stochasticGW – User manual and Tutorial (version 3.0)

Minh Nguyen,[1] *Christopher Arntsen,*[2] *Wenfei Li,*[1], *Vojtěch Vlček,*[3], *Phillip Thomas*[4] *and Daniel Neuhauser*[1]

[1] *Department of Chemistry and Biochemistry, University of California, Los Angeles California 90095, USA.*

[2] *Department of Chemistry, Youngstown State University, Youngstown, Ohio 44555*

[3] *Department of Chemistry and Biochemistry, University of California, Santa Barbara California 93106, USA*

[4] *National Energy Research Scientific Computing Center, Lawrence Berkeley National Laboratory, California 94720, USA*

Contents

1 Introduction 3

2 License 4

3 Overview 5

4 Code status 7

5 Downloading, compilation and installation 8

5.1 Downloading and installation 8

5.2 Compilation 8

5.3 Implementation 9

6 Input files 10

6.1 Brief overview of the input files 10

6.2 Detailed description of the files and flags 10

7 Output 20

8 Utilities 22

8.1 Extracting wavefunctions from Quantum ESPRESSO runs 22

9 Tutorial 23

9.1 H_2: Combined Stochastic and Deterministic treatment 23

9.2 H_2: Extracting wavefunctions from Quantum ESPRESSO and running GW/PBE 26

9.3 C_{60}: Fully stochastic calculation 29

9.4 Graphene: orbital energies of a 2D material 31

9.5 Si-214: GPU-accelerated Gapped-Filtering calculation 33

10 Acknowledgements 37

1 Introduction

This manual details *STOCHASTICGW*, a software for large scale linear-scaling G_0W_0 stochastic calculations. The code combines the success of the G_0W_0 approximation, a quantitative approach for vertical ionization energies and electron affinities, with the ability of stochastic methods to tackle very large systems. The methodology was introduced and detailed in several recent references, especially:

Ref. I : D. Neuhauser, Y. Gao, C. Arntsen, C. Karshenas, E. Rabani and R. Baer, Breaking the theoretical scaling limit for predicting quasiparticle energies: The stochastic GW approach, Phys. Rev. Lett., 113, 076402 (2014).
`https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.113.076402`

Ref. II : V. Vlcek, E. Rabani, D. Neuhauser and R. Baer, Stochastic GW calculations for molecules, J. Chem. Theo. Comput., 13, 4997 (2017).
`https://arxiv.org/abs/1612.08999`

Ref. III : V. Vlcek, W. Li, R. Baer, E. Rabani and D. Neuhauser, Swift GW beyond 10,000 electrons using sparse stochastic compression, Phys. Rev. B, 98, 075107 (2018).
`https://doi.org/10.1103/PhysRevB.98.075107`

Ref. IV : M. Nguyen and D. Neuhauser, Gapped-filtering for efficient Chebyshev expansion of the density projection operator, Chem. Phys. Lett. 806, 140036 (2022).
`https://doi.org/10.1016/j.cplett.2022.140036`

→ Sections: 3, 5 & 9

If this is your first time using this manual, read the Overview and Downloading and compilation section, and then jump to the Tutorial.

2 License

3 Overview

stochasticGW calculates a matrix element of the self-energy in the G_0W_0 approximation

$$\Sigma(\omega) = \left\langle \phi \middle| \hat{\Sigma}(\omega) \middle| \phi \right\rangle, \tag{1}$$

where ϕ is a single particle orbital of the molecule studied (typically a HOMO or LUMO), associated with a DFT Hamiltonian $\hat{H}$; $\hat{\Sigma}$ is the self-energy operator, composed of a static exchange component (X) and a polarization part (P)

$$\left\langle \phi \middle| \hat{\Sigma}(\omega) \middle| \phi \right\rangle = \left\langle \phi \middle| \hat{\Sigma}_X \middle| \phi \right\rangle + \left\langle \phi \middle| \hat{\Sigma}_P(\omega) \middle| \phi \right\rangle, \tag{2}$$

where the latter is a Fourier transform

$$\left\langle \phi \middle| \hat{\Sigma}_P(\omega) \middle| \phi \right\rangle = \int e^{i\omega t} \left\langle \phi \middle| \hat{\Sigma}_P(t) \middle| \phi \right\rangle \mathrm{d}t. \tag{3}$$

In the G_0W_0 approximation the coordinate representation of $\hat{\Sigma}_P(t)$ is extremely simple:

$$\Sigma_P(\mathbf{r}, \mathbf{r}', t) = G_0(\mathbf{r}, \mathbf{r}', t)\, W_P\left(\mathbf{r}, \mathbf{r}', t^+\right), \tag{4}$$

where we introduced the non-interacting Green's function and the dynamical part of the effective interaction.

The starting point (used for G_0 and W), is a Kohn-Sham Hamiltonian which is constructed from ground state orbitals ϕ. At the moment (version 3.0) the *stochasticGW* code is limited to an LDA starting point – this restriction will be lifted in the upcoming versions of the code. Orbitals ϕ are represented on a real-space grid and have to be supplied by the user.

The matrix element of the self-energy is evaluated as a statistical average over multiple stochastic realizations - the number of Monte Carlo samples. The stochastic approach uses multiple resolutions of the identity with random functions for several purposes:

- Separating the product of G_0 and W_P, converting the calculation of Σ_P into evaluating a matrix element of W_P.
- Evaluating a matrix element of W_{PR} (where the R subscript indicates a causal, i.e., retarded effective interaction) by stochastic TDH (time-dependent Hartree, yielding RPA) or stochastic TDDFT .
- Converting the matrix elements of W_{PR} to matrix elements of W_P (the time-ordered effective interaction) through a stochastic fragment basis approach (Ref. III).

The eventual method scales practically linearly (or better) with system size; for details of the method and scaling read Refs. I & II and, for the closest description to the present version, Ref. III

The program outputs the matrix element of the polarization self-energy, $\left\langle \phi \middle| \hat{\Sigma}_P(\omega) \middle| \phi \right\rangle$, over a wide range of frequencies, as well as the exchange self-energy $\left\langle \phi \middle| \hat{\Sigma}_X \middle| \phi \right\rangle$. The code also prints the estimate of the quasiparticle energy of state ϕ for the given number of Monte Carlo samples, obtained from solving the quasiparticle equation:

$$\varepsilon = \varepsilon_{\mathrm{KS}} + \left\langle \phi \middle| \hat{\Sigma}_X + \hat{\Sigma}_P\left(\varepsilon\right) - \hat{v}_{xc} \middle| \phi \right\rangle, \tag{5}$$

where $\varepsilon_{\mathrm{KS}}$ is Kohn-Sham eigenvalue of the eigenstate ϕ and $\hat{v}_{xc}$ is the (mean-field) exchange-correlation potential. The `input` and output are detailed in the sections below.

4 Code status

Several items in this version have not been implemented or tested. Specifically, note that:

- The code has not been tested for open-shell systems.
- The core-shell exchange correction is implemented but not tested.
- Only a local LDA exchange-correlation potential is implemented directly in the code. A wrapper linking to LibXC allows one to use other exchange-correlation potentials.
- The numbers of points in each dimension, `nx`, `ny`, `nz` need all to be even.
- The maximum angular momentum in the nonlocal pseudopotentials needs to be `1` or `2` or `3`. If it is `1` then it is necessary to have `lloc=2`.

5 Downloading, compilation and installation

5.1 Downloading and installation

STOCHASTICGW is a Fortran code requiring MPI parallelization; The source code is available from:
https://github.com/stochasticGW/stochasticGW

For installation, first clone the repository:

```
git clone https://github.com/stochasticGW/stochasticGW.git
```

Upon unpacking, you will get a directory `stochasticGW-GPU` with a README file and 5 subdirectories:

```
examples/  PP/  README.md scripts/  src/  stochasticGW_license.pdf
utils/
```

The installation of the package is detailed then in `src/README` and is repeated here.

! → An installation of the `FFTW` library (version 3.1 or higher) is a prerequisite; it can be downloaded from http://www.fftw.org.

5.2 Compilation

In the `scripts` directory you will find a sample `Makefile` that needs a little modification depending on your system settings, as described below.

