



Université catholique de Louvain
Ecole Polytechnique de Louvain

:

Consistency check for Diamond

Samuel Poncé

Xavier Gonze
Paul Boulanger
Gabriel Antonius
Michel Côté

November 8, 2012

Contents

1	Introduction	3
1.1	Janak theorem	3
1.2	Temperature dependence	3
2	Test of the ZPM renormalisation at Γ by 3 different methods	5
2.1	Change of the occupation number using ABINIT for Diamond for the HOMO band	6
2.2	Finite difference using ABINIT for Diamond for the HOMO band	6
2.3	AHC and NDDW by finite difference	8
3	Test of the three method at different q points	9
3.1	Finite difference on the eigenenergies	9
3.1.1	Treatment of some possible problems	10
3.1.2	Coming back to the initial problem...	11
3.1.3	Example	12
3.1.4	Coming back again to the initial problem	13
3.2	Finite difference on the occupation	13
3.3	AHC + NDDW	13
3.3.1	Theory	13
3.3.2	For the L point	14
3.3.3	Other method	18
3.3.4	AHC	18
3.3.5	Summary	18
3.3.6	Richardson extrapolation	18
4	Point $\frac{2L}{3}$	23
4.1	AHC	23
4.2	Finite difference on eigenenergies	23
4.3	NDDW	24
4.4	Summary	24
5	Annex	26
5.1	ABINIT input file	26
5.2	Dynamical matrix	28
5.3	Python script for solving the eigenvalue problem.	29
5.4	NDDW using finite difference at the q=L point	30
5.5	NDDW using finite difference at the q=L point	32
5.6	NDDW using finite difference at the q=L point	36
5.7	NDDW using finite difference at the q=L point	38
5.8	AHC at the q=L point	41
5.9	Python scripts I	42
5.10	Python scripts II	45
5.11	Finite diff. at the q=2/3L	49
5.12	Finite diff. at the q=2/3L	57
5.13	NDDW by Finite diff. at the q=2/3L	63
5.14	NDDW by Finite diff. at the q=2/3L	67
6	Side notes [Just for me...]	70

1 Introduction

The total energy can be written as:

$$E[\{f_i\}, \mathbf{R}_\tau] = \min_{\substack{\varphi_i \\ \langle \varphi_i | \varphi_j \rangle = \delta_{ij}}} \left[\sum_i f_i \langle \varphi_i | T | \varphi_i \rangle + \int d\mathbf{r} n(\mathbf{r}) \sum_\tau v_\tau(\mathbf{r} - \mathbf{R}_\tau) + E_{Hxc}[n(\mathbf{r})] \right] \quad (1)$$

$$\text{with } n(\mathbf{r}) = \sum_i f_i \varphi_i^*(\mathbf{r}) \varphi_i(\mathbf{r}) \quad (2)$$

where f_i is the occupation number and τ and index number identifying an atom.

The energy of an electron in state i is obtained as the eigenstate of the Hamiltonian:

$$\varepsilon_i = \langle \varphi_i | H | \varphi_i \rangle \quad (3)$$

$$H = T + \sum_\tau v_\tau v_\tau(\mathbf{r} - \mathbf{R}_\tau) + v_{Hxc}(\mathbf{r}) \quad (4)$$

$$v_{Hxc}(\mathbf{r}) = \frac{\partial E}{\partial n(\mathbf{r})} \quad (5)$$

1.1 Janak theorem

$$\varepsilon_i = \frac{\partial E}{\partial f_i} = \langle \varphi_i | T | \varphi_i \rangle + \int d\mathbf{r} \varphi_i^*(\mathbf{r}) \varphi_i(\mathbf{r}) \sum_\tau v_\tau(\mathbf{r} - \mathbf{R}_\tau) + v_{Hxc}(\mathbf{r}) \quad (6)$$

$$\frac{dE}{df_i} = \frac{\partial E}{\partial f_i} + \int d(\mathbf{r}) \sum_j \frac{\cancel{\partial E}}{\cancel{\partial \varphi_j(\mathbf{r})}} \frac{\partial \varphi_j}{\partial f_i} + \int d(\mathbf{r}) \frac{\cancel{\partial E}}{\cancel{\partial n(\mathbf{r})}} \frac{\partial n(\mathbf{r})}{\partial f_i} \quad (7)$$

where the simplification is obtained by using the fact that we are at an energetic minimum. The partial derivative of the energy with respect to the wave function is then zero.

1.2 Temperature dependence

The evolution of the electronic level with temperature are obtained by applying the following correction to the ground state electronic levels:

$$\Delta \varepsilon_i = \frac{\partial^2 \varepsilon_i}{\partial \mathbf{R}_\tau \partial \mathbf{R}_{\tau'}} \quad (8)$$

There is different ways to calculate this second derivative of the eigenstates.

Finite difference on \mathbf{R} : This technique work for small system (e.g. the H_2 molecules) but is non optimal for other systems.

Finite difference on f_i :

$$\frac{\partial^2 \varepsilon_i}{\partial \mathbf{R}_\tau \partial \mathbf{R}_{\tau'}} \stackrel{\text{Janak}}{=} \frac{\partial^3 E}{\partial f_i \partial \mathbf{R}_\tau \partial \mathbf{R}_{\tau'}} = \lim_{\Delta f_i \rightarrow 0} \frac{1}{\Delta f_i} \left(\left. \frac{\partial^2 E}{\partial \mathbf{R}_\tau \partial \mathbf{R}_{\tau'}} \right|_{f_i + \Delta f_i} - \left. \frac{\partial^2 E}{\partial \mathbf{R}_\tau \partial \mathbf{R}_{\tau'}} \right|_{f_i} \right) \quad (9)$$

$$= \lim_{\Delta f_i \rightarrow 0} \frac{1}{\Delta f_i} \left(\mathfrak{D}_{\tau\tau'}|_{f_i + \Delta f_i} - \mathfrak{D}_{\tau\tau'}|_{f_i} \right) \quad (10)$$

The Allen-Heine-Cardona method :

$$\begin{aligned} \frac{\partial^2 \varepsilon_i}{\partial \mathbf{R}_\tau \partial \mathbf{R}_{\tau'}} = & \left\langle \varphi_i^{(0)} \left| \frac{\partial^2 H}{\partial \mathbf{R}_\tau \partial \mathbf{R}_{\tau'}} \right| \varphi_i^{(0)} \right\rangle + \left\langle \frac{\partial \varphi_i}{\partial \mathbf{R}_\tau} \left| \frac{\partial H}{\partial \mathbf{R}_{\tau'}} \right| \varphi_i^{(0)} \right\rangle + \left\langle \frac{\partial \varphi_i}{\partial \mathbf{R}_{\tau'}} \left| \frac{\partial H}{\partial \mathbf{R}_\tau} \right| \varphi_i^{(0)} \right\rangle \\ & + \left\langle \varphi_i^{(0)} \left| \frac{\partial H}{\partial \mathbf{R}_\tau} \right| \frac{\partial \varphi_i}{\partial \mathbf{R}_{\tau'}} \right\rangle + \left\langle \varphi_i^{(0)} \left| \frac{\partial H}{\partial \mathbf{R}_{\tau'}} \right| \frac{\partial \varphi_i}{\partial \mathbf{R}_\tau} \right\rangle + \left\langle \frac{\partial \varphi_i}{\partial \mathbf{R}_\tau} \left| H^{(0)} \right| \frac{\partial \varphi_i}{\partial \mathbf{R}_{\tau'}} \right\rangle \end{aligned} \quad (11)$$

The first term is called the Debye-Waller (DW) term and the four next the FAN's terms. The DW term can be rewritten in term of a diagonal and non-diagonal part:

$$\delta_{\tau\tau'} \left\langle \varphi_i^{(0)} \left| \frac{\partial^2 H}{\partial \mathbf{R}_\tau \partial \mathbf{R}_{\tau'}} \right| \varphi_i^{(0)} \right\rangle + (1 - \delta_{\tau\tau'}) \left\langle \varphi_i^{(0)} \left| \frac{\partial^2 V_{Hxc}}{\partial \mathbf{R}_\tau \partial \mathbf{R}_{\tau'}} \right| \varphi_i^{(0)} \right\rangle \quad (12)$$

The non-diagonal DW term can be also written in integral form as:

$$\left\langle \varphi_i^{(0)} \left| \frac{\partial^2 V_{Hxc}}{\partial \mathbf{R}_\tau \partial \mathbf{R}_{\tau'}} \right| \varphi_i^{(0)} \right\rangle = \int d\mathbf{r} \frac{\partial^2 V_{Hxc}(\mathbf{r})}{\partial \mathbf{R}_\tau \partial \mathbf{R}_{\tau'}} n_i^{(0)}(\mathbf{r}) \quad (13)$$

We then see that eigenenergies are obtained by the partial density of the considered state i only. This is due to the fact that V_{Hxc} is a local potential in DFT. To treat the non-diagonal DW part we can add a frozen density that has the value of the density of the consider state and consequently the change in phonon eigenvalues will give the NDDW term.

$$\begin{aligned} E[\lambda, \mathbf{R}_\tau] = & \min_{\substack{\varphi_i \\ \langle \varphi_i | \varphi_j \rangle = \delta_{ij}}} \left[\sum_i f_i \langle \varphi_i | T | \varphi_i \rangle + \int d\mathbf{r} (n(\mathbf{r}) + \lambda \Delta n(\mathbf{r})) \sum_\tau v_\tau(\mathbf{r} - \mathbf{R}_\tau) \right. \\ & \left. + E_{Hxc}[n(\mathbf{r}) + \lambda \Delta n(\mathbf{r})] \right] \end{aligned} \quad (14)$$

We can also derive using (5):

$$\frac{\partial E}{\partial \lambda} = \int d\mathbf{r} \frac{\partial E}{\partial n(\mathbf{r})} \frac{\partial n(\mathbf{r})}{\partial \lambda} = \int d\mathbf{r} v_{Hxc}(\mathbf{r}) \Delta n(\mathbf{r}) \quad (15)$$

Using back eq (13) we have:

$$E_{\text{NDDW}}^{(2)} = \int d\mathbf{r} \frac{\partial^2 V_{Hxc}(\mathbf{r})}{\partial \mathbf{R}_\tau \partial \mathbf{R}_{\tau'}} n_i^{(0)}(\mathbf{r}) = \frac{\partial^2}{\partial \mathbf{R}_\tau \partial \mathbf{R}_{\tau'}} \int d\mathbf{r} V_{Hxc}(\mathbf{r}) n_i^{(0)}(\mathbf{r}) \quad (16)$$

$$\text{using } n_i^{(0)}(\mathbf{r}) \stackrel{=}{=} \Delta n(\mathbf{r}) \quad \frac{\partial^2}{\partial \mathbf{R}_\tau \partial \mathbf{R}_{\tau'}} \left(\frac{\partial E}{\partial \lambda} \right) = \frac{\partial}{\partial \lambda} \left(\frac{\partial^2 E}{\partial \mathbf{R}_\tau \partial \mathbf{R}_{\tau'}} \right) = \frac{\partial}{\partial \lambda} (\mathfrak{D}_{\tau\tau'}) \quad (17)$$

Adding a frozen density to the system make the eigenvalues of the phonon change and this give the NDDW term.

The last term can be calculated using finite difference:

$$\frac{\partial}{\partial \lambda} (\mathfrak{D}_{\tau\tau'}) = \lim_{\lambda \rightarrow 0} \frac{1}{\lambda} (\mathfrak{D}_{\tau\tau'}[n(\mathbf{r}) + \lambda n_i(\mathbf{r})] - \mathfrak{D}_{\tau\tau'}[n(\mathbf{r})]) \quad (18)$$

$$= \lim_{\lambda \rightarrow 0} \frac{1}{2\lambda} (\mathfrak{D}_{\tau\tau'}[n(\mathbf{r}) + \lambda n_i(\mathbf{r})] - \mathfrak{D}_{\tau\tau'}[n(\mathbf{r}) - \lambda n_i(\mathbf{r})]) + \mathcal{O}(\lambda^2) \quad (19)$$

We obtain the variation of the eigen energies from the total energy (Janak th.) thanks to the partial density.

2 Test of the ZPM renormalisation at Γ by 3 different methods

The interatomic force constant is defined as follow:

$$C_{\kappa\alpha, \kappa'\alpha'} = \frac{\partial^2 E}{\partial R_{\kappa\alpha} \partial R_{\kappa'\alpha'}} \quad (20)$$

The dynamical equation for the 6-eigendisplacements vectors ${}^6U(\kappa\alpha) = (\alpha_1 \alpha_2 \alpha_3 0 0 0)$ and ${}^6U(\kappa'\alpha') = (0 0 0 \alpha'_1 \alpha'_2 \alpha'_3)$:

$$\sum_{\kappa'\alpha'} C_{\kappa\alpha, \kappa'\alpha'} {}^6U_m(\kappa'\alpha') = M_\kappa \omega_m^2 {}^6U_m(\kappa\alpha) \quad (21)$$

Due to the completeness relation $\sum_m {}^6U_m(\kappa\alpha) {}^6U_m^*(\kappa'\alpha') \sqrt{M_\kappa M_{\kappa'}} = \delta_{\kappa\kappa'} \delta_{\alpha\alpha'}$ we have:

$$\omega_m^2 = \sum_{\kappa\alpha} \sum_{\kappa'\alpha'} {}^6U_m^*(\kappa'\alpha') C_{\kappa\alpha, \kappa'\alpha'} {}^6U_m(\kappa'\alpha') \quad (22)$$

The normalized mass-scaled displacements is ${}^6\xi_m(\kappa\alpha) = \sqrt{M_\kappa} {}^6U_m(\kappa\alpha)$ and the associated dynamical matrix $\sum_{\kappa\kappa'} D_{\kappa\alpha, \kappa'\alpha'} = \frac{1}{\sqrt{M_\kappa}} C_{\kappa\alpha, \kappa'\alpha'} \frac{1}{\sqrt{M_{\kappa'}}}$

We then have:

$$\sum_{\kappa'\alpha'} D_{\kappa\alpha, \kappa'\alpha'} {}^6\xi_m(\kappa'\alpha') = \omega_m^2 {}^6\xi_m(\kappa\alpha) \quad (23)$$

The total energy of an harmonic oscillator (including temperature dependence) is given by:

$$E(T) = E_{\text{without el-ph}} + \sum_m \omega_m (< n_m > (T) + 1/2) \quad (24)$$

with $< n_m > = \frac{1}{e^{\frac{E_m - \mu}{k_B T}} - 1}$ the Bose-Einstein distribution for phonon.

Using the Janak/Brooks theorem we have:

$$\epsilon_n(T) = \frac{\partial E(T)}{\partial f_n} = \epsilon_{\text{without el-ph}} + \sum_m \frac{\partial \omega_m}{\partial f_n} (< n_m > (T) + 1/2) \quad (25)$$

with f_n the occupation factor.

Using equation (22) we have:

$$2\omega_m \frac{\partial \omega_m}{\partial f_n} = \sum_{\kappa\alpha} \sum_{\kappa'\alpha'} {}^6U_m^*(\kappa\alpha) \frac{\partial C_{\kappa\alpha, \kappa'\alpha'}}{\partial f_n} {}^6U_m(\kappa'\alpha') \quad (26)$$

We then have:

$$\epsilon_n(T) - \epsilon_{\text{without el-ph}} = \sum_m \frac{1}{2\omega_m} \sum_{\kappa, \alpha} \sum_{\kappa', \alpha'} {}^6U_m^*(\kappa\alpha) \frac{\partial^2 \epsilon_n}{\partial R_{\kappa, \alpha} \partial R_{\kappa', \alpha'}} {}^6U_m(\kappa'\alpha') \left[< n_m > (T) + \frac{1}{2} \right] \quad (27)$$

$$= \sum_m \frac{1}{2\omega_m} \sum_{\kappa, \alpha} \sum_{\kappa', \alpha'} \frac{1}{\sqrt{M_\kappa}} {}^6\xi_m^*(\kappa\alpha) \frac{\partial^2 \epsilon_n}{\partial R_{\kappa, \alpha} \partial R_{\kappa', \alpha'}} \frac{1}{\sqrt{M_{\kappa'}}} {}^6\xi_m(\kappa'\alpha') \left[< n_m > (T) + \frac{1}{2} \right] \quad (28)$$

2.1 Change of the occupation number using ABINIT for Diamond for the HOMO band

The charge is calculated by the change in occupation number weighted by the number of k points.

$$\text{charge} = 8e^-/\text{unitcell} - \sum_k^{kmax} \text{occ}/\text{nbkpt} = 8 - \frac{31 * 8 + 7.97}{32} = 9.375 * 10^{-4} \quad (29)$$

charge	$f_{\text{Homo}}(\text{x3})$	Etot[Ha]	$\omega[\text{Ha}]$		
0	2	-12.069553133	0.0062041997898	0.006204199791	0.0062041997918
0.0009375	1.99	-12.069993427	0.0062035533518	0.006203553353	0.0062035533539
0.001875	1.98	-12.070433701	0.0062029068329	0.0062029068341	0.0062029068349

The first order derivative by decentered finite difference of second order is:

$$\frac{\partial E_{tot}}{\partial f_n} \approx \frac{3E_0 - 4E_{-h} + E_{-2h}}{2h} = 0.46965760000054 \quad (30)$$

At 0K the second term is obtained by the same finite difference taking the average over the three optical frequencies :

$$\frac{\partial \omega_m}{\partial f_n} \approx \frac{3\omega_{m0} - 4\omega_{-mh} + \omega_{-2mh}}{2h} \quad (31)$$

Using (25) we have at 0K:

$$\epsilon_n(T) - \epsilon_{\text{without el-ph}} = \sum_m \frac{\partial \omega_m}{\partial f_n} \frac{1}{2} = 28.142991182723 \text{meV} \quad (32)$$

At Γ the acoustic modes are identically 0 and we only have the contribution of the 3 optical modes.

2.2 Finite difference using ABINIT for Diamond for the HOMO band

The centred second order second derivative of a function of 2 variables can be obtained by a Taylor expansion of the following different terms:

$$f(x_1 + h, x_2) = f(x_1, x_2) + h \frac{\partial f}{\partial x_1} + \frac{1}{2} \left[h^2 \frac{\partial^2 f}{\partial x_1^2} \right] + \frac{1}{6} \left[h^3 \frac{\partial^3 f}{\partial x_1^3} \right] + \dots \quad (33)$$

$$f(x_1 - h, x_2) = f(x_1, x_2) - h \frac{\partial f}{\partial x_1} + \frac{1}{2} \left[h^2 \frac{\partial^2 f}{\partial x_1^2} \right] - \frac{1}{6} \left[h^3 \frac{\partial^3 f}{\partial x_1^3} \right] + \dots \quad (34)$$

$$f(x_1, x_2 + w) = f(x_1, x_2) + w \frac{\partial f}{\partial x_2} + \frac{1}{2} \left[w^2 \frac{\partial^2 f}{\partial x_2^2} \right] + \frac{1}{6} \left[w^3 \frac{\partial^3 f}{\partial x_2^3} \right] + \dots \quad (35)$$

$$f(x_1, x_2 - w) = f(x_1, x_2) - w \frac{\partial f}{\partial x_2} + \frac{1}{2} \left[w^2 \frac{\partial^2 f}{\partial x_2^2} \right] - \frac{1}{6} \left[w^3 \frac{\partial^3 f}{\partial x_2^3} \right] + \dots \quad (36)$$

$$\begin{aligned} f(x_1 + h, x_2 + w) &= f(x_1, x_2) + h \frac{\partial f}{\partial x_1} + w \frac{\partial f}{\partial x_2} + \frac{1}{2} \left[h^2 \frac{\partial^2 f}{\partial x_1^2} + 2hw \frac{\partial^2 f}{\partial x_1 \partial x_2} + w^2 \frac{\partial^2 f}{\partial x_2^2} \right] \\ &\quad + \frac{1}{6} \left[h^3 \frac{\partial^3 f}{\partial x_1^3} + 3h^2w \frac{\partial^3}{\partial x_1^2 \partial x_2} + 3hw^2 \frac{\partial^3}{\partial x_1 \partial x_2^2} + w^3 \frac{\partial^3 f}{\partial x_2^3} \right] + \dots \end{aligned} \quad (37)$$

$$\begin{aligned} f(x_1 - h, x_2 - w) &= f(x_1, x_2) - h \frac{\partial f}{\partial x_1} - w \frac{\partial f}{\partial x_2} + \frac{1}{2} \left[h^2 \frac{\partial^2 f}{\partial x_1^2} + 2hw \frac{\partial^2 f}{\partial x_1 \partial x_2} + w^2 \frac{\partial^2 f}{\partial x_2^2} \right] \\ &\quad - \frac{1}{6} \left[h^3 \frac{\partial^3 f}{\partial x_1^3} + 3h^2w \frac{\partial^3}{\partial x_1^2 \partial x_2} + 3hw^2 \frac{\partial^3}{\partial x_1 \partial x_2^2} + w^3 \frac{\partial^3 f}{\partial x_2^3} \right] + \dots \end{aligned} \quad (38)$$

$$\begin{aligned} f(x_1 + h, x_2 - w) &= f(x_1, x_2) + h \frac{\partial f}{\partial x_1} - w \frac{\partial f}{\partial x_2} + \frac{1}{2} \left[h^2 \frac{\partial^2 f}{\partial x_1^2} - 2hw \frac{\partial^2 f}{\partial x_1 \partial x_2} + w^2 \frac{\partial^2 f}{\partial x_2^2} \right] \\ &\quad + \frac{1}{6} \left[h^3 \frac{\partial^3 f}{\partial x_1^3} - 3h^2w \frac{\partial^3}{\partial x_1^2 \partial x_2} + 3hw^2 \frac{\partial^3}{\partial x_1 \partial x_2^2} - w^3 \frac{\partial^3 f}{\partial x_2^3} \right] + \dots \end{aligned} \quad (39)$$

$$\begin{aligned} f(x_1 - h, x_2 + w) &= f(x_1, x_2) - h \frac{\partial f}{\partial x_1} + w \frac{\partial f}{\partial x_2} + \frac{1}{2} \left[h^2 \frac{\partial^2 f}{\partial x_1^2} - 2hw \frac{\partial^2 f}{\partial x_1 \partial x_2} + w^2 \frac{\partial^2 f}{\partial x_2^2} \right] \\ &\quad + \frac{1}{6} \left[-h^3 \frac{\partial^3 f}{\partial x_1^3} + 3h^2w \frac{\partial^3}{\partial x_1^2 \partial x_2} - 3hw^2 \frac{\partial^3}{\partial x_1 \partial x_2^2} + w^3 \frac{\partial^3 f}{\partial x_2^3} \right] + \dots \end{aligned} \quad (40)$$

with $f = f(x_1, x_2)$

We then have the following relationship:

$$\frac{\partial^2 f(x_1, x_2)}{\partial x_1^2} = \frac{f(x_1 + h, x_2) + f(x_1 - h, x_2) - 2f(x_1, x_2)}{h^2} + \mathcal{O}(h^2) \quad (41)$$

$$\frac{\partial^2 f(x_1, x_2)}{\partial x_2^2} = \frac{f(x_1, x_2 + w) + f(x_1, x_2 - w) - 2f(x_1, x_2)}{w^2} + \mathcal{O}(w^2) \quad (42)$$

$$\frac{\partial^2 f(x_1, x_2)}{\partial x_1 \partial x_2} = \frac{f(x_1 + h, x_2 + w) + f(x_1 - h, x_2 - w) - f(x_1 + h, x_2 - w) - f(x_1 - h, x_2 + w)}{4hw} + \mathcal{O}(hw) \quad (43)$$

Here we choose to have the same displacement for all atoms ($h = w$). We could also have pick some $h = w$ displacement for equation (41) and (42) and $h/2 = w/2$ for (43) in order to keep the same total displacement.

The only thing left to do is to link equations (41), (42) and (43) to equation (28).

As we have impose that eigenvector ${}^6\xi = \begin{bmatrix} {}^3\xi_1 \\ {}^3\xi_2 \end{bmatrix}$ has to be normalized we have $|{}^6\xi| = 1$ and this mean $|{}^3\xi_1| = |{}^3\xi_2| = \frac{1}{\sqrt{2}}$

In our case we have:

We can make the following parallelism by Taylor expanding ε_n :

$$f(x_1 + h, x_2) = \varepsilon_n({}^6R_0 + h\sqrt{2} \begin{bmatrix} {}^3\xi_1 \\ 0 \end{bmatrix}) = \varepsilon_n^0 + \frac{1}{2} \sum_{\alpha\alpha'} 2(h^2){}^3\xi_1(\alpha) \frac{\partial^2 \varepsilon_n}{\partial R_{1\alpha} \partial R_{1\alpha'}} {}^3\xi_1(\alpha') \quad (44)$$

charge	Etot[Ha]	ϵ_{HOMO} [Ha]		
$f(x_1, x_2)$	-12.069553133	0.4696587868	0.4696587868	0.4696587873
$f(x_1+h, x_2)$	-12.069489243	0.4636328481	0.4726927819	0.4726927819
$f(x_1-h, x_2)$	-12.069490604	0.4666565363	0.4666565363	0.4757055329
$f(x_1, x_2+w)$	-12.069490604	0.4666565364	0.4666565364	0.475705533
$f(x_1, x_2-w)$	-12.069489243	0.4636328481	0.4726927819	0.4726927819
$f(x_1+h, x_2+w)$	-12.069553133	0.469658787	0.469658787	0.4696587874
$f(x_1-h, x_2-w)$	-12.069553133	0.469658787	0.469658787	0.4696587874
$f(x_1+h, x_2-w)$	-12.069294833	0.4576285145	0.475757834	0.475757834
$f(x_1-h, x_2+w)$	-12.069305726	0.463686737	0.463686737	0.4817722542

And similarly for the other. We thus obtain:

$$\begin{aligned} \frac{\partial^2 f(x_1, x_2)}{\partial x_1^2} &= \frac{f(x_1 + h, x_2) + f(x_1 - h, x_2) - 2f(x_1, x_2)}{h^2} \\ &= 2 \sum_{\alpha\alpha'} {}^3\xi_1(\alpha) \frac{\partial^2 \epsilon_n}{\partial R_{1\alpha} \partial R_{1\alpha'}} {}^3\xi_1(\alpha') = 0.28098533333261 \text{ Ha} \end{aligned} \quad (45)$$

$$\begin{aligned} \frac{\partial^2 f(x_1, x_2)}{\partial x_2^2} &= \frac{f(x_1, x_2 + w) + f(x_1, x_2 - w) - 2f(x_1, x_2)}{w^2} \\ &= 2 \sum_{\alpha\alpha'} {}^3\xi_2(\alpha) \frac{\partial^2 \epsilon_n}{\partial R_{2\alpha} \partial R_{2\alpha'}} {}^3\xi_2(\alpha') = 0.28098633333418 \text{ Ha} \end{aligned} \quad (46)$$

Now we have to consider the off diagonal terms. As we are only considering the optical mode we have ${}^3\xi_{m1} = -{}^3\xi_{m2}$ therefore leading to:

$$f(x_1 + h, x_2 + w) = \epsilon_n ({}^6R_0 + h\sqrt{2} \begin{bmatrix} {}^3\xi_1 \\ 0 \end{bmatrix} - h\sqrt{2} \begin{bmatrix} 0 \\ {}^3\xi_2 \end{bmatrix}) = \epsilon_n^0 - \frac{1}{2} \sum_{\alpha\alpha'} 2(h^2) {}^3\xi_1(\alpha) \frac{\partial^2 \epsilon_n}{\partial R_{1\alpha} \partial R_{2\alpha'}} {}^3\xi_2(\alpha') \quad (47)$$

And similarly for the other. We thus obtain:

$$\begin{aligned} \frac{\partial^2 f(x_1, x_2)}{\partial x_1 \partial x_2} &= \frac{f(x_1 + h, x_2 + w) + f(x_1 - h, x_2 - w) - f(x_1 + h, x_2 - w) - f(x_1 - h, x_2 + w)}{4hw} \\ &= - \sum_{\alpha\alpha'} {}^3\xi_1(\alpha) \frac{\partial^2 \epsilon_n}{\partial R_{1\alpha} \partial R_{2\alpha'}} {}^3\xi_2(\alpha') = -0.28098991666637 \text{ Ha} \end{aligned} \quad (48)$$

The zero point motion correction is then:

$$\begin{aligned} \epsilon_n(T) - \epsilon_{\text{without el-ph}} &= \sum_m \frac{1}{2\omega_m} \sum_{\kappa, \alpha} \sum_{\kappa', \alpha'} \frac{1}{\sqrt{M_\kappa}} {}^6\xi_m^*(\kappa\alpha) \frac{\partial^2 \epsilon_n}{\partial R_{\kappa, \alpha} \partial R_{\kappa', \alpha'}} \frac{1}{\sqrt{M_{\kappa'}}} {}^6\xi_m(\kappa'\alpha') \left[\frac{1}{2} \right] \quad (49) \\ &= \frac{27.211383 * 1000}{4 * 0.006204201 * 21894.16693} \left[\frac{0.28098533333261 + 0.28098633333418}{2} \right. \\ &\quad \left. + \frac{2 * 0.28098991666637}{1} \right] = 28.1445244522198 \text{ meV} \end{aligned} \quad (50)$$

2.3 AHC and NDDW by finite difference

Using the rigid ion approximation we can compute the zero point motion correction using the following two input files:

This gives us a correction of the HOMO band of $\epsilon_n(T) - \epsilon_{\text{without el-ph}} = 23.5012772629637 \text{ meV}$

By using the rigid ion approximation we neglect the non diagonal part of the Debye-Waller term. This term can be compute by finite difference.

$$\delta\varepsilon^{\text{NDDW}} = \sum_m \sum_{\kappa \neq \kappa'} \frac{1}{2\sqrt{M_\kappa M_{\kappa'}}\omega_m} \int d\mathbf{r}^3 \xi_{m\kappa}^* \Psi_{(0)}^*(\mathbf{r}) \frac{\partial^2 V_{Hxc}(\mathbf{r})}{\partial \mathbf{R}_\kappa \partial \mathbf{R}_{\kappa'}} \Psi_{(0)}(\mathbf{r})^3 \xi_{m\kappa'}^* \left[\langle n_m \rangle (T) + \frac{1}{2} \right] \quad (51)$$

where V_{Hxc} can be obtained by the Abinit program using the variable `prtvhxc 1` and the handle thought the post processing tool CUT3D.

$$\frac{\partial^2 V_{Hxc}(\mathbf{r})}{\partial \mathbf{R}_\kappa \partial \mathbf{R}_{\kappa'}} = \frac{V_{Hxc}(x_1 + h, x_2 + w) + V_{Hxc}(x_1 - h, x_2 - w) - V_{Hxc}(x_1 + h, x_2 - w) - V_{Hxc}(x_1 - h, x_2 + w)}{4hw} \quad (52)$$

The numerical integration of the finite difference derivative of V_{Hxc} with the unperturbed wavefunction is done with a small python program.

This gives us the following result:

$$\delta\varepsilon^{\text{NDDW}} = 4.64174889748 \text{ meV} \quad (53)$$

We then have a total correction of $23.5012772629637 + 4.64174889748 = 28.1430261604437 \text{ meV}$

We have now confidence in the AHC implementation inside ABINIT as the three different method lead to the same result (up to 10-2 meV) for the HOMO of diamond.

Finite difference	28.1445244522198
Brooks	28.142991182723
AHC+NDDW	28.1430261604437

3 Test of the three method at different q points

3.1 Finite difference on the eigenenergies

The expansion of the total energy of a periodic crystal with respect to small deviations of the atomics position from the equilibrium ones is (at second order):

$$E_{tot}(\Delta \vec{R}) = E_{tot}^0 + \sum_{a\kappa\alpha} \sum_{a'\kappa'\alpha'} \frac{1}{2} \underbrace{\left[\frac{\partial^2 E_{tot}}{\partial R_{\kappa\alpha}^a \partial R_{\kappa'\alpha'}^{a'}} \right]}_{C_{\kappa\alpha, \kappa'\alpha'}(a, a')} \Delta R_{\kappa\alpha}^a \Delta R_{\kappa'\alpha'}^{a'} \quad (54)$$

where $\Delta R_{\kappa\alpha}^a$ is the displacement along direction α of the atom κ in the cell labelled a . It's Fourier transform is:

$$\tilde{C}_{\kappa\alpha, \kappa'\alpha'}(\vec{q}) = \frac{1}{N} \sum_{aa'} C_{\kappa\alpha, \kappa'\alpha'}(a, a') e^{-i\vec{q} \cdot (\mathbf{R}_a - \mathbf{R}_{a'})} \quad (55)$$

$$= \sum_c C_{\kappa\alpha, \kappa'\alpha'}(0, c) e^{i\vec{q} \cdot \vec{R}_c} \quad (56)$$

In the last expression we have use the translational invariance by defining $R_c = R_{a'} - R_a$

Using Bloch theorem the eigendisplacement can be written as $U_m(\kappa\alpha a) = e^{i\vec{q} \cdot \mathbf{R}_a} U_{mq}(\kappa\alpha)$ and are solution of the following eigenvalue problem equation:

$$\sum_{\kappa'\alpha'a'} C_{\kappa\alpha,\kappa'\alpha'}(a,a')e^{iq\cdot R_{a'}}U_{mq}(\kappa'\alpha') = M_\kappa\omega_m^2 e^{iq\cdot R_a}U_{mq}(\kappa\alpha) \quad (57)$$

$$\sum_{\kappa'\alpha'a'} e^{-iq\cdot R_a}C_{\kappa\alpha,\kappa'\alpha'}(a,a')e^{iq\cdot R_{a'}}U_{mq}(\kappa'\alpha') = M_\kappa\omega_m^2 U_{mq}(\kappa\alpha) \quad (58)$$

$$\sum_{\kappa'\alpha'c} C_{\kappa\alpha,\kappa'\alpha'}(0,c)e^{iq\cdot R_c}U_{mq}(\kappa'\alpha') = M_\kappa\omega_m^2 U_{mq}(\kappa\alpha) \quad (59)$$

$$\tilde{C}_{\kappa\alpha,\kappa'\alpha'}(q)U_{mq}(\kappa'\alpha') = M_\kappa\omega_m^2 U_{mq}(\kappa\alpha) \quad (60)$$

The dynamical matrix (computed using DFPT in ABINIT) allows us to compute the phonon frequencies and the eigenvectors as solution of the following generalized eigenvalue problem:

$$\sum_{\kappa'\alpha'} \tilde{C}_{\kappa\alpha,\kappa'\alpha'}(\vec{q})\xi_{\kappa'\alpha'}(\vec{q}) = \sqrt{M_\kappa M_{\kappa'}}\omega_{\vec{q}}^2 \xi_{\kappa\alpha}(\vec{q}) \quad (61)$$

In electron atomic unit the carbon mass is $M_c = 12.0107u \cdot 1822.888502u^{-1} = 21894.16693$ with $1822.888502u^{-1}$ the mass of an electron in atomic mass unit.

