Quantum Espresso Developers Meeting

Henry Lambert & Feliciano Giustino
20/01/2016
Sternheimer-GW

- Program for treating electron correlation in extended systems using the GW approximation.
- Package currently hosted locally at Oxford and kept in line with the development branch of Quantum Espresso (QE).
- Sternheimer-GW package also mirrored on QE-Forge as private repository (soon to be public).
- The package is deeply reliant on QE routines and philosophy.
The Key Equations

DFT
\[ \left[ \hat{H}^0(r) + V^{xc}(r) \right] \psi_n(r) = \epsilon_n \psi_n(r) \]

GW
\[ \hat{H}^0(r) \psi_n(r) + \int \Sigma(r, r'; E_n) \psi_n(r') dr' = E_n \psi_n(r) \]

Essentially trying to calculate three quantities:

\[ \Sigma(i\omega_0) = \int_{-\infty}^{\infty} \frac{-1}{2\pi} G(i(\omega + \omega')) W(i\omega') d\omega' \]
(\hat{H} - \epsilon_n \pm \omega) \Delta \psi_{n[r\omega]}(r') = - (1 - \hat{P}_{occ}) \Delta V_{r\omega}(r') \psi_n(r')

\Delta n_{[r\omega]} = 2 \sum_n \psi_n(\Delta \psi^+_{n[r\omega]} + \Delta \psi^-_{n[r\omega]})

\Delta V^H(r, r'; \omega) = \int \Delta n_{[r\omega]}(r'') v(r'', r') dr''

W(r, r'; \omega) = v(r, r') + \Delta V^H(r, r'; \omega)
Both the Sternheimer equation:

$$(\hat{H} - \epsilon_n \pm \omega) \Delta \psi_{n[r\omega]}(\mathbf{r}') = -(1 - \hat{P}_{occ}) \Delta V_{[r\omega]}(\mathbf{r}') \psi_n(\mathbf{r}')$$

and the Green's function:

$$(\hat{H} - \omega - i\eta) G^A_{[\mathbf{r},\omega]} = -\delta_{[\mathbf{r}]}$$

are both linear systems of the form:

$$(A + \omega I)x = b$$

Allows full frequency dependence of operators to be calculated rapidly.

\textit{cbicg\_solve.f90, coul\_multi.f90, green\_multishift.f90}

Frommer, A. Computing \textbf{70} 87 (2003)
Flow Chart

Ground State Calculation:
PWSCF
\( \psi_k(r), n(r) \)

Calculate \( W(r, r'; \omega) \):
\[ [H - \epsilon_k \pm \omega] \Delta \psi_{k+q}^\pm = -\hat{P}_c \Delta V_q \psi_k \]
do_stern.f90

Calculate \( G_{k-q}(i(\omega + \omega')) \):
green_linsys_im.f90

Calculate \( \hat{\Sigma}_k(r, r'; \omega') \):
sigcim.f90
\[ \sum_q \int G_{k-q}(i(\omega + \omega')) W_q(i\omega') d\omega' \]

SGW Post-Processing:
selfenergy.py
mustar.f90
sgwvis.f90

\( \hat{\Sigma}_k(i\omega) \) Matrix Elements:
sigma_matel.f90
\[ E_{nk}^{QP} = \epsilon_{nk}^{LDA} + Z(\hat{\Sigma}(\epsilon_{nk}^{LDA}) - V_{nk}^{xc}) \]
Clear Duplication:
- orthogonalize.f90,
- cft_wave.f90,
- smallgq.f90,
- q_points.f90.

Slight Modification:
- ch_psi_all.f90,
- ch_psi_all_green.f90

Licensing provisions: GNU General Public License v3.0
No. of lines in distributed program, including test data, etc.: 14 507
No. of bytes in distributed program, including test data, etc.: 1 750 000
Programming Language: Fortran, Python
Computer: Linux/UNIX clusters
Operating system: Variety of Linux Environments, Opteron
RAM: Variable dependent on system size.
External routines: BLAS, LAPACK, FFTW, ScaLAPACK, MPI
Evolution of the FFT

- Original implementation had its own routines for generating custom grids.
  - G and W represented on a coarser grid.
- Latest version uses fft_custom.f90 and fft_structures.