Copy our choice of sample `Makefile` from `scripts` to `src/Makefile` and specify your parallel compiler (`FCMPI`), the paths to your `FFTW`, CUDA and CUDA-math libraries (`FFTPATH` `CUFLG`, and `CULIB`, respectively), and whether you are building for GPUs and with LibXC (`USEGPU` and `USELIBXC`, respectively). The flags for the parallel compiler (`MPIFLG`) and preprocessor flags (`PREPROCFLG`) can usually be left unaltered.

A sample header of `makefile` reads:

```
FCMPI        = mpifort
MPIFLG       = -DMPI -O3
PREPROCFLG   = -Mpreprocess
USEGPU       = yes
USELIBXC     = no
FFTPATH      = -lfftw3
CUFLG        = -lcudart
CULIB        = -lcufft
```

The *STOCHASTICGW* code is built simply by executing:

```
make
```

in the `src` directory.

After successful compilation, the *STOCHASTICGW* executable, either `sgw_gpu.x` or `sgw_cpu.x`, will be found in the `src` directory.

Make sure `sgw_cpu.x` or `sgw_gpu.x` has been created

```
ls -l sgw*.x
```

5.3 Implementation

Now you are ready to run the code! For example, consider the H_2 test case. Navigate to the example directory ∼/`stochasticGW-GPU/examples/H2` and follow the instructions in the `README.md` file.

```
cd ~/stochasticGW-GPU/examples/H2
./01-link_files.sh
cd 2-sgw
```

The script `01-link_files.sh` makes symbolic links to input files from common directory ∼/`stochasticGW-GPU/examples/common`, `counter.inp` and `random.inp`, along with a link to the pseudopotential directory ∼/`stochasticGW-GPU/PP` and to the executable, `sgw_gpu.x`. At this point, your `2-sgw` directory should contain the following:

```
cnt.ini
counter.inp
INPUT
REF_sGW_H2.out
PP
random.inp
README.md
sgw_gpu.x
wf.txt
```

All these files are inputs except for `REF_sGW_H2.out` and `README.md`; the latter has additional instructions for running.

To run the H_2 example (from the Tutorial) on 4 cores, run

```
mpirun -n 4 ./sgw.x >& log &
```

The output file `log` should match `REF_sGW_H2.out`.

After the H_2 test is finished, you can run any other molecule/material you want. You'll need to change a few items, detailed below and in the Tutorial.

- Change INPUT as needed – especially make sure that you change `ntddft` to be between 8 and 30.
- Change the atomic coordinates (`cnt.ini`).
- Bring your own `wf.txt` (or better yet, `wf.bin`) file that contains the occupied DFT energies and functions. Alternatively, place atomic coordinates and DFT wavefunctions in `sgwinp.txt`.
- And remember to include in `PP/` any pseudopotential you need.

6 Input files

6.1 Brief overview of the input files

Two groups of input files are required in the working directory. One set includes general parameters, and the other system-specific files.

Seed, labels and flags

`random.inp` :	seed file for the random number generator.
`counter.inp` :	a label file with a counter for modifying the random number input and for labeling the input. Using this file simplifies working simultaneously on multiple runs.
`INPUT` :	general input file with flags and parameters.

System specific files

`cnt.ini` :	atomic position file (in Angstroms or atomic units).
`*UPF/*fhi` :	pseudopotential files (only two formats are currently supported). These files should be placed in a subdirectory of the working directory, `PP/`
`wf.txt` or `wf.bin` :	text or binary file with all occupied (and possibly a few unoccupied) wavefunctions.
`orb.txt` (optional) :	an orbital wavefunction ("ϕ" in the previous discussion) for which the self-energy matrix element is needed. Usually this file will not be given and instead ϕ will be extracted from `wf.txt` or `wf.bin`.
`orbj.txt` (optional) :	text file containing the j^{th} orbital wavefunctions for which the general matrix elements of the Σ operator, $\left\langle \phi_j \middle| \hat{\Sigma} \middle| \phi_i \right\rangle$, are desired, where the i^{th} orbital is either taken from `orb.txt` or from `wf.txt` or `wf.bin`.
`sgwinp.txt` (optional) :	text file containing the coordinates of the atoms, grid defined for the density and orbitals, and values for the density and at least one orbital on the grid. This file is an alternative format and should not be given in combination with `cnt.ini` and `wf.txt` or `wf.bin` or `orb.txt`.

6.2 Detailed description of the files and flags

FILE: `random.inp`

A file with 6 lines, each with 4 integer numbers (Fortran format `integer*8`). These are the seeds for the KISS random-number algorithm. Each line in is a seed for a different random variable used in the StochasticGW algorithm.

FILE: `counter.inp`

A file with a single integer that labels the output directories and files and also modifies the seed. It alleviates the need for different runs to use different random.inp files when the calculation is repeated to reduce the statistical error (see example below)

Example

Let's say we want `nmctot`=2000 stochastic samples.[a] This can be achieved by a single run with 2000 stochastic samples, or perhaps, due to computer jobs scheduling issues, it may be easier to submit 5 different jobs, each with 400 stochastic samples.

For these 5 jobs we would need, in principle, 5 different `random.inp` files with totally different values, which is cumbersome. We therefore use `counter.inp`. Each counter modifies the values of the variables from `random.inp`. So in our example we would only need to submit 5 different jobs, each with the same `random.inp` file, and in each run the `counter.inp` file contains a different integer from, say, 0 to 4 (or any other 5 different integers).

Then, at the end of the 5 runs, average the 5 output quasiparticle energies or 5 output self-energies files.

[a] see also the `buffer_size` variable, which controls the parallelization of multiple stochastic states.

FILE: `INPUT`

Main file for flags and parameters. For most variables the default values are appropriate. Examples are shown in the Tutorial.

Specifically, only one set of variables has to be specified in INPUT: the names of the pseudopotentials (variable: `pps`). Other parameters need to be included in the `INPUT` file only if they differ from their default values. The format for most variables (except for the pseudopotential names) is:

```
variable name
<value>
<empty line>
```

The order is not important, but no variable should be repeated; lines starting with `#` are ignored. The input variables are:

`pps` : a label indicating the beginning of the list of pseudopotentials. After the last pseudopotential put a line starting with `#`. The pseudopotentials listed must be present in a subdirectory of the working directory (where the `INPUT` file is) called `PP/`. It is recommended to use well-tested pseudopotentials. At present, the supported formats are *UPF and *fhi. In addition, the base code currently supports only the LDA functional. Other functionals can be used by linking to the `LibXC` library.

! → Note that the program will issue a warning if the pseudopotential wavefunctions are not properly normalized, but it will then normalize them properly.

`scratch` : (default: `GW_SCRATCH`) the prefix of the name of the scratch directory; the scratch directory is then appended by the counter (from the `counter.inp` file).

Example

If `counter.inp` contains the number `0`, and the relevant segment from INPUT is

```
scratch
/scratch/john/GW_SCRATCH
<empty line>
```

then the scratch directory will be `/scratch/john/GW_SCRATCH.0`

The scratch directory contains several intermediate files. It should be ideally not placed on the main shared disk but on fast disks connected to each core, if such disks are available.

Each core will try to create this scratch directory, unless it was created earlier by another core accessing the same scratch disk.

`nmctot` : (default: 1000) the number of stochastic realizations used for statistical averaging (Monte Carlo iterations). Typically 300-2000 iterations are needed for acceptable convergence. The error in the output self-energy and quasiparticle energies decreases as $\frac{1}{\sqrt{\texttt{nmctot}}}$.

If `nmctot` is larger than the number of MPI processes $\texttt{N}_{\texttt{proc}}$ the calculation will be repeated ($\texttt{nmctot}/\texttt{N}_{\texttt{proc}}$) times. For instance: if you set $\texttt{nmctot}=1000$ and distribute the job over $\texttt{N}_{\texttt{proc}}=100$ cores, the calculation will be repeated 10 times on each core.

If needed, the program increases `nmctot` so that $\texttt{mod}(\texttt{nmctot}, \texttt{N}_{\texttt{proc}}) = 0$.

`binary` : (default: `.TRUE.`) a logical variable; designating whether the occupied wavefunctions are stored in a binary file `wf.bin` or a text one `wf.txt`.

`dim_periodic` : (default: `0`), either 0, 2, or 3; an integer variable designating the number of spatial degrees of freedom over which the calculation is periodic (restricted to Γ-point sampling) Selecting `2` implies periodicity over x, y directions.