3.1.1 Treatment of some possible problems

The dynamical equation in real space goes as follow:

$$\sum_{\kappa'\beta b} C_{\kappa\alpha,\kappa'\beta}(a,b)U_m(\kappa'\alpha'b) = M_\kappa\omega_m^2 U_m(\kappa\alpha a) \quad (62)$$

with the IFC matrix $C_{\kappa\alpha,\kappa'\beta}(a,b) = \frac{\partial^2 E_{tot}}{\partial R_{\kappa\alpha}^a \partial R_{\kappa'\beta}^b}$ an hermitian symmetric matrix. Therefore the IFC matrix is made of real numbers. Nevertheless the eigenvectors can be complex:

$$\sum_{\kappa'\beta b} C_{\kappa\alpha,\kappa'\beta}(a,b)(\Re U_m(\kappa'\alpha'b) + i\Im U_m(\kappa'\alpha'b)) = M_\kappa\omega_m^2 (\Re U_m(\kappa'\alpha'b) + i\Im U_m(\kappa'\alpha'b)) \quad (63)$$

Hence we can solve two equivalent equation either for the real or the imaginary part. Both will give the same result up to a phase factor $e^{i\theta}$.

Another problem arise when we solve the dynamical equation in reciprocal space, namely the fact that the dynamical matrices $\tilde{C}_{\kappa\alpha,\kappa'\beta}(q)$ can be a complex matrix.

First we can realise that quantities in q are related to quantities in $-q$ (time-reversal symmetry).

$$\tilde{C}_{\kappa\alpha,\kappa'\beta}(q) = \sum_b C_{\kappa\alpha,\kappa'\beta}(0,b)e^{iq\cdot R_b} \quad (64)$$

$$\tilde{C}_{\kappa\alpha,\kappa'\beta}(-q) = \sum_b C_{\kappa\alpha,\kappa'\beta}(0,b)e^{-iq\cdot R_b} \quad (65)$$

$$= \left(\sum_b C_{\kappa\alpha,\kappa'\beta}(0,b)e^{iq\cdot R_b} \right)^* \quad (66)$$

$$= (\tilde{C}_{\kappa\alpha,\kappa'\beta}(q))^* \quad (67)$$

$$\sum_{\kappa'\beta} \tilde{C}_{\kappa\alpha,\kappa'\beta}(-q)U_{m(-q)}(\kappa'\beta) = M_\kappa\omega_{m(-q)}^2 U_{m(-q)}(\kappa\alpha) \quad (68)$$

$$\sum_{\kappa'\beta} \tilde{C}_{\kappa\alpha,\kappa'\beta}^*(q)U_{m(-q)}(\kappa'\beta) = M_\kappa\omega_{m(-q)}^2 U_{m(-q)}(\kappa\alpha) \quad (69)$$

$$(70)$$

By taking the complex conjugate of the entire last equation we have:

$$\sum_{\kappa'\beta} \tilde{C}_{\kappa\alpha,\kappa'\beta}(q) U_{m(-q)}^*(\kappa'\beta) = M_\kappa \omega_{m(-q)}^2 U_{m(-q)}^*(\kappa\alpha) \quad (71)$$

Therefore it must exist a mode m such as $\omega_{mq}^2 = \omega_{m(-q)}^2$ and $U_{mq}(\kappa\beta) = U_{m(-q)}^*(\kappa'\beta)$. We can then define a real and imaginary displacement as $\Re z = \frac{1}{2}(z + z^*)$:

$$U_{m+}(\kappa'\beta b) = \frac{1}{\sqrt{2}} \left[e^{iq \cdot R_b U_{mq}(\kappa'\beta) + e^{-iq \cdot R_b U_{mq}^*(\kappa'\beta)} \right] \quad (72)$$

$$U_{m-}(\kappa'\beta b) = \frac{i}{\sqrt{2}} \left[e^{iq \cdot R_b U_{mq}(\kappa'\beta) - e^{-iq \cdot R_b U_{mq}^*(\kappa'\beta)} \right] \quad (73)$$

$$(74)$$

where the prefactor is there to keep normalization:

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} & \frac{-i}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{-i}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{i}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad (75)$$

It can be wise at this point to illustrate this with an example (see 3.1.3).

3.1.2 Coming back to the initial problem...

Using the python script 5.3 we find the following phonon frequencies and eigenvectors at L :

phonon freq [Ha]	eigenvector			normalized			type
	atom 1			atom 2			
0.00484778357031	-0.4082485	0.4082485	0.4082485	-0.40824808	0.40824808	0.40824808	LA
0.00579170275934	-0.40824809	0.40824808	0.40824808	0.4082485	-0.4082485	-0.40824849	LO
0.00248351966296	0.47691054	0.52026611	-0.04335557	0.47691035	0.52026591	-0.04335555	TA
0.00248351966648	0.28867519	-0.28867519	0.57735038	0.28867508	-0.28867508	0.57735015	TA
0.00566655738902	-0.5135939	-0.48519546	-0.02839843	0.5135941	0.48519566	0.02839845	TO
0.00566655739139	0.23827501	-0.33629489	0.5745699	-0.2382751	0.33629502	-0.57457013	TO

As we can see the transverse mode are degenerate and orthogonal one to another. This mean that any linear combination of these two is also a valid eigenvectors. We can then choose another eigenvector by doing a rotation of the two orthogonal eigenvectors and have a "better" set of eigenvectors. By better we mean that respect more the symmetry of the lattice. We will not do that here.

The zinc blende structure of diamond being describe by the following primitive vectors in the direct lattice:

```
rprim 0.0 0.5 0.5
        0.5 0.0 0.5
        0.5 0.5 0.0
```

and in the reciprocal lattice

```
kprim -1.0 1.0 1.0
        1.0 -1.0 1.0
        1.0 1.0 -1.0    (x2\pi)
```

going from one to the other through the following relationship¹:

¹We have the following relationship: $\vec{R}_i = \sum_j rprimd(i, j) \cdot \vec{e}_{direct, j}$ and $\vec{G}_l = \sum_m gprimd(l, m) \cdot \vec{e}_{recip, m}$

$$k_1 = 2\pi \frac{a_2 \times a_3}{a_1 \cdot a_2 \times a_3} \quad k_2 = 2\pi \frac{a_3 \times a_1}{a_2 \cdot a_3 \times a_1} \quad k_3 = 2\pi \frac{a_1 \times a_2}{a_1 \cdot a_2 \times a_2} \quad (76)$$

The q-vector reduced reciprocal coordinate is $q = X = (0.0 \ 0.5 \ 0.5)$. This mean that we take zero times the first vector (= ligne) of **kprim** plus 0.5 times the second and third: $0 \cdot (-1.0 \ 1.0 \ 1.0) + 0.5 \cdot (1.0 \ -1.0 \ 1.0) + 0.5 \cdot (1.0 \ 1.0 \ -1.0) = (1.0 \ 0.0 \ 0.0)$.

The polarization vector is $e^{i(\vec{q} \cdot \vec{R})} = \cos(\vec{q} \cdot \vec{R}) + i \sin(\vec{q} \cdot \vec{R})$. We then have:

$$\vec{q} \cdot \vec{R} = \begin{pmatrix} 0.0 & 0.5 & 0.5 \\ 0.5 & 0.0 & 0.5 \\ 0.5 & 0.5 & 0.0 \end{pmatrix} \begin{pmatrix} 1.0 \\ 0.0 \\ 0.0 \end{pmatrix} = \begin{pmatrix} 0.0 \\ 0.5 \\ 0.5 \end{pmatrix} \quad (77)$$

This mean that in the second and third direction, we will have $\cos(2\pi \frac{1}{2})$. Therefore we need a supercell 1x2x2.

For the point L we have $\vec{q} \cdot \vec{R} = (0.5 \ 0.0 \ 0.0)$ and then a supercell 2x1x1.

It is also important to note that if we have degenerate eigenvalue then the curvature of the total energy will be the same for the two displacements but the slope of the eigenenergies will not. We then have to make an average of the degenerate states.

Fortunately enough for those two q-points we have that the imaginary part die. However for an arbitrary q-point this is necessary true. We then have to add the complex conjugate of the perturbation so that the imaginary part die. This implies that we will also take the response of the system in a linear way. It is then crucial to take into account the quadratic term that play a role in the temperature dependence (because we need second derivative of the potential).

$$\text{perturbation } \phi e^{iq \cdot r} \rightarrow \text{response } \alpha \phi e^{iq \cdot r} \quad (78)$$

$$\phi^* e^{-iq \cdot r} \rightarrow \text{response } \alpha^* \phi^* e^{iq \cdot r} \quad (79)$$

$$+ \phi^2 e^{2iq \cdot r} + \phi \phi^* e^{2i(q-q) \cdot r} + \phi^{*2} e^{-2iq \cdot r} \quad (80)$$

3.1.3 Example

Let us choose the q-point $q = (0.5\frac{2}{3} \ 0.0 \ 0.0)$. We have:

$$\frac{1}{3} \cdot (-1.0 \ 1.0 \ 1.0) + 0.0 \cdot (1.0 \ -1.0 \ 1.0) + 0.0 \cdot (1.0 \ 1.0 \ -1.0) = (-\frac{1}{3} \ \frac{1}{3} \ \frac{1}{3}) \quad (81)$$

$$\vec{q} \cdot \vec{R} = (-\frac{1}{3} \ \frac{1}{3} \ \frac{1}{3}) \begin{pmatrix} 0.0 & 0.5 & 0.5 \\ 0.5 & 0.0 & 0.5 \\ 0.5 & 0.5 & 0.0 \end{pmatrix} = (\frac{1}{3} \ 0.0 \ 0.0) \quad (82)$$

The polarization vector is $e^{i(\vec{q} \cdot \vec{R})} = \cos(\vec{q} \cdot \vec{R}) + i \sin(\vec{q} \cdot \vec{R}) = \cos(\frac{2\pi}{3}) + i \sin(\frac{2\pi}{3})$. We then have the figure 1 and the associate table 1 :

Table 1: Tabular of polazisation vectors

atom #	U_{m+}	U_{m-}
1-2	eq+ Δ_n	eq+0
3-4	eq- $\frac{1}{2}\Delta_n$	eq+ $\frac{\sqrt{3}}{2}\Delta_n$
5-6	eq- $\frac{1}{2}\Delta_n$	eq- $\frac{\sqrt{3}}{2}\Delta_n$

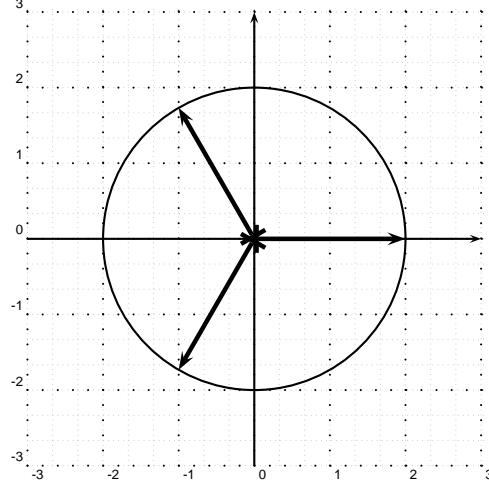


Figure 1: Vector representation in the complex plane

Table 2: Contribution of the q-point L by finite difference on the electronic eigenenergies at Γ . The two modes TA and TO are doubly degenerate.

Mode	Etotal [Ha]	$\epsilon_{\Gamma}^{\text{HOMO}}$			ω [Ha]	h	ZPM [meV]
EQ	-12.069553133	0.4696587869	0.4696587869	0.4696587876			
LA	-12.069538939	0.4696477345	0.4696586807	0.4696586807	0.0048011	$0.0075 + \mathcal{O}(h^2)$	-8.64085987
	-12.069496347	0.469614579	0.4696583609	0.4696583609	0.0048015	$0.015 + \mathcal{O}(h^2)$	-8.63986862
LO					0.0048009	$\mathcal{O}(h^4)$	-8.64119034
	-12.069532518	0.4696677623	0.4696932118	0.4696932118	0.0057860	$0.0075 + \mathcal{O}(h^2)$	49.53173726
	-12.069470678	0.4696946868	0.4697963469	0.4697963469	0.0057859	$0.015 + \mathcal{O}(h^2)$	49.48887681
					0.0057861	$\mathcal{O}(h^4)$	49.54602351
TA	-12.0695494265	0.4696517289	0.4696562006	0.469663362	0.0024534	$0.0075 + \mathcal{O}(h^2)$	-7.60985325
	-12.0695383075	0.4696295381	0.4696501798	0.4696763688	0.0024533	$0.015 + \mathcal{O}(h^2)$	-7.60814283
TO					0.0024534	$\mathcal{O}(h^4)$	-7.61042338
	-12.0695333275	0.4696614745	0.4697129162	0.4697208717	0.0056712	$0.0075 + \mathcal{O}(h^2)$	77.206078266
	-12.0694739095	0.4696684806	0.4698871093	0.4698956828	0.0056713	$0.015 + \mathcal{O}(h^2)$	77.092900567
					0.0056712	$\mathcal{O}(h^4)$	77.243804642
Total						$\mathcal{O}(h^4)$	180.1715957

3.1.4 Coming back again to the initial problem

Using what has been developed previously we obtain the table 2.

3.2 Finite difference on the occupation

This is not possible with this version of ABINIT.

3.3 AHC + NDDW

3.3.1 Theory

The Hamiltonian of the Schrödinger equation can be split into an unperturbed Hamiltonian and a perturbed one:

$$H = H_0(r, R) + \sum_{i=1}^{N_N} -\frac{1}{2M_i} \nabla_{R_i}^2 \quad (83)$$

The energy is defined as the expectation value of this Hamiltonian:

$$E = \langle N | H_0 | N \rangle + \sum_Q \hbar \omega_Q \left[\langle n_Q \rangle + \frac{1}{2} \right] \quad (84)$$

with N the many body wavefunction, n_Q the Bose-Einstein distribution and the right part of the equation coming from the decoupled harmonic oscillator.

We can see that if we take the partial derivative of the total energy with respect to this bosonic occupation number we have: $\frac{\partial E}{\partial \langle n_i \rangle} = \hbar \omega_i$ for each mode i .

Therefore the electron-phonon renormalization of the total energy is:

$$E - E_{\text{without e-ph}} = \delta E = \sum_Q \frac{\partial E}{\partial \langle n_i \rangle} \left[\langle n_Q \rangle + \frac{1}{2} \right] \quad (85)$$

By making use of the Janak theorem we have: $\varepsilon_i = \frac{\partial E}{\partial f_i}$ with f_i the electronic occupation number.

$$\delta \left(\frac{\partial E}{\partial f_i} \right) = \sum_Q \frac{\partial^2 E}{\partial f_i \partial n_i} \left[\langle n_Q \rangle + \frac{1}{2} \right] \quad (86)$$

$$\delta \varepsilon_n = \sum_Q \frac{\partial \varepsilon_n}{\partial \langle n_Q \rangle} \left[\langle n_Q \rangle + \frac{1}{2} \right] \quad (87)$$

As a side remark, in the second quantization formalism the bosonic occupation can be described in term of creation and annihilation operators:

$$\langle n_Q \rangle = \langle N | a_Q^\dagger a_Q | N \rangle \quad (88)$$

$$\varepsilon_n = \langle n_i | H_0 | n_i \rangle + \sum_Q \hbar \omega_Q \left[\langle n_i | a_Q^\dagger a_Q | n_i \rangle + \frac{1}{2} \right] \quad (89)$$

3.3.2 For the L point

We can create a function that compute the second derivative by finite difference:

$$F(V_{Hxc}[h_i, {}^6\xi_m]) = \frac{\partial^2 V_{Hxc}}{\partial h^2} \quad (90)$$

So for example if we only take one h then we have a second order second finite difference derivative and if we define ${}^6\xi_m$ as previously by ${}^6\xi_m = \begin{bmatrix} {}^3\xi_1 \\ {}^3\xi_2 \end{bmatrix}$ and ${}^6\xi_m^\dagger = \begin{bmatrix} {}^3\xi_1 \\ -{}^3\xi_2 \end{bmatrix}$ we get:

$$\begin{aligned} F(V_{Hxc}[h_i, {}^6\xi_m]) &= \frac{V_{Hxc} \left[{}^6R_0 + h \begin{bmatrix} {}^3\xi_1 \\ {}^3\xi_2 \end{bmatrix} \right] + V_{Hxc} \left[{}^6R_0 - h \begin{bmatrix} {}^3\xi_1 \\ {}^3\xi_2 \end{bmatrix} \right] - 2V_{Hxc} [{}^6R_0]}{h^2} \\ &= \sum_{\kappa\alpha, \kappa'\beta} {}^6\xi_{\kappa\alpha} \frac{\partial^2 V_{Hxc}}{\partial R_{\kappa\alpha} \partial R_{\kappa'\beta}} {}^6\xi_{\kappa'\beta} e^{iq \cdot (R_{\kappa'\beta} - R_{\kappa\alpha})} \end{aligned} \quad (91)$$

and also:

$$\begin{aligned} F(V_{Hxc}[h_i, {}^6\xi_m^\dagger]) &= \frac{V_{Hxc} \left[{}^6R_0 + h \begin{bmatrix} {}^3\xi_1 \\ -{}^3\xi_2 \end{bmatrix} \right] + V_{Hxc} \left[{}^6R_0 - h \begin{bmatrix} {}^3\xi_1 \\ -{}^3\xi_2 \end{bmatrix} \right] - 2V_{Hxc} [{}^6R_0]}{h^2} \\ &= \sum_{\alpha\beta, \kappa=\kappa'} {}^6\xi_{\kappa\alpha} \frac{\partial^2 V_{Hxc}}{\partial R_{\kappa\alpha} \partial R_{\kappa'\beta}} {}^6\xi_{\kappa'\beta} - \sum_{\alpha\beta, \kappa \neq \kappa'} {}^6\xi_{\kappa\alpha} \frac{\partial^2 V_{Hxc}}{\partial R_{\kappa\alpha} \partial R_{\kappa'\beta}} {}^6\xi_{\kappa'\beta} e^{iq \cdot (R_{\kappa'\beta} - R_{\kappa\alpha})} \end{aligned} \quad (92)$$

with ${}^6\xi_{\kappa\alpha} = \begin{bmatrix} 3\xi_1 \\ 0 \end{bmatrix}$, ${}^6\xi_{\kappa'\beta} = \begin{bmatrix} 0 \\ 3\xi_2 \end{bmatrix}$ in the case of two atoms and with $\kappa \neq \kappa'$.

Following this notation the non-diagonal term will be obtain by using the following combination:

$$\frac{1}{2} (F(V_{Hxc}[h_i, {}^6\xi_m]) - F(V_{Hxc}[h_i, {}^6\xi_m^\dagger])) = \sum_{\alpha\beta, \kappa \neq \kappa'} {}^6\xi_{\kappa\alpha} \frac{\partial^2 V_{Hxc}}{\partial R_{\kappa\alpha} \partial R_{\kappa'\beta}} {}^6\xi_{\kappa'\beta} e^{iq \cdot (R_{\kappa'\beta} - R_{\kappa\alpha})} \quad (93)$$

This approach will give us the non diagonal part of V_{Hxc} but not the non diagonal Debye-Waller term!

Let us go back to the original set of equations:

$$\delta\varepsilon_{\vec{k}n} = \frac{1}{N_{\vec{q}}} \sum_{\vec{q}j} \frac{\partial \varepsilon_{\vec{k}n}}{\partial n_{\vec{q}j}} \left(\langle \hat{n}_{\vec{q}j} \rangle + \frac{1}{2} \right) \quad (94)$$

$$\frac{\partial \varepsilon_{\vec{k}n}}{\partial n_{\vec{q}j}} = \frac{\partial \varepsilon_{\vec{k}n}^{(DW^{diag})}}{\partial n_{\vec{q}j}} + \frac{\partial \varepsilon_{\vec{k}n}^{(DW^{non-diag})}}{\partial n_{\vec{q}j}} + \frac{\partial \varepsilon_{\vec{k}n}^{(DW^{aFan})}}{\partial n_{\vec{q}j}} \quad (95)$$

$$\begin{aligned} \frac{\partial \varepsilon_{\vec{k}n}^{(DW^{diag})}}{\partial n_{\vec{q}j}} &= -\frac{\hbar}{2} \omega_{\vec{q}j} \sum_{\substack{\kappa, \kappa' \\ n' \alpha \beta}} \frac{\langle \phi_{\vec{k}n} | \nabla_{\kappa, \alpha} V_{\kappa} | \phi_{\vec{k}n'} \rangle \langle \phi_{\vec{k}n'} | \nabla_{\kappa', \beta} V_{\kappa'} | \phi_{\vec{k}n} \rangle}{\varepsilon_{\vec{k}n} - \varepsilon_{\vec{k}n'}} \\ &\times \left[\frac{\vec{\xi}_{\alpha}(\kappa, \vec{q}j) \vec{\xi}_{\beta}(\kappa, -\vec{q}j)}{M_{\kappa}} + \frac{\vec{\xi}_{\alpha}(\kappa', \vec{q}j) \vec{\xi}_{\beta}(\kappa', -\vec{q}j)}{M_{\kappa'}} \right] \end{aligned} \quad (96)$$

$$\begin{aligned} \frac{\partial \varepsilon_{\vec{k}n}^{(DW^{non-diag})}}{\partial n_{\vec{q}j}} &= \frac{\hbar}{2\omega_{\vec{q}j}} \sum_{\substack{\kappa, \kappa' \\ n' \alpha \beta}} \langle \phi_{\vec{k}n} | \nabla_{\kappa, \alpha} \nabla_{\kappa', \beta} H | \phi_{\vec{k}n} \rangle \\ &\times \left\{ \frac{\vec{\xi}_{\alpha}(\kappa, \vec{q}j) \vec{\xi}_{\beta}(\kappa', -\vec{q}j)}{\sqrt{M_{\kappa} M_{\kappa'}}} e^{i\vec{q} \cdot (\vec{R}_{\kappa'} - \vec{R}_{\kappa})} \right. \\ &\left. - \frac{1}{2} \left[\frac{\vec{\xi}_{\alpha}(\kappa, \vec{q}j) \vec{\xi}_{\beta}(\kappa, -\vec{q}j)}{M_{\kappa}} + \frac{\vec{\xi}_{\alpha}(\kappa', \vec{q}j) \vec{\xi}_{\beta}(\kappa', -\vec{q}j)}{M_{\kappa'}} \right] \right\} \end{aligned} \quad (97)$$

$$\begin{aligned} \frac{\partial \varepsilon_{\vec{k}n}^{(aFan)}}{\partial n_{\vec{q}j}} &= \frac{\hbar}{\omega_{\vec{q}j}} \sum_{\kappa, \kappa', n'} \sqrt{\frac{1}{M_{\kappa} M_{\kappa'}}} \langle \phi_{\vec{k}n} | \vec{\xi}_{\kappa}(\vec{q}j) \cdot \nabla_{\kappa} V_{\kappa} | \phi_{\vec{k}+\vec{q}n'} \rangle \\ &\times \frac{\langle \phi_{\vec{k}+\vec{q}n'} | \vec{\xi}_{\kappa'}(-\vec{q}j) \cdot \nabla_{\kappa'} V_{\kappa'} | \phi_{\vec{k}n} \rangle}{\varepsilon_{\vec{k}n} - \varepsilon_{\vec{k}+\vec{q}n'}} \end{aligned} \quad (98)$$

The equation (97) for the non-diagonal Debye-Waller term is made off two terms. The first one contain a phase factor that will force us to use supercell while the other does not.

To be completely clear, let us fully develop this term for each of the two atoms (working in atomic unit $\hbar = 1$):

$$\begin{aligned} \frac{\partial \varepsilon_{\Gamma n}^{NDDW}}{\partial n_{\vec{q}j}} &= \frac{1}{2\omega_{\vec{q}j} M_{\text{carbon}}} \sum_{\alpha\beta} \left[\vec{\xi}_{1\alpha} \frac{\partial^2 V_{Hxc}}{\partial R_{1\alpha} \partial R_{1\beta}} \vec{\xi}_{1\beta} e^{i\vec{q} \cdot (\vec{R}_1 - \vec{R}_1)} + \vec{\xi}_{1\alpha} \frac{\partial^2 V_{Hxc}}{\partial R_{1\alpha} \partial R_{2\beta}} \vec{\xi}_{2\beta} e^{i\vec{q} \cdot (\vec{R}_2 - \vec{R}_1)} \right. \\ &+ \vec{\xi}_{2\alpha} \frac{\partial^2 V_{Hxc}}{\partial R_{2\alpha} \partial R_{1\beta}} \vec{\xi}_{1\beta} e^{i\vec{q} \cdot (\vec{R}_1 - \vec{R}_2)} + \vec{\xi}_{2\alpha} \frac{\partial^2 V_{Hxc}}{\partial R_{2\alpha} \partial R_{2\beta}} \vec{\xi}_{2\beta} e^{i\vec{q} \cdot (\vec{R}_2 - \vec{R}_2)} \\ &- \frac{1}{2} \left(\vec{\xi}_{1\alpha} \frac{\partial^2 V_{Hxc}}{\partial R_{1\alpha} \partial R_{1\beta}} \vec{\xi}_{1\beta} + \vec{\xi}_{1\alpha} \frac{\partial^2 V_{Hxc}}{\partial R_{1\alpha} \partial R_{1\beta}} \vec{\xi}_{1\beta} + \vec{\xi}_{1\alpha} \frac{\partial^2 V_{Hxc}}{\partial R_{1\alpha} \partial R_{2\beta}} \vec{\xi}_{1\beta} + \vec{\xi}_{2\alpha} \frac{\partial^2 V_{Hxc}}{\partial R_{1\alpha} \partial R_{2\beta}} \vec{\xi}_{2\beta} \right. \\ &\left. \left. + \vec{\xi}_{2\alpha} \frac{\partial^2 V_{Hxc}}{\partial R_{2\alpha} \partial R_{1\beta}} \vec{\xi}_{2\beta} + \vec{\xi}_{1\alpha} \frac{\partial^2 V_{Hxc}}{\partial R_{2\alpha} \partial R_{1\beta}} \vec{\xi}_{1\beta} + \vec{\xi}_{2\alpha} \frac{\partial^2 V_{Hxc}}{\partial R_{2\alpha} \partial R_{2\beta}} \vec{\xi}_{2\beta} + \vec{\xi}_{2\alpha} \frac{\partial^2 V_{Hxc}}{\partial R_{2\alpha} \partial R_{2\beta}} \vec{\xi}_{2\beta} \right) \right] \end{aligned} \quad (99)$$

As we can see $\frac{\partial \varepsilon_{fn}^{\text{NDDW}}}{\partial n_{\vec{q}j}} = 0$ when $\kappa = \kappa'$. This is the reason why we call it (maybe misleading the reader) the non diagonal Debye-Waller term although it is not the full off diagonal Debye-Waller term. Indeed if we insert a $(1 - \delta_{\kappa\kappa'})$ inside the DW equation:

$$\frac{\partial \varepsilon_{kn}^{\text{DW}}}{\partial n_{\vec{q}j}} = \frac{1}{2N_Q \omega_{qj} \sqrt{M_\kappa M_{\kappa'}}} \sum_{\kappa\alpha, \kappa'\beta} \langle \phi_{\vec{k}n} | \nabla_{\kappa\alpha} \nabla_{\kappa'\beta} V | \phi_{\vec{k}n} \rangle \xi_{\kappa\alpha}(\vec{q}_j) \xi_{\kappa'\beta}(-\vec{q}_j) e^{-i\vec{q} \cdot (\vec{R}_{\kappa'} - \vec{R}_\kappa)} \quad (100)$$

we obtain

$$\begin{aligned} \frac{\partial \varepsilon_{kn}^{\text{true NDDW}}}{\partial n_{\vec{q}j}} &= \frac{1}{2N_Q \omega_{qj} \sqrt{M_\kappa M_{\kappa'}}} \sum_{\kappa\alpha, \kappa'\beta} \langle \phi_{\vec{k}n} | \nabla_{\kappa\alpha} \nabla_{\kappa'\beta} V_{\text{Hxc}} | \phi_{\vec{k}n} \rangle \left[\xi_{\kappa\alpha}(\vec{q}_j) \xi_{\kappa'\beta}(-\vec{q}_j) e^{i\vec{q} \cdot (\vec{R}_{\kappa'} - \vec{R}_\kappa)} \right. \\ &\quad \left. - \frac{1}{2} (\xi_{\kappa\alpha}(\vec{q}_j) \xi_{\kappa\beta}(\vec{q}_j) \delta_{\kappa\kappa'} + \xi_{\kappa'\alpha}(\vec{q}_j) \xi_{\kappa'\beta}(\vec{q}_j) \delta_{\kappa\kappa'}) \right] \end{aligned} \quad (101)$$

For the derivation of what we call non diagonal Debye-Waller see Xavier's notes and Paul's thesis. Let us come back to the equation (99). The first term is obtained on a supercell 2x1x1 using the following finite difference relationship:

$$\delta \varepsilon_{\Gamma\text{Homo}}^{\text{NDDW part 1}}(\vec{q}_j) = \frac{1}{4\omega_{\vec{q}j} M_{\text{carbon}} h^2} \left(V_{\text{Hxc}} \left[{}^6R_0 + h \begin{bmatrix} {}^3\xi_1 \\ {}^3\xi_2 \end{bmatrix} \right] + V_{\text{Hxc}} \left[{}^6R_0 - h \begin{bmatrix} {}^3\xi_1 \\ {}^3\xi_2 \end{bmatrix} \right] - 2V_{\text{Hxc}} [{}^6R_0] \right) \quad (102)$$

The eigen vector are obtained by solving the eigenvalue problem of the Dynamical matrix (this matrix has previously been calculated using perturbation theory with ABINIT). The script used to compute this using finite difference is given in annex 5.4. The script used to post process the result and actually perform the finite difference is given in annex 5.5.

The second term is obtained on the primitive cell:

$$\delta \varepsilon_{\Gamma\text{Homo}}^{\text{NDDW part 2}}(\vec{q}_j) = \frac{-1}{8\omega_{\vec{q}j} M_{\text{carbon}}} \sum_{\alpha\kappa, \kappa'\beta} \langle \phi_{\vec{k}n} | \nabla_{\kappa\alpha} \nabla_{\kappa'\beta} V_{\text{Hxc}} | \phi_{\vec{k}n} \rangle (\xi_{\kappa\alpha}(\vec{q}_j) \xi_{\kappa\beta}(\vec{q}_j) + \xi_{\kappa'\alpha}(\vec{q}_j) \xi_{\kappa'\beta}(\vec{q}_j)) \quad (103)$$

Let us develop the first part of this last equation assuming that the integration with the unperturbed wavefunction has been done:

$$\sum_{\alpha\kappa, \beta\kappa'} \frac{\partial^2 V_{\text{Hxc}}}{\partial R_{\kappa\alpha} \partial R_{\kappa'\beta}} [\xi_{\kappa\alpha}(\vec{q}_j) \xi_{\kappa\beta}(\vec{q}_j)] = \sum_{\alpha, \kappa'\beta} \left[\frac{\partial^2 V_{\text{Hxc}}}{\partial R_{1\alpha} \partial R_{\kappa'\beta}} \xi_{1\alpha}(\vec{q}_j) \xi_{1\beta}(\vec{q}_j) + \frac{\partial^2 V_{\text{Hxc}}}{\partial R_{2\alpha} \partial R_{\kappa'\beta}} \xi_{2\alpha}(\vec{q}_j) \xi_{2\beta}(\vec{q}_j) \right] \quad (104)$$

Let us define a collective translation displacement as:

$$\frac{\partial}{\partial T_\beta} \triangleq \sum_{\kappa'} \frac{\partial}{\partial R_{\kappa'\beta}} \quad (105)$$

We can replace the first and second part of equation (104) by:

$$\sum_{\kappa'} \frac{\partial^2 V_{\text{Hxc}}}{\partial R_{1\alpha} \partial R_{\kappa'\beta}} \xi_{1\alpha}(\vec{q}_j) \xi_{1\beta}(\vec{q}_j) = \frac{\partial^2 V_{\text{Hxc}}}{\partial R_{1\alpha} \partial T_\beta} \xi_{1\alpha}(\vec{q}_j) \xi_{1\beta}(\vec{q}_j) \quad (106)$$

Displacing every atoms in direction β is indeed equivalent to make a collective translation of all the atoms in direction β .

We can make additional definitions :

$$(S_\beta)_{\kappa\alpha} \triangleq \xi_{\kappa\alpha}(\vec{q}_j)\xi_{\kappa\beta}(-\vec{q}_j) \quad (107)$$

$$\frac{\partial}{\partial S_\beta} \triangleq \sum_{\kappa\alpha} S_{\beta\kappa\alpha} \frac{\partial}{\partial R_{\kappa\alpha}} \quad (108)$$

The vector $(S_\beta)_{\kappa\alpha}$ can be seen as the second order displacement vector of atoms κ in the direction α .

Using these definition we obtain the following relation:

$$\sum_{\kappa\alpha\beta} \frac{\partial^2 V_{\text{Hxc}}}{\partial R_{\kappa\alpha} \partial T_\beta} \xi_{\kappa\alpha}(\vec{q}_j) \xi_{\kappa\beta}(-\vec{q}_j) = \sum_{\beta} \frac{\partial^2 V_{\text{Hxc}}}{\partial S_\beta \partial T_\beta} = \frac{\partial^2 V_{\text{Hxc}}}{\partial S_x \partial T_x} + \frac{\partial^2 V_{\text{Hxc}}}{\partial S_y \partial T_y} + \frac{\partial^2 V_{\text{Hxc}}}{\partial S_z \partial T_z} \quad (109)$$

We can develop the second part of equation (103) with $(\kappa \leftrightarrow \kappa')$.

The second derivative with respect to two different variable is given using finite difference at second order by:

$$\frac{\partial^2 V_{\text{Hxc}}}{\partial S_x \partial T_x} = \frac{1}{4\Delta S_x \Delta T_x} [V_{\text{Hxc}}(\Delta S_x, \Delta T_x) - V_{\text{Hxc}}(-\Delta S_x, \Delta T_x) - V_{\text{Hxc}}(\Delta S_x, -\Delta T_x) + V_{\text{Hxc}}(-\Delta S_x, -\Delta T_x)] \quad (110)$$

The script used to create the input file is in annex 5.6 and the according post processing script is in annex 5.7.

The results are summarize in table 3. In this table the phonon frequency used are those obtained using response function into ABINIT.

Mode	Freq [Ha]	NDDW (1st term) [meV]	NDDW (2d term) [meV]	h
LA	0.00480084	-4.59786358263	-7.49197847011	0.05
		-4.60299232487	-7.5042229987763882	0.01
		-4.60409836002	-7.5047178442787414	0.002
LO	0.005785943	-6.65295481802	-6.21640933624	0.05
		-6.65429556431	-6.22655697987	0.01
		-6.65446148352	-6.22696403862	0.002
TA1	0.002453274	-17.2206027567	-14.6601573579	0.05
		-17.233165437	-14.6850504687	0.01
		-17.2425160304	-14.6860419527	0.002
TA2		-17.2205996576		0.05
		-17.2331446103		0.01
		-17.24189929		0.002
TO1	0.005671174	-4.14905724703	-6.3417818152	0.05
		-4.15438904168	-6.35255171028	0.01
		-4.15577332906	-6.35298184519	0.002
TO2		-4.14905589227		0.05
		-4.15438867483		0.01
		-4.15576539564		0.002
Total		-53.9901339542	55.71226615255	0.05
		-54.0323756529	55.8059843366064	0.01
		-54.0545138887	55.8097294786787	0.002
Grand total		1.7552155899786968		0.002

Table 3: NDDW contribution of the $q = L$ point at $k = \Gamma$ computed using finite difference of second order for diamond in a supercell.

