Université catholique de Louvain
Ecole Polytechnique de Louvain

:

Consistency check for Diamond

Samuel Poncé

Xavier Gonze
Paul Boulanger
Gabriel Antonius
Michel Coté

November 8, 2012

Contents

1 Introduction

The total energy can be written as:

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

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

where f_i if 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:

$$\varepsilon_i = \langle\varphi_i|H|\varphi_i\rangle \tag{3}$$

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

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

1.1 Janak theorem

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

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

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:

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

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

Finite difference on 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 :

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

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

The Allen-Heine-Cardona method :

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

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:

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

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

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

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.

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

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:

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

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

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:

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

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

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:

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

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

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

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:

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

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

We then have:

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

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

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

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

Using the Janak/Brooks theorem we have:

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

with f_n the occupation factor.

Using equation (22) we have:

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

We then have:

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

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

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.

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

charge	f_{Homo}(x3)	Etot[Ha]	ω[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:

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

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

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

Using (25) we have at 0K:

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

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:

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

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

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

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

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

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

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

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

with $f = f(x_1,x_2)$

We then have the following relationship:

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

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

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

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 $\left|{}^6\xi\right| = 1$ and this mean $\left|{}^3\xi_1\right| = \left|{}^3\xi_2\right| = \frac{1}{\sqrt{2}}$

In our case we have:

We can make the following parallelism by Taylor expanding ε_n:

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

charge	Etot[Ha]	$\varepsilon_{\rm HOMO}$[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 similarily for the other. We thus obtain:

$$\frac{\partial^2 f(x_1,x_2)}{\partial x_1^2} = \frac{f(x_1+h,x_2)+f(x_1-h,x_2)-2f(x_1,x_2)}{h^2}$$
$$= 2\sum_{\alpha\alpha'} {}^3\xi_1(\alpha)\frac{\partial^2\varepsilon_n}{\partial R_{1\alpha}\partial R_{1\alpha'}}{}^3\xi_1(\alpha') = 0.28098533333261 \text{ Ha} \quad (45)$$
$$\frac{\partial^2 f(x_1,x_2)}{\partial x_2^2} = \frac{f(x_1,x_2+w)+f(x_1,x_2-w)-2f(x_1,x_2)}{h^2}$$
$$= 2\sum_{\alpha\alpha'} {}^3\xi_2(\alpha)\frac{\partial^2\varepsilon_n}{\partial R_{2\alpha}\partial R_{2\alpha'}}{}^3\xi_2(\alpha') = 0.28098633333418 \text{ Ha} \quad (46)$$

Now we have to consider the off diagonal terms. As we are only considering the optical mode we have ${}^3\xi_m 1 = -{}^3\xi_m 2$ therefore leading to:

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

And similarily for the other. We thus obtain:

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

The zero point motion correction is then:

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

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 $\epsilon_n(T) - \epsilon_{\text{without el-ph}} = 23.5012772629637$ 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.

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

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

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

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:

$$\delta\varepsilon^{\mathrm{NDDW}} = 4.64174889748 \text{ meV} \tag{53}$$

We then have a total correction of $23.5012772629637 + 4.64174889748 = 28.1430261604437$ 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):

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

where $\Delta R^a_{\kappa\alpha}$ is the displacement along direction α of the atom κ in the cell labelled a.

It's Fourier transform is:

$$\tilde{C}_{\kappa\alpha,\kappa'\alpha'}(\vec{q}) = \frac{1}{N} \sum_{aa'} C_{\kappa\alpha,\kappa'\alpha'}(a,a') e^{-iq\cdot(R_a - R_{a'})} \tag{55}$$

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

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 $U_m(\kappa\alpha a) = e^{iq\cdot R_a} U_{mq}(\kappa\alpha)$ and are solution of the following eigenvalue problem equation:

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

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

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

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

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:

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

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:

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

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

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

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).

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

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

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

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

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

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

$$\tag{70}$$

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

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

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

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

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

$$\tag{74}$$

where the prefactor is there to keep normalization:

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

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 relationship[1]:

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

$$k_1 = 2\pi \frac{a_2 \times a_3}{a_1 \cdot a_2 \times a_3} \qquad k_2 = 2\pi \frac{a_3 \times a_1}{a_2 \cdot a_3 \times a_1} \qquad k_3 = 2\pi \frac{a_1 \times a_2}{a_1 \cdot a_2 \times a_2} \tag{76}$$

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

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:

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

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 $\vec{q}\cdot\vec{R} = \begin{pmatrix} 0.5 & 0.0 & 0.0 \end{pmatrix}$ 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).

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

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

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

3.1.3 Example

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

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

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

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-$\frac{\sqrt{3}}{2}\Delta_n$

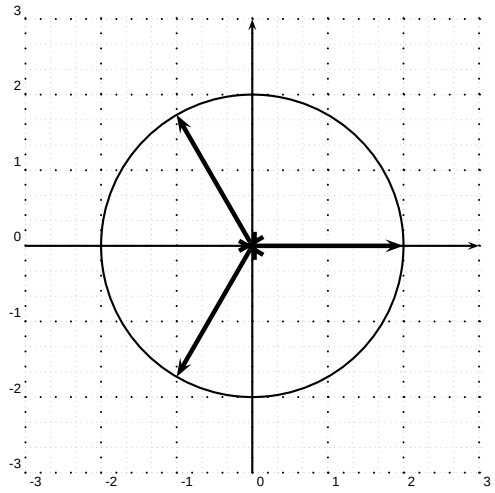

Figure 1: Vector representation in the complex plane

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]	$\varepsilon_\Gamma^{\mathrm{HOMO}}$			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
					0.0048009	$\mathcal{O}(h^4)$	-8.64119034
LO	-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
					0.0024534	$\mathcal{O}(h^4)$	-7.61042338
TO	-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 SchrÃ¶dinger equation can be split into an unperturbed Hamiltonian an a perturbed one:

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

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

$$E =< N|H_0|N > + \sum_Q \hbar\omega_Q \left[< n_Q > + \frac{1}{2} \right] \tag{84}$$

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: $\frac{\partial E}{\partial < n_i >} = \hbar\omega_i$ for each mode i.

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

$$E - E_{\text{without e-ph}} = \delta E = \sum_Q \frac{\partial E}{\partial < n_i >} \left[\langle n_q \rangle + \frac{1}{2} \right] \tag{85}$$

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

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

$$\delta\varepsilon_n = \sum_Q \frac{\partial \varepsilon_n}{\partial < n_Q >} \left[< n_Q > + \frac{1}{2} \right] \tag{87}$$

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

$$< n_Q > =< N|a_Q^+ a_Q|N > \tag{88}$$

$$\varepsilon_n =< n_i|H_0|n_i > + \sum_Q \hbar\omega_Q \left[< n_i|a_Q^+ a_Q|n_i > + \frac{1}{2} \right] \tag{89}$$

3.3.2 For the L point

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

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

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

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

and also:

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

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:

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

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:

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

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

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

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

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

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 $\hbar = 1$):

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

As we can see $\frac{\partial \varepsilon_{\Gamma n}^{\mathrm{NDDW}}}{\partial n_{\vec{q}j}} = 0$ 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:

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

we obtain

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

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:

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

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:

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

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

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

Let us define a collective translation displacement as:

$$\frac{\partial}{\partial T_\beta} \triangleq \sum_{\kappa'} \frac{\partial}{\partial R_{\kappa'\beta}} \tag{105}$$

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

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

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 :

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

$$\frac{\partial}{\partial S_\beta} \triangleq \sum_{\kappa\alpha} S_{\beta\kappa\alpha}\frac{\partial}{\partial R_{\kappa\alpha}} \tag{108}$$

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:

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

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:

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

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:

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

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
		-4.68829685445	-7.50473030456	num. err.	0.0005
LO	0.005785943	-6.65295481802	-6.22402647327	-0.428928344	0.05
		-6.65429556431	-6.22686242355	-0.427433140	0.01
		-6.65446148352	-6.22697561364	-0.427485869	0.002
			-6.2269805723	-0.427480911	0.002 and 0.004
		-6.65440525359	-6.22697527827	num. err.	0.0005
TA	0.002453274	-17.2206027567	-14.6794488903	-2.541153866	0.05
		-17.233165437	-14.6858244554	-2.547340981	0.01
		-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
TO	0.005671174	-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 shows by Taylor expanding each term that if we use the following relationship:

$$D_{i,k} = \frac{D_{i,k-1} - \frac{1}{2^{2k}} D_{i-1,k-1}}{1 - \frac{1}{2^{2k}}} \tag{112}$$

then for each iteration we increasing 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	49.547116	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:

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

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:

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

$$= \frac{1}{4\omega_{qj}M_c} \sum_{\kappa\alpha\kappa'\beta} \frac{\partial^2\varepsilon}{\partial R_{\kappa\alpha}\partial R_{\kappa'\beta}} \left[(\xi^r_{\kappa\alpha} - i\xi^i_{\kappa\alpha})(\xi^r_{\kappa'\beta} + i\xi^i_{\kappa'\beta})(\cos(q\cdot(R_{\kappa'} - R_\kappa)) + i\sin(q\cdot(R_{\kappa'} - R_\kappa)))\right] \tag{114}$$

with $\xi^r_{\kappa\alpha}$ and $\xi^i_{\kappa\alpha}$ 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:

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

$$= \frac{1}{4\omega_{qj}M_c} \sum_{\kappa\alpha\kappa'\beta} \frac{\partial^2\varepsilon}{\partial R_{\kappa\alpha}\partial R_{\kappa'\beta}} \left[U^r_{\kappa\alpha}U^r_{\kappa'\beta} + U^i_{\kappa\alpha}U^i_{\kappa'\beta} + i\left(U^r_{\kappa\alpha}U^i_{\kappa'\beta} - U^i_{\kappa\alpha}U^r_{\kappa'\beta}\right)\right] \tag{116}$$

$$U^r_{\kappa\alpha} = U^r_{\kappa\alpha}\cos(q\cdot R_\kappa) - U^i_{\kappa\alpha}\sin(q\cdot R_\kappa) \tag{117}$$

$$U^i_{\kappa\alpha} = U^i_{\kappa\alpha}\cos(q\cdot R_\kappa) + U^r_{\kappa\alpha}\sin(q\cdot R_\kappa) \tag{118}$$

The imaginary part of equation (116) vannish when we do the full summation over $\kappa\alpha\kappa'\beta$.
The frequency are obtained from the secular equation:

$$\omega_m^2 = \sum_{\kappa\alpha\kappa'\beta} \frac{1}{M_c} \xi^*_{\kappa\alpha} \frac{\partial^2 E}{\partial R_{\kappa\alpha} \partial R_{\kappa'\beta}} \xi_{\kappa'\beta} \tag{119}$$

$$= \sum_{\kappa\alpha\kappa'\beta} \frac{1}{M_c} \frac{\partial^2 E}{\partial R_{\kappa\alpha} \partial R_{\kappa'\beta}} (\xi^r_{\kappa\alpha} - i\xi^i_{\kappa\alpha})(\xi^r_{\kappa'\beta} + i\xi^i_{\kappa'\beta}) \tag{120}$$

$$= \sum_{\kappa\alpha\kappa'\beta} \frac{1}{M_c} \frac{\partial^2 E}{\partial R_{\kappa\alpha} \partial R_{\kappa'\beta}} (\xi^r_{\kappa\alpha}\xi^r_{\kappa'\beta} + \xi^i_{\kappa\alpha}\xi^i_{\kappa'\beta} + i(\xi^r_{\kappa\alpha}\xi^i_{\kappa'\beta} - \xi^i_{\kappa\alpha}\xi^r_{\kappa'\beta})) \tag{121}$$

4.3 NDDW

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

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:

$$\frac{\partial}{\partial T_\beta} \equiv \sum_{\kappa'} \frac{\partial}{\partial R_{\kappa'\beta}} \tag{123}$$

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

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

Then the equation reduces itself to:

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

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

```
# C in diamond structure; 2x2x2 FCC special point grid; low ecut.
# psp         6c.pspnc
# Mcarbone    21894.16693
ndtset 2

#Dataset 1 : ground state density
nqpt1      0
ieig2rf1   0
smdelta1   0
rfphon1    0
getwfk1    0         # Use GS wave functions from dataset1

nqpt       1
ieig2rf    1
bdeigrf    8
smdelta    1
getwfk     1         # Use GS wave functions from dataset1
kptopt     3         # Need full k-point set for finite-Q response
rfphon     1         # Do phonon response
rfatpol    1 2       # Treat displacements of all atoms
rfdir      1 1 1     # Do all directions (symmetry will be used)

#Dataset 2-5 : phonon frequencies and band corrections
qpt2  0.0 0.0 0.0
rfasr2 1

#Size-dependent parameters
 acell 3*6.70346805
 rprim  0 .5 .5 .5 0 .5 .5 .5 0

 natom 2
 typat 1 1
 xred 3*0.00d0 3*0.25d0

 nband 32*8

 ngkpt 2 2 2
 nshiftk 4
 shiftk 0.0 0.0 0.0
        0.0 0.5 0.5
        0.5 0.0 0.5
        0.5 0.5 0.0
 occopt 2
 occ 4*2.0 4*0.0 #1
     4*2.0 4*0.0 #2
     4*2.0 4*0.0 #3
         ...
     4*2.0 4*0.0 #18
     4*2.0 4*0.0 #19
#    4*2.0 3*0.02 0.0 # Gamma point LUMO 3x
     2.0 3*1.99 4*0.0 # Gamma point HOMO 3x
     4*2.0 4*0.0
     4*2.0 4*0.0
         ...
     4*2.0 4*0.0
# charge -0.0009375 #for 0.01 LUMO 3x
 charge 0.0009375  #for 0.01 HOMO 3x


# Miscellaneous
 ntypat 1
 znucl 6
 diemac 6.0d0
 ecut 30
 enunit 2
 nstep 40
 nsym 1
 tolwfr 1.0d-18
```

Code 1: ABINIT input file

```
#   C in diamond structure; 2x2x2 FCC special point grid; low ecut.

ndtset 9

getwfk    -1         # Use GS wave functions from dataset1
```

```
kptopt     3        # Need full k-point set for finite-Q response


#Size-dependent parameters
 acell 3*6.70346805
 rprim  0 .5 .5 .5 0 .5 .5 .5 0

 natom 2
 typat 1 1

 xcart1  3*0.0   3*1.675867013     # f(x1,x2)
 xcart2  3*0.01  3*1.675867013     # f(x1+h,x2)
 xcart3  3*-0.01 3*1.675867013     # f(x1-h,x2)
 xcart4  3*0.0   3*1.685867013     # f(x1,x2+w)
 xcart5  3*0.0   3*1.665867013     # f(x1,x2-w)
 xcart6  3*0.01  3*1.685867013     # f(x1+h,x2+w)
 xcart7  3*-0.01 3*1.665867013     # f(x1-h,x2-w)
 xcart8  3*0.01  3*1.665867013     # f(x1+h,x2-w)
 xcart9  3*-0.01 3*1.685867013     # f(x1-h,x2+w)

 nband 8

 ngkpt 2 2 2
 nshiftk 4
 shiftk 0.0 0.0 0.0
        0.0 0.5 0.5
        0.5 0.0 0.5
        0.5 0.5 0.0


# Miscellaneous
 ntypat 1
 znucl 6
 diemac 6.0d0
 ecut 30
 enunit 2
 nstep 40
 nsym 1
 tolwfr 1.0d-18
```

