DCS-Flow

Lucas Samir Ramalho Cavalcante

Aug 11, 2022

CONTENTS:

CHAPTER

ONE

INSTALL

1.1 DCS-Flow Requirements

- Atomic Simulation Environment
- DFTB+
- Phonopy
- Oclimax

1.2 Linux Installation

- Install ASE:

```
pip install --upgrade --user ase
```

- Install DFTB+:
  1. Download Slater-Koster files (parameters files for the DFTB method)
  2. Download DFTB+ binaries
  3. Extract the file and put it in your home folder
- Installing Phonopy

```
pip install --upgrade --user phonopy
```

- Installing OCLIMAX
  1. OCLIMAX uses the DOCKER platform to run the application. Please install it
  2. Now download OCLIMAX

```
curl -sL https://sites.google.com/site/ornliceman/getoclimax | bash
oclimax pull
```

- Installing DCS-Flow

```
git clone https://gitlab.com/lucassamir1/DCS-Flow.git
```

- Installing ChIMES

If you want to use the machine learning method called ChIMES, please contact Nir Goldman (goldman14@llnl.gov) in order to receive the following files:

1. DFTB+_ChIMES. This is a version of the DFTB+ package that has an interface to ChIMES. It should substitute the standart DFTB+ package.
2. ChIMES binaries (chimes_lsq, chimes_md, lsq2.py). Put these files inside the git directory DCS-Flow/dcs/.

1.3 Set environment variables

Add these lines to your configuration file (.bashrc). The following code uses example paths and must be edited according to your system.

```
export DFTB_PREFIX=/my_disk/my_name/slako/mio/mio-1-1/ # (path to Slako files)
export ASE_DFTB_COMMAND=/my_disk/my_name/dftbplus-20.1/bin/dftb+ > PREFIX.out # (path to␣
↪dftb+)
export PATH=/my_disk/my_name/dftbplus-20.1/bin:$PATH (path to dftb+ files)
export PYTHONPATH=/my_disk/my_name/DCS-Flow:$PYTHONPATH #(path to DCS file)
export PATH=/my_disk/my_name/DCS-Flow/dcs:$PATH #(path to file within DCS folder)
```

1.4 Usage

- Workflow

The workflow will relax the structure, create supercell displacements, calculate forces, run oclimax. The easiest way to start is through the command line interface

1. Get the parameters file

```
dcs workflow --get-params
```

2. Define parameters in the file
3. Run workflow

```
dcs workflow
```

1.5 MacOS Installation

- Install ASE:

```
pip3 install --upgrade --user ase
```

- Install DFTB+:
  1. Download Slater-Koster files (parameters files for the DFTB method)
  2. Download DFTB+

```
git clone https://github.com/dftbplus/dftbplus.git
cd dftbplus
git submodule update --init --recursive
```

_ Debugging Step: If getting an error related to WITH_OMP, open CMakeLists.txt file and add in the following line of code before if(WITH_OMP):

```
option(WITH_OMP FALSE)
```

1. Build DFTB+ (make sure to use your specific Fortran and C compilers)

```
mkdir build
cd build
FC=gfortran CC=gcc cmake ..
```

If configuration was successful

```
./dftb+
make -j
```

Test it

```
ctest
```

1. Install DFTB+

```
make install
```

- Install Phonopy

```
pip3 install --upgrade --user phonopy
```

- Install OCLIMAX

1. OCLIMAX uses the DOCKER platform to run the application. Please install it
2. Download OCLIMAX

```
curl -sL https://sites.google.com/site/ornliceman/getoclimax | bash
oclimax pull
```

- Install DCS-Flow

```
git clone https://gitlab.com/lucassamir1/DCS-Flow.git
```

- Installing ChIMES

If you want to use the machine learning method called ChIMES, please contact Nir Goldman (goldman14@llnl.gov) in order to receive the following files:

1. DFTB+_ChIMES. This is a version of the DFTB+ package that has an interface to ChIMES. It should substitute the standart DFTB+ package.
2. ChIMES binaries (chimes_lsq, chimes_md, lsq2.py). Put these files inside the git directory DCS-Flow/dcs/.

1.6 Set environment variables

Add these lines to your configuration file (~/.bash_profile). The following code uses example paths and must be edited according to your system.

```
export DFTB_PREFIX=/Users/my_name/slako/mio/mio-1-1/
↪#(Path to Slako files)
export ASE_DFTB_COMMAND=/Users/my_name/dftbplus/build/install/bin/dftb+ >PREFIX.out
↪#(Path to dftb+)
export PATH=/Users/my_name/dftbplus/build/install/bin:$PATH
↪#(Path to dftb+)
export PYTHONPATH=/Users/my_name/dftbplus/build/install/bin/dftb+:$PYTHONPATH
↪#(Python path to dftb+)
export PATH=/Users/my_name/DCS-Flow/dcs:$PATH                                    #(Path to␣
↪DCS-Flow file)
export PYTHONPATH=/Users/my_name/DCS-Flow:$PYTHONPATH                             #(Python␣
↪path to DCS-Flow file)
export PATH=/Users/my_name/.local/bin:$PATH
↪#(Path to ase file)
export PYTHONPATH=/Users/my_name/.local/bin/ase:$PYTHONPATH
↪#(Python path to ase file)
```

1.7 Usage

- Workflow

The workflow will relax the structure, create supercell displacements, calculate forces, run oclimax. The easiest way to start is through the command line interface

1. Get the parameters file

   ```
   dcs workflow --get-params
   ```

2. Define parameters in the file
3. Run workflow

   ```
   dcs workflow
   ```

CHAPTER

TWO

DOCUMENTATION

DCS-Flow is a collection of the following scripts:

- Relax: Optimizes structure.
- Phonons: Calculates phonons modes with the supercell method.
- Oclimax: Runs oclimax simulation creating a INS sprectrum.
- MD: Runs molecular dynamics simulations.
- Chimes: Creates the coefficients for the Chebyshev Interaction Model for Efficient Simulation.
- Train: Automated workflow that calls the functions necessary to create the ChIMES coefficients.
- Workflow: Automates the main workflow functions to relax structure, calculate phonons, and calculate the INS spectrum.