3.3.3 Other method

There is a much simpler method to compute the second part of equation (99). This second term is obtained on a primitive cell using the following finite difference relationship:

$$\delta\varepsilon_{\Gamma\text{Homo}}^{\text{NDDW part 2}}(\vec{q}_j) = \frac{1}{4\omega_{\vec{q}_j} M_{\text{carbon}} h^2} \left(V_{\text{Hxc}} \left[{}^6R_0 + h \begin{bmatrix} {}^3\xi_1 \\ {}^3\xi_1 \end{bmatrix} \right] + V_{\text{Hxc}} \left[{}^6R_0 - h \begin{bmatrix} {}^3\xi_1 \\ {}^3\xi_1 \end{bmatrix} \right] - 2V_{\text{Hxc}} [{}^6R_0] \right) \quad (111)$$

Within the specific case we are working on we have that $\vec{\xi}_{1\alpha} \frac{\partial^2 V_{\text{Hxc}}}{\partial R_{1\alpha} \partial R_{1\beta}} \vec{\xi}_{1\beta} = \vec{\xi}_{2\alpha} \frac{\partial^2 V_{\text{Hxc}}}{\partial R_{2\alpha} \partial R_{2\beta}} \vec{\xi}_{2\beta} = \vec{\xi}_{2\alpha} \frac{\partial^2 V_{\text{Hxc}}}{\partial R_{1\alpha} \partial R_{2\beta}} \vec{\xi}_{2\beta} = ..$ and we can simply take $8\vec{\xi}_{1\alpha} \frac{\partial^2 V_{\text{Hxc}}}{\partial R_{1\alpha} \partial R_{1\beta}} \vec{\xi}_{1\beta}$

As this method is simpler to deal with we can push the convergence further as in table 4.

Mode	Freq [Ha]	NDDW (1st) [meV]	NDDW (2d) [meV]	Sum	h
LA	0.00480084	-4.59786358263	-7.50117398499	2.903310402	0.05
		-4.60299232487	-7.50459181892	2.901599494	0.01
		-4.60409836002	-7.50472757408	2.900629214	0.002
			-7.50473322957	2.900634869	0.002 and 0.004
LO	0.005785943	-4.68829685445	-7.50473030456	num. err.	0.0005
		-6.65295481802	-6.22402647327	-0.428928344	0.05
		-6.65429556431	-6.22686242355	-0.427433140	0.01
		-6.65446148352	-6.22697561364	-0.427485869	0.002
TA	0.002453274		-6.2269805723	-0.427480911	0.002 and 0.004
		-6.65440525359	-6.22697527827	num. err.	0.0005
		-17.2206027567	-14.6794488903	-2.541153866	0.05
		-17.233165437	-14.6858244554	-2.547340981	0.01
TO	0.005671174	-17.2425160304	-14.6860828891	-2.556433141	0.002
			-14.6860945599	-2.556421470	0.002 and 0.004
		-17.3956029576	-14.6860889531	num. err.	0.0005
		-4.14905724703	-6.35012817688	2.201070929	0.05
		-4.15438904168	-6.35288617935	2.198497137	0.01
		-4.15577332906	-6.35299789184	2.197224562	0.002
			-6.35300285232	2.197229523	0.002 and 0.004
		-4.18070472645	-6.35299642894	num. err.	0.0005
Grand total		1.7547700639999997			0.002

Table 4: NDDW contribution of the $q = L$ point at $k = \Gamma$ computed using finite difference.

3.3.4 AHC

We will now compute the contribution of each mode of the q-point L on the eigenenergies at Γ . The Abinit input file is given in annex 5.8. In order to obtain the contribution of the different mode we need to hack Abinit. Inside the routine `72_response/elph2_fanddw.F90` just comment the loop over mode line 95 and explicitly add a line specifying the mode in question. This gives table 5.

3.3.5 Summary

We can summarize our result in table 6

3.3.6 Richardson extrapolation

As we can see the results are not satisfactory. We need to push the convergence even further. The Richardson extrapolation allow one to have an estimation of the value of the derivative from values of the displacement h in geometrical progression.

Table 5: Contribution of the q-point L on the electronic eigenenergies at Γ using the AHC formalism. The two modes TA and TO are doubly degenerate.

Mode	ω [Ha]	ZPM [meV]
LA	0.00480084	-11.543939432856
LO	0.005785943	49.91955422733
TA	0.002453274	-5.0605389924462
TO	0.005671174	74.9525190767118
Total		178.159574963005

Table 6: Consistency check for Diamond.

Mode	ω [Ha]	NDDW [meV]	AHC [meV]	NDDW+AHC	Eigen en. [meV]
LA	0.00480084	2.900634869	-11.543939432856	-8.64330456	-8.64119034
LO	0.005785943	-0.427480911	49.91955422733	49.49207331	49.54602351
TA	0.002453274	-2.556421470	-5.0605389924462	-7.61696046	-7.61042338
TO	0.005671174	2.197229523	74.9525190767118	77.14974859	77.243804642
Total		1.754770063999	178.159574963	179.914345027	180.171595691

Let us call D the second derivative using centred finite difference and $D_{i,0} = D(h/2^i)$. It can be shown by Taylor expanding each term that if we use the following relationship:

$$D_{i,k} = \frac{D_{i,k-1} - \frac{1}{2^{2k}} D_{i-1,k-1}}{1 - \frac{1}{2^{2k}}} \quad (112)$$

then for each iteration we increase the order (accuracy) of the derivative. We obtain table 7, 8 and 9. The two scripts used are given in annex 5.9 and 5.10.

Taking each time the best order available we get table 10

Table 7: Richardson extrapolation for finite difference on eigenenergies

i	$D_{i,0}$	$D_{i,1}$	$D_{i,2}$	$D_{i,3}$	$D_{i,4}$	$D_{i,5}$
LA mode						
0	-8.63914325					
1	-8.64085195	-8.64142152				
2	-8.64146356	-8.64166743	-8.64168383			
3	-8.64221632	-8.64246724	-8.64252056	-8.64253384		
4	-8.64478204	-8.64563728	-8.64584861	-8.64590144	-8.64591465	
5	-8.65515632	-8.65861441	-8.65947955	-8.65969592	-8.65975001	-8.65976354
LO mode						
0	49.13417611					
1	49.44254196	49.54533058				
2	49.52090862	49.54703084	49.54714419			
3	49.54056016	49.54711068	49.5471116	49.54711555		
4	49.54545543	49.54708718	49.54708561	49.54708513	49.54708501	
5	49.54662999	49.54702151	49.54701713	49.54701604	49.54701577	49.5470157
TA mode						
0	-7.59542176					
1	-7.60690455	-7.61073214				
2	-7.61006497	-7.61111844	-7.6111442			
3	-7.61199254	-7.61263506	-7.61273617	-7.61276144		
4	-7.61704745	-7.61873243	-7.61913892	-7.61924055	-7.61926596	
5	-7.63667783	-7.64322129	-7.64485388	-7.64526206	-7.6453641	-7.64538961
TO mode						
0	76.20416222					
1	76.97939365	77.23780413				
2	77.17849665	77.24486431	77.24533499			
3	77.22843547	77.24508174	77.24509623	77.24509244		
4	77.24020582	77.24412927	77.24406577	77.24404941	77.24404532	
5	77.24020986	77.2402112	77.23995	77.23988467	77.23986834	77.23986426
	$\mathcal{O}(h^2)$	$\mathcal{O}(h^4)$	$\mathcal{O}(h^6)$	$\mathcal{O}(h^8)$	$\mathcal{O}(h^{10})$	$\mathcal{O}(h^{12})$

Table 8: Richardson extrapolation for the first NDDW term

i	$D_{i,0}$	$D_{i,1}$	$D_{i,2}$	$D_{i,3}$
LA mode				
0	-4.59977314			
1	-4.60234996	-4.60320889		
2	-4.60318005	-4.60345675	-4.60347328	
3	-4.60398656	-4.60425539	-4.60430864	-4.6043219
LO mode				
0	-6.65345629			
1	-6.65412452	-6.65434727		
2	-6.65429671	-6.6543541	-6.65435456	
3	-6.65436109	-6.65438255	-6.65438444	-6.65438492
TA mode				
0	-17.2251852			
1	-17.23136797	-17.2334289		
2	-17.23320069	-17.2338116	-17.23383711	
3	-17.23479542	-17.235327	-17.23542802	-17.23545328
TO mode				
0	-4.1510386			
1	-4.15369455	-4.15457986		
2	-4.15440484	-4.15464161	-4.15464572	
3	-4.1547653	-4.15488546	-4.15490171	-4.15490578
	$\mathcal{O}(h^2)$	$\mathcal{O}(h^4)$	$\mathcal{O}(h^6)$	$\mathcal{O}(h^8)$

Table 9: Richardson extrapolation for the second NDDW term

i	$D_{i,0}$	$D_{i,1}$	$D_{i,2}$	$D_{i,3}$	$D_{i,4}$	$D_{i,5}$
LA mode						
0	-7.50245559					
1	-7.50416521	-7.50473508				
2	-7.50459329	-7.50473598	-7.50473604			
3	-7.50470183	-7.50473801	-7.50473815	-7.50473818		
4	-7.50473206	-7.50474214	-7.50474242	-7.50474248	-7.5047425	
5	-7.50474557	-7.50475008	-7.50475061	-7.50475074	-7.50475077	-7.50475078
LO mode						
0	-6.22508988					
1	-6.2265084	-6.22698123				
2	-6.22686364	-6.22698206	-6.22698211			
3	-6.22695361	-6.2269836	-6.2269837	-6.22698373		
4	-6.22697888	-6.2269873	-6.22698755	-6.22698761	-6.22698763	
5	-6.22699061	-6.22699452	-6.226995	-6.22699512	-6.22699515	-6.22699516
TA mode						
0	-14.68184134					
1	-14.6850326	-14.68609636				
2	-14.68583414	-14.68610132	-14.68610165			
3	-14.68604245	-14.68611188	-14.68611258	-14.68611276		
4	-14.68611201	-14.6861352	-14.68613676	-14.68613714	-14.68613724	
5	-14.68616087	-14.68617715	-14.68617995	-14.68618064	-14.68618081	-14.68618085
TO mode						
0	-6.35116095					
1	-6.35253914	-6.35299854				
2	-6.35288164	-6.35299581	-6.35299563			
3	-6.35296354	-6.35299084	-6.35299051	-6.35299043		
4	-6.35297785	-6.35298262	-6.35298207	-6.35298194	-6.3529819	
5	-6.35296141	-6.35295593	-6.35295415	-6.35295371	-6.3529536	-6.35295357
	$\mathcal{O}(h^2)$	$\mathcal{O}(h^4)$	$\mathcal{O}(h^6)$	$\mathcal{O}(h^8)$	$\mathcal{O}(h^{10})$	$\mathcal{O}(h^{12})$

Table 10: Consistency check for Diamond.

Mode	ω [Ha]	NDDW [meV]	AHC [meV]	NDDW+AHC	Eigen en. [meV]
LA	0.00480084	2.90042888	-11.543939432856	-8.643510552856	-8.65976354
LO	0.005785943	-0.42738976	49.91955422733	49.49216446733	49.5470157
TA	0.002453274	-2.54927243	-5.0605389924462	-7.6098114224462	-7.64538961
TO	0.005671174	2.19804779	74.9525190767118	77.1505668667118	77.23986426
Total		1.77058984	178.159574963005	179.930164803005	180.07620146

4 Point $\frac{2L}{3}$

We choose this point because it gives us the smallest supercell with imaginary component for the displacement eigenvectors.

For the moment it is not possible to perform finite difference on the occupation using Abinit. It is not possible to change the occupation of a q point different from 0.

4.1 AHC

Solving the dynamical equation $\mathbf{D}\vec{\xi} = \omega^2\vec{\xi}$ gives us the following eigenvectors:

$$\begin{aligned}
\xi_{\text{LA}} &= [(0.4082483, -0.4082483, -0.4082483) \\
&\quad (0.3556 + 0.20058j, -0.3556 - 0.20058j, -0.3556 - 0.20058j)] \\
\xi_{\text{LO}} &= [(-4082483, 0.4082483, 0.4082483) \\
&\quad (0.3556 + 0.20058j, -0.3556 - 0.20058j, -0.3556 - 0.20058j)] \\
\xi_{\text{TA1}} &= [(0.5463285 - 0.10296j, 0.4026163 - 0.08936j, 0.1437122 - 0.013605j) \\
&\quad (0.5559466, 0.4122009 - 0.013246j, 0.1437457 + 0.013246j)] \\
\xi_{\text{TA2}} &= [(0.1729278 - 0.063874j, -0.3817674 + 0.0406679j, 0.55469523 - 0.10454196j) \\
&\quad (0.18176599 - 0.03074164j, -0.38269466 - 0.03074164j, 0.56446065)] \\
\xi_{\text{TO1}} &= [(-0.53400085 + 0.10064174j, -0.4341383 + 0.07483009j, -0.0998626 + 0.02581166j) \\
&\quad (0.54340194, 0.44048649 + 0.00686987j, 0.10291544 - 0.00686988j)] \\
\xi_{\text{TO2}} &= [(0.19352971 + 0.01834087j, -0.37395091 + 0.01834086j, 0.56748057) \\
&\quad (-0.18678471 - 0.05386659j, 0.37087825 + 0.05123471j, -0.55766291 - 0.10510126j)]
\end{aligned}$$

We used an unshifted 3x3x3 k point grid and an imaginary small parameter of 0.1eV.

4.2 Finite difference on eigenenergies

We have to solve the following equation:

$$\delta\varepsilon_{\Gamma n}(T, V = cst) = \frac{1}{4\omega_{qj}} \sum_{\kappa\alpha\kappa'\beta} \frac{\partial^2\varepsilon}{\partial R_{\kappa\alpha}\partial R_{\kappa'\beta}} \frac{\xi_{\kappa\alpha}^*(qj)\xi_{\kappa'\beta}}{\sqrt{M_{\kappa}M_{\kappa'}}} \quad (113)$$

$$\begin{aligned}
&= \frac{1}{4\omega_{qj}M_c} \sum_{\kappa\alpha\kappa'\beta} \frac{\partial^2\varepsilon}{\partial R_{\kappa\alpha}\partial R_{\kappa'\beta}} [(\xi_{\kappa\alpha}^r - i\xi_{\kappa\alpha}^i)(\xi_{\kappa'\beta}^r + i\xi_{\kappa'\beta}^i)(\cos(q \cdot (R_{\kappa'} - R_{\kappa})) + i\sin(q \cdot (R_{\kappa'} - R_{\kappa}))) \\
&\quad (114)
\end{aligned}$$

with $\xi_{\kappa\alpha}^r$ and $\xi_{\kappa\alpha}^i$ respectively the real and imaginary part of the eigendisplacement.

If we define $U_{\kappa\alpha} = \xi_{\kappa\alpha} e^{iq \cdot R_{\kappa}}$ the equation becomes:

$$\delta\varepsilon_{\Gamma n}(T, V = cst) = \frac{1}{4\omega_{qj}M_c} \sum_{\kappa\alpha\kappa'\beta} \frac{\partial^2\varepsilon}{\partial R_{\kappa\alpha}\partial R_{\kappa'\beta}} U_{\kappa\alpha}^* U_{\kappa'\beta} \quad (115)$$

$$\begin{aligned}
&= \frac{1}{4\omega_{qj}M_c} \sum_{\kappa\alpha\kappa'\beta} \frac{\partial^2\varepsilon}{\partial R_{\kappa\alpha}\partial R_{\kappa'\beta}} [U_{\kappa\alpha}^r U_{\kappa'\beta}^r + U_{\kappa\alpha}^i U_{\kappa'\beta}^i \\
&\quad + i(U_{\kappa\alpha}^r U_{\kappa'\beta}^i - U_{\kappa\alpha}^i U_{\kappa'\beta}^r)] \quad (116)
\end{aligned}$$

$$U_{\kappa\alpha}^r = U_{\kappa\alpha}^r \cos(q \cdot R_{\kappa}) - U_{\kappa\alpha}^i \sin(q \cdot R_{\kappa}) \quad (117)$$

$$U_{\kappa\alpha}^i = U_{\kappa\alpha}^i \cos(q \cdot R_{\kappa}) + U_{\kappa\alpha}^r \sin(q \cdot R_{\kappa}) \quad (118)$$

The imaginary part of equation (116) vannah when we do the full summation over $\kappa\alpha\kappa'\beta$.

The frequency are obtained from the secular equation:

$$\omega_m^2 = \sum_{\kappa\alpha\kappa'\beta} \frac{1}{M_c} \xi_{\kappa\alpha}^* \frac{\partial^2 E}{\partial R_{\kappa\alpha} \partial R_{\kappa'\beta}} \xi_{\kappa'\beta} \quad (119)$$

$$= \sum_{\kappa\alpha\kappa'\beta} \frac{1}{M_c} \frac{\partial^2 E}{\partial R_{\kappa\alpha} \partial R_{\kappa'\beta}} (\xi_{\kappa\alpha}^r - i\xi_{\kappa\alpha}^i)(\xi_{\kappa'\beta}^r + i\xi_{\kappa'\beta}^i) \quad (120)$$

$$= \sum_{\kappa\alpha\kappa'\beta} \frac{1}{M_c} \frac{\partial^2 E}{\partial R_{\kappa\alpha} \partial R_{\kappa'\beta}} (\xi_{\kappa\alpha}^r \xi_{\kappa'\beta}^r + \xi_{\kappa\alpha}^i \xi_{\kappa'\beta}^i + i(\xi_{\kappa\alpha}^r \xi_{\kappa'\beta}^i - \xi_{\kappa\alpha}^i \xi_{\kappa'\beta}^r)) \quad (121)$$

4.3 NDDW

$$\delta\varepsilon_{\Gamma n}^{NDDW}(T, V = cst) = \frac{1}{4\omega_{qj}M_c} \sum_{\kappa\alpha\kappa'\beta} \langle \psi_{\vec{k}n}^0 | \frac{\partial^2 H}{\partial R_{\kappa\alpha} \partial R_{\kappa'\beta}} | \psi_{\vec{k}n}^0 \rangle \left[\xi_{\kappa\alpha}(qj)\xi_{\kappa'\beta}(-qj)e^{iq\cdot(R_{\kappa'}-R_{\kappa})} - \frac{1}{2}(\xi_{\kappa\alpha}(qj)\xi_{\kappa\beta}(qj) + \xi_{\kappa'\alpha}(qj)\xi_{\kappa'\beta}) \right] \quad (122)$$

The first term is computed in exactly the same way as for the finite difference on eigenenergy using a supercell. The only difference being that we compute the second order derivative of the V_{HXC} instead of the eigenenergies. The second part of the NDDW is computed on a primitive cell as there is no phase factor. The second term is divide in two part. One for κ and one for κ' . As we perform a sum over those two indicies we have that they are equivalent. We can then only compute one of the two and take twice the result.

If we define two new variables we can rewrite the equation. Let us start by defining a collective displacement of all atoms as:

$$\frac{\partial}{\partial T_\beta} \equiv \sum_{\kappa'} \frac{\partial}{\partial R_{\kappa'\beta}} \quad (123)$$

$$(S_\beta)_{\kappa\alpha} = \xi_{\kappa\alpha} \xi_{\kappa\beta} \quad (124)$$

$$\frac{\partial}{\partial S_\beta} \equiv \sum_{\kappa\alpha} (S_\beta)_{\kappa\alpha} \frac{\partial}{\partial R_{\kappa\alpha}} \quad (125)$$

Then the equation reduces itself to:

$$\sum_{\kappa\alpha\kappa'\beta} \xi_{\kappa\alpha} \frac{\partial^2 V_{Hxc}}{\partial R_{\kappa\alpha} \partial R_{\kappa'\beta}} \xi_{\kappa\beta} = \sum_{\beta} \frac{\partial^2 V_{Hxc}}{\partial S_\beta \partial T_\beta} \quad (126)$$

4.4 Summary

The 4 scripts needed to produce the results are given in annex 5.11,5.12,5.13 and 5.14 and give the following results:

Table 11: Consistency check for Diamond at the q=2/3L point.

Eigenenergies				
Mode	HOMO [meV]	LUMO [meV]	ω [Ha]	
LA	-2.33314397	-11.53194289	0.00391312	
LO	54.44206791	-15.33013152	0.00608781	
TA	-4.24130494	-4.96811934	0.0021616	
TO	91.97849436	-122.06663082	0.00588964	
Total	227.58330278	-280.93157473		
AHC				
	HOMO [meV]	LUMO [meV]	ω [Ha]	
LA	-3.777607	-12.4774635	0.0039156	
LO	53.7276709	-15.1947408	0.006088662	
TA	-2.2650791	-3.2132302	0.0021637	
TO	90.0877978	-122.4889228	0.005890923	
Total	225.5955013	-279.0765103		
NDDW				
	1st term		2st term	
	HOMO [meV]	LUMO [meV]	HOMO [meV]	LUMO [meV]
LA	-7.82479036	-4.42053004	-9.5495477	-5.54256536
LO	-5.32793379	-3.57859453	-5.956124785	-3.456939727
TA	-18.73709541	-11.46457077	-16.607838981	-9.63920309
TO	-4.43823861	-2.97156954	-6.156047497	-3.572974741
Total	-59.50339219	-36.87140519	-61.03344544	-35.42386076
Diff.				
	HOMO [meV]	LUMO [meV]	ω [Ha]	
LA	-0.28029	-0.1765147	2.48E-6	
LO	0.086206	-0.01373591	8.52E-7	
TA	0.1530305	0.0704785	2.1E-6	
TO	0.1728876	-0.1791132	-3.005E-3	
Total	0.4577482	-0.407520003		

5 Annex

5.1 ABINIT input file

```
1 # C in diamond structure; 2x2x2 FCC special point grid; low ecut.
2 # psp 6c.pspnc
3 # Mcarbne 21894.16693
4 ndtset 2
5
6 #Dataset 1 : ground state density
7 nqpt1 0
8 ieig2rf1 0
9 smdelta1 0
10 rfphon1 0
11 getwfk1 0 # Use GS wave functions from dataset1
12
13 nqpt 1
14 ieig2rf 1
15 bdeigrf 8
16 smdelta 1
17 getwfk 1 # Use GS wave functions from dataset1
18 kptopt 3 # Need full k-point set for finite-Q response
19 rfphon 1 # Do phonon response
20 rfatpol 1 2 # Treat displacements of all atoms
21 rfdir 1 1 1 # Do all directions (symmetry will be used)
22
23 #Dataset 2-5 : phonon frequencies and band corrections
24 qpt2 0.0 0.0 0.0
25 rfasr2 1
26
27 #Size-dependent parameters
28 acell 3*6.70346805
29 rprim 0.5 .5 .5 0 .5 .5 0
30
31 natom 2
32 typat 1 1
33 xred 3*0.00d0 3*0.25d0
34
35 nband 32*8
36
37 ngkpt 2 2 2
38 nshiftk 4
39 shiftk 0.0 0.0 0.0
40 0.0 0.5 0.5
41 0.5 0.0 0.5
42 0.5 0.5 0.0
43 occopt 2
44 occ 4*2.0 4*0.0 #1
45 4*2.0 4*0.0 #2
46 4*2.0 4*0.0 #3
47 ...
48 4*2.0 4*0.0 #18
49 4*2.0 4*0.0 #19
50 # 4*2.0 3*0.02 0.0 # Gamma point LUMO 3x
51 2.0 3*1.99 4*0.0 # Gamma point HOMO 3x
52 4*2.0 4*0.0
53 4*2.0 4*0.0
54 ...
55 4*2.0 4*0.0
56 # charge -0.0009375 #for 0.01 LUMO 3x
57 charge 0.0009375 #for 0.01 HOMO 3x
58
59 # Miscellaneous
60 ntypat 1
61 znucl 6
62 diemac 6.0d0
63 ecut 30
64 enunit 2
65 nstep 40
66 nsym 1
67 tolwfr 1.0d-18
```

Code 1: ABINIT input file

```
1 # C in diamond structure; 2x2x2 FCC special point grid; low ecut.
2
3 ndtset 9
4
5 getwfk -1 # Use GS wave functions from dataset1
```

```

6 kptopt 3 # Need full k-point set for finite-Q response
7
8
9 #Size-dependent parameters
10 acell 3*6.70346805
11 rprim 0 .5 .5 .5 0 .5 .5 0
12
13 natom 2
14 typat 1 1
15
16 xcart1 3*0.0 3*1.675867013 # f(x1,x2)
17 xcart2 3*0.01 3*1.675867013 # f(x1+h,x2)
18 xcart3 3*-0.01 3*1.675867013 # f(x1-h,x2)
19 xcart4 3*0.0 3*1.685867013 # f(x1,x2+w)
20 xcart5 3*0.0 3*1.665867013 # f(x1,x2-w)
21 xcart6 3*0.01 3*1.685867013 # f(x1+h,x2+w)
22 xcart7 3*-0.01 3*1.665867013 # f(x1-h,x2-w)
23 xcart8 3*0.01 3*1.665867013 # f(x1+h,x2-w)
24 xcart9 3*-0.01 3*1.685867013 # f(x1-h,x2+w)
25
26 nband 8
27
28 ngkpt 2 2 2
29 nshiftk 4
30 shiftk 0.0 0.0 0.0
31 0.0 0.5 0.5
32 0.5 0.0 0.5
33 0.5 0.5 0.0
34
35
36 # Miscellaneous
37 ntypat 1
38 znuc1 6
39 diemac 6.0d0
40 ecut 30
41 enunit 2
42 nstep 40
43 nsym 1
44 tolwfr 1.0d-18

```

Code 2: ABINIT input file

```

1 # C in diamond structure; 2x2x2 FCC special point grid; low ecut.
2
3 ndtset 2
4
5
6 #Dataset 1 : ground state density
7 nqpt1 0
8 ieig2rf1 0
9 smdelta1 0
10 rfphon1 0
11 getwfk1 0 # Use GS wave functions from dataset1
12 #ecutsm1 0.5
13
14 nqpt 1
15 ieig2rf 1
16 bdeigrf 8
17 smdelta 1
18 getwfk 1 # Use GS wave functions from dataset1
19 kptopt 3 # Need full k-point set for finite-Q response
20 rfphon 1 # Do phonon response
21 rfatpol 1 2 # Treat displacements of all atoms
22 rfdir 1 1 1 # Do all directions (symmetry will be used)
23
24 #Dataset 2-5 : phonon frequencies and band corrections
25 qpt2 0.0 0.0 0.0
26
27 #Size-dependent parameters
28 acell 3*6.70346805 #relaxed param.
29 rprim 0 .5 .5 .5 0 .5 .5 0
30
31 natom 2
32 typat 1 1
33 xred 3*0.00d0 3*0.25d0
34
35 nband 8
36
37 ngkpt 2 2 2
38 nshiftk 4
39 shiftk 0.0 0.0 0.0
40 0.0 0.5 0.5

```

```

41      0.5 0.0 0.5
42      0.5 0.5 0.0
43
44 # Miscellaneous
45 ntypat 1
46 znuc1 6
47 diemac 6.0d0
48 ecut 30
49 enunit 2
50 nstep 30
51 nsym 1
52 tolwfr 1.0d-16

```

Code 3: ABINIT input file

```

1 !Input file for the anaddd code. Analysis of the Diamond DDB
2
3 !Flags
4 ifcflag 1 ! Interatomic force constant flag
5 thmflag 3
6 telphint 1
7 ntemper 10
8 tempermin 100
9 temperinc 100
10 !Wavevector grid number 1 (coarse grid, from DDB)
11 brav 1 ! Bravais Lattice : 1-S.C., 2-F.C., 3-B.C., 4-Hex.)
12 ngqpt 1 1 1 ! Monkhorst-Pack indices
13 nqshft 1 ! number of q-points in repeated basic q-cell
14 q1shft 3*0.0
15      0.0 0.5 0.5
16      0.5 0.0 0.5
17      0.5 0.5 0.0
18
19 !Effective charges
20 asr 1 ! Acoustic Sum Rule. 1 => imposed asymmetrically
21 chneut 1 ! Charge neutrality requirement for effective charges.
22
23 !Interatomic force constant info
24 dipdip 0 ! Dipole-dipole interaction treatment
25
26 !Wavevector list number 1 (Reduced coordinates and normalization factor)
27 nph11 1 ! number of phonons in list 1
28 qph11 0.00000000E+00 0.00000000E+00 0.00000000E+00 1.0

```

Code 4: ANADDB input file

5.2 Dynamical matrix

1	1	1	1	1	0.4875111415	0.0000000000
2	1	1	2	1	-0.0684816112	0.0000000000
3	1	1	3	1	-0.0684816108	0.0000000000
4	1	1	1	2	-0.2259726921	0.0000000000
5	1	1	2	2	-0.0580165771	0.0000000000
6	1	1	3	2	-0.0580165772	0.0000000000
7						
8	2	1	1	1	-0.0684816113	0.0000000000
9	2	1	2	1	0.4875111415	0.0000000000
10	2	1	3	1	0.0684816108	0.0000000000
11	2	1	1	2	-0.0580165771	0.0000000000
12	2	1	2	2	-0.2259726919	0.0000000000
13	2	1	3	2	0.0580165774	0.0000000000
14						
15	3	1	1	1	-0.0684816109	0.0000000000
16	3	1	2	1	0.0684816109	0.0000000000
17	3	1	3	1	0.4875111412	0.0000000000
18	3	1	1	2	-0.0580165772	0.0000000000
19	3	1	2	2	0.0580165773	0.0000000000
20	3	1	3	2	-0.2259726918	0.0000000000
21						
22	1	2	1	1	-0.2259726919	0.0000000000
23	1	2	2	1	-0.0580165771	0.0000000000
24	1	2	3	1	-0.0580165772	0.0000000000
25	1	2	1	2	0.4875113661	0.0000000000
26	1	2	2	2	-0.0684816112	0.0000000000
27	1	2	3	2	-0.0684816109	0.0000000000
28						
29	2	2	1	1	-0.0580165773	0.0000000000
30	2	2	2	1	-0.2259726919	0.0000000000

31	2	2	3	1	0.0580165773	0.0000000000
32	2	2	1	2	-0.0684816111	0.0000000000
33	2	2	2	2	0.4875113663	0.0000000000
34	2	2	3	2	0.0684816109	0.0000000000
35						
36	3	2	1	1	-0.0580165774	0.0000000000
37	3	2	2	1	0.0580165774	0.0000000000
38	3	2	3	1	-0.2259726918	0.0000000000
39	3	2	1	2	-0.0684816108	0.0000000000
40	3	2	2	2	0.0684816109	0.0000000000
41	3	2	3	2	0.4875113666	0.0000000000

Code 5: Dynamical matrix (Fourier transform of the interatomic force constant) obtained in DFPT by ABINIT

5.3 Python script for solving the eigenvalue problem.

```

1  # Find the eigenvector from the dynamical matrice
2
3  import numpy as np
4
5  def read_dynamical_matrix(datafile):
6      with open(datafile,'r') as f:
7          raw = list()
8          for line in f:
9              if line[0]=='#': continue
10             parts=line.split()
11             if not parts: continue
12             val=float(parts[4])
13             if np.abs(val) < 0.000000001:
14                 val=0.0
15             raw.append(val)
16             dynmat = np.array(raw)
17             dynmat = dynmat.reshape(6,6)
18             return dynmat
19
20
21  L_file = '/home/Samuel/Dropbox/WorkDiam/DerivEig_L/Dyanmical_mat.dat'
22
23  dynmatL=read_dynamical_matrix(L_file)
24
25  [eigvalL,eigvectL]=np.linalg.eig(dynmatL)
26
27  eigvectL=np.transpose(eigvectL)
28  Mc=21894.16693
29
30  print "eigenvalues at L : "
31  for n in eigvalL:
32      a = np.sqrt(n/Mc)
33      print a
34
35  print "eigenvectors at L : "
36  for n in eigvectL: print n
37
38  # Create the xcart needed for abinit input file
39
40  disp = 0.03
41  rprim = np.array([13.4069361,6.70346805,6.70346805])
42
43
44  print "# Equilibrium position"
45  print " xcart1 0.0 0.0 0.0 # 0.0 0.0 0.0 "
46  print "      1.67586701 1.67586701 1.67586701 # 0.125 0.25 0.25"
47  print "      6.70346805 0.00 0.00 # 0.5 0.0 0.0 "
48  print "      8.37933506 1.67586701 1.67586701 # 0.625 0.25 0.25 "
49  print " "
50  print "# LA mode"
51  i = 0
52  print " xcart2 %.10f %.10f %.10f"%(rprim[0]*0.0+eigvectL[i][0]*disp,
53  rprim[1]*0.0+eigvectL[i][1]*disp,rprim[2]*0.0+eigvectL[i][2]*disp)
54  print "      %.10f %.10f %.10f"%(rprim[0]*0.125+eigvectL[i][3]*disp,
55  rprim[1]*0.25+eigvectL[i][4]*disp,rprim[2]*0.25+eigvectL[i][5]*disp)
56  print "      %.10f %.10f %.10f"%(rprim[0]*0.5-eigvectL[i][0]*disp,
57  rprim[1]*0.0-eigvectL[i][1]*disp,rprim[2]*0.0-eigvectL[i][2]*disp)
58  print "      %.10f %.10f %.10f"%(rprim[0]*0.625-eigvectL[i][3]*disp,
59  rprim[1]*0.25-eigvectL[i][4]*disp,rprim[2]*0.25-eigvectL[i][5]*disp)
60  print " "
61  print "# LO mode "
62  i = 1

```

```

63 print " xcart3 %.10f %.10f %.10f"%(rprim[0]*0.0+eigvectL[i][0]*disp,
64 rprim[1]*0.0+eigvectL[i][1]*disp,rprim[2]*0.0+eigvectL[i][2]*disp)
65 print " %.10f %.10f %.10f"%(rprim[0]*0.125+eigvectL[i][3]*disp,
66 rprim[1]*0.25+eigvectL[i][4]*disp,rprim[2]*0.25+eigvectL[i][5]*disp)
67 print " %.10f %.10f %.10f"%(rprim[0]*0.5-eigvectL[i][0]*disp,
68 rprim[1]*0.0-eigvectL[i][1]*disp,rprim[2]*0.0-eigvectL[i][2]*disp)
69 print " %.10f %.10f %.10f"%(rprim[0]*0.625-eigvectL[i][3]*disp,
70 rprim[1]*0.25-eigvectL[i][4]*disp,rprim[2]*0.25-eigvectL[i][5]*disp)
71 print ""
72 print "# TA mode "
73 i = 2
74 ...