`box` : (default: `pos`) a character*3 variable designating the way the calculation box is constructed. Two values are allowed, `pos` and `sym` .

The first choice, `pos`, implies that the calculation box is (in Bohr!) `[0,nx*dx][0,ny*dy][0,nz*dz]`. The wavefunctions in `wf.txt` are then interpreted to start at the corner of this box, `(0,0,0)`. Such box convention is used in Quantum Espresso and most planewave codes. Note that in our code the coordinates in the `cnt.ini` file need to fit in the box (or more preceisely, if they are given in Angstrom, their values in Bohr need to fit in the box). If they do not fit in the box the program will stop.

The second, `sym`, assumes that the box of the calculation is

`[-nx*dx/2,nx*dx/2][-ny*dy/2,ny*dy/2][-nz*dz/2,nz*dz/2]` (again in Bohr), i.e., its center is the origin. The wavefunctions in `wf.txt` are again interpreted to start at the corner of the box, which is now `(-nx*dx/2,-ny*dy/2,-nz*dz/2)`. Again, the coordinates in `cnt.ini` need to fit in the box (once expressed in Bohr). This symmetric box convention is more natural for non-periodic molecules.

`multiplicity` : (default: 1) either 1 or 2; 1 indicates a closed-shell calculation, 2 an open shell. **Note: the program was not tested for open-shell systems, and at this stage should be trusted only for closed-shell calculations**.

`ekcut` : (default: 25.0) a cutoff (in Hartree) on the kinetic energy. Even though there is a formal maximum on the kinetic energy on the grid (and the grid is supplied elsewhere, see below) it is numerically better to use a lower cutoff. In practice one should use a value somewhat higher than the depth of the pseudopotentials. For calculations with a new element check the convergence of the calculations with `ekcut`=15 to 30 Hartree.

`gamma` : (default: 0.06) energy broadening parameter for Σ_P (Section 3 in Ref. II and the supplementary material in Ref. I). Generally 0.06 should be fine. An increased `gamma` reduces the computational effort and improves somewhat the statistics, but if `gamma` is too large the broadening will be too severe leading to wrong quasiparticle predictions.

`orb_indx` : (default: -1). If `orb_indx`>0 its value designates the orbital ϕ which will be used in the calculation of the self-energy (e.g., if `orb_indx`=5 then ϕ is the 5th eigenstate). If `orb_indx` is -1 then ϕ should be read from `orb.txt`.

`nj` : (default: `0`) Number of j orbitals for which one wants to evaluate the Σ matrix element $\left\langle \phi_j \middle| \hat{\Sigma} \middle| \phi_i \right\rangle$ where the i^{th} orbital is specified by `orb_indx`. If `orbj_indx` is included, then this variable must either be omitted or match the number of entries in `orbj_indx`. If `orbj_indx` is omitted then this variable must either be omitted or match the number of orbitals in `orbj.txt`.

`orbj_indx` : (default: not used) List of integers specifying the indices of the j orbitals used for evaluating the matrix elements of the Σ operator, $\left\langle \phi_j \middle| \hat{\Sigma} \middle| \phi_i \right\rangle$. Relevant only if `sgwinp.txt` is present; otherwise this list must be omitted and j orbitals are read from `orbj.txt` instead. Each integer in the list must correspond to an orbital contained in `sgwinp.txt`.

`ntddft` : (default: 15) the number of stochastic states used in the time-dependent RPA or TDDFT calculation (in the W part of GW). For very small systems, use ~ 30. For large systems with hundreds of electrons or more, it is enough to use ~ 8 orbitals.

For open-shell systems (`multiplicity`=2) `ntddft` denotes the number of stochastic TDRPA/TDDFT states for each spin.

! $\rightarrow$ For a non-stochastic TDRPA/TDDFT propagation, use `ntddft` = -1. Then a deterministic TDRPA/TDDFT calculation is used for calculating the action of W. In this case all occupied orbitals are excited and propagated. This choice should only be done for small systems, since it increases the computational time considerably. A non-stochastic calculation of the action of W is neither feasible nor needed for large systems, since the stochastic calculation of the action of W is getting progressively better for bigger systems.

`projection` :	(default: `.FALSE.`) a logical variable; when set to `.TRUE.` stochastic orbitals are generated by projecting random orbitals onto the set of occupied states. When set to `.FALSE.`, stochastic orbitals are generated by applying a Chebyshev filter to the set of random orbitals.
`mu` :	(default: not used); a floating point variable, that, when specified, determines the rate of cutoff of a temperature-dependent Heaviside filter. When omitted, a Gapped filter is used instead.
`Tp` :	(default: `0.01`) Temperature (in Kelvins) used to define the Heaviside filter used to generate stochastic orbitals. Only relevant when `projection` is turned off and `mu` is provided.
`nchb` :	(default: `-1`) Number of terms in the Chebyshev expansion of the filter. Higher values increase the quality of the initial stochastic states but increase the computational cost of filtering. Only relevant when `projection` is turned off.
`HOMO` :	(default: not used) DFT energy of the Highest Occupied Molecular Orbital; required for constructing the filter only when Gapped filtering is used to prepare the initial stochastic states. If omitted from `INPUT` the program will try to read this value from `sgwinp.txt` instead.
`LUMO` :	(default: not used) DFT energy of the Lowest Unoccupied Molecular Orbital; required for constructing the filter only when Gapped filtering is used to prepare the initial stochastic states. If omitted from `INPUT` the program will try to read this value from `sgwinp.txt` instead..
`usegpu` :	(default: `.FALSE.`) Set to `.TRUE.` to enable GPU execution. The GPU implementation can be used to accelerate the core filtering and propagation routines of the code. Note that GPU routines have not been implemented for all calculation types; in particular, the GPU-accelerated code cannot be used when `flgdyn` is set to `.TRUE.` or when `nxi` < 1.
`units_nuc` :	(default: `Bohr`) either `Angstrom`, or `Bohr`. Controls the units for the nuclear positions input (in the `cnt.ini` file).
`flg_dyn` :	(default: `.FALSE.`) logical variable; `.FALSE.` implies an RPA calculation; `.TRUE.` yields a TDDFT calculation of the action of W where the exchange-correlation potential is updated at each time-step.

The following variables could usually be safely kept at their default values: :

`dt` :	(default: 0.05) time step in atomic units for the TD propagation of W (note: 0.05 a.u. $\simeq$ 1.2 attosec). If desired, `dt` could be reduced to 0.03, although 0.05 generally works well.
`nxi` :	(default: 10,000) Number of random spanning vectors (denoted N_ξ in Ref.-III). Usually there's no need to change the default.
`segment_fraction` :	(default: 0.003) The ratio of the size of each stochastic vector ξ to the total grid length. See Ref. III. Usually there's no need to change the default.
`sm` :	(default: 0.0001) The perturbation parameter (denoted τ in Eq. 29 in Ref. II). The results should not vary for `sm` in the range 10^{-5}–10^{-3}.

`scale_vh` : (default: 2; but will be set to 1 by the program if periodic) allowed values 1 or 2. Controls whether a Martyna-Tuckerman[1] grid-doubling is used in the TDRPA/TDDFT calculations. If the calculation is periodic, `scale_vh` would be set in the program to 1 regardless of the input value. For non-periodic molecules, `scale_vh` formally needs to be set at 2. So since the program automatically sets `scale_vh`, there is usually no need to touch this variable.

`nrppmx` : (default: 1200) an upper bound on the number of radial grid points for the pseudopotential. No need to change unless a new pseudopotential with more than 1200 grid points is used.

`buffer_size` : (default: 1) the number of cores used in each independent stochastic runs. For intermediate or large size systems (hundreds or thousands of electrons) choose the default (1) which uses the least total CPU resources. But for very large systems, the wall-time of the calculation could be too long when using a `buffer_size` of 1. Generally, it is a good idea to use a `buffer_size` which either equals to the number of cores on each CPU, or divides it (e.g., if there are 12 cores on each CPU, use a `buffer_size` of 1,2,3,4,6 or 12).

Also, if `buffer_size` is >1, aim at having `mod(ntddft+1,buffer_size)`$=0$. For instance: if `buffer_size` is 12, use `ntddft`$=11$, 23 or 35. If `ntddft` is bigger the convergence is somewhat faster, but the effort grows, so values in the range 8-30 are usually the most efficient.