Code 2: ABINIT input file

```
#   C in diamond structure; 2x2x2 FCC special point grid; low ecut.

ndtset 2


#Dataset 1 : ground state density
nqpt1      0
ieig2rf1   0
smdelta1   0
rfphon1    0
getwfk1    0         # Use GS wave functions from dataset1
#ecutsm1    0.5

nqpt       1
ieig2rf    1
bdeigrf    8
smdelta    1
getwfk     1         # Use GS wave functions from dataset1
kptopt     3         # Need full k-point set for finite-Q response
rfphon     1         # Do phonon response
rfatpol    1 2       # Treat displacements of all atoms
rfdir      1 1 1     # Do all directions (symmetry will be used)

#Dataset 2-5 : phonon frequencies and band corrections
qpt2   0.0 0.0 0.0

#Size-dependent parameters
 acell 3*6.70346805 #relaxed param.
 rprim  0 .5 .5 .5 0 .5 .5 .5 0

 natom 2
 typat 1 1
 xred 3*0.00d0 3*0.25d0

 nband 8

 ngkpt 2 2 2
 nshiftk 4
 shiftk 0.0 0.0 0.0
        0.0 0.5 0.5
```

```
       0.5 0.0 0.5
       0.5 0.5 0.0

# Miscellaneous
 ntypat 1
 znucl 6
 diemac 6.0d0
 ecut 30
 enunit 2
 nstep 30
 nsym 1
 tolwfr 1.0d-16
```

Code 3: ABINIT input file

```
!Input file for the anaddb code. Analysis of the Diamond DDB

!Flags
 ifcflag  1    ! Interatomic force constant flag
 thmflag 3
 telphint 1
 ntemper 10
 tempermin 100
 temperinc 100
!Wavevector grid number 1 (coarse grid, from DDB)
  brav   1     ! Bravais Lattice : 1-S.C., 2-F.C., 3-B.C., 4-Hex.)
  ngqpt  1 1 1  ! Monkhorst-Pack indices
  nqshft 1        ! number of q-points in repeated basic q-cell
  q1shft 3*0.0
         0.0 0.5 0.5
         0.5 0.0 0.5
         0.5 0.5 0.0

!Effective charges
     asr  1    ! Acoustic Sum Rule. 1 => imposed asymetrically
  chneut  1    ! Charge neutrality requirement for effective charges.

!Interatomic force constant info
  dipdip 0     ! Dipole-dipole interaction treatment

!Wavevector list number 1 (Reduced coordinates and normalization factor)
  nph1l   1    ! number of phonons in list 1
  qph1l  0.00000000E+00 0.00000000E+00 0.00000000E+00 1.0
```

Code 4: ANADDB input file

5.2 Dynamical matrix

```
   1    1   1    1        0.4875111415       0.0000000000
   1    1   2    1       -0.0684816112       0.0000000000
   1    1   3    1       -0.0684816108       0.0000000000
   1    1   1    2       -0.2259726921       0.0000000000
   1    1   2    2       -0.0580165771       0.0000000000
   1    1   3    2       -0.0580165772       0.0000000000

   2    1   1    1       -0.0684816113       0.0000000000
   2    1   2    1        0.4875111415       0.0000000000
   2    1   3    1        0.0684816108       0.0000000000
   2    1   1    2       -0.0580165771       0.0000000000
   2    1   2    2       -0.2259726919       0.0000000000
   2    1   3    2        0.0580165774       0.0000000000

   3    1   1    1       -0.0684816109       0.0000000000
   3    1   2    1        0.0684816109       0.0000000000
   3    1   3    1        0.4875111412       0.0000000000
   3    1   1    2       -0.0580165772       0.0000000000
   3    1   2    2        0.0580165773       0.0000000000
   3    1   3    2       -0.2259726918       0.0000000000

   1    2   1    1       -0.2259726919       0.0000000000
   1    2   2    1       -0.0580165771       0.0000000000
   1    2   3    1       -0.0580165772       0.0000000000
   1    2   1    2        0.4875113661       0.0000000000
   1    2   2    2       -0.0684816112       0.0000000000
   1    2   3    2       -0.0684816109       0.0000000000

   2    2   1    1       -0.0580165773       0.0000000000
   2    2   2    1       -0.2259726919       0.0000000000
```

```
   2   2   3   1        0.0580165773       0.0000000000
   2   2   1   2       -0.0684816111       0.0000000000
   2   2   2   2        0.4875113663       0.0000000000
   2   2   3   2        0.0684816109       0.0000000000

   3   2   1   1       -0.0580165774       0.0000000000
   3   2   2   1        0.0580165774       0.0000000000
   3   2   3   1       -0.2259726918       0.0000000000
   3   2   1   2       -0.0684816108       0.0000000000
   3   2   2   2        0.0684816109       0.0000000000
   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.

```
# Find the eigenvector from the dynamical matrice

import numpy as np

def read_dynamical_matrix(datafile):
       with open(datafile,'r') as f:
         raw = list()
         for line in f:
           if line[0]=='#': continue
           parts=line.split()
           if not parts: continue
           val=float(parts[4])
           if np.abs(val) < 0.000000001:
             val=0.0
           raw.append(val)
         dynmat = np.array(raw)
         dynmat = dynmat.reshape(6,6)
       return dynmat


L_file = '/home/Samuel/Dropbox/WorkDiam/DerivEig_L/Dyanmical_mat.dat'

dynmatL=read_dynamical_matrix(L_file)

[eigvalL,eigvectL]=np.linalg.eig(dynmatL)

eigvectL=np.transpose(eigvectL)
Mc=21894.16693

print "eigenvalues at L :"
for n in eigvalL:
  a = np.sqrt(n/Mc)
  print a

print "eigenvectors at L :"
for n in eigvectL: print n

# Create the xcart needed for abinit input file

disp = 0.03
rprim = np.array([13.4069361,6.70346805,6.70346805])


print "# Equilibrium position"
print " xcart1 0.0 0.0  0.0                    # 0.0 0.0 0.0 "
print "        1.67586701 1.67586701 1.67586701 # 0.125 0.25 0.25"
print "        6.70346805 0.00 0.00             # 0.5 0.0 0.0 "
print "        8.37933506 1.67586701 1.67586701 # 0.625 0.25 0.25 "
print " "
print "# LA mode"
i = 0
print " xcart2 %.10f %.10f %.10f "%(rprim[0]*0.0+eigvectL[i][0]*disp,
rprim[1]*0.0+eigvectL[i][1]*disp,rprim[2]*0.0+eigvectL[i][2]*disp)
print "        %.10f %.10f %.10f "%(rprim[0]*0.125+eigvectL[i][3]*disp,
rprim[1]*0.25+eigvectL[i][4]*disp,rprim[2]*0.25+eigvectL[i][5]*disp)
print "        %.10f %.10f %.10f "%(rprim[0]*0.5-eigvectL[i][0]*disp,
 rprim[1]*0.0-eigvectL[i][1]*disp,rprim[2]*0.0-eigvectL[i][2]*disp)
print "        %.10f %.10f %.10f "%(rprim[0]*0.625-eigvectL[i][3]*disp,
 rprim[1]*0.25-eigvectL[i][4]*disp,rprim[2]*0.25-eigvectL[i][5]*disp)
print ""
print "# LO mode "
i = 1
```

```
print " xcart3 %.10f %.10f %.10f  "%(rprim[0]*0.0+eigvectL[i][0]*disp,
rprim[1]*0.0+eigvectL[i][1]*disp,rprim[2]*0.0+eigvectL[i][2]*disp)
print "        %.10f %.10f %.10f  "%(rprim[0]*0.125+eigvectL[i][3]*disp,
 rprim[1]*0.25+eigvectL[i][4]*disp,rprim[2]*0.25+eigvectL[i][5]*disp)
print "        %.10f %.10f %.10f  "%(rprim[0]*0.5-eigvectL[i][0]*disp,
rprim[1]*0.0-eigvectL[i][1]*disp,rprim[2]*0.0-eigvectL[i][2]*disp)
print "        %.10f %.10f %.10f  "%(rprim[0]*0.625-eigvectL[i][3]*disp,
rprim[1]*0.25-eigvectL[i][4]*disp,rprim[2]*0.25-eigvectL[i][5]*disp)
print ""
print "# TA mode "
i = 2
...
```

Code 6: Python script for solving the eigenvalue problem.

.

5.4 NDDW using finite difference at the q=L point

```
# Find the eigenvector from the dynamical matrice

import numpy as np

def read_dynamical_matrix(datafile):
       with open(datafile,'r') as f:
         raw = list()
         for line in f:
           if line[0]=='#': continue
           parts=line.split()
           if not parts: continue
           val=float(parts[4])
           if np.abs(val) < 0.000000001:
             val=0.0
           raw.append(val)
         dynmat = np.array(raw)
         dynmat = dynmat.reshape(6,6)
       return dynmat


L_file = '/home/Samuel/Dropbox/WorkDiam/DerivEig_L/Dyanmical_mat.dat'

dynmatL=read_dynamical_matrix(L_file)

[eigvalL,eigvectL]=np.linalg.eig(dynmatL)

eigvectL=np.transpose(eigvectL)
Mc=21894.16693

print "eigenvalues at L :"
for n in eigvalL:
  a = np.sqrt(n/Mc)
  print a

print "eigenvectors at L :"
for n in eigvectL: print n

# Create the xcart needed for abinit input file

disp = 0.01
acell = np.array([13.4069361,6.70346805,6.70346805])
rprim = np.array([[0.0,0.5,0.5],[0.5,0.0,0.5],[0.5,0.5,0.0]])

cell = np.mat([rprim[0]*acell[0],rprim[1]*acell[1],rprim[2]*acell[2]])
print "Cell geometry: ", cell

#Reduce coordinate of atoms

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]])
print xred
print (xred[1]*cell)[0,0]


#Count for nb of xcart
j = 1

print "# Equilibrium position"
print " xcart%.0f  %.10f %.10f %.10f  "%(j,(xred[0]*cell)[0,0],(xred[0]*cell)[0,1]
,(xred[0]*cell)[0,2])
print "         %.10f %.10f %.10f  "%((xred[1]*cell)[0,0],(xred[1]*cell)[0,1],(xre
d[1]*cell)[0,2])
```

```
print "          %.10f  %.10f  %.10f  "%((xred[2]*cell)[0,0],(xred[2]*cell)[0,1],(xre
d[2]*cell)[0,2])
print "          %.10f  %.10f  %.10f  "%((xred[3]*cell)[0,0],(xred[3]*cell)[0,1],(xre
d[3]*cell)[0,2])

j += 1

# LA mode (+h)
i = 0
print " xcart%.0f  %.10f  %.10f  %.10f  "%(j,(xred[0]*cell)[0,0]+eigvectL[i][0]*disp
,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,(xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
print "          %.10f  %.10f  %.10f  "%((xred[1]*cell)[0,0]+eigvectL[i][3]*disp,(xre
d[1]*cell)[0,1]+eigvectL[i][4]*disp,(xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
print "          %.10f  %.10f  %.10f  "%((xred[2]*cell)[0,0]-eigvectL[i][0]*disp,(xre
d[2]*cell)[0,1]-eigvectL[i][1]*disp,(xred[2]*cell)[0,2]-eigvectL[i][2]*disp)
print "          %.10f  %.10f  %.10f  "%((xred[3]*cell)[0,0]-eigvectL[i][3]*disp,(xre
d[3]*cell)[0,1]-eigvectL[i][4]*disp,(xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
j += 1

# LA mode (-h)
print " xcart%.0f  %.10f  %.10f  %.10f  "%(j,(xred[0]*cell)[0,0]-eigvectL[i][0]*disp
,(xred[0]*cell)[0,1]-eigvectL[i][1]*disp,(xred[0]*cell)[0,2]-eigvectL[i][2]*disp)
print "          %.10f  %.10f  %.10f  "%((xred[1]*cell)[0,0]-eigvectL[i][3]*disp,(xre
d[1]*cell)[0,1]-eigvectL[i][4]*disp,(xred[1]*cell)[0,2]-eigvectL[i][5]*disp)
print "          %.10f  %.10f  %.10f  "%((xred[2]*cell)[0,0]+eigvectL[i][0]*disp,(xre
d[2]*cell)[0,1]+eigvectL[i][1]*disp,(xred[2]*cell)[0,2]+eigvectL[i][2]*disp)
print "          %.10f  %.10f  %.10f  "%((xred[3]*cell)[0,0]+eigvectL[i][3]*disp,(xre
d[3]*cell)[0,1]+eigvectL[i][4]*disp,(xred[3]*cell)[0,2]+eigvectL[i][5]*disp)
j += 1

# LO mode (+h)
i = 1
print " xcart%.0f  %.10f  %.10f  %.10f  "%(j,(xred[0]*cell)[0,0]+eigvectL[i][0]*disp
,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,(xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
print "          %.10f  %.10f  %.10f  "%((xred[1]*cell)[0,0]+eigvectL[i][3]*disp,(xre
d[1]*cell)[0,1]+eigvectL[i][4]*disp,(xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
print "          %.10f  %.10f  %.10f  "%((xred[2]*cell)[0,0]-eigvectL[i][0]*disp,(xre
d[2]*cell)[0,1]-eigvectL[i][1]*disp,(xred[2]*cell)[0,2]-eigvectL[i][2]*disp)
print "          %.10f  %.10f  %.10f  "%((xred[3]*cell)[0,0]-eigvectL[i][3]*disp,(xre
d[3]*cell)[0,1]-eigvectL[i][4]*disp,(xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
j += 1

# LO mode (-h)
print " xcart%.0f  %.10f  %.10f  %.10f  "%(j,(xred[0]*cell)[0,0]-eigvectL[i][0]*disp
,(xred[0]*cell)[0,1]-eigvectL[i][1]*disp,(xred[0]*cell)[0,2]-eigvectL[i][2]*disp)
print "          %.10f  %.10f  %.10f  "%((xred[1]*cell)[0,0]-eigvectL[i][3]*disp,(xre
d[1]*cell)[0,1]-eigvectL[i][4]*disp,(xred[1]*cell)[0,2]-eigvectL[i][5]*disp)
print "          %.10f  %.10f  %.10f  "%((xred[2]*cell)[0,0]+eigvectL[i][0]*disp,(xre
d[2]*cell)[0,1]+eigvectL[i][1]*disp,(xred[2]*cell)[0,2]+eigvectL[i][2]*disp)
print "          %.10f  %.10f  %.10f  "%((xred[3]*cell)[0,0]+eigvectL[i][3]*disp,(xre
d[3]*cell)[0,1]+eigvectL[i][4]*disp,(xred[3]*cell)[0,2]+eigvectL[i][5]*disp)
j += 1

# TA mode (+h)
i = 2
print " xcart%.0f  %.10f  %.10f  %.10f  "%(j,(xred[0]*cell)[0,0]+eigvectL[i][0]*disp
,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,(xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
print "          %.10f  %.10f  %.10f  "%((xred[1]*cell)[0,0]+eigvectL[i][3]*disp,(xre
d[1]*cell)[0,1]+eigvectL[i][4]*disp,(xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
print "          %.10f  %.10f  %.10f  "%((xred[2]*cell)[0,0]-eigvectL[i][0]*disp,(xre
d[2]*cell)[0,1]-eigvectL[i][1]*disp,(xred[2]*cell)[0,2]-eigvectL[i][2]*disp)
print "          %.10f  %.10f  %.10f  "%((xred[3]*cell)[0,0]-eigvectL[i][3]*disp,(xre
d[3]*cell)[0,1]-eigvectL[i][4]*disp,(xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
j += 1