```

Code 6: Python script for solving the eigenvalue problem.

5.4 NDDW using finite difference at the q=L point

```

1 # Find the eigenvector from the dynamical matrice
2
3 import numpy as np
4
5 def read_dynamical_matrix(datafile):
6     with open(datafile,'r') as f:
7         raw = list()
8         for line in f:
9             if line[0]!='#': continue
10            parts=line.split()
11            if not parts: continue
12            val=float(parts[4])
13            if np.abs(val) < 0.000000001:
14                val=0.0
15            raw.append(val)
16        dynmat = np.array(raw)
17        dynmat = dynmat.reshape(6,6)
18        return dynmat
19
20
21 L_file = '/home/Samuel/Dropbox/WorkDiam/DerivEig_L/Dyanmical_mat.dat'
22
23 dynmatL=read_dynamical_matrix(L_file)
24
25 [eigvall,eigvectL]=np.linalg.eig(dynmatL)
26
27 eigvectL=np.transpose(eigvectL)
28 Mc=21894.16693
29
30 print "eigenvalues at L :"
31 for n in eigvall:
32     a = np.sqrt(n/Mc)
33     print a
34
35 print "eigenvectors at L :"
36 for n in eigvectL: print n
37
38 # Create the xcart needed for abinit input file
39
40 disp = 0.01
41 acell = np.array([13.4069361,6.70346805,6.70346805])
42 rprim = np.array([[0.0,0.5,0.5],[0.5,0.0,0.5],[0.5,0.5,0.0]])
43
44 cell = np.mat([rprim[0]*acell[0],rprim[1]*acell[1],rprim[2]*acell[2]])
45 print "Cell geometry: ", cell
46
47 #Reduce coordinate of atoms
48
49 xred = np.mat([[0.0,0.0,0.0],[0.125,0.25,0.25],[0.5,0.0,0.0],[0.625,0.25,0.25]])
50 print xred
51 print (xred[1]*cell)[0,0]
52
53
54 #Count for nb of xcart
55 j = 1
56
57 print "# Equilibrium position"
58 print " xcart%.0f %.10f %.10f %.10f"%(j,(xred[0]*cell)[0,0],(xred[0]*cell)[0,1],
59 (xred[0]*cell)[0,2])
60 print " %.10f %.10f %.10f "%((xred[1]*cell)[0,0],(xred[1]*cell)[0,1],(xred[1]*cell)[0,2])
61

```

```

62 print "          %.10f %.10f %.10f "%((xred[2]*cell)[0,0],(xred[2]*cell)[0,1],(xre
63 d[2]*cell)[0,2])
64 print "          %.10f %.10f %.10f "%((xred[3]*cell)[0,0],(xred[3]*cell)[0,1],(xre
65 d[3]*cell)[0,2])
66
67 j += 1
68
69 # LA mode (+h)
70 i = 0
71 print " xcart%.0f %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]+eigvectL[i][0]*disp
72 ,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,(xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
73 print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]+eigvectL[i][3]*disp,(xre
74 d[1]*cell)[0,1]+eigvectL[i][4]*disp,(xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
75 print "          %.10f %.10f %.10f "%((xred[2]*cell)[0,0]-eigvectL[i][0]*disp,(xre
76 d[2]*cell)[0,1]-eigvectL[i][1]*disp,(xred[2]*cell)[0,2]-eigvectL[i][2]*disp)
77 print "          %.10f %.10f %.10f "%((xred[3]*cell)[0,0]-eigvectL[i][3]*disp,(xre
78 d[3]*cell)[0,1]-eigvectL[i][4]*disp,(xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
79 j += 1
80
81 # LA mode (-h)
82 print " xcart%.0f %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]-eigvectL[i][0]*disp
83 ,(xred[0]*cell)[0,1]-eigvectL[i][1]*disp,(xred[0]*cell)[0,2]-eigvectL[i][2]*disp)
84 print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]-eigvectL[i][3]*disp,(xre
85 d[1]*cell)[0,1]-eigvectL[i][4]*disp,(xred[1]*cell)[0,2]-eigvectL[i][5]*disp)
86 print "          %.10f %.10f %.10f "%((xred[2]*cell)[0,0]+eigvectL[i][0]*disp,(xre
87 d[2]*cell)[0,1]+eigvectL[i][1]*disp,(xred[2]*cell)[0,2]+eigvectL[i][2]*disp)
88 print "          %.10f %.10f %.10f "%((xred[3]*cell)[0,0]+eigvectL[i][3]*disp,(xre
89 d[3]*cell)[0,1]+eigvectL[i][4]*disp,(xred[3]*cell)[0,2]+eigvectL[i][5]*disp)
90 j += 1
91
92 # L0 mode (+h)
93 i = 1
94 print " xcart%.0f %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]+eigvectL[i][0]*disp
95 ,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,(xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
96 print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]+eigvectL[i][3]*disp,(xre
97 d[1]*cell)[0,1]+eigvectL[i][4]*disp,(xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
98 print "          %.10f %.10f %.10f "%((xred[2]*cell)[0,0]-eigvectL[i][0]*disp,(xre
99 d[2]*cell)[0,1]-eigvectL[i][1]*disp,(xred[2]*cell)[0,2]-eigvectL[i][2]*disp)
100 print "          %.10f %.10f %.10f "%((xred[3]*cell)[0,0]-eigvectL[i][3]*disp,(xre
101 d[3]*cell)[0,1]-eigvectL[i][4]*disp,(xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
102 j += 1
103
104 # L0 mode (-h)
105 print " xcart%.0f %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]-eigvectL[i][0]*disp
106 ,(xred[0]*cell)[0,1]-eigvectL[i][1]*disp,(xred[0]*cell)[0,2]-eigvectL[i][2]*disp)
107 print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]-eigvectL[i][3]*disp,(xre
108 d[1]*cell)[0,1]-eigvectL[i][4]*disp,(xred[1]*cell)[0,2]-eigvectL[i][5]*disp)
109 print "          %.10f %.10f %.10f "%((xred[2]*cell)[0,0]+eigvectL[i][0]*disp,(xre
110 d[2]*cell)[0,1]+eigvectL[i][1]*disp,(xred[2]*cell)[0,2]+eigvectL[i][2]*disp)
111 print "          %.10f %.10f %.10f "%((xred[3]*cell)[0,0]+eigvectL[i][3]*disp,(xre
112 d[3]*cell)[0,1]+eigvectL[i][4]*disp,(xred[3]*cell)[0,2]+eigvectL[i][5]*disp)
113 j += 1
114
115 # TA mode (+h)
116 i = 2
117 print " xcart%.0f %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]+eigvectL[i][0]*disp
118 ,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,(xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
119 print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]+eigvectL[i][3]*disp,(xre
120 d[1]*cell)[0,1]+eigvectL[i][4]*disp,(xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
121 print "          %.10f %.10f %.10f "%((xred[2]*cell)[0,0]-eigvectL[i][0]*disp,(xre
122 d[2]*cell)[0,1]-eigvectL[i][1]*disp,(xred[2]*cell)[0,2]-eigvectL[i][2]*disp)
123 print "          %.10f %.10f %.10f "%((xred[3]*cell)[0,0]-eigvectL[i][3]*disp,(xre
124 d[3]*cell)[0,1]-eigvectL[i][4]*disp,(xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
125 j += 1
126
127 # TA mode (-h)
128 print " xcart%.0f %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]-eigvectL[i][0]*disp
129 ,(xred[0]*cell)[0,1]-eigvectL[i][1]*disp,(xred[0]*cell)[0,2]-eigvectL[i][2]*disp)
130 print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]-eigvectL[i][3]*disp,(xre
131 d[1]*cell)[0,1]-eigvectL[i][4]*disp,(xred[1]*cell)[0,2]-eigvectL[i][5]*disp)
132 print "          %.10f %.10f %.10f "%((xred[2]*cell)[0,0]+eigvectL[i][0]*disp,(xre
133 d[2]*cell)[0,1]+eigvectL[i][1]*disp,(xred[2]*cell)[0,2]+eigvectL[i][2]*disp)
134 print "          %.10f %.10f %.10f "%((xred[3]*cell)[0,0]+eigvectL[i][3]*disp,(xre
135 d[3]*cell)[0,1]+eigvectL[i][4]*disp,(xred[3]*cell)[0,2]+eigvectL[i][5]*disp)
136 j += 1
137
138 # TA mode (2ieme) (+h)
139 i = 3
140 print " xcart%.0f %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]+eigvectL[i][0]*disp
141 ,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,(xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
142 print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]+eigvectL[i][3]*disp,(xre
143 d[1]*cell)[0,1]+eigvectL[i][4]*disp,(xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
144 print "          %.10f %.10f %.10f "%((xred[2]*cell)[0,0]-eigvectL[i][0]*disp,(xre
145 d[2]*cell)[0,1]-eigvectL[i][1]*disp,(xred[2]*cell)[0,2]-eigvectL[i][2]*disp)

```

```

146 print "          %.10f %.10f %.10f "%((xred[3]*cell)[0,0]-eigvectL[i][3]*disp,(xre
147 d[3]*cell)[0,1]-eigvectL[i][4]*disp,(xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
148 j += 1
149
150 # TA mode (-h)
151 print " xcart%.0f %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]-eigvectL[i][0]*disp
152 ,(xred[0]*cell)[0,1]-eigvectL[i][1]*disp,(xred[0]*cell)[0,2]-eigvectL[i][2]*disp)
153 print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]-eigvectL[i][3]*disp,(xre
154 d[1]*cell)[0,1]-eigvectL[i][4]*disp,(xred[1]*cell)[0,2]-eigvectL[i][5]*disp)
155 print "          %.10f %.10f %.10f "%((xred[2]*cell)[0,0]-eigvectL[i][0]*disp,(xre
156 d[2]*cell)[0,1]-eigvectL[i][1]*disp,(xred[2]*cell)[0,2]-eigvectL[i][2]*disp)
157 print "          %.10f %.10f %.10f "%((xred[3]*cell)[0,0]-eigvectL[i][3]*disp,(xre
158 d[3]*cell)[0,1]-eigvectL[i][4]*disp,(xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
159 j += 1
160
161 # T0 mode (+h)
162 i = 4
163 print " xcart%.0f %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]+eigvectL[i][0]*disp
164 ,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,(xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
165 print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]+eigvectL[i][3]*disp,(xre
166 d[1]*cell)[0,1]+eigvectL[i][4]*disp,(xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
167 print "          %.10f %.10f %.10f "%((xred[2]*cell)[0,0]-eigvectL[i][0]*disp,(xre
168 d[2]*cell)[0,1]-eigvectL[i][1]*disp,(xred[2]*cell)[0,2]-eigvectL[i][2]*disp)
169 print "          %.10f %.10f %.10f "%((xred[3]*cell)[0,0]-eigvectL[i][3]*disp,(xre
170 d[3]*cell)[0,1]-eigvectL[i][4]*disp,(xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
171 j += 1
172
173 # T0 mode (-h)
174 print " xcart%.0f %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]-eigvectL[i][0]*disp
175 ,(xred[0]*cell)[0,1]-eigvectL[i][1]*disp,(xred[0]*cell)[0,2]-eigvectL[i][2]*disp)
176 print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]-eigvectL[i][3]*disp,(xre
177 d[1]*cell)[0,1]-eigvectL[i][4]*disp,(xred[1]*cell)[0,2]-eigvectL[i][5]*disp)
178 print "          %.10f %.10f %.10f "%((xred[2]*cell)[0,0]+eigvectL[i][0]*disp,(xre
179 d[2]*cell)[0,1]+eigvectL[i][1]*disp,(xred[2]*cell)[0,2]+eigvectL[i][2]*disp)
180 print "          %.10f %.10f %.10f "%((xred[3]*cell)[0,0]+eigvectL[i][3]*disp,(xre
181 d[3]*cell)[0,1]+eigvectL[i][4]*disp,(xred[3]*cell)[0,2]+eigvectL[i][5]*disp)
182 j += 1
183
184 # T0 mode (2ieme) (+h)
185 i = 5
186 print " xcart%.0f %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]+eigvectL[i][0]*disp
187 ,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,(xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
188 print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]+eigvectL[i][3]*disp,(xre
189 d[1]*cell)[0,1]+eigvectL[i][4]*disp,(xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
190 print "          %.10f %.10f %.10f "%((xred[2]*cell)[0,0]-eigvectL[i][0]*disp,(xre
191 d[2]*cell)[0,1]-eigvectL[i][1]*disp,(xred[2]*cell)[0,2]-eigvectL[i][2]*disp)
192 print "          %.10f %.10f %.10f "%((xred[3]*cell)[0,0]-eigvectL[i][3]*disp,(xre
193 d[3]*cell)[0,1]-eigvectL[i][4]*disp,(xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
194 j += 1
195
196 # T0 mode (-h)
197 print " xcart%.0f %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]-eigvectL[i][0]*disp
198 ,(xred[0]*cell)[0,1]-eigvectL[i][1]*disp,(xred[0]*cell)[0,2]-eigvectL[i][2]*disp)
199 print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]-eigvectL[i][3]*disp,(xre
200 d[1]*cell)[0,1]-eigvectL[i][4]*disp,(xred[1]*cell)[0,2]-eigvectL[i][5]*disp)
201 print "          %.10f %.10f %.10f "%((xred[2]*cell)[0,0]+eigvectL[i][0]*disp,(xre
202 d[2]*cell)[0,1]+eigvectL[i][1]*disp,(xred[2]*cell)[0,2]+eigvectL[i][2]*disp)
203 print "          %.10f %.10f %.10f "%((xred[3]*cell)[0,0]+eigvectL[i][3]*disp,(xre
204 d[3]*cell)[0,1]+eigvectL[i][4]*disp,(xred[3]*cell)[0,2]+eigvectL[i][5]*disp)
205 j += 1

```

Code 7: Python script used to create input files to solve the first part of the NDDW at q

5.5 NDDW using finite difference at the $q=L$ point

```

1 import math
2 import os
3 import numpy as np
4
5 # Transform binary VHXC file into text file
6
7 os.system("rm cut3d.files")
8 with open("cut3d.files","a") as files:
9     files.write("outputo_DS1_VHXC \n")
10    files.write("1 \n")
11    files.write("5 \n") #3D formatted data (output the bare 3D data - one column)
12    files.write("EQ \n")
13    files.write("0 \n")

```



```

14 | os.system("cut3d < cut3d.files ")
15 |
16 | os.system("rm cut3d.files")
17 | with open("cut3d.files","a") as files:
18 |     files.write("outputo_DS2_VHXC \n")
19 |     files.write("1 \n")
20 |     files.write("5 \n")
21 |     files.write("LA+ \n")
22 |     files.write("0 \n")
23 | os.system("cut3d < cut3d.files ")
24 |
25 | os.system("rm cut3d.files")
26 | with open("cut3d.files","a") as files:
27 |     files.write("outputo_DS3_VHXC \n")
28 |     files.write("1 \n")
29 |     files.write("5 \n")
30 |     files.write("LA- \n")
31 |     files.write("0 \n")
32 | os.system("cut3d < cut3d.files ")
33 |
34 | os.system("rm cut3d.files")
35 | with open("cut3d.files","a") as files:
36 |     files.write("outputo_DS4_VHXC \n")
37 |     files.write("1 \n")
38 |     files.write("5 \n")
39 |     files.write("LO+ \n")
40 |     files.write("0 \n")
41 | os.system("cut3d < cut3d.files ")
42 |
43 | os.system("rm cut3d.files")
44 | with open("cut3d.files","a") as files:
45 |     files.write("outputo_DS5_VHXC \n")
46 |     files.write("1 \n")
47 |     files.write("5 \n")
48 |     files.write("LO- \n")
49 |     files.write("0 \n")
50 | os.system("cut3d < cut3d.files ")
51 |
52 | os.system("rm cut3d.files")
53 | with open("cut3d.files","a") as files:
54 |     files.write("outputo_DS6_VHXC \n")
55 |     files.write("1 \n")
56 |     files.write("5 \n")
57 |     files.write("TA1+ \n")
58 |     files.write("0 \n")
59 | os.system("cut3d < cut3d.files ")
60 |
61 | os.system("rm cut3d.files")
62 | with open("cut3d.files","a") as files:
63 |     files.write("outputo_DS7_VHXC \n")
64 |     files.write("1 \n")
65 |     files.write("5 \n")
66 |     files.write("TA1- \n")
67 |     files.write("0 \n")
68 | os.system("cut3d < cut3d.files ")
69 |
70 | os.system("rm cut3d.files")
71 | with open("cut3d.files","a") as files:
72 |     files.write("outputo_DS8_VHXC \n")
73 |     files.write("1 \n")
74 |     files.write("5 \n")
75 |     files.write("TA2+ \n")
76 |     files.write("0 \n")
77 | os.system("cut3d < cut3d.files ")
78 |
79 | os.system("rm cut3d.files")
80 | with open("cut3d.files","a") as files:
81 |     files.write("outputo_DS9_VHXC \n")
82 |     files.write("1 \n")
83 |     files.write("5 \n")
84 |     files.write("TA2- \n")
85 |     files.write("0 \n")
86 | os.system("cut3d < cut3d.files ")
87 |
88 | os.system("rm cut3d.files")
89 | with open("cut3d.files","a") as files:
90 |     files.write("outputo_DS10_VHXC \n")
91 |     files.write("1 \n")
92 |     files.write("5 \n")
93 |     files.write("T01+ \n")
94 |     files.write("0 \n")
95 | os.system("cut3d < cut3d.files ")
96 |
97 | os.system("rm cut3d.files")

```

```

98 | with open("cut3d.files","a") as files:
99 |     files.write("outputo_DS11_VHXC \n")
100 |     files.write("1 \n")
101 |     files.write("5 \n")
102 |     files.write("T01- \n")
103 |     files.write("0 \n")
104 | os.system("cut3d < cut3d.files ")
105 |
106 | os.system("rm cut3d.files")
107 | with open("cut3d.files","a") as files:
108 |     files.write("outputo_DS12_VHXC \n")
109 |     files.write("1 \n")
110 |     files.write("5 \n")
111 |     files.write("T02- \n")
112 |     files.write("0 \n")
113 | os.system("cut3d < cut3d.files ")
114 |
115 | os.system("rm cut3d.files")
116 | with open("cut3d.files","a") as files:
117 |     files.write("outputo_DS13_VHXC \n")
118 |     files.write("1 \n")
119 |     files.write("5 \n")
120 |     files.write("T02- \n")
121 |     files.write("0 \n")
122 | os.system("cut3d < cut3d.files ")
123 |
124 | # -----
125 | # Make centered finite difference of order 2 of Vhxc for each mode
126 | # d2Vhxc/dR1dR2=(2*Vhxc(mode)-2*Vhxc(eq))/(h^2)
127 | # -----
128 |
129 | mode = np.array(["EQ","LA+","LA-","LO+","LO-","TA1+","TA1-","TA2+","TA2-","
130 | "T01+","T01-","T02+","T02-"])
131 |
132 | # The phonon frequency are eigenvalue of the Dynamical eigenvalue problem.
133 |
134 | omega = np.array([0.0,0.00480084,0.00480084,0.005785943,0.005785943,0.002453274,
135 | 0.002453274,0.002453274,0.002453274,0.005671174,0.005671174,0.005671174,0.005671174])
136 |
137 | # -----
138 | # Prepare the integration : de2/dR1dR2=int d2Vhxc/dR1dR2 psi*psi(unperturbed)
139 | # -----
140 |
141 | os.system("rm wfk.files")
142 | with open("wfk.files","a") as files:
143 |     files.write("outputo_DS1_WFK \n") #unperturbed wfk
144 |     files.write("1 \n")
145 |     files.write("0 \n")
146 |     files.write("4 \n") # Gamma point
147 |     files.write("6 \n") # Band nb 6
148 |     files.write("0 \n")
149 |     files.write("0 \n")
150 |     files.write("2 \n") # real 3D data one column
151 |     files.write("wfk \n")
152 |     files.write("0 \n")
153 | os.system("cut3d < wfk.files ")
154 |
155 | os.system("rm wfk.files")
156 | with open("wfk.files","a") as files:
157 |     files.write("outputo_DS1_WFK \n") #unperturbed wfk
158 |     files.write("1 \n")
159 |     files.write("0 \n")
160 |     files.write("4 \n") # Gamma point
161 |     files.write("7 \n") # Band nb 7
162 |     files.write("0 \n")
163 |     files.write("0 \n")
164 |     files.write("2 \n") # real 3D data one column
165 |     files.write("wfk \n")
166 |     files.write("0 \n")
167 | os.system("cut3d < wfk.files ")
168 |
169 | os.system("rm wfk.files")
170 | with open("wfk.files","a") as files:
171 |     files.write("outputo_DS1_WFK \n") #unperturbed wfk
172 |     files.write("1 \n")
173 |     files.write("0 \n")
174 |     files.write("4 \n") # Gamma point
175 |     files.write("8 \n") # Band nb 8
176 |     files.write("0 \n")
177 |     files.write("0 \n")
178 |     files.write("2 \n") # real 3D data one column
179 |     files.write("wfk \n")
180 |     files.write("0 \n")
181 | os.system("cut3d < wfk.files ")

```

```

182
183 # -----
184 # Perform the integration
185 # -----
186
187 Sum_mode = 0.0
188 modetmp = 0.0
189
190 for i in range(len(mode)):
191     # band nb 6
192     fichier1 = str(mode[i])
193     fichier2 = "wfk_k4_b6_s1"
194     reader1 = open(fichier1, "r")
195     reader2 = open(fichier2, "r")
196     X1 = reader1.readlines()
197     X2 = reader2.readlines()
198     summa = 0.0
199     for l in range(len(X2)):
200         summa = summa + eval(X2[l].split()[0]) * eval(X2[l].split()[0])
201
202     int = 0.0
203     for ii in range(len(X1)):
204         int = int + eval(X2[ii].split()[0]) * eval(X2[ii].split()[0]) * eval(X1[ii].split()[0])
205
206     result1 = int/summa
207
208     # band nb 7
209     fichier1 = str(mode[i])
210     fichier2 = "wfk_k4_b7_s1"
211     reader1 = open(fichier1, "r")
212     reader2 = open(fichier2, "r")
213     X1 = reader1.readlines()
214     X2 = reader2.readlines()
215     summa = 0.0
216     for l in range(len(X2)):
217         summa = summa + eval(X2[l].split()[0]) * eval(X2[l].split()[0])
218
219     int = 0.0
220     for ii in range(len(X1)):
221         int = int + eval(X2[ii].split()[0]) * eval(X2[ii].split()[0]) * eval(X1[ii].split()[0])
222
223     result2 = int/summa
224
225     # band nb 8
226     fichier1 = str(mode[i])
227     fichier2 = "wfk_k4_b8_s1"
228     reader1 = open(fichier1, "r")
229     reader2 = open(fichier2, "r")
230     X1 = reader1.readlines()
231     X2 = reader2.readlines()
232     summa = 0.0
233     for l in range(len(X2)):
234         summa = summa + eval(X2[l].split()[0]) * eval(X2[l].split()[0])
235
236     int = 0.0
237     for ii in range(len(X1)):
238         int = int + eval(X2[ii].split()[0]) * eval(X2[ii].split()[0]) * eval(X1[ii].split()[0])
239
240     result3 = int/summa
241
242     #Make the average of the three band
243     average = (result1+result2+result3)/3
244
245     if mode[i][0:2] == "EQ":
246         equili = average
247
248     #-----
249     # Mode contribution
250     #-----
251     print "<Psi0|VHXC|Psi0> ", average, " for mode", mode[i]
252
253     if mode[i][2:3] == "+" or mode[i][3:4] == "+" or mode[i][0:2] == "EQ":
254         modetmp = average
255
256     else:
257         emodetmp = (average+modetmp-2*equili)/(0.01**2)
258
259     Mcarbhone = 21894.16693
260
261     if mode[i][2:3] == "-" or mode[i][3:4] == "-":
262         emode = (1/(4*omega[i]*Mcarbhone))*emodetmp*(27.211383*1000) #(1/2 mode)
263         print ("Mode :"+mode[i]+" "+str(emode))
264
265     else:

```

```

266     emode = 0.0
267
268     Sum_mode += emode
269
270     # -----
271     # Compute the total NDDW ZPM
272     # -----
273
274     ZPM=Sum_mode
275
276     print ("ZPM[meV]"+str(ZPM))

```

Code 8: Python script used to solve the first part of the NDDW at q

5.6 NDDW using finite difference at the q=L point

```

1  # Find the eigenvector from the dynamical matrice
2
3  import numpy as np
4
5  def read_dynamical_matrix(datafile):
6      with open(datafile,'r') as f:
7          raw = list()
8          for line in f:
9              if line[0]=='#': continue
10             parts=line.split()
11             if not parts: continue
12             val=float(parts[4])
13             if np.abs(val) < 0.000000001:
14                 val=0.0
15             raw.append(val)
16             dynmat = np.array(raw)
17             dynmat = dynmat.reshape(6,6)
18             return dynmat
19
20  L_file = '/home/Samuel/Dropbox/WorkDiam/DerivEig_L/Dyanmical_mat.dat'
21
22  dynmatL=read_dynamical_matrix(L_file)
23
24  [eigvall,eigvectL]=np.linalg.eig(dynmatL)
25
26  eigvectL=np.transpose(eigvectL)
27  Mc=21894.16693
28
29  print "eigenvalues at L :"
30  for n in eigvall:
31      a = np.sqrt(n/Mc)
32      print a
33
34  print "eigenvectors at L :"
35  for n in eigvectL: print n
36
37  # Create the xcart needed for abinit input file
38
39  disp1 = 0.05
40  disp2 = 0.05
41  rprim = np.array([6.70346805,6.70346805,6.70346805])
42
43  print "# Equilibrium position"
44  print " xcart1 0.0 0.0 0.0 # 0.0 0.0 0.0 "
45  print "      1.67586701 1.67586701 1.67586701 # 0.25 0.25 0.25"
46  print " "
47  print "# LA mode (+h)"
48
49  # Direction LA
50  # Atome K
51  i = 0
52  SBX1 = np.array([eigvectL[i][0],eigvectL[i][1],eigvectL[i][2]])*eigvectL[i][0]*d
53  isp1
54  SBY1 = np.array([eigvectL[i][0],eigvectL[i][1],eigvectL[i][2]])*eigvectL[i][1]*d
55  isp1
56  SBZ1 = np.array([eigvectL[i][0],eigvectL[i][1],eigvectL[i][2]])*eigvectL[i][2]*d
57  isp1
58  SBX2 = np.array([eigvectL[i][3],eigvectL[i][4],eigvectL[i][5]])*eigvectL[i][3]*d
59  isp1
60  SBY2 = np.array([eigvectL[i][3],eigvectL[i][4],eigvectL[i][5]])*eigvectL[i][4]*d
61  isp1
62  SBZ2 = np.array([eigvectL[i][3],eigvectL[i][4],eigvectL[i][5]])*eigvectL[i][5]*d

```

```

63 | ispr1
64 |
65 | # Calculs translation
66 | i = 0
67 | TrX = np.array([1,0,0])*disp2
68 | TrY = np.array([0,1,0])*disp2
69 | TrZ = np.array([0,0,1])*disp2
70 |
71 | # -----
72 | # Calcul du terme d2Vhxc/dSBdT
73 | # Atomes K
74 | # -----
75 |
76 | #VHX(x,y)
77 | print " xcart2 %.10f %.10f %.10f "(rprim[0]*0.0+(SBX1[0]+TrX[0]), rprim[1]*
78 | 0.0+(SBX1[1]+TrX[1]),rprim[2]*0.0+(SBX1[2]+TrX[2]))
79 | print " %.10f %.10f %.10f "(rprim[0]*0.25+(SBX2[0]+TrX[0]), rprim[1]
80 | *0.25+(SBX2[1]+TrX[1]),rprim[2]*0.25+(SBX2[2]+TrX[2]))
81 | print ""
82 |
83 | #VHXC(-x,y)
84 | print " xcart3 %.10f %.10f %.10f "(rprim[0]*0.0+(-SBX1[0]+TrX[0]), rprim[1]
85 | *0.0+(-SBX1[1]+TrX[1]),rprim[2]*0.0+(-SBX1[2]+TrX[2]))
86 | print " %.10f %.10f %.10f "(rprim[0]*0.25+(-SBX2[0]+TrX[0]), rprim[1]
87 | *0.25+(-SBX2[1]+TrX[1]),rprim[2]*0.25+(-SBX2[2]+TrX[2]))
88 | print ""
89 |
90 | #VHXC(x,-y)
91 | print " xcart4 %.10f %.10f %.10f "(rprim[0]*0.0+(SBX1[0]-TrX[0]), rprim[1]*
92 | 0.0+(SBX1[1]-TrX[1]),rprim[2]*0.0+(SBX1[2]-TrX[2]))
93 | print " %.10f %.10f %.10f "(rprim[0]*0.25+(SBX2[0]-TrX[0]), rprim[1]
94 | *0.25+(SBX2[1]-TrX[1]),rprim[2]*0.25+(SBX2[2]-TrX[2]))
95 | print ""
96 |
97 | #VHXC(-x,-y)
98 | print " xcart5 %.10f %.10f %.10f "(rprim[0]*0.0+(-SBX1[0]-TrX[0]), rprim[1]
99 | *0.0+(-SBX1[1]-TrX[1]),rprim[2]*0.0+(-SBX1[2]-TrX[2]))
100 | print " %.10f %.10f %.10f "(rprim[0]*0.25+(-SBX2[0]-TrX[0]), rprim[1]
101 | *0.25+(-SBX2[1]-TrX[1]),rprim[2]*0.25+(-SBX2[2]-TrX[2]))
102 | print ""
103 |
104 | #VHY(x,y)
105 | print " xcart6 %.10f %.10f %.10f "(rprim[0]*0.0+(SBY1[0]+TrY[0]), rprim[1]*
106 | 0.0+(SBY1[1]+TrY[1]),rprim[2]*0.0+(SBY1[2]+TrY[2]))
107 | print " %.10f %.10f %.10f "(rprim[0]*0.25+(SBY2[0]+TrY[0]), rprim[1]
108 | *0.25+(SBY2[1]+TrY[1]),rprim[2]*0.25+(SBY2[2]+TrY[2]))
109 | print ""
110 |
111 | #VHYC(-x,y)
112 | print " xcart7 %.10f %.10f %.10f "(rprim[0]*0.0+(-SBY1[0]+TrY[0]), rprim[1]
113 | *0.0+(-SBY1[1]+TrY[1]),rprim[2]*0.0+(-SBY1[2]+TrY[2]))
114 | print " %.10f %.10f %.10f "(rprim[0]*0.25+(-SBY2[0]+TrY[0]), rprim[1]
115 | *0.25+(-SBY2[1]+TrY[1]),rprim[2]*0.25+(-SBY2[2]+TrY[2]))
116 | print ""
117 |
118 | #VHYC(x,-y)
119 | print " xcart8 %.10f %.10f %.10f "(rprim[0]*0.0+(SBY1[0]-TrY[0]), rprim[1]*
120 | 0.0+(SBY1[1]-TrY[1]),rprim[2]*0.0+(SBY1[2]-TrY[2]))
121 | print " %.10f %.10f %.10f "(rprim[0]*0.25+(SBY2[0]-TrY[0]), rprim[1]
122 | *0.25+(SBY2[1]-TrY[1]),rprim[2]*0.25+(SBY2[2]-TrY[2]))
123 | print ""
124 |
125 | #VHYC(-x,-y)
126 | print " xcart9 %.10f %.10f %.10f "(rprim[0]*0.0+(-SBY1[0]-TrY[0]), rprim[1]
127 | *0.0+(-SBY1[1]-TrY[1]),rprim[2]*0.0+(-SBY1[2]-TrY[2]))
128 | print " %.10f %.10f %.10f "(rprim[0]*0.25+(-SBY2[0]-TrY[0]), rprim[1]
129 | *0.25+(-SBY2[1]-TrY[1]),rprim[2]*0.25+(-SBY2[2]-TrY[2]))
130 | print ""
131 |
132 | #VHZ(x,y)
133 | print " xcart10 %.10f %.10f %.10f "(rprim[0]*0.0+(SBZ1[0]+TrZ[0]), rprim[1]
134 | *0.0+(SBZ1[1]+TrZ[1]),rprim[2]*0.0+(SBZ1[2]+TrZ[2]))
135 | print " %.10f %.10f %.10f "(rprim[0]*0.25+(SBZ2[0]+TrZ[0]), rprim[1]
136 | *0.25+(SBZ2[1]+TrZ[1]),rprim[2]*0.25+(SBZ2[2]+TrZ[2]))
137 | print ""
138 |
139 | #VHZC(-x,y)
140 | print " xcart11 %.10f %.10f %.10f "(rprim[0]*0.0+(-SBZ1[0]+TrZ[0]), rprim[1]
141 | *0.0+(-SBZ1[1]+TrZ[1]),rprim[2]*0.0+(-SBZ1[2]+TrZ[2]))
142 | print " %.10f %.10f %.10f "(rprim[0]*0.25+(-SBZ2[0]+TrZ[0]), rprim[1]
143 | *0.25+(-SBZ2[1]+TrZ[1]),rprim[2]*0.25+(-SBZ2[2]+TrZ[2]))
144 | print ""
145 |
146 | #VHZC(x,-y)

```

```

147 print " xcart12 %.10f %.10f %.10f"%(rprim[0]*0.0+(SBZ1[0]-TrZ[0]), rprim[1]
148 *0.0+(SBZ1[1]-TrZ[1]),rprim[2]*0.0+(SBZ1[2]-TrZ[2]))
149 print "          %.10f %.10f %.10f"%(rprim[0]*0.25+(SBZ2[0]-TrZ[0]), rprim[1]
150 *0.25+(SBZ2[1]-TrZ[1]),rprim[2]*0.25+(SBZ2[2]-TrZ[2]))
151 print ""
152
153 #VHZC(-x,-y)
154 print " xcart13 %.10f %.10f %.10f"%(rprim[0]*0.0+(-SBZ1[0]-TrZ[0]), rprim[1]
155 *0.0+(-SBZ1[1]-TrZ[1]),rprim[2]*0.0+(-SBZ1[2]-TrZ[2]))
156 print "          %.10f %.10f %.10f"%(rprim[0]*0.25+(-SBZ2[0]-TrZ[0]), rprim[1]
157 *0.25+(-SBZ2[1]-TrZ[1]),rprim[2]*0.25+(-SBZ2[2]-TrZ[2]))
158 print ""
159
160 #Calculs de Delta lambda1, Delta lambda2
161 h=np.sqrt(dis1**2+dis2**2)
162 print "h :", h

```

Code 9: Python script used to create input files to solve the second part of the NDDW at q