Example

Let's say that a giant system requires about 1000 Monte Carlo iterations. One could use 1000 cores and then each core will be used for a single Monte Carlo iteration. But say that such a single calculation took 2 days (i.e., each core takes 2 days), while the computer-center allocation limits each run to be 1 day long. In that case, it is necessary to spread each single calculation to several cores. For example, use then a `buffer_size` of 4 and 4000 mpi cores to do the 1000 Monte-Carlo iterations, and each group of 4 cores will do a single stochastic run. The calculation will then take a shorter wall-time.

Note: to spread the work in each separate run to several cores, we use the `mpi_comm_split` routine, which divides the cores to subsets, usually labeled as `colors`. Further details can be found on the web. [a]

[a] a good introduction is in http://www.bu.edu/tech/support/research/training-consulting/online-tutorials/mpi/more/mpi_comm_split/

FILE: `cnt.ini`

Nuclear positions in a Cartesian format:

```
<element name>  <x-coordinate>  <y-coordinate>  <z-coordinate>
```

[1] G.J. Martyna, and M.E. Tuckerman, "A reciprocal space based method for treating long range interactions in ab-initio and force-field-based calculation in clusters", J. Chem. Phys. 110, 2810 (1999),doi:10.1063/1.477923

The element name is in a 1- or 2-character format (e.g., H or Si). The nuclear units could be in Angstrom or Bohr (the units are specified by the `units_nuc` variable, the default of which is Bohr), but recall that all other quantities in the code are calculated in atomic units, i.e., Hartree for energies and Bohr for distances.

FILE: `wf.txt`

A wavefunction input file (required if `binary` = `.FALSE.`) with all occupied (and potentially a few unoccupied) eigenstates. The first 6 lines detail the grid used (lengths in Bohr), followed by the multiplicity (which needs to match the default value or, if given, the value in INPUT), and the Kohn-Sham eigenvalues. Then each wavefunction is given, preceded by the state-indicies (and possibly the spin-index, if `multiplicity=2`). The file contains the state index on the 12th, 14th, . . . lines:

```
nx                         <number of points along the x direction>
ny                         <number of points along the y direction>
nz                         <number of points along the z direction>
dx                         <grid spacing in the x direction>
dy                         <grid spacing in the y direction>
dz                         <grid spacing in the z direction>
nsp                        <multiplicity>
nstates                    <number of states in this wf.txt file>
evls
<kohn-Sham eigenvalues>
orbitals
1                          <spin-index of 1st orbital>
<1st orbital>
2                          <spin-index of 2nd orbital>
<2nd orbital>
⋮
```

Note that the list of Kohn-Sham eigenvalues should include all "nstates" eigenvalues.

The program determines the number of states to be read based on the ionic charges provided in the pseudopotential files. If the normalization of any orbital is incorrect a warning is issued but the program continues.

The orbital is given as usual in Fortran with the left-most index (x) varying the fastest. Each orbital should contain `nx` · `ny` · `nz` points.

If the multiplicity is 2 (open-shell), the spin-index of each orbital should be 1 or 2 (indicating an α or β spin). The spin-index is not required if the multiplicity is 1 (closed-shell).

FILE: `wf.bin`

The wavefunction input file (required if `binary` = `.TRUE.`). Analogous to `wf.txt`. When preparing this binary file from your favorite DFT program output, you should follow the example of the following code segment:

```
real*8 hw(1:nstates)
real*8 orbitals(1:ngrid,1:nstates)
...
open(9,file='wf.bin',status='old',form='unformatted')
write(9)'nx       ',nx
write(9)'ny       ',ny
write(9)'nz       ',nz
write(9)'dx       ',dx
write(9)'dy       ',dy
write(9)'dz       ',dz
write(9)'nsp      ',nsp
write(9)'nstates  ',nstates
write(9)'evls     '
write(9)(hw(i),i=1,nstates)
write(9)'orbitals '

do i=1,nstates
! ispin: 1 for closed shell; 1 or 2 for open shell.
      write(9)i, ispin(i)
      write(9)u(1:ngrid,i)
enddo
close(9)
```

Each text string in lines 1-9 and 11 is of length 9.

At present, the user needs to prepare `wf.bin` or obtain this file from the output of Quantum ESPRESSO (see Tutorial section below); future versions may include routines that process the output from other DFT programs to produce the proper `wf.bin` file.

FILE: `orb.txt`

Orbital ϕ for which the quasiparticle energy is sought. This orbital is usually taken from a DFT code. Typically it will be the HOMO or the LUMO. The file is organized similarly to the wavefunction file, and the heading values must match those in `wf.txt` or `wf.bin`

```
 nx   <number of points along the x direction>
 ny   <number of points along the y direction>
 nz   <number of points along the z direction>
 dx   <grid spacing in the x direction>
 dy   <grid spacing in the y direction>
 dz   <grid spacing in the z direction>
 nsp  <multiplicity>
 orb  <kohn sham orbital energy for this specific orbital>
      <values of the orbital φ on the grid>
```

The program issues an error warning if the normalization of ϕ is wrong (but will not stop).

If the `orb_indx` variable is specified, with a value bigger than 0, then `orb.txt` is not needed and ϕ will be taken directly from the wavefunction file.

FILE: `orbj.txt`

Orbitals ϕ_j for which Σ matrix elements $\left\langle \phi_j \middle| \hat{\Sigma} \middle| \phi_i \right\rangle$ are sought. These orbitals are usually taken from a DFT code. The file is organized similarly to the wavefunction file, and the heading values must match those in `wf.txt` or `wf.bin`

```
nx    <number of points along the x direction>
ny    <number of points along the y direction>
nz    <number of points along the z direction>
dx    <grid spacing in the x direction>
dy    <grid spacing in the y direction>
dz    <grid spacing in the z direction>
isp   <multiplicity>
orbj
      <values of first orbital φj on the grid>
      <values of second orbital φj on the grid>
...
      <values of last orbital φj on the grid>
```

FILE: `sgwinp.txt`

The file `sgwinp.txt` contains the atomic positions, the grid definitions, one or more orbitals, and the density in a unified format. This can be useful for transferring a selection of orbitals from a DFT code, such as Quantum ESPRESSO, to STOCHASTICGW. The file contains three sections which may be listed in arbitrary order: `$GEOMETRY`, `$GRID`, and `$ORBITALS`:

```
$GEOMETRY
<element name>  <x-coordinate>  <y-coordinate>  <z-coordinate>

$GRID
nx    <number of points along the x direction>
ny    <number of points along the y direction>
nz    <number of points along the z direction>
dx    <grid spacing in the x direction>
dy    <grid spacing in the y direction>
dz    <grid spacing in the z direction>
nsp   <multiplicity>
homo  <energy of HOMO>
lumo  <energy of LUMO>

$ORBITALS
ORB   <orbital index>  <energy>  <multiplicity>
      <values of first orbital on the grid>
...
ORB   <orbital index>  <energy>  <multiplicity>
      <values of last orbital on the grid>
DENS        0            0.0           0
      <values of density on the grid>
```

FILE: `*UPF/*fhi`

A pseudopotential file is needed for each of the elements in the molecule (i.e., each of the different element in `cnt.ini`). The names of the pseudopotential files are specified by variable `pps`. The pseudopotentials should be placed in a subdirectory, `PP/`

At present the program only handles pseudopotentials with a single Kleinman-Bylander[2] state for each angular momentum. Note that such pseudopotentials are known to have ghost states for heavier elements and some choices of specific pseudopotentials. [3] The user should verify that there are no ghost states for each of the pseudopotentials. This can be verified, for example, with a DFT code that uses the Kleinman-Bylander decomposition (e.g., Quantum ESPRESSO). Such a DFT code should be used for checking for ghost states in very small systems (atoms or small molecules) each of which contains one or more of the atoms in the system with its associated pseudopotentials.

Also note that we have not extensively checked/verified pseudopotentials with core-charge exchange corrections.

Future versions of the code would incorporate more general pseudopotentials.