# TA mode (-h)
print " xcart%.0f  %.10f  %.10f  %.10f  "%(j,(xred[0]*cell)[0,0]-eigvectL[i][0]*disp
,(xred[0]*cell)[0,1]-eigvectL[i][1]*disp,(xred[0]*cell)[0,2]-eigvectL[i][2]*disp)
print "          %.10f  %.10f  %.10f  "%((xred[1]*cell)[0,0]-eigvectL[i][3]*disp,(xre
d[1]*cell)[0,1]-eigvectL[i][4]*disp,(xred[1]*cell)[0,2]-eigvectL[i][5]*disp)
print "          %.10f  %.10f  %.10f  "%((xred[2]*cell)[0,0]+eigvectL[i][0]*disp,(xre
d[2]*cell)[0,1]+eigvectL[i][1]*disp,(xred[2]*cell)[0,2]+eigvectL[i][2]*disp)
print "          %.10f  %.10f  %.10f  "%((xred[3]*cell)[0,0]+eigvectL[i][3]*disp,(xre
d[3]*cell)[0,1]+eigvectL[i][4]*disp,(xred[3]*cell)[0,2]+eigvectL[i][5]*disp)
j += 1

# TA mode (2ieme) (+h)
i = 3
print " xcart%.0f  %.10f  %.10f  %.10f  "%(j,(xred[0]*cell)[0,0]+eigvectL[i][0]*disp
,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,(xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
print "          %.10f  %.10f  %.10f  "%((xred[1]*cell)[0,0]+eigvectL[i][3]*disp,(xre
d[1]*cell)[0,1]+eigvectL[i][4]*disp,(xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
print "          %.10f  %.10f  %.10f  "%((xred[2]*cell)[0,0]-eigvectL[i][0]*disp,(xre
d[2]*cell)[0,1]-eigvectL[i][1]*disp,(xred[2]*cell)[0,2]-eigvectL[i][2]*disp)
```

```
print "          %.10f %.10f %.10f "%((xred[3]*cell)[0,0]-eigvectL[i][3]*disp,(xre
d[3]*cell)[0,1]-eigvectL[i][4]*disp,(xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
j += 1

# TA mode (-h)
print " xcart%.0f  %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]-eigvectL[i][0]*disp
,(xred[0]*cell)[0,1]-eigvectL[i][1]*disp,(xred[0]*cell)[0,2]-eigvectL[i][2]*disp)
print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]-eigvectL[i][3]*disp,(xre
d[1]*cell)[0,1]-eigvectL[i][4]*disp,(xred[1]*cell)[0,2]-eigvectL[i][5]*disp)
print "          %.10f %.10f %.10f "%((xred[2]*cell)[0,0]+eigvectL[i][0]*disp,(xre
d[2]*cell)[0,1]+eigvectL[i][1]*disp,(xred[2]*cell)[0,2]+eigvectL[i][2]*disp)
print "          %.10f %.10f %.10f "%((xred[3]*cell)[0,0]+eigvectL[i][3]*disp,(xre
d[3]*cell)[0,1]+eigvectL[i][4]*disp,(xred[3]*cell)[0,2]+eigvectL[i][5]*disp)
j += 1

# TO mode (+h)
i = 4
print " xcart%.0f  %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]+eigvectL[i][0]*disp
,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,(xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]+eigvectL[i][3]*disp,(xre
d[1]*cell)[0,1]+eigvectL[i][4]*disp,(xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
print "          %.10f %.10f %.10f "%((xred[2]*cell)[0,0]-eigvectL[i][0]*disp,(xre
d[2]*cell)[0,1]-eigvectL[i][1]*disp,(xred[2]*cell)[0,2]-eigvectL[i][2]*disp)
print "          %.10f %.10f %.10f "%((xred[3]*cell)[0,0]-eigvectL[i][3]*disp,(xre
d[3]*cell)[0,1]-eigvectL[i][4]*disp,(xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
j += 1

# TO mode (-h)
print " xcart%.0f  %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]-eigvectL[i][0]*disp
,(xred[0]*cell)[0,1]-eigvectL[i][1]*disp,(xred[0]*cell)[0,2]-eigvectL[i][2]*disp)
print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]-eigvectL[i][3]*disp,(xre
d[1]*cell)[0,1]-eigvectL[i][4]*disp,(xred[1]*cell)[0,2]-eigvectL[i][5]*disp)
print "          %.10f %.10f %.10f "%((xred[2]*cell)[0,0]+eigvectL[i][0]*disp,(xre
d[2]*cell)[0,1]+eigvectL[i][1]*disp,(xred[2]*cell)[0,2]+eigvectL[i][2]*disp)
print "          %.10f %.10f %.10f "%((xred[3]*cell)[0,0]+eigvectL[i][3]*disp,(xre
d[3]*cell)[0,1]+eigvectL[i][4]*disp,(xred[3]*cell)[0,2]+eigvectL[i][5]*disp)
j += 1

# TO mode (2ieme) (+h)
i = 5
print " xcart%.0f  %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]+eigvectL[i][0]*disp
,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,(xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]+eigvectL[i][3]*disp,(xre
d[1]*cell)[0,1]+eigvectL[i][4]*disp,(xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
print "          %.10f %.10f %.10f "%((xred[2]*cell)[0,0]-eigvectL[i][0]*disp,(xre
d[2]*cell)[0,1]-eigvectL[i][1]*disp,(xred[2]*cell)[0,2]-eigvectL[i][2]*disp)
print "          %.10f %.10f %.10f "%((xred[3]*cell)[0,0]-eigvectL[i][3]*disp,(xre
d[3]*cell)[0,1]-eigvectL[i][4]*disp,(xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
j += 1

# TO mode (-h)
print " xcart%.0f  %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]-eigvectL[i][0]*disp
,(xred[0]*cell)[0,1]-eigvectL[i][1]*disp,(xred[0]*cell)[0,2]-eigvectL[i][2]*disp)
print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]-eigvectL[i][3]*disp,(xre
d[1]*cell)[0,1]-eigvectL[i][4]*disp,(xred[1]*cell)[0,2]-eigvectL[i][5]*disp)
print "          %.10f %.10f %.10f "%((xred[2]*cell)[0,0]+eigvectL[i][0]*disp,(xre
d[2]*cell)[0,1]+eigvectL[i][1]*disp,(xred[2]*cell)[0,2]+eigvectL[i][2]*disp)
print "          %.10f %.10f %.10f "%((xred[3]*cell)[0,0]+eigvectL[i][3]*disp,(xre
d[3]*cell)[0,1]+eigvectL[i][4]*disp,(xred[3]*cell)[0,2]+eigvectL[i][5]*disp)
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

```
import math
import os
import numpy as np

# Transform binary VHXC file into text file

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
              files.write("outputo_DS1_VHXC \n")
              files.write("1 \n")
              files.write("5 \n") #3D formatted data (output the bare 3D data - one column)
              files.write("EQ \n")
              files.write("0 \n")
```

```
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS2_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("LA+ \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS3_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("LA- \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS4_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("LO+ \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS5_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("LO- \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS6_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("TA1+ \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS7_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("TA1- \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS8_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("TA2+ \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS9_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("TA2- \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS10_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("TO1+ \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
```

```
with open("cut3d.files","a") as files:
               files.write("outputo_DS11_VHXC \n")
               files.write("1 \n")
               files.write("5 \n")
               files.write("TO1- \n")
               files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
               files.write("outputo_DS12_VHXC \n")
               files.write("1 \n")
               files.write("5 \n")
               files.write("TO2+ \n")
               files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
               files.write("outputo_DS13_VHXC \n")
               files.write("1 \n")
               files.write("5 \n")
               files.write("TO2- \n")
               files.write("0 \n")
os.system("cut3d < cut3d.files ")

# ------------------------------------------------------------------------
# Make centered finite difference of order 2 of Vhxc for each mode
# d2Vhxc/dR1dR2=(2*Vhxc(mode)-2*Vhxc(eq))/(h^2)
# ------------------------------------------------------------------------

mode = np.array(["EQ","LA+","LA-","LO+","LO-","TA1+","TA1-","TA2+","TA2-",
"TO1+","TO1-","TO2+","TO2-"])

# The phonon frequency are eigenvalue of the Dynamical eigenvalue problem.

omega = np.array([0.0,0.00480084,0.00480084,0.005785943,0.005785943,0.002453274,
0.002453274,0.002453274,0.002453274,0.005671174,0.005671174,0.005671174,0.005671174])

# ------------------------------------------------------------------------------
# Prepare the integration : de2/dR1dR2=int d2Vhxc/dR1dR2 psi*psi(unperturbed)
# ------------------------------------------------------------------------------

os.system("rm wfk.files")
with open("wfk.files","a") as files:
               files.write("outputo_DS1_WFK \n") #unperturbed wfk
               files.write("1 \n")
               files.write("0 \n")
               files.write("4 \n") # Gamma point
               files.write("6 \n") # Band nb 6
               files.write("0 \n")
               files.write("0 \n")
               files.write("2 \n") # real 3D data one column
               files.write("wfk \n")
               files.write("0 \n")
os.system("cut3d < wfk.files ")

os.system("rm wfk.files")
with open("wfk.files","a") as files:
               files.write("outputo_DS1_WFK \n") #unperturbed wfk
               files.write("1 \n")
               files.write("0 \n")
               files.write("4 \n") # Gamma point
               files.write("7 \n") # Band nb 7
               files.write("0 \n")
               files.write("0 \n")
               files.write("2 \n") # real 3D data one column
               files.write("wfk \n")
               files.write("0 \n")
os.system("cut3d < wfk.files ")

os.system("rm wfk.files")
with open("wfk.files","a") as files:
               files.write("outputo_DS1_WFK \n") #unperturbed wfk
               files.write("1 \n")
               files.write("0 \n")
               files.write("4 \n") # Gamma point
               files.write("8 \n") # Band nb 8
               files.write("0 \n")
               files.write("0 \n")
               files.write("2 \n") # real 3D data one column
               files.write("wfk \n")
               files.write("0 \n")
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   Mcarbone = 21894.16693
260
261   if mode[i][2:3] == "-" or mode[i][3:4]== "-":
262     emode = (1/(4*omega[i]*Mcarbone))*emodetmp*(27.211383*1000) #(1/2 mode)
263     print ("Mode :"+mode[i]+" "+str(emode))
264
265   else:
```

```
    emode = 0.0

  Sum_mode += emode

# ----------------------------------------
# Compute the total NDDW ZPM
# ----------------------------------------

ZPM=Sum_mode

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

```
# Find the eigenvector from the dynamical matrice

import numpy as np

def read_dynamical_matrix(datafile):
       with open(datafile,'r') as f:
         raw = list()
         for line in f:
           if line[0]=='#': continue
           parts=line.split()
           if not parts: continue
           val=float(parts[4])
           if np.abs(val) < 0.000000001:
             val=0.0
           raw.append(val)
         dynmat = np.array(raw)
         dynmat = dynmat.reshape(6,6)
       return dynmat

L_file = '/home/Samuel/Dropbox/WorkDiam/DerivEig_L/Dyanmical_mat.dat'

dynmatL=read_dynamical_matrix(L_file)

[eigvalL,eigvectL]=np.linalg.eig(dynmatL)

eigvectL=np.transpose(eigvectL)
Mc=21894.16693

print "eigenvalues at L :"
for n in eigvalL:
  a = np.sqrt(n/Mc)
  print a

print "eigenvectors at L :"
for n in eigvectL: print n

# Create the xcart needed for abinit input file

disp1 = 0.05
disp2 = 0.05
rprim = np.array([6.70346805,6.70346805,6.70346805])

print "# Equilibrium position"
print " xcart1 0.0 0.0  0.0                    # 0.0 0.0 0.0 "
print "        1.67586701 1.67586701 1.67586701 # 0.25 0.25 0.25"
print " "
print "# LA mode (+h)"

# Direction LA
# Atome K
i = 0
SBX1 = np.array([eigvectL[i][0],eigvectL[i][1],eigvectL[i][2]])*eigvectL[i][0]*d
isp1
SBY1 = np.array([eigvectL[i][0],eigvectL[i][1],eigvectL[i][2]])*eigvectL[i][1]*d
isp1
SBZ1 = np.array([eigvectL[i][0],eigvectL[i][1],eigvectL[i][2]])*eigvectL[i][2]*d
isp1
SBX2 = np.array([eigvectL[i][3],eigvectL[i][4],eigvectL[i][5]])*eigvectL[i][3]*d
isp1
SBY2 = np.array([eigvectL[i][3],eigvectL[i][4],eigvectL[i][5]])*eigvectL[i][4]*d
isp1
SBZ2 = np.array([eigvectL[i][3],eigvectL[i][4],eigvectL[i][5]])*eigvectL[i][5]*d
```

```
isp1

# Calculs translation
i = 0
TrX  = np.array([1,0,0])*disp2
TrY  = np.array([0,1,0])*disp2
TrZ  = np.array([0,0,1])*disp2

# --------------------------------
# Calcul du terme d2Vhxc/dSBdTB
# Atomes K
# --------------------------------

#VHX(x,y)
print " xcart2 %.10f %.10f %.10f "%(rprim[0]*0.0+(SBX1[0]+TrX[0]), rprim[1]*
0.0+(SBX1[1]+TrX[1]),rprim[2]*0.0+(SBX1[2]+TrX[2]))
print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(SBX2[0]+TrX[0]), rprim[1]
*0.25+(SBX2[1]+TrX[1]),rprim[2]*0.25+(SBX2[2]+TrX[2]))
print ""

#VHXC(-x,y)
print " xcart3 %.10f %.10f %.10f "%(rprim[0]*0.0+(-SBX1[0]+TrX[0]), rprim[1]
*0.0+(-SBX1[1]+TrX[1]),rprim[2]*0.0+(-SBX1[2]+TrX[2]))
print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(-SBX2[0]+TrX[0]), rprim[1
]*0.25+(-SBX2[1]+TrX[1]),rprim[2]*0.25+(-SBX2[2]+TrX[2]))
print ""

#VHXC(x,-y)
print " xcart4 %.10f %.10f %.10f "%(rprim[0]*0.0+(SBX1[0]-TrX[0]), rprim[1]*
0.0+(SBX1[1]-TrX[1]),rprim[2]*0.0+(SBX1[2]-TrX[2]))
print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(SBX2[0]-TrX[0]), rprim[1]
*0.25+(SBX2[1]-TrX[1]),rprim[2]*0.25+(SBX2[2]-TrX[2]))
print ""

#VHXC(-x,-y)
print " xcart5 %.10f %.10f %.10f "%(rprim[0]*0.0+(-SBX1[0]-TrX[0]), rprim[1]
*0.0+(-SBX1[1]-TrX[1]),rprim[2]*0.0+(-SBX1[2]-TrX[2]))
print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(-SBX2[0]-TrX[0]), rprim[1
]*0.25+(-SBX2[1]-TrX[1]),rprim[2]*0.25+(-SBX2[2]-TrX[2]))
print ""

#VHY(x,y)
print " xcart6 %.10f %.10f %.10f "%(rprim[0]*0.0+(SBY1[0]+TrY[0]), rprim[1]*
0.0+(SBY1[1]+TrY[1]),rprim[2]*0.0+(SBY1[2]+TrY[2]))
print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(SBY2[0]+TrY[0]), rprim[1]
*0.25+(SBY2[1]+TrY[1]),rprim[2]*0.25+(SBY2[2]+TrY[2]))
print ""

#VHYC(-x,y)
print " xcart7 %.10f %.10f %.10f "%(rprim[0]*0.0+(-SBY1[0]+TrY[0]), rprim[1]
*0.0+(-SBY1[1]+TrY[1]),rprim[2]*0.0+(-SBY1[2]+TrY[2]))
print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(-SBY2[0]+TrY[0]), rprim[1
]*0.25+(-SBY2[1]+TrY[1]),rprim[2]*0.25+(-SBY2[2]+TrY[2]))
print ""

#VHYC(x,-y)
print " xcart8 %.10f %.10f %.10f "%(rprim[0]*0.0+(SBY1[0]-TrY[0]), rprim[1]*
0.0+(SBY1[1]-TrY[1]),rprim[2]*0.0+(SBY1[2]-TrY[2]))
print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(SBY2[0]-TrY[0]), rprim[1]
*0.25+(SBY2[1]-TrY[1]),rprim[2]*0.25+(SBY2[2]-TrY[2]))
print ""

