

SSCHA School 2023

Hands-on-session1:

First SSCHA simulations: free energy and
structural relaxations

Diego Martinez

May 5, 2023

Chapter 1

Installing SSCHA

1.1 Getting SSCHA

In order to use the SSCHA code you will need to download both the CellConstructor package and the python-sscha package. These packages can be downloaded directly from the following github pages:

`github.com/SSCHAcode/CellConstructor`

`github.com/SSCHAcode/python-sscha`

In case you want to use the force field calculator used in the SnTe tutorial, you will need to download as well the F3ToyModel package from this link:

`github.com/SSCHAcode/F3ToyModel`

The commands for downloading this using git is

```
git@github.com:SSCHAcode/CellConstructor.git
```

```
git@github.com:SSCHAcode/python-sscha.git
```

If you are using GitHub CLI

```
gh repo clone SSCHAcode/CellConstructor
```

```
gh repo clone SSCHAcode/python-sscha
```

1.2 Requirements for installation

In order to install both CellConstructor and python-sscha packages, you need to install previously python and several python packages. As python-sscha depends on CellConstructor, the former will not work unless the latter is installed. As

the SSCHA code also is partly written in Fortran, you will also need a Fortran compiler as well as Lapack and Blas libraries.

The full list of dependencies to install CellConstructor and python-sscha packages is:

- python
- numpy
- matplotlib
- Lapack
- Blas
- A FORTRAN compiler. For example gfortran.
- ASE (Atomic Simulation Environment)
- SPGLIB

All the needed python dependencies can be easily installed for CellConstructor or python-sscha by simply running

```
pip install -r requirements.txt
```

in the folder of each package.

To install the code, a fortran compiler is required. We recommend gfortran. For example, on Ubuntu, the fortran prerequisites may be installed with:

```
sudo apt-get install libblas-dev liblapack-dev liblapacke-dev gfortran
```

for RedHat

```
sudo dnf install libblas-dev liblapack-dev liblapacke-dev gfortran
```

for Arch-base linux

```
sudo pacman -S libblas-dev liblapack-dev liblapacke-dev gfortran
```

It is also recommended to use anaconda, conda or miniconda for development security. The instructions for the miniconda installation can be found here:

<https://docs.conda.io/en/latest/miniconda.html>

Once installed we can create a new environment for SSCHA with

```
conda create --name SSCHA
```

and activate the environment with

```
conda activate SSCHA
```

then installing the prerequisites with conda is

```
conda install numpy
```

1.3 Installation

To install the CellConstructor and python-sscha packages it is recommended to use the last version of anaconda-python2, which comes with all the updated numpy and matplotlib packages already compiled to work in parallel. Moreover the last version of matplotlib will allow the user to modify the plots after they are produced.

Once all the dependencies have been installed, the CellConstructor and python-sscha codes can be easily installed from the command line as:

```
python setup.py install
```

This command must be executed in the directory where the setup.py script is, inside the CellConstructor and python-sscha folders. If you installed python in a system directory, administration rights may be requested (add a sudo before the command).

Installing CellConstructor and python-sscha in clusters may be more tricky and one needs to adapt the setup.py to the cluster characteristics. For instance, if you use the intel compiler, you need to delete the lapack linking from the setup.py and include -mkl. Note that you must force to use the same linker as the one used for the compilation. For example, specific setup.py scripts are provided with the distribution to install CellConstructor easily in FOSS or INTEL clusters.

1.3.1 Installation through pip

Alternatively, both CellConstructor and python-sscha can be installed through pip simply as:

```
pip install CellConstructor
```

and

```
pip install python-sscha
```

1.3.2 Installation through docker

If you are not able to compile the code, or you want to use it on a cluster, where compilation could be cumbersome, we provide a docker container with the SSCHA code already compiled and installed.

You can download it from the docker hub. To run the docker command you need Docker already installed.

```
docker pull mesonepigreco/python-sscha
```

If you get an error of permissions, you need to add your user to the docker group.

This can be done with the commands:

```
sudo usermod -aG docker $USER
```

```
newgrp docker
```

Most clusters provide docker through a module load command, please, ask the cluster maintainer how to run a docker container on your favorite HPC system.

Once the container is installed, you can access it with

```
docker run -it mesonepigreco/python-sscha
```

the previous command opens a new shell with the python-sscha code installed. To share the content of the current directory with the container, execute the command with

```
docker run -it -v $PWD:/root mesonepigreco/python-sscha
```

This loads the content of the local directory inside the home (/root) of the container with python-sscha.

Chapter 2

Free energy calculation

2.1 Pre-processing

2.1.1 Create the object for the structure

```
structure = CC.Structure.Structure()
```

2.1.2 Read structure into structure object

```
structure.read_generic_file("Au.cif")
```

2.1.3 Prepare calculator

```
calculator = ase.calculators.emt() #as example
```

2.1.4 Set the Relax object with the structure and calculator to get the harmonic

```
relax = CC.calculators.Relax(structure, calculator)
```

2.1.5 Relax and get the dynamical matrix χ and save the dyn

```
relaxed = relax.static_relax() dyn = CC.Phonons.compute_phonons_finite_displacements(structure,  
calculator, supercell = (4,4,4))
```

Save data

```
dyn.save_qe(namefile)
```

2.2 Free energy minimization

2.2.1 Read the dynamical matrixes

```
dyn = CC.Phonons.Phonons(namefile, NQIRR)
```

2.2.2 Apply the sum rule and symmetrization

```
dyn.Symmetrize()
```

2.2.3 remove imaginary frequencies

```
dyn.ForcePositiveDefinite()
```

show frequencies after/before

```
W_harmonic, pols_harmonic = dyn.DiagonalizeSupercell()
```

2.2.4 Generate the ensemble

```
ensemble = sscha.Ensemble.Ensemble(dyn, Temperature, supercell = dyn.GetSupercell())  
ensemble.generate(N)
```

2.2.5 calculate forces/energies/(stress)

```
[...]
```

Calculators

```
[...]
```

2.2.6 Create the minimization object

```
minimizer = sscha.SchaMinimizer.SSCHA_Minimizer(ensemble)
```

2.2.7 set the minimization parameters

```
minimizer.min_step_dyn = 0.005 minimizer.min_step_struc = 0.05 minimizer.gradi_op  
= "all" minimizer.kong_liu_ratio = 0.5 minimizer.meaningfull_factor = 0.000001
```

2.2.8 Minimization cycle in two ways:

1. Step by step calculation `minimizer.init()` `minimizer.run()` `minimizer.finalize()`
2. Automatic relaxation `relax = sscha.Relax.SSCHA(minimizer, ase_calculator
= calculator, N_configs = N, max_pop = 20)` `relax.relax()`

2.3 Post-processing

Chapter 3

Structural relaxation

Chapter 4

Calculations of second-order phase transitions with the SSCHA

In the previous classes we had seen the theoretical

4.1 Structural instability

4.1.1 Free energy Hessian

For structural phase transitions, the order parameter is associated to phonon atomic displacements. So we just need to calculate the Free energy Hessian. the SSCHA provides an analytical equation for the free energy Hessian, derived by Raffaello Bianco in the work Bianco et. al. Phys. Rev. B 96, 014111.

4.2 Phase transition

4.2.1 title