[2] L. Kleinman and D. M. Bylander, “Efficacious form for model pseudopotentials.", Phys. Rev. Lett. 48, 1425 (1982), https://doi.org/10.1103/PhysRevLett.48.1425

[3] A specific example for a problematic pseudopotential is the usual one for hydrogen when `lmax=3` is used. Therefore, for H use `lmax=1` and `lloc=2`, see the `H.pw-mt_fhi.UPF` pseudopotential in the supplied `PP` directory

7 Output

During the run a standard output (log file) details the input variables, possible errors, warnings, timing and a few additional details, as well as the output energies. We separately supply the self-energy as a function of frequency.

If the calculation is repeated for several Monte-Carlo steps on each core, the variable designating the current step has the range:

$$\texttt{MC\_STEP} = 1, ..., \frac{\texttt{nmctot} \times \texttt{buffer\_size}}{\texttt{ncores}} \tag{6}$$

(see `nmctot` and `buffer_size` for details). The quasiparticle energy is estimated at each cycle together with its stochastic error (see the Tutorial).

Here is an example of the tail of the `REF_sGW_H2.out` file from the Tutorial, for an MPI run using `n_cores` = 8 cores and with `buffer_size` = 1.

```
######## Accumulative # of Monte Carlo steps :  MC=        120 ########
Completed Monte Carlo steps in each core : MC_STEP=         15


                 -0.378528                        E
                 -0.424763                        <V>
         120     -0.646902       0.000000         MC, <X>          +-stat.err
         120      0.018592       0.012779         MC, Sig_R(e_qp)  +-stat.err
         120     -0.018665       0.008623         MC, Sig_R(e_ks)  +-stat.err
         120     -0.000029       0.000007         MC, Sig_I(e_ks)  +-stat.err
         120      0.894335                        MC, Z
         120     -0.593887                        MC, Elin
         120     -0.582075       0.012779         MC, E_qp         +-stat.err
         120      0.003776       0.000547         MC, Ei_qp        +-stat.err

 Time:  MC_STEP:            15 => time:     120.7143
 Time:  program end     120.7146
```

The tail details the `MC_STEP` = 15th calculation step, so
`nmctot` = 120 = `MC_STEP` × `n_cores` total calculations were done.

The DFT orbital energy is `-0.378528`, followed by `<V>` $\equiv \langle\phi|\hat{v}_{xc}|\phi\rangle$.

The following lines are all in the same format: number of Monte Carlo samples (`120` – denoted `MC`; since this is the final step of the calculation, it coincides with the value of `nmctot`); variable name, its statistical error and labels. Specifically:

- Expectation value of the non-local Hartree-Fock exchange, $<\texttt{X}> = \left\langle\phi\middle|\hat{\Sigma}_X\middle|\phi\right\rangle$, calculated deterministically, without statistical error.
- `Sig_R` is the real part of the self-energy evaluated either at the quasiparticle energy (`e_qp`) or at the starting point KS eigenvalue energy (`e_ks`).
- `Sig_I` is the imaginary part of the self-energy.
- `Z` is the renormalization factor.
- `Elin` is the often-used but crude linear extrapolation quasiparticle energy estimate based on a renormalization factor `Z`.
- Finally, `E_qp` and `Ei_qp` are the real and imaginary parts of the quasiparticle energy obtained by properly solving Eq. 5.

The last two lines provide CPU wall-timing in seconds..

The quasiparticle energy can also be read directly from the header of the self-energy curve which is provided in the file `SigW.txt` in the `GW_OUTPUT.X` directory (where `X` is the number in `counter.inp` file). If the run crashes for some reason prior to completion, SigW.txt will contain the most up-to-date information.

Example

Here is a sample `SigW.txt` file header.

```
#
# SigW output from stochastic GW
#        8192  plotting frequencies
#
# MC_STEP (Monte Carlo steps on each core)
#           4
#
# N_cores
#          30
#
# Buffer_size (# cores per indep. run)
#           1
#
# MC (accumulative # Monte Carlo runs) =MC_STEP*(N_cores/buffer_size)
#         120
#
# KS energy
#  -0.378696352
#
#  <X>              d<X>
#  -0.646901488       0.00000000
#
# <vxc>
#  -0.424763411
#
# e_qp_real,       de_qp_real
#  -0.582239330       1.27781946E-02
#
# e_qp_imag,       de_qp_imag
#    3.74985230E-03   5.42658614E-04
#
#     w            SigW_real       SigW_imag        dSigW_real      dSigw_imag
#
```

The header first reports that the file contains the self-energy at `8192` frequency points. It then repeats the information from the output log file on the number of MC steps, the Kohn-Sham energy, the expectation values, the polarization self-energies and the quasiparticle energies. The rest of the file, after the header, contains the frequency-dependent polarization self-energies and their statistical deviation.

Each line of the header is introduced with `#`, so that it is ignored by many plotting programs. Therefore, the polarization part of the self-energy contained in the `SigW.txt` file can easily be visualized (see the example in the Tutorial).

Further, intermediate information is placed in the directory `GW_WORK.X` (where `X` stands for number specified in `counter.inp`); it contains files similar to `SigW.txt` taken at intermediate number of Monte Carlo iterations. It also contains a file (`details_output.txt`) with detailed information on the run that in case of trouble could be useful for debugging. The scratch directory (specified by the `scratch` variable) contains intermediate files that can be safely discarded after the run.

8 Utilities

8.1 Extracting wavefunctions from Quantum ESPRESSO runs

We offer two options for converting Quantum ESPRESSO output into stochasticGW input wavefunctions. The first option uses the BerkeleyGW code. In Section 9.2 we provide an example where we use this approach to extract wavefunctions from Quantum ESPRESSO for an H_2 molecule and perform GW/PBE calculation on the system. The second approach uses wrappers provided with *stochasticGW* and which are located in ∼`/stochasticGW-GPU/utils/qe2sgw/`.

`Option 1`

Apart from Quantum ESPRESSO, Option 1 also requires installation of the BerkeleyGW code, available from their website: `https://berkeleygw.org/download/`. This is a two-step conversion. The first step utilizes the `pw2bgw.x` utility from Quantum ESPRESSO, and the second uses the `bgw2sgw.x` utility from BerkeleyGW. The input descriptions of the two utilities are available at the following links:

- `https://www.quantum-espresso.org/Doc/INPUT_pw2bgw.html` for `pw2bgw.x`
- `http://manual.berkeleygw.org/3.0/bgw2sgw-keywords/` for `bgw2sgw.x`.

Note that Option 1 only produces wavefunction files in binary format.

`Option 2`

This option uses one of the `qe2sgw.x` or `qe2sgw_stepwise.x` wrappers to convert Quantum ESPRESSO wavefunctions directly to *stochasticGW* format. One must compile the wrappers separately from the rest of the *stochasticGW* code. We recommend using the GNU Fortran compiler to build these binaries. See the examples in Sections 9.4 and 9.5 for an example of the ordinary and stepwise wrappers, respectively.

9 Tutorial

In this section we illustrate using two examples how to execute the program, and discuss the meaning of the most important input variables.

! → The examples presented in this section were selected to quickly illustrate how the *STOCHASTICGW* code works. For production runs the user should carefully investigate the convergence with respect to the grid size, spacing and other parameters.

9.1 H_2: Combined Stochastic and Deterministic treatment

In this example, we calculate the ionization potential of an H_2 molecule. Note that **calculations for small molecules require relatively much more effort vs. the extremely large systems for which *STOCHASTICGW* was developed**.

The DFT estimate of the H_2 ionization potential is -10.3 eV, which is too low compared with the experimental result[4] (-15.4 eV) and is corrected by the GW calculation in the next step. Before we can proceed with running the *STOCHASTICGW* code, we need the occupied wavefunctions, `wf.txt`, which is provided in `examples` directory. User can obtain it from other codes, such as Quantum ESPRESSO[5] or others.

Running the *STOCHASTICGW* Code:

The example `INPUT` file is:

```
pps
H.pw-mt_fhi.UPF
#

scratch_path
.

ekcut
15d0

nmctot
120

ntddft
-1

gamma
0.06

binary
F
```

[4] http://webbook.nist.gov/cgi/cbook.cgi?ID=C1333740&Mask=20
[5] http://www.quantum-espresso.org/wp-content/uploads/Doc/INPUT_PP.html

```
orb_indx
1

scale_vh
1

box
sym

projection
T
```

! → The user should be familiar with all the input variables listed here. Note specifically:

- We used `scale_vh`$=1$ to expedite the calculation, although formally it needs to be 2.
- We set `ntddft=-1` to indicate that the time-dependent Hartree (RPA) calculation of the effect of W_P should be deterministic, using here the single occupied state.
- We set `projection=T` to indicate that the initial stochastic orbitals should be generated by projecting onto the set of occupied orbitals. This setting is more efficient for small systems with few occupied states.