#VHYC(-x,-y)
print " xcart9 %.10f %.10f %.10f "%(rprim[0]*0.0+(-SBY1[0]-TrY[0]), rprim[1]
*0.0+(-SBY1[1]-TrY[1]),rprim[2]*0.0+(-SBY1[2]-TrY[2]))
print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(-SBY2[0]-TrY[0]), rprim[1
]*0.25+(-SBY2[1]-TrY[1]),rprim[2]*0.25+(-SBY2[2]-TrY[2]))
print ""

#VHZ(x,y)
print " xcart10 %.10f %.10f %.10f "%(rprim[0]*0.0+(SBZ1[0]+TrZ[0]), rprim[1]
*0.0+(SBZ1[1]+TrZ[1]),rprim[2]*0.0+(SBZ1[2]+TrZ[2]))
print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(SBZ2[0]+TrZ[0]), rprim[1]
*0.25+(SBZ2[1]+TrZ[1]),rprim[2]*0.25+(SBZ2[2]+TrZ[2]))
print ""

#VHZC(-x,y)
print " xcart11 %.10f %.10f %.10f "%(rprim[0]*0.0+(-SBZ1[0]+TrZ[0]), rprim[1
]*0.0+(-SBZ1[1]+TrZ[1]),rprim[2]*0.0+(-SBZ1[2]+TrZ[2]))
print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(-SBZ2[0]+TrZ[0]), rprim[1
]*0.25+(-SBZ2[1]+TrZ[1]),rprim[2]*0.25+(-SBZ2[2]+TrZ[2]))
print ""

#VHZC(x,-y)
```

```
print " xcart12 %.10f %.10f %.10f "%(rprim[0]*0.0+(SBZ1[0]-TrZ[0]), rprim[1]
*0.0+(SBZ1[1]-TrZ[1]),rprim[2]*0.0+(SBZ1[2]-TrZ[2]))
print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(SBZ2[0]-TrZ[0]), rprim[1]
*0.25+(SBZ2[1]-TrZ[1]),rprim[2]*0.25+(SBZ2[2]-TrZ[2]))
print ""

#VHZC(-x,-y)
print " xcart13 %.10f %.10f %.10f "%(rprim[0]*0.0+(-SBZ1[0]-TrZ[0]), rprim[1
]*0.0+(-SBZ1[1]-TrZ[1]),rprim[2]*0.0+(-SBZ1[2]-TrZ[2]))
print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(-SBZ2[0]-TrZ[0]), rprim[1
]*0.25+(-SBZ2[1]-TrZ[1]),rprim[2]*0.25+(-SBZ2[2]-TrZ[2]))
print ""

#Calculs de Delta lambda1, Delta lambda2
h=np.sqrt(disp1**2+disp2**2)
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

```
import math
import os
import numpy as np

# Transform binary VHXC file into text file

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS1_VHXC \n")
                files.write("1 \n")
                files.write("5 \n") #3D formatted data (output the bare 3D data - one column)
                files.write("EQ \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS2_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("LA+ \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS3_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("LA- \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS4_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("LO+ \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS5_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("LO- \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS6_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("TA1+ \n")
```

```
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS7_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("TA1- \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS8_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("TA2+ \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS9_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("TA2- \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS10_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("TO1+ \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS11_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("TO1- \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS12_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("TO2+ \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

os.system("rm cut3d.files")
with open("cut3d.files","a") as files:
                files.write("outputo_DS13_VHXC \n")
                files.write("1 \n")
                files.write("5 \n")
                files.write("TO2- \n")
                files.write("0 \n")
os.system("cut3d < cut3d.files ")

# ------------------------------------------------------------------------
# Make centered finite difference of order 2 of Vhxc for each mode
# d2Vhxc/dR1dR2=(2*Vhxc(mode)-2*Vhxc(eq))/(h^2)
# ------------------------------------------------------------------------

mode = np.array(["EQ","LA+","LA-","LO+","LO-","TA1+","TA1-","TA2+","TA2-",
"TO1+","TO1-","TO2+","TO2-"])

# The phonon frequency are eigenvalue of the Dynamical eigenvalue problem.

omega = np.array([0.0,0.00480084,0.00480084,0.005785943,0.005785943,0.002453274,
0.002453274,0.002453274,0.002453274,0.005671174,0.005671174,0.005671174,0.005671174])

# ------------------------------------------------------------------------
# Prepare the integration : de2/dR1dR2=int d2Vhxc/dR1dR2 psi*psi(unperturbed)
# ------------------------------------------------------------------------

os.system("rm wfk.files")
```

```
with open("wfk.files","a") as files:
                files.write("outputo_DS1_WFK \n") #unperturbed wfk
                files.write("1 \n")
                files.write("0 \n")
                files.write("4 \n") # Gamma point
                files.write("6 \n") # Band nb 6
                files.write("0 \n")
                files.write("0 \n")
                files.write("2 \n") # real 3D data one column
                files.write("wfk \n")
                files.write("0 \n")
os.system("cut3d < wfk.files ")

os.system("rm wfk.files")
with open("wfk.files","a") as files:
                files.write("outputo_DS1_WFK \n") #unperturbed wfk
                files.write("1 \n")
                files.write("0 \n")
                files.write("4 \n") # Gamma point
                files.write("7 \n") # Band nb 7
                files.write("0 \n")
                files.write("0 \n")
                files.write("2 \n") # real 3D data one column
                files.write("wfk \n")
                files.write("0 \n")
os.system("cut3d < wfk.files ")

os.system("rm wfk.files")
with open("wfk.files","a") as files:
                files.write("outputo_DS1_WFK \n") #unperturbed wfk
                files.write("1 \n")
                files.write("0 \n")
                files.write("4 \n") # Gamma point
                files.write("8 \n") # Band nb 8
                files.write("0 \n")
                files.write("0 \n")
                files.write("2 \n") # real 3D data one column
                files.write("wfk \n")
                files.write("0 \n")
os.system("cut3d < wfk.files ")

# ----------------------------------------
# Perform the integration
# ----------------------------------------

Sum_mode = 0.0
modetmp = 0.0

for i in range(len(mode)):
  # band nb 6
  fichier1 = str(mode[i])
  fichier2 = "wfk_k4_b6_s1"
  reader1 = open(fichier1,"r")
  reader2 = open(fichier2,"r")
  X1 = reader1.readlines()
  X2 = reader2.readlines()
  summa = 0.0
  for l in range(len(X2)):
    summa = summa + eval(X2[l].split()[0]) * eval(X2[l].split()[0])

  int = 0.0
  for ii in range(len(X1)):
    int = int + eval(X2[ii].split()[0]) * eval(X2[ii].split()[0]) * eval(X1[ii].split()[0])

  result1 = int/summa

  # band nb 7
  fichier1 = str(mode[i])
  fichier2 = "wfk_k4_b7_s1"
  reader1 = open(fichier1,"r")
  reader2 = open(fichier2,"r")
  X1 = reader1.readlines()
  X2 = reader2.readlines()
  summa = 0.0
  for l in range(len(X2)):
    summa = summa + eval(X2[l].split()[0]) * eval(X2[l].split()[0])

  int = 0.0
  for ii in range(len(X1)):
    int = int + eval(X2[ii].split()[0]) * eval(X2[ii].split()[0]) * eval(X1[ii].split()[0])

  result2 = int/summa

  # band nb 8
```

```
  fichier1 = str(mode[i])
  fichier2 = "wfk_k4_b8_s1"
  reader1 = open(fichier1,"r")
  reader2 = open(fichier2,"r")
  X1 = reader1.readlines()
  X2 = reader2.readlines()
  summa = 0.0
  for l in range(len(X2)):
    summa = summa + eval(X2[l].split()[0]) * eval(X2[l].split()[0])

  int = 0.0
  for ii in range(len(X1)):
    int = int + eval(X2[ii].split()[0]) * eval(X2[ii].split()[0]) * eval(X1[ii].split()[0])

  result3 = int/summa

  #Make the average of the three band
  average = (result1+result2+result3)/3

  if mode[i][0:2] == "EQ":
    equili = average

  #--------------------------------
  # Mode contribution
  #--------------------------------
  print "<Psi0|VHXC|Psi0> ", average, " for mode", mode[i]

  if mode[i][2:3] == "+" or mode[i][3:4]== "+" or mode[i][0:2] == "EQ":
    modetmp = average

  else:
    emodetmp = (average+modetmp-2*equili)/(0.01**2)

  Mcarbone = 21894.16693

  if mode[i][2:3] == "-" or mode[i][3:4]== "-":
    emode = (1/(4*omega[i]*Mcarbone))*emodetmp*(27.211383*1000) #(1/2 mode)
    print ("Mode :"+mode[i]+" "+str(emode))

  else:
    emode = 0.0

  Sum_mode += emode

# ----------------------------------------
# Compute the total NDDW ZPM
# ----------------------------------------

ZPM=Sum_mode

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

```
#   C in diamond structure; 2x2x2 FCC special point grid; low ecut.
ndtset 4

jdtset 11 12 21 22

elph2_imagden 0.1 eV

# The first point must be the Gamma one
qpt11 0.0 0.0 0.0
qpt12 0.0 0.0 0.0
qpt21 0.5 0.0 0.0
qpt22 0.5 0.0 0.0

# Ground state density
  getwfk11 0
  getden11 0
  nqpt11 0
  ieig2rf11 0
  rfphon11 0
  smdelta11 0
  iscf11 7

```

```
# RF at q=0
  getwfq12 0

# Non self-consistent calculation on a q point
  ieig2rf?1 0
  smdelta?1 0
  rfphon?1 0
  iscf?1 -2

# Computation at q
  getgam_eig2nkq?2 12
  getwfq?2 -1

# Common input variables, to be superceded in some cases
 nqpt 1
 ieig2rf 1
 bdeigrf 8
 smdelta 1
 getwfk 11 # Use GS wf from q=0
 getden 11 # Use density from q=0
 kptopt 3 # Need full k-point set for finite q response
 rfphon 1 # Do phonon response
 rfatpol 1 2 # Treat displacements of all atoms
 rfdir 1 1 1 # Do all directions (symmetry will be used)

#Size-dependent parameters
 acell 3*6.70346805 #relaxed param.
 rprim  0 .5 .5 .5 0 .5 .5 .5 0

 natom 2
 typat 1 1
 xred 3*0.00d0 3*0.25d0

 nband 8

 ngkpt 2 2 2
 nshiftk 4
 shiftk 0.0 0.0 0.0
        0.0 0.5 0.5
        0.5 0.0 0.5
        0.5 0.5 0.0

# Miscellaneous
 ntypat 1
 znucl 6
 diemac 6.0d0
 ecut 30
 enunit 2
 nstep 30
 nsym 1
 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

```
# ----------------------------------------------------------------------
# This script genereate cartesian coordinate for Abinit input files.
# Specifically for q = L = (0.5 0.0 0.0)
# Create inputs for finite diff. on eigenenergies and NDDW
# ----------------------------------------------------------------------

import numpy as np

# ----------------------------------------------------------------------
# Find the eigenvector and eigenenergies from the dynamical matrix
# computed using DFPT in ABINIT
# ----------------------------------------------------------------------

# Location of the Dynamical matrix file
L_file = '/home/Samuel/Dropbox/WorkDiam/DerivEig_L/Dyanmical_mat.dat'
# Carbon mass in a.u.
Mc=21894.16693

def read_dynamical_matrix(datafile):
  with open(datafile,'r') as f:
    raw = list()
    for line in f:
```

```
      if line[0]=='#': continue
      parts=line.split()
      if not parts: continue
      val=float(parts[4])
      if np.abs(val) < 0.000000001:
        val=0.0
      raw.append(val)
    dynmat = np.array(raw)
    dynmat = dynmat.reshape(6,6)
  return dynmat

dynmatL=read_dynamical_matrix(L_file)
[eigvalL,eigvectL]=np.linalg.eig(dynmatL)
eigvectL=np.transpose(eigvectL)

print "eigenvalues at L :"
for n in eigvalL:
  a = np.sqrt(n/Mc)
  print a

print "eigenvectors at L :"
for n in eigvectL: print n

# ----------------------------------------------------------------------
# Create Abinit input file for finite difference on eigen energies
# and the first part of NDDW. On a supercell 2x1x1
# ----------------------------------------------------------------------

# Value and number of the displacement (always 1/2 of the previous)
h = np.array([0.04,0.02,0.01,0.005,0.0025,0.00125,0.000625])

# Cell parameters
acell = np.array([13.4069361,6.70346805,6.70346805])
rprim = np.array([[0.0,0.5,0.5],[0.5,0.0,0.5],[0.5,0.5,0.0]])
cell = np.mat([rprim[0]*acell[0],rprim[1]*acell[1],rprim[2]*acell[2]])
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]])


# In the specific case of the L point, h = -h
# Indice that record the number of xcart in the input file
j = 1

# Equilibrium position
print " "
print "Contribution of the L point on the electronic eigenenergies at Gamma"
print "and the first part of the NDDW."
print "Using finite difference."
print " "
print " xcart%.0f  %.15f %.15f %.15f "%(j,(xred[0]*cell)[0,0],\
      (xred[0]*cell)[0,1],(xred[0]*cell)[0,2])
print "          %.15f %.15f %.15f "%((xred[1]*cell)[0,0],\
      (xred[1]*cell)[0,1],(xred[1]*cell)[0,2])
print "          %.15f %.15f %.15f "%((xred[2]*cell)[0,0],\
      (xred[2]*cell)[0,1],(xred[2]*cell)[0,2])
print "          %.15f %.15f %.15f "%((xred[3]*cell)[0,0],\
      (xred[3]*cell)[0,1],(xred[3]*cell)[0,2])
print " "
j += 1

for disp in h:
# LA mode
  i = 0
  print "Displacement: ", disp
  print " xcart%.0f  %.15f %.15f %.15f "%(j,(xred[0]*cell)[0,0]+\
        eigvectL[i][0]*disp,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,\
        (xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
  print "          %.15f %.15f %.15f "%((xred[1]*cell)[0,0]+\
        eigvectL[i][3]*disp,(xred[1]*cell)[0,1]+eigvectL[i][4]*disp,\
        (xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
  print "          %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
      eigvectL[i][0]*disp,(xred[2]*cell)[0,1]-eigvectL[i][1]*disp,\
      (xred[2]*cell)[0,2]-eigvectL[i][2]*disp)
  print "          %.15f %.15f %.15f "%((xred[3]*cell)[0,0]-\
      eigvectL[i][3]*disp,(xred[3]*cell)[0,1]-eigvectL[i][4]*disp,\
      (xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
  print " "
  j += 1
# LO mode
  i = 1
  print " xcart%.0f  %.15f %.15f %.15f "%(j,(xred[0]*cell)[0,0]+\
        eigvectL[i][0]*disp,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,\
        (xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
  print "          %.15f %.15f %.15f "%((xred[1]*cell)[0,0]+\
```

```
      eigvectL[i][3]*disp,(xred[1]*cell)[0,1]+eigvectL[i][4]*disp,\
      (xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
  print "          %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
      eigvectL[i][0]*disp,(xred[2]*cell)[0,1]-eigvectL[i][1]*disp,\
      (xred[2]*cell)[0,2]-eigvectL[i][2]*disp)
  print "          %.15f %.15f %.15f "%((xred[3]*cell)[0,0]-\
      eigvectL[i][3]*disp,(xred[3]*cell)[0,1]-eigvectL[i][4]*disp,\
      (xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
  print " "
  j += 1
# TA mode
  i = 2
  print " xcart%.0f  %.15f %.15f %.15f "%(j,(xred[0]*cell)[0,0]+\
      eigvectL[i][0]*disp,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,\
      (xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
  print "          %.15f %.15f %.15f "%((xred[1]*cell)[0,0]+\
      eigvectL[i][3]*disp,(xred[1]*cell)[0,1]+eigvectL[i][4]*disp,\
      (xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
  print "          %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
      eigvectL[i][0]*disp,(xred[2]*cell)[0,1]-eigvectL[i][1]*disp,\
      (xred[2]*cell)[0,2]-eigvectL[i][2]*disp)
  print "          %.15f %.15f %.15f "%((xred[3]*cell)[0,0]-\
      eigvectL[i][3]*disp,(xred[3]*cell)[0,1]-eigvectL[i][4]*disp,\
      (xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
  print " "
  j += 1
# TO mode
  i = 4
  print " xcart%.0f  %.15f %.15f %.15f "%(j,(xred[0]*cell)[0,0]+\
      eigvectL[i][0]*disp,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,\
      (xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
  print "          %.15f %.15f %.15f "%((xred[1]*cell)[0,0]+\
      eigvectL[i][3]*disp,(xred[1]*cell)[0,1]+eigvectL[i][4]*disp,\
      (xred[1]*cell)[0,2]+eigvectL[i][5]*disp)
  print "          %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
      eigvectL[i][0]*disp,(xred[2]*cell)[0,1]-eigvectL[i][1]*disp,\
      (xred[2]*cell)[0,2]-eigvectL[i][2]*disp)
  print "          %.15f %.15f %.15f "%((xred[3]*cell)[0,0]-\
      eigvectL[i][3]*disp,(xred[3]*cell)[0,1]-eigvectL[i][4]*disp,\
      (xred[3]*cell)[0,2]-eigvectL[i][5]*disp)
  print " "
  j += 1

# ----------------------------------------------------------------------
# Second part of the NDDW on a primitive cell
# We only compute 6xi=(3xi_1,3xi_1) as eigen displacement
# and then mulpliply by 2 because 6xi=(3xi_2,3xi_2) should be the same.
# ----------------------------------------------------------------------

acell = np.array([6.70346805,6.70346805,6.70346805])
cell = np.mat([rprim[0]*acell[0],rprim[1]*acell[1],rprim[2]*acell[2]])
xred = np.mat([[0.0,0.0,0.0],[0.25,0.25,0.25]])

print " "
print "Contribution of the L point on the electronic eigenenergies at Gamma"
print "and the second part of the NDDW."
print "Using finite difference."
print " "

j = 1
# Equilibrium position
print " xcart%.0f  %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0],\
      (xred[0]*cell)[0,1],(xred[0]*cell)[0,2])
print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0],\
      (xred[1]*cell)[0,1],(xred[1]*cell)[0,2])
print " "
j += 1

for disp in h:
  print "Displacement: ", disp
# LA mode
  i = 0
  print " xcart%.0f  %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]\
      +eigvectL[i][0]*disp,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,\
      (xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
  print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]+\
      eigvectL[i][0]*disp,(xred[1]*cell)[0,1]+eigvectL[i][1]*disp,\
      (xred[1]*cell)[0,2]+eigvectL[i][2]*disp)
  print " "
  j += 1
# LO mode
  i = 1
  print " xcart%.0f  %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]\
      +eigvectL[i][0]*disp,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,\
```

```
        (xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
  print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]+\
      eigvectL[i][0]*disp,(xred[1]*cell)[0,1]+eigvectL[i][1]*disp,\
      (xred[1]*cell)[0,2]+eigvectL[i][2]*disp)
  print " "
  j += 1
# TA mode
  i = 2
  print " xcart%.0f  %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]\
       +eigvectL[i][0]*disp,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,\
       (xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
  print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]+\
      eigvectL[i][0]*disp,(xred[1]*cell)[0,1]+eigvectL[i][1]*disp,\
      (xred[1]*cell)[0,2]+eigvectL[i][2]*disp)
  print " "
  j += 1
# TO mode
  i = 4
  print " xcart%.0f  %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]\
       +eigvectL[i][0]*disp,(xred[0]*cell)[0,1]+eigvectL[i][1]*disp,\
       (xred[0]*cell)[0,2]+eigvectL[i][2]*disp)
  print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]+\
      eigvectL[i][0]*disp,(xred[1]*cell)[0,1]+eigvectL[i][1]*disp,\
      (xred[1]*cell)[0,2]+eigvectL[i][2]*disp)
  print " "
  j += 1
```

