README

MTDG derivative structure code

version 0.0

by Nenian Charles

January 4, 2017

1 Introduction

The approach above is tailored to manipulate configurational degrees of freedom in crystalline structures by controlling the tiling of atomic sites in a parent lattice. The derivative structures (children) are obtained by using an Ising model that assigns site occupancies (-1/+1) to specific positions in a parent lattice. The Ising framework used here is similar to the one employed in cluster expansion methods. [1, 2] While the common cluster expansion method is able to manipulate lattice site occupancies within varying ranges of stoichiometry, being primarily designed to investigate the energetic stability of alloys the most code lacks the ability to explicitly control the local environments of ligand BBUs without significant user involvement. In fact, because the typical cluster expansion is designed to use a minimal set of structures to ascertain the most stable lattice configuration, the method relies on selection rules based on interatomic interactions of nearest/next-nearest neighbors to choose "suitable" clusters. [1, 3, 4] Given the selection criterion for cluster expansion methods it is safe to assume that many formally inequivalent structures are being eliminated.

We are interested in exhaustively sampling the configurational space in a parent lattice with strict stoichiometric control to identifying the effect of anion/cation substitution on space group symmetry. This allows us to simply select resulting lattice configurations based on their ability to support interesting properties such as ferroelectricity, chirality and nonlinear optical activity. Moreover, we have the capability to include addition structural distortions into the derivative configurations, which provide an additional functionality over the prototypical cluster expansion methods.

2 Technical Details

2.1 Requirements

This code is written in Python 2.7 and has the following dependencies:

1. Numpy
2. Python Materials Genomics (pymatgen). Please see http://pymatgen.org/# for installations details.

2.2 Installation

To install the code:

1. Download from https://github.com/nenian/.
2. Save the package in a secure location.
   Example /home/software/MTDGderiv/.
3. Add the path to this folder to your bashrc.
   Example, PATH=$PATH:/home/software/MTDGderiv/

2.3 Execution

To execute the code:

1. Make a new directory where you'd like to execute code and save derivative structures
2. Save the parent structure in folder
3. Create an input file with calculation parameters (example included in download)
4. In the *run folder* execute the command
   main.py -i input-file -p parent-file
5. Outputs will print to current folder
6. The file ORDERING_OUTPUT.txt contains a summary of the calculation settings and output data.
7. Derivative structures are sorted by space group symmetry and stored in folders identified by /space_group-No-#, where # is the space group number
8. Each derivative structure is named after the folder it's in and the count of unique structures. Example, space_group-No-#-str-1.vasp. -str-1 signifies that it is the first unique structure of this space group.
9. If secondary distortions are requested the distorted structures for *each* derivative structure is stored in /space_group-No-#/distortions-str-#

3 Input Variables

To execute this code the user is **required** to provide an input file containing the parameters of the calculation. Keywords can be provided in any order and are not case sensitive. Keywords or characters after # are treated as comments. Most keywords have a default value that is used unless specified otherwise in the input. Keywords can be set in the following ways

Keyword = variable
or
Keyword : variable

A list of keywords and their descriptions are provided in Table 1.

Table 1: Keywords and descriptions

Keyword	type	Default	Description
cenat	string	None	Focal atom around which anion ordering occurs
focalcats	list of integers	None	Specifies which cation to order anions around based on the position in the fractional coordinates list. This is useful in cases where the ordering is restricted to only a few site of a particular atomic specie in the cell. **NB** List index starts from 0.
cations	string	None	List of cations in cell. Each element separated by a space
anions	string	None	List of anions in cell. Each element separated by a space
ionsub	string	None	The new atom to be substituted in
calc_mode	integer	None	Calculation modes are: 0 = apply distortions only 1 = cation substitution 2 = anion substitution

Keyword	type	Default	Description
subratio	real number	None	The ratio of substitution, i.e. ionsub/(cation, anion) Can be a fraction 1/2 or a decimal 0.5.
P1_EVAL	boolean	.FALSE.	Should the code evaluate unique structures with P1 symmetry?
distortions	boolean	.FALSE.	Should the code apply distortions after unique derivatives structures are found?
NPROCS	integer	1	Number of processors to use. Default is serial execution
RAND_SEARCH	boolean	.FALSE.	Turns on random search of configurational space. RAND_SEARCH = .TRUE. improves scalability of code particularly for anion substitution.
max_rand	integer	None	Maximum number of random configurations to sample.
maxbin	integer	None	Maximum number of sample configurations for each space group. This reduces redundancy of the search and improves scalability

4 Post-processing

4.1 plotting

The script **plotting.py** enables the user to quickly plot the statistics of an ordering exercise. Executing plotting.py in the run folder outputs the file **ordering_stats.pdf**, which contains a bar graph and a pie chart. The bar graph illustrates the count of unique derivative structures obtained per space group symmetry. The pie chart shows what percentage of unique structures belong to the seven Bravais lattices.

4.2 geometric isomerism

The script **isomer_configs.py** was designed enables the user to determine the configurations of the anions in heteroanionic systems. The code can only capable

of handling build block units with two unique anion species. To execute the code open the **isomer_configs.py** script in a text editor and modify variables **cati, ani, cutoff** to suitable values for your system. See code header for more details. Once the variables are appropriately set the code will identify *cis*, *trans*, fac, *mer* and *mixed* configurations in the specified polyhedral units. Here *mixed* refers to systems where both *cis* and *trans* or fac and *mer* exists in the same cell. Run the code in the work directory that contains *all* the derivative structures and it will go through each folder then output the file **isomer_conformations.txt**.

References

[1] A. van de Walle and G. Ceder. Automating first-principles phase diagram calculations. *Journal of Phase Equilibria*, 23(4):348–359, 2002.

[2] J.M. Sanchez, F. Ducastelle, and D. Gratias. Generalized cluster description of multicomponent systems. *Physica A: Statistical Mechanics and its Applications*, 128(1):334 – 350, 1984.

[3] Volker Blum, Gus L. W. Hart, Michael J. Walorski, and Alex Zunger. Using genetic algorithms to map first-principles results to model hamiltonians: Application to the generalized ising model for alloys. *Phys. Rev. B*, 72:165113, Oct 2005.

[4] Gus L. W. Hart and Rodney W. Forcade. Algorithm for generating derivative structures. *Phys. Rev. B*, 77:224115, Jun 2008.