OpenAtom for GW calculations

by OpenAtom developers

1 Introduction

The GW method is one of the most accurate *ab initio* methods for the prediction of electronic band structures. Despite its power, the GW method is not routinely applied to large scale materials physics or chemistry problems due to its unfavorable computational scaling: standard implementations scale as $O(N^4)$ where N is the number of electrons in the system. To develop GW software that can tackle heavy computational load and large memory requirement, we have developed and implemented algorithms that work in real space for the canonical plane-wave pseudopotential approach to electronic structure calculations.

This documentation will explain how to run the OpenAtom GW software with step-by-step instructions and pointers to examples. If you have further questions, comments, issues, or bugs to report, please contact the OpenAtom developer team.

2 Download & Compilation

OpenAtom is hosted using git and can be downloaded using the following command:[1]

```
git clone https://charm.cs.illinois.edu/gerrit/openatom.git --depth=1
```

You will also need to download `charm++` using the following command:

```
git clone https://charm.cs.illinois.edu/gerrit/charm.git
```

At this moment, compiling the GW part is independent from the main ground-state OpenAtom compilation (this may change in the near future). To create the GW executable, go to the `src_gw_bse` subdirectory and follow the instruction in the `README` file. Once you compile successfully, two executables (`charmrun` and `gw_bse`) will appear in `build` subdirectory.

[1] If you want the entire history, do not use depth option

3 Examples

This document will use a 2-atom unit cell bulk Si system to explain how to run GW calculations. We strongly recommend users to download examples before proceeding to next sections. Example files with full data sets for small 2-atom unit cells of Si and GaAs are available via the OpenAtom git repository:

```
git clone https://charm.cs.illinois.edu/gerrit/datasets/gwbse/Si2
git clone https://charm.cs.illinois.edu/gerrit/datasets/gwbse/GaAs2
```

The Si2 and GaAs2 examples include all input files for DFT calculations (Quantum Espresso files) and OpenAtom GW. The OpenAtom GW output files resulting from running the software can also be found in the `output` subdirectory to permit for checking.

4 DFT Calculations

4.1 Generating ψ_{nk} and E_{nk}

For the current release, we rely on the Quantum Espresso (QE) software[2] to generate ground state wavefunctions and energies. In the example directory (`Si2/QuantumEspresso`), input files for `scf` and `bands` calculations with QE are included. Once `bands` calculation is done, use our converter (`pw2openatom.x`) to generate states that OpenAtom can read. The converter is located in `external_conversion` subdirectory. Follow the instruction in the `README` there to set up the converter.

To prevent the Coulomb operator from diverging at zero wave vector when computing the dielectric response, you must also separately generate wavefunctions and energies for occupied states with shifted k grids. A small shift is recommended (e.g., $(0, 0, 0.001)$ in our example in lattice units).

OpenAtom GW is designed for large systems which do not require dense k grid. Hence, no symmetries are considered in OpenAtom GW. If you wish to run a small system with many k points, we recommend to using another other GW software which has symmetry reduction built-in such as BerkeleyGW[3].

For educational purposes, we describe in more detail an example of 2-atom Si with 8 k points

[2]P. Giannozzi et al., QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials, Journal of Physics: Condensed Matter, vol. 21, no. 39, p. 395502, 2009. http://www.quantum-espresso.org/

[3]Deslippe, xSamsonidze, Strubbe, Jain, Cohen and Louie, *Computer Physics Communications* **183**, 1269-1289, (2012). DOI 10.1016/j.cpc.2011.12.006 . http://berkeleygw.org/

here and 52 bands. The key steps for preparing the GW calculation are:

1. Run the QE `scf` calculation (e.g., `pw.x < in.scf`)
2. Run the QE `bands` calculations with the desired number of empty states (e.g., `pw.x < in.bands`) for the original (unshifted) k grid.
3. Create the `STATES` directory. Use `makedir.sh`. For this example, we run the command `./makedir.sh 8 1` on the command line
4. Run the converter (`pw2openatom.x < in.pw2openatom`) with `shift_flag=.false.`
5. Run `bands` calculations with shifted k grid (e.g. `pw.x < in.bandsq`). Only occupied states are needed.
6. Run converter again with `shift_flag=.true.`

A description of the input options for `pw2openatom.x` is available at the end of the `README` file.