Code 12: Python scripts use to create ABINIT input files

.

5.10 Python scripts II

```
# ----------------------------------------------------------------------
# This python script is used to compute the contribution of the
# L point to the Gamma eigenenergies using finite difference.
# ----------------------------------------------------------------------
import os
import numpy as np

def read_eigenenergies(datafile):
  flag = False
  with open(datafile,'r') as f:
    for line in f:
      if flag:
        eigenenergies = map(float,line.split())
        break
      if "kpt= 0.0000  0.0000  0.0000" in line:
        flag = True
  return eigenenergies

def use_cut3d(datafile,out,gamma,band):
  if datafile[len(datafile)-4:len(datafile)] == "VHXC":
    os.system("rm cut3d.files")
    with open("cut3d.files","a") as files:
      files.write(str(datafile)+"\n")
      files.write("1 \n")
      files.write("5 \n") #3D formatted data
                          #(output the bare 3D data - one column)
      files.write(str(out)+"\n")
      files.write("0 \n")
    os.system("cut3d < cut3d.files ")
  if datafile[len(datafile)-3:len(datafile)] == "WFK":
    os.system("rm wfk.files")
    with open("wfk.files","a") as files:
      files.write(str(datafile)+"\n")
      files.write("1 \n")
      files.write("0 \n")
      files.write(str(gamma)+" \n") # Gamma point
      files.write(str(band)+" \n") # Band nb 6
      files.write("0 \n")
      files.write("0 \n")
      files.write("2 \n") # real 3D data one column
      files.write(str(out)+" \n")
      files.write("0 \n")
    os.system("cut3d < wfk.files ")

def integration(VHXC,wfk):
  psi_file=open(wfk,'r')
  v_file=open(VHXC,'r')
```

```
  S=0.0; N=0.0
  while True:
    psi_l=psi_file.readline()
    v_l=v_file.readline()
    if not psi_l or not v_l: break
    psi=float(psi_l)
    v=float(v_l)
    S+=psi*v*psi
    N+=psi*psi
  psi_file.close()
  v_file.close()
  S=S/N
  return S

# The base name for the supercell files are assumed to be named "supercell"
# and the primitive as assumed to be named "primitive"

# Import the eigenenergies of the equilibrium state in Ha
eigenenergiesEQ = read_eigenenergies("supercello_DS1_EIG")

# Value and number of the displacement
h = np.array([0.04,0.02,0.01,0.005,0.0025,0.00125,0.000625])

# Initialisation values
k = 0

# The second dimension is the number of h in geometric progression
# h,h/2,h/4,h/8 etc..
# The third index correspond to modes (here 4)
# The first index correspond to the Richardson extrapolation
D = np.zeros([len(h),len(h),4])

# Frequency (computed in DFPT using ABINIT)
omega = np.array([0.00480084,0.005785943,0.002453274,0.005671174])

# Carbon mass in a.u.
Mc = 21894.16693

while k < len(h):

# LA mode
  eigenenergiesLA = read_eigenenergies("supercello_DS"+str(4*k+2)+"_EIG")
# LO mode
  eigenenergiesLO = read_eigenenergies("supercello_DS"+str(4*k+3)+"_EIG")
# TA mode
  eigenenergiesTA = read_eigenenergies("supercello_DS"+str(4*k+4)+"_EIG")
# TO mode
  eigenenergiesTO = read_eigenenergies("supercello_DS"+str(4*k+5)+"_EIG")

# Second order derivative (meV)
# LA mode
  D[0,k,0] = 2*(eigenenergiesLA[5]+eigenenergiesLA[6]+eigenenergiesLA[7]\
           -eigenenergiesEQ[5]-eigenenergiesEQ[6]-eigenenergiesEQ[7])/\
          (3*(h[k]**2))*(1.0/(4*Mc*omega[0]))*27.211383*1000
# LO mode
  D[0,k,1] = 2*(eigenenergiesLO[5]+eigenenergiesLO[6]+eigenenergiesLO[7]\
           -eigenenergiesEQ[5]-eigenenergiesEQ[6]-eigenenergiesEQ[7])/\
          (3*(h[k]**2))*(1.0/(4*Mc*omega[1]))*27.211383*1000
# TA mode
  D[0,k,2] = 2*(eigenenergiesTA[5]+eigenenergiesTA[6]+eigenenergiesTA[7]\
           -eigenenergiesEQ[5]-eigenenergiesEQ[6]-eigenenergiesEQ[7])/\
          (3*(h[k]**2))*(1.0/(4*Mc*omega[2]))*27.211383*1000
# TO mode
  D[0,k,3] = 2*(eigenenergiesTO[5]+eigenenergiesTO[6]+eigenenergiesTO[7]\
           -eigenenergiesEQ[5]-eigenenergiesEQ[6]-eigenenergiesEQ[7])/\
          (3*(h[k]**2))*(1.0/(4*Mc*omega[3]))*27.211383*1000

  k += 1

# ------------------------------------------------------------------------
# We will now use the Richardson Extrapolation to speed up the convergence
# ------------------------------------------------------------------------

l = 0
while l < len(h):
  k = 0
  while k < len(h):
    if (l-1) >= 0 and (k-1) >= 0:
#     LA mode
      D[l,k,0] = (D[l-1,k,0]-(D[l-1,k-1,0]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,0] == 0.0:
        D[l,k,0] = 0.0
#     LO mode
```

```
      D[l,k,1] = (D[l-1,k,1]-(D[l-1,k-1,1]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,1] == 0.0:
        D[l,k,1] = 0.0
#     TA mode
      D[l,k,2] = (D[l-1,k,2]-(D[l-1,k-1,2]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,2] == 0.0:
        D[l,k,2] = 0.0
#     TO mode
      D[l,k,3] = (D[l-1,k,3]-(D[l-1,k-1,3]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,3] == 0.0:
        D[l,k,3] = 0.0
    k += 1
  l += 1

Deigen = D

print "Richardson extrapolation of eigenenergies"
print Deigen

# ------------------------------------------------
# Compute the first part the the NDDW term
# ------------------------------------------------

# Second derivative using finite centred differences
# The second dimension is the number of h in geometric progression
# h,h/2,h/4,h/8 etc..
# The third index correspond to modes (here 4)
# The first index correspond to the Richardson extrapolation
D = np.zeros([len(h),len(h),4])

# Prepare equilibrium files
use_cut3d("supercello_DS1_VHXC","EQ",0,0)
use_cut3d("supercello_DS1_WFK","wfk",7,6)
use_cut3d("supercello_DS1_WFK","wfk",7,7)
use_cut3d("supercello_DS1_WFK","wfk",7,8)
averageEQ = (integration("EQ","wfk_k7_b6_s1")+\
       integration("EQ","wfk_k7_b7_s1")+integration("EQ","wfk_k7_b8_s1"))/3

k = 0
while k < len(h):

# Transform binary files into .txt files using cut3d
  use_cut3d("supercello_DS"+str(4*k+2)+"_VHXC","LA",0,0)
  use_cut3d("supercello_DS"+str(4*k+3)+"_VHXC","LO",0,0)
  use_cut3d("supercello_DS"+str(4*k+4)+"_VHXC","TA",0,0)
  use_cut3d("supercello_DS"+str(4*k+5)+"_VHXC","TO",0,0)

# Perform the integration int(psi_0*VHXC*psi_0)
  averageLA = (integration("LA","wfk_k7_b6_s1")+\
       integration("LA","wfk_k7_b7_s1")+integration("LA","wfk_k7_b8_s1"))/3
  averageLO = (integration("LO","wfk_k7_b6_s1")+\
       integration("LO","wfk_k7_b7_s1")+integration("LO","wfk_k7_b8_s1"))/3
  averageTA = (integration("TA","wfk_k7_b6_s1")+\
       integration("TA","wfk_k7_b7_s1")+integration("TA","wfk_k7_b8_s1"))/3
  averageTO = (integration("TO","wfk_k7_b6_s1")+\
       integration("TO","wfk_k7_b7_s1")+integration("TO","wfk_k7_b8_s1"))/3

# Second order derivative (meV)
# LA mode
  D[0,k,0] = 2*(averageLA-averageEQ)/\
         ((h[k]**2))*(1.0/(4*Mc*omega[0]))*27.211383*1000
# LO mode
  D[0,k,1] = 2*(averageLO-averageEQ)/\
         ((h[k]**2))*(1.0/(4*Mc*omega[1]))*27.211383*1000
# TA mode
  D[0,k,2] = 2*(averageTA-averageEQ)/\
         ((h[k]**2))*(1.0/(4*Mc*omega[2]))*27.211383*1000
# TO mode
  D[0,k,3] = 2*(averageTO-averageEQ)/\
         ((h[k]**2))*(1.0/(4*Mc*omega[3]))*27.211383*1000
  k += 1

# ------------------------------------------------------------------------
# We will now use the Richardson Extrapolation to speed up the convergence
# ------------------------------------------------------------------------

l = 0
while l < len(h):
  k = 0
  while k < len(h):
    if (l-1) >= 0 and (k-1) >= 0:
```

```
#     LA mode
      D[l,k,0] = (D[l-1,k,0]-(D[l-1,k-1,0]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,0] == 0.0:
        D[l,k,0] = 0.0
#     LO mode
      D[l,k,1] = (D[l-1,k,1]-(D[l-1,k-1,1]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,1] == 0.0:
        D[l,k,1] = 0.0
#     TA mode
      D[l,k,2] = (D[l-1,k,2]-(D[l-1,k-1,2]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,2] == 0.0:
        D[l,k,2] = 0.0
#     TO mode
      D[l,k,3] = (D[l-1,k,3]-(D[l-1,k-1,3]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,3] == 0.0:
        D[l,k,3] = 0.0
    k += 1
  l += 1

Dfirst = D
print "Richardson extrapolation of the first term of NDDW"
print Dfirst

# ---------------------------------------------------------------
# Compute the second part the the NDDW term on a primitive cell
# ---------------------------------------------------------------

D = np.zeros([len(h),len(h),4])

# Prepare equilibrium files
use_cut3d("primitiveo_DS1_VHXC","EQ",0,0)
use_cut3d("primitiveo_DS1_WFK","wfk",20,2)
use_cut3d("primitiveo_DS1_WFK","wfk",20,3)
use_cut3d("primitiveo_DS1_WFK","wfk",20,4)
averageEQ = (integration("EQ","wfk_k20_b2_s1")+\
       integration("EQ","wfk_k20_b3_s1")+integration("EQ","wfk_k20_b4_s1"))/3

k = 0
while k < len(h):

# Transform binary files into .txt files using cut3d
  use_cut3d("primitiveo_DS"+str(4*k+2)+"_VHXC","LA",0,0)
  use_cut3d("primitiveo_DS"+str(4*k+3)+"_VHXC","LO",0,0)
  use_cut3d("primitiveo_DS"+str(4*k+4)+"_VHXC","TA",0,0)
  use_cut3d("primitiveo_DS"+str(4*k+5)+"_VHXC","TO",0,0)

# Perform the integration int(psi_0*VHXC*psi_0)
  averageLA = (integration("LA","wfk_k20_b2_s1")+\
       integration("LA","wfk_k20_b3_s1")+integration("LA","wfk_k20_b4_s1"))/3
  averageLO = (integration("LO","wfk_k20_b2_s1")+\
       integration("LO","wfk_k20_b3_s1")+integration("LO","wfk_k20_b4_s1"))/3
  averageTA = (integration("TA","wfk_k20_b2_s1")+\
       integration("TA","wfk_k20_b3_s1")+integration("TA","wfk_k20_b4_s1"))/3
  averageTO = (integration("TO","wfk_k20_b2_s1")+\
       integration("TO","wfk_k20_b3_s1")+integration("TO","wfk_k20_b4_s1"))/3


# Second order derivative (meV)
# LA mode
  D[0,k,0] = 2*(averageLA-averageEQ)/\
         ((h[k]**2))*(1.0/(4*Mc*omega[0]))*27.211383*1000
# LO mode
  D[0,k,1] = 2*(averageLO-averageEQ)/\
         ((h[k]**2))*(1.0/(4*Mc*omega[1]))*27.211383*1000
# TA mode
  D[0,k,2] = 2*(averageTA-averageEQ)/\
         ((h[k]**2))*(1.0/(4*Mc*omega[2]))*27.211383*1000
# TO mode
  D[0,k,3] = 2*(averageTO-averageEQ)/\
         ((h[k]**2))*(1.0/(4*Mc*omega[3]))*27.211383*1000
  k += 1

# ---------------------------------------------------------------------
# We will now use the Richardson Extrapolation to speed up the convergence
# ---------------------------------------------------------------------

l = 0
while l < len(h):
  k = 0
  while k < len(h):
```

```
    if (l-1) >= 0 and (k-1) >= 0:
#     LA mode
      D[l,k,0] = (D[l-1,k,0]-(D[l-1,k-1,0]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,0] == 0.0:
        D[l,k,0] = 0.0
#     LO mode
      D[l,k,1] = (D[l-1,k,1]-(D[l-1,k-1,1]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,1] == 0.0:
        D[l,k,1] = 0.0
#     TA mode
      D[l,k,2] = (D[l-1,k,2]-(D[l-1,k-1,2]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,2] == 0.0:
        D[l,k,2] = 0.0
#     TO mode
      D[l,k,3] = (D[l-1,k,3]-(D[l-1,k-1,3]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,3] == 0.0:
        D[l,k,3] = 0.0
    k += 1
  l += 1


print "Richardson extrapolation of eigenenergies"
print Deigen

print "Richardson extrapolation of the first term of NDDW"
print Dfirst

print "Richardson extrapolation of the second term of NDDW"
print D
```

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

.