DCS-Flow has a command line interface implemented. Examples for how to use it are included under each main function.

2.1 Main Functions

2.1.1 Relax

- `relax(krelax=[6, 6, 6], fmax=0.05, geo=None, calc='dftbp', T=5):`
  - Finds the geometry file and optimizes the structure using the specified calculator (Populates 1-optimization folder with results).
  - The following input args are defined in the workflow parameters file:
    - krelax (list, optional): Number of k points for relaxation. Defaults to [6, 6, 6].
    - fmax (float, optional): Maximum allowed force for convergence between atoms. Defaults to 0.01.
    - **geo (str, optional): Geometry file or structure.**
      Allowed file types are .cif, .gen, .sdf, or .xyz. Defaults to None.
    - calc (str, optional): Calculator used. Options are 'dftbp', 'chimes', 'castep', or 'vasp'. Defaults to 'dftbp'.
    - T (int, optional): Simulation temperature. Defaults to 5K. Only used for DFTB+ and ChIMES.

```
dcs relax --krelax 6 6 6 --fmax 0.05 --geo TCNQ.cif --calc dftbp --temp 5
```

2.1.2 Phonons

- `phonons(dim=[4, 4, 4], kforce=[1, 1, 1], mesh=[8, 8, 8], calc='dftbp', T=5):`
  - Runs phonon supercell displacement calculations, populates 2-phonons folder with results.
  - The following input args are defined in the workflow parameters file:
    - dim (list, optional): Dimensions of the supercell. Defaults to [4, 4, 4].
    - kforce (list, optional): Number of k points for force calculations. Defaults to [1, 1, 1].
    - mesh (list, optional): Uniform meshes for each axis. Defaults to [8, 8, 8].
    - calc (str, optional): Calculator used for task. Options are 'dftbp', 'chimes', 'vasp', or 'castep'. Defaults to 'dftbp'.
    - T (int, optional): Simulation temperature. Defaults to 5K. Only used for DFTB+ and ChIMES.

```
dcs phonons --dim 4 4 4 --kforce 1 1 1 --mesh 8 8 8 --calc dftbp --temp 5
```

2.1.3 Oclimax

- `oclimax(params=None, task=0, e_unit=0):`
  - Creates folder 3-oclimax, writes oclimax parameters file in folder and runs oclimax simulation.
  - The following input args are defined in the workflow parameters file:
    - params (str, optional): Oclimax parameters file name if exists. Otherwise, it will be created in write_params function. Defaults to None.
    - **task (int, optional): Defines approximation method.**
      0:inc approx. 1:coh+inc. 2:single-xtal Q-E. 3:single-xtal Q-Q. Defaults to 0.
    - e_unit (int, optional): Defines energy unit. Defaults to 0.

```
dcs oclimax --task 0 --e_unit 0
```

2.1.4 MD

- `md(optgeo=None, calc='vasp', T=300, md_size=[1,1,1], steps=5000, time_step=1, dump_interval=100):`
  - Runs molecular dynamics simulation using vasp or castep (Creates 2-molecular_dynamics folder inside 0-train).
  - The following inputs are defined in the training parameters file (train_params.json):
    - optgeo (NoneType, optional): Optimized geometry file, only true if optgeo defined. Defaults to None.
    - calc (str, optional): Specifies calculator.Options are 'vasp' or 'castep'. Defaults to 'vasp'.
    - T (int, optional): Simulation temperature. Defaults to 300.
    - md_size (list, optional): Size of supercell. Defaults to [1,1,1].
    - steps (int, optional): Maximum number of ionic steps. Defaults to 5000.
    - time_step (int, optional): Md time step in fs. Defaults to 1.
    - dump_interval (int, optional): Step size of frames to be saved in the trajectory file. Defaults to 100.

```
dcs md --calc vasp --T 300 --md_size 1 1 1 --steps 5000 --time_step 1 --dump_interval 100
```

2.1.5 Chimes

- `chimes(trajfile=None, b2=12, b3=8, T=5):`
  - Calculates force difference between DFT and DFTB (training set), and fits the Chebyshev polynomials coefficients. Creates 3-chimes folder (inside 0-train directory) and writes params.txt file.
  - The following inputs are required:
    - trajfile (list, optional): Trajectory file output from md simulation. Defaults to None.
    - b2 (int, optional): Second body order of Chebyshev polynomial. Defaults to 12.
    - b3 (int, optional): Third body order of Chebyshev polynomial. Defaults to 8.
    - T (int, optional): Temperature for simulation in Kevin. Defaults to 5.

```
dcs chimes --b2 12 --b3 8 --T 5
```

2.1.6 Train

- `train(dct=None):`
  - Calls functions related to the training workflow (relax, md, chimes) with a timer using specified parameters in train_params.json, else with default parameters. Creates 0-train directory.
  - The following input is required:
    - dct (dict, optional): JSON file with specified parameters for relax, md, and chimes functions. Defaults to 'train_params.json'.

```
dcs train
```

2.1.7 Workflow

- `workflow(dct=None):`
  - Calls all workflow functions (relax, phonons, oclimax) with a timer using specified parameters in workflow_params.json, else with default parameters.
  - The following input dictionary is the workflow parameters files:
    - dct (dict, optional): JSON file with specified parameters for relax, phonons, and oclimax functions. Defaults to 'workflow_params.json'.

```
dcs workflow
```

CHAPTER

THREE

EXAMPLES

3.1 Example 1: Main Workflow using DFTB+ for TCNQ on PC

The following example shows the primary workflow using dftb+ as the calculator run on a personal terminal (as opposed to a super computer).