5.7 NDDW using finite difference at the q=L point

```

1 import math
2 import os
3 import numpy as np
4
5 # Transform binary VHXC file into text file
6
7 os.system("rm cut3d.files")
8 with open("cut3d.files","a") as files:
9     files.write("outputo_DS1_VHXC \n")
10    files.write("1 \n")
11    files.write("5 \n") #3D formatted data (output the bare 3D data - one column)
12    files.write("EQ \n")
13    files.write("0 \n")
14 os.system("cut3d < cut3d.files ")
15
16 os.system("rm cut3d.files")
17 with open("cut3d.files","a") as files:
18     files.write("outputo_DS2_VHXC \n")
19     files.write("1 \n")
20     files.write("5 \n")
21     files.write("LA+ \n")
22     files.write("0 \n")
23 os.system("cut3d < cut3d.files ")
24
25 os.system("rm cut3d.files")
26 with open("cut3d.files","a") as files:
27     files.write("outputo_DS3_VHXC \n")
28     files.write("1 \n")
29     files.write("5 \n")
30     files.write("LA- \n")
31     files.write("0 \n")
32 os.system("cut3d < cut3d.files ")
33
34 os.system("rm cut3d.files")
35 with open("cut3d.files","a") as files:
36     files.write("outputo_DS4_VHXC \n")
37     files.write("1 \n")
38     files.write("5 \n")
39     files.write("LO+ \n")
40     files.write("0 \n")
41 os.system("cut3d < cut3d.files ")
42
43 os.system("rm cut3d.files")
44 with open("cut3d.files","a") as files:
45     files.write("outputo_DS5_VHXC \n")
46     files.write("1 \n")
47     files.write("5 \n")
48     files.write("LO- \n")
49     files.write("0 \n")
50 os.system("cut3d < cut3d.files ")
51
52 os.system("rm cut3d.files")
53 with open("cut3d.files","a") as files:
54     files.write("outputo_DS6_VHXC \n")
55     files.write("1 \n")
56     files.write("5 \n")
57     files.write("TA1+ \n")

```

```

58         files.write("0 \n")
59 os.system("cut3d < cut3d.files ")
60
61 os.system("rm cut3d.files")
62 with open("cut3d.files","a") as files:
63     files.write("outputo_DS7_VHXC \n")
64     files.write("1 \n")
65     files.write("5 \n")
66     files.write("TA1- \n")
67     files.write("0 \n")
68 os.system("cut3d < cut3d.files ")
69
70 os.system("rm cut3d.files")
71 with open("cut3d.files","a") as files:
72     files.write("outputo_DS8_VHXC \n")
73     files.write("1 \n")
74     files.write("5 \n")
75     files.write("TA2+ \n")
76     files.write("0 \n")
77 os.system("cut3d < cut3d.files ")
78
79 os.system("rm cut3d.files")
80 with open("cut3d.files","a") as files:
81     files.write("outputo_DS9_VHXC \n")
82     files.write("1 \n")
83     files.write("5 \n")
84     files.write("TA2- \n")
85     files.write("0 \n")
86 os.system("cut3d < cut3d.files ")
87
88 os.system("rm cut3d.files")
89 with open("cut3d.files","a") as files:
90     files.write("outputo_DS10_VHXC \n")
91     files.write("1 \n")
92     files.write("5 \n")
93     files.write("T01+ \n")
94     files.write("0 \n")
95 os.system("cut3d < cut3d.files ")
96
97 os.system("rm cut3d.files")
98 with open("cut3d.files","a") as files:
99     files.write("outputo_DS11_VHXC \n")
100     files.write("1 \n")
101     files.write("5 \n")
102     files.write("T01- \n")
103     files.write("0 \n")
104 os.system("cut3d < cut3d.files ")
105
106 os.system("rm cut3d.files")
107 with open("cut3d.files","a") as files:
108     files.write("outputo_DS12_VHXC \n")
109     files.write("1 \n")
110     files.write("5 \n")
111     files.write("T02+ \n")
112     files.write("0 \n")
113 os.system("cut3d < cut3d.files ")
114
115 os.system("rm cut3d.files")
116 with open("cut3d.files","a") as files:
117     files.write("outputo_DS13_VHXC \n")
118     files.write("1 \n")
119     files.write("5 \n")
120     files.write("T02- \n")
121     files.write("0 \n")
122 os.system("cut3d < cut3d.files ")
123
124 # -----
125 # Make centered finite difference of order 2 of Vhxc for each mode
126 # d2Vhxc/dR1dR2=(2*Vhxc(mode)-2*Vhxc(eq))/(h^2)
127 # -----
128
129 mode = np.array(["EQ","LA+","LA-","LO+","LO-","TA1+","TA1-","TA2+","TA2-","
130 "T01+","T01-","T02+","T02-"])
131
132 # The phonon frequency are eigenvalue of the Dynamical eigenvalue problem.
133
134 omega = np.array([0.0,0.00480084,0.00480084,0.005785943,0.005785943,0.002453274,
135 0.002453274,0.002453274,0.002453274,0.005671174,0.005671174,0.005671174,0.005671174])
136
137 # -----
138 # Prepare the integration : de2/dR1dR2=int d2Vhxc/dR1dR2 psi*psi(unperturbed)
139 # -----
140
141 os.system("rm wfk.files")

```

```

142 with open("wfk.files", "a") as files:
143     files.write("outputo_DS1_WFK \n") #unperturbed wfk
144     files.write("1 \n")
145     files.write("0 \n")
146     files.write("4 \n") # Gamma point
147     files.write("6 \n") # Band nb 6
148     files.write("0 \n")
149     files.write("0 \n")
150     files.write("2 \n") # real 3D data one column
151     files.write("wfk \n")
152     files.write("0 \n")
153 os.system("cut3d < wfk.files ")
154
155 os.system("rm wfk.files")
156 with open("wfk.files", "a") as files:
157     files.write("outputo_DS1_WFK \n") #unperturbed wfk
158     files.write("1 \n")
159     files.write("0 \n")
160     files.write("4 \n") # Gamma point
161     files.write("7 \n") # Band nb 7
162     files.write("0 \n")
163     files.write("0 \n")
164     files.write("2 \n") # real 3D data one column
165     files.write("wfk \n")
166     files.write("0 \n")
167 os.system("cut3d < wfk.files ")
168
169 os.system("rm wfk.files")
170 with open("wfk.files", "a") as files:
171     files.write("outputo_DS1_WFK \n") #unperturbed wfk
172     files.write("1 \n")
173     files.write("0 \n")
174     files.write("4 \n") # Gamma point
175     files.write("8 \n") # Band nb 8
176     files.write("0 \n")
177     files.write("0 \n")
178     files.write("2 \n") # real 3D data one column
179     files.write("wfk \n")
180     files.write("0 \n")
181 os.system("cut3d < wfk.files ")
182
183 # -----
184 # Perform the integration
185 # -----
186
187 Sum_mode = 0.0
188 modetmp = 0.0
189
190 for i in range(len(mode)):
191     # band nb 6
192     fichier1 = str(mode[i])
193     fichier2 = "wfk_k4_b6_s1"
194     reader1 = open(fichier1, "r")
195     reader2 = open(fichier2, "r")
196     X1 = reader1.readlines()
197     X2 = reader2.readlines()
198     summa = 0.0
199     for l in range(len(X2)):
200         summa = summa + eval(X2[l].split()[0]) * eval(X2[l].split()[0])
201
202     int = 0.0
203     for ii in range(len(X1)):
204         int = int + eval(X2[ii].split()[0]) * eval(X2[ii].split()[0]) * eval(X1[ii].split()[0])
205
206     result1 = int/summa
207
208     # band nb 7
209     fichier1 = str(mode[i])
210     fichier2 = "wfk_k4_b7_s1"
211     reader1 = open(fichier1, "r")
212     reader2 = open(fichier2, "r")
213     X1 = reader1.readlines()
214     X2 = reader2.readlines()
215     summa = 0.0
216     for l in range(len(X2)):
217         summa = summa + eval(X2[l].split()[0]) * eval(X2[l].split()[0])
218
219     int = 0.0
220     for ii in range(len(X1)):
221         int = int + eval(X2[ii].split()[0]) * eval(X2[ii].split()[0]) * eval(X1[ii].split()[0])
222
223     result2 = int/summa
224
225     # band nb 8

```



```

226 fichier1 = str(mode[i])
227 fichier2 = "wfk_k4_b8_s1"
228 reader1 = open(fichier1,"r")
229 reader2 = open(fichier2,"r")
230 X1 = reader1.readlines()
231 X2 = reader2.readlines()
232 summa = 0.0
233 for l in range(len(X2)):
234     summa = summa + eval(X2[l].split()[0]) * eval(X2[l].split()[0])
235
236 int = 0.0
237 for ii in range(len(X1)):
238     int = int + eval(X2[ii].split()[0]) * eval(X2[ii].split()[0]) * eval(X1[ii].split()[0])
239
240 result3 = int/summa
241
242 #Make the average of the three band
243 average = (result1+result2+result3)/3
244
245 if mode[i][0:2] == "EQ":
246     equili = average
247
248 #-----
249 # Mode contribution
250 #-----
251 print "<Psi0|VHXC|Psi0> ", average, " for mode", mode[i]
252
253 if mode[i][2:3] == "+" or mode[i][3:4] == "+" or mode[i][0:2] == "EQ":
254     modetmp = average
255
256 else:
257     emodetmp = (average+modetmp-2*equili)/(0.01**2)
258
259 Mcarbne = 21894.16693
260
261 if mode[i][2:3] == "-" or mode[i][3:4] == "-":
262     emode = (1/(4*omega[i]*Mcarbne))*emodetmp*(27.211383*1000) #(1/2 mode)
263     print ("Mode : "+mode[i]+" "+str(emode))
264
265 else:
266     emode = 0.0
267
268 Sum_mode += emode
269
270 # -----
271 # Compute the total NDDW ZPM
272 # -----
273
274 ZPM=Sum_mode
275
276 print ("ZPM[meV]"+str(ZPM))

```

Code 10: Python script used to solve the second part of the NDDW at q

5.8 AHC at the q=L point

```

1 # C in diamond structure; 2x2x2 FCC special point grid; low ecut.
2 ndtset 4
3
4 jdtset 11 12 21 22
5
6 elph2_imagden 0.1 eV
7
8 # The first point must be the Gamma one
9 qpt11 0.0 0.0 0.0
10 qpt12 0.0 0.0 0.0
11 qpt21 0.5 0.0 0.0
12 qpt22 0.5 0.0 0.0
13
14 # Ground state density
15 getwfk11 0
16 getden11 0
17 nqpt11 0
18 ieig2rf11 0
19 rfphon11 0
20 smdelta11 0
21 iscf11 7
22

```

```

23 # RF at q=0
24 getwfk12 0
25
26 # Non self-consistent calculation on a q point
27 ieig2rf?1 0
28 smdelta?1 0
29 rfphon?1 0
30 iscf?1 -2
31
32 # Computation at q
33 getgam_eig2nkq?2 12
34 getwfk?2 -1
35
36 # Common input variables, to be superceded in some cases
37 nkpt 1
38 ieig2rf 1
39 bdeigrf 8
40 smdelta 1
41 getwfk 11 # Use GS wf from q=0
42 getden 11 # Use density from q=0
43 kptopt 3 # Need full k-point set for finite q response
44 rfphon 1 # Do phonon response
45 rfatpol 1 2 # Treat displacements of all atoms
46 rfdir 1 1 1 # Do all directions (symmetry will be used)
47
48 #Size-dependent parameters
49 acell 3*6.70346805 #relaxed param.
50 rprim 0 .5 .5 .5 0 .5 .5 .5 0
51
52 natom 2
53 typat 1 1
54 xred 3*0.00d0 3*0.25d0
55
56 nband 8
57
58 ngkpt 2 2 2
59 nshiftk 4
60 shiftk 0.0 0.0 0.0
61         0.0 0.5 0.5
62         0.5 0.0 0.5
63         0.5 0.5 0.0
64
65 # Miscellaneous
66 ntypat 1
67 znuc1 6
68 diemac 6.0d0
69 ecut 30
70 enunit 2
71 nstep 30
72 nsym 1
73 tolwfr 1.0d-16

```

Code 11: ABINIT input file to obtain the contribution of the q-point L for Γ using the AHC formalism.

5.9 Python scripts I

```

1 # -----
2 # This script genereate cartesian coordinate for Abinit input files.
3 # Specifically for q = L = (0.5 0.0 0.0)
4 # Create inputs for finite diff. on eigenenergies and NDDW
5 # -----
6
7 import numpy as np
8
9 # -----
10 # Find the eigenvector and eigenenergies from the dynamical matrix
11 # computed using DFPT in ABINIT
12 # -----
13
14 # Location of the Dynamical matrix file
15 L_file = '/home/Samuel/Dropbox/WorkDiam/DerivEig_L/Dyanmical_mat.dat'
16 # Carbon mass in a.u.
17 Mc=21894.16693
18
19 def read_dynamical_matrix(datafile):
20     with open(datafile,'r') as f:
21         raw = list()
22         for line in f:

```

```

23     if line[0]=='#': continue
24     parts=line.split()
25     if not parts: continue
26     val=float(parts[4])
27     if np.abs(val) < 0.000000001:
28         val=0.0
29         raw.append(val)
30     dynmat = np.array(raw)
31     dynmat = dynmat.reshape(6,6)
32     return dynmat
33
34 dynmatL=read_dynamical_matrix(L_file)
35 [eigvall,eigvectL]=np.linalg.eig(dynmatL)
36 eigvectL=np.transpose(eigvectL)
37
38 print "eigenvalues at L : "
39 for n in eigvall:
40     a = np.sqrt(n/Mc)
41     print a
42
43 print "eigenvectors at L : "
44 for n in eigvectL: print n
45
46 # -----
47 # Create Abinit input file for finite difference on eigen energies
48 # and the first part of NDDW. On a supercell 2x1x1
49 # -----
50
51 # Value and number of the displacement (always 1/2 of the previous)
52 h = np.array([0.04,0.02,0.01,0.005,0.0025,0.00125,0.000625])
53
54 # Cell parameters
55 acell = np.array([13.4069361,6.70346805,6.70346805])
56 rprim = np.array([[0.0,0.5,0.5],[0.5,0.0,0.5],[0.5,0.5,0.0]])
57 cell = np.mat([rprim[0]*acell[0],rprim[1]*acell[1],rprim[2]*acell[2]])
58 xred = np.mat([[0.0,0.0,0.0],[0.125,0.25,0.25],[0.5,0.0,0.0],\
59 [0.625,0.25,0.25]])
60
61
62 # In the specific case of the L point, h = -h
63 # Indice that record the number of xcart in the input file
64 j = 1
65
66 # Equilibrium position
67 print " "
68 print "Contribution of the L point on the electronic eigenenergies at Gamma"
69 print "and the first part of the NDDW."
70 print "Using finite difference."
71 print " "
72 print " xcart%.0f %.15f %.15f %.15f"%(j,(xred[0]*cell)[0,0],\
73 (xred[0]*cell)[0,1],(xred[0]*cell)[0,2])
74 print " %.15f %.15f %.15f"%((xred[1]*cell)[0,0],\
75 (xred[1]*cell)[0,1],(xred[1]*cell)[0,2])
76 print " %.15f %.15f %.15f"%((xred[2]*cell)[0,0],\
77 (xred[2]*cell)[0,1],(xred[2]*cell)[0,2])
78 print " %.15f %.15f %.15f"%((xred[3]*cell)[0,0],\
79 (xred[3]*cell)[0,1],(xred[3]*cell)[0,2])
80 print " "
81 j += 1
82
83 for disp in h:
84     # LA mode
85     i = 0
86     print "Displacement: ", disp
87     print " xcart%.0f %.15f %.15f %.15f"%(j,(xred[0]*cell)[0,0]+\
88 eigvectL[i][0]*disp,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,\
89 (xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
90     print " %.15f %.15f %.15f"%((xred[1]*cell)[0,0]+\
91 eigvectL[i][3]*disp,(xred[1]*cell)[0,1]+eigvectL[i][4]*disp,\
92 (xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
93     print " %.15f %.15f %.15f"%((xred[2]*cell)[0,0]-\
94 eigvectL[i][0]*disp,(xred[2]*cell)[0,1]-eigvectL[i][1]*disp,\
95 (xred[2]*cell)[0,2]-eigvectL[i][2]*disp)
96     print " %.15f %.15f %.15f"%((xred[3]*cell)[0,0]-\
97 eigvectL[i][3]*disp,(xred[3]*cell)[0,1]-eigvectL[i][4]*disp,\
98 (xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
99     print " "
100    j += 1
101    # L0 mode
102    i = 1
103    print " xcart%.0f %.15f %.15f %.15f"%(j,(xred[0]*cell)[0,0]+\
104 eigvectL[i][0]*disp,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,\
105 (xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
106    print " %.15f %.15f %.15f"%((xred[1]*cell)[0,0]+\

```

```

107 eigvectL[i][3]*disp,(xred[1]*cell)[0,1]+eigvectL[i][4]*disp,\
108 (xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
109 print " %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
110 eigvectL[i][0]*disp,(xred[2]*cell)[0,1]-eigvectL[i][1]*disp,\
111 (xred[2]*cell)[0,2]-eigvectL[i][2]*disp)
112 print " %.15f %.15f %.15f "%((xred[3]*cell)[0,0]-\
113 eigvectL[i][3]*disp,(xred[3]*cell)[0,1]-eigvectL[i][4]*disp,\
114 (xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
115 print " "
116 j += 1
117 # TA mode
118 i = 2
119 print " xcart%.0f %.15f %.15f %.15f "%(j,(xred[0]*cell)[0,0]+\
120 eigvectL[i][0]*disp,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,\
121 (xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
122 print " %.15f %.15f %.15f "%((xred[1]*cell)[0,0]+\
123 eigvectL[i][3]*disp,(xred[1]*cell)[0,1]+eigvectL[i][4]*disp,\
124 (xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
125 print " %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
126 eigvectL[i][0]*disp,(xred[2]*cell)[0,1]-eigvectL[i][1]*disp,\
127 (xred[2]*cell)[0,2]-eigvectL[i][2]*disp)
128 print " %.15f %.15f %.15f "%((xred[3]*cell)[0,0]-\
129 eigvectL[i][3]*disp,(xred[3]*cell)[0,1]-eigvectL[i][4]*disp,\
130 (xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
131 print " "
132 j += 1
133 # TO mode
134 i = 4
135 print " xcart%.0f %.15f %.15f %.15f "%(j,(xred[0]*cell)[0,0]+\
136 eigvectL[i][0]*disp,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,\
137 (xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
138 print " %.15f %.15f %.15f "%((xred[1]*cell)[0,0]+\
139 eigvectL[i][3]*disp,(xred[1]*cell)[0,1]+eigvectL[i][4]*disp,\
140 (xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
141 print " %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
142 eigvectL[i][0]*disp,(xred[2]*cell)[0,1]-eigvectL[i][1]*disp,\
143 (xred[2]*cell)[0,2]-eigvectL[i][2]*disp)
144 print " %.15f %.15f %.15f "%((xred[3]*cell)[0,0]-\
145 eigvectL[i][3]*disp,(xred[3]*cell)[0,1]-eigvectL[i][4]*disp,\
146 (xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
147 print " "
148 j += 1
149
150 # -----
151 # Second part of the NDDW on a primitive cell
152 # We only compute 6xi=(3xi_1,3xi_1) as eigen displacement
153 # and then multiply by 2 because 6xi=(3xi_2,3xi_2) should be the same.
154 # -----
155
156 acell = np.array([6.70346805,6.70346805,6.70346805])
157 cell = np.mat([rprim[0]*acell[0],rprim[1]*acell[1],rprim[2]*acell[2]])
158 xred = np.mat([[0.0,0.0,0.0],[0.25,0.25,0.25]])
159
160 print " "
161 print "Contribution of the L point on the electronic eigenenergies at Gamma"
162 print "and the second part of the NDDW."
163 print "Using finite difference."
164 print " "
165
166 j = 1
167 # Equilibrium position
168 print " xcart%.0f %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0],\
169 (xred[0]*cell)[0,1],(xred[0]*cell)[0,2])
170 print " %.10f %.10f %.10f "%((xred[1]*cell)[0,0],\
171 (xred[1]*cell)[0,1],(xred[1]*cell)[0,2])
172 print " "
173 j += 1
174
175 for disp in h:
176 print "Displacement: ", disp
177 # LA mode
178 i = 0
179 print " xcart%.0f %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]\
180 +eigvectL[i][0]*disp,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,\
181 (xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
182 print " %.10f %.10f %.10f "%((xred[1]*cell)[0,0]+\
183 eigvectL[i][3]*disp,(xred[1]*cell)[0,1]+eigvectL[i][4]*disp,\
184 (xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
185 print " "
186 j += 1
187 # LO mode
188 i = 1
189 print " xcart%.0f %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]\
190 +eigvectL[i][0]*disp,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,\

```

```

191     (xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
192 print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]+\
193     eigvectL[i][0]*disp,(xred[1]*cell)[0,1]+eigvectL[i][1]*disp,\
194     (xred[1]*cell)[0,2]+eigvectL[i][2]*disp)
195 print " "
196 j += 1
197 # TA mode
198 i = 2
199 print " xcart%.0f %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]\
200     +eigvectL[i][0]*disp,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,\
201     (xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
202 print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]+\
203     eigvectL[i][0]*disp,(xred[1]*cell)[0,1]+eigvectL[i][1]*disp,\
204     (xred[1]*cell)[0,2]+eigvectL[i][2]*disp)
205 print " "
206 j += 1
207 # T0 mode
208 i = 4
209 print " xcart%.0f %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]\
210     +eigvectL[i][0]*disp,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,\
211     (xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
212 print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]+\
213     eigvectL[i][0]*disp,(xred[1]*cell)[0,1]+eigvectL[i][1]*disp,\
214     (xred[1]*cell)[0,2]+eigvectL[i][2]*disp)
215 print " "
216 j += 1

```

Code 12: Python scripts use to create ABINIT input files

5.10 Python scripts II

```

1  # -----
2  # This python script is used to compute the contribution of the
3  # L point to the Gamma eigenenergies using finite difference.
4  # -----
5  import os
6  import numpy as np
7
8  def read_eigenenergies(datafile):
9      flag = False
10     with open(datafile,'r') as f:
11         for line in f:
12             if flag:
13                 eigenenergies = map(float,line.split())
14                 break
15             if "kpt= 0.0000 0.0000 0.0000" in line:
16                 flag = True
17     return eigenenergies
18
19 def use_cut3d(datafile,out,gamma,band):
20     if datafile[len(datafile)-4:len(datafile)] == "VHXC":
21         os.system("rm cut3d.files")
22         with open("cut3d.files","a") as files:
23             files.write(str(datafile)+"\n")
24             files.write("1 \n")
25             files.write("5 \n") #3D formatted data
26             # (output the bare 3D data - one column)
27             files.write(str(out)+"\n")
28             files.write("0 \n")
29     os.system("cut3d < cut3d.files ")
30     if datafile[len(datafile)-3:len(datafile)] == "WFK":
31         os.system("rm wfk.files")
32         with open("wfk.files","a") as files:
33             files.write(str(datafile)+"\n")
34             files.write("1 \n")
35             files.write("0 \n")
36             files.write(str(gamma)+" \n") # Gamma point
37             files.write(str(band)+" \n") # Band nb 6
38             files.write("0 \n")
39             files.write("0 \n")
40             files.write("2 \n") # real 3D data one column
41             files.write(str(out)+" \n")
42             files.write("0 \n")
43         os.system("cut3d < wfk.files ")
44
45 def integration(VHXC,wfk):
46     psi_file=open(wfk,'r')
47     v_file=open(VHXC,'r')

```

```

48 S=0.0; N=0.0
49 while True:
50     psi_l=psi_file.readline()
51     v_l=v_file.readline()
52     if not psi_l or not v_l: break
53     psi=float(psi_l)
54     v=float(v_l)
55     S+=psi*v*psi
56     N+=psi*psi
57 psi_file.close()
58 v_file.close()
59 S=S/N
60 return S
61
62 # The base name for the supercell files are assumed to be named "supercell"
63 # and the primitive as assumed to be named "primitive"
64
65 # Import the eigenenergies of the equilibrium state in Ha
66 eigenenergiesEQ = read_eigenenergies("supercello_DS1_EIG")
67
68 # Value and number of the displacement
69 h = np.array([0.04,0.02,0.01,0.005,0.0025,0.00125,0.000625])
70
71 # Initialisation values
72 k = 0
73
74 # The second dimension is the number of h in geometric progression
75 # h,h/2,h/4,h/8 etc..
76 # The third index correspond to modes (here 4)
77 # The first index correspond to the Richardson extrapolation
78 D = np.zeros([len(h),len(h),4])
79
80 # Frequency (computed in DFPT using ABINIT)
81 omega = np.array([0.00480084,0.005785943,0.002453274,0.005671174])
82
83 # Carbon mass in a.u.
84 Mc = 21894.16693
85
86 while k < len(h):
87
88     # LA mode
89     eigenenergiesLA = read_eigenenergies("supercello_DS"+str(4*k+2)+"_EIG")
90     # LO mode
91     eigenenergiesLO = read_eigenenergies("supercello_DS"+str(4*k+3)+"_EIG")
92     # TA mode
93     eigenenergiesTA = read_eigenenergies("supercello_DS"+str(4*k+4)+"_EIG")
94     # TO mode
95     eigenenergiesTO = read_eigenenergies("supercello_DS"+str(4*k+5)+"_EIG")
96
97     # Second order derivative (meV)
98     # LA mode
99     D[0,k,0] = 2*(eigenenergiesLA[5]+eigenenergiesLA[6]+eigenenergiesLA[7]\
100         -eigenenergiesEQ[5]-eigenenergiesEQ[6]-eigenenergiesEQ[7])/\
101         (3*(h[k]**2))*(1.0/(4*Mc*omega[0]))*27.211383*1000
102     # LO mode
103     D[0,k,1] = 2*(eigenenergiesLO[5]+eigenenergiesLO[6]+eigenenergiesLO[7]\
104         -eigenenergiesEQ[5]-eigenenergiesEQ[6]-eigenenergiesEQ[7])/\
105         (3*(h[k]**2))*(1.0/(4*Mc*omega[1]))*27.211383*1000
106     # TA mode
107     D[0,k,2] = 2*(eigenenergiesTA[5]+eigenenergiesTA[6]+eigenenergiesTA[7]\
108         -eigenenergiesEQ[5]-eigenenergiesEQ[6]-eigenenergiesEQ[7])/\
109         (3*(h[k]**2))*(1.0/(4*Mc*omega[2]))*27.211383*1000
110     # TO mode
111     D[0,k,3] = 2*(eigenenergiesTO[5]+eigenenergiesTO[6]+eigenenergiesTO[7]\
112         -eigenenergiesEQ[5]-eigenenergiesEQ[6]-eigenenergiesEQ[7])/\
113         (3*(h[k]**2))*(1.0/(4*Mc*omega[3]))*27.211383*1000
114
115     k += 1
116
117 # -----
118 # We will now use the Richardson Extrapolation to speed up the convergence
119 # -----
120
121 l = 0
122 while l < len(h):
123     k = 0
124     while k < len(h):
125         if (l-1) >= 0 and (k-1) >= 0:
126             # LA mode
127             D[l,k,0] = (D[l-1,k,0]-(D[l-1,k-1,0]*(1.0/(2**(2*l)))))/\
128                 (1-(1.0/(2**(2*l))))
129             if D[l-1,k-1,0] == 0.0:
130                 D[l,k,0] = 0.0
131             # LO mode

```

```

132     D[l,k,1] = (D[l-1,k,1]-(D[l-1,k-1,1]*(1.0/(2**(2*l)))))/\
133             (1-(1.0/(2**(2*l))))
134     if D[l-1,k-1,1] == 0.0:
135         D[l,k,1] = 0.0
136 #   TA mode
137     D[l,k,2] = (D[l-1,k,2]-(D[l-1,k-1,2]*(1.0/(2**(2*l)))))/\
138             (1-(1.0/(2**(2*l))))
139     if D[l-1,k-1,2] == 0.0:
140         D[l,k,2] = 0.0
141 #   T0 mode
142     D[l,k,3] = (D[l-1,k,3]-(D[l-1,k-1,3]*(1.0/(2**(2*l)))))/\
143             (1-(1.0/(2**(2*l))))
144     if D[l-1,k-1,3] == 0.0:
145         D[l,k,3] = 0.0
146     k += 1
147     l += 1
148
149 Deigen = D
150
151 print "Richardson extrapolation of eigenenergies"
152 print Deigen
153
154 # -----
155 # Compute the first part the the NDDW term
156 # -----
157
158 # Second derivative using finite centred differences
159 # The second dimension is the number of h in geometric progression
160 # h,h/2,h/4,h/8 etc..
161 # The third index correspond to modes (here 4)
162 # The first index correspond to the Richardson extrapolation
163 D = np.zeros([len(h),len(h),4])
164
165 # Prepare equilibrium files
166 use_cut3d("supercello_DS1_VHXC","EQ",0,0)
167 use_cut3d("supercello_DS1_WFK","wfk",7,6)
168 use_cut3d("supercello_DS1_WFK","wfk",7,7)
169 use_cut3d("supercello_DS1_WFK","wfk",7,8)
170 averageEQ = (integration("EQ","wfk_k7_b6_s1")+
171             integration("EQ","wfk_k7_b7_s1")+integration("EQ","wfk_k7_b8_s1"))/3
172
173 k = 0
174 while k < len(h):
175
176 # Transform binary files into .txt files using cut3d
177 use_cut3d("supercello_DS"+str(4*k+2)+"_VHXC","LA",0,0)
178 use_cut3d("supercello_DS"+str(4*k+3)+"_VHXC","LO",0,0)
179 use_cut3d("supercello_DS"+str(4*k+4)+"_VHXC","TA",0,0)
180 use_cut3d("supercello_DS"+str(4*k+5)+"_VHXC","TO",0,0)
181
182 # Perform the integration int(psi_0*VHXC*psi_0)
183 averageLA = (integration("LA","wfk_k7_b6_s1")+
184             integration("LA","wfk_k7_b7_s1")+integration("LA","wfk_k7_b8_s1"))/3
185 averageLO = (integration("LO","wfk_k7_b6_s1")+
186             integration("LO","wfk_k7_b7_s1")+integration("LO","wfk_k7_b8_s1"))/3
187 averageTA = (integration("TA","wfk_k7_b6_s1")+
188             integration("TA","wfk_k7_b7_s1")+integration("TA","wfk_k7_b8_s1"))/3
189 averageTO = (integration("TO","wfk_k7_b6_s1")+
190             integration("TO","wfk_k7_b7_s1")+integration("TO","wfk_k7_b8_s1"))/3
191
192 # Second order derivative (meV)
193 # LA mode
194 D[0,k,0] = 2*(averageLA-averageEQ)/\
195         (h[k]**2)*(1.0/(4*Mc*omega[0]))*27.211383*1000
196 # LO mode
197 D[0,k,1] = 2*(averageLO-averageEQ)/\
198         (h[k]**2)*(1.0/(4*Mc*omega[1]))*27.211383*1000
199 # TA mode
200 D[0,k,2] = 2*(averageTA-averageEQ)/\
201         (h[k]**2)*(1.0/(4*Mc*omega[2]))*27.211383*1000
202 # T0 mode
203 D[0,k,3] = 2*(averageTO-averageEQ)/\
204         (h[k]**2)*(1.0/(4*Mc*omega[3]))*27.211383*1000
205     k += 1
206
207 # -----
208 # We will now use the Richardson Extrapolation to speed up the convergence
209 # -----
210
211 l = 0
212 while l < len(h):
213     k = 0
214     while k < len(h):
215         if (l-1) >= 0 and (k-1) >= 0:

```

```

216 # LA mode
217 D[1,k,0] = (D[1-1,k,0]-(D[1-1,k-1,0]*(1.0/(2**(2*1)))))/\
218           (1-(1.0/(2**(2*1))))
219 if D[1-1,k-1,0] == 0.0:
220     D[1,k,0] = 0.0
221 # LO mode
222 D[1,k,1] = (D[1-1,k,1]-(D[1-1,k-1,1]*(1.0/(2**(2*1)))))/\
223           (1-(1.0/(2**(2*1))))
224 if D[1-1,k-1,1] == 0.0:
225     D[1,k,1] = 0.0
226 # TA mode
227 D[1,k,2] = (D[1-1,k,2]-(D[1-1,k-1,2]*(1.0/(2**(2*1)))))/\
228           (1-(1.0/(2**(2*1))))
229 if D[1-1,k-1,2] == 0.0:
230     D[1,k,2] = 0.0
231 # TO mode
232 D[1,k,3] = (D[1-1,k,3]-(D[1-1,k-1,3]*(1.0/(2**(2*1)))))/\
233           (1-(1.0/(2**(2*1))))
234 if D[1-1,k-1,3] == 0.0:
235     D[1,k,3] = 0.0
236 k += 1
237 l += 1
238
239 Dfirst = D
240 print "Richardson extrapolation of the first term of NDDW"
241 print Dfirst
242
243 # -----
244 # Compute the second part the the NDDW term on a primitive cell
245 # -----
246
247 D = np.zeros([len(h),len(h),4])
248
249 # Prepare equilibrium files
250 use_cut3d("primitiveo_DS1_VHXC", "EQ", 0, 0)
251 use_cut3d("primitiveo_DS1_WFK", "wfk", 20, 2)
252 use_cut3d("primitiveo_DS1_WFK", "wfk", 20, 3)
253 use_cut3d("primitiveo_DS1_WFK", "wfk", 20, 4)
254 averageEQ = (integration("EQ", "wfk_k20_b2_s1")+
255              integration("EQ", "wfk_k20_b3_s1")+integration("EQ", "wfk_k20_b4_s1"))/3
256
257 k = 0
258 while k < len(h):
259
260     # Transform binary files into .txt files using cut3d
261     use_cut3d("primitiveo_DS"+str(4*k+2)+"_VHXC", "LA", 0, 0)
262     use_cut3d("primitiveo_DS"+str(4*k+3)+"_VHXC", "LO", 0, 0)
263     use_cut3d("primitiveo_DS"+str(4*k+4)+"_VHXC", "TA", 0, 0)
264     use_cut3d("primitiveo_DS"+str(4*k+5)+"_VHXC", "TO", 0, 0)
265
266     # Perform the integration int(psi_0*VHXC*psi_0)
267     averageLA = (integration("LA", "wfk_k20_b2_s1")+
268                 integration("LA", "wfk_k20_b3_s1")+integration("LA", "wfk_k20_b4_s1"))/3
269     averageLO = (integration("LO", "wfk_k20_b2_s1")+
270                 integration("LO", "wfk_k20_b3_s1")+integration("LO", "wfk_k20_b4_s1"))/3
271     averageTA = (integration("TA", "wfk_k20_b2_s1")+
272                 integration("TA", "wfk_k20_b3_s1")+integration("TA", "wfk_k20_b4_s1"))/3
273     averageTO = (integration("TO", "wfk_k20_b2_s1")+
274                 integration("TO", "wfk_k20_b3_s1")+integration("TO", "wfk_k20_b4_s1"))/3
275
276     # Second order derivative (meV)
277     # LA mode
278     D[0,k,0] = 2*(averageLA-averageEQ)/\
279              ((h[k]**2)*(1.0/(4*Mc*omega[0])))*27.211383*1000
280     # LO mode
281     D[0,k,1] = 2*(averageLO-averageEQ)/\
282              ((h[k]**2)*(1.0/(4*Mc*omega[1])))*27.211383*1000
283     # TA mode
284     D[0,k,2] = 2*(averageTA-averageEQ)/\
285              ((h[k]**2)*(1.0/(4*Mc*omega[2])))*27.211383*1000
286     # TO mode
287     D[0,k,3] = 2*(averageTO-averageEQ)/\
288              ((h[k]**2)*(1.0/(4*Mc*omega[3])))*27.211383*1000
289     k += 1
290
291 # -----
292 # We will now use the Richardson Extrapolation to speed up the convergence
293 # -----
294
295 l = 0
296 while l < len(h):
297     k = 0
298     while k < len(h):