As explained in Section 5, the working directory should contain and `INPUT` and several other input files: `wf.txt`, `cnt.ini`, `counter.inp`, `random.inp` as well as the `PP/` pseudopotential directory that needs to contain (at least) the `H.pw-mt_fhi.UPF` hydrogen pseudopotential file.

For simplicity we assume that the working directory contains also the `sgw_cpu.x` or `sgw_gpu.x` executable.

The *stochasticGW* calculation is then run; for example, to run on 8 MPI cores, write

```
mpirun -n 8  ./sgw_gpu.x  >& log
```

The `log` file prints all parameters and information about all files as well as the starting point energy in Hartree units :

```
<phi|H|phi> computed from orbital from file:   -0.3785284
                      vs. eorb read from file:   -0.3786314
```

This orbital energy slightly differs from the DFT input value; this is because the Hamiltonian in the underlying DFT calculation slightly differs from the Hamiltonian constructed in *stochasticGW* .

After the calculation finishes, we look at the end of the `log` file where the quasi-particle energy is provided :

```
          120      -0.582075       0.012779          MC, E_qp          +-stat.e
```

This value, which is the solution of Eq. 5, is in the right ballpark (-15.8 ± 0.3 eV compared to the experimental value of -15.4 eV), but shows significant statistical error. Since the statistical error is generally proportional to $1/\sqrt{\texttt{nmctot}}$, it is necessary to use `nmctot` $\sim 120*(0.3/0.05)^2 \sim 4000$ Monte Carlo iterations for a statistical accuracy of 0.05 eV.

The program finds the ε that solves Eq. 5 self-consistently, automatically, i.e., since $\Sigma(\omega)$ is given at all frequencies the program searches for the energy (ε) where Eq. 5 is fulfilled (or more precisely, the it searches for the closest solution to the Kohn-Sham energy).

Plotting the Output:

It is instructive however to graphically do this procedure, by plotting the real-part of the self-energy from the `SigW.txt` located here in the `GW_OUTPUT.0` directory (where `0` is the number in the `counter.inp` file).

The `SigW.txt` file has a header that, for completeness, essentially duplicates the output in the log file, as presented in Section 7. The header was presented in the 9.

After the header the frequency dependent polarization part of the self-energy is given. We concentrate on the first and second columns showing to the frequency (`w`) and the real part of the self-energy (`SigW_real`, i.e., $\Sigma_{\mathrm{p}}(\omega)$).

Specifically, we use gnuplot[6] to find the graphical solution to Eq. 5. As mentioned, lines starting with `#` are ignored. In the `GW_OUTPUT.X` directory start gnuplot by typing

```
gnuplot
```

and (inside gnuplot) write:

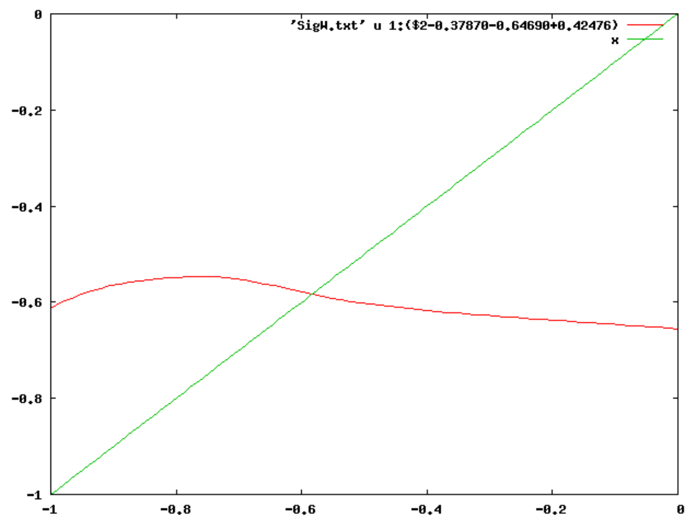
```
gnuplot> set xrange [-1.0:0.0]
gnuplot> set yrange [-1.0:0.0]
gnuplot> p 'SigW.txt' u 1:(-0.378528+0.646902+$2+0.424763) w l, x w l
```

The first two lines set the range of the horizontal ω axis (`x`) and vertical axis to be in a large region in which the quasiparticle energy is expected.

The third line plots the right hand side of Eq. 5, $\varepsilon_{\mathrm{KS}}+\left\langle\phi\middle|\hat{\Sigma}_X+\hat{\Sigma}_P(\varepsilon)-\hat{v}_{xc}\middle|\phi\right\rangle =$ `(-0.378528+0.646902+$2+0.424763)`. Here, \$2 stands for the second column of the file, `SigW_real`$=\Sigma_P(\omega)$. The remaining values in the brackets are the Kohn-Sham orbital energy, exchange part of the self-energy, and negative of the exchange-correlation potential energy, all given in the header.

The second plotted line is the frequency (its values are identical to the horizontal `x` axis). The crossing of these two curves occurs at the quasiparticle energy, as shown below.

[6] http://www.gnuplot.info/



By zooming in we see that the intersection occurs at $\varepsilon = -0.582\,\text{Hartree} = -15.8$ eV, i.e., close to the value found in the `SigW.txt` file.

! $\rightarrow$ If the Kohn-Sham eigenvalue is very far from the predicted quasiparticle energy or the self-energy is very oscillatory, then check the results graphically, since there could be several graphical solutions to Eq. 5.

The calculation should now be repeated with a higher value of `nmctot` to see how the statistical error decreases with the number of MC samples.

In the production runs, carefully check the value of the energy broadening parameter `gamma`.

9.2 H_2: Extracting wavefunctions from Quantum ESPRESSO and running GW/PBE

In this example, we generate wavefunctions from Quantum ESPRESSO and prepare them as input for SGW using the two options described in Section 8.

Running Quantum Espresso:

The example input file `h2.in` for `pw.x` is:

```
&control
   prefix = 'h2'
   calculation = 'scf'
   tstress = .true.
   etot_conv_thr = 1d-5
   forc_conv_thr = 1d-4
/
&system
   ibrav = 0
   nat = 2
   ntyp = 1
   ecutwfc = 28.0
   ecutrho = 112.0
   nbnd = 2
   assume_isolated='mt'
   input_dft='pbe'
   nosym_evc = .true.
/
&electrons
   electron_maxstep = 100
   conv_thr = 1.0d-10
   mixing_mode = 'plain'
   mixing_beta = 0.7
   mixing_ndim = 8
   diagonalization = 'david'
   diago_david_ndim = 4
   diago_full_acc = .true.
/

CELL_PARAMETERS bohr
15.0  0.0  0.0
 0.0 15.0  0.0
 0.0  0.0 15.0

ATOMIC_SPECIES
  H 1.00d0 H.pw-mt_fhi.UPF

ATOMIC_POSITIONS (bohr)
  H  6.8  7.5  7.5
  H  8.2  7.5  7.5
K_POINTS automatic
1 1 1 0 0 0
```

! → `pw2bgw.x` cannot be used along with 'K_POINTS gamma' option of Quantum ESPRESSO. So even if we are carrying out gamma point calculation, the k-point grid must be specified. To run the Quantum ESPRESSO calculation, execute:

```
pw.x -in h2.in >& h2.out
```

Example output of the `pw.x` calculation is stored in `REF_QE_h2.out` file. It contains the orbital energies obtained with Quantum ESPRESSO, given by:

```
highest occupied, lowest unoccupied level (ev):   -10.3826   -0.2590
```

Running the `pw2bgw.x` Utility:

The input file in this example is given by `pw2bgw.in`:

```
&input_pw2bgw
   prefix = 'h2'
   real_or_complex = 2
   wfng_flag=.true.
   rhog_flag=.true.
/
```

Running the command:

```
pw2bgw.x -i pw2bgw.in
```

will generate two files: WFN and RHO, which contains the wavefunction and electron density in BerkeleyGW format. Example output is given in `REF_pw2bgw.out`.