First, create a folder containing the geometry file (.cif, .gen, .sdf, or .xyz). The folder used in this example, named TCNQ, can be downloaded from the Uploads Folder.

In the TCNQ folder, create the workflow parameters file, `workflow_params.json`, using the following command.

```
dcs workflow --get-params
```

Edit the workflow parameters file to match the following values.

```
{
    "krelax": [
        4,
        4,
        2
    ],
    "fmax": 0.05,
    "geo": null,
    "calc": "dftbp",
    "T":5,
    "dim": [
        2,
        2,
        1
    ],
    "kforce": [
        1,
        1,
        1
    ],
    "mesh": [
        8,
        8,
        8
    ],
    "params": null,
    "task": 0,
```

(continues on next page)

(continued from previous page)

```
    "e_unit": 0
}
```

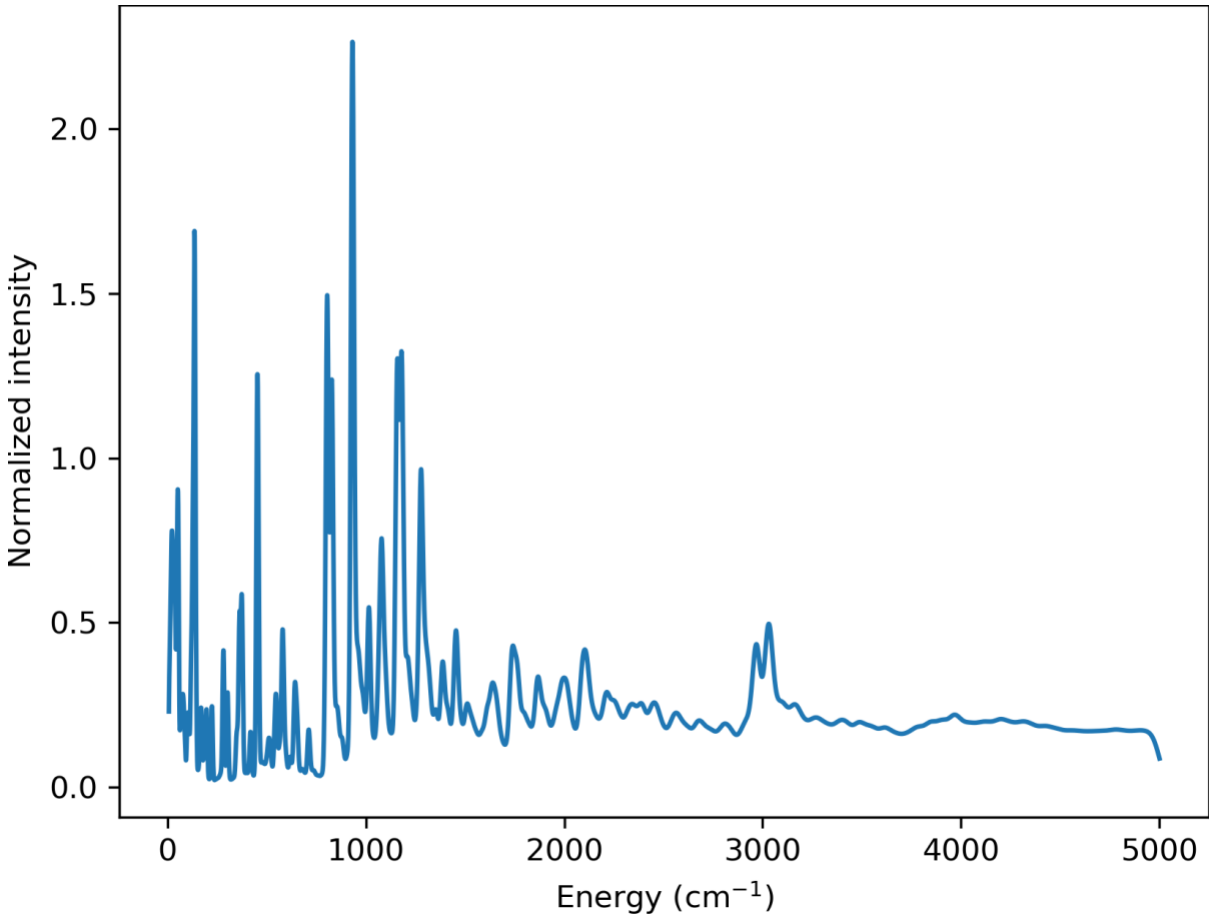
The TCNQ folder, or current directory, now has the structure file (tcnq.cif) and the edited parameters file (workflow_params.json). Use the following command to begin the calculation:

```
dcs workflow
```

Once the job has completed, the following files can be found in the TCNQ folder.

```
1-optimization      3-oclimax       err.out         run_tcnq.py
2-phonons       out.out         workflow_params.json
```

Open the 3-oclimax folder and click on the png file to view to INS specta.



3.2 Example 2: Main Workflow using DFTB+ for TCNQ on NERSC

The following example shows the primary workflow using dftb+ as the calculator using the NERSC super computer.

First, there is no need to install the DCS-Flow package and its dependencies. Just access the super computer via the terminal, and load the DCS module using the following commands:

```
module use /global/common/software/m2734/DCS/modulefiles
module load dcs
```

You can add these commands to your bashrc file in your NERSC home folder to load the DCS module every time you access NERSC.

Create a folder containing the geometry file (.cif, .gen, .sdf, or .xyz) and a run_tcnq.py bash script (for NERSC). This folder, named TCNQ, can be downloaded from the Uploads Folder.

Upload the TCNQ folder to NERSC using a file transfer software like Globus.