4.2 Vxc

In this first release, we have not implemented a converter that calculates V_{xc} matrix elements. For V_{xc} matrix elements, we encourage users to run the Quantum Espresso `pw2bgw.x` program with the `vxc_flag` set to be on. In our next release, we will provide our own converter which calculates V_{xc} matrix elements.

Here are the steps to follow in order to compute the V_{xc} elements using Quantum Espresso:

1. Go to the `PP/src` directory in the Quantum Espresso package.
2. Run `make pw2bgw.x`
3. Run `pw2bgw.x` in your working directory with these input options in its input file:
   - `vxc_flag = .true.`
   - `vxc_diag_nmin = n1`
   - `vxc_diag_nmax = n2`

where `n1` and `n2` are the smallest and largest band indices for which you want the Vxc matrix elements.

If off diagonal elements are desired, use the `vxc_offdiag_nmin` and `vxc_offdiag_nmax` keywords instead.

5 OpenAtom GW Input Files

User must create 4 input files to run OpenAtom GW.

1. `klist.dat` : specifying k lists and their weight as well as shifting vector
2. `lattice.dat` : specifying lattice vectors
3. `band_list.dat` : specifying which Σ elements the user wants to calculate
4. simulation keywords file (`config` file in Si2 or GaAs examples) : the simulation keywords file controls all the options used in GW calculations. This file can have any name.

5.1 klist.dat

This file tells the code how many k points are used and what they actually are. The format of this file follows below:

nk
$k_x^1\ k_y^1\ k_z^1\ w_{k^1}$
...
$k_x^{nk}\ k_y^{nk}\ k_z^{nk}\ w_{k^{nk}}$
$s_x\ s_y\ s_z$
$nk_1\ nk_2\ nk_3$

nk - number of k points
$k_x\ k_y\ k_z$ - k point coordinates (crystal, i.e. lattice, coordinates)
w_k - weight
$s_x\ s_y\ s_z$ - shift vector (crysta/lattice coordinates)
$nk_1\ nk_2\ nk_3$ - positive integers specifying density of k grid along each reciprocal lattice vector

Example of `klist.dat` with 8 k points:

```
8
0.000000000  0.000000000  0.000000000     0.125
0.000000000  0.000000000  0.500000000     0.125
0.000000000  0.500000000  0.000000000     0.125
0.000000000  0.500000000  0.500000000     0.125
0.500000000  0.000000000  0.000000000     0.125
0.500000000  0.000000000  0.500000000     0.125
```

```
0.500000000  0.500000000  0.000000000      0.125
0.500000000  0.500000000  0.500000000      0.125
0.0          0.0          0.001
2            2            2
```

5.2 lattice.dat

This file tells the shape of the simulation cell and reciprocal lattice vector. The format of this file follows below:

l_a
$a1_x$ $a1_y$ $a1_z$
$a2_x$ $a2_y$ $a2_z$
$a3_x$ $a3_y$ $a3_z$
$b1_x$ $b1_y$ $b1_z$
$b2_x$ $b2_y$ $b2_z$
$b3_x$ $b3_y$ $b3_z$

l_a - lattice parameter (Bohr radius [atomic] units)
$a1$, $a2$, $a3$: lattice vector (units of l_a)
$b1$, $b2$, $b3$: reciprocal lattice vector (Cartesian coordinates in units $2\pi/l_a$)

Example of `lattice.dat`:

```
10.2612
0.000000000   0.500000000   0.500000000
0.500000000   0.000000000   0.500000000
0.500000000   0.500000000   0.000000000
 -1.000000    1.000000    1.000000
  1.000000   -1.000000    1.000000
  1.000000    1.000000   -1.000000
```

