 gensqs

```
This code requires 3 input files:
1) A lattice file (by default lat.in) in the same format as for
   maps or corrdump.
2) A cluster file (by default clusters.out), as generated with
    the corrdump utility.
3) A target correlation file (by default tcorr.out) which
   contains the value of desired correlations for each of
   the clusters listed in the cluster file.
```

```
A typical caling sequence would be:

# the following command can be used to generate the target correlation file tcorr.out

corrdump -noe -clus -2=maxradius -rnd -s=conc.in > tcorr.out

# where maxradius is the length of the longest pair desired
# and where conc.in contains an (ordered) structure having the
# desired concentration.
# The geometry of the structure in the conc.in file is basically ignored
# only its concentration will be used.

#this looks for possible sqs of 8 atoms/ cell
gensqs -n=8 > sqs8.out

corrdump -2=anotherradius -3=anotherradius  -noe -s=sqs8.out
# this helps you decide which sqs is best based on other correlations
# associated with clusters (pairs and triplets) of diamter less than
# anotherradius.

Caution:
 If you give too many correlations to match, the code may not
 output anything.
 Finding an 8-atom sqs takes a few minutes, an 16-atom sqs, a few hours
 and a 32-atom sqs, a few days!
 The exact speed depends on the symmetry of the lattice and on your
 computer.
```

5.12 Command line options

5.12.1 maps

```
MIT Ab initio Phase Stability (MAPS) code 2.30, by Axel van de Walle
  -h          Display more help
  -l=[string] Input file defining the lattice (default: lat.in)
  -z=[real]   Tolerance for finding symmetry operations (default: 1e-3)
  -c=[real]   Exponent of the order of complexity (default: 3)
  -t=[int]    Time between disk reads in sec (default: 10 sec)
  -m=[int]    Maximum number of points in cluster (default 4)
  -g=[int]    Extend ground state search up to structures having at least that many atoms.
 -c0=[real]   [c0,c1] is the concentration range where ground states must be correct.
 -c1=[real]
 -2d          Find supercells along a and b axes only
  -p=[string] Predictor plugins to use (examples: -p=cs or -p=cs_el)
 -ks=[string] same as -p
 -fa=[string] Select fitting algorithm (default: built-in)
  -q          Quiet mode (do not print status to stderr)
-sig=[int]    Number of significant digits to print in output files
  -d          Use all default values
```

5.12.2 mmaps

```
MIT Ab initio Phase Stability (MAPS) code 2.30, by Axel van de Walle
  -h          Display more help
```

```
  -l=[string] Input file defining the lattice (default: lat.in)
  -z=[real]   Tolerance for finding symmetry operations (default: 1e-3)
  -c=[real]   Exponent of the order of complexity (default: 3)
  -t=[int]    Time between disk reads in sec (default: 10 sec)
  -m=[int]    Maximum number of points in cluster (default 4)
  -g=[int]    Extend ground state search up to structures having at least that many atoms.
 -cr=[string] Concentration region input file
 -he=[real]   Highest predicted energy, above ground state hull, allowed when generating structures (def
 -2d          Find supercells along a and b axes only
  -p=[string] Predictor plugins to use (examples: -p=cs or -p=cs_el)
 -ks=[string] same as -p
 -fa=[string] Select fitting algorithm (default: built-in)
 -cf=[string] Select correlation functions (default: trigo)
 -pn=[string] Property to cluster expand (default: energy)
 -ig          Ignore whether cluster expansion predicts correct ground states
 -pa          Quantity to expand is already per atom
  -q          Quiet mode (do not print status to stderr)
-sig=[int]    Number of significant digits to print in output files
  -d          Use all default values
```