Inside the TCNQ directory, create the workflow parameters file, `workflow_params.json`, using the following commands.

```
cd TCNQ
dcs workflow --get-params
```

Edit the workflow parameters file to match the following values.

```
{
    "krelax": [
        4,
        4,
        2
    ],
    "fmax": 0.05,
    "geo": null,
    "calc": "dftbp",
    "T":5,
    "dim": [
        2,
        2,
        1
    ],
    "kforce": [
        1,
        1,
        1
    ],
    "mesh": [
        8,
        8,
        8
    ],
    "params": null,
    "task": 0,
    "e_unit": 0
}
```

The TCNQ folder, or current directory, now has the structure file (tcnq.cif), the edited parameters file (workflow_params.json), and the run script (run_tcnq.py.). The run_tcnq.py bash script contains information for the NERSC super computer such as number of allocated nodes, processors and run hours. The final lines contain the commands to be evaluated, in this case `eval $'dcs workflow'`.

Submit the job and check it's progress using the following commands:

```
sbatch run_tcnq.py
sqs
```

Once the job has completed, the following files can be found in the TCNQ folder.

```
1-optimization      3-oclimax       err.out         run_tcnq.py
2-phonons       out.out         workflow_params.json
```

Use a file transfer software like Globus to transfer the wanted files to your personal computer. Open the 3-oclimax folder and click on the png file to view to INS specta. The resulted INS spectrum will be the same as simulated in the example before.

3.3 Example 3: Training and Main Workflow for TTF-TCNQ

The following example walks through the training workflow, with ChIMES, using the NERSC super computer.

First access the super computer via the terminal, and load the DCS module using the following commands:

```
module use /global/common/software/m2734/DCS/modulefiles
module load dcs
```

Create a folder containing the geometry file (.cif, .gen, .sdf, or .xyz) and a run_tcnq.py bash script (for NERSC). This folder, named TTF-TCNQ, can be downloaded from the Uploads Folder.

Upload the TTF-TCNQ folder to NERSC using a file transfer software like Globus.

Inside the TTF-TCNQ directory, generate the training parameters file, `train_params.json`, using the following commands:

```
cd TTF-TCNQ
dcs train --get-params
```

Edit the training parameters file to match the following values.

```
{
    "krelax": [
        6,
        6,
        6
    ],
    "fmax": 0.05,
    "geo": null,
    "calc": "castep",
    "optgeo": null,
    "T": 5,
    "md_size": [
```

(continues on next page)

(continued from previous page)

```
        1,
        1,
        1
    ],
    "steps": 5000,
    "time_step": 1,
    "dump_interval": 100,
    "trajfile": null,
    "b2": 12,
    "b3": 8
}
```

The TCNQ folder, or current directory, now has the structure file (TTF-TCNQ.cif), the edited parameters file (train_params.json), and the run script (run_ttf-tcnq.py.). The run_ttf-tcnq.py bash script contains information for the NERSC super computer such as number of allocated nodes, processors and run hours. The final lines contain the commands to be evaluated, in this case `eval $'dcs train`.

Submit the job and check it's progress using the following commands:

```
sbatch run_ttf-tcnq.py
sqs
```

Once the job has completed, the following files can be found in the TTF-TCNQ folder.

```
0-train         err.out         run_ttf-tcnq.py         params.txt
TTF-TCNQ.cif        out.out         train_params.json
```

Once the training has successfuly run, create a workflow parameters file in the TTF-TCNQ folder using the following commands.

```
dcs workflow --get-params
```

In the workflow parameters, edit the calculator to chimes and change the default parameters as follows:

```
{
    "krelax": [
        4,
        4,
        2
    ],
    "fmax": 0.05,
    "geo": null,
    "calc": "chimes",
    "T":5,
    "dim": [
        2,
        2,
        1
    ],
    "kforce": [
        1,
        1,
        1
```

(continues on next page)

(continued from previous page)

```
    ],
    "mesh": [
        8,
        8,
        8
    ],
    "params": null,
    "task": 0,
    "e_unit": 0
}
```

The TTF-TCNQ folder, or current directory, now has the structure file (TTF-TCNQ.cif), the chimes output (params.txt), the edited parameters file (workflow_params.json) and the run script (run_ttf-tcnq.py.). The final lines of the run script should contain the commands to be evaluated, in this case `eval $'dcs workflow'`.