5.3 band_list.dat

This file sets which matrix element to calculate for Σ, $\langle\psi_{n_1}|\,\Sigma\,|\psi_{n_2}\rangle$. The format of this file is:

n_{lm}

l_1 m_1
l_2 m_2
...
$l_{n_{lm}}$ $m_{n_{lm}}$

n_{lm} : How many lm pair that a user wants to calculate
l : band index of ψ_l
m : band index of ψ_m

Example of `band_list.dat` file:

```
2
4 4
5 5
```

5.4 simulation keywords file (configuration file)

Simulation keywords file sets relevant variables used for GW runs. The example of simulation keywords file is below. Full list of keywords and arguments are explained in the next section.

```
~gen_GW[
\num_tot_state{52}       Total number of the states
\num_occ_state{4}       Number of the occupied states
\num_unocc_state{48}    Number of the unoccupied states
\num_kpoint{8}    Number of k-point - should be consistent with klist.dat
\num_spin{1}    Number of spin
\statefile_binary_opt{off}
]


~GW_epsilon[
\EcutFFT{12} FFT size
\Ecuteps{10} Epsilon cutoff
]

~GW_sigma[
]

~GW_parallel[
\rows_per_chare{10}
```

```
\cols_per_chare{10}
\pipeline_stages{1}
]
```

6 Running OpenAtomGW

To run OpenAtom GW, copy `charmrun` and `gw_bse` in the `build` subdirectory to the dataset subdirectory (`Si2`) and run with this command:

```
./charmrun +p1 ./gw_bse config
```

You can change the number of processors by changing `p1` to `pX` where `X` is the number of processors. The output using 1 processor run is available in `output` sdirectory.

7 Simulation Keyword Dictionary

The simulation keyword file contains 5 subsections (we call them meta-keywords):

```
~gen_GW[ ]
~GW_epsilon[ ]
~GW_sigma[ ]
~GW_file[ ]
~GW_parallel[]
```

Each meta-keyword requires keywords and key-arguments.

If multiple options are available, boldface indicates a default value.

~gen_GW [7 keywords]

1. \num_tot_state{1} :
   Total number of states including occupied and unoccupied (also known as the number of bands in other software approaches). In direct reference to the next two options, note that num_tot_state = num_occ_state + num_unocc_state

2. \num_occ_state{1} :
   Number of occupied states (i.e., number of valence bands)

3. \num_unocc_state{1} :
   Number of unoccupied states (i.e., number of conduction bands)

4. \num_kpoint{1} :
   Number of k points.

5. \num_qpt_all{**on**,off} : If on, calculate epsilon matrix for all q points which differences between the k vectors. If off, it will look for \qpt{} keywords to see which q points to calculate.

6. \qpt{0,2,3} : If \num_qpt_all{off}, user asked to select which q points of epsilon matrix are calculated. No default value for this keyword. Zero-based indexing is used.

7. \statefile_binary_opt{**off**,on,off_gzip,on_gzip} :
   How the wavefunction coefficients are stored in STATES files. If you have converted the wavefunctions from Quantum Espresso, the option should be **off**.

   - off : plain ASCII text format
   - on : binary
   - off_gzip : plane ASCII but compressed by gzip
   - on_gzip : binary and compressed by gzip

   — planned for the near future—
   \num_spin : number of spin
   \coulb_trunc_opt : Coulomb truncation options.

~GW_epsilon [4 keywords]

1. \EcutFFT{1} :
   Wavefunctions are fast-Fourier transformed (FFTed) from reciprocal to real space for computation of the polarizability. This energy cutoff (in Ryd) determines the spatial resolution used and thus the size of the FFT grid. This is a physical convergence parameter.

2. \Ecuteps{1} :
   Energy cutoff for epsilon matrix (in Ryd). Determines the size of the epsilon matrix. This number has to be less than or equal to EcutFFT.

