**Quick note:** This guide is assuming that you have already completed the install instructions for your machine, and have successfully setup MC-EMMA, FUSE, ChemDASH and pymatgen.

**1. Example 1. The structure of SrTiO3 using MC-EMMA (this is the longest of the worked examples)**

First we need to create a copy of the MC-EMMA input template where we’re going to run the example.

In the “hands\_on\_files” folder there is an empty folder called “compute\_probe\_structures”. go into this folder and create a new directory called “SrTiO3”.

The “hands\_on\_files”, contains the folder “input templates”, this contains three folders, each containing an example input for MC-EMMA, FUSE and ChemDASH.

Create a copy of the “mc-emma” folder, and move it into the SrTiO3 folder that you have created above.

This folder contains the example input file (perovskite\_example.py), the library for the interatomic potentials (lib2.lib) and a series of cifs, each containing a module for MC-EMMA.

The input file

Now we’re going to take a look at the input file for MC-EMMA and recap, open the example file called perovskite\_example.py .

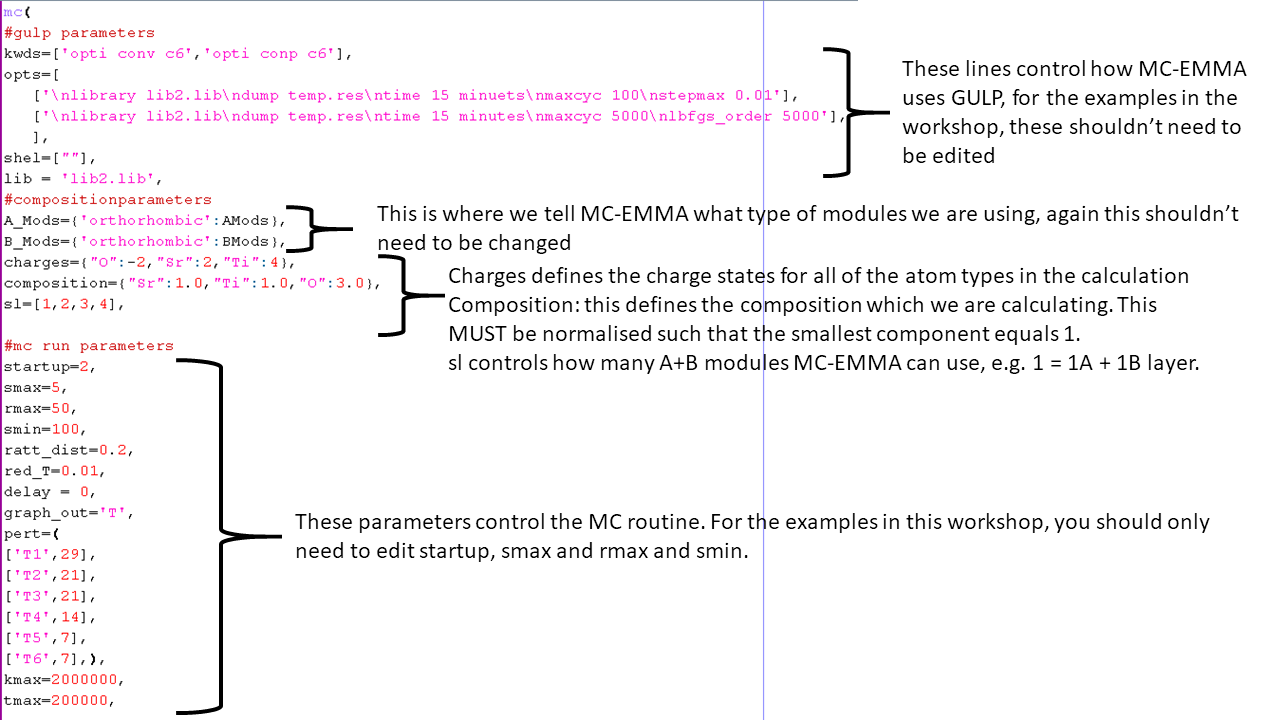
the first part of the input looks like this:

Diagram

Description automatically generated

This is setup to work with the module set provided, and should not need to be edited.

The second half looks like this:

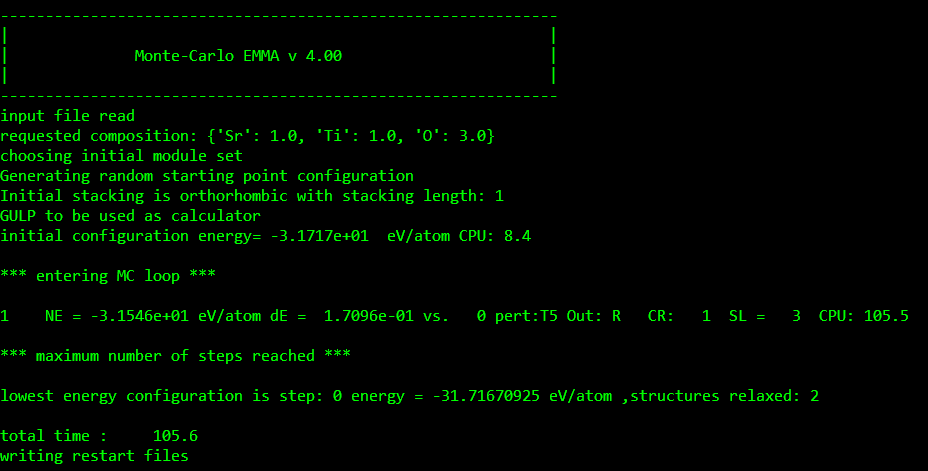


Again, the majority of this input should not need to be changed for this workshop. For running SrTiO3 we are going to edit the parameters smax, smin and rmax. smax controls how many structures are generated once we start the calculation, note that when starting a fresh calculation, there will be one additional structure for the initial random guess (so smax = 5 will generate 6 structures), set smax to 1. rmax controls the convergence of the MC routine, MC-EMMA counts how many structures have been generated, since it created the lowest energy structure it has seen so far, in the output file, this is the parameter CR. Once CR is equal to rmax, the calculation will stop. For this example, set rmax = 5. smin adds a condition to the routine: MC-EMMA must complete this many structures before it is allowed to converge (this overrides the condition above with CR and rmax), for this example, set smin = 1. Save the changes to the file and exit.