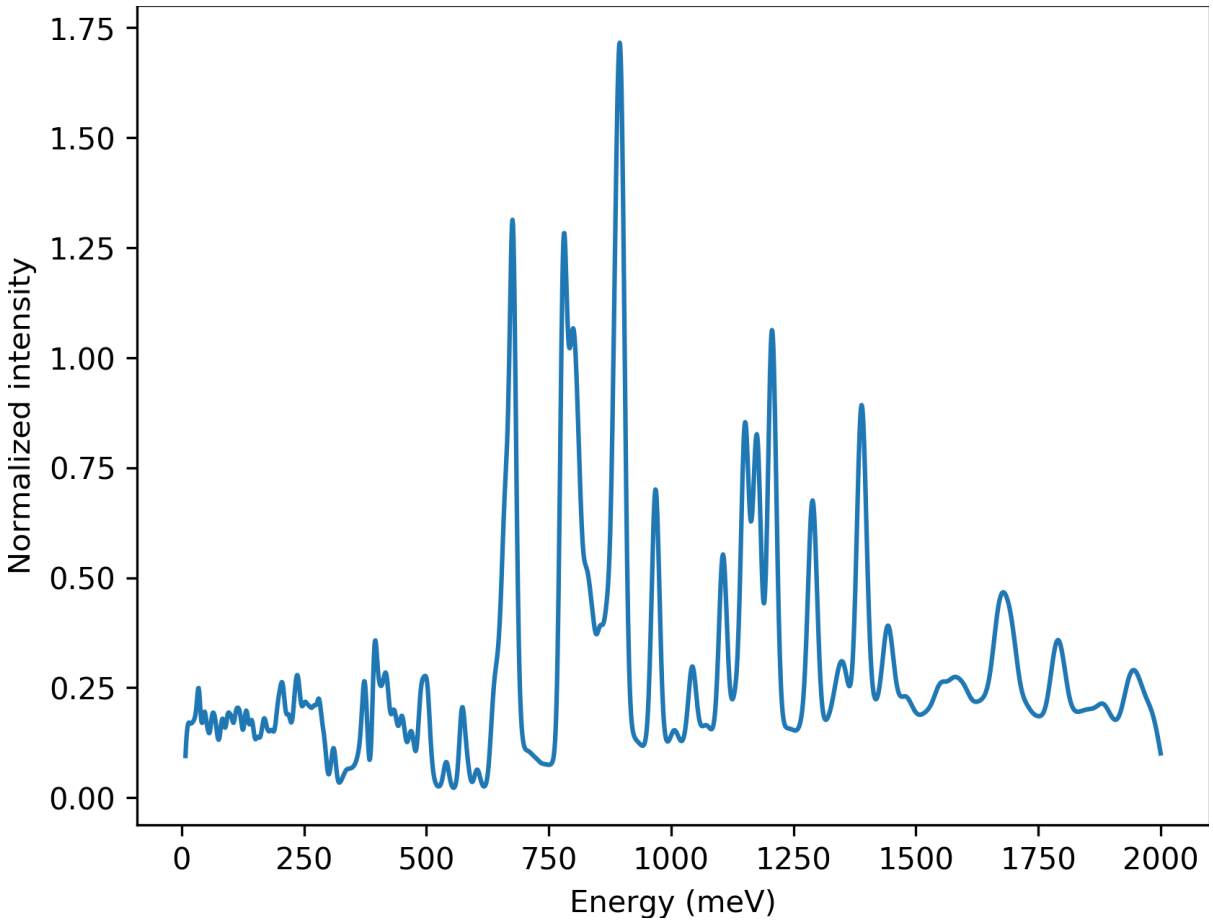
Submit the job and using the following commands:

```
sbatch run_ttf-tcnq.py
```

Once the job has completed, the following files will be added to the TTF-TCNQ folder.

```
1-optimization      3-oclimax
2-phonons
```

Use a file transfer software like Globus to transfer the wanted files to your personal computer. Open the 3-oclimax folder and click on the png file to view to INS specta.


1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Normalized intensity
0
250
500
750
1000
1250
1500
1750
2000
Energy (meV)

CHAPTER

FOUR

DCS-FLOW

4.1 dcs package

4.1.1 Submodules

4.1.2 dcs.chimes module

class dcs.chimes.**CLICommand**

Bases: object

Chebyshev Interaction Model for Efficient Simulation (ChIMES)

static add_arguments(*parser*)

Sets up command line to run Chimes i.e. recognize arguments and commands.

Args:
parser (argparse): Arguments to be added.

static run(*args*)

Runs chimes function using command line arguments.

Args:
args (argparse): Command line arguments added to parser using the function add_arguments.

dcs.chimes.**chimes**(*trajfile=None*, *b2=12*, *b3=8*, *T=5*)

Calculates force difference between DFT and DFTB (training set),
and fits the Chebyshev polynomials coefficients. Creates 3-chimes folder (inside 0-train directory) and writes params.txt file.

Args:
trajfile (list, optional): Trajectory file (ist of atoms objects) output from md simulation. Defaults to None. b2 (int, optional): Second body order of Chebyshev polynomial. Defaults to 12. b3 (int, optional): Third body order of Chebyshev polynomial. Defaults to 8. T (int, optional): Temperature for simulation in Kevin. Defaults to 5.

dcs.chimes.**dftb_fmatch_input**(*T*, *frame*)

Runs DFTB force calculations using frame from DFT;
calculates the force difference between DFT and DFTB. Creates a list with the chemical symbols, atomic positions, and force differences.

Args:
T (int): Temperature for DFTB simulation. frame (list): ASE atoms object with atomic positions, forces, energies, etc from DFT calculation.

Returns:
list: Contains the chemical symbols, atom positions, and difference in DFTB and DFT forces.

dcs.chimes.**fm_setup_input**(*nframes*, *b2*, *b3*, *setsymbols*, *smax*)

Writes force match setup input file (fm_setup.in) with given arguments.

Args:
nframes (int): Number of trajectories. b2 (int): Second body order of Chebyshev polynomial. b3 (int): Third body order of Chebyshev polynomial. setsymbols (set): Chemical symbols. smax (float): Half of the minimum cell length.

dcs.chimes.**lsq**()

Runs chimes_lsq binary and creates A and B matrices (A.txt, B.txt). Runs lsq fitting process and writes ChIMES coefficients to params.txt.

dcs.chimes.**multi_fmatch**(*T*)

Runs dftb_fmatch_input command for each trajectory, n processes at a time.
Returns tuple with trajectory information.

Args:
T (int): Temperature for DFTB simulation.

Returns:

tuple: (nframes, setsymbols, smax)
nframes (int) - Number of ASE atoms object. setsymbols (set) - Chemical symbols. smax (float) - Half of the minimum cell length.

dcs.chimes.**rdf**(*smax*, *pair*)

Finds the radial distribution function for elements defined in pair.
Analyses the RDF, and determines the min and max distances in a atomic pair interaction. Returns a tuple with mlambda, rmin and rmax.

Args:
smax (float): Half of the minimum cell length. pair (list): List with two chemical symbols of a pair interaction, e.g., [‘C’, ‘H’]

Returns:

tuple: (mlambda, rmin, rmax)
mlambda (float) - Morse lambda factor rmin (float) - Minimum distance considered in a atomic pair interaction. rmax (float) - Maximum distance considered in a atomic pair interaction.

dcs.chimes.**run_md_input**(*folder*)

Writes MD input file (run_md.in) if it does not exist in folder.