5.12.3 emc2

```
Eazy Monte Carlo Code 2.30, by Axel van de Walle
     -h          Help
   -mu0=[real]   initial chemical potential
    -T0=[real]   initial temperature
   -mu1=[real]   final chemical potential
    -T1=[real]   final temperature
   -dmu=[real]   chemical potential step
    -dT=[real]   temperature step
    -db=[real]   inverse temperature step
    -cm          Set Canonical mode
     -x=[real]   Set concentration (implies -cm)
   -abs          take chemical potentials as absolute quantities (as in mc.out)
  -phi0=[real]   initial (grand) canonical potential
    -er=[real]   set the system  size so that a sphere of that radius must fit inside the simulation cel
    -eq=[int]    number of equilibration passes
     -n=[int]    number of averaging passes
    -dx=[real]   Target precision for the average concentration (optional, replaces -n and -eq)
    -aq=[int]    Alternative quantity that must meet the tolerance specified by -dx. 0: energy, 1: conce
    -gs=[int]    which ground state to use as initial config (-gs=-1 to use random state, c=1/2)
-innerT          inner loop over T
 -tstat=[real]   Critical value of the test for discontinuity
-sigdig=[int]    Number of significant digits printed
     -q          Quiet (do not write to stdout)
     -o=[string] Output file (default: mc.out)
     -k=[real]   Boltzman's constant (conversion factor from T to energy)
   -keV          Set Boltzman's constant to 8.617e-5 so that temperature is in K when energy is in eV
    -sd=[int]    Seed for random number generation (default: use clock)
    -dl          Drop the last data point of each inner loop (after the phase transition occured)
   -g2c          Convert output to canonical rather than grand-canonical quantities
    -is=[string] User specified initial configuration (replaces -gs)
    -ks=[string] Specify how k space ECI are calculated (e.g. -ks=cs).
```

5.12.4 phb

```
PHase Boundary2.30, by Axel van de Walle
     -h          Help
    -mu=[real]   initial chemical potential
     -T=[real]   initial temperature
   -dmu=[real]   chemical potential adjustment step
    -dT=[real]   temperature step
   -mug=[real]   Gap between the mu in phase 1 and mu in phase 2 (default: 0)
  -ltep=[real]   threshold free energy precision to use MC instead of LTE (in units of T) (default: alwa
    -er=[real]   enclosed radius
   -gs1=[int]    ground state for phase #1
   -gs2=[int]    ground state for phase #2
    -d1=[string] directory for phase #1 (default: current dir)
    -d2=[string] directory for phase #2 (default: current dir)
 -tstat=[real]   Critical value of the test for discontinuity
  -smax=[real]   Maximum step (experimental feature)
-sigdig=[int]    Number of significant digits printed
     -q          Quiet (do not write to stdout)
     -o=[string] Output file (default: mc.out)
     -k=[real]   Boltzman's constant (conversion factor from T to energy)
   -keV          Set Boltzman's constant to 8.617e-5 so that temperature is in K when energy is in eV
    -sd=[int]    Seed for random number generation (default: use clock)
    -dn          Go down in temperature
    -dx=[real]   Concentration Precision
```

5.12.5 checkcell

```
check cell distortion 2.30, by Axel van de Walle
-d          default
-q          be quiet
-p          print strain
```

5.12.6 corrdump

```
CORRelation DUMPer 2.30, by Axel van de Walle
   -h          Display more help
   -2=[real]   Maximum distance between two points within a pair
   -3=[real]   Maximum distance between two points within a triplet
   -4=[real]   Maximum distance between two points within a quadruplet
   -5=[real]   Maximum distance between two points within a quintuplet
   -6=[real]   Maximum distance between two points within a sextuplet
   -l=[string] Input file defining the lattice   (default: lat.in)
   -s=[string] Input file defining the structure (default: str.out)
 -sym          Just find space group
-clus          Just find clusters
   -c          Read clusters.out file instead of writing it
   -z=[real]   Tolerance for finding symmetry operations (default: 1e-3)
 -sig=[int]    Number of significant digits printed (default: 5)
 -noe          Do not include empty cluster
 -rnd          Print correlation of the random state of the same composition as the input structure
 -eci=[string] Predict quantity using ECI in specified file
  -mi          Multiplicities are already included in ECI file
  -cf=[string] Select correlation functions (default: trigo)
  -nb          Print structure number
```

5.12.7 kmesh

```
-d          Use all default options
-q          Quiet!
-r          Round output to next largest integer
-e          Force nb of kpoints to be an even number
```

5.12.8 genstr

```
GENerate STRuctures 2.30, by Axel van de Walle
  -n=[int]    maximum nb of atom/unit cell
-sig=[int]    Number of significant digits to print in output files
 -2d          Find supercells along a and b axes only
  -l=[string] Input file defining the lattice (default: lat.in)
```