3. \tol_iter_mtxinv{0.001} :
   We invert ϵ matrix using an iterative matrix inversion method.[4] The code compares all of the matrix elements between N^{th} and $N+1^{th}$ iteration, and if the largest difference is below this tolerance, matrix inversion routine terminates.

4. \max_iter{100}:
   Maximum number of iteration for the matrix inversion. If maximum number of iteration used, it does not perform iteration even if the tolerance is not achieved.

[4] Adi Ben-Israel. An iterative method for computing the generalized inverse of an arbitrary matrix. Math. Comp., 19:452–455, 1965.

~GW_sigma [0 keyword]

— planned for the near future—
\screened_coulomb_cutoff{1} : Energy cutoff for screened coulomb interaction (in Ryd).
\PP_num_mode{1} : number of Plasmon-Pole modes to be summed over

~GW_io [18 keywords]

Following options can be used to read and write P Matrix, Epsilon and Epsilon Inverse matrices to file. Each row of a matrix is read from or written to a file with suffix _row${row_index}.

1. \p_matrix_read{on,**off**} :
   If on, read in P matrix from file.

2. \p_matrix_read_prefix{PMatrixIn} :
   Specify the directory with P matrix files if p_matrix_read is on.

3. \p_matrix_write{on,**off**} :
   If on, write P matrix to file.

4. \p_matrix_write_prefix{PMatrixOut} :
   Specify the directory for P matrix files if p_matrix_write is on.

5. \p_matrix_verify{on,**off**} :
   If on, compare and verify P matrix with data in file.

6. \p_matrix_verify_prefix{PMatrixIn} :
   Specify the directory for P matrix verification files if p_matrix_verify is on.

7. \epsilon_read{on,**off**} :
   If on, read in Epsilon matrix from file.

8. \epsilon_read_prefix{EpsIn} :
   Specify the directory with Epsilon matrix files if epsilon_matrix_read is on.

9. \epsilon_write{on,**off**} :
   If on, write Epsilon matrix to file.

10. \epsilon_write_prefix{EpsOut} :
    Specify the directory for Epsilon matrix files if epsilon_matrix_write is on.

11. \epsilon_verify{on,**off**} :
    If on, compare and verify Epsilon matrix with data in file.

12. \epsilon_verify_prefix{EpsIn} :
    Specify the directory for Epsilon matrix verification files if epsilon_matrix_verify is on.

13. \epsilon_inv_read{on,**off**} :
    If on, read in Epsilon inverse matrix from file.

14. \epsilon_inv_read_prefix{EpsInvIn} :
    Specify the directory with Epsilon inverse matrix files if epsilon_inv_matrix_read is on.

15. \epsilon_inv_write{on,**off**} :
    If on, write Epsilon inverse matrix to file.

16. \epsilon_inv_write_prefix{EpsInvOut} :
    Specify the directory for Epsilon inverse matrix files if epsilon_inv_matrix_write is on.

17. \epsilon_inv_verify{on,**off**} :
    If on, compare and verify Epsilon inverse matrix with data in file.

18. \epsilon_inv_verify_prefix{EpsInvIn} :
    Specify the directory for Epsilon inverse matrix verification files if epsilon_inv_matrix_verify is on.

~GW_parallel [4 keywords]

1. \pipeline_stages{1} :
   Number of unoccupied states to concurrently send during P matrix computation. Increasing the number of stages should yield better CPU utilization, however each additional stage requires memory equal to the size of all occupied states. After a certain number of stages the benefit diminishes as the CPU gets saturated.

2. \rows_per_chare{1} :
   Parameter controlling the decomposition of the P matrix. Each chare holds a tile of the matrix with this many rows. Larger tiles reduces some scheduling overhead but allows for less overlap of communication and computation. May also effect cache utilization. Creating enough tiles to have 4-16 per core is a good starting point.

3. \cols_per_chare{1} :
   See \rows_per_chare.

4. \transpose_stages{1} : when transposing the P matrix, how many slices to cut the P matrix into in order to do the transposition in pieces to avoid memory overflow.