```



```

300     if (l-1) >= 0 and (k-1) >= 0:
301     # LA mode
302     D[l,k,0] = (D[l-1,k,0]-(D[l-1,k-1,0]*(1.0/(2**(2*l)))))/\
303             (1-(1.0/(2**(2*l))))
304     if D[l-1,k-1,0] == 0.0:
305         D[l,k,0] = 0.0
306     # LO mode
307     D[l,k,1] = (D[l-1,k,1]-(D[l-1,k-1,1]*(1.0/(2**(2*l)))))/\
308             (1-(1.0/(2**(2*l))))
309     if D[l-1,k-1,1] == 0.0:
310         D[l,k,1] = 0.0
311     # TA mode
312     D[l,k,2] = (D[l-1,k,2]-(D[l-1,k-1,2]*(1.0/(2**(2*l)))))/\
313             (1-(1.0/(2**(2*l))))
314     if D[l-1,k-1,2] == 0.0:
315         D[l,k,2] = 0.0
316     # TO mode
317     D[l,k,3] = (D[l-1,k,3]-(D[l-1,k-1,3]*(1.0/(2**(2*l)))))/\
318             (1-(1.0/(2**(2*l))))
319     if D[l-1,k-1,3] == 0.0:
320         D[l,k,3] = 0.0
321     k += 1
322     l += 1
323
324
325     print "Richardson extrapolation of eigenenergies"
326     print Deigen
327
328     print "Richardson extrapolation of the first term of NDDW"
329     print Dfirst
330
331     print "Richardson extrapolation of the second term of NDDW"
332     print D

```

Code 13: Python scripts use to compute finite difference using the Richardson extrapolation

5.11 Finite diff. at the $q=2/3L$

```

1  # -----
2  # This script generate cartesian coordinate for Abinit input files.
3  # Specifically for q = L = (1/3 0.0 0.0)
4  # Create inputs for finite diff. on eigenenergies and NDDW
5  # -----
6
7  import numpy as np
8
9  # -----
10 # Find the eigenvector and eigenenergies from the dynamical matrix
11 # computed using DFPT in ABINIT
12 # -----
13
14 # Location of the Dynamical matrix file
15 L_file = '/home/Samuel/Dropbox/WorkDiam/2L3_Global/Dynamical_mat.dat'
16 # Carbon mass in a.u.
17 Mc=21894.16693
18
19 def read_dynamical_matrix(datafile):
20     with open(datafile,'r') as f:
21         raw = list()
22         for line in f:
23             if line[0]=='#': continue
24             parts=line.split()
25             if not parts: continue
26             valR=float(parts[4]) #real
27             valI=float(parts[5]) #imaginary
28             if np.abs(valR) < 0.000000001:
29                 valR=0.0
30             if np.abs(valI) < 0.000000001:
31                 valI=0.0
32             raw.append(complex(valR,valI))
33         dynmat = np.array(raw)
34         dynmat = dynmat.reshape(6,6)
35         return dynmat
36
37 dynmat=read_dynamical_matrix(L_file)
38 [eigval,eigvect]=np.linalg.eig(dynmat)
39 eigvect=np.transpose(eigvect)
40

```

```

41 print "eigenvalues at 2/3L :"
42 for n in eigval:
43     a = np.sqrt(n/Mc)
44     print a
45
46 print "eigenvectors at 2/3L :"
47 for n in eigvect: print n
48
49 # -----
50 # Create Abinit input file for finite difference on eigen energies
51 # and the first part of NDDW. On a supercell 3x1x1
52 # -----
53
54 # Value and number of the displacement (always 1/2 of the previous)
55 h = np.array([0.02,0.01,0.005,0.0025,0.00125])
56
57 # Cell parameters
58 acell = np.array([20.110404150000001,6.70346805,6.70346805])
59 rprim = np.array([[0.0,0.5,0.5],[0.5,0.0,0.5],[0.5,0.5,0.0]])
60 cell = np.mat([rprim[0]*acell[0],rprim[1]*acell[1],rprim[2]*acell[2]])
61 xred = np.mat([[0.0,0.0,0.0],[1.0/12,0.25,0.25],[1.0/3,0.0,0.0],\
62 [5.0/12,0.25,0.25],[2.0/3,0.0,0.0],[9.0/12,0.25,0.25]])
63
64
65 # In the specific case of the L point, h = -h
66 # Indice that record the number of xcart in the input file
67 j = 1
68
69 # Equilibrium position
70 print " "
71 print " Here we compute \sum_kk' (d^2e/dRkdRk')Re(Uk)Re(Uk') "
72 print "Contribution of the 2/3L point on the electronic eigenenergies at Gamma"
73 print "and the first part of the NDDW."
74 print "Using finite difference."
75 print " "
76 print " xcart%.0f %.15f %.15f %.15f"%(j,(xred[0]*cell)[0,0],\
77 (xred[0]*cell)[0,1],(xred[0]*cell)[0,2])
78 print " %.15f %.15f %.15f"%(xred[1]*cell)[0,0],\
79 (xred[1]*cell)[0,1],(xred[1]*cell)[0,2])
80 print " %.15f %.15f %.15f"%(xred[2]*cell)[0,0],\
81 (xred[2]*cell)[0,1],(xred[2]*cell)[0,2])
82 print " %.15f %.15f %.15f"%(xred[3]*cell)[0,0],\
83 (xred[3]*cell)[0,1],(xred[3]*cell)[0,2])
84 print " %.15f %.15f %.15f"%(xred[4]*cell)[0,0],\
85 (xred[4]*cell)[0,1],(xred[4]*cell)[0,2])
86 print " %.15f %.15f %.15f"%(xred[5]*cell)[0,0],\
87 (xred[5]*cell)[0,1],(xred[5]*cell)[0,2])
88 print " "
89 j += 1
90
91 # -----
92 # We need to compute \sum_kk' (d^2e/dRkdRk')[Re(Uk)Re(Uk')+Im(Uk)Im(Uk')]
93 # Uk = \xi_k e^{iqRk} = (Re(\xi_k)+Im(\xi_k))(cosqR_k+isingR_k)
94 # Re(Uk) = Re(\xi_k)cosqR_k - Im(\xi_k)sinqR_k
95 # Im(Uk) = Im(\xi_k)cosqR_k + Re(\xi_k)sinqR_k
96 # We need to make a frozen phonon and to compute the quadratic term
97 # -----
98
99 # -----
100 # Here we compute \sum_kk' (d^2e/dRkdRk')Re(Uk)Re(Uk')
101 # -----
102
103 for disp in h:
104     # -----
105     # We have the following phase factor: e^{(q.R)}=cos(2\pi/3)+isin(2\pi/3)
106     # So here for the real part we have:
107     # Atoms 1 and 2 = eq + 1*disp
108     # Atoms 3 and 4 = eq - 0.5*disp
109     # Atoms 5 and 6 = eq - 0.5*disp
110     # For the imaginary part we have:
111     # Atoms 1 and 2 = eq + 0
112     # Atoms 3 and 4 = eq + sqrt(3)/2*disp
113     # Atoms 5 and 6 = eq - sqrt(3)/2*disp
114     # -----
115
116     # -----
117     # LA mode
118     # -----
119     i = 0
120     # Atoms 1 and 2
121     print " xcart%.0f %.15f %.15f %.15f"%(j,(xred[0]*cell)[0,0]+\
122 eigvect.real[i][0]*disp,(xred[0]*cell)[0,1]+eigvect.real[i][1]*disp,\
123 (xred[0]*cell)[0,2]+eigvect.real[i][2]*disp)
124     print " %.15f %.15f %.15f"%(xred[1]*cell)[0,0]+\

```

```

125 eigvect.real[i][3]*disp, (xred[1]*cell)[0,1]+eigvect.real[i][4]*disp,\
126 (xred[1]*cell)[0,2]+eigvect.real[i][5]*disp)
127 # Atoms 3 and 4
128 print " %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
129 0.5*eigvect.real[i][0]*disp - (np.sqrt(3)/2)*eigvect.imag[i][0]*disp,\
130 (xred[2]*cell)[0,1] - 0.5*eigvect.real[i][1]*disp - \
131 (np.sqrt(3)/2)*eigvect.imag[i][1]*disp, (xred[2]*cell)[0,2]-\
132 0.5*eigvect.real[i][2]*disp - (np.sqrt(3)/2)*eigvect.imag[i][2]*disp)
133 print " %.15f %.15f %.15f "%((xred[3]*cell)[0,0]-\
134 0.5*eigvect.real[i][3]*disp - (np.sqrt(3)/2)*eigvect.imag[i][3]*disp,\
135 (xred[3]*cell)[0,1] - 0.5*eigvect.real[i][4]*disp - \
136 (np.sqrt(3)/2)*eigvect.imag[i][4]*disp, (xred[3]*cell)[0,2]-\
137 0.5*eigvect.real[i][5]*disp - (np.sqrt(3)/2)*eigvect.imag[i][5]*disp)
138 # Atoms 5 and 6
139 print " %.15f %.15f %.15f "%((xred[4]*cell)[0,0]-\
140 0.5*eigvect.real[i][0]*disp + (np.sqrt(3)/2)*eigvect.imag[i][0]*disp,\
141 (xred[4]*cell)[0,1] - 0.5*eigvect.real[i][1]*disp + \
142 (np.sqrt(3)/2)*eigvect.imag[i][1]*disp, (xred[4]*cell)[0,2]-\
143 0.5*eigvect.real[i][2]*disp + (np.sqrt(3)/2)*eigvect.imag[i][2]*disp)
144 print " %.15f %.15f %.15f "%((xred[5]*cell)[0,0]-\
145 0.5*eigvect.real[i][3]*disp + (np.sqrt(3)/2)*eigvect.imag[i][3]*disp,\
146 (xred[5]*cell)[0,1] - 0.5*eigvect.real[i][4]*disp + \
147 (np.sqrt(3)/2)*eigvect.imag[i][4]*disp, (xred[5]*cell)[0,2]-\
148 0.5*eigvect.real[i][5]*disp + (np.sqrt(3)/2)*eigvect.imag[i][5]*disp)
149 print " "
150 j += 1
151 # -----
152 # L0 mode
153 # -----
154 i = 1
155 # Atoms 1 and 2
156 print " xcart%.0f %.15f %.15f %.15f "%(j, (xred[0]*cell)[0,0]+\
157 eigvect.real[i][0]*disp, (xred[0]*cell)[0,1]+eigvect.real[i][1]*disp,\
158 (xred[0]*cell)[0,2]+eigvect.real[i][2]*disp)
159 print " %.15f %.15f %.15f "%((xred[1]*cell)[0,0]+\
160 eigvect.real[i][3]*disp, (xred[1]*cell)[0,1]+eigvect.real[i][4]*disp,\
161 (xred[1]*cell)[0,2]+eigvect.real[i][5]*disp)
162 # Atoms 3 and 4
163 print " %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
164 0.5*eigvect.real[i][0]*disp - (np.sqrt(3)/2)*eigvect.imag[i][0]*disp,\
165 (xred[2]*cell)[0,1] - 0.5*eigvect.real[i][1]*disp - \
166 (np.sqrt(3)/2)*eigvect.imag[i][1]*disp, (xred[2]*cell)[0,2]-\
167 0.5*eigvect.real[i][2]*disp - (np.sqrt(3)/2)*eigvect.imag[i][2]*disp)
168 print " %.15f %.15f %.15f "%((xred[3]*cell)[0,0]-\
169 0.5*eigvect.real[i][3]*disp - (np.sqrt(3)/2)*eigvect.imag[i][3]*disp,\
170 (xred[3]*cell)[0,1] - 0.5*eigvect.real[i][4]*disp - \
171 (np.sqrt(3)/2)*eigvect.imag[i][4]*disp, (xred[3]*cell)[0,2]-\
172 0.5*eigvect.real[i][5]*disp - (np.sqrt(3)/2)*eigvect.imag[i][5]*disp)
173 # Atoms 5 and 6
174 print " %.15f %.15f %.15f "%((xred[4]*cell)[0,0]-\
175 0.5*eigvect.real[i][0]*disp + (np.sqrt(3)/2)*eigvect.imag[i][0]*disp,\
176 (xred[4]*cell)[0,1] - 0.5*eigvect.real[i][1]*disp + \
177 (np.sqrt(3)/2)*eigvect.imag[i][1]*disp, (xred[4]*cell)[0,2]-\
178 0.5*eigvect.real[i][2]*disp + (np.sqrt(3)/2)*eigvect.imag[i][2]*disp)
179 print " %.15f %.15f %.15f "%((xred[5]*cell)[0,0]-\
180 0.5*eigvect.real[i][3]*disp + (np.sqrt(3)/2)*eigvect.imag[i][3]*disp,\
181 (xred[5]*cell)[0,1] - 0.5*eigvect.real[i][4]*disp + \
182 (np.sqrt(3)/2)*eigvect.imag[i][4]*disp, (xred[5]*cell)[0,2]-\
183 0.5*eigvect.real[i][5]*disp + (np.sqrt(3)/2)*eigvect.imag[i][5]*disp)
184 print " "
185 j += 1
186 # -----
187 # TA mode
188 # -----
189 i = 2
190 # Atoms 1 and 2
191 print " xcart%.0f %.15f %.15f %.15f "%(j, (xred[0]*cell)[0,0]+\
192 eigvect.real[i][0]*disp, (xred[0]*cell)[0,1]+eigvect.real[i][1]*disp,\
193 (xred[0]*cell)[0,2]+eigvect.real[i][2]*disp)
194 print " %.15f %.15f %.15f "%((xred[1]*cell)[0,0]+\
195 eigvect.real[i][3]*disp, (xred[1]*cell)[0,1]+eigvect.real[i][4]*disp,\
196 (xred[1]*cell)[0,2]+eigvect.real[i][5]*disp)
197 # Atoms 3 and 4
198 print " %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
199 0.5*eigvect.real[i][0]*disp - (np.sqrt(3)/2)*eigvect.imag[i][0]*disp,\
200 (xred[2]*cell)[0,1] - 0.5*eigvect.real[i][1]*disp - \
201 (np.sqrt(3)/2)*eigvect.imag[i][1]*disp, (xred[2]*cell)[0,2]-\
202 0.5*eigvect.real[i][2]*disp - (np.sqrt(3)/2)*eigvect.imag[i][2]*disp)
203 print " %.15f %.15f %.15f "%((xred[3]*cell)[0,0]-\
204 0.5*eigvect.real[i][3]*disp - (np.sqrt(3)/2)*eigvect.imag[i][3]*disp,\
205 (xred[3]*cell)[0,1] - 0.5*eigvect.real[i][4]*disp - \
206 (np.sqrt(3)/2)*eigvect.imag[i][4]*disp, (xred[3]*cell)[0,2]-\
207 0.5*eigvect.real[i][5]*disp - (np.sqrt(3)/2)*eigvect.imag[i][5]*disp)
208 # Atoms 5 and 6

```

```

209 print "          %.15f %.15f %.15f "%((xred[4]*cell)[0,0]-\
210 0.5*eigvect.real[i][0]*disp + (np.sqrt(3)/2)*eigvect.imag[i][0]*disp,\
211 (xred[4]*cell)[0,1] - 0.5*eigvect.real[i][1]*disp + \
212 (np.sqrt(3)/2)*eigvect.imag[i][1]*disp, (xred[4]*cell)[0,2]-\
213 0.5*eigvect.real[i][2]*disp + (np.sqrt(3)/2)*eigvect.imag[i][2]*disp)
214 print "          %.15f %.15f %.15f "%((xred[5]*cell)[0,0]-\
215 0.5*eigvect.real[i][3]*disp + (np.sqrt(3)/2)*eigvect.imag[i][3]*disp,\
216 (xred[5]*cell)[0,1] - 0.5*eigvect.real[i][4]*disp + \
217 (np.sqrt(3)/2)*eigvect.imag[i][4]*disp, (xred[5]*cell)[0,2]-\
218 0.5*eigvect.real[i][5]*disp + (np.sqrt(3)/2)*eigvect.imag[i][5]*disp)
219 print " "
220 j += 1
221 # -----
222 # T0 mode
223 # -----
224 i = 4
225 # Atoms 1 and 2
226 print " xcart%.0f %.15f %.15f %.15f "%(j, (xred[0]*cell)[0,0]+\
227 eigvect.real[i][0]*disp, (xred[0]*cell)[0,1]+eigvect.real[i][1]*disp,\
228 (xred[0]*cell)[0,2]+eigvect.real[i][2]*disp)
229 print "          %.15f %.15f %.15f "%((xred[1]*cell)[0,0]+\
230 eigvect.real[i][3]*disp, (xred[1]*cell)[0,1]+eigvect.real[i][4]*disp,\
231 (xred[1]*cell)[0,2]+eigvect.real[i][5]*disp)
232 # Atoms 3 and 4
233 print "          %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
234 0.5*eigvect.real[i][0]*disp - (np.sqrt(3)/2)*eigvect.imag[i][0]*disp,\
235 (xred[2]*cell)[0,1] - 0.5*eigvect.real[i][1]*disp - \
236 (np.sqrt(3)/2)*eigvect.imag[i][1]*disp, (xred[2]*cell)[0,2]-\
237 0.5*eigvect.real[i][2]*disp - (np.sqrt(3)/2)*eigvect.imag[i][2]*disp)
238 print "          %.15f %.15f %.15f "%((xred[3]*cell)[0,0]-\
239 0.5*eigvect.real[i][3]*disp - (np.sqrt(3)/2)*eigvect.imag[i][3]*disp,\
240 (xred[3]*cell)[0,1] - 0.5*eigvect.real[i][4]*disp - \
241 (np.sqrt(3)/2)*eigvect.imag[i][4]*disp, (xred[3]*cell)[0,2]-\
242 0.5*eigvect.real[i][5]*disp - (np.sqrt(3)/2)*eigvect.imag[i][5]*disp)
243 # Atoms 5 and 6
244 print "          %.15f %.15f %.15f "%((xred[4]*cell)[0,0]-\
245 0.5*eigvect.real[i][0]*disp + (np.sqrt(3)/2)*eigvect.imag[i][0]*disp,\
246 (xred[4]*cell)[0,1] - 0.5*eigvect.real[i][1]*disp + \
247 (np.sqrt(3)/2)*eigvect.imag[i][1]*disp, (xred[4]*cell)[0,2]-\
248 0.5*eigvect.real[i][2]*disp + (np.sqrt(3)/2)*eigvect.imag[i][2]*disp)
249 print "          %.15f %.15f %.15f "%((xred[5]*cell)[0,0]-\
250 0.5*eigvect.real[i][3]*disp + (np.sqrt(3)/2)*eigvect.imag[i][3]*disp,\
251 (xred[5]*cell)[0,1] - 0.5*eigvect.real[i][4]*disp + \
252 (np.sqrt(3)/2)*eigvect.imag[i][4]*disp, (xred[5]*cell)[0,2]-\
253 0.5*eigvect.real[i][5]*disp + (np.sqrt(3)/2)*eigvect.imag[i][5]*disp)
254 print " "
255 j += 1
256
257 # -----
258 # Second part of the NDDW on a primitive cell
259 # We only compute 6xi=(3xi_1,3xi_1) as eigen displacement
260 # and then multiply by 2 because 6xi=(3xi_2,3xi_2) should be the same.
261 # -----
262
263 acell = np.array([6.70346805,6.70346805,6.70346805])
264 cell = np.mat([rprim[0]*acell[0],rprim[1]*acell[1],rprim[2]*acell[2]])
265 xred = np.mat([[0.0,0.0,0.0],[0.25,0.25,0.25]])
266
267 print " "
268 print "Contribution of the L point on the electronic eigenenergies at Gamma"
269 print "and the second part of the NDDW."
270 print "Using finite difference."
271 print "Real eigenvector "
272 print " "
273
274 j = 1
275 # Equilibrium position
276 print " xcart%.0f %.10f %.10f %.10f "%(j, (xred[0]*cell)[0,0],\
277 (xred[0]*cell)[0,1],(xred[0]*cell)[0,2])
278 print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0],\
279 (xred[1]*cell)[0,1],(xred[1]*cell)[0,2])
280 print " "
281 j += 1
282
283 for disp in h:
284 # -----
285 # LA mode
286 # -----
287 i = 0
288 print " xcart%.0f %.10f %.10f %.10f "%(j, (xred[0]*cell)[0,0]\
289 +eigvect.real[i][0]*disp, (xred[0]*cell)[0,1]+\
290 eigvect.real[i][1]*disp, (xred[0]*cell)[0,2]+\
291 eigvect.real[i][2]*disp)
292 print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]\

```

```

293         +eigvect.real[i][0]*disp, (xred[1]*cell)[0,1]+\
294         eigvect.real[i][1]*disp, (xred[1]*cell)[0,2]+\
295         eigvect.real[i][2]*disp)
296     print " "
297     j += 1
298     # -----
299     # L0 mode
300     # -----
301     i = 1
302     print " xcart%.0f %.10f %.10f %.10f"%(j, (xred[0]*cell)[0,0]\
303         +eigvect.real[i][0]*disp, (xred[0]*cell)[0,1]+\
304         eigvect.real[i][1]*disp, (xred[0]*cell)[0,2]+\
305         eigvect.real[i][2]*disp)
306     print "          %.10f %.10f %.10f"%( (xred[1]*cell)[0,0]\
307         +eigvect.real[i][0]*disp, (xred[1]*cell)[0,1]+\
308         eigvect.real[i][1]*disp, (xred[1]*cell)[0,2]+\
309         eigvect.real[i][2]*disp)
310     print " "
311     j += 1
312     # -----
313     # TA mode
314     # -----
315     i = 2
316     print " xcart%.0f %.10f %.10f %.10f"%(j, (xred[0]*cell)[0,0]\
317         +eigvect.real[i][0]*disp, (xred[0]*cell)[0,1]+\
318         eigvect.real[i][1]*disp, (xred[0]*cell)[0,2]+\
319         eigvect.real[i][2]*disp)
320     print "          %.10f %.10f %.10f"%( (xred[1]*cell)[0,0]\
321         +eigvect.real[i][0]*disp, (xred[1]*cell)[0,1]+\
322         eigvect.real[i][1]*disp, (xred[1]*cell)[0,2]+\
323         eigvect.real[i][2]*disp)
324     print " "
325     j += 1
326     # -----
327     # T0 mode
328     # -----
329     i = 4
330     print " xcart%.0f %.10f %.10f %.10f"%(j, (xred[0]*cell)[0,0]\
331         +eigvect.real[i][0]*disp, (xred[0]*cell)[0,1]+\
332         eigvect.real[i][1]*disp, (xred[0]*cell)[0,2]+\
333         eigvect.real[i][2]*disp)
334     print "          %.10f %.10f %.10f"%( (xred[1]*cell)[0,0]\
335         +eigvect.real[i][0]*disp, (xred[1]*cell)[0,1]+\
336         eigvect.real[i][1]*disp, (xred[1]*cell)[0,2]+\
337         eigvect.real[i][2]*disp)
338     print " "
339     j += 1
340
341     print ''
342     print ' -----'
343     print ' IMAGINARY PART '
344     print ' -----'
345     print ''
346
347     # -----
348     # Here we compute \sum_kk' (d^2e/dRkdRk')Im(Uk)Im(Uk')
349     # -----
350
351     # Cell parameters
352     acell = np.array([20.110404150000001, 6.70346805, 6.70346805])
353     rprim = np.array([[0.0, 0.5, 0.5], [0.5, 0.0, 0.5], [0.5, 0.5, 0.0]])
354     cell = np.mat([rprim[0]*acell[0], rprim[1]*acell[1], rprim[2]*acell[2]])
355     xred = np.mat([[0.0, 0.0, 0.0], [1.0/12, 0.25, 0.25], [1.0/3, 0.0, 0.0], \
356     [5.0/12, 0.25, 0.25], [2.0/3, 0.0, 0.0], [9.0/12, 0.25, 0.25]])
357
358     # Indice that record the number of xcart in the input file
359     j = 1
360
361     # Equilibrium position
362     print " "
363     print "Here we compute \sum_kk' (d^2e/dRkdRk')Im(Uk)Im(Uk') "
364     print "Contribution of the 2/3L point on the electronic eigenenergies at Gamma"
365     print "and the first part of the NDDW."
366     print "Using finite difference."
367     print " "
368     print " xcart%.0f %.15f %.15f %.15f"%(j, (xred[0]*cell)[0,0], \
369         (xred[0]*cell)[0,1], (xred[0]*cell)[0,2])
370     print "          %.15f %.15f %.15f"%( (xred[1]*cell)[0,0], \
371         (xred[1]*cell)[0,1], (xred[1]*cell)[0,2])
372     print "          %.15f %.15f %.15f"%( (xred[2]*cell)[0,0], \
373         (xred[2]*cell)[0,1], (xred[2]*cell)[0,2])
374     print "          %.15f %.15f %.15f"%( (xred[3]*cell)[0,0], \
375         (xred[3]*cell)[0,1], (xred[3]*cell)[0,2])

```

```

377 print "          %.15f %.15f %.15f "%((xred[4]*cell)[0,0],\
378 (xred[4]*cell)[0,1],(xred[4]*cell)[0,2])
379 print "          %.15f %.15f %.15f "%((xred[5]*cell)[0,0],\
380 (xred[5]*cell)[0,1],(xred[5]*cell)[0,2])
381 print " "
382 j += 1
383
384 # -----
385 #hh=0
386 #while hh < 6:
387 #   g =0
388 #   while g < 6:
389 #       if abs(eigvect.imag[hh][g]) < 0.0001:
390 #           eigvect.imag[hh][g] = 0.0
391 #       g += 1
392 #       hh += 1
393 #
394 #print eigvect
395 # -----
396
397 for disp in h:
398 # -----
399 # We have the following phase factor:  $e^{-i(q \cdot R)} = \cos(2\pi/3) + i\sin(2\pi/3)$ 
400 # So here for the real part we have:
401 # Atoms 1 and 2 = eq + 1*disp
402 # Atoms 3 and 4 = eq - 0.5*disp
403 # Atoms 5 and 6 = eq - 0.5*disp
404 # For the imaginary part we have:
405 # Atoms 1 and 2 = eq + 0
406 # Atoms 3 and 4 = eq + sqrt(3)/2*disp
407 # Atoms 5 and 6 = eq - sqrt(3)/2*disp
408 # -----
409
410 # -----
411 # LA mode
412 # -----
413 i = 0
414 # Atoms 1 and 2
415 print " xcart%.Of %.15f %.15f %.15f "%(j,(xred[0]*cell)[0,0]+\
416 eigvect.imag[i][0]*disp,(xred[0]*cell)[0,1]+eigvect.imag[i][1]*disp,\
417 (xred[0]*cell)[0,2]+eigvect.imag[i][2]*disp)
418 print "          %.15f %.15f %.15f "%((xred[1]*cell)[0,0]+\
419 eigvect.imag[i][3]*disp,(xred[1]*cell)[0,1]+eigvect.imag[i][4]*disp,\
420 (xred[1]*cell)[0,2]+eigvect.imag[i][5]*disp)
421 # Atoms 3 and 4
422 print "          %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
423 0.5*eigvect.imag[i][0]*disp + (np.sqrt(3)/2)*eigvect.real[i][0]*disp,\
424 (xred[2]*cell)[0,1] - 0.5*eigvect.imag[i][1]*disp + \
425 (np.sqrt(3)/2)*eigvect.real[i][1]*disp, (xred[2]*cell)[0,2]-\
426 0.5*eigvect.imag[i][2]*disp + (np.sqrt(3)/2)*eigvect.real[i][2]*disp)
427 print "          %.15f %.15f %.15f "%((xred[3]*cell)[0,0]-\
428 0.5*eigvect.imag[i][3]*disp + (np.sqrt(3)/2)*eigvect.real[i][3]*disp,\
429 (xred[3]*cell)[0,1] - 0.5*eigvect.imag[i][4]*disp + \
430 (np.sqrt(3)/2)*eigvect.real[i][4]*disp, (xred[3]*cell)[0,2]-\
431 0.5*eigvect.imag[i][5]*disp + (np.sqrt(3)/2)*eigvect.real[i][5]*disp)
432 # Atoms 5 and 6
433 print "          %.15f %.15f %.15f "%((xred[4]*cell)[0,0]-\
434 0.5*eigvect.imag[i][0]*disp - (np.sqrt(3)/2)*eigvect.real[i][0]*disp,\
435 (xred[4]*cell)[0,1] - 0.5*eigvect.imag[i][1]*disp - \
436 (np.sqrt(3)/2)*eigvect.real[i][1]*disp, (xred[4]*cell)[0,2]-\
437 0.5*eigvect.imag[i][2]*disp - (np.sqrt(3)/2)*eigvect.real[i][2]*disp)
438 print "          %.15f %.15f %.15f "%((xred[5]*cell)[0,0]-\
439 0.5*eigvect.imag[i][3]*disp - (np.sqrt(3)/2)*eigvect.real[i][3]*disp,\
440 (xred[5]*cell)[0,1] - 0.5*eigvect.imag[i][4]*disp - \
441 (np.sqrt(3)/2)*eigvect.real[i][4]*disp, (xred[5]*cell)[0,2]-\
442 0.5*eigvect.imag[i][5]*disp - (np.sqrt(3)/2)*eigvect.real[i][5]*disp)
443 print " "
444 j += 1
445 # L0 mode
446 i = 1
447 # Atoms 1 and 2
448 print " xcart%.Of %.15f %.15f %.15f "%(j,(xred[0]*cell)[0,0]+\
449 eigvect.imag[i][0]*disp,(xred[0]*cell)[0,1]+eigvect.imag[i][1]*disp,\
450 (xred[0]*cell)[0,2]+eigvect.imag[i][2]*disp)
451 print "          %.15f %.15f %.15f "%((xred[1]*cell)[0,0]+\
452 eigvect.imag[i][3]*disp,(xred[1]*cell)[0,1]+eigvect.imag[i][4]*disp,\
453 (xred[1]*cell)[0,2]+eigvect.imag[i][5]*disp)
454 # Atoms 3 and 4
455 print "          %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
456 0.5*eigvect.imag[i][0]*disp + (np.sqrt(3)/2)*eigvect.real[i][0]*disp,\
457 (xred[2]*cell)[0,1] - 0.5*eigvect.imag[i][1]*disp + \
458 (np.sqrt(3)/2)*eigvect.real[i][1]*disp, (xred[2]*cell)[0,2]-\
459 0.5*eigvect.imag[i][2]*disp + (np.sqrt(3)/2)*eigvect.real[i][2]*disp)
460 print "          %.15f %.15f %.15f "%((xred[3]*cell)[0,0]-\