5.12.9 gensqs

```
GENerate Special Quasirandom Structures 2.30, by Axel van de Walle
  -n=[int]    nb of atom/unit cell
-sig=[int]    Number of significant digits to print in output files
 -cf=[string] Input file defining the clusters (default: clusters.out)
 -tc=[string] Input file defining the target correlations (default: tcorr.out)
  -l=[string] Input file defining the lattice (default: lat.in)
 -rc          Read unit cells from file
 -tp=[int]    Total number of processes (for parallel operation)
 -ip=[int]    Index of current process (0,...,tp-1) (for parallel operation)
  -h          Display more help
```

5.12.10 nntouch

```
nntouch 2.30, by Axel van de Walle
 Scales a lattice so that nearest neighbor atom just touch.
  -l=[string] Input file defining the lattice (default: lat.in)
  -r=[string] Input file defining the atomic radii (default: rad.in)
  -s=[real]   Multiplicative scaling factor
-sig=[int]    Number of significant digits to print in output files
  -d          Use all default values
```

5.12.11 fixcell

```
fixcell 2.30, by Axel van de Walle
  -d          Use all default values
  -c          read and print cell in cartesian only (no axes specified)
  -b          print bravais lattice type only and conventional (potentially not primitive) cell
-sig=[int]    Number of significant digits to print in output files
```

5.12.12 csfit

```
Constituent Strain FITter 2.30, by Axel van de Walle
  -h          Display more help
 -nc=[int]    Number of points in concentration mesh (default 50)
 -ns=[int]    Number of points in mesh used to look for energy minimum (default 100)
 -np=[int]    Number of points in stretching mesh in the direction perpendicular to the k-vector(default
 -nl=[int]    Number of points in stretching mesh in the direction parallel to the k-vector (default 3)
 -ml=[real]   Maximum parallel stretching (default 0.05)
```

```
  -l=[string] Input file defining the lattice (default: lat.in)
 -pa=[string] Directory containing the pure A calculations (default 0/)
 -pb=[string] Directory containing the pure B calculations (default 1/)
 -ds=[string] File containing a list of stretching directions (default: dir.in)
  -t=[int]    Time between disk reads in sec (default: 10 sec)
-sig=[int]    Number of significant digits to print in output files
  -d          Use all default values
```

5.12.13 cv

```
Cross-Validation code 2.30, by Axel van de Walle
-g          Favor cluster choices where the ground state line is correct
-w=[real]   Weight structures by w/(struct_energy-gs_energy+w)
-p=[real]   Penality for structures that lie below the ground state line
-d          Use all default values
-h          Display Help
```

5.12.14 cellcvrt

```
cellcvrt 2.30, by Axel van de Walle
Converts structure (or lattice) files between fractional or cartesian format.
Reads from stdin, writes to stdout.
  -c          Use cartesian coordinates
  -f          Use fractional coordinates
-abc          Use a b c alpha beta gamma format
  -u=[string] User-specified coordinate system input file (optional)
-uss=[string] User-specified supercell
  -s          Look for smaller unit cell
 -sh=[string] Shift all atoms (default 0,0,0).
 -sg=[string] Space group file
 -wi          Wrap all atoms inside unit cell
-wiu          Wrap all atoms inside unit cell and undo shift
 -rr          Remove redundant atoms
 -ar          Add redundant atoms
-osf=[string] Original setting file (optional)
-fsf=[string] Final setting file (optional)
  -r          Print reciprocal unit cell
 -fs=[int]    Index of first structure to process (default: 1)
 -ns=[int]    Number of structures to process
 -pn          Print the number of atoms in the structure only
 -pv          Print volume of cell only
 -sc=[real]   Scale factor (default: 1)
-sig=[int]    Number of significant digits to print in output files
```

5.12.15 lsfit

```
Least-Square FIT 2.30, by Axel van de Walle
    -x=[string] x file
    -y=[string] y file
    -w=[string] weight file
    -r=[string] regularization parameters file
   -pw=[string] number of powers of each column to regress on
    -s=[string] select columns to regress on (one number by column, 0: ignore, 1: use)
    -1          add a column of ones as regressor
```

```
-colin          Ignore colinear columns in x file
   -cv          Print crossvalidation score
   -se          Print standard errors
   -ty          Print trues values of y
    -p          Print predicted values of y
    -e          Print prediction error
```