To run the calculation, open a command prompt window and navigate to the directory containing your input file, and type:

“python perovskite\_example.py”

and let this run, you should see output which looks like this (this will take a couple of minutes to run):

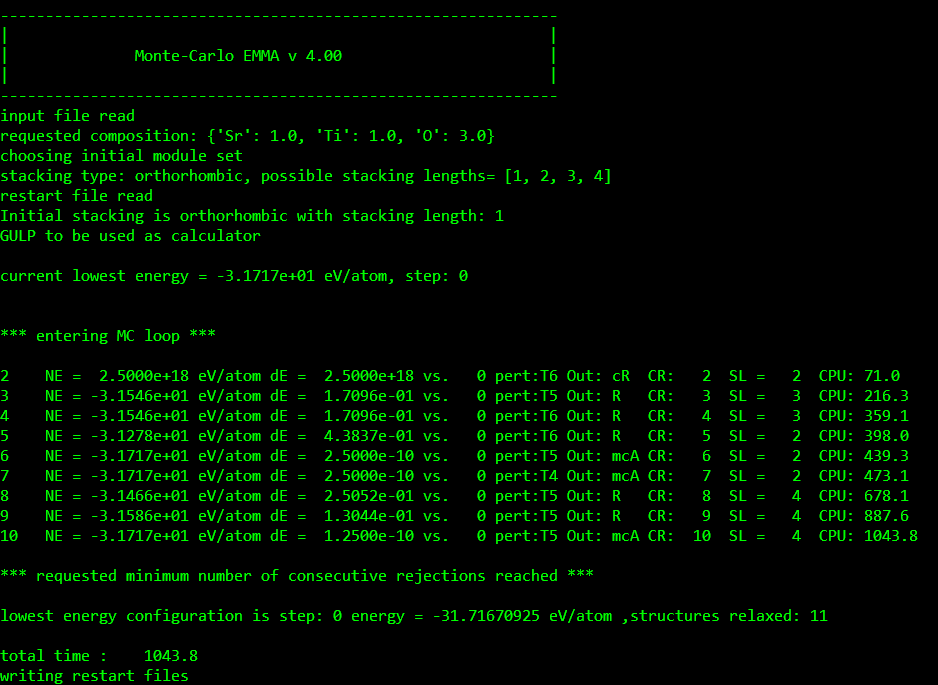


This output is tells us the calculation has stopped because we have reached the maximum number of structures from the input file (smax). To complete the calculation, we need to re-run it until the parameter CR equals rmax (set to 5). To restart the calculation, open input file (perovskite\_example.py) and change smax to equal 20. To tell MC-EMMA that we wish to restart a previous calculation, change the parameter startup to equal 3. Save the changes and close the file.

you can then restart the calculation by typing:

“python perovskite\_example.py”

This may take a little while to run, but you should get output looking like this (this will vary, but this took about 17 minutes to run on my laptop, the time in the output file is reported in seconds):



As you can see, once CR reaches 10, the calculation converges (when I ran this example, rmax was set to 10), and stops with the message (note as outlined above, your calculation should stop once CR reaches 5, as we have set rmax = 5 in the input file):

\*\*\* requested minimum number of consecutive rejections reached \*\*\*

and gives us the energy of the lowest energy structure.

Now this is completed, we can have a look at the files which MC-EMMA writes:

There is a folder called “steps”. this contains the cif of the structure for each step as computed by GULP, where the name of the file corresponds to the step number in the output file. Note that the numbering starts from 0, which is the initial random structure.

Additionally, there is the folder called “accepted”, this contains each structure which was accepted by the MC routine, starting from the initial structure, labelled as for the “steps folder”. Where a structure is accepted with an increase in energy, the name of the cif is appended with “MCA”.

There is also a csv file written called “log\_file”, this creates a log of each structure generated, with the energy of the structure, with the module sequence it was generated from, and the module sequence after the MC move was applied, along with the outcome of the MC step.

Lastly, there is the cif “lowest\_energy\_structure”. This cif corresponds to the lowest energy structure that MC-EMMA generates, with the step number at the bottom of the output file, in my example this is the initial structure.

In my example I get:

Chart

Description automatically generated

**Note** that this is a distorted from of the SrTiO3 perovskite. In the force field which is used in the workshop, the distorted and non-distorted structures are very close in energy, and with the relatively small number of structures we are calculating, you may end up with either structure.

We will come back to this structure, once we have completed the rest of the examples with FUSE and ChemDASH.

**Example 2. The structure of SrTiO3 with FUSE**

In your compute\_probe\_structures folder, in your SrTiO3 folder, create a copy of the “fuse” folder from the input\_examples.

This folder should now contain the example input file (input\_gulp\_example.py) and the interatomic potentials library (lib2.lib).

Let’s recap how the input file for FUSE looks:

Diagram

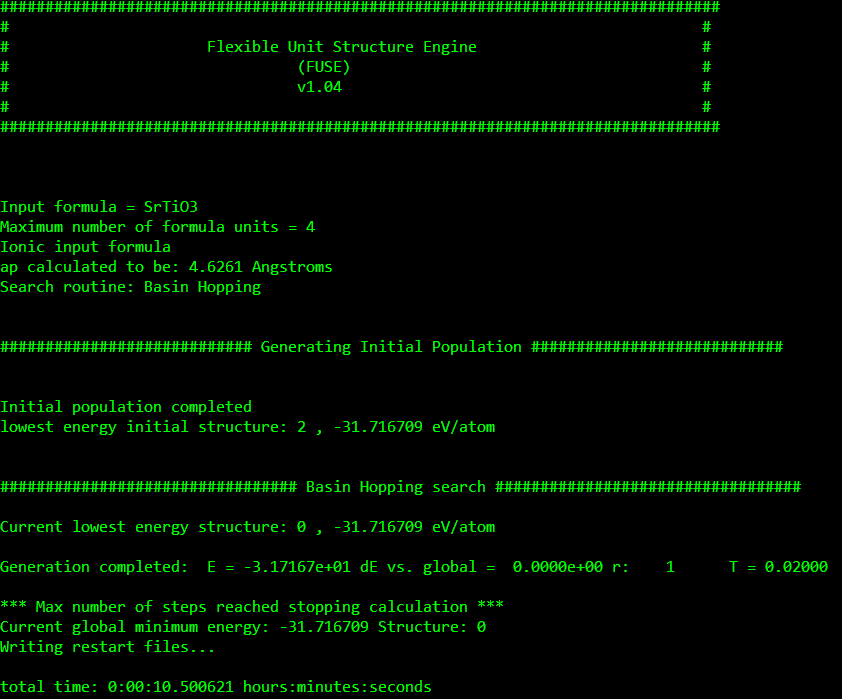
Description automatically generated

This file is already setup to run SrTiO3. For the first instance we’re just going to compute the first 6 structures. To do this change iterations to equal 6. In the version circulated. To recap, each entry in the composition should have the format ‘X’:[Y,Z] where X is the element symbol, Y is the number of that element in the empirical formula and Z is the formal charge state.

To run the example, in a command prompt window, navigate to the directory containing the input\_gulp\_example.py and then type:

“python input\_gulp\_example.py”

After letting it run, you will get output looking something like this (note that because FUSE is capable of building smaller structures than MC-EMMA, it completes the calculation a lot faster, on my machine ~ 10 seconds):



This calculation has created and found the energy for the initial population of structures and the first step of the basin hopping search. We are now going to restart the calculation and run it until the parameter r reaches rmax, note, these are exactly equivalent to the CR and rmax from MC-EMMA.