```

```

461     0.5*eigvect.imag[i][3]*disp + (np.sqrt(3)/2)*eigvect.real[i][3]*disp,\
462     (xred[3]*cell)[0,1] - 0.5*eigvect.imag[i][4]*disp + \
463     (np.sqrt(3)/2)*eigvect.real[i][4]*disp, (xred[3]*cell)[0,2]-\
464     0.5*eigvect.imag[i][5]*disp + (np.sqrt(3)/2)*eigvect.real[i][5]*disp)
465 # Atoms 5 and 6
466 print "      %.15f %.15f %.15f "%((xred[4]*cell)[0,0]-\
467     0.5*eigvect.imag[i][0]*disp - (np.sqrt(3)/2)*eigvect.real[i][0]*disp,\
468     (xred[4]*cell)[0,1] - 0.5*eigvect.imag[i][1]*disp - \
469     (np.sqrt(3)/2)*eigvect.real[i][1]*disp, (xred[4]*cell)[0,2]-\
470     0.5*eigvect.imag[i][2]*disp - (np.sqrt(3)/2)*eigvect.real[i][2]*disp)
471 print "      %.15f %.15f %.15f "%((xred[5]*cell)[0,0]-\
472     0.5*eigvect.imag[i][3]*disp - (np.sqrt(3)/2)*eigvect.real[i][3]*disp,\
473     (xred[5]*cell)[0,1] - 0.5*eigvect.imag[i][4]*disp - \
474     (np.sqrt(3)/2)*eigvect.real[i][4]*disp, (xred[5]*cell)[0,2]-\
475     0.5*eigvect.imag[i][5]*disp - (np.sqrt(3)/2)*eigvect.real[i][5]*disp)
476 print " "
477 j += 1
478 # TA mode
479 i = 2
480 # Atoms 1 and 2
481 print " xcart%.Of %.15f %.15f %.15f "%(j, (xred[0]*cell)[0,0]+\
482     eigvect.imag[i][0]*disp, (xred[0]*cell)[0,1]+eigvect.imag[i][1]*disp,\
483     (xred[0]*cell)[0,2]+eigvect.imag[i][2]*disp)
484 print "      %.15f %.15f %.15f "%((xred[1]*cell)[0,0]+\
485     eigvect.imag[i][3]*disp, (xred[1]*cell)[0,1]+eigvect.imag[i][4]*disp,\
486     (xred[1]*cell)[0,2]+eigvect.imag[i][5]*disp)
487 # Atoms 3 and 4
488 print "      %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
489     0.5*eigvect.imag[i][0]*disp + (np.sqrt(3)/2)*eigvect.real[i][0]*disp,\
490     (xred[2]*cell)[0,1] - 0.5*eigvect.imag[i][1]*disp + \
491     (np.sqrt(3)/2)*eigvect.real[i][1]*disp, (xred[2]*cell)[0,2]-\
492     0.5*eigvect.imag[i][2]*disp + (np.sqrt(3)/2)*eigvect.real[i][2]*disp)
493 print "      %.15f %.15f %.15f "%((xred[3]*cell)[0,0]-\
494     0.5*eigvect.imag[i][3]*disp + (np.sqrt(3)/2)*eigvect.real[i][3]*disp,\
495     (xred[3]*cell)[0,1] - 0.5*eigvect.imag[i][4]*disp + \
496     (np.sqrt(3)/2)*eigvect.real[i][4]*disp, (xred[3]*cell)[0,2]-\
497     0.5*eigvect.imag[i][5]*disp + (np.sqrt(3)/2)*eigvect.real[i][5]*disp)
498 # Atoms 5 and 6
499 print "      %.15f %.15f %.15f "%((xred[4]*cell)[0,0]-\
500     0.5*eigvect.imag[i][0]*disp - (np.sqrt(3)/2)*eigvect.real[i][0]*disp,\
501     (xred[4]*cell)[0,1] - 0.5*eigvect.imag[i][1]*disp - \
502     (np.sqrt(3)/2)*eigvect.real[i][1]*disp, (xred[4]*cell)[0,2]-\
503     0.5*eigvect.imag[i][2]*disp - (np.sqrt(3)/2)*eigvect.real[i][2]*disp)
504 print "      %.15f %.15f %.15f "%((xred[5]*cell)[0,0]-\
505     0.5*eigvect.imag[i][3]*disp - (np.sqrt(3)/2)*eigvect.real[i][3]*disp,\
506     (xred[5]*cell)[0,1] - 0.5*eigvect.imag[i][4]*disp - \
507     (np.sqrt(3)/2)*eigvect.real[i][4]*disp, (xred[5]*cell)[0,2]-\
508     0.5*eigvect.imag[i][5]*disp - (np.sqrt(3)/2)*eigvect.real[i][5]*disp)
509 print " "
510 j += 1
511 # T0 mode
512 i = 4
513 # Atoms 1 and 2
514 print " xcart%.Of %.15f %.15f %.15f "%(j, (xred[0]*cell)[0,0]+\
515     eigvect.imag[i][0]*disp, (xred[0]*cell)[0,1]+eigvect.imag[i][1]*disp,\
516     (xred[0]*cell)[0,2]+eigvect.imag[i][2]*disp)
517 print "      %.15f %.15f %.15f "%((xred[1]*cell)[0,0]+\
518     eigvect.imag[i][3]*disp, (xred[1]*cell)[0,1]+eigvect.imag[i][4]*disp,\
519     (xred[1]*cell)[0,2]+eigvect.imag[i][5]*disp)
520 # Atoms 3 and 4
521 print "      %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
522     0.5*eigvect.imag[i][0]*disp + (np.sqrt(3)/2)*eigvect.real[i][0]*disp,\
523     (xred[2]*cell)[0,1] - 0.5*eigvect.imag[i][1]*disp + \
524     (np.sqrt(3)/2)*eigvect.real[i][1]*disp, (xred[2]*cell)[0,2]-\
525     0.5*eigvect.imag[i][2]*disp + (np.sqrt(3)/2)*eigvect.real[i][2]*disp)
526 print "      %.15f %.15f %.15f "%((xred[3]*cell)[0,0]-\
527     0.5*eigvect.imag[i][3]*disp + (np.sqrt(3)/2)*eigvect.real[i][3]*disp,\
528     (xred[3]*cell)[0,1] - 0.5*eigvect.imag[i][4]*disp + \
529     (np.sqrt(3)/2)*eigvect.real[i][4]*disp, (xred[3]*cell)[0,2]-\
530     0.5*eigvect.imag[i][5]*disp + (np.sqrt(3)/2)*eigvect.real[i][5]*disp)
531 # Atoms 5 and 6
532 print "      %.15f %.15f %.15f "%((xred[4]*cell)[0,0]-\
533     0.5*eigvect.imag[i][0]*disp - (np.sqrt(3)/2)*eigvect.real[i][0]*disp,\
534     (xred[4]*cell)[0,1] - 0.5*eigvect.imag[i][1]*disp - \
535     (np.sqrt(3)/2)*eigvect.real[i][1]*disp, (xred[4]*cell)[0,2]-\
536     0.5*eigvect.imag[i][2]*disp - (np.sqrt(3)/2)*eigvect.real[i][2]*disp)
537 print "      %.15f %.15f %.15f "%((xred[5]*cell)[0,0]-\
538     0.5*eigvect.imag[i][3]*disp - (np.sqrt(3)/2)*eigvect.real[i][3]*disp,\
539     (xred[5]*cell)[0,1] - 0.5*eigvect.imag[i][4]*disp - \
540     (np.sqrt(3)/2)*eigvect.real[i][4]*disp, (xred[5]*cell)[0,2]-\
541     0.5*eigvect.imag[i][5]*disp - (np.sqrt(3)/2)*eigvect.real[i][5]*disp)
542 print " "
543 j += 1
544

```

```

545
546
547 # -----
548 # Second part of the NDDW on a primitive cell
549 # We only compute 6xi=(3xi_1,3xi_1) as eigen displacement
550 # and then multiply by 2 because 6xi=(3xi_2,3xi_2) should be the same.
551 # -----
552
553 acell = np.array([6.70346805,6.70346805,6.70346805])
554 cell = np.mat([rprim[0]*acell[0],rprim[1]*acell[1],rprim[2]*acell[2]])
555 xred = np.mat([[0.0,0.0,0.0],[0.25,0.25,0.25]])
556
557 print " "
558 print "Contribution of the L point on the electronic eigenenergies at Gamma"
559 print "and the second part of the NDDW."
560 print "Using finite difference."
561 print "Imaginary eigenvector "
562 print " "
563
564 j = 1
565 # Equilibrium position
566 print " xcart%.0f %.10f %.10f %.10f"%(j,(xred[0]*cell)[0,0],\
567      (xred[0]*cell)[0,1],(xred[0]*cell)[0,2])
568 print "      %.10f %.10f %.10f"%(xred[1]*cell)[0,0],\
569      (xred[1]*cell)[0,1],(xred[1]*cell)[0,2])
570 print " "
571 j += 1
572
573 for disp in h:
574 # -----
575 # LA mode
576 # -----
577 i = 0
578 print " xcart%.0f %.10f %.10f %.10f"%(j,(xred[0]*cell)[0,0]\
579      +eigvect.imag[i][0]*disp, (xred[0]*cell)[0,1]+\
580      eigvect.imag[i][1]*disp, (xred[0]*cell)[0,2]+\
581      eigvect.imag[i][2]*disp)
582 print "      %.10f %.10f %.10f"%(xred[1]*cell)[0,0]\
583      +eigvect.real[i][0]*disp+eigvect.imag[i][0]*disp, (xred[1]*cell)[0,1]+\
584      eigvect.imag[i][1]*disp, (xred[1]*cell)[0,2]+\
585      eigvect.imag[i][2]*disp)
586 print " "
587 j += 1
588 # -----
589 # L0 mode
590 # -----
591 i = 1
592 print " xcart%.0f %.10f %.10f %.10f"%(j,(xred[0]*cell)[0,0]\
593      +eigvect.imag[i][0]*disp, (xred[0]*cell)[0,1]+\
594      eigvect.imag[i][1]*disp, (xred[0]*cell)[0,2]+\
595      eigvect.imag[i][2]*disp)
596 print "      %.10f %.10f %.10f"%(xred[1]*cell)[0,0]\
597      +eigvect.imag[i][0]*disp, (xred[1]*cell)[0,1]+\
598      eigvect.imag[i][1]*disp, (xred[1]*cell)[0,2]+\
599      eigvect.imag[i][2]*disp)
600 print " "
601 j += 1
602 # -----
603 # TA mode
604 # -----
605 i = 2
606 print " xcart%.0f %.10f %.10f %.10f"%(j,(xred[0]*cell)[0,0]\
607      +eigvect.imag[i][0]*disp, (xred[0]*cell)[0,1]+\
608      eigvect.imag[i][1]*disp, (xred[0]*cell)[0,2]+\
609      eigvect.imag[i][2]*disp)
610 print "      %.10f %.10f %.10f"%(xred[1]*cell)[0,0]\
611      +eigvect.imag[i][0]*disp, (xred[1]*cell)[0,1]+\
612      eigvect.imag[i][1]*disp, (xred[1]*cell)[0,2]+\
613      eigvect.imag[i][2]*disp)
614 print " "
615 j += 1
616 # -----
617 # T0 mode
618 # -----
619 i = 4
620 print " xcart%.0f %.10f %.10f %.10f"%(j,(xred[0]*cell)[0,0]\
621      +eigvect.imag[i][0]*disp, (xred[0]*cell)[0,1]+\
622      eigvect.imag[i][1]*disp, (xred[0]*cell)[0,2]+\
623      eigvect.imag[i][2]*disp)
624 print "      %.10f %.10f %.10f"%(xred[1]*cell)[0,0]\
625      +eigvect.imag[i][0]*disp, (xred[1]*cell)[0,1]+\
626      eigvect.imag[i][1]*disp, (xred[1]*cell)[0,2]+\
627      eigvect.imag[i][2]*disp)
628 print " "

```


Code 14: Python scripts use to create ABINIT input files

5.12 Finite diff. at the $q=2/3L$

```

1  # -----
2  # This python script is used to compute the contribution of the
3  # L point to the Gamma eigenenergies using finite difference.
4  # -----
5  import os
6  import numpy as np
7
8  def read_eigenenergies(datafile):
9      flag1 = False
10     flag2 = False
11     with open(datafile, 'r') as f:
12         for line in f:
13             if flag2:
14                 eigenenergies = np.append(eigenenergies, [map(float, line.split())])
15                 break
16             if flag1:
17                 eigenenergies = map(float, line.split())
18                 flag2 = True
19                 if "kpt= 0.0000 0.0000 0.0000" in line:
20                     flag1 = True
21     return eigenenergies
22
23 def read_totalenergies(datafile):
24     etotal = [(0.0)]
25     with open(datafile, 'r') as f:
26         for line in f:
27             if "etotal" in line:
28                 etot = line.split()
29                 if etot[0] == 0:
30                     etotal[0] = etot[1]
31             else:
32                 etotal = np.append(etotal, etot[1])
33     return etotal
34
35 a = read_totalenergies("supercell1R.outD")
36 print a
37
38 def use_cut3d(datafile, out, gamma, band):
39     if datafile[len(datafile)-4:len(datafile)] == "VHXC":
40         os.system("rm cut3d.files")
41         with open("cut3d.files", "a") as files:
42             files.write(str(datafile)+"\n")
43             files.write("1 \n")
44             files.write("5 \n") #3D formatted data
45             # (output the bare 3D data - one column)
46             files.write(str(out)+"\n")
47             files.write("0 \n")
48         os.system("cut3d < cut3d.files ")
49     if datafile[len(datafile)-3:len(datafile)] == "WFK":
50         os.system("rm wfk.files")
51         with open("wfk.files", "a") as files:
52             files.write(str(datafile)+"\n")
53             files.write("1 \n")
54             files.write("0 \n")
55             files.write(str(gamma)+" \n") # Gamma point
56             files.write(str(band)+" \n") # Band nb 6
57             files.write("0 \n")
58             files.write("0 \n")
59             files.write("2 \n") # real 3D data one column
60             files.write(str(out)+" \n")
61             files.write("0 \n")
62         os.system("cut3d < wfk.files ")
63
64 def integration(VHXC, wfk):
65     psi_file=open(wfk, 'r')
66     v_file=open(VHXC, 'r')
67     S=0.0; N=0.0
68     while True:
69         psi_l=psi_file.readline()
70         v_l=v_file.readline()
71         if not psi_l or not v_l: break
72     psi=float(psi_l)

```

```

73     v=float(v_l)
74     S+=psi*v*psi
75     N+=psi*psi
76     psi_file.close()
77     v_file.close()
78     S=S/N
79     return S
80
81     # The base name for the supercell files are assumed to be named "supercell"
82     # and the primitive as assumed to be named "primitive"
83
84     # Output file name
85     outputfile = "supercellR.outD"
86
87     # Import the total of the equilibrium state in Ha (same for imag)
88     totalenergyEQ = float(read_totalenergies(outputfile)[0])/3
89
90     # Value and number of the displacement
91     h = np.array([0.02,0.01,0.005,0.0025,0.00125])
92
93     # -----
94     # Computation of the frequency using total energies
95     # -----
96
97     # Initialisation values
98     k = 0
99
100    # The second dimension is the number of h in geometric progression
101    # h,h/2,h/4,h/8 etc..
102    # The third index correspond to modes (here 4)
103    # The first index correspond to the Richardson extrapolation
104    D = np.zeros([len(h),len(h),4])
105
106    # Frequency (computed in DFPT using ABINIT)
107    omega = np.array([0.003915559,0.006088587,0.002163719,0.00589085])
108
109    # Carbon mass in a.u.
110    Mc = 21894.16693
111
112    while k < len(h):
113
114        # LA mode
115        totalenergyRLA = float(read_totalenergies(outputfile)[4*k+1])/3
116        # LO mode
117        totalenergyRLO = float(read_totalenergies(outputfile)[4*k+2])/3
118        # TA mode
119        totalenergyRTA = float(read_totalenergies(outputfile)[4*k+3])/3
120        # TO mode
121        totalenergyRTO = float(read_totalenergies(outputfile)[4*k+4])/3
122        # LA mode
123        totalenergyILA = float(read_totalenergies(outputfile)[4*k+1])/3
124        # LO mode
125        totalenergyILO = float(read_totalenergies(outputfile)[4*k+2])/3
126        # TA mode
127        totalenergyITA = float(read_totalenergies(outputfile)[4*k+3])/3
128        # TO mode
129        totalenergyITO = float(read_totalenergies(outputfile)[4*k+4])/3
130
131        # Second order derivative (meV)
132        # LA mode
133        D[0,k,0] = np.sqrt(2*(totalenergyRLA - totalenergyEQ)/\
134            (h[k]**2)*(1.0/(Mc)) + \
135            2*(totalenergyILA-totalenergyEQ)/\
136            (h[k]**2)*(1.0/(Mc)))
137        # LO mode
138        D[0,k,1] = np.sqrt(2*(totalenergyRLO-totalenergyEQ)/\
139            (h[k]**2)*(1.0/(Mc)) + \
140            2*(totalenergyILO-totalenergyEQ)/\
141            (h[k]**2)*(1.0/(Mc)))
142        # TA mode
143        D[0,k,2] = np.sqrt(2*(totalenergyRTA-totalenergyEQ)/\
144            (h[k]**2)*(1.0/(Mc)) + \
145            2*(totalenergyITA-totalenergyEQ)/\
146            (h[k]**2)*(1.0/(Mc)))
147        # TO mode
148        D[0,k,3] = np.sqrt(2*(totalenergyRTO-totalenergyEQ)/\
149            (h[k]**2)*(1.0/(Mc)) + \
150            2*(totalenergyITO-totalenergyEQ)/\
151            (h[k]**2)*(1.0/(Mc)))
152
153        k += 1
154
155    # -----
156    # We will now use the Richardson Extrapolation to speed up the convergence

```

```

157 # -----
158
159 l = 0
160 while l < len(h):
161     k = 0
162     while k < len(h):
163         if (l-1) >= 0 and (k-1) >= 0:
164             # LA mode
165             D[l,k,0] = (D[l-1,k,0]-(D[l-1,k-1,0]*(1.0/(2**(2*l)))))/\
166                     (1-(1.0/(2**(2*l))))
167             if D[l-1,k-1,0] == 0.0:
168                 D[l,k,0] = 0.0
169             # LO mode
170             D[l,k,1] = (D[l-1,k,1]-(D[l-1,k-1,1]*(1.0/(2**(2*l)))))/\
171                     (1-(1.0/(2**(2*l))))
172             if D[l-1,k-1,1] == 0.0:
173                 D[l,k,1] = 0.0
174             # TA mode
175             D[l,k,2] = (D[l-1,k,2]-(D[l-1,k-1,2]*(1.0/(2**(2*l)))))/\
176                     (1-(1.0/(2**(2*l))))
177             if D[l-1,k-1,2] == 0.0:
178                 D[l,k,2] = 0.0
179             # TO mode
180             D[l,k,3] = (D[l-1,k,3]-(D[l-1,k-1,3]*(1.0/(2**(2*l)))))/\
181                     (1-(1.0/(2**(2*l))))
182             if D[l-1,k-1,3] == 0.0:
183                 D[l,k,3] = 0.0
184             k += 1
185         l += 1
186
187 Dtotal = D
188 print Dtotal
189
190 # -----
191 # Computation of ZPM using finite difference on eigenenergies
192 # -----
193
194 # Import the total of the equilibrium state in Ha (same for imag)
195 eigenenergiesEQ = read_eigenenergies("supercellRo_DS1_EIG")
196
197 # Initialisation values
198 k = 0
199
200 # The second dimension is the number of h in geometric progression
201 # h,h/2,h/4,h/8 etc..
202 # The third index correspond to modes (here 4)
203 # The first index correspond to the Richardson extrapolation
204 D = np.zeros([len(h),len(h),4])
205
206 # Frequency (computed in DFPT using ABINIT)
207 omega = np.array([0.003915559,0.006088587,0.002163719,0.00589085])
208
209 # Level: Bandes
210 # HOMO
211 #lvl1 = 9 ; lvl2 = 10 ; lvl3 = 11
212 # LUMO
213 lvl1 = 12 ; lvl2 = 13 ; lvl3 = 14
214
215 # Carbon mass in a.u.
216 Mc = 21894.16693
217
218 while k < len(h):
219
220     # LA mode
221     eigenenergiesRLA = read_eigenenergies("supercellRo_DS"+str(4*k+2)+"_EIG")
222     # LO mode
223     eigenenergiesRLO = read_eigenenergies("supercellRo_DS"+str(4*k+3)+"_EIG")
224     # TA mode
225     eigenenergiesRTA = read_eigenenergies("supercellRo_DS"+str(4*k+4)+"_EIG")
226     # TO mode
227     eigenenergiesRTO = read_eigenenergies("supercellRo_DS"+str(4*k+5)+"_EIG")
228     # LA mode
229     eigenenergiesILA = read_eigenenergies("supercellIo_DS"+str(4*k+2)+"_EIG")
230     # LO mode
231     eigenenergiesILO = read_eigenenergies("supercellIo_DS"+str(4*k+3)+"_EIG")
232     # TA mode
233     eigenenergiesITA = read_eigenenergies("supercellIo_DS"+str(4*k+4)+"_EIG")
234     # TO mode
235     eigenenergiesITO = read_eigenenergies("supercellIo_DS"+str(4*k+5)+"_EIG")
236
237
238 # Second order derivative (meV)
239 # LA mode
240 D[0,k,0] = 2*(eigenenergiesRLA[lvl1]+eigenenergiesRLA[lvl2]+eigenenergiesRLA[lvl3])\

```

```

241         -eigenenergiesEQ[lvl1]-eigenenergiesEQ[lvl2]-eigenenergiesEQ[lvl3]))/\
242         (3*(h[k]**2))*(1.0/(4*Mc*omega[0]))*27.211383*1000+\
243         2*(eigenenergiesILA[lvl1]+eigenenergiesILA[lvl2]+eigenenergiesILA[lvl3]\
244         -eigenenergiesEQ[lvl1]-eigenenergiesEQ[lvl2]-eigenenergiesEQ[lvl3]))/\
245         (3*(h[k]**2))*(1.0/(4*Mc*omega[0]))*27.211383*1000
246
247     # L0 mode
248     D[0,k,1] = 2*(eigenenergiesRLO[lvl1]+eigenenergiesRLO[lvl2]+eigenenergiesRLO[lvl3]\
249     -eigenenergiesEQ[lvl1]-eigenenergiesEQ[lvl2]-eigenenergiesEQ[lvl3]))/\
250     (3*(h[k]**2))*(1.0/(4*Mc*omega[1]))*27.211383*1000+\
251     2*(eigenenergiesILO[lvl1]+eigenenergiesILO[lvl2]+eigenenergiesILO[lvl3]\
252     -eigenenergiesEQ[lvl1]-eigenenergiesEQ[lvl2]-eigenenergiesEQ[lvl3]))/\
253     (3*(h[k]**2))*(1.0/(4*Mc*omega[1]))*27.211383*1000
254
255     # TA mode
256     D[0,k,2] = 2*(eigenenergiesRTA[lvl1]+eigenenergiesRTA[lvl2]+eigenenergiesRTA[lvl3]\
257     -eigenenergiesEQ[lvl1]-eigenenergiesEQ[lvl2]-eigenenergiesEQ[lvl3]))/\
258     (3*(h[k]**2))*(1.0/(4*Mc*omega[2]))*27.211383*1000+\
259     2*(eigenenergiesITA[lvl1]+eigenenergiesITA[lvl2]+eigenenergiesITA[lvl3]\
260     -eigenenergiesEQ[lvl1]-eigenenergiesEQ[lvl2]-eigenenergiesEQ[lvl3]))/\
261     (3*(h[k]**2))*(1.0/(4*Mc*omega[2]))*27.211383*1000
262
263     # T0 mode
264     D[0,k,3] = 2*(eigenenergiesRTO[lvl1]+eigenenergiesRTO[lvl2]+eigenenergiesRTO[lvl3]\
265     -eigenenergiesEQ[lvl1]-eigenenergiesEQ[lvl2]-eigenenergiesEQ[lvl3]))/\
266     (3*(h[k]**2))*(1.0/(4*Mc*omega[3]))*27.211383*1000+\
267     2*(eigenenergiesITO[lvl1]+eigenenergiesITO[lvl2]+eigenenergiesITO[lvl3]\
268     -eigenenergiesEQ[lvl1]-eigenenergiesEQ[lvl2]-eigenenergiesEQ[lvl3]))/\
269     (3*(h[k]**2))*(1.0/(4*Mc*omega[3]))*27.211383*1000
270
271     k += 1
272
273     # -----
274     # We will now use the Richardson Extrapolation to speed up the convergence
275     # -----
276
277     l = 0
278     while l < len(h):
279         k = 0
280         while k < len(h):
281             if (l-1) >= 0 and (k-1) >= 0:
282                 # LA mode
283                 D[l,k,0] = (D[l-1,k,0]-(D[l-1,k-1,0]*(1.0/(2**(2*l)))))/\
284                 (1-(1.0/(2**(2*l))))
285                 if D[l-1,k-1,0] == 0.0:
286                     D[l,k,0] = 0.0
287                 # L0 mode
288                 D[l,k,1] = (D[l-1,k,1]-(D[l-1,k-1,1]*(1.0/(2**(2*l)))))/\
289                 (1-(1.0/(2**(2*l))))
290                 if D[l-1,k-1,1] == 0.0:
291                     D[l,k,1] = 0.0
292                 # TA mode
293                 D[l,k,2] = (D[l-1,k,2]-(D[l-1,k-1,2]*(1.0/(2**(2*l)))))/\
294                 (1-(1.0/(2**(2*l))))
295                 if D[l-1,k-1,2] == 0.0:
296                     D[l,k,2] = 0.0
297                 # T0 mode
298                 D[l,k,3] = (D[l-1,k,3]-(D[l-1,k-1,3]*(1.0/(2**(2*l)))))/\
299                 (1-(1.0/(2**(2*l))))
300                 if D[l-1,k-1,3] == 0.0:
301                     D[l,k,3] = 0.0
302                 k += 1
303             l += 1
304
305     Deigen = D
306
307     print "Richardson extrapolation of eigenenergies"
308     print Deigen
309
310     # -----
311     # Compute the first part the the NDDW term
312     # -----
313
314     # Second derivative using finite centred differences
315     # The second dimension is the number of h in geometric progression
316     # h,h/2,h/4,h/8 etc..
317     # The third index correspond to modes (here 4)
318     # The first index correspond to the Richardson extrapolation
319     D = np.zeros([len(h),len(h),4])
320
321     # Prepare equilibrium files
322     use_cut3d("supercellRo_DS1_VHXC","EQ",0,0)
323     use_cut3d("supercellRo_DS1_WFK","wfk",1,lv1+1)
324     use_cut3d("supercellRo_DS1_WFK","wfk",1,lv2+1)
325     use_cut3d("supercellRo_DS1_WFK","wfk",1,lv3+1)
326     averageEQ = (integration("EQ","wfk_k1_b"+str(lv1+1)+"_s1")+
327     integration("EQ","wfk_k1_b"+str(lv2+1)+"_s1")+
328     integration("EQ","wfk_k1_b"+str(lv3+1)+"_s1"))/3

```

```

325 k = 0
326 while k < len(h):
327
328
329 # Transform binary files into .txt files using cut3d
330 use_cut3d("supercellRo_DS"+str(4*k+2)+"_VHXC", "LAR", 0, 0)
331 use_cut3d("supercellRo_DS"+str(4*k+3)+"_VHXC", "LOR", 0, 0)
332 use_cut3d("supercellRo_DS"+str(4*k+4)+"_VHXC", "TAR", 0, 0)
333 use_cut3d("supercellRo_DS"+str(4*k+5)+"_VHXC", "TOR", 0, 0)
334 use_cut3d("supercellIo_DS"+str(4*k+2)+"_VHXC", "LAI", 0, 0)
335 use_cut3d("supercellIo_DS"+str(4*k+3)+"_VHXC", "LOI", 0, 0)
336 use_cut3d("supercellIo_DS"+str(4*k+4)+"_VHXC", "TAI", 0, 0)
337 use_cut3d("supercellIo_DS"+str(4*k+5)+"_VHXC", "TOI", 0, 0)
338
339 # Perform the integration int(psi_0*VHXC*psi_0)
340 averageLAR = (integration("LAR", "wfk_k1_b"+str(lvl1+1)+"_s1")+\
341 integration("LAR", "wfk_k1_b"+str(lvl2+1)+"_s1")+\
342 integration("LAR", "wfk_k1_b"+str(lvl3+1)+"_s1"))/3
343 averageLOR = (integration("LOR", "wfk_k1_b"+str(lvl1+1)+"_s1")+\
344 integration("LOR", "wfk_k1_b"+str(lvl2+1)+"_s1")+\
345 integration("LOR", "wfk_k1_b"+str(lvl3+1)+"_s1"))/3
346 averageTAR = (integration("TAR", "wfk_k1_b"+str(lvl1+1)+"_s1")+\
347 integration("TAR", "wfk_k1_b"+str(lvl2+1)+"_s1")+\
348 integration("TAR", "wfk_k1_b"+str(lvl3+1)+"_s1"))/3
349 averageTOR = (integration("TOR", "wfk_k1_b"+str(lvl1+1)+"_s1")+\
350 integration("TOR", "wfk_k1_b"+str(lvl2+1)+"_s1")+\
351 integration("TOR", "wfk_k1_b"+str(lvl3+1)+"_s1"))/3
352 averageLAI = (integration("LAI", "wfk_k1_b"+str(lvl1+1)+"_s1")+\
353 integration("LAI", "wfk_k1_b"+str(lvl2+1)+"_s1")+\
354 integration("LAI", "wfk_k1_b"+str(lvl3+1)+"_s1"))/3
355 averageLOI = (integration("LOI", "wfk_k1_b"+str(lvl1+1)+"_s1")+\
356 integration("LOI", "wfk_k1_b"+str(lvl2+1)+"_s1")+\
357 integration("LOI", "wfk_k1_b"+str(lvl3+1)+"_s1"))/3
358 averageTAI = (integration("TAI", "wfk_k1_b"+str(lvl1+1)+"_s1")+\
359 integration("TAI", "wfk_k1_b"+str(lvl2+1)+"_s1")+\
360 integration("TAI", "wfk_k1_b"+str(lvl3+1)+"_s1"))/3
361 averageTOI = (integration("TOI", "wfk_k1_b"+str(lvl1+1)+"_s1")+\
362 integration("TOI", "wfk_k1_b"+str(lvl2+1)+"_s1")+\
363 integration("TOI", "wfk_k1_b"+str(lvl3+1)+"_s1"))/3
364
365 # Second order derivative (meV)
366 # LA mode
367 D[0,k,0] = 2*(averageLAR-averageEQ)/\
368 ((h[k]**2)*(1.0/(4*Mc*omega[0]))*27.211383*1000+\
369 2*(averageLAI-averageEQ)/\
370 (h[k]**2)*(1.0/(4*Mc*omega[0]))*27.211383*1000
371 # LO mode
372 D[0,k,1] = 2*(averageLOR-averageEQ)/\
373 ((h[k]**2)*(1.0/(4*Mc*omega[1]))*27.211383*1000+\
374 2*(averageLOI-averageEQ)/\
375 (h[k]**2)*(1.0/(4*Mc*omega[1]))*27.211383*1000
376 # TA mode
377 D[0,k,2] = 2*(averageTAR-averageEQ)/\
378 ((h[k]**2)*(1.0/(4*Mc*omega[2]))*27.211383*1000+\
379 2*(averageTAI-averageEQ)/\
380 (h[k]**2)*(1.0/(4*Mc*omega[2]))*27.211383*1000
381 # TO mode
382 D[0,k,3] = 2*(averageTOR-averageEQ)/\
383 ((h[k]**2)*(1.0/(4*Mc*omega[3]))*27.211383*1000+\
384 2*(averageTOI-averageEQ)/\
385 (h[k]**2)*(1.0/(4*Mc*omega[3]))*27.211383*1000
386 k += 1
387
388 # -----
389 # We will now use the Richardson Extrapolation to speed up the convergence
390 # -----
391
392 l = 0
393 while l < len(h):
394     k = 0
395     while k < len(h):
396         if (l-1) >= 0 and (k-1) >= 0:
397             # LA mode
398             D[1,k,0] = (D[l-1,k,0]-(D[l-1,k-1,0]*(1.0/(2**(2*l)))))/\
399                 (1-(1.0/(2**(2*l))))
400             if D[l-1,k-1,0] == 0.0:
401                 D[1,k,0] = 0.0
402             # LO mode
403             D[1,k,1] = (D[l-1,k,1]-(D[l-1,k-1,1]*(1.0/(2**(2*l)))))/\
404                 (1-(1.0/(2**(2*l))))
405             if D[l-1,k-1,1] == 0.0:
406                 D[1,k,1] = 0.0
407             # TA mode
408             D[1,k,2] = (D[l-1,k,2]-(D[l-1,k-1,2]*(1.0/(2**(2*l)))))/\

```

```

409         (1-(1.0/(2**(2*1))))
410     if D[l-1,k-1,2] == 0.0:
411         D[l,k,2] = 0.0
412 #    T0 mode
413     D[l,k,3] = (D[l-1,k,3]-(D[l-1,k-1,3]*(1.0/(2**(2*1)))))/\
414         (1-(1.0/(2**(2*1))))
415     if D[l-1,k-1,3] == 0.0:
416         D[l,k,3] = 0.0
417     k += 1
418     l += 1
419
420 Dfirst = D
421 print "Richardson extrapolation of the first term of NDDW"
422 print Dfirst
423
424 # -----
425 # Compute the second part the the NDDW term on a primitive cell
426 # -----
427
428 D = np.zeros([len(h),len(h),4])
429
430 # Level: Bandes
431 # HOMO
432 #lvl1 = 2 ; lvl2 = 3 ; lvl3 = 4
433 # LUMO
434 lvl1 = 5 ; lvl2 = 6 ; lvl3 = 7
435
436
437 # Prepare equilibrium files
438 use_cut3d("primitiveRo_DS1_VHXC","EQ",0,0)
439 use_cut3d("primitiveRo_DS1_WFK","wfk",1,lvl1)
440 use_cut3d("primitiveRo_DS1_WFK","wfk",1,lvl2)
441 use_cut3d("primitiveRo_DS1_WFK","wfk",1,lvl3)
442 averageEQ = (integration("EQ","wfk_k1_b"+str(lvl1)+"_s1")+\\
443     integration("EQ","wfk_k1_b"+str(lvl2)+"_s1")+\\
444     integration("EQ","wfk_k1_b"+str(lvl3)+"_s1"))/3
445
446 k = 0
447 while k < len(h):
448
449 # Transform binary files into .txt files using cut3d
450 use_cut3d("primitiveRo_DS"+str(4*k+2)+"_VHXC","LAR",0,0)
451 use_cut3d("primitiveRo_DS"+str(4*k+3)+"_VHXC","LOR",0,0)
452 use_cut3d("primitiveRo_DS"+str(4*k+4)+"_VHXC","TAR",0,0)
453 use_cut3d("primitiveRo_DS"+str(4*k+5)+"_VHXC","TOR",0,0)
454 use_cut3d("primitiveIo_DS"+str(4*k+2)+"_VHXC","LAI",0,0)
455 use_cut3d("primitiveIo_DS"+str(4*k+3)+"_VHXC","LOI",0,0)
456 use_cut3d("primitiveIo_DS"+str(4*k+4)+"_VHXC","TAI",0,0)
457 use_cut3d("primitiveIo_DS"+str(4*k+5)+"_VHXC","TOI",0,0)
458
459 # Perform the integration int(psi_0*VHXC*psi_0)
460 averageLAR = (integration("LAR","wfk_k1_b"+str(lvl1)+"_s1")+\\
461     integration("LAR","wfk_k1_b"+str(lvl2)+"_s1")+\\
462     integration("LAR","wfk_k1_b"+str(lvl3)+"_s1"))/3
463 averageLOR = (integration("LOR","wfk_k1_b"+str(lvl1)+"_s1")+\\
464     integration("LOR","wfk_k1_b"+str(lvl2)+"_s1")+\\
465     integration("LOR","wfk_k1_b"+str(lvl3)+"_s1"))/3
466 averageTAR = (integration("TAR","wfk_k1_b"+str(lvl1)+"_s1")+\\
467     integration("TAR","wfk_k1_b"+str(lvl2)+"_s1")+\\
468     integration("TAR","wfk_k1_b"+str(lvl3)+"_s1"))/3
469 averageTOR = (integration("TOR","wfk_k1_b"+str(lvl1)+"_s1")+\\
470     integration("TOR","wfk_k1_b"+str(lvl2)+"_s1")+\\
471     integration("TOR","wfk_k1_b"+str(lvl3)+"_s1"))/3
472 averageLAI = (integration("LAI","wfk_k1_b"+str(lvl1)+"_s1")+\\
473     integration("LAI","wfk_k1_b"+str(lvl2)+"_s1")+\\
474     integration("LAI","wfk_k1_b"+str(lvl3)+"_s1"))/3
475 averageLOI = (integration("LOI","wfk_k1_b"+str(lvl1)+"_s1")+\\
476     integration("LOI","wfk_k1_b"+str(lvl2)+"_s1")+\\
477     integration("LOI","wfk_k1_b"+str(lvl3)+"_s1"))/3
478 averageTAI = (integration("TAI","wfk_k1_b"+str(lvl1)+"_s1")+\\
479     integration("TAI","wfk_k1_b"+str(lvl2)+"_s1")+\\
480     integration("TAI","wfk_k1_b"+str(lvl3)+"_s1"))/3
481 averageTOI = (integration("TOI","wfk_k1_b"+str(lvl1)+"_s1")+\\
482     integration("TOI","wfk_k1_b"+str(lvl2)+"_s1")+\\
483     integration("TOI","wfk_k1_b"+str(lvl3)+"_s1"))/3
484
485
486 # Second order derivative (meV)
487 # LA mode
488 D[0,k,0] = 2*(averageLAR-averageEQ)/\\
489     (h[k]**2)*(1.0/(4*Mc*omega[0]))*27.211383*1000+\\
490     2*(averageLAI-averageEQ)/\\
491     (h[k]**2)*(1.0/(4*Mc*omega[0]))*27.211383*1000
492 # L0 mode