5.12.16 fitsvsl

```
Fit Stiffness VS Length transferable force constants 2.30, by Axel van de Walle
  -f          Fit force constants (otherwise, generate pertubations)
  -l=[string] Input file defining the lattice (default: lat.in)
 -dn=[string] Input file listing the directories containing the structures used to calculate force const
 -er=[real]   Minimum distance between displaced atoms
 -dr=[real]   Displacement of the perturbed atom (default: 0.2)
 -ms=[real]   Strain of the maximum volume sampled (default: 0.01)
 -ns=[int]    Number of volume sample (default: 2)
 -op=[int]    Order of the polynomial used to fit stiffness vs length
 -dd          Use direction-dependent force constants
 -pc=[int]    Maximum power of composition used in fit (default: no composition dependence)
-msl=[real]   Maximum spring length
 -sf=[string] Extra strain file
-eqt=[real]   Tolerance for excluding force constants in plots.
-sig=[int]    Number of significant digits printed (default: 5)
  -z=[real]   Tolerance for finding symmetry operations (default: 1e-3)
  -h          Display more help
```

5.12.17 svsl

```
Vibrational free energy calculator using the Stiffness VS Length method 2.30, by Axel van de Walle
  -l=[string] Input file defining the lattice (defaults: lat.in, ../lat.in, str.out)
 -us=[string] Input file defining the unrelaxed structure (default: str.out)
 -rs=[string] Input file defining the   relaxed structure (default: str_relax.out)
 -sp=[string] Input file defining the springs (default: slspring.out)
  -m=[string] Input file defining the atomic masses (default: ${atatdir}/data/masses.in)
  -b=[real]   Bulk modulus
 -bf=[string] Bulk modulus file
 -ms=[real]   Maximum stretching of the lattice parameter (default: 0.05)
 -ns=[int]    Number of lattice parameter stretching step (default: 1 => harmonic approximation)
 -T0=[real]   Minimum temperature (default: 0)
 -T1=[real]   Maximum temperature (default: 2000)
 -dT=[real]   Temperature step (default: 100)
 -kp=[real]   Number of k-points per reciprocal atom (default: 1000)
 -sx=[real]   k-point shift (along 1st recip lat. vect.)
 -sy=[real]   k-point shift (along 2nd recip lat. vect.)
 -sz=[real]   k-point shift (along 3rd recip lat. vect.)
 -hp=[real]   Planck's constant (default in (eV s))
 -kb=[real]   Boltzman's constant (default in eV/K)
-cfk=[real]   Conversion factor for force constants into energy/dist^2 (default: converts eV/A^2 into J/
 -mu=[real]   Mass units (default: converts a.u. mass into kg)
 -pa          Output free energy per atom instead of per unit cell
 -sc=[real]   Correction factor if spectator ion are present (default: 1)
 -fn          Force continuation of calculations even if unstable
 -df=[string] Phonon dispersion curve calculation input file.
```

```
 -sf=[string] Extra strain file
-msl=[real]   Maximum spring length
-sig=[int]    Number of significant digits to print in output files
  -d          Use all default values
  -h          Display more help
```

5.12.18 felec

```
Electronic free energy calculator 2.30, by Axel van de Walle
-dos=[string] DOS input file name (default: dos.out)
 -T0=[real]   Minimum temperature (default: 0)
 -T1=[real]   Maximum temperature (default: 2000)
 -dT=[real]   Temperature step (default: 100)
 -kb=[real]   Boltzman's constant (default: in eV/K)
 -sc=[real]   Correction factor if spectator ion are present
 -sd=[real]   Smooth DOS with a Gaussian this width
-sig=[int]    Number of significant digits to print in output files
  -h          Display more help
  -d          Use all default values
```

5.12.19 pdef

```
Point DEFect generator 2.30, by Axel van de Walle
  -l=[string] Input file defining the lattice (default: lat.in)
  -s=[string] Input file defining the lattice (default: str.out)
  -p=[string] Prefix of the directories that will contain the defected structures (Default pdef)
 -er=[real]   Minimum distance between displaced atoms
-sig=[int]    Number of significant digits printed (default: 5)
  -h          Display more help
```