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

```
# ----------------------------------------------------------------------
# This script genereate cartesian coordinate for Abinit input files.
# Specifically for q = L = (1/3 0.0 0.0)
# Create inputs for finite diff. on eigenenergies and NDDW
# ----------------------------------------------------------------------

import numpy as np

# ----------------------------------------------------------------------
# Find the eigenvector and eigenenergies from the dynamical matrix
# computed using DFPT in ABINIT
# ----------------------------------------------------------------------

# Location of the Dynamical matrix file
L_file = '/home/Samuel/Dropbox/WorkDiam/2L3_Global/Dyanmical_mat.dat'
# Carbon mass in a.u.
Mc=21894.16693

def read_dynamical_matrix(datafile):
  with open(datafile,'r') as f:
    raw = list()
    for line in f:
      if line[0]=='#': continue
      parts=line.split()
      if not parts: continue
      valR=float(parts[4]) #real
      valI=float(parts[5]) #imaginary
      if np.abs(valR) < 0.000000001:
        valR=0.0
      if np.abs(valI) < 0.000000001:
        valI=0.0
      raw.append(complex(valR,valI))
    dynmat = np.array(raw)
    dynmat = dynmat.reshape(6,6)
  return dynmat

dynmat=read_dynamical_matrix(L_file)
[eigval,eigvect]=np.linalg.eig(dynmat)
eigvect=np.transpose(eigvect)