```

```

493 D[0,k,1] = 2*(averageLOR-averageEQ)/\
494         (h[k]**2)*(1.0/(4*Mc*omega[1]))*27.211383*1000+\
495         2*(averageLOI-averageEQ)/\
496         (h[k]**2)*(1.0/(4*Mc*omega[0]))*27.211383*1000
497 # TA mode
498 D[0,k,2] = 2*(averageTAR-averageEQ)/\
499         (h[k]**2)*(1.0/(4*Mc*omega[2]))*27.211383*1000+\
500         2*(averageTAI-averageEQ)/\
501         (h[k]**2)*(1.0/(4*Mc*omega[0]))*27.211383*1000
502 # TO mode
503 D[0,k,3] = 2*(averageTOR-averageEQ)/\
504         (h[k]**2)*(1.0/(4*Mc*omega[3]))*27.211383*1000+\
505         2*(averageTOI-averageEQ)/\
506         (h[k]**2)*(1.0/(4*Mc*omega[0]))*27.211383*1000
507 k += 1
508
509 # -----
510 # We will now use the Richardson Extrapolation to speed up the convergence
511 # -----
512
513 l = 0
514 while l < len(h):
515     k = 0
516     while k < len(h):
517         if (l-1) >= 0 and (k-1) >= 0:
518             # LA mode
519             D[l,k,0] = (D[l-1,k,0]-(D[l-1,k-1,0]*(1.0/(2**(2*l)))))/\
520                     (1-(1.0/(2**(2*l))))
521             if D[l-1,k-1,0] == 0.0:
522                 D[l,k,0] = 0.0
523             # LO mode
524             D[l,k,1] = (D[l-1,k,1]-(D[l-1,k-1,1]*(1.0/(2**(2*l)))))/\
525                     (1-(1.0/(2**(2*l))))
526             if D[l-1,k-1,1] == 0.0:
527                 D[l,k,1] = 0.0
528             # TA mode
529             D[l,k,2] = (D[l-1,k,2]-(D[l-1,k-1,2]*(1.0/(2**(2*l)))))/\
530                     (1-(1.0/(2**(2*l))))
531             if D[l-1,k-1,2] == 0.0:
532                 D[l,k,2] = 0.0
533             # TO mode
534             D[l,k,3] = (D[l-1,k,3]-(D[l-1,k-1,3]*(1.0/(2**(2*l)))))/\
535                     (1-(1.0/(2**(2*l))))
536             if D[l-1,k-1,3] == 0.0:
537                 D[l,k,3] = 0.0
538             k += 1
539         l += 1
540
541 print "Richardson extrapolation of eigenenergies"
542 print Deigen
543
544 print "Richardson extrapolation of the first term of NDDW"
545 print Dfirst
546
547 print "Richardson extrapolation of the second term of NDDW"
548 print D
549

```

Code 15: Python scripts use to compute finite difference using the Richardson extrapolation

5.13 NDDW by Finite diff. at the $q=2/3L$

```

1 # -----
2 # This script only create the input file for the second
3 # NDDW term using another approach based on Tb and Sb
4 # -----
5
6 # Find the eigenvector from the dynamical matrixe
7
8 import numpy as np
9
10 # Location of the Dynamical matrix file
11 L_file = '/home/Samuel/Dropbox/WorkDiam/2L3_Global/Dyanmical_mat.dat'
12 # Carbon mass in a.u.
13 Mc=21894.16693
14
15 def read_dynamical_matrix(datafile):
16     with open(datafile,'r') as f:

```

```

17     raw = list()
18     for line in f:
19         if line[0]!='#': continue
20         parts=line.split()
21         if not parts: continue
22         valR=float(parts[4]) #real
23         valI=float(parts[5]) #imaginary
24         if np.abs(valR) < 0.000000001:
25             valR=0.0
26         if np.abs(valI) < 0.000000001:
27             valI=0.0
28         raw.append(complex(valR,valI))
29     dynmat = np.array(raw)
30     dynmat = dynmat.reshape(6,6)
31     return dynmat
32
33 dynmat=read_dynamical_matrix(L_file)
34 [eigval,eigvect]=np.linalg.eig(dynmat)
35 eigvect=np.transpose(eigvect)
36
37 print "eigenvalues at 2/3L : "
38 for n in eigval:
39     a = np.sqrt(n/Mc)
40     print a
41
42 print "eigenvectors at 2/3L : "
43 for n in eigvect: print n
44
45
46 # Create the xcart needed for abinit input file
47
48 h = np.array([0.005])
49
50 rprim = np.array([6.70346805,6.70346805,6.70346805])
51
52
53 print "# Equilibrium position"
54 print " xcart1 0.0 0.0 0.0 # 0.0 0.0 0.0 "
55 print "      1.67586701 1.67586701 1.67586701 # 0.25 0.25 0.25"
56 print " "
57 print "# LA mode (+h)"
58 j = 2
59
60 for disp in h:
61     # -----
62     # LA mode
63     # -----
64     i = 1
65     # Atome K
66     SBX1 = np.array([eigvect.real[i][0],eigvect.real[i][1],\
67                     eigvect.real[i][2]])*eigvect.real[i][0]*disp
68     SBY1 = np.array([eigvect.real[i][0],eigvect.real[i][1],\
69                     eigvect.real[i][2]])*eigvect.real[i][1]*disp
70     SBZ1 = np.array([eigvect.real[i][0],eigvect.real[i][1],\
71                     eigvect.real[i][2]])*eigvect.real[i][2]*disp
72     SBX2 = np.array([eigvect.real[i][3],eigvect.real[i][4],\
73                     eigvect.real[i][5]])*eigvect.real[i][3]*disp
74     SBY2 = np.array([eigvect.real[i][3],eigvect.real[i][4],\
75                     eigvect.real[i][5]])*eigvect.real[i][4]*disp
76     SBZ2 = np.array([eigvect.real[i][3],eigvect.real[i][4],\
77                     eigvect.real[i][5]])*eigvect.real[i][5]*disp
78
79 # Calcule translation
80 TrX = np.array([1,0,0])*disp
81 TrY = np.array([0,1,0])*disp
82 TrZ = np.array([0,0,1])*disp
83
84 # -----
85 # Calcul du terme d2Vhxc/dSBdTB
86 # Atomes K
87 # -----
88
89 # VHXC(x,y)
90 print " xcart%.0f %.10f %.10f %.10f"%(j,rprim[0]*0.0+(SBX1[0]+TrX[0]),\
91     rprim[1]*0.0+(SBX1[1]+TrX[1]),rprim[2]*0.0+(SBX1[2]+TrX[2]))
92 print "      %.10f %.10f %.10f"%(rprim[0]*0.25+(SBX2[0]+TrX[0]),\
93     rprim[1]*0.25+(SBX2[1]+TrX[1]),rprim[2]*0.25+(SBX2[2]+TrX[2]))
94 print ""
95 j += 1
96
97 # VHXC(-x,y)
98 print " xcart%.0f %.10f %.10f %.10f"%(j,rprim[0]*0.0+(-SBX1[0]+TrX[0]),\
99     rprim[1]*0.0+(-SBX1[1]+TrX[1]),rprim[2]*0.0+(-SBX1[2]+TrX[2]))
100 print "      %.10f %.10f %.10f"%(rprim[0]*0.25+(-SBX2[0]+TrX[0]),\

```



```

101         rprim[1]*0.25+(-SBX2[1]+TrX[1]),rprim[2]*0.25+(-SBX2[2]+TrX[2]))
102     print " "
103     j += 1
104
105 # VHXC(x,-y)
106     print " xcart%.0f %.10f %.10f %.10f"%(j,rprim[0]*0.0+(SBX1[0]-TrX[0]),\
107         rprim[1]*0.0+(SBX1[1]-TrX[1]),rprim[2]*0.0+(SBX1[2]-TrX[2]))
108     print "          %.10f %.10f %.10f"%(rprim[0]*0.25+(SBX2[0]-TrX[0]),\
109         rprim[1]*0.25+(SBX2[1]-TrX[1]),rprim[2]*0.25+(SBX2[2]-TrX[2]))
110     print ""
111     j += 1
112
113 # VHXC(-x,-y)
114     print " xcart%.0f %.10f %.10f %.10f"%(j,rprim[0]*0.0+(-SBX1[0]-TrX[0]),\
115         rprim[1]*0.0+(-SBX1[1]-TrX[1]),rprim[2]*0.0+(-SBX1[2]-TrX[2]))
116     print "          %.10f %.10f %.10f"%(rprim[0]*0.25+(-SBX2[0]-TrX[0]),\
117         rprim[1]*0.25+(-SBX2[1]-TrX[1]),rprim[2]*0.25+(-SBX2[2]-TrX[2]))
118     print ""
119     j += 1
120
121 # VHXC(x,y)
122     print " xcart%.0f %.10f %.10f %.10f"%(j,rprim[0]*0.0+(SBY1[0]+TrY[0]),\
123         rprim[1]*0.0+(SBY1[1]+TrY[1]),rprim[2]*0.0+(SBY1[2]+TrY[2]))
124     print "          %.10f %.10f %.10f"%(rprim[0]*0.25+(SBY2[0]+TrY[0]),\
125         rprim[1]*0.25+(SBY2[1]+TrY[1]),rprim[2]*0.25+(SBY2[2]+TrY[2]))
126     print ""
127     j += 1
128
129 # VHXC(-x,y)
130     print " xcart%.0f %.10f %.10f %.10f"%(j,rprim[0]*0.0+(-SBY1[0]+TrY[0]),\
131         rprim[1]*0.0+(-SBY1[1]+TrY[1]),rprim[2]*0.0+(-SBY1[2]+TrY[2]))
132     print "          %.10f %.10f %.10f"%(rprim[0]*0.25+(-SBY2[0]+TrY[0]),\
133         rprim[1]*0.25+(-SBY2[1]+TrY[1]),rprim[2]*0.25+(-SBY2[2]+TrY[2]))
134     print ""
135     j += 1
136
137 # VHXC(x,-y)
138     print " xcart%.0f %.10f %.10f %.10f"%(j,rprim[0]*0.0+(SBY1[0]-TrY[0]),\
139         rprim[1]*0.0+(SBY1[1]-TrY[1]),rprim[2]*0.0+(SBY1[2]-TrY[2]))
140     print "          %.10f %.10f %.10f"%(rprim[0]*0.25+(SBY2[0]-TrY[0]),\
141         rprim[1]*0.25+(SBY2[1]-TrY[1]),rprim[2]*0.25+(SBY2[2]-TrY[2]))
142     print ""
143     j += 1
144
145 # VHXC(-x,-y)
146     print " xcart%.0f %.10f %.10f %.10f"%(j,rprim[0]*0.0+(-SBY1[0]-TrY[0]),\
147         rprim[1]*0.0+(-SBY1[1]-TrY[1]),rprim[2]*0.0+(-SBY1[2]-TrY[2]))
148     print "          %.10f %.10f %.10f"%(rprim[0]*0.25+(-SBY2[0]-TrY[0]),\
149         rprim[1]*0.25+(-SBY2[1]-TrY[1]),rprim[2]*0.25+(-SBY2[2]-TrY[2]))
150     print ""
151     j += 1
152
153 # VHXC(x,y)
154     print " xcart%.0f %.10f %.10f %.10f"%(j,rprim[0]*0.0+(SBZ1[0]+TrZ[0]),\
155         rprim[1]*0.0+(SBZ1[1]+TrZ[1]),rprim[2]*0.0+(SBZ1[2]+TrZ[2]))
156     print "          %.10f %.10f %.10f"%(rprim[0]*0.25+(SBZ2[0]+TrZ[0]),\
157         rprim[1]*0.25+(SBZ2[1]+TrZ[1]),rprim[2]*0.25+(SBZ2[2]+TrZ[2]))
158     print ""
159     j += 1
160
161 # VHXC(-x,y)
162     print " xcart%.0f %.10f %.10f %.10f"%(j,rprim[0]*0.0+(-SBZ1[0]+TrZ[0]),\
163         rprim[1]*0.0+(-SBZ1[1]+TrZ[1]),rprim[2]*0.0+(-SBZ1[2]+TrZ[2]))
164     print "          %.10f %.10f %.10f"%(rprim[0]*0.25+(-SBZ2[0]+TrZ[0]),\
165         rprim[1]*0.25+(-SBZ2[1]+TrZ[1]),rprim[2]*0.25+(-SBZ2[2]+TrZ[2]))
166     print ""
167     j += 1
168
169 # VHXC(x,-y)
170     print " xcart%.0f %.10f %.10f %.10f"%(j,rprim[0]*0.0+(SBZ1[0]-TrZ[0]),\
171         rprim[1]*0.0+(SBZ1[1]-TrZ[1]),rprim[2]*0.0+(SBZ1[2]-TrZ[2]))
172     print "          %.10f %.10f %.10f"%(rprim[0]*0.25+(SBZ2[0]-TrZ[0]),\
173         rprim[1]*0.25+(SBZ2[1]-TrZ[1]),rprim[2]*0.25+(SBZ2[2]-TrZ[2]))
174     print ""
175     j += 1
176
177 # VHXC(-x,-y)
178     print " xcart%.0f %.10f %.10f %.10f"%(j,rprim[0]*0.0+(-SBZ1[0]-TrZ[0]),\
179         rprim[1]*0.0+(-SBZ1[1]-TrZ[1]),rprim[2]*0.0+(-SBZ1[2]-TrZ[2]))
180     print "          %.10f %.10f %.10f"%(rprim[0]*0.25+(-SBZ2[0]-TrZ[0]),\
181         rprim[1]*0.25+(-SBZ2[1]-TrZ[1]),rprim[2]*0.25+(-SBZ2[2]-TrZ[2]))
182     print ""
183     j += 1
184
185 # -----
186 # LA mode

```

```

185 # -----
186 # Atome K
187 SBX1 = np.array([eigvect.imag[i][0], eigvect.imag[i][1], \
188 eigvect.imag[i][2]])*eigvect.imag[i][0]*disp
189 SBY1 = np.array([eigvect.imag[i][0], eigvect.imag[i][1], \
190 eigvect.imag[i][2]])*eigvect.imag[i][1]*disp
191 SBZ1 = np.array([eigvect.imag[i][0], eigvect.imag[i][1], \
192 eigvect.imag[i][2]])*eigvect.imag[i][2]*disp
193 SBX2 = np.array([eigvect.imag[i][3], eigvect.imag[i][4], \
194 eigvect.imag[i][5]])*eigvect.imag[i][3]*disp
195 SBY2 = np.array([eigvect.imag[i][3], eigvect.imag[i][4], \
196 eigvect.imag[i][5]])*eigvect.imag[i][4]*disp
197 SBZ2 = np.array([eigvect.imag[i][3], eigvect.imag[i][4], \
198 eigvect.imag[i][5]])*eigvect.imag[i][5]*disp
199
200 # Calculs translation
201 TrX = np.array([1,0,0])*disp
202 TrY = np.array([0,1,0])*disp
203 TrZ = np.array([0,0,1])*disp
204
205 # -----
206 # Calcul du terme d2Vhxc/dSBdTB
207 # Atomes K
208 # -----
209
210 # VHXC(x,y)
211 print " xcart%.0f %.10f %.10f %.10f"%(j,rprim[0]*0.0+(SBX1[0]+TrX[0]), \
212 rprim[1]*0.0+(SBX1[1]+TrX[1]), rprim[2]*0.0+(SBX1[2]+TrX[2]))
213 print " %.10f %.10f %.10f"%(rprim[0]*0.25+(SBX2[0]+TrX[0]), \
214 rprim[1]*0.25+(SBX2[1]+TrX[1]), rprim[2]*0.25+(SBX2[2]+TrX[2]))
215 print ""
216 j += 1
217
218 # VHXC(-x,y)
219 print " xcart%.0f %.10f %.10f %.10f"%(j,rprim[0]*0.0+(-SBX1[0]+TrX[0]), \
220 rprim[1]*0.0+(-SBX1[1]+TrX[1]), rprim[2]*0.0+(-SBX1[2]+TrX[2]))
221 print " %.10f %.10f %.10f"%(rprim[0]*0.25+(-SBX2[0]+TrX[0]), \
222 rprim[1]*0.25+(-SBX2[1]+TrX[1]), rprim[2]*0.25+(-SBX2[2]+TrX[2]))
223 print ""
224 j += 1
225
226 # VHXC(x,-y)
227 print " xcart%.0f %.10f %.10f %.10f"%(j,rprim[0]*0.0+(SBX1[0]-TrX[0]), \
228 rprim[1]*0.0+(SBX1[1]-TrX[1]), rprim[2]*0.0+(SBX1[2]-TrX[2]))
229 print " %.10f %.10f %.10f"%(rprim[0]*0.25+(SBX2[0]-TrX[0]), \
230 rprim[1]*0.25+(SBX2[1]-TrX[1]), rprim[2]*0.25+(SBX2[2]-TrX[2]))
231 print ""
232 j += 1
233
234 # VHXC(-x,-y)
235 print " xcart%.0f %.10f %.10f %.10f"%(j,rprim[0]*0.0+(-SBX1[0]-TrX[0]), \
236 rprim[1]*0.0+(-SBX1[1]-TrX[1]), rprim[2]*0.0+(-SBX1[2]-TrX[2]))
237 print " %.10f %.10f %.10f"%(rprim[0]*0.25+(-SBX2[0]-TrX[0]), \
238 rprim[1]*0.25+(-SBX2[1]-TrX[1]), rprim[2]*0.25+(-SBX2[2]-TrX[2]))
239 print ""
240 j += 1
241
242 # VHXC(x,y)
243 print " xcart%.0f %.10f %.10f %.10f"%(j,rprim[0]*0.0+(SBY1[0]+TrY[0]), \
244 rprim[1]*0.0+(SBY1[1]+TrY[1]), rprim[2]*0.0+(SBY1[2]+TrY[2]))
245 print " %.10f %.10f %.10f"%(rprim[0]*0.25+(SBY2[0]+TrY[0]), \
246 rprim[1]*0.25+(SBY2[1]+TrY[1]), rprim[2]*0.25+(SBY2[2]+TrY[2]))
247 print ""
248 j += 1
249
250 # VHXC(-x,y)
251 print " xcart%.0f %.10f %.10f %.10f"%(j,rprim[0]*0.0+(-SBY1[0]+TrY[0]), \
252 rprim[1]*0.0+(-SBY1[1]+TrY[1]), rprim[2]*0.0+(-SBY1[2]+TrY[2]))
253 print " %.10f %.10f %.10f"%(rprim[0]*0.25+(-SBY2[0]+TrY[0]), \
254 rprim[1]*0.25+(-SBY2[1]+TrY[1]), rprim[2]*0.25+(-SBY2[2]+TrY[2]))
255 print ""
256 j += 1
257
258 # VHXC(x,-y)
259 print " xcart%.0f %.10f %.10f %.10f"%(j,rprim[0]*0.0+(SBY1[0]-TrY[0]), \
260 rprim[1]*0.0+(SBY1[1]-TrY[1]), rprim[2]*0.0+(SBY1[2]-TrY[2]))
261 print " %.10f %.10f %.10f"%(rprim[0]*0.25+(SBY2[0]-TrY[0]), \
262 rprim[1]*0.25+(SBY2[1]-TrY[1]), rprim[2]*0.25+(SBY2[2]-TrY[2]))
263 print ""
264 j += 1
265
266 # VHXC(-x,-y)
267 print " xcart%.0f %.10f %.10f %.10f"%(j,rprim[0]*0.0+(-SBY1[0]-TrY[0]), \
268 rprim[1]*0.0+(-SBY1[1]-TrY[1]), rprim[2]*0.0+(-SBY1[2]-TrY[2]))

```

```

269 print "          %.10f %.10f %.10f" %(rprim[0]*0.25+(-SBY2[0]-TrY[0]),\
270      rprim[1]*0.25+(-SBY2[1]-TrY[1]),rprim[2]*0.25+(-SBY2[2]-TrY[2]))
271 print ""
272 j += 1
273 # VHXC(x,y)
274 print " xcart%.0f %.10f %.10f %.10f" %(j,rprim[0]*0.0+(SBZ1[0]+TrZ[0]),\
275      rprim[1]*0.0+(SBZ1[1]+TrZ[1]),rprim[2]*0.0+(SBZ1[2]+TrZ[2]))
276 print "          %.10f %.10f %.10f" %(rprim[0]*0.25+(SBZ2[0]+TrZ[0]),\
277      rprim[1]*0.25+(SBZ2[1]+TrZ[1]),rprim[2]*0.25+(SBZ2[2]+TrZ[2]))
278 print ""
279 j += 1
280
281 # VHXC(-x,y)
282 print " xcart%.0f %.10f %.10f %.10f" %(j,rprim[0]*0.0+(-SBZ1[0]+TrZ[0]),\
283      rprim[1]*0.0+(-SBZ1[1]+TrZ[1]),rprim[2]*0.0+(-SBZ1[2]+TrZ[2]))
284 print "          %.10f %.10f %.10f" %(rprim[0]*0.25+(-SBZ2[0]+TrZ[0]),\
285      rprim[1]*0.25+(-SBZ2[1]+TrZ[1]),rprim[2]*0.25+(-SBZ2[2]+TrZ[2]))
286 print ""
287 j += 1
288 # VHXC(x,-y)
289 print " xcart%.0f %.10f %.10f %.10f" %(j,rprim[0]*0.0+(SBZ1[0]-TrZ[0]),\
290      rprim[1]*0.0+(SBZ1[1]-TrZ[1]),rprim[2]*0.0+(SBZ1[2]-TrZ[2]))
291 print "          %.10f %.10f %.10f" %(rprim[0]*0.25+(SBZ2[0]-TrZ[0]),\
292      rprim[1]*0.25+(SBZ2[1]-TrZ[1]),rprim[2]*0.25+(SBZ2[2]-TrZ[2]))
293 print ""
294 j += 1
295
296 # VHXC(-x,-y)
297 print " xcart%.0f %.10f %.10f %.10f" %(j,rprim[0]*0.0+(-SBZ1[0]-TrZ[0]),\
298      rprim[1]*0.0+(-SBZ1[1]-TrZ[1]),rprim[2]*0.0+(-SBZ1[2]-TrZ[2]))
299 print "          %.10f %.10f %.10f" %(rprim[0]*0.25+(-SBZ2[0]-TrZ[0]),\
300      rprim[1]*0.25+(-SBZ2[1]-TrZ[1]),rprim[2]*0.25+(-SBZ2[2]-TrZ[2]))
301 print ""
302 j += 1

```

Code 16: Python scripts use to create ABINIT input files

5.14 NDDW by Finite diff. at the $q=2/3L$

```

1 # -----
2 # This python script is used to compute the contribution of the
3 # L point to the Gamma eigenenergies using finite difference.
4 # -----
5 import os
6 import numpy as np
7
8 def read_eigenenergies(datafile):
9     flag1 = False
10    flag2 = False
11    with open(datafile,'r') as f:
12        for line in f:
13            if flag2:
14                eigenenergies = np.append([eigenenergies],[map(float,line.split())])
15                break
16            if flag1:
17                eigenenergies = map(float,line.split())
18                flag2 = True
19            if "kpt= 0.0000 0.0000 0.0000" in line:
20                flag1 = True
21    return eigenenergies
22
23 def use_cut3d(datafile,out,gamma,band):
24     if datafile[len(datafile)-4:len(datafile)] == "VHXC":
25         os.system("rm cut3d.files")
26         with open("cut3d.files","a") as files:
27             files.write(str(datafile)+"\n")
28             files.write("1 \n")
29             files.write("5 \n") #3D formatted data
30             # (output the bare 3D data - one column)
31             files.write(str(out)+"\n")
32             files.write("0 \n")
33         os.system("cut3d < cut3d.files ")
34     if datafile[len(datafile)-3:len(datafile)] == "WFK":
35         os.system("rm wfk.files")
36         with open("wfk.files","a") as files:
37             files.write(str(datafile)+"\n")
38             files.write("1 \n")
39             files.write("0 \n")

```

```

40     files.write(str(gamma)+" \n") # Gamma point
41     files.write(str(band)+" \n") # Band nb 6
42     files.write("0 \n")
43     files.write("0 \n")
44     files.write("2 \n") # real 3D data one column
45     files.write(str(out)+" \n")
46     files.write("0 \n")
47     os.system("cut3d < wfk.files ")
48
49 def integration(VHXC,wfk):
50     psi_file=open(wfk,'r')
51     v_file=open(VHXC,'r')
52     S=0.0; N=0.0
53     while True:
54         psi_l=psi_file.readline()
55         v_l=v_file.readline()
56         if not psi_l or not v_l: break
57         psi=float(psi_l)
58         v=float(v_l)
59         S+=psi*v*psi
60         N+=psi*psi
61     psi_file.close()
62     v_file.close()
63     S=S/N
64     return S
65
66 # The base name for the supercell files are assumed to be named "supercell"
67 # and the primitive as assumed to be named "primitive"
68
69 # Value and number of the displacement
70 h = np.array([0.005])
71
72 # Initialisation values
73 k = 0
74
75 # Frequency (computed in DFPT using ABINIT)
76 omega = np.array([0.003915559,0.006088587,0.002163719,0.00589085])
77
78 # Carbon mass in a.u.
79 Mc = 21894.16693
80
81 # -----
82 # Compute the second part the the NDDW term on a primitive cell
83 # -----
84
85 # Level: Bandes
86 # HOMO
87 #lvl1 = 2 ; lvl2 = 3 ; lvl3 = 4
88 # LUMO
89 #lvl1 = 5 ; lvl2 = 6 ; lvl3 = 7
90
91
92 # Prepare equilibrium files
93 use_cut3d("primitiveo_DS1_VHXC","EQ",0,0)
94 use_cut3d("primitiveo_DS1_WFK","wfk",1,lvl1)
95 use_cut3d("primitiveo_DS1_WFK","wfk",1,lvl2)
96 use_cut3d("primitiveo_DS1_WFK","wfk",1,lvl3)
97 averageEQ = (integration("EQ","wfk_k1_b"+str(lvl1)+"_s1")+
98             integration("EQ","wfk_k1_b"+str(lvl2)+"_s1")+
99             integration("EQ","wfk_k1_b"+str(lvl3)+"_s1"))/3
100
101 k = 0
102 eigen = [0.0,0.0]
103 while k < 2:
104
105 # Transform binary files into .txt files using cut3d
106 use_cut3d("primitiveo_DS"+str(12*k+2)+"_VHXC","VX++",0,0)
107 use_cut3d("primitiveo_DS"+str(12*k+3)+"_VHXC","VX+-",0,0)
108 use_cut3d("primitiveo_DS"+str(12*k+4)+"_VHXC","VX-+",0,0)
109 use_cut3d("primitiveo_DS"+str(12*k+5)+"_VHXC","VX--",0,0)
110 use_cut3d("primitiveo_DS"+str(12*k+6)+"_VHXC","VY++",0,0)
111 use_cut3d("primitiveo_DS"+str(12*k+7)+"_VHXC","VY+-",0,0)
112 use_cut3d("primitiveo_DS"+str(12*k+8)+"_VHXC","VY-+",0,0)
113 use_cut3d("primitiveo_DS"+str(12*k+9)+"_VHXC","VY--",0,0)
114 use_cut3d("primitiveo_DS"+str(12*k+10)+"_VHXC","VZ++",0,0)
115 use_cut3d("primitiveo_DS"+str(12*k+11)+"_VHXC","VZ+-",0,0)
116 use_cut3d("primitiveo_DS"+str(12*k+12)+"_VHXC","VZ-+",0,0)
117 use_cut3d("primitiveo_DS"+str(12*k+13)+"_VHXC","VZ--",0,0)
118
119 # Perform the integration int(psi_0*VHXC*psi_0)
120 averageVX1 = (integration("VX++","wfk_k1_b"+str(lvl1)+"_s1")+
121             integration("VX++","wfk_k1_b"+str(lvl2)+"_s1")+
122             integration("VX++","wfk_k1_b"+str(lvl3)+"_s1"))/3

```

```

124 averageVX2 = (integration("VX+", "wfk_k1_b"+str(lvl1)+"_s1")+\\
125     integration("VX+", "wfk_k1_b"+str(lvl2)+"_s1")+\\
126     integration("VX+", "wfk_k1_b"+str(lvl3)+"_s1"))/3
127 averageVX3 = (integration("VX+", "wfk_k1_b"+str(lvl1)+"_s1")+\\
128     integration("VX+", "wfk_k1_b"+str(lvl2)+"_s1")+\\
129     integration("VX+", "wfk_k1_b"+str(lvl3)+"_s1"))/3
130 averageVX4 = (integration("VX+", "wfk_k1_b"+str(lvl1)+"_s1")+\\
131     integration("VX+", "wfk_k1_b"+str(lvl2)+"_s1")+\\
132     integration("VX+", "wfk_k1_b"+str(lvl3)+"_s1"))/3
133 averageVY1 = (integration("VY+", "wfk_k1_b"+str(lvl1)+"_s1")+\\
134     integration("VY+", "wfk_k1_b"+str(lvl2)+"_s1")+\\
135     integration("VY+", "wfk_k1_b"+str(lvl3)+"_s1"))/3
136 averageVY2 = (integration("VY+", "wfk_k1_b"+str(lvl1)+"_s1")+\\
137     integration("VY+", "wfk_k1_b"+str(lvl2)+"_s1")+\\
138     integration("VY+", "wfk_k1_b"+str(lvl3)+"_s1"))/3
139 averageVY3 = (integration("VY+", "wfk_k1_b"+str(lvl1)+"_s1")+\\
140     integration("VY+", "wfk_k1_b"+str(lvl2)+"_s1")+\\
141     integration("VY+", "wfk_k1_b"+str(lvl3)+"_s1"))/3
142 averageVY4 = (integration("VY+", "wfk_k1_b"+str(lvl1)+"_s1")+\\
143     integration("VY+", "wfk_k1_b"+str(lvl2)+"_s1")+\\
144     integration("VY+", "wfk_k1_b"+str(lvl3)+"_s1"))/3
145 averageVZ1 = (integration("VZ+", "wfk_k1_b"+str(lvl1)+"_s1")+\\
146     integration("VZ+", "wfk_k1_b"+str(lvl2)+"_s1")+\\
147     integration("VZ+", "wfk_k1_b"+str(lvl3)+"_s1"))/3
148 averageVZ2 = (integration("VZ+", "wfk_k1_b"+str(lvl1)+"_s1")+\\
149     integration("VZ+", "wfk_k1_b"+str(lvl2)+"_s1")+\\
150     integration("VZ+", "wfk_k1_b"+str(lvl3)+"_s1"))/3
151 averageVZ3 = (integration("VZ+", "wfk_k1_b"+str(lvl1)+"_s1")+\\
152     integration("VZ+", "wfk_k1_b"+str(lvl2)+"_s1")+\\
153     integration("VZ+", "wfk_k1_b"+str(lvl3)+"_s1"))/3
154 averageVZ4 = (integration("VZ+", "wfk_k1_b"+str(lvl1)+"_s1")+\\
155     integration("VZ+", "wfk_k1_b"+str(lvl2)+"_s1")+\\
156     integration("VZ+", "wfk_k1_b"+str(lvl3)+"_s1"))/3
157
158 # Second order derivative (meV)
159 # LA mode
160 eigen[k] = (averageVX1-averageVX2-averageVX3+averageVX4+\\
161     averageVY1-averageVY2-averageVY3+averageVY4+\\
162     averageVZ1-averageVZ2-averageVZ3+averageVZ4)/\\
163     (4*(h[0]**2))*(1.0/(4*Mc*omega[1]))*27.211383*1000
164
165 k += 1
166
167 print eigen

```

Code 17: Python script used to compute finite difference using the Richardson extrapolation

6 Side notes [Just for me...]

- Koopmans' theorem: states that in closed-shell Hartree-Fock theory, the first ionization energy of a molecular system is equal to the negative of the orbital energy of the highest occupied molecular orbital (HOMO).
- Brook's theorem: if we add an electron to a system the total energy will be modified by the affinity energy of the system.
- The idee for the NDDW is to add a frozen density and compute the change of phonon spectrum. We then not change the occupation number (so that we don't have problem with the wave vector) but the electronic density. The fact of adding an electron break the sym. Le changement de densité fig. 5.56 un système qui est isolant dans un calcul de phonon entraîne qu'on observe un changement de valeur propre des phonons qui donne le NDDW.

BUG remarqué par Paul sur le TR sym.

$$\sum_{\alpha,\beta} \sum_{\kappa,\kappa'} \sum_j w_Q \frac{e^{iQ \cdot (r_\kappa - r_{\kappa'})} \xi_\alpha(-Qj, \kappa) \xi_\beta(Qj, \kappa')}{\sqrt{M_\kappa M_{\kappa'}} \omega_{Qj}} \quad (127)$$

Dans cette equation, α et β sont des coordonnes cartesiennes, κ et κ' sont les indexs des atomes, j identifie les branches phononiques, w_Q est le poids du point Q , les ξ sont les vecteurs de polarisation phononique, les M_κ sont les masse atomiques et enfin ω_{Qj} les frequences phononiques. Bien sur, la presence des masses et de la frenquence signifie que ces "poids" ne sont pas reellement normalises donc ils ne se somment pas a 1. Par contre, si j'imprime ces valeurs, j'obtiens pour la grille complete sans-symetrie (64 points Q , grille de 4x4x4 pour les tests):

BUG remarqué par Xavier dans la routine thmeig.F90. Le facteur de phase a été calculé deux fois.

Il y a en effet deux façons de définir la matrice dynamique (avec la cellule unitaire considérée) (voir eq 5.56 these de Paul):

$$D_{\alpha\kappa,\beta\kappa'}^I(Q) = \frac{\partial^2 E}{\partial R_{\alpha\kappa l} \partial R_{\beta\kappa' l'}} e^{iq(R_{\kappa l} - R_{\kappa' l'})} \quad (128)$$

$$D_{\alpha\kappa,\beta\kappa'}^{II}(Q) = \frac{\partial^2 E}{\partial R_{\alpha\kappa l} \partial R_{\beta\kappa' l'}} e^{iq(R_l - R_{l'})} \quad (129)$$

$$D_{\alpha\kappa,\beta\kappa'}^I(Q) = e^{iq(R_\kappa - R_{\kappa'})} D_{\alpha\kappa,\beta\kappa'}^{II}(Q) \quad (130)$$

Si on a la relation suivante:

$$\sum_{\beta\kappa'} D_{\alpha\kappa,\beta\kappa'}^I(Q) \xi_{\beta\kappa'}(q) = M_\kappa \omega^2 \xi_{\alpha\kappa}(q) \quad (131)$$

$$\sum_{\beta\kappa'} e^{iq(R_\kappa)} D_{\alpha\kappa,\beta\kappa'}^{II}(Q) e^{-iq(R_{\kappa'})} \xi_{\beta\kappa'}(q) = M_\kappa \omega^2 \xi_{\alpha\kappa}(q) \quad (132)$$

$$\sum_{\beta\kappa'} D_{\alpha\kappa,\beta\kappa'}^{II}(Q) \left[e^{-iq(R_{\kappa'})} \xi_{\beta\kappa'}(q) \right] = M_\kappa \omega^2 \left[e^{-iq(R_\kappa)} \xi_{\alpha\kappa}(q) \right] \quad (133)$$

$$\sum_{\beta\kappa'} D_{\alpha\kappa,\beta\kappa'}^{II}(Q) \xi'_{\beta\kappa'} = M_\kappa \omega^2 \xi'_{\alpha\kappa} \quad (134)$$