Args:
folder (str): Directory that contains params.txt file.

Raises:
Exception: Raises exception if there is no params.txt file in folder.

4.1.3 dcs.main module

dcs.main.**main**()

Looks up ase_main function, sets up command line.

4.1.4 dcs.md module

class dcs.md.**CLICommand**

Bases: object

Molecular dynamics with constant temperature

static add_arguments(*parser*)

Sets up command line to run molecular dynamics with constant temperature i.e. recognize arguments and commands.

Args:
parser (argparse): Arguments to be added.

static run(*args*)

Runs md function using command line arguments.

Args:
args (argparse): Command line arguments added to parser using the function add_arguments.

dcs.md.**md**(*optgeo=None*, *calc='vasp'*, *T=300*, *md_size=[1, 1, 1]*, *steps=5000*, *time_step=1*, *dump_interval=100*)

Runs molecular dynamics simulation using vasp or castep (Creates 2-molecular_dynamics folder inside 0-train)

Args: optgeo (NoneType, optional): Optimized geometry file, only true if optgeo defined. Defaults to None. calc (str, optional): Specifies calculator. Options are 'vasp' or 'castep'. Defaults to 'vasp'. T (int, optional): Simulation temperature. Defaults to 300. md_size (list, optional): Size of supercell. Defaults to [1,1,1]. steps (int, optional): Maximum number of ionic steps. Defaults to 5000. time_step (int, optional): Md time step in fs. Defaults to 1. dump_interval (int, optional): Step size of frames to be saved in the trajectory file. Defaults to 100.

Raises: NotImplementedError: If calculator other than vasp or castep specified.

dcs.md.**run_castep_md**(*atoms*, *T*, *steps*, *time_step*, *dump_interval*)

Runs castep md calculation on atoms for specified intervals.

Args:
atoms (list): Atoms object involved in calc from ASE. T (int): Initial and final simulation temperature. steps (int): Maximum number of ionic steps, defines the total simulation time. time_step (int): md time step in fs. dump_interval (int): Step size of frames to be saved in the trajectory file.

dcs.md.**run_vasp_md**(*atoms*, *T*, *steps*, *time_step*, *dump_interval*)

Runs vasp md calculation on atoms for specified intervals.

Args:
atoms (list): Atoms object involved in calc from ASE. T (int): Initial and final simulation temperature. steps (int): Maximum number of ionic steps, defines the total simulation time. time_step (int): md time step in fs. dump_interval (int): Step size of frames to be saved in the trajectory file.

dcs.md.**set_castep_pbc**()

4.1.5 dcs.oclimax module

class dcs.oclimax.**CLICommand**

Bases: object

Run OCLIMAX simulations

static add_arguments(*parser*)

Sets up command line to run OCLIMAX i.e. recognize arguments and commands.

Args:
parser (argparse): Arguments to be added.

static run(*args*)

Runs oclimax function using command line arguments.

Args:
args (argparse): Command line arguments added to parser using the function add_arguments.

dcs.oclimax.**oclimax**(*params=None*, *task=0*, *e_unit=0*)

Creates folder 3-oclimax, writes oclimax parameters file in folder and runs oclimax simulation.

Args:
params (str, optional): Oclimax parameters file defined in write_params function. Defaults to None. task (int, optional): Defines approximation method.

0:inc approx. 1:coh+inc. 2:single-xtal Q-E. 3:single-xtal Q-Q. Defaults to 0.

e_unit (int, optional): Defines energy unit. Defaults to 0.

dcs.oclimax.**plot**()

Creates plot using csv file (oclimax output with INS data) and saves as a png. Plots energy (meV) versus Normalized intensity.

dcs.oclimax.**run_oclimax**(*params*)

If convert.done file doesn't exist, runs OCLIMAX convert. Converts input mesh to out.oclimax;
out.oclimax is the OCLIMAX format for mesh type files.

Args:
params (str): Oclimax parameters file name defined in write_params function.

dcs.oclimax.**write_params**(*task*, *e_unit*)

Creates parameters file for Oclimax.

Args:

task (int): Defines approximation method where
0:inc approx. 1:coh+inc. 2:single-xtal Q-E. 3:single-xtal Q-Q.

e_unit (int): Defines energy units such that 0:cm-1 1:meV 2:THz.

4.1.6 dcs.phonons module

class dcs.phonons.**CLICommand**

Bases: object

Calculate phonons

static add_arguments(*parser*)

Sets up command line to run phonopy i.e. recognize arguments and commands.

Args:
parser (argparse): Arguments to be added.

static run(*args*)

Runs phonons function using command line arguments.

Args:
args (argparse): Command line arguments added to parser using the function add_arguments.

dcs.phonons.**calculate_forces**(*mode*, *kwargs*, *dir*)

Runs single point energy calculation.

Args:
kforce (list): Number of k points for force calculations. mode (str): Calculator used for task. Options are 'dftbp', 'chimes', 'vasp', or 'castep'. T (int, optional): Simulation temperature. Only used for DFTB+ and ChIMES. dir (str): Directory to change to and run calculator in.

dcs.phonons.**calculate_mesh**(*mesh*, *mode*)

Creates Phonopy force sets for specified calculator and runs mesh sampling phonon calculation.

Args:
mesh (list): Uniform meshes for each axis. mode (str): Calculator used for task. Options are 'dftbp', 'chimes', 'vasp', or 'castep'.

dcs.phonons.**calculator_kwargs**(*kforce*, *mode*, *T*, *folder='.'*)

dcs.phonons.**generate_supercell**(*dim*, *mode*)

Reads in the molecule structure file and creates displacements.