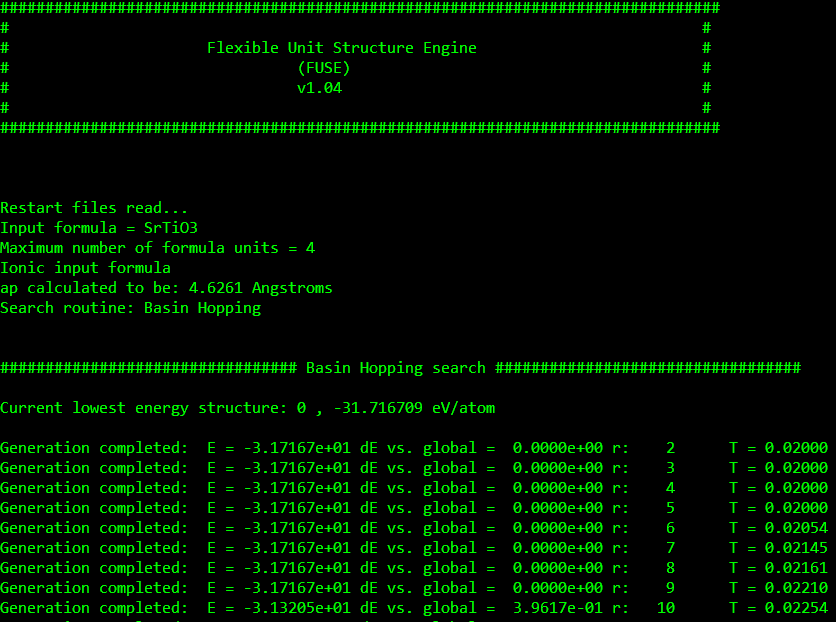
Reopen the input\_gulp\_example.py.

To tell FUSE to restart the calculation, change the parameter “restart” to equal True (note the capital letter is important!). Then to make sure that the routine can reach the rmax parameter, we need to increase the number of structures we will compute this time, change iterations to 100. Save the file and exit.

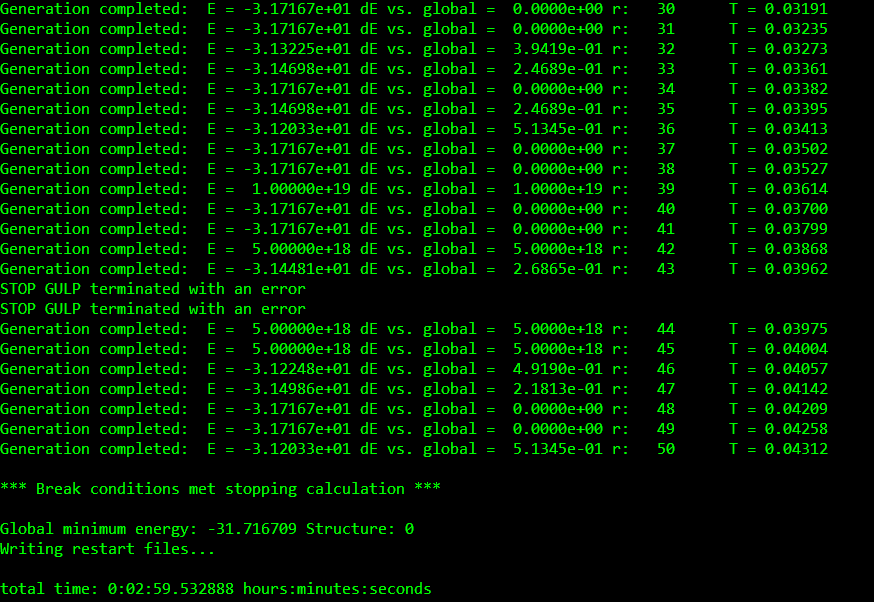
now to restart the calculation, type:

“python input\_gulp\_example.py”

After letting this run, the start of output should look something like (this will take a few minutes):



And the end of the output like:



The last few lines as with MC-EMMA, tell us that the search routine has met its convergence threshold and stopped, if you do not see this message, please restart the calculation again until you do! (for this example, this is very unlikely). Note in my example partway down the output, there is an error message: “STOP GULP terminated with an error”, this is telling you that GULP has crashed on an individual structure, this is expected to happen occasionally, if it is only happening occasionally, there is nothing to worry about! Although please do raise it with the demonstrators if this is happening with every structure something may not be configured correctly in your input file!

This output also tells you the lowest energy structure obtained, along with the structure number and energy.

During the run, FUSE will create a folder called “structures” which will contain all of the structure generated during a run.

In this folder are all of the structures which FUSE has generated, there are two types of label for a cif:

1) if the file starts with “I” then this was part of the initial structure set, theseare numbered in the order in which they are generated.

2) if the file starts with a “S” it was generated as part of the search routine. These are numbered starting from zero, where structure zero is the lowest energy structure from the initial population, that became the basis for the basin hopping search.

In the main directory where you ran the calculation, the other output files to look at are:

“run\_output.png” : this is a graph of the basin hopping search, it will look something like this:

Chart, scatter chart

Description automatically generated

This graph is indicating what the search routine has done. Each red point indicates the energy of a generated structure. The green line indicates the energy of the structure which FUSE is using as the basis to generate new structures, the green line starts at the structure where the basin hopping search begins. The black line indicates the “thermal parameter” T from the search – the higher the number the greater the change of FUSE accepting a structure which increases the energy, when FUSE accepts a structure which decreases the energy this is reset back to the starting value. All of the information used to plot this graph is available in the output file “graph\_output.csv”, note that this file also contains the name of the cif which corresponds to each step in the search along with the move type used to generate the structure.

Lastly, as with MC-EMMA, the lowest energy structure from the search is written out in the file “lowest\_energy\_structure.cif”. In my case I get:

Chart, radar chart

Description automatically generated

In my case, this corresponds to the high symmetry cubic structure of SrTiO3. Note that as this is using the same interatomic potentials as the MC-EMMA example, you may also end up with the distorted structure. Additionally, as there are many ways for FUSE to create a structure which relaxes into the structure of SrTiO3 it is also perfectly possible to arrive at a structure with an odd triclinic setting as opposed to one which is essentially cubic, this is normal.

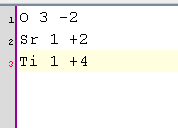
Now we have found the structure for SrTiO3 with FUSE, we can put this to one side, and now do the same for ChemDASH

**Example 3. the structure of SrTiO3 with ChemDASH**

As with the previous two examples, in your SrTiO3 folder, create a copy of the chemdash folder from the input examples.