Running the `bgw2sgw.x` Utility

The input file in example is given by `bgw2sgw.inp`. The values of input file parameters are given by:

```
input_wfn_file WFN
input_rho_file RHO
output_rho_file dens.txt
output_wfn_file wf.bin
output_structure_file cnt.ini
number_states -1
precision_low
```

The actual input file also contains description of the variables. This utility will generate `wf.bin`, `dens.txt` and `cnt.ini` files in the format required by the *stochasticGW* code. The `cnt.ini` file generated by `bgw2sgw.x` is centered around the middle of the box. Therefore, it should be used along with option:

```
   box
   pos
```

in the INPUT file while running *stochasticGW* .

Running *stochasticGW* :

To run the *stochasticGW* calculation, copy the files `wf.bin` and `cnt.ini` to the directory of the *stochasticGW* run. The example input file is given:

```
pps
H.pw-mt_fhi.UPF
#

scratch_path
.

functional
101 130

ekcut
15d0

nmctot
120

ntddft
-1

gamma
0.06

binary
```

```
T

orb_indx
1

scale_vh
1

box
pos

projection
T

usegpu
T
```

The output can be interpreted similar to the previous example. A reference output is provided in `REF_sGW_fromBGW.out` where the quasi-particle energy is given by:

```
120      -0.578402      0.011770         MC, E_qp           +-stat.err
```

9.3 C_{60}: Fully stochastic calculation

Here we discuss a fully stochastic calculation of the electron affinity of C_{60}; this system is already big enough to profit from the stochastic treatment, since using all the 120 occupied states for the time propagation is tedious and won't bring a large speedup over conventional approaches. We also illustrate how the timing and accuracy depends on the number of stochastic states (`ntddft`), on the energy broadening parameter (`gamma`), and on the `nxi` and `segment_fraction` parameters,

The user should first finish the previous example and get familiar with preparation of the input files and the *stochasticGW* output.

We use a grid of 80^3 points, with a spacing of 0.4 Bohr. The LDA estimate of electron affinity is 0.1645 Hartree, i.e., 4.48 eV, which is too high compared with the experimental value of 2.69 eV.[7]

This example is found in ~`/stochasticGW-GPU/examples/C60`, with instructions detailed in the accompanied `README.md`.

The `INPUT` file is now:

```
pps
06-C.LDA.fhi
#

scratch_path
.

ekcut
20d0

nmctot
840
```

[7] https://aip.scitation.org/doi/abs/10.1063/1.4881421?journalCode=jcp

```
ntddft
8

units_nuc
bohr

gamma
0.06

binary
T

orb_indx
121

flg_dyn
T

box
sym

projection
T

usegpu
F
```

Note that for the LUMO energy we use a TDDFT effective potential instead of an RPA (i.e., we set `flg_dyn` to be `T`). As explained in Ref. 1, this yields a much better LUMO energy. Note also that `flg_dyn` = `T` must be performed with GPU acceleration disabled.

In addition, the `wf.bin` waveufunction file for C_{60} is quite large (almost 1 gigabyte) so it is stored separately and you should download a zipped version of it from
`http://www.chem.ucla.edu/dept/Faculty/dxn/pdf/wf.bin.gz`
and then write

```
gunzip wf.bin.gz
```

The calculation should be again done via mpi, e.g.,

```
mpirun -n 840 ./sgw_gpu.x >& log &
```

or, if one wants to use a lower number of cores and get the same exact numbers, use, e.g.,

```
mpirun -n 120 ./sgw_gpu.x >& log &
```

albeit the calculation will be then approximately `840/120=7` times slower.

At the end of the calculation, the `output` file shows a quasiparticle energy (real and imaginary parts) of

```
  840    -0.096131   0.001865    MC, E_qp   +-stat.err
```

so the real part of the GW quasi-particle energy is -0.096131 `a.u.=-2.62eV` with a statistical error of `0.001865 a.u.=0.05 eV`. This value is very close to the 2.69 eV experimental electron affinity of a single C_{60}.

When run on a modern supercomputer, the wall time of the calculation (read in the last few lines of the `log` file) was only 3337 sec=0.93 hr.

For verification, we also ran, beyond this first run, several runs with different parameters: an increased number of stochastic TDDFT states `ntddft=16`, a reduced energy width $\gamma = 0.04$ and different combination of the stochastic fragment parameters, `nxi=5000-20,000` and `segment_fraction=0.001, 0.003`. There was essentially no change in the results, and the real-part of the quasiparticle energy changed by only 0.03 eV, i.e., less than the statistical error.

Finally, we also ran the RPA simulations (by removing the `flg_dyn parameter`, which defaults to .false.), leading to a much deeper LUMO,

```
  840    -0.124622   0.001648    MC, E_qp   +-stat.err
```

i.e., -0.124622 `a.u.=-3.39eV`.

9.4 Graphene: orbital energies of a 2D material

In this example we will demonstrate how to use the `qe2sgw` utility to prepare orbitals from a preliminary Quantum ESPRESSO calculation. We will then perform a stochastic GW calculation on each orbital to evaluate the self-energy.

Running Quantum ESPRESSO:

First, navigate to ∼`/stochasticGW-GPU/examples/graphene` and follow the instructions in `README.md`. The first step is to generate the symbolic links to inputs, pseudopotentials, executables, and data files. Then go into the `1-scf` directory and note the following files:

- `README.md` contains run instructions.
- `pwmt-graphene.in` is the Quantum ESPRESSO input file for performing the SCF calculation.
- `qe2sgw.in` is the input for `qe2sgw.x`.

Run the SCF calculation as described in `README.md`. Next, we must convert the orbitals computed by Quantum ESPRESSO into the correct format. To do so, we must run the `pp.x` utility in Quantum ESPRESSO to generate the cube files for the orbitals and then process the cube files to obtain the `wf.txt` and `cnt.ini` input files required by *STOCHASTICGW* . The `qe2sgw.x` executable automates this process.

Running the `qe2sgw.x` Utility:

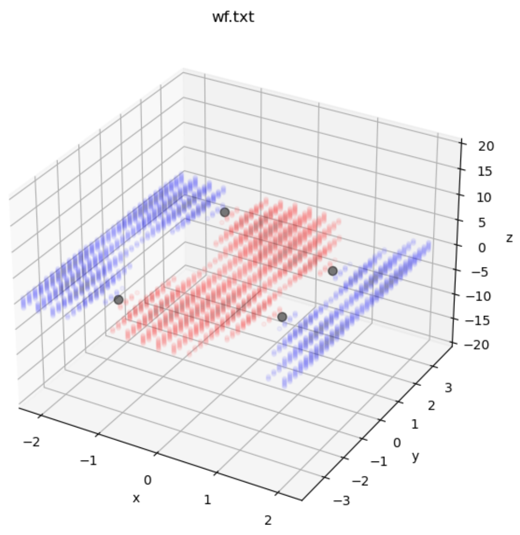
Examine `qe2sgw.in`, which reads:

```
output   graphene.out
prefix   graphene
qe_bin
wf_bin   F
output   full
nocc     8
orb
```

The 'output' setting, 'full', means that all SCF orbitals will be processed and packed into a single file, `wf.txt`. Alternatively, one can generate a binary file `wf.bin` by setting the 'wf_bin' parameter to 'T'. When using 'output' set to 'full', the 'orb' section must be left blank.

Now run `qe2sgw.x`. Once the orbitals are processed, one can use the tool `plotorbital.py` (note: requires `Python3` and `MatPlotLib`) to visualize an individual orbital; for instance, to plot orbital 5, run:

```
python plotorbital.py wf.txt 5 cnt.ini
```



Running STOCHASTICGW :

The next step is to perform the stochastic GW runs. Navigate to the directory `../2-sgw` and see the `README.md` file for instructions. In the `INPUT` file,

```
pps
C.pw-mt_fhi.UPF
#

scratch_path
.

ekcut
32

nmctot
1024

ntddft
-1

units_nuc
bohr

gamma
0.06

binary
F

orb_indx
1

scale_vh
1
```

```
box
sym

dim_periodic
2

projection
T

usegpu
T
```

note that `dim_periodic` is set to `2`, meaning that the cell is periodic in the x- and y-directions. Run the stochastic GW calculation to get the energy of the lowest orbital. Then modify `orb_indx` to values of `2, 3 ... 16` and repeat the calculation for each value.