```

```
print "eigenvalues at 2/3L :"
for n in eigval:
  a = np.sqrt(n/Mc)
  print a

print "eigenvectors at 2/3L :"
for n in eigvect: print n

# ----------------------------------------------------------------------
# Create Abinit input file for finite difference on eigen energies
# and the first part of NDDW. On a supercell 3x1x1
# ----------------------------------------------------------------------

# Value and number of the displacement (always 1/2 of the previous)
h = np.array([0.02,0.01,0.005,0.0025,0.00125])

# Cell parameters
acell = np.array([20.110404150000001,6.70346805,6.70346805])
rprim = np.array([[0.0,0.5,0.5],[0.5,0.0,0.5],[0.5,0.5,0.0]])
cell = np.mat([rprim[0]*acell[0],rprim[1]*acell[1],rprim[2]*acell[2]])
xred = np.mat([[0.0,0.0,0.0],[1.0/12,0.25,0.25],[1.0/3,0.0,0.0],\
[5.0/12,0.25,0.25],[2.0/3,0.0,0.0],[9.0/12,0.25,0.25]])


# In the specific case of the L point, h = -h
# Indice that record the number of xcart in the input file
j = 1

# Equilibrium position
print " "
print " Here we compute \sum_kk' (d^2e/dRkdRk')Re(Uk)Re(Uk') "
print "Contribution of the 2/3L point on the electronic eigenenergies at Gamma"
print "and the first part of the NDDW."
print "Using finite difference."
print " "
print " xcart%.0f  %.15f %.15f %.15f "%(j,(xred[0]*cell)[0,0],\
      (xred[0]*cell)[0,1],(xred[0]*cell)[0,2])
print "         %.15f %.15f %.15f "%((xred[1]*cell)[0,0],\
      (xred[1]*cell)[0,1],(xred[1]*cell)[0,2])
print "         %.15f %.15f %.15f "%((xred[2]*cell)[0,0],\
      (xred[2]*cell)[0,1],(xred[2]*cell)[0,2])
print "         %.15f %.15f %.15f "%((xred[3]*cell)[0,0],\
      (xred[3]*cell)[0,1],(xred[3]*cell)[0,2])
print "         %.15f %.15f %.15f "%((xred[4]*cell)[0,0],\
      (xred[4]*cell)[0,1],(xred[4]*cell)[0,2])
print "         %.15f %.15f %.15f "%((xred[5]*cell)[0,0],\
      (xred[5]*cell)[0,1],(xred[5]*cell)[0,2])
print " "
j += 1

# ----------------------------------------------------------------------
# We need to compute \sum_kk' (d^2e/dRkdRk')[Re(Uk)Re(Uk')+Im(Uk)Im(Uk')]
# Uk = \xi_k e^iqRk = (Re(\xi_k)+Im(\xi_k))(cosqR_k+isinqR_k)
# Re(Uk) = Re(\xi_k)cosqR_k - Im(\xi_k)sinqR_k
# Im(Uk) = Im(\xi_k)cosqR_k + Re(\xi_k)sinqR_k
# We need to make a frozen phonon and to compute the quadratic term
# ----------------------------------------------------------------------

# ------------------------------------------------------
# Here we compute \sum_kk' (d^2e/dRkdRk')Re(Uk)Re(Uk')
# ------------------------------------------------------

for disp in h:
# ----------------------------------------------------------------------
# We have the following phase factor: e^(q.R)=cos(2\pi/3)+isin(2\pi/3)
# So here for the real part we have:
# Atoms 1 and 2 = eq + 1*disp
# Atoms 3 and 4 = eq - 0.5*disp
# Atoms 5 and 6 = eq - 0.5*disp
# For the imaginary part we have:
# Atoms 1 and 2 = eq + 0
# Atoms 3 and 4 = eq + sqrt(3)/2*disp
# Atoms 5 and 6 = eq - sqrt(3)/2*disp
# ----------------------------------------------------------------------

# ---------
# LA mode
# ---------
  i = 0
# Atoms 1 and 2
  print " xcart%.0f  %.15f %.15f %.15f "%(j,(xred[0]*cell)[0,0]+\
        eigvect.real[i][0]*disp,(xred[0]*cell)[0,1]+eigvect.real[i][1]*disp,\
        (xred[0]*cell)[0,2]+eigvect.real[i][2]*disp)
  print "         %.15f %.15f %.15f "%((xred[1]*cell)[0,0]+\
```

```
      eigvect.real[i][3]*disp,(xred[1]*cell)[0,1]+eigvect.real[i][4]*disp,\
      (xred[1]*cell)[0,2]+eigvect.real[i][5]*disp)
# Atoms 3 and 4
  print "         %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
      0.5*eigvect.real[i][0]*disp - (np.sqrt(3)/2)*eigvect.imag[i][0]*disp,\
      (xred[2]*cell)[0,1] - 0.5*eigvect.real[i][1]*disp - \
      (np.sqrt(3)/2)*eigvect.imag[i][1]*disp, (xred[2]*cell)[0,2]-\
      0.5*eigvect.real[i][2]*disp - (np.sqrt(3)/2)*eigvect.imag[i][2]*disp)
  print "         %.15f %.15f %.15f "%((xred[3]*cell)[0,0]-\
      0.5*eigvect.real[i][3]*disp - (np.sqrt(3)/2)*eigvect.imag[i][3]*disp,\
      (xred[3]*cell)[0,1] - 0.5*eigvect.real[i][4]*disp - \
      (np.sqrt(3)/2)*eigvect.imag[i][4]*disp, (xred[3]*cell)[0,2]-\
      0.5*eigvect.real[i][5]*disp - (np.sqrt(3)/2)*eigvect.imag[i][5]*disp)
# Atoms 5 and 6
  print "         %.15f %.15f %.15f "%((xred[4]*cell)[0,0]-\
      0.5*eigvect.real[i][0]*disp + (np.sqrt(3)/2)*eigvect.imag[i][0]*disp,\
      (xred[4]*cell)[0,1] - 0.5*eigvect.real[i][1]*disp + \
      (np.sqrt(3)/2)*eigvect.imag[i][1]*disp, (xred[4]*cell)[0,2]-\
      0.5*eigvect.real[i][2]*disp + (np.sqrt(3)/2)*eigvect.imag[i][2]*disp)
  print "         %.15f %.15f %.15f "%((xred[5]*cell)[0,0]-\
      0.5*eigvect.real[i][3]*disp + (np.sqrt(3)/2)*eigvect.imag[i][3]*disp,\
      (xred[5]*cell)[0,1] - 0.5*eigvect.real[i][4]*disp + \
      (np.sqrt(3)/2)*eigvect.imag[i][4]*disp, (xred[5]*cell)[0,2]-\
      0.5*eigvect.real[i][5]*disp + (np.sqrt(3)/2)*eigvect.imag[i][5]*disp)
  print " "
  j += 1
# --------
# LO mode
# --------
  i = 1
# Atoms 1 and 2
  print " xcart%.0f  %.15f %.15f %.15f "%(j,(xred[0]*cell)[0,0]+\
      eigvect.real[i][0]*disp,(xred[0]*cell)[0,1]+eigvect.real[i][1]*disp,\
      (xred[0]*cell)[0,2]+eigvect.real[i][2]*disp)
  print "         %.15f %.15f %.15f "%((xred[1]*cell)[0,0]+\
      eigvect.real[i][3]*disp,(xred[1]*cell)[0,1]+eigvect.real[i][4]*disp,\
      (xred[1]*cell)[0,2]+eigvect.real[i][5]*disp)
# Atoms 3 and 4
  print "         %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
      0.5*eigvect.real[i][0]*disp - (np.sqrt(3)/2)*eigvect.imag[i][0]*disp,\
      (xred[2]*cell)[0,1] - 0.5*eigvect.real[i][1]*disp - \
      (np.sqrt(3)/2)*eigvect.imag[i][1]*disp, (xred[2]*cell)[0,2]-\
      0.5*eigvect.real[i][2]*disp - (np.sqrt(3)/2)*eigvect.imag[i][2]*disp)
  print "         %.15f %.15f %.15f "%((xred[3]*cell)[0,0]-\
      0.5*eigvect.real[i][3]*disp - (np.sqrt(3)/2)*eigvect.imag[i][3]*disp,\
      (xred[3]*cell)[0,1] - 0.5*eigvect.real[i][4]*disp - \
      (np.sqrt(3)/2)*eigvect.imag[i][4]*disp, (xred[3]*cell)[0,2]-\
      0.5*eigvect.real[i][5]*disp - (np.sqrt(3)/2)*eigvect.imag[i][5]*disp)
# Atoms 5 and 6
  print "         %.15f %.15f %.15f "%((xred[4]*cell)[0,0]-\
      0.5*eigvect.real[i][0]*disp + (np.sqrt(3)/2)*eigvect.imag[i][0]*disp,\
      (xred[4]*cell)[0,1] - 0.5*eigvect.real[i][1]*disp + \
      (np.sqrt(3)/2)*eigvect.imag[i][1]*disp, (xred[4]*cell)[0,2]-\
      0.5*eigvect.real[i][2]*disp + (np.sqrt(3)/2)*eigvect.imag[i][2]*disp)
  print "         %.15f %.15f %.15f "%((xred[5]*cell)[0,0]-\
      0.5*eigvect.real[i][3]*disp + (np.sqrt(3)/2)*eigvect.imag[i][3]*disp,\
      (xred[5]*cell)[0,1] - 0.5*eigvect.real[i][4]*disp + \
      (np.sqrt(3)/2)*eigvect.imag[i][4]*disp, (xred[5]*cell)[0,2]-\
      0.5*eigvect.real[i][5]*disp + (np.sqrt(3)/2)*eigvect.imag[i][5]*disp)
  print " "
  j += 1
# --------
# TA mode
# --------
  i = 2
# Atoms 1 and 2
  print " xcart%.0f  %.15f %.15f %.15f "%(j,(xred[0]*cell)[0,0]+\
      eigvect.real[i][0]*disp,(xred[0]*cell)[0,1]+eigvect.real[i][1]*disp,\
      (xred[0]*cell)[0,2]+eigvect.real[i][2]*disp)
  print "         %.15f %.15f %.15f "%((xred[1]*cell)[0,0]+\
      eigvect.real[i][3]*disp,(xred[1]*cell)[0,1]+eigvect.real[i][4]*disp,\
      (xred[1]*cell)[0,2]+eigvect.real[i][5]*disp)
# Atoms 3 and 4
  print "         %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
      0.5*eigvect.real[i][0]*disp - (np.sqrt(3)/2)*eigvect.imag[i][0]*disp,\
      (xred[2]*cell)[0,1] - 0.5*eigvect.real[i][1]*disp - \
      (np.sqrt(3)/2)*eigvect.imag[i][1]*disp, (xred[2]*cell)[0,2]-\
      0.5*eigvect.real[i][2]*disp - (np.sqrt(3)/2)*eigvect.imag[i][2]*disp)
  print "         %.15f %.15f %.15f "%((xred[3]*cell)[0,0]-\
      0.5*eigvect.real[i][3]*disp - (np.sqrt(3)/2)*eigvect.imag[i][3]*disp,\
      (xred[3]*cell)[0,1] - 0.5*eigvect.real[i][4]*disp - \
      (np.sqrt(3)/2)*eigvect.imag[i][4]*disp, (xred[3]*cell)[0,2]-\
      0.5*eigvect.real[i][5]*disp - (np.sqrt(3)/2)*eigvect.imag[i][5]*disp)
# Atoms 5 and 6
```

```
  print "          %.15f %.15f %.15f "%((xred[4]*cell)[0,0]-\
        0.5*eigvect.real[i][0]*disp + (np.sqrt(3)/2)*eigvect.imag[i][0]*disp,\
        (xred[4]*cell)[0,1] - 0.5*eigvect.real[i][1]*disp + \
        (np.sqrt(3)/2)*eigvect.imag[i][1]*disp, (xred[4]*cell)[0,2]-\
        0.5*eigvect.real[i][2]*disp + (np.sqrt(3)/2)*eigvect.imag[i][2]*disp)
  print "          %.15f %.15f %.15f "%((xred[5]*cell)[0,0]-\
        0.5*eigvect.real[i][3]*disp + (np.sqrt(3)/2)*eigvect.imag[i][3]*disp,\
        (xred[5]*cell)[0,1] - 0.5*eigvect.real[i][4]*disp + \
        (np.sqrt(3)/2)*eigvect.imag[i][4]*disp, (xred[5]*cell)[0,2]-\
        0.5*eigvect.real[i][5]*disp + (np.sqrt(3)/2)*eigvect.imag[i][5]*disp)
  print " "
  j += 1
# --------
# TO mode
# --------
  i = 4
# Atoms 1 and 2
  print " xcart%.0f  %.15f %.15f %.15f "%(j,(xred[0]*cell)[0,0]+\
        eigvect.real[i][0]*disp,(xred[0]*cell)[0,1]+eigvect.real[i][1]*disp,\
        (xred[0]*cell)[0,2]+eigvect.real[i][2]*disp)
  print "          %.15f %.15f %.15f "%((xred[1]*cell)[0,0]+\
        eigvect.real[i][3]*disp,(xred[1]*cell)[0,1]+eigvect.real[i][4]*disp,\
        (xred[1]*cell)[0,2]+eigvect.real[i][5]*disp)
# Atoms 3 and 4
  print "          %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
        0.5*eigvect.real[i][0]*disp - (np.sqrt(3)/2)*eigvect.imag[i][0]*disp,\
        (xred[2]*cell)[0,1] - 0.5*eigvect.real[i][1]*disp - \
        (np.sqrt(3)/2)*eigvect.imag[i][1]*disp, (xred[2]*cell)[0,2]-\
        0.5*eigvect.real[i][2]*disp - (np.sqrt(3)/2)*eigvect.imag[i][2]*disp)
  print "          %.15f %.15f %.15f "%((xred[3]*cell)[0,0]-\
        0.5*eigvect.real[i][3]*disp - (np.sqrt(3)/2)*eigvect.imag[i][3]*disp,\
        (xred[3]*cell)[0,1] - 0.5*eigvect.real[i][4]*disp - \
        (np.sqrt(3)/2)*eigvect.imag[i][4]*disp, (xred[3]*cell)[0,2]-\
        0.5*eigvect.real[i][5]*disp - (np.sqrt(3)/2)*eigvect.imag[i][5]*disp)
# Atoms 5 and 6
  print "          %.15f %.15f %.15f "%((xred[4]*cell)[0,0]-\
        0.5*eigvect.real[i][0]*disp + (np.sqrt(3)/2)*eigvect.imag[i][0]*disp,\
        (xred[4]*cell)[0,1] - 0.5*eigvect.real[i][1]*disp + \
        (np.sqrt(3)/2)*eigvect.imag[i][1]*disp, (xred[4]*cell)[0,2]-\
        0.5*eigvect.real[i][2]*disp + (np.sqrt(3)/2)*eigvect.imag[i][2]*disp)
  print "          %.15f %.15f %.15f "%((xred[5]*cell)[0,0]-\
        0.5*eigvect.real[i][3]*disp + (np.sqrt(3)/2)*eigvect.imag[i][3]*disp,\
        (xred[5]*cell)[0,1] - 0.5*eigvect.real[i][4]*disp + \
        (np.sqrt(3)/2)*eigvect.imag[i][4]*disp, (xred[5]*cell)[0,2]-\
        0.5*eigvect.real[i][5]*disp + (np.sqrt(3)/2)*eigvect.imag[i][5]*disp)
  print " "
  j += 1

# ----------------------------------------------------------------------
# Second part of the NDDW on a primitive cell
# We only compute 6xi=(3xi_1,3xi_1) as eigen displacement
# and then mulpliply by 2 because 6xi=(3xi_2,3xi_2) should be the same.
# ----------------------------------------------------------------------

acell = np.array([6.70346805,6.70346805,6.70346805])
cell = np.mat([rprim[0]*acell[0],rprim[1]*acell[1],rprim[2]*acell[2]])
xred = np.mat([[0.0,0.0,0.0],[0.25,0.25,0.25]])

print " "
print "Contribution of the L point on the electronic eigenenergies at Gamma"
print "and the second part of the NDDW."
print "Using finite difference."
print "Real eigenvector "
print " "

j = 1
# Equilibrium position
print " xcart%.0f  %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0],\
       (xred[0]*cell)[0,1],(xred[0]*cell)[0,2])
print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0],\
       (xred[1]*cell)[0,1],(xred[1]*cell)[0,2])
print " "
j += 1

for disp in h:
# --------
# LA mode
# --------
  i = 0
  print " xcart%.0f  %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]\
       +eigvect.real[i][0]*disp, (xred[0]*cell)[0,1]+\
        eigvect.real[i][1]*disp, (xred[0]*cell)[0,2]+\
        eigvect.real[i][2]*disp)
  print "          %.10f %.10f %.10f "%((xred[1]*cell)[0,0]\
```

```
       +eigvect.real[i][0]*disp, (xred[1]*cell)[0,1]+\
        eigvect.real[i][1]*disp, (xred[1]*cell)[0,2]+\
        eigvect.real[i][2]*disp)
  print " "
  j += 1
# --------
# LO mode
# --------
  i = 1
  print " xcart%.0f  %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]\
       +eigvect.real[i][0]*disp, (xred[0]*cell)[0,1]+\
        eigvect.real[i][1]*disp, (xred[0]*cell)[0,2]+\
        eigvect.real[i][2]*disp)
  print "         %.10f %.10f %.10f "%((xred[1]*cell)[0,0]\
       +eigvect.real[i][0]*disp, (xred[1]*cell)[0,1]+\
        eigvect.real[i][1]*disp, (xred[1]*cell)[0,2]+\
        eigvect.real[i][2]*disp)
  print " "
  j += 1
# --------
# TA mode
# --------
  i = 2
  print " xcart%.0f  %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]\
       +eigvect.real[i][0]*disp, (xred[0]*cell)[0,1]+\
        eigvect.real[i][1]*disp, (xred[0]*cell)[0,2]+\
        eigvect.real[i][2]*disp)
  print "         %.10f %.10f %.10f "%((xred[1]*cell)[0,0]\
       +eigvect.real[i][0]*disp, (xred[1]*cell)[0,1]+\
        eigvect.real[i][1]*disp, (xred[1]*cell)[0,2]+\
        eigvect.real[i][2]*disp)
  print " "
  j += 1
# --------
# TO mode
# --------
  i = 4
  print " xcart%.0f  %.10f %.10f %.10f "%(j,(xred[0]*cell)[0,0]\
       +eigvect.real[i][0]*disp, (xred[0]*cell)[0,1]+\
        eigvect.real[i][1]*disp, (xred[0]*cell)[0,2]+\
        eigvect.real[i][2]*disp)
  print "         %.10f %.10f %.10f "%((xred[1]*cell)[0,0]\
       +eigvect.real[i][0]*disp, (xred[1]*cell)[0,1]+\
        eigvect.real[i][1]*disp, (xred[1]*cell)[0,2]+\
        eigvect.real[i][2]*disp)
  print " "
  j += 1


print ''
print ' ----------------------------------------'
print ' IMAGINARY PART '
print '----------------------------------------'
print ''

# ---------------------------------------------------------
# Here we compute \sum_kk' (d^2e/dRkdRk')Im(Uk)Im(Uk')
# ---------------------------------------------------------

# Cell parameters
acell = np.array([20.110404150000001,6.70346805,6.70346805])
rprim = np.array([[0.0,0.5,0.5],[0.5,0.0,0.5],[0.5,0.5,0.0]])
cell = np.mat([rprim[0]*acell[0],rprim[1]*acell[1],rprim[2]*acell[2]])
xred = np.mat([[0.0,0.0,0.0],[1.0/12,0.25,0.25],[1.0/3,0.0,0.0],\
[5.0/12,0.25,0.25],[2.0/3,0.0,0.0],[9.0/12,0.25,0.25]])

# Indice that record the number of xcart in the input file
j = 1

# Equilibrium position
print " "
print "Here we compute \sum_kk' (d^2e/dRkdRk')Im(Uk)Im(Uk') "
print "Contribution of the 2/3L point on the electronic eigenenergies at Gamma"
print "and the first part of the NDDW."
print "Using finite difference."
print " "
print " xcart%.0f  %.15f %.15f %.15f "%(j,(xred[0]*cell)[0,0],\
      (xred[0]*cell)[0,1],(xred[0]*cell)[0,2])
print "         %.15f %.15f %.15f "%((xred[1]*cell)[0,0],\
      (xred[1]*cell)[0,1],(xred[1]*cell)[0,2])
print "         %.15f %.15f %.15f "%((xred[2]*cell)[0,0],\
      (xred[2]*cell)[0,1],(xred[2]*cell)[0,2])
print "         %.15f %.15f %.15f "%((xred[3]*cell)[0,0],\
      (xred[3]*cell)[0,1],(xred[3]*cell)[0,2])
```

```
print "         %.15f %.15f %.15f "%((xred[4]*cell)[0,0],\
      (xred[4]*cell)[0,1],(xred[4]*cell)[0,2])
print "         %.15f %.15f %.15f "%((xred[5]*cell)[0,0],\
      (xred[5]*cell)[0,1],(xred[5]*cell)[0,2])
print " "
j += 1

# ------------------------------------------------
#hh=0
#while hh < 6:
#  g =0
#  while g < 6:
#    if abs(eigvect.imag[hh][g]) < 0.0001:
#      eigvect.imag[hh][g] = 0.0
#    g += 1
#  hh += 1
#
#print eigvect
# ------------------------------------------------

for disp in h:
# ------------------------------------------------------------------------
# We have the following phase factor: e^(q.R)=cos(2\pi/3)+isin(2\pi/3)
# So here for the real part we have:
# Atoms 1 and 2 = eq + 1*disp
# Atoms 3 and 4 = eq - 0.5*disp
# Atoms 5 and 6 = eq - 0.5*disp
# For the imaginary part we have:
# Atoms 1 and 2 = eq + 0
# Atoms 3 and 4 = eq + sqrt(3)/2*disp
# Atoms 5 and 6 = eq - sqrt(3)/2*disp
# ------------------------------------------------------------------------

# --------
# LA mode
# --------
  i = 0
# Atoms 1 and 2
  print " xcart%.0f  %.15f %.15f %.15f "%(j,(xred[0]*cell)[0,0]+\
        eigvect.imag[i][0]*disp,(xred[0]*cell)[0,1]+eigvect.imag[i][1]*disp,\
        (xred[0]*cell)[0,2]+eigvect.imag[i][2]*disp)
  print "         %.15f %.15f %.15f "%((xred[1]*cell)[0,0]+\
        eigvect.imag[i][3]*disp,(xred[1]*cell)[0,1]+eigvect.imag[i][4]*disp,\
        (xred[1]*cell)[0,2]+eigvect.imag[i][5]*disp)
# Atoms 3 and 4
  print "         %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
        0.5*eigvect.imag[i][0]*disp + (np.sqrt(3)/2)*eigvect.real[i][0]*disp,\
        (xred[2]*cell)[0,1] - 0.5*eigvect.imag[i][1]*disp + \
        (np.sqrt(3)/2)*eigvect.real[i][1]*disp, (xred[2]*cell)[0,2]-\
        0.5*eigvect.imag[i][2]*disp + (np.sqrt(3)/2)*eigvect.real[i][2]*disp)
  print "         %.15f %.15f %.15f "%((xred[3]*cell)[0,0]-\
        0.5*eigvect.imag[i][3]*disp + (np.sqrt(3)/2)*eigvect.real[i][3]*disp,\
        (xred[3]*cell)[0,1] - 0.5*eigvect.imag[i][4]*disp + \
        (np.sqrt(3)/2)*eigvect.real[i][4]*disp, (xred[3]*cell)[0,2]-\
        0.5*eigvect.imag[i][5]*disp + (np.sqrt(3)/2)*eigvect.real[i][5]*disp)
# Atoms 5 and 6
  print "         %.15f %.15f %.15f "%((xred[4]*cell)[0,0]-\
        0.5*eigvect.imag[i][0]*disp - (np.sqrt(3)/2)*eigvect.real[i][0]*disp,\
        (xred[4]*cell)[0,1] - 0.5*eigvect.imag[i][1]*disp - \
        (np.sqrt(3)/2)*eigvect.real[i][1]*disp, (xred[4]*cell)[0,2]-\
        0.5*eigvect.imag[i][2]*disp - (np.sqrt(3)/2)*eigvect.real[i][2]*disp)
  print "         %.15f %.15f %.15f "%((xred[5]*cell)[0,0]-\
        0.5*eigvect.imag[i][3]*disp - (np.sqrt(3)/2)*eigvect.real[i][3]*disp,\
        (xred[5]*cell)[0,1] - 0.5*eigvect.imag[i][4]*disp - \
        (np.sqrt(3)/2)*eigvect.real[i][4]*disp, (xred[5]*cell)[0,2]-\
        0.5*eigvect.imag[i][5]*disp - (np.sqrt(3)/2)*eigvect.real[i][5]*disp)
  print " "
  j += 1
# LO mode
  i = 1
# Atoms 1 and 2
  print " xcart%.0f  %.15f %.15f %.15f "%(j,(xred[0]*cell)[0,0]+\
        eigvect.imag[i][0]*disp,(xred[0]*cell)[0,1]+eigvect.imag[i][1]*disp,\
        (xred[0]*cell)[0,2]+eigvect.imag[i][2]*disp)
  print "         %.15f %.15f %.15f "%((xred[1]*cell)[0,0]+\
        eigvect.imag[i][3]*disp,(xred[1]*cell)[0,1]+eigvect.imag[i][4]*disp,\
        (xred[1]*cell)[0,2]+eigvect.imag[i][5]*disp)
# Atoms 3 and 4
  print "         %.15f %.15f %.15f "%((xred[2]*cell)[0,0]-\
        0.5*eigvect.imag[i][0]*disp + (np.sqrt(3)/2)*eigvect.real[i][0]*disp,\
        (xred[2]*cell)[0,1] - 0.5*eigvect.imag[i][1]*disp + \
        (np.sqrt(3)/2)*eigvect.real[i][1]*disp, (xred[2]*cell)[0,2]-\
        0.5*eigvect.imag[i][2]*disp + (np.sqrt(3)/2)*eigvect.real[i][2]*disp)
  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%.0f  %.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 # TO mode
512   i = 4
513 # Atoms 1 and 2
514   print " xcart%.0f  %.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 mulpliply 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 # LO 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 # TO 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 " "
```

```
  j += 1
```

Code 14: Python scripts use to create ABINIT input files

.