For chemdash, each of the input files needs to have the same name before the file extension. The input is made up of three files with the extensions: .atoms, .input and .lib

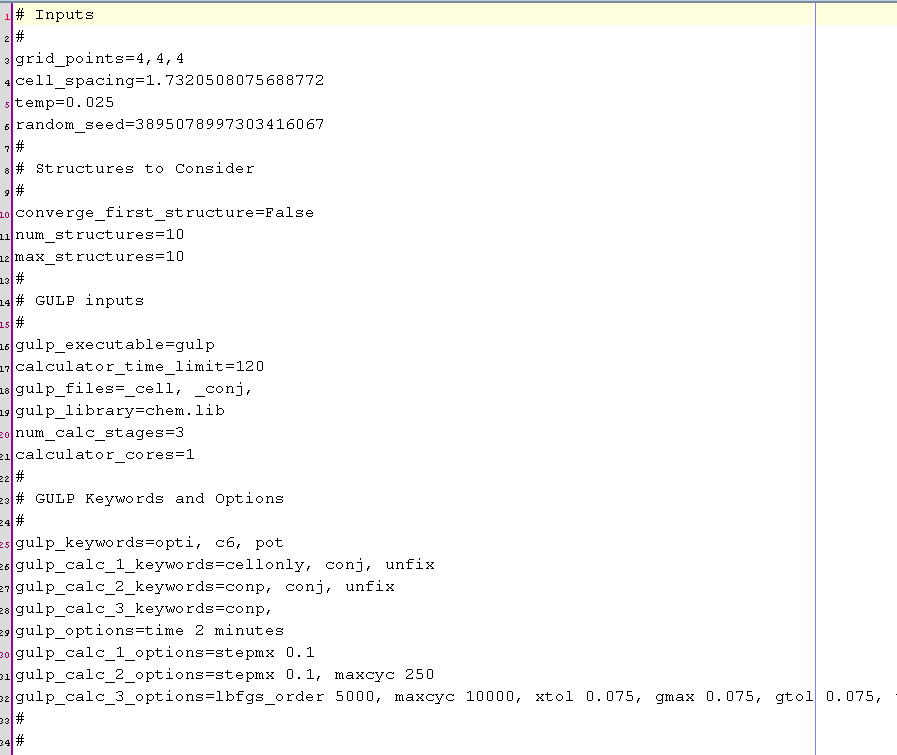
the .atoms file is simple and looks like this:



As a reminder, this file contains information on the atoms which are to be used in ChemDASH, in the format: <Symbol> <number of atoms> <charge state>.

This file has already been configured for the SrTiO3 example.

the second is the .input file:



The only lines which may need editing during this workshop are the “grid\_points”, “num\_structures” and “max\_structures”. The “grid\_points” parameter sets the number of grid points used for each of the anion and cation lattices, for this example, 4 x 4 x 4 is fine. “num\_structures”, is the number of structures which will be computed when we start the calculation, set this to 1. “max\_structures” indicates the maximum number of structures for ChemDASH to compute while looking for a probe structure, set this to 25. You can now save and close the file.

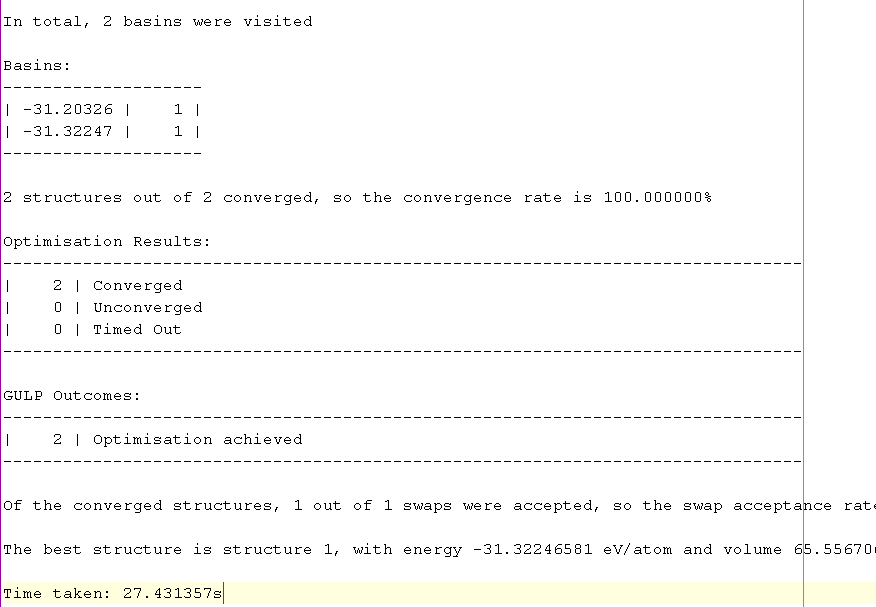
the .lib file contains the interatomic potentials to be used in the calculation, these are identical to those used in the FUSE and MC-EMMA examples above.

Note, the folder called “chemdash” contains a copy of the ChemDASH code itself, and it is necessary to have a copy of it with each input until we have been able to write a setup script to install it (this is in development).

To run the example, in a command prompt window, navigate to the directory containing your chem.atoms, chem.input and chem.lib files. Then type:

“python chemdash chem”

note, that when running, ChemDASH does not print any output. In the folder where you have run ChemDASH, you will see that a number of new files have been produced, all of the .gin, .got and .res files are the input / output files for GULP for each structure (two in this case). You can observe what ChemDASH has done by opening the new file chem.chemdash in your text editor. In particular at the end of the file, you should see a summary of the calculation, including the current best energy and the time taken to run the calculation (in my case ~ 30 seconds):



As we set that we want to run a total of 25 structures, we now need to set ChemDASH to restart. to do this, reopen the .input file. Under the line “max\_structures”, insert the line:

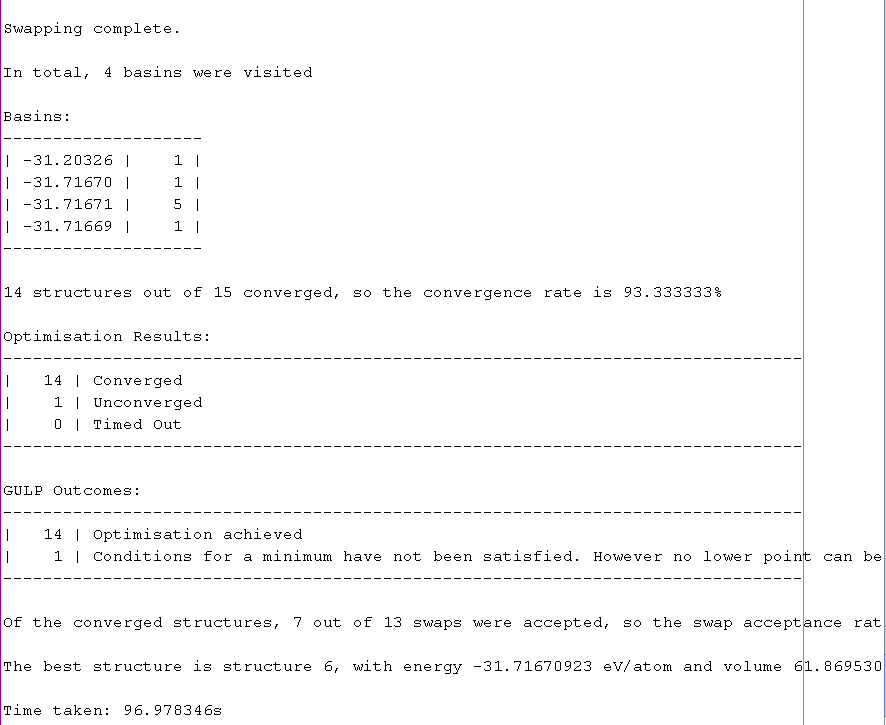
“restart=True”

and to make sure that we now generate and compute enough structures to complete our calculation, set num\_structures equal to 25. Save and close the file.

you can now restart the calculation by typing:

“python chemdash chem”

this will then run and append output to the .input file. In my case, this ran for about 100 seconds:



Which now indicates that a total of 15 unique structures were visited and the lowest energy. In my case, my run found the cubic structure:

Chart, radar chart

Description automatically generated

In ChemDASH, the lowest energy structure is then written to the file “best.cif”, you will use this along with the structures from examples 1 & 2 in example 4 below to assemble the convex hull for the know phases in the Y-Sr-Ti-O phase field.