5.12.20 fitfc

```
Fit Stiffness VS Length transferable force constants 2.30, by Axel van de Walle
  -f          Fit force constants (otherwise, generate pertubations)
 -si=[string] Input file defining the ideal structure (default: str.out)
 -sr=[string] Input file defining the relaxed structure (default: str_relax.out)
 -er=[real]   Minimum distance between displaced atoms
 -fr=[real]   Force constant range
 -dr=[real]   Displacement of the perturbed atom (default: 0.2)
 -ms=[real]   Strain of the maximum volume sampled (default: 0.01)
 -ns=[int]    Number of volume sample (default: 2)
-nrr          Do not rerelax structures at each new volume
-ncs          No check for singular matrix in fit
 -sf=[string] Extra strain file
  -m=[string] Input file defining the atomic masses (default: ${atatdir}/data/masses.in)
 -T0=[real]   Minimum temperature (default: 0)
 -T1=[real]   Maximum temperature (default: 2000)
 -dT=[real]   Temperature step (default: 100)
 -kp=[real]   Number of k-points per reciprocal atom (default: 1000)
 -sx=[real]   k-point shift (along 1st recip lat. vect.)
 -sy=[real]   k-point shift (along 2nd recip lat. vect.)
 -sz=[real]   k-point shift (along 3rd recip lat. vect.)
 -df=[string] Phonon dispersion curve calculation input file.
 -hp=[real]   Planck's constant (default in (eV s))
```

```
 -kb=[real]   Boltzman's constant (default in eV/K)
-cfk=[real]   Conversion factor for force constants into energy/dist^2 (default: converts eV/A^2 into J/
 -mu=[real]   Mass units (default: converts a.u. mass into kg)
 -pa          Output free energy per atom instead of per unit cell
 -sc=[real]   Correction factor if spectator ion are present (default: 1)
-me0          Subtract energy at 0K
 -fn          Force continuation of calculations even if unstable
-sfc=[int]    Simplify force constants: 1=streching+bending, 2=symmetric
-sig=[int]    Number of significant digits printed (default: 5)
  -z=[real]   Tolerance for finding symmetry operations (default: 1e-3)
  -h          Display more help
```

5.12.21 nnshell

```
Nearest Neighbor Shell2.30, by Axel van de Walle
  -l=[string] Input file defining the lattice (default: lat.in)
 -lb          List bond types in given structures
-sig=[int]    Number of significant digits to print in output files
  -d          Use all default values
```

5.12.22 memc2

```
Eazy Monte Carlo Code 2.30, by Axel van de Walle
     -h          Help
    -er=[real]   set the system  size so that a sphere of that radius must fit inside the simulation cel
    -eq=[int]    number of equilibration passes
     -n=[int]    number of averaging passes
    -tp=[real]   Target precision (optional, replaces -n and -eq)
    -aq=[int]    Quantity that must meet the tolerance specified by -tp. 0: energy (default), 1: long-ra
    -gs=[int]    which ground state to use as initial config (-gs=-1 to use random state, c=1/2)
 -tstat=[real]   Critical value of the test for discontinuity
-sigdig=[int]    Number of significant digits printed
     -q          Quiet (do not write to stdout)
     -o=[string] Output file (default: mc.out)
     -k=[real]   Boltzman's constant (conversion factor from T to energy)
   -keV          Set Boltzman's constant to 8.617e-5 so that temperature is in K when energy is in eV
    -sd=[int]    Seed for random number generation (default: use clock)
    -dl          Drop the last data point of each inner loop (after the phase transition occured)
   -g2c          Convert output to canonical rather than grand-canonical quantities
    -is=[string] User specified initial configuration (replaces -gs)
```