Args:
dim (list): Dimensions of the supercell. mode (str): Calculator used for the task. Options are 'dftbp' or 'chimes'.

dcs.phonons.**multi_forces**(*kforce*, *mode*, *T*, *folder*, *mpi=False*)

Calls calculate_forces function using parallel processing,
calculates forces for specified mode.

Args:
kforce (list): Number of k points for force calculations. mode (str): Calculator used for task. Options are 'dftbp', 'chimes', 'vasp', or 'castep'. mpi (bool, optional): Not currently implemented. Defaults to False.

dcs.phonons.**organize_folders**(*mode*)

Finds supercell displacement files and formats the name.
Creates directory with supercell displacement number,

and moves supercell displacement file into created directory.

Args:
mode (str): Calculator used for task. Options are 'dftbp', 'chimes', 'vasp', or 'castep'.

dcs.phonons.**phonons**(*dim=[4, 4, 4]*, *kforce=[1, 1, 1]*, *mesh=[8, 8, 8]*, *calc='dftbp'*, *T=5*)

Runs phonon supercell displacement calculations, populates 2-phonons folder with results.

Args:

dim (list, optional): Dimensions of the supercell. Defaults to [4, 4, 4]. kforce (list, optional): Number of k points for force calculations. Defaults to [1, 1, 1]. mesh (list, optional): Uniform meshes for each axis. Defaults to [8, 8, 8]. calc (str, optional): Calculator used for task. Options are 'dftbp', 'chimes', 'vasp', or 'castep'. Defaults to 'dftbp'. T (int, optional): Simulation temperature. Defaults to 5K. Only used for DFTB+ and ChIMES.

Raises:

NotImplementedError: Raised if 'calc' specified is not available.

dcs.phonons.**write_params**(*kforce*, *mode*, *T*)

Writes json file with default arguments for the relaxation calculator.

4.1.7 dcs.relax module

class dcs.relax.**CLICommand**

Bases: object

Optimize structure

static add_arguments(*parser*)

Sets up command line to run relax script i.e. recognize arguments and commands.

Args:

parser (argparse): Arguments to be added.

static run(*args*)

Runs relax.py functions using command line arguments.

Args:

args (argparse): Command line arguments added to parser using the function add_arguments.

dcs.relax.**calculator_kwargs**(*krelax*, *fmax*, *geo*, *mode*, *T*, *folder='.'*)

dcs.relax.**find_geo**(*folder*)

Searches the current working directory for the molecular structure file.

Allowed file types are .cif, .gen, .sdf, or .xyz.

Args:

folder (str): The current working directory.

Returns:

str: Molecular structure file.

dcs.relax.**relax**(*krelax=[6, 6, 6]*, *fmax=0.05*, *geo=None*, *calc='dftbp'*, *T=5*)

Finds the geometry file and optimizes the structure using the specified calculator (Populates 1-optimization folder with results).

Args:

krelax (list, optional): Number of k points for relaxation. Defaults to [6, 6, 6]. fmax (float, optional): Maximum allowed force for convergence between atoms. Defaults to 0.01. geo (str, optional): Geometry file or structure.

Allowed file types are .cif, .gen, .sdf, or .xyz. Defaults to None.

calc (str, optional): Calculator used. Options are 'dftbp', 'chimes', or 'vasp'. Defaults to 'dftbp'. T (int, optional): Simulation temperature. Defaults to 5K. Only used for DFTB+ and ChIMES.

dcs.relax.**relax_structure**(*krelax*, *fmax*, *geo*, *mode*, *T*, *folder*)

Defines arguments for specified calculator and optimizes the structure.

Args:

krelax (list): Number of k points for relaxation. fmax (float): Maximum allowed force for convergence between atoms. geo (str): Geometry file or structure. Allowed file types are .cif, .gen, .sdf, or .xyz. mode (str): Calculator used. Options are 'dftbp', 'chimes', 'vasp', or 'castep'. T (int, optional): Simulation temperature. Only used for DFTB+ and ChIMES.

Raises:

NotImplementedError: If specified calculator is not an option for mode, error raised.

dcs.relax.**write_params**(*krelax*, *fmax*, *geo*, *mode*, *T*)

Writes json file with default arguments for the relaxation calculator.

4.1.8 dcs.train module

class dcs.train.**CLICommand**

Bases: object

Workflow to run DFT-MD and train ChIMES model

static add_arguments(*parser*)

Sets up command line to run train script i.e. recognize arguments and commands.

Args:

parser (argparse): Arguments to be added.

static run(*args*)

Runs train function using command line arguments.

Args:

args (argparse): Command line arguments added to parser using the function add_arguments.

dcs.train.**train**(*dct=None*)

Calls functions related to the training workflow (relax, md, chimes) with a timer using specified parameters in train_params.json, else with default parameters. Creates 0-train directory.

Args:

dct (dict, optional): JSON file with specified parameters for relax, md, and chimes functions. Defaults to 'train_params.json'.

dcs.train.**write_params**()

Writes a json file with the arguments and default values for realx, md, and chimes functions.

4.1.9 dcs.workflow module

class dcs.workflow.**CLICommand**

Bases: object

Workflow to relax structure, calculate phonons and calculate INS spectrum

static add_arguments(*parser*)

Sets up command line to run workflow script i.e. recognize arguments and commands.

Args:
parser (argparse): Arguments to be added.

static run(*args*)

Runs workflow function using command line arguments.

Args:
args (argparse): Command line arguments added to parser using the function add_arguments.

dcs.workflow.**workflow**(*dct=None*)

Calls all workflow functions (relax, phonon, oclimax) with a timer using specified parameters in workflow_params.json, else with default parameters.

Args:
dct (dict, optional): Specified parameters for relax, phonons, and oclimax functions. Defaults to None.

dcs.workflow.**write_params**()

Writes json file with default arguments from relax, phonons, and oclimax scripts.

4.1.10 Module contents

dcs.**chdir**(*folder*)

Changes the working directory to folder if not already current wd.