Original Interface

```fortran
call cfft3d (aux, nr1sco, nr2sco, nr3sco, nr1sco, nr2sco, nr3sco, +1)
```

Most Recent Interface

```fortran
call invfft('Custom', aux, sigma_c_st%dffft)
```
FFT Routines

- The custom grid and fft structure allows a lot more flexibility and semantic compression.

```fortran
sigma_x_st%ecutt = ecutsex
sigma_x_st%gcutmt = ecutsex/tpiba2
Gcut = (SORT (sigma_x_st%ecutt) / tpiba + gcut)**2

! Generate auxiliary exchange grid.
  do ng = 1, ngm
    if ( gl( igtongl (ng) ) .le. sigma_x_st%gcutmt ) sigma_x_st%ngmt = ng
    if ( gl( igtongl (ng) ) .le. sigma_x_st%gcutmt ) sigma_x_st%ngmt_g = ng
    if ( gl( igtongl (ng) ) .le. sigma_x_st%gcutmt ) gexecut = ng
  enddo
  CALL set_custom_grid(sigma_x_st)
  CALL realspace_grid_init(sigma_x_st%dfftt, at, bg, sigma_x_st%gcutmt)
  CALL pstickset_custom( gamma_only, bg, sigma_x_st%gcutmt, gcut, sigma_x_st%gcutmt, &
                        dfftp, sigma_x_st%dfftt, ngw_, ngm_, ngs_, me, root, nproc, &
                        intra_comm, ngrp )
  CALL gvec_init(sigma_x_st, sigma_x_st%ngmt, intra_comm)
  sigma_x_st%initialized = .true.
  CALL ggent(sigma_x_st)
```
Evolution of the Linear System solvers?

- Conjugate gradients, bi-conjugate gradients, complex bi-conjugates, GMRES, bicgstab(l), complex bi-conjugate multishift.
  - A standard format accept generic derived types.
  - Some internal standard as well.
  - Essentials already in place just need to polish.

```call
CALL cbcg_solve(ch_psi_all, cg_psi, etc(1,ikk), dvpsi, dpsipm(:, :, 1), h_diag, npwx, npwq, thresh, ik, lter, conv_root, anorm, nbnd_occ(ikk), npol, cw, maxter_coul, .true.)
```

```call
CALL cbcg_solve(H_struct, psi_struct, convergence_struct)
```
Parallel Implementation

- Exploits current parallel structure of Quantum Espresso: Images, pools, and can be extended to utilize QE’s impressive G-vector parallelization.

\[
W_{[q,G]}(G')
\]

\[
\hat{\Sigma}_k(i\omega_0) = \sum_{\omega'} \sum_{q} G_{k-q}(i(\omega_0 \pm \omega')) W_q(i\omega')
\]
Post-processing tools

- The post-processing routines of espresso are also referenced.
- We will be interested in visualization and extracting Data from the W and self-energy matrices.
  - Coulomb matrix elements.
  - Visualization of spectral properties.
  - GW Eigenvalues.
What to do with W?
What to do with W?

coulmat.x

\[ \mu = N(0) \langle \langle W_{n,k,n',k'} \rangle \rangle_{FS} \]

\[ \mu^* = \frac{\mu}{1 + \mu \log \frac{\omega_n}{\omega_c}} \]

\[
\begin{array}{c|ccc}
 & \mu & \omega_f & \mu^* \\
\hline
\text{RPA} & 0.258 & 2.0 & 0.236 \\
 & 0.258 & 12.4 & 0.165 \\
\text{RPA+Vxc} & 0.289 & 2.0 & 0.254 \\
 & 0.289 & 12.4 & 0.173 \\
\end{array}
\]

<table>
<thead>
<tr>
<th>Material</th>
<th>N(0)</th>
<th>LDA Bandwidth</th>
<th>( \mu )</th>
<th>( \mu^* )</th>
<th>( \mu )</th>
<th>( \mu^* )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Al</td>
<td>0.186</td>
<td>11.42</td>
<td>0.236</td>
<td>0.100</td>
<td>0.215</td>
<td>0.96</td>
</tr>
<tr>
<td>Pb</td>
<td>0.246</td>
<td>11.74</td>
<td>0.244</td>
<td>0.100</td>
<td>0.191</td>
<td>0.91</td>
</tr>
<tr>
<td>MgB2</td>
<td>0.350</td>
<td>12.40</td>
<td>0.196</td>
<td>0.091</td>
<td>0.224</td>
<td>0.12</td>
</tr>
</tbody>
</table>

