Third order DFPT via “$2n + 1$”, ph-ph interaction and thermal transport

Lorenzo Paulatto

IMPMC - Université Pierre et Marie Curie, Paris 6/CNRS - Paris (France)

lorenzo.paulatto@impmc.upmc.fr

January 18, 2016
Introduction

This work is divided in two parts, one that overlap strongly with the DFPT phonon code, another that is related to Phonon post-processing tools (q2r, matdyn, qha, ...)

1. The $2n + 1$ theorem and the D3 code
   - What does it do?
   - How is it done?
   - What can be backported into phonon?
   - What would I like from upstream?

2. The ph-ph interaction and thermal transport codes
   - What do they do?
   - Should it go back to phonon?
   - What do they do?
The 2n + 1 theorem and the D3 code

Theorem (The 2n + 1 theorem)

The (2n + 1)th derivative of the eigenenergy of a Hamiltonian may be determined with only a knowledge of the change in the eigenfunctions up to order n.

Kohn-Sham formulation: Gonze and Vigneron 1989, Gonze 1995

What does it do?
The D3 code computes the third derivative of the total energy with respect to three harmonic perturbations using this theorem.
How is it done/1: workflow

Step 1
We do a normal self-consisted calculation with pw.x

Step 2
We do a phonon dispersion calculation on a regular symmetry-reduced grid with ph.x; the variation of charge density \( n^1(q) \) is rotated on the star of \( q \) and stored to files (a few 100MB)

Step 3
- We build a grid of \( q_1, q_2, q_3 = -q_1 - q_2 \) and reduce it with symmetry and permutations
- For every triplet, compute the \( D^3 \) matrix:
  \[
  \frac{\partial^3 E}{\partial u_i(q_1) \partial u_j(q_2) \partial u_k(q_3)}
  \]
Why do we store only $n^1(q)$ from ph.x?

- The perturbation of the wavefunctions $\psi^1(q)$ take orders of magnitude more disk space.
- We would still need to transform $k \rightarrow k + G$ (annoying in parallel).
- There is a subtle difference in the case of metals.

Why don’t you recompute $n^1(q)$/do this inside phonon?

- It is several orders of magnitude more complicated.
- It is probably slower.
For every triplet $q_1, q_2, q_3$:

- Restart from file, do an nscf calculation for every $k, k + q_x, k - q_x$, $x=1,2,3$

$\Rightarrow$ code detects special cases to reduce the grid ($q = \Gamma, q_x = q_y$)

$\Rightarrow$ $k + q = k' + G$ not implemented (too complex)

- Find the files containing $n^1(q_x) + G$, read it and eventually apply a phase

- Solve 12 times the non-self consistently the Sternheimer equation:

$$ (H_{scf} + \alpha P_v - \epsilon_n) \Delta^q \psi_n(k) = P_c \Delta^q V \psi_n(k) $$

$\Rightarrow$ $\Delta^q \psi_n(k - q), \Delta^{-q} \psi_n(k + q), \Delta^q \psi_n(k), \Delta^{-q} \psi_n(k)$
For every triplet $q_1$, $q_2$, $q_3$:

$$\Rightarrow \Delta^q\psi_n(k - q), \Delta^{-q}\psi_n(k + q), \Delta^q\psi_n(k), \Delta^{-q}\psi_n(k)$$

- $\langle \Delta^{-q_1}\psi_{k+q_1} | \Delta^{-q_2} V | \psi_{k+q_2} \rangle$
- $\langle \Delta^{-q_3}\psi_k | \Delta^q_2 V | \delta^q_1\psi_k \rangle$
- $\langle \delta^{-q_3}\psi_{k+q_3} | \delta^q_1\psi_{k-q_1} \rangle \langle \psi_{k-q_1} | \Delta^q_2 V | \psi_{k+q_3} \rangle$
- $\langle \Delta^{-q_1}\psi_k | \Delta^q_2, q_3 V | \psi_k \rangle$
- $\langle \psi_k | \Delta^q_1, q_2, q_3 V | \psi_k \rangle$

Ewald (Ion-Ion) term

- terms depending on the Fermi energy
- derivative of the functional $\delta E_{xc} / \delta n^1(q_1) \delta n^1(q_2) \delta n^1(q_3)$
What do we use from \texttt{ph.x}