Using the preceding `INPUT`, we obtain the following orbital energies and statistical errors (in hartrees):

```
01:  -2.614336  +/-  0.004916
02:  -2.250561  +/-  0.004365
03:  -2.046369  +/-  0.005259
04:  -1.733681  +/-  0.003061
05:  -1.994494  +/-  0.008459
06:  -1.557517  +/-  0.004542
07:  -1.477453  +/-  0.008087
08:  -1.460186  +/-  0.004760
09:  -0.007935  +/-  0.003433
10:  -0.093975  +/-  0.004511
11:  -0.091757  +/-  0.005321
12:  -0.091102  +/-  0.006031
13:  -0.062988  +/-  0.005641
14:  -0.010147  +/-  0.005150
15:   0.023845  +/-  0.005069
16:   0.256133  +/-  0.004303
```

9.5 Si-214: GPU-accelerated Gapped-Filtering calculation

In this section we outline the procedure for performing stochastic GW calculations on *large* systems; as an example, we predict the HOMO-LUMO gap for a 214-atom Silicon cluster containing a defect. Here, we use the `qe2sgw_stepwise` utility and the gapped filtering technique to reduce the storage and MPI communication burdens of handling large wavefunction files. As in the previous example we will use Quantum ESPRESSO to generate the initial wavefunction; then we wll use the tool `qe2sgw_stepwise.x` to convert the orbitals and density to the format needed by *stochasticGW* . In the final step, we will perform stochastic GW calculations on the HOMO and LUMO individually to obtain corrected energies.

Running Quantum ESPRESSO:

First, navigate to ∼`/stochasticGW-GPU/examples/Si214` and follow the instructions in `README.md`. Use the script provided to generate the symbolic links to inputs, pseudopotentials, executables, and data files. Then go into the `1-scf` directory and run the SCF calculation as described in `README.md`.

Running the `qe2sgw_stepwise.x` Utility:

The next step is to convert the orbitals and density from Quantum ESPRESSO into the format required by *stochasticGW* . For a large system with hundreds or more orbitals, it is more efficient to build the initial stochastic orbitals by filtering out the unoccupied states than it is to project them onto the set of occupied states. As a result, we do not need all of the orbitals from the SCF run; we need only those for which the self-energy is desired (in this case, the HOMO and LUMO). The `qe2sgw_stepwise.x` utility allows us to construct a file, `sgwinp.txt`, which contains the atomic coordinates, the density, and only the orbitals of interest needed for a stochastic GW calculation for which filtering is to be used to prepare the initial states.

This utility also uses `qe2sgw.in`, which, in this example, reads:

```
output   Si214_fhi_scf.out
prefix   Si214_fhi
qe_bin
wf_bin   F
output   unified
nocc     428
orb      428 429
```

The 'orb' field contains a list of orbitals for which the self-energy is desired, in this case only orbitals 428 and 429 (the HOMO and LUMO, respectively). The 'output' setting, 'unified', instructs the utility to generate `sgwinp.txt`.

Now run `qe2sgw_stepwise.x`. Unlike `qe2sgw.x`, which automates a single call to `pp.x` to process all orbitals, the stepwise version generates individual input files for `pp.x`, one for the density and one for each orbital listed in `qe2sgw.in`. The `qe2sgw_stepwise.x` routine attempts to generate `sgwinp.txt` from the output cube files of `pp.x`, and, if any of the cube files are not found, `qe2sgw_stepwise.x` will generate a list of input files for `pp.x` instead. Next, run these as:

```
pp.x -i pp_orb_428.in > pp_orb_428.out
pp.x -i pp_orb_429.in > pp_orb_429.out
pp.x -i pp_dens.in > pp_dens.out
```

After these have finished, running `qe2sgw_stepwise.x` a second time produces the desired `sgwinp.txt`.

Running *stochasticGW* :

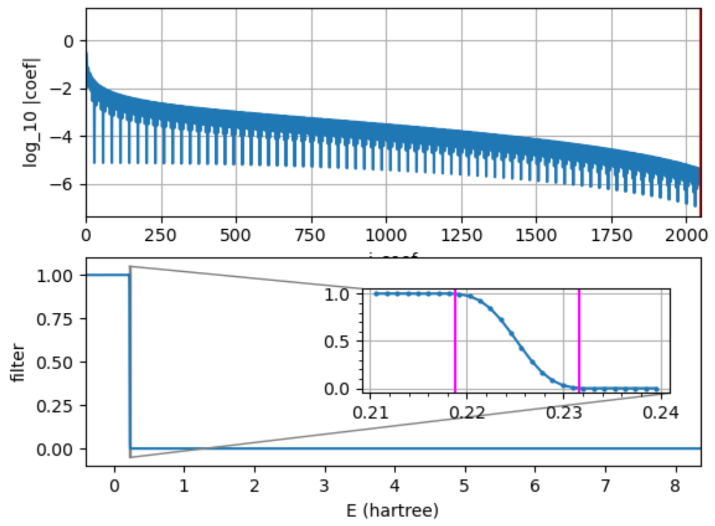
The next step is the *stochasticGW* run. Navigate to the directory `../2-sgw` and see the `README.md` file for instructions. Note the content of `INPUT`,

```
Si.pw-mt_fhi.UPF
#

scratch_path
.

ekcut
8

nmctot
1 #1024

ntddft
32

units_nuc
bohr

gamma
2.4 #0.06
```

```
binary
F

orb_indx
428

scale_vh
1

box
sym

dim_periodic
3

nchb
2048

usegpu
T
```

where filtering is chosen over projection by default and gapped filtering is chosen by default when the μ parameter is absent from `INPUT`. Note also that the values of the HOMO and LUMO energies are included in `sgwinp.txt` and are therefore not needed in `INPUT`.

One should run a preliminary calculation with a small value of `nmctot` and a large value of `gamma` (to reduce numerical effort) to ensure that there are enough filter coefficients to give a smooth filter (as controlled by the `nchb` variable). After running, one can examine the filter visually with the `plotfilter.py` utility (note: requires `Python3` and `MatPlotLib`). The top panel shows the logarithm of the magnitude of the filter coefficients as a function of Chebyshev polynomial order. The bottom panel shows the reconstruction of the filter with a blow-up of the gap region. The vertical lines indicate the HOMO and LUMO energies from the SCF calculation. Ideally one wants the filter is be = 1 at energies below the HOMO and = 0 at energies above the LUMO.



Once satisfied with the quality of the filter, one is ready to evaluate the self-energy of the HOMO and LUMO. This must be done as two separate runs. Change `nmctot` and `gamma` to values needed for the desired statistical error and energy resolution, respectively, and then run a stochastic GW calculation for the HOMO (`orb_indx = 428`) and one for the LUMO (`orb_indx = 429`).

We obtained the following values, giving a HOMO-LUMO gap of 0.0212 a.u. = 0.577 eV:

```
HOMO (orbital 428)
          1024       0.230027       0.001362          MC, E_qp           +-stat.err
LUMO (orbital 429)
          1024       0.251231       0.001376          MC, E_qp           +-stat.err
```

10 Acknowledgements

We are grateful for support by multiple sources.

This work was supported by the Center for Computational Study of Excited-State Phenomena in Energy Materials at the Lawrence Berkeley National Laboratory, which is funded by the U.S. Department of Energy, Office of Science, Basic Energy Sciences, Materials Sciences and Engineering Division under Contract No. DE-AC02-05CH11231, as part of the Computational Materials Sciences Program.

The work was further supported by the National Science Foundation, grants DMR-1611382 and CHE-1112500 of Daniel Neuhauser, and CHE-1465064 of Eran Rabani.

The Israel Science Foundation supported the work through grant 189/14 to Roi Baer and through a FIRST Program grant (1700/14) to Roi Baer and Eran Rabani. Roi Baer also acknowledges support by the Binational Science Foundation, Grant 2015687.

The code was tested and optimized within the XSEDE[8] computational project TG-CHE170058.

The stochasticGW code was mostly self-written, but it has several publicly available routines, primarily the KISS random number generator of George Marsaglia; we use, and slightly modified, the implementation of Jean-Michel Brankart. The stochasticGW package also includes several spline routines from Numerical Recipes.[9]

[8] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, et al. Xsede: accelerating scientific discovery. Comput. Sci. Eng. 16,62 (2014). https://aip.scitation.org/doi/abs/10.1109/MCSE.2014.80

[9] `http://www.nr.com/`