**Note**: Occasionally, ChemDASH will leave potential vacancy sites in the final cif, these will appear with the symbol “XX”. If this occurs, you will need to remove them before proceeding to example 4. You can do this by opening the cif in your text editor and deleting the lines in the structure with the symbol “XX”, you can then save and close the file.

**Example 4. Assembling the basic convex hull for Y-Sr-Ti-O**

To Assemble the convex hull for the Y-Sr-Ti-O phase field, we need to first gather the lowest energy cifs from the previous examples into one place.

In the attached files for the workshop, in the “hands\_on\_files” there is a folder called “Y-Sr-Ti-O\_convex\_hull”. In here we need to create a directory in which to place the cifs from examples 1-3. In my case I have labelled it by composition (SrTiO3). Copy all of your lowest energy structure cifs into this directory.

The convex hull directory, additionally contains a folder called “ICSD”, which has folders for the ordered structures from the ICSD used in the original publication for FUSE (<https://pubs.rsc.org/en/content/articlelanding/2018/fd/c8fd00045j#!divAbstract>). This also contains a copy of the “lib2.lib” library file and a python script called “process\_hull.py”.

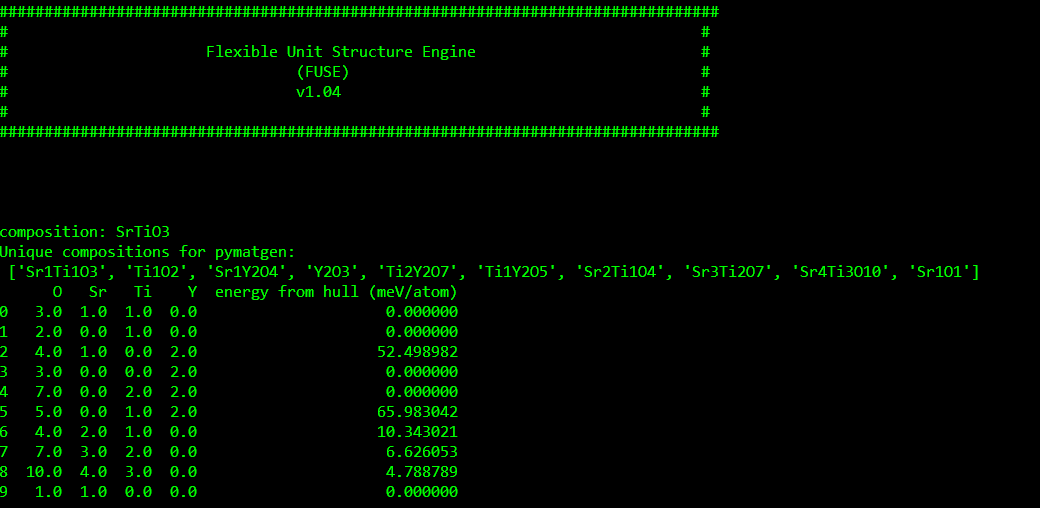
The “process\_hull.py” is a script which will combine your output cifs with the structures in the ICSD folder, get the energy of each structure and then write out input files for pymatgen, note where more than one structure is present for a unique composition, the script will use the structure with the lowest energy.

In the second half of the script, it will then run all of the created pymatgen files, process their outputs and write two csv files containing your results.

To use the script, in a command prompt, navigate to the directory containing the “process\_hull.py” script, and run it by typing:

“python process\_hull.py”

This may take a couple of minutes to run, you should get output which looks like this:

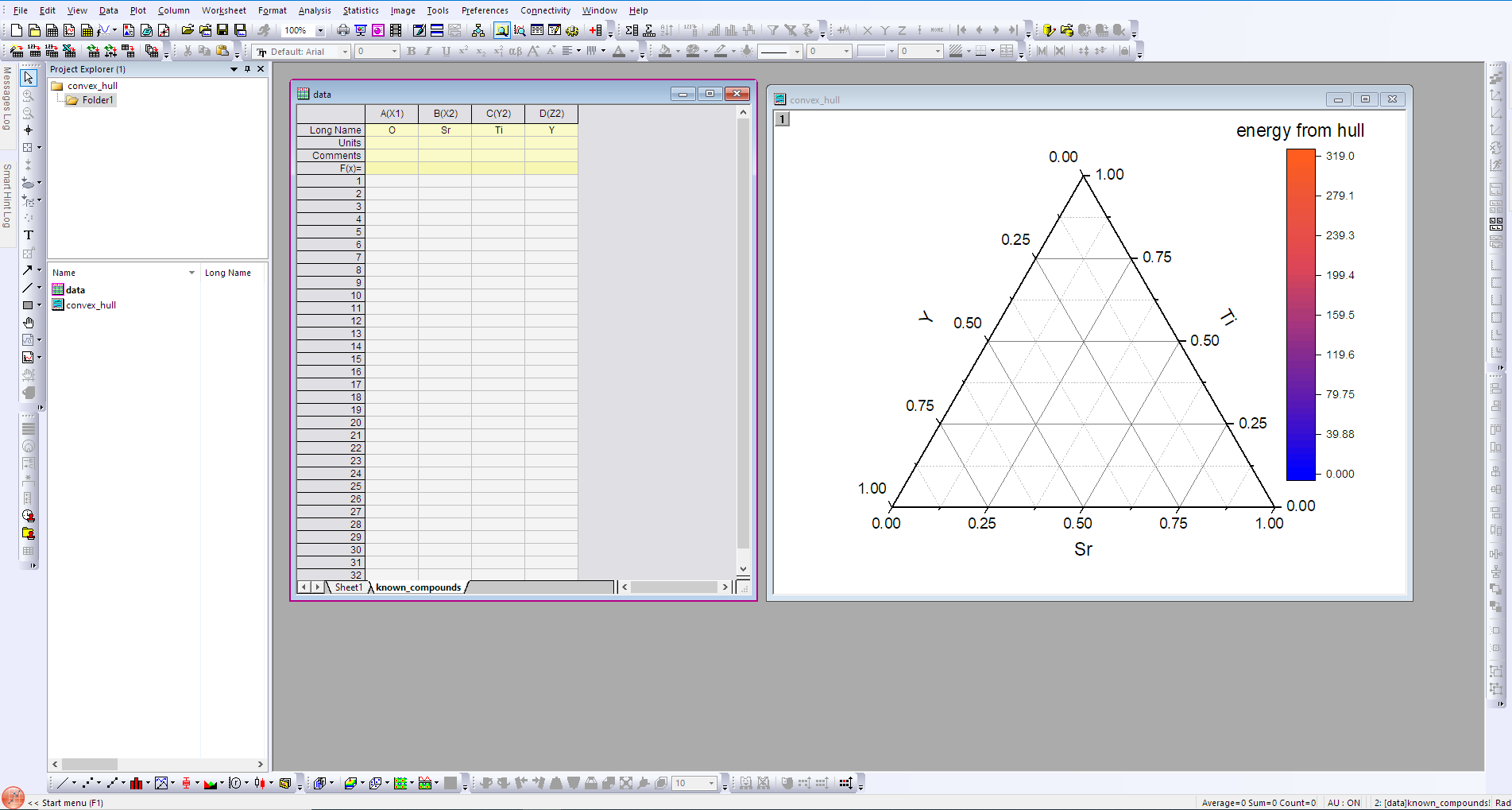


(The FUSE header appears, as the process\_hull.py uses some of the functions from FUSE to obtain the energies). In the output, the “composition” line(s) will confirm the names of the directories, which are being used (excluding the ICSD folder). It then prints a list of the unique compositions which are used for the pymatgen files, and then it prints a table yielding the compositions and their energies above the convex hull (this is for information only).

The script creates a new directory called “pymatgen”, this contains all of the input and output files for pymatgen, these files are numbered in the same order as the output in the printed data table.

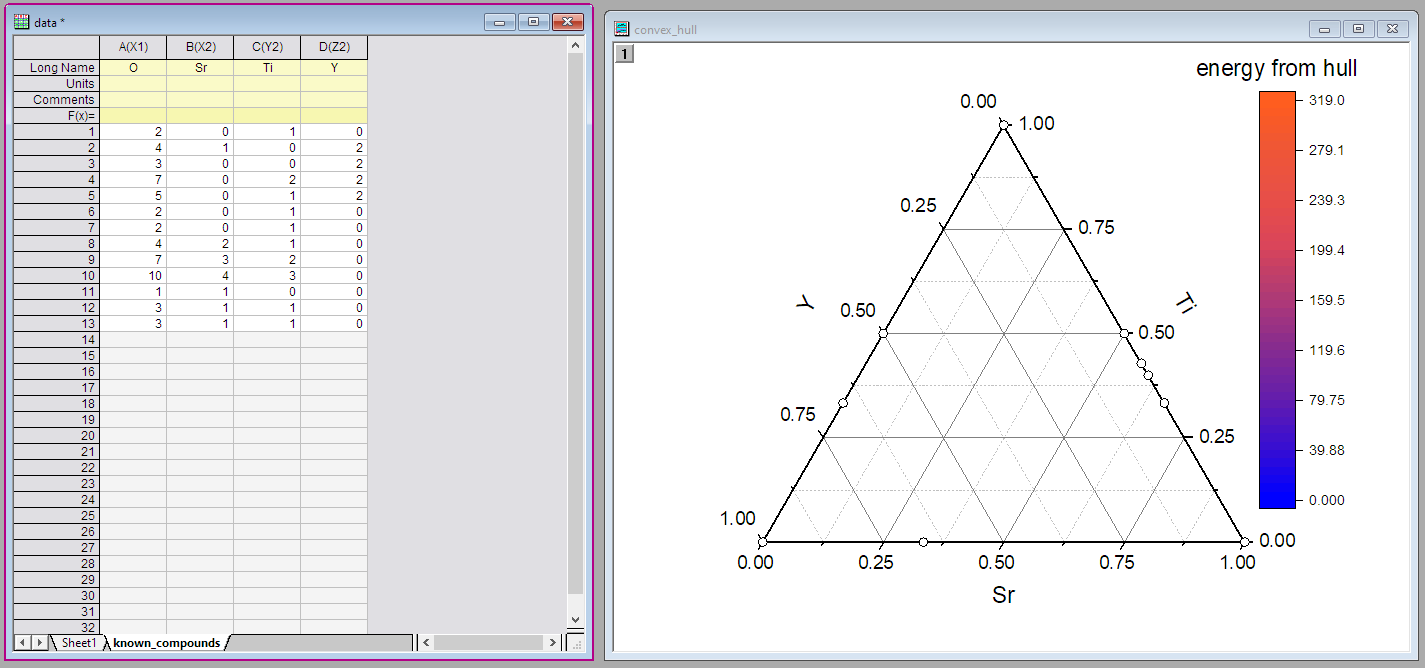
In the same folder as the “process\_hull.py”, the script should’ve written two csv files called “hull” and “known\_materials”. The “hull.csv” file contains the data table which is printed out by the “process\_hull.py”, and “known\_materials.csv” contains each of the unique compositions found in the ICSD folder. You can then use these csv files to create a ternary colourmap, with the x, y and z axes being the Sr, Ti and Y content, and the energy above the convex hull to create the colour plot. You can additionally use the entries in the “known\_compounds.csv” to make the points on your ternary of your known compounds.

**Plotting the convex hull with Origin**: In the “hands\_on\_files” folder, there is an origin project called “convex\_hull.opj”. If you open this in origin you should see the following:



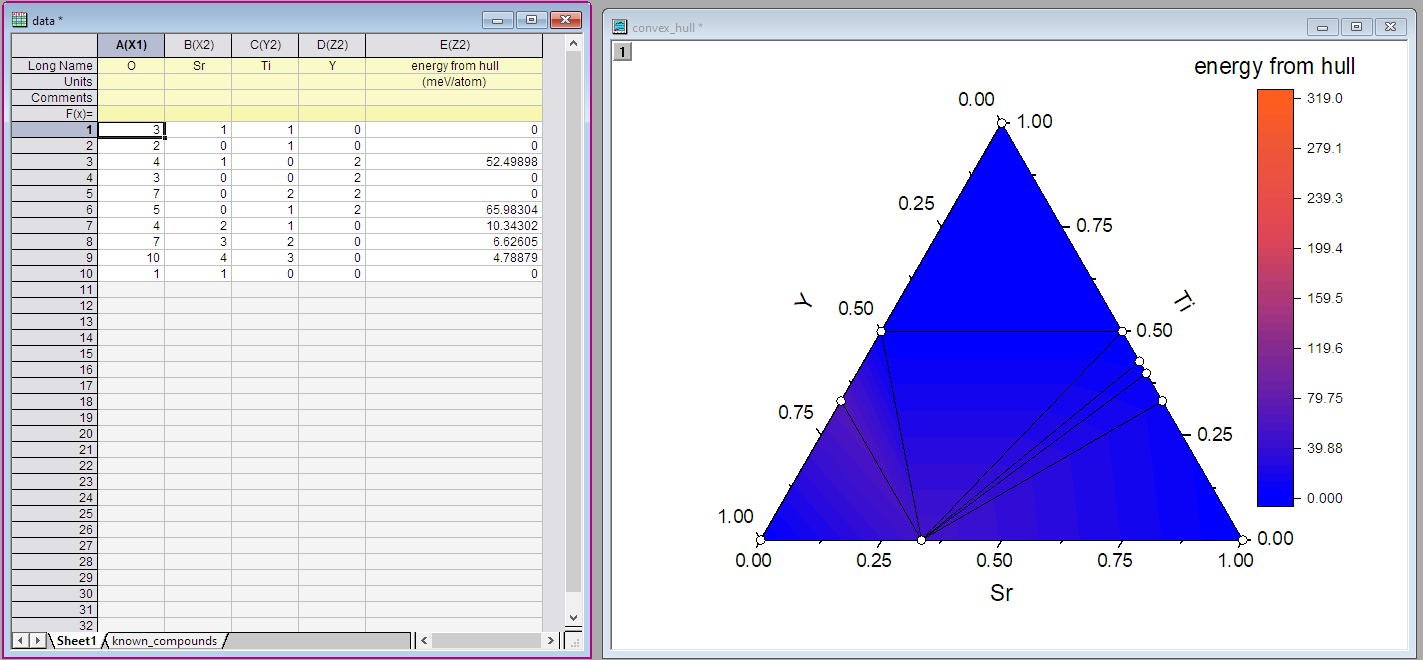
This is a template to plot the convex hull in Origin. The work book contains two tabs, “sheet 1” and “known compounds”.

