MC-EMMA is generally avalible from: <http://pcwww.liverpool.ac.uk/~msd30/software/MC-EMMA.html>**,** under the GNU licence. The version provided is to “future proof” the input files provided in the data repository, should the input format change in a future version of the code.

How to run MC-EMMA:

To run an MC-EMMA calculation, you will need to have a copy of “mc\_emma.py”, “restart\_gulp.py”, your input file and a potential library for GULP calculations. To execute the calculation, run the following command from the command line:

python < [MY\_INPUT]].py > [MY\_OUTPUT]

Guide to MC-EMMA input file:

In this data repository, there are several example input files for MC-EMMA, for the calculations we performed in this work, below is a brief guide to create new input files for MC-EMMA:

1) Import commands: you will need to include the following two import statements:

“from mc\_emma import \* “

“import numpy”

2) creating/importing modules:

If creating new modules in the input file, you will need to specify the following:

ap = x >> a floating point number, to set the realspace size of input modules

rep = [a,b,c] >> required if you wish to tile any of the input modules in x,y or z by a,b or c repeats

making new modules:

To run an MC-EMMA calculation, you will need to create at least one a and one b module. This can be done in one of two ways:

1. Create a module as a new “Atoms” object from the atomic simulation environment (see <https://wiki.fysik.dtu.dk/ase/>, for full documentation), briefly, here is an example of creating a new module:

*Create the atoms object (by setting the atomic formula, then the unit cell then periodic boundary conditions:*

A1=Atoms("Ba4O4",cell=[2.\*ap,2.\*ap,0.5\*ap],pbc=[1,1,1])

*Set the scaled positions for each of the atom*s:

A1.set\_scaled\_positions([[0.,0.,0.5],[0.5,0.,0.5],[0.,0.5,0.5],[0.5,0.5,0.5],[0.25,0.25,0.5],[0.75,0.25,0.5],[0.25,0.75,0.5],[0.75,0.75,0.5]])

*If required tile the new atoms object:*

A1=A1.repeat(rep)

1. By reading in an existing structure file to use as a module:

This can be done with the following command:

A1= read(“my\_module.cif”)

**NOTE**: Each of the modules needs to have the same a and b lattice parameters (although a does not have to be equal to b!), but different c parameters is permitted!

Grouping modules:

Before executing the MC-EMMA calculation, you will need to group your modules in to either “a” or “b” modules, you can do this by setting the following lists (after defining your modules):

AMods=[A1,A2, … ,An]

BMods=[B1,B2, … ,Bn]

The mc function:

This is the main function to execute an MC-EMMA calculation, In order to use, you need to have the following command at the end of your input file:

mc(<options>)

The options are as follows (highlighted in **bold** are the options required for a typical calculation):

**kwds**: a list containing keywords used for a gulp calculation (e.g. kwds=[‘opti conp’])

**opts:** a listing containin options for a gulp calculation (e.g. opts=[‘library library.lib’])

shel: a dictionay object, required if your interatomic potentials require any shells (e.g. shel={“Ca”:0 , “O”:0} to apply shells to Ca and O ions)

**A\_Mods**: the dictionary object to define the “a” modules for the calculation, the key to the dictionary is the type of stackging to use for the calculation: ‘orthorhombic’ indicates that MC-EMMA should use cubic packing to assemble structures. ‘hexagonal’ indicates that MC-EMMA should use close packing. the input should look like: A\_Mods={‘orthorhombic’:AMods}

**B\_Mods**: the dictionary object to defins the “b” modules for the calculation, with the same formatting as for A\_Mods.

**charges**: Dictionary object to indicate the formal charges on each of the input species (e.g. charges={“Ca”:+2, “O”:-2}

**composition:** dictionary object to indicate the empirical formula for the calculation, this needs to be normalised such that the smallest component is equal to 1. (e.g for YBa2Fe3O8 use: composition = {“Y”:1,”Ba”:2,”Fe”:3,”O”:8}, note that the order of the species does not matter.

**sl:** the set of stacking lengths to use for the calculation: can be specified as a range or a list, when using a range, make the largest number 1 higher that the maximum required length (e.g sl= range(1,5) or sl= [1,2,3,4] to generate structures with stacking lengths between 1 and 4 layers)

**startup**: tells MC-EMMA how to start: 2 = start fresh calculation, 3 = restart from a previous run

**smax**: this is how many steps to perform in one run; this exisits so that an entire calculation does not need to be performed in one go, useful for when executing calculations on clusters where it would take longer than the allowed cpu time! (after the first run set startup = 3 to continue the calculation)

**rmax**: the parameter used to dertermine when to stop the calculation

**smin**: the minimum number of structures required before stopping a calculation

ratt\_dist : a parameter by which atoms are randomly displaced from their starting positions before starting a geometry relaxation. The number specified is a standard deviation in Angstroms. (e.g ratt\_dist= 0.1)

**red\_T**: the reduced temperature parameter for the Monte-Carlo loop

delay (typically zero): if this is specified as non-zero, MC-EMMA will only use swap types 1 and 2 for n steps after accepting a new structure: this can be useful to allow a calculation to settle out after accepting a swap.

pert: this is a list to indicate which swaps to use (as defined in the paper), if it is not specified, MC-EMMA will use default values) and the weighting applied to each swap type, note that you do not have to specify all 6 possible swap types!. (e.g.pert = ([‘T1’,10],[‘T2’,5] … etc )

min\_bond: a number indicating the minimum acceptable value for a bond in proposed structures: this avoids creating structures where atoms are placing ontop of each other

res: either equals ‘T’ (default) or ‘F’, should be set to ‘T’, set so that MC-EMMA will continue counting how many steps since the lowest energy structure was found, this is used for convergence. If for some reason you wish to restart the calculation resetting this count set equal to ‘F’.

kmax: you should not need to alter this; this is the total number of attempts MC-EMMA will try to make a random structure with a given module set.

tmax: when initialising the calculation, this is the number of attempts MC-EMMA will try to generate a structure at a given stacking length before increasing (it will start with the smallest and move towards the largest). the number should be increased if MC-EMMA returns an error message indicating it could not make a starting structure. Although increasing this may lead to a significant increase in computing time to generate the initial structure. Note: the other reason for this error message may be that it is not possible to generate a structure with your input modules and/or composition (please double check both of these manually to ensure that you input is correct!)

r\_gulp: boolian; set to True if you wish to split your gulp calculation into two steps (e.g if you wish to start a calculation with 100 conjugate gradient steps before reverting to a bfgs minimiser)

if r\_gulp is set to True, the following additional commands are required:

target: a string to indicate the name of the gulp restart file MC-EMMA should look for (e.g. ‘temp.res’)

restart= name for the gulp input file for MC-EMMA to use to restart gulp runs (e.g ‘restart.res’)

head = name for a text file for MC-EMMA to use to create a header for restarting gulp calculations (e.g ‘head.txt’)

r\_kwrds: a string to contain the keywords for the restarted gulp calculation (e.g. r\_kerds= ‘opti conp’)

r\_opts: a string to indicate gulp options for restarted gulp calculations, note that as this is a python string to write directly to a file, each option has to end with a “\n”. (e.g. (r\_opts = ‘library library.lib\nmaxcyc 10000’).