Args:
folder (str): Name of folder to make wd.

dcs.**done**(*mode*)

Create .done file for task as f and closes.

Args:
mode (str): Name of task finished.

dcs.**get_default_parameters**(*func*)

Inspects function func for argument names and default arguments.

Args:
func (func): Input function to inspect.

Returns:
dict: Keys are argumnent names, values are the default arguments.

dcs.**isdone**(*mode*)

Checks for .done file.

Args:
mode (str): Calculator used for task/specified.

Returns:
bool: True if mode.done files exists, false if mode.done doesn't exist.

dcs.**mkdir**(*folder*)

Creates folder in current wd unless OSError occurs.

Args:
folder (str): Name of folder to create.

dcs.**mktempdir**()

Creates a temporary directory.

Yields:
str: Temporary directory name.

dcs.**out**(*task*)

Creates .out and .err files for specified task.

Args:
task (str): Name of task.

dcs.**read_json**(*filename*)

Reads specified json file.

Args:
filename (str): Full name of json file.

Returns:
dict: Dictionary with data specified by json file.

dcs.**write_json**(*filename*, *data*)

Creates json file with specified name and data.

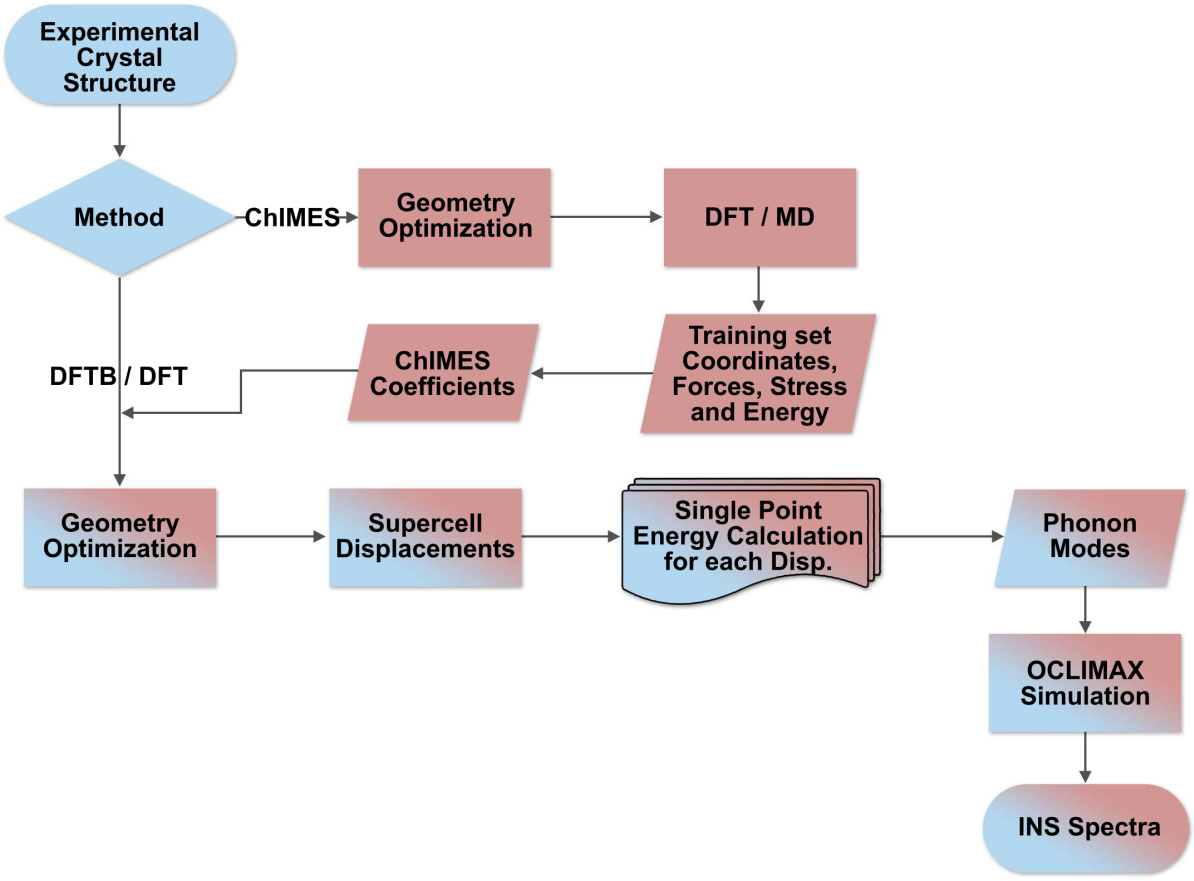
Args:
filename (string): Name of json file. Must include .json. data (dict): Dictionary with data.

CHAPTER

FIVE

WORKFLOW DESCRIPTION

The Davis Computational Spectroscopy workflow (DCS-Flow) was designed to connect and automate different material sciences tools which facilitate the use and comparison of electronic methods such as DFT, DFTB and machine learning to simulate the structure and dynamics of materials properties for improved structure property prediction. DCS-Flow provides an efficient workflow to create databases of Inelastic Neutron Scattering simulations (DCS Discover Database).



In the figure above, we present an overview of the complete workflow of the DCS-Flow method. The main part of it is composed by steps in blue or that contain a blue gradient. With the experimental crystal structure file and a set of workflow parameters as inputs, we optimize the structure, simulate the lattice dynamics with the supercell method and calculate the INS spectrum. This is called the main workflow.

In the other hand, the red boxes represent the training workflow, in which the ChIMES model is employed. It starts with the optimization of the structure, followed by a DFT-MD simulation from which a training set of forces, stress

tensors and energies is extracted. Finally, the ChIMES model is trained to this set creating coefficients that correct the DFTB calculations. The possibility of correcting the main workflow is represented by a red gradient.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX

O

P

R

S

T

W