First, copy the contents of the “known\_compounds.csv” (excluding the column headers) into the known compounds sheet:



This will then plot each of the known compounds on the ternary as white dots.

Secondly into “sheet 1” copy the contents of the “hull.csv”, and this should then plot the colour map, again, you do not need to copy the column headers:

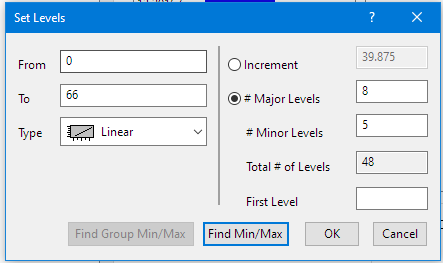


The last thing to do is rescale the colour map. To do this, double click anywhere within the ternary plot, and you should get the following window pop up:

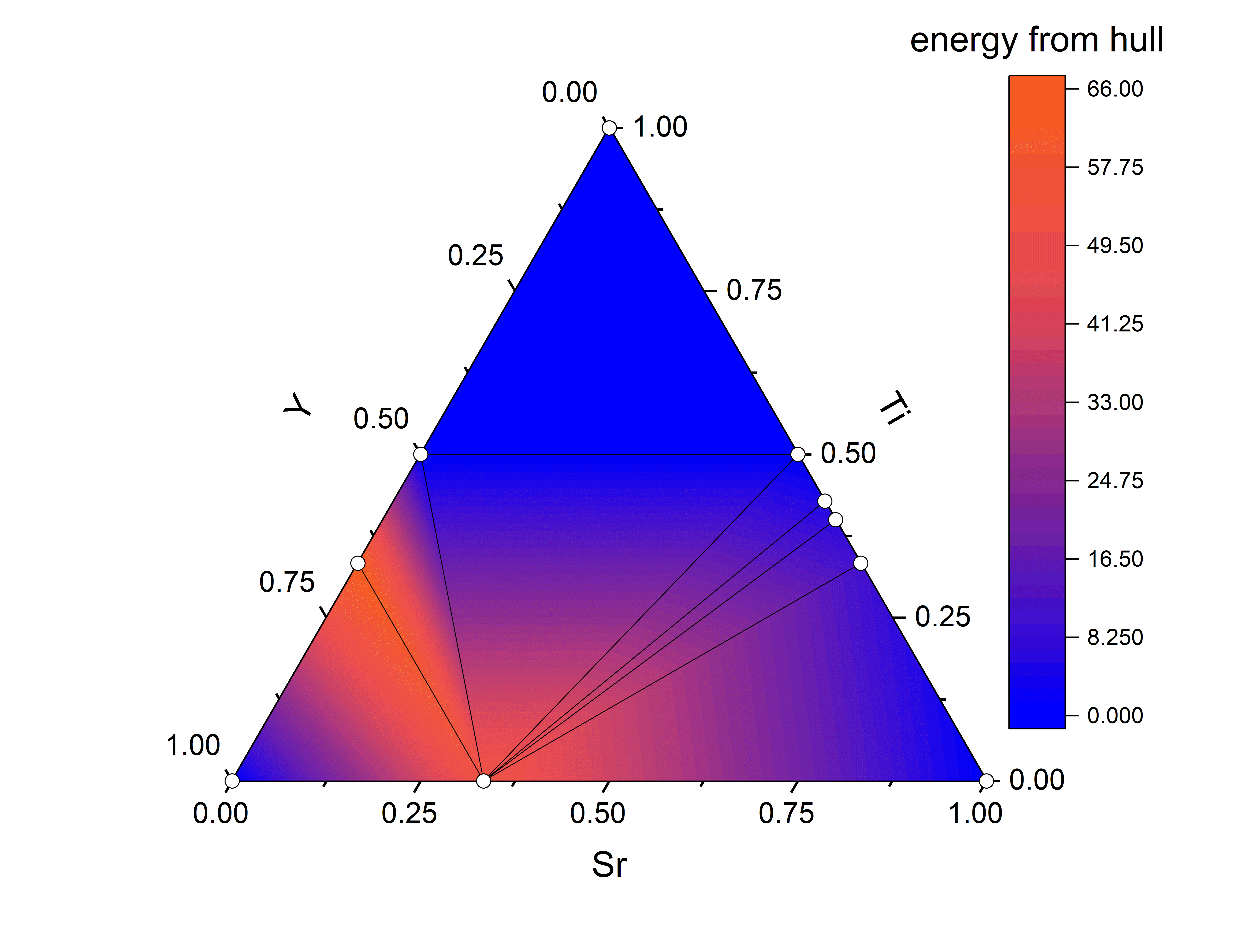
Graphical user interface, application

Description automatically generated

now click on the “level” column highlighted above, and press find min/max:



And then click “OK” to close the window, and click “OK” again to close the plot details window. You should now have a convex hull plot which looks like this: (which you can export from origin “ctrl+g”.



**Additional work: exploring Quaternary compositions.**

With the remaining time, you can now move to add some quaternary compositions to your convex hull. Below is a list of some suggested quaternary compositions you may wish to investigate (these are the compositions used in the FUSE paper linked above), note, this list is also included in the “suggested\_compositions.txt” file:

Sr2Ti8Y2O21

SrTi3Y2O10

Sr2Ti4Y6O19

SrTiY4O9

Sr2Ti4Y2O13

SrTiY2O6

Sr4Ti6Y2O19

Sr2Ti2Y2O9

Sr4Ti2Y6O17

Sr4Ti2Y2O11

Sr3TiY2O8

Sr8Ti2Y2O15

Sr6Ti4Y2O17

For this, we recommend that you use one CSP code per composition, so that you will have the time to explore a few different compositions (though this is only a recommendation...).

To run any of these compositions, in the compute probe structures folder, create a new directory and in it create a copy of one of the template input files for one of the codes. Then edit the input file(s) to alter the composition. A couple of points to bear in mind:

For **MC-EMMA**: you need to make sure that you normalise the composition such that the smallest component is equal to 1.0 if it not already.