*FIG. 4: \( \varepsilon^{-1}(q, \omega = 0) \) for \( q_{||} \) to the boron sheets (blue line), \( q_{\perp} \) to the boron sheets (black line), and the Lindhard dielectric function (dashed yellow line).*
selfenergy.py

Quantum Espresso/SGW parsing/workflow/plotting scripts — Edit

Branch: master → pw-sgwpyp / +

- template: just combining everything into a python module.
  2 years ago
- README: update readme
  2 months ago
- __init__.py: just combining everything into a python module.
  2 years ago
- bands.py: added fermi_surf
  a month ago
- crystal_structure.py: moved graphviz to flake
  26 seconds ago
- extract.py: added extract file
  10 months ago
- fermi_surf.py: updated graphviz recursive patterning algorithm
  18 days ago
- flake.py: moved graphviz to flake
  26 seconds ago
Silicon Quasiparticle Properties

Flow Chart

Ground State Calculation:
PWSCF
\( \psi_k(r), n(r) \)

Calculate \( W(r, r'; \omega) \):
do_stern.f90
\[ [H - \epsilon_k \pm \omega] \Delta \psi_{k+q}^\pm = -\hat{P}_c \Delta V_q \psi_k \]

Calculate \( G_{k-q}(i(\omega + \omega')) \):
green_linsys_im.f90

Calculate \( \hat{\Sigma}_k(r, r'; \omega') \):
sigcim.f90
\[ \sum_q \int G_{k-q}(i(\omega + \omega')) W_q(i\omega') d\omega' \]

SGW Post-Processing:
selfenergy.py
mustar.f90
sgwvis.f90

\( \hat{\Sigma}_k(i\omega) \) Matrix Elements:
sigma_matel.f90
\[ E_{nk}^{\text{QP}} = \epsilon_{nk}^{\text{LDA}} + Z(\hat{\Sigma}(\epsilon_{nk}^{\text{LDA}}) - V_{nk}^{\text{xc}}) \]
Drawing from the QE Well

- Espresso Routines:
  - PWSCF.
  - FFT structures, igk/igkq arrays.
  - Image, pool, band group parallel environments.
  - The Hamiltonian!

- SGW developments:
  - Linear Algebra routines used and included in SGW distribution: Multishift, cBiCG, BICGSTAB(l).
    - Would benefit from some semantic compression.
    - Standardized template for Krylov methods.
  - Symmetrized routines for rotating the dielectric matrix and performing convolutions.
  - Efficient 2-pt FFT routines.
# Workflow and Visualization scripts

Montmorency / **pw-sgwpy**

Quantum Espresso/SGW parsing/workflow/plotting scripts — Edit

- **26** commits
- **1** branch
- **0** releases
- **1** contributor

Branch: **master** | **pw-sgwpy** / +

**Montmorency** moved graphoxII to flake

<table>
<thead>
<tr>
<th>File</th>
<th>Content</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>template</td>
<td>just combining everything into a python module.</td>
<td>2 years ago</td>
</tr>
<tr>
<td>README</td>
<td>update readme</td>
<td>2 months ago</td>
</tr>
<tr>
<td><strong>init</strong>.py</td>
<td>just combining everything into a python module.</td>
<td>2 years ago</td>
</tr>
<tr>
<td>bands.py</td>
<td>added fermi_surf</td>
<td>a month ago</td>
</tr>
<tr>
<td>crystal_structure.py</td>
<td>moved graphoxII to flake</td>
<td>26 seconds ago</td>
</tr>
<tr>
<td>extract.py</td>
<td>added extract file</td>
<td>10 months ago</td>
</tr>
<tr>
<td>fermi_surf.py</td>
<td>updated graphoxII recursive patterning algorithm</td>
<td>18 days ago</td>
</tr>
<tr>
<td>flake.py</td>
<td>moved graphoxII to flake</td>
<td>26 seconds ago</td>
</tr>
</tbody>
</table>
• Thanks to QE community.
• Thanks to Nourdine Zibouche, Martin Schlipf, Kun Cao.
• Thank you for your time.