### What we use

- \texttt{cg\_solve\_all} is used as-is, with custom (aka ancient) \texttt{ch\_psi\_all} and \texttt{cg\_psi}; it uses the same \texttt{h\_psi} as phonon
- Custom version of \texttt{solve\_linter} (non-scf, partial list of \texttt{k}-points)
- Simplified \texttt{dq\_vscf}

### Can we do better?

- For some subroutines it is just a matter of testing: \texttt{cg\_psi}
- For some a bit more work is required \texttt{dv\_of\_drho} (but I do not want to make the original subroutine a mess)
- For others the additional complexity outside the subroutine would hinder the simplification (es. \texttt{ch\_psi\_all})
Why we do not use more subroutines from phonon.

```
example: cg_psi

IF(tddfpt) THEN
  ikq = ik
  evq => evc
ELSE
  ikq = ikqs(ik)
ENDIF
```

- We would love to use `dv_of_vdrho` & company but they got way too complicated: too many global variables, used very deeply (i.e. $q$-point, $n^1$, $evc/evq$) makes the outcome practically unpredictable.
- One especially offending variable: `ikqs`

Is there a tradeoff between having two similar subroutines or one subroutine that nobody can read?

No: but you will have to use subroutine arguments, instead of global variables. This requires writing a few more lines.
How to fix it

Why is it difficult

I can easily generalize the subroutines in ph.x and my code, but will it break everyone else code? Will everybody hate me?

A possible workaround

subroutine old_style()
    use module, only : variable
    call new_style(variable)
end subroutine

subroutine new_style(variable)
    kind :: variable
    ! old body is actually here
end subroutine

I would rather get rid of old_style immediately
What can be backported into phonon?

Already backported:

- The subroutines to rotate $n^1$ (or $V^1$) and write it to a $q$-defined file

To do:

- Use a different scratch directory per $q$-point, no name conflicts
- Open all $pw.x$ files read-only to prevent accidental corruption of scf data
- Keep track of all opened files
- $\Rightarrow$ Difference instances of the code can work on the same prefix/outdir as long as they have difference $q$-points
What would I like from upstream?

- The possibility to do a read-only restart
- Read a wafunction by \( k \)-point, instead of \( k \)-point number (analogous to what I do in \( d3 \) for \( q \)-points)
- more “bottom-up” and self-contained basic subroutines, especially
- symmetry
- derived types that contain big chunks of the global variables (i.e. lattice+basis+symmetry, \( k \)-points and \( g \) vectors,...)
Phonon-Phonon interaction

What do they do

Integrate the ph-ph interaction equation:

\[
\gamma_{qj}(T) = \frac{1}{\tau_{qj}(T)} = \frac{\pi}{\hbar^2 N_q} \sum_{q', j', j''} \left| V_{qj, q'j', q''j''}^{(3)} \right|^2
\]
\[
\times \left[ (1 + n_{q'j'} + n_{q''j''}) \delta(\omega_{qj} - \omega_{q'j'} - \omega_{q''j''}) + 2(n_{q'j'} - n_{q''j''}) \delta(\omega_{qj} + \omega_{q'j'} - \omega_{q''j''}) \right]
\]

How is it done

\(D^3\) and \(D^2\) matrices need to be Fourier-interpolated on a fine double-grid of \(q\)-points.
Thermal transport (SMA)

\[ \kappa^\alpha,\beta_L = \frac{\hbar^2}{Nq\Omega K_B T^2} \sum_{q\nu} c^{\alpha}_{q\nu} c^{\beta}_{q\nu} \omega^2_{q\nu} n_{q\nu}(n_{q\nu} + 1) \tau_{q\nu} \]

- We have two nested sums over double q-point grid
- We have libraries to quickly and reliably compute phonon frequencies, group velocities, occupations...
- Everything has to be parallelised with MPI
What do they do?

- A set of small specialized codes and libraries that:
  1. Take care of complicate input/output, initial setup, sum rules, etc.
  2. Perform one and only one core function (Fourier interpolate, sum integral, BE occupation, ...)

- In total 13k lines, around 3.9k are rewrites of q2r and matdyn
- q2r+matdyn is 3.4k lines
  ⇒ but the “matdyn” equivalent code is only about 100 lines

Effective charges are still not implemented.
Should I put them back in phonon?

**Pros**

- More people may use them
- Some people may find bugs and complain
- Some bug may even get fixed
- Profit

**Pros**

- I may not like some fixes
  ⇒ Will my code rot?
  ⇒ Will it rot faster if I keep a tight grip on it?
The End