Chapter 6

Troubleshooting

6.1 FAQ

Q. I can do a least-squares by hand with a much lower mean square error (MSE) than MAPS can. Yet, you claim that MAPS is "optimal". How is this possible?
A. The MSE is not a very good measure of the predictive power of a cluster expansion. It can be made arbitrarily close to zero but merely including enough ECI in the fit! But having a small MSE is no guarantee that the error on the predicted energy of structures not inlcuded in the fit is small. The cross-validation (CV) score, which is used in MAPS, is a better measure of the predictive power of a least-squares fit. The CV score does not systematically decrease as the number of ECI increases and it has been shown that choosing the number of ECI such that the CV score is minimized is an asymptotically optimal strategy. Researchers often have a tendency to use too many ECI because this appears to reduce the error, but this is a illusion. MAPS gives you an unbiased estimaqte of the prediction error, and the magnitude of that number is larger than many would like to believe.
Q. MAPS is frozen. What is going on?
A. MAPS sometimes needs a few minutes to generates new clusters or structures. This is a complex operation, be patient. This generation can occur in the middle of the calculations (finding the best cluster expansion or the best structures) because MAPS only generates these clusters or structures when they are needed.
Q. Where are the results?
A. In a variety of files whose descriptions you can view by typing `maps -h | more`. A utility called `mapsrep` lets you display the most useful output data using `gnuplot`.
Q. What files do I need to get started with MAPS?
A. A `lat.in` files to specify the geometry of the lattice and a "wrapping" file that specifies the parameters of the first-principles code. If you are using vasp, this file is usually called `vasp.wrap`. Examples of those files can be found in the `example` directory.
Q. I already have structures and energies. What if I just want to fit them?
A. You need a `lat.in` files to specify the geometry of the lattice and you need to create one subdirectory for each structure (name them as you which). In each subdirectory, you need a `str.out` file to specify the geometry of the structure and an `energy` file to indicate the energy of that structure. For the format of those files, type `maps -h | more`. Note that the energy must per unit cell of the structure (not the lattice).
Q. When I type `maps` I only get the command line option help. How can I run it?
A. Type `maps -d` to use all the default options. You can also specify any option you want. Either way, the help message will not be displayed and the code will run. If the help is still displayed, it is because there is an error in the options you specified.
Q. All the options I specify are correct by MAPS still displays a help message.
A. Make sure that there is no space between the option, the equal sign and the number following it (e.g. `maps -g=8 -m=3`). Switches (on/off options) do not require any equal sign.
Q. When I superimpose the energies of `fit.out` with the energies obtained from the Monte Carlo code, I can see that they don't match. What is happening?

A. The "energies" given by the Monte Carlo code are in fact $E - \mu x$ so you need to add μx to get the true energy.
Q. How can I verify that the structures do not relax to a superstructure of a different type of lattice (e.g. fcc becoming bcc)?
A. Run the utility called `checkrelax`. It will give you a list all structures along with a measure of the strain they each experienced during relaxation.
Q. When I look at the calculated and fitted energies in the `fit.out` file there are structures where the difference is very large.
A. Make sure that these structures have properly relaxed. A common problem is to start the first-principles calculations with an unrelaxed geometry where the atoms are too close to one another. Restart the energy calculations of the offending structures by using larger starting volumes.
Q. When I try to compile `atat` I get errors messages complaining that the file `strstream.h` does not exist.
A. This is most likely due to a bug in gcc 3.3.1. To work around it, do the following:

```
cd atat/src
ln -s /usr/common/usg/gcc/3.3/include/c++/3.3/backward/strstream strstream.h
# the precise location of this ^ file may depend on your installation.

#In the makefile, change
CXXFLAGS=
#to
CXXFLAGS=-I.

# fix to make the standard commands visible!!!
#In the file machdep.h add the line:

  using namespace std;

#Then, to compile:
cd ..
make
make install
```

6.2 Common mistakes

- Write the energy per atom (or per unit cell of the lattice) in the `*/energy` files. One should write the energy per unit cell of the structure (that is, in the way a first-principles code usually gives it).
- Errors in the `lat.in` file. Use the `corrdump -sym ; more sym.out` command to analyse the symmetry of the lattice and see if it correspond to your expectations. You can also use `corrdump -2=10 -clus ; more clusters.out` to verify the symmetrically distrinct pairs are want you expect them to be.
- When the structures are not generated by MAPS (i.e. you typed them it), beware of typos. MAPS will detect if a structure is incompatible with the lattice but it will not detect another very common mistake: putting the energy and str.out files that correspond to different structures in the same directory. To check for this error, look at the residual plot generated by `mapsrep`.
- When copying the `lat.in` files provided as examples, make sure you make all the required changes. (i) Change the atom names. (ii) Set the lattice parameters (first 3 numbers in the file) to values that are at least as large as the true lattice parameter (otherwise, first-principles code can be unable to properly relax the geometry of the structure is the atoms are too close to one another initially).

Chapter 7

Organization of the Toolkit

- src (source code)
- glue (utilities to interface MAPS with other codes)
- glue/jobctrl (utilities allowing maps to automatically lauch jobs)
- glue/vasp (interface between vasp and maps)
- doc (user guide and documentation regarding the inner workings of the code)