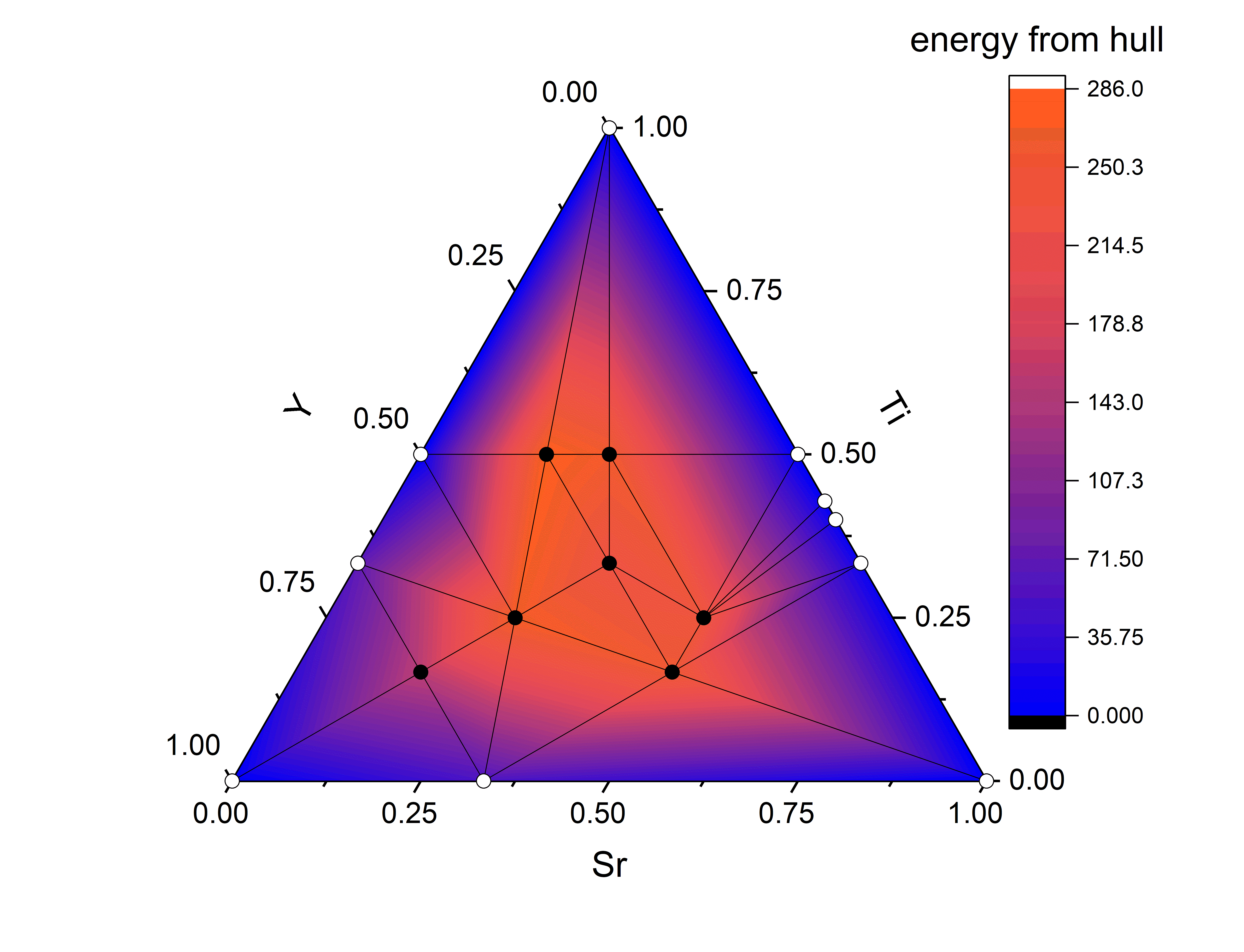
For **FUSE**: if the total number of atoms in the composition exceeds 20, you will need to edit both the “imax\_atoms” and “max\_atoms” parameters in the input file to be equal to the number of atoms in the formula unit

For **ChemDASH**: You will need to bear in mind the grid size you are using in order to make sure that you are not creating structures which are overly dense!

With the timescale for the workshop, we recommend limiting your probe structure searches to ~ 50 structures for ChemDASH or FUSE or ~ 10 + the initial guess for MC-EMMA (smax = 10). This will give you the time to generate some probe structures for a few different compositions.

Once you have explored a few different compositions, for each one, you can create a new folder in the Y-Sr-Ti-O\_convex\_hull folder and copy in your probe structures, as for the SrTiO3 example and rerun the “process\_hull.py”, this will then include your new probe structures in your convex hull and allow you to add them to your plot!

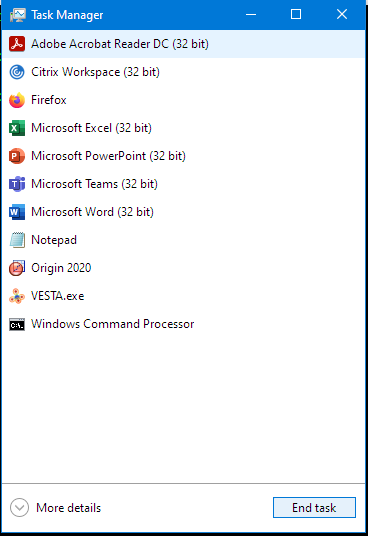
**Note:** To give context, when producing the probe structure search in the paper for FUSE, I computed ~ 1.25M structures, over several weeks using a high performance computing cluster, so we are not expecting you to be able to reproduce it on a desktop in an afternoon! As an example, in an afternoon using the recommendations above (and having developed most of the code) I was able to obtain the following plot:



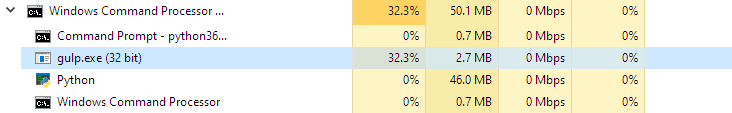
**Troubleshooting:**

**GULP is taking a very long time…**

When using GULP in windows, very occasionally, will get stuck (typically if it has been on one structure for more than 10 minutes). To manually stop GULP, press ctrl + shift + esc to open the task manager, to start, you want to look at the minimal version, if it doesn’t look something like this, you will need to click on “fewer details” at the bottom of the window to collapse the window:



click on the “Windows Command Processor” and then click “More Details”. Then expand the list by clicking on the small arrow to the left, and you should get something like this:



Click on “gulp.exe” and then press “End task” in the bottom right of the Window, this will terminate the current GULP calculation without aborting the code which is using it.

**The “process\_gull.py” script doesn’t run**

On some systems, we’ve had issues with the process hull script not running successfully. We have gotten around this by installing an older version of pymatgen. You can uninstall the current version by typing:

“Conda uninstall pymatgen”

Once this has run, we then need to install the version of pymatgen which we used when testing this workshop. You can do this by typing:

“python -m pip install pymatgen==2021.2.8.1”

And then:

“python -m pip install ase”

This should then allow you to run process\_hull.py correctly.