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

```
# ------------------------------------------------------------------
# This python script is used to compute the contribution of the
# L point to the Gamma eigenenergies using finite difference.
# ------------------------------------------------------------------
import os
import numpy as np

def read_eigenenergies(datafile):
  flag1 = False
  flag2 = False
  with open(datafile,'r') as f:
    for line in f:
      if flag2:
        eigenenergies = np.append([eigenenergies],[map(float,line.split())])
        break
      if flag1:
        eigenenergies = map(float,line.split())
        flag2 = True
      if "kpt= 0.0000  0.0000  0.0000" in line:
        flag1 = True
  return eigenenergies

def read_totalenergies(datafile):
  etotal = [(0.0)]
  with open(datafile,'r') as f:
    for line in f:
      if "etotal" in line:
        etot = line.split()
        if etotal[0] == 0:
          etotal[0] = etot[1]
        else:
          etotal = np.append(etotal,etot[1])
  return etotal

a = read_totalenergies("supercellR.outD")
print a

def use_cut3d(datafile,out,gamma,band):
  if datafile[len(datafile)-4:len(datafile)] == "VHXC":
    os.system("rm cut3d.files")
    with open("cut3d.files","a") as files:
      files.write(str(datafile)+"\n")
      files.write("1 \n")
      files.write("5 \n") #3D formatted data
                          #(output the bare 3D data - one column)
      files.write(str(out)+"\n")
      files.write("0 \n")
    os.system("cut3d < cut3d.files ")
  if datafile[len(datafile)-3:len(datafile)] == "WFK":
    os.system("rm wfk.files")
    with open("wfk.files","a") as files:
      files.write(str(datafile)+"\n")
      files.write("1 \n")
      files.write("0 \n")
      files.write(str(gamma)+" \n") # Gamma point
      files.write(str(band)+" \n") # Band nb 6
      files.write("0 \n")
      files.write("0 \n")
      files.write("2 \n") # real 3D data one column
      files.write(str(out)+" \n")
      files.write("0 \n")
    os.system("cut3d < wfk.files ")

def integration(VHXC,wfk):
  psi_file=open(wfk,'r')
  v_file=open(VHXC,'r')
  S=0.0; N=0.0
  while True:
    psi_l=psi_file.readline()
    v_l=v_file.readline()
    if not psi_l or not v_l: break
    psi=float(psi_l)
```

```
    v=float(v_l)
    S+=psi*v*psi
    N+=psi*psi
  psi_file.close()
  v_file.close()
  S=S/N
  return S

# The base name for the supercell files are assumed to be named "supercell"
# and the primitive as assumed to be named "primitive"

# Output file name
outputfile = "supercellR.outD"

# Import the total of the equilibrium state in Ha (same for imag)
totalenergyEQ = float(read_totalenergies(outputfile)[0])/3

# Value and number of the displacement
h = np.array([0.02,0.01,0.005,0.0025,0.00125])

# ---------------------------------------------------
# Computation of the frequency using total energies
# ---------------------------------------------------

# Initialisation values
k = 0

# The second dimension is the number of h in geometric progression
# h,h/2,h/4,h/8 etc..
# The third index correspond to modes (here 4)
# The first index correspond to the Richardson extrapolation
D = np.zeros([len(h),len(h),4])

# Frequency (computed in DFPT using ABINIT)
omega = np.array([0.003915559,0.006088587,0.002163719,0.00589085])

# Carbon mass in a.u.
Mc = 21894.16693

while k < len(h):

# LA mode
  totalenergyRLA = float(read_totalenergies(outputfile)[4*k+1])/3
# LO mode
  totalenergyRLO = float(read_totalenergies(outputfile)[4*k+2])/3
# TA mode
  totalenergyRTA = float(read_totalenergies(outputfile)[4*k+3])/3
# TO mode
  totalenergyRTO = float(read_totalenergies(outputfile)[4*k+4])/3
# LA mode
  totalenergyILA = float(read_totalenergies(outputfile)[4*k+1])/3
# LO mode
  totalenergyILO = float(read_totalenergies(outputfile)[4*k+2])/3
# TA mode
  totalenergyITA = float(read_totalenergies(outputfile)[4*k+3])/3
# TO mode
  totalenergyITO = float(read_totalenergies(outputfile)[4*k+4])/3

# Second order derivative (meV)
# LA mode
  D[0,k,0] = np.sqrt(2*(totalenergyRLA - totalenergyEQ)/\
            (h[k]**2)*(1.0/(Mc))+\
             2*(totalenergyILA-totalenergyEQ)/\
            (h[k]**2)*(1.0/(Mc)))
# LO mode
  D[0,k,1] = np.sqrt(2*(totalenergyRLO-totalenergyEQ)/\
            (h[k]**2)*(1.0/(Mc))+\
             2*(totalenergyILO-totalenergyEQ)/\
            (h[k]**2)*(1.0/(Mc)))
# TA mode
  D[0,k,2] = np.sqrt(2*(totalenergyRTA-totalenergyEQ)/\
            (h[k]**2)*(1.0/(Mc))+\
             2*(totalenergyITA-totalenergyEQ)/\
            (h[k]**2)*(1.0/(Mc)))
# TO mode
  D[0,k,3] = np.sqrt(2*(totalenergyRTO-totalenergyEQ)/\
            (h[k]**2)*(1.0/(Mc))+\
             2*(totalenergyITO-totalenergyEQ)/\
            (h[k]**2)*(1.0/(Mc)))

  k += 1

# ---------------------------------------------------------------------
# We will now use the Richardson Extrapolation to speed up the convergence
```

```
# ------------------------------------------------------------------

l = 0
while l < len(h):
  k = 0
  while k < len(h):
    if (l-1) >= 0 and (k-1) >= 0:
#     LA mode
      D[l,k,0] = (D[l-1,k,0]-(D[l-1,k-1,0]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,0] == 0.0:
        D[l,k,0] = 0.0
#     LO mode
      D[l,k,1] = (D[l-1,k,1]-(D[l-1,k-1,1]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,1] == 0.0:
        D[l,k,1] = 0.0
#     TA mode
      D[l,k,2] = (D[l-1,k,2]-(D[l-1,k-1,2]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,2] == 0.0:
        D[l,k,2] = 0.0
#     TO mode
      D[l,k,3] = (D[l-1,k,3]-(D[l-1,k-1,3]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,3] == 0.0:
        D[l,k,3] = 0.0
    k += 1
  l += 1

Dtotal = D
print Dtotal

# ------------------------------------------------------------
# Computation of ZPM using finite difference on eigenenergies
# ------------------------------------------------------------

# Import the total of the equilibrium state in Ha (same for imag)
eigenenergiesEQ = read_eigenenergies("supercellRo_DS1_EIG")

# Initialisation values
k = 0

# The second dimension is the number of h in geometric progression
# h,h/2,h/4,h/8 etc..
# The third index correspond to modes (here 4)
# The first index correspond to the Richardson extrapolation
D = np.zeros([len(h),len(h),4])

# Frequency (computed in DFPT using ABINIT)
omega = np.array([0.003915559,0.006088587,0.002163719,0.00589085])

# Level: Bandes
# HOMO
#lvl1 = 9 ; lvl2 = 10 ; lvl3 = 11
# LUMO
lvl1 = 12 ; lvl2 = 13 ; lvl3 = 14

# Carbon mass in a.u.
Mc = 21894.16693

while k < len(h):

# LA mode
  eigenenergiesRLA = read_eigenenergies("supercellRo_DS"+str(4*k+2)+"_EIG")
# LO mode
  eigenenergiesRLO = read_eigenenergies("supercellRo_DS"+str(4*k+3)+"_EIG")
# TA mode
  eigenenergiesRTA = read_eigenenergies("supercellRo_DS"+str(4*k+4)+"_EIG")
# TO mode
  eigenenergiesRTO = read_eigenenergies("supercellRo_DS"+str(4*k+5)+"_EIG")
# LA mode
  eigenenergiesILA = read_eigenenergies("supercellIo_DS"+str(4*k+2)+"_EIG")
# LO mode
  eigenenergiesILO = read_eigenenergies("supercellIo_DS"+str(4*k+3)+"_EIG")
# TA mode
  eigenenergiesITA = read_eigenenergies("supercellIo_DS"+str(4*k+4)+"_EIG")
# TO mode
  eigenenergiesITO = read_eigenenergies("supercellIo_DS"+str(4*k+5)+"_EIG")


# Second order derivative (meV)
# LA mode
  D[0,k,0] = 2*(eigenenergiesRLA[lvl1]+eigenenergiesRLA[lvl2]+eigenenergiesRLA[lvl3]\
```

```
           -eigenenergiesEQ[lvl1]-eigenenergiesEQ[lvl2]-eigenenergiesEQ[lvl3])/\
           (3*(h[k]**2))*(1.0/(4*Mc*omega[0]))*27.211383*1000+\
           2*(eigenenergiesILA[lvl1]+eigenenergiesILA[lvl2]+eigenenergiesILA[lvl3]\
           -eigenenergiesEQ[lvl1]-eigenenergiesEQ[lvl2]-eigenenergiesEQ[lvl3])/\
           (3*(h[k]**2))*(1.0/(4*Mc*omega[0]))*27.211383*1000
# LO mode
  D[0,k,1] = 2*(eigenenergiesRLO[lvl1]+eigenenergiesRLO[lvl2]+eigenenergiesRLO[lvl3]\
           -eigenenergiesEQ[lvl1]-eigenenergiesEQ[lvl2]-eigenenergiesEQ[lvl3])/\
           (3*(h[k]**2))*(1.0/(4*Mc*omega[1]))*27.211383*1000+\
           2*(eigenenergiesILO[lvl1]+eigenenergiesILO[lvl2]+eigenenergiesILO[lvl3]\
           -eigenenergiesEQ[lvl1]-eigenenergiesEQ[lvl2]-eigenenergiesEQ[lvl3])/\
           (3*(h[k]**2))*(1.0/(4*Mc*omega[1]))*27.211383*1000
# TA mode
  D[0,k,2] = 2*(eigenenergiesRTA[lvl1]+eigenenergiesRTA[lvl2]+eigenenergiesRTA[lvl3]\
           -eigenenergiesEQ[lvl1]-eigenenergiesEQ[lvl2]-eigenenergiesEQ[lvl3])/\
           (3*(h[k]**2))*(1.0/(4*Mc*omega[2]))*27.211383*1000+\
           2*(eigenenergiesITA[lvl1]+eigenenergiesITA[lvl2]+eigenenergiesITA[lvl3]\
           -eigenenergiesEQ[lvl1]-eigenenergiesEQ[lvl2]-eigenenergiesEQ[lvl3])/\
           (3*(h[k]**2))*(1.0/(4*Mc*omega[2]))*27.211383*1000
# TO mode
  D[0,k,3] = 2*(eigenenergiesRTO[lvl1]+eigenenergiesRTO[lvl2]+eigenenergiesRTO[lvl3]\
           -eigenenergiesEQ[lvl1]-eigenenergiesEQ[lvl2]-eigenenergiesEQ[lvl3])/\
           (3*(h[k]**2))*(1.0/(4*Mc*omega[3]))*27.211383*1000+\
           2*(eigenenergiesITO[lvl1]+eigenenergiesITO[lvl2]+eigenenergiesITO[lvl3]\
           -eigenenergiesEQ[lvl1]-eigenenergiesEQ[lvl2]-eigenenergiesEQ[lvl3])/\
           (3*(h[k]**2))*(1.0/(4*Mc*omega[3]))*27.211383*1000
  k += 1

# ------------------------------------------------------------------------
# We will now use the Richardson Extrapolation to speed up the convergence
# ------------------------------------------------------------------------

l = 0
while l < len(h):
  k = 0
  while k < len(h):
    if (l-1) >= 0 and (k-1) >= 0:
#     LA mode
      D[l,k,0] = (D[l-1,k,0]-(D[l-1,k-1,0]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,0] == 0.0:
        D[l,k,0] = 0.0
#     LO mode
      D[l,k,1] = (D[l-1,k,1]-(D[l-1,k-1,1]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,1] == 0.0:
        D[l,k,1] = 0.0
#     TA mode
      D[l,k,2] = (D[l-1,k,2]-(D[l-1,k-1,2]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,2] == 0.0:
        D[l,k,2] = 0.0
#     TO mode
      D[l,k,3] = (D[l-1,k,3]-(D[l-1,k-1,3]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,3] == 0.0:
        D[l,k,3] = 0.0
    k += 1
  l += 1

Deigen = D

print "Richardson extrapolation of eigenenergies"
print Deigen

# ----------------------------------------------------
# Compute the first part the the NDDW term
# ----------------------------------------------------

# Second derivative using finite centred differences
# The second dimension is the number of h in geometric progression
# h,h/2,h/4,h/8 etc..
# The third index correspond to modes (here 4)
# The first index correspond to the Richardson extrapolation
D = np.zeros([len(h),len(h),4])

# Prepare equilibrium files
use_cut3d("supercellRo_DS1_VHXC","EQ",0,0)
use_cut3d("supercellRo_DS1_WFK","wfk",1,lvl1+1)
use_cut3d("supercellRo_DS1_WFK","wfk",1,lvl2+1)
use_cut3d("supercellRo_DS1_WFK","wfk",1,lvl3+1)
averageEQ = (integration("EQ","wfk_k1_b"+str(lvl1+1)+"_s1")+\
             integration("EQ","wfk_k1_b"+str(lvl2+1)+"_s1")+\
             integration("EQ","wfk_k1_b"+str(lvl3+1)+"_s1"))/3
```

```
325
326 k = 0
327 while k < len(h):
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[l,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[l,k,0] = 0.0
402 #     LO mode
403       D[l,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[l,k,1] = 0.0
407 #     TA mode
408       D[l,k,2] = (D[l-1,k,2]-(D[l-1,k-1,2]*(1.0/(2**(2*l)))))/\
```

```
                (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,2] == 0.0:
        D[l,k,2] = 0.0
#     TO mode
      D[l,k,3] = (D[l-1,k,3]-(D[l-1,k-1,3]*(1.0/(2**(2*l)))))/\
                (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,3] == 0.0:
        D[l,k,3] = 0.0
    k += 1
  l += 1

Dfirst = D
print "Richardson extrapolation of the first term of NDDW"
print Dfirst

# ------------------------------------------------------------
# Compute the second part the the NDDW term on a primitive cell
# ------------------------------------------------------------

D = np.zeros([len(h),len(h),4])

# Level: Bandes
# HOMO
#lvl1 = 2 ; lvl2 = 3 ; lvl3 = 4
# LUMO
lvl1 = 5 ; lvl2 = 6 ; lvl3 = 7


# Prepare equilibrium files
use_cut3d("primitiveRo_DS1_VHXC","EQ",0,0)
use_cut3d("primitiveRo_DS1_WFK","wfk",1,lvl1)
use_cut3d("primitiveRo_DS1_WFK","wfk",1,lvl2)
use_cut3d("primitiveRo_DS1_WFK","wfk",1,lvl3)
averageEQ = (integration("EQ","wfk_k1_b"+str(lvl1)+"_s1")+\
       integration("EQ","wfk_k1_b"+str(lvl2)+"_s1")+\
       integration("EQ","wfk_k1_b"+str(lvl3)+"_s1"))/3

k = 0
while k < len(h):

# Transform binary files into .txt files using cut3d
  use_cut3d("primitiveRo_DS"+str(4*k+2)+"_VHXC","LAR",0,0)
  use_cut3d("primitiveRo_DS"+str(4*k+3)+"_VHXC","LOR",0,0)
  use_cut3d("primitiveRo_DS"+str(4*k+4)+"_VHXC","TAR",0,0)
  use_cut3d("primitiveRo_DS"+str(4*k+5)+"_VHXC","TOR",0,0)
  use_cut3d("primitiveIo_DS"+str(4*k+2)+"_VHXC","LAI",0,0)
  use_cut3d("primitiveIo_DS"+str(4*k+3)+"_VHXC","LOI",0,0)
  use_cut3d("primitiveIo_DS"+str(4*k+4)+"_VHXC","TAI",0,0)
  use_cut3d("primitiveIo_DS"+str(4*k+5)+"_VHXC","TOI",0,0)

# Perform the integration int(psi_0*VHXC*psi_0)
  averageLAR = (integration("LAR","wfk_k1_b"+str(lvl1)+"_s1")+\
       integration("LAR","wfk_k1_b"+str(lvl2)+"_s1")+\
       integration("LAR","wfk_k1_b"+str(lvl3)+"_s1"))/3
  averageLOR = (integration("LOR","wfk_k1_b"+str(lvl1)+"_s1")+\
       integration("LOR","wfk_k1_b"+str(lvl2)+"_s1")+\
       integration("LOR","wfk_k1_b"+str(lvl3)+"_s1"))/3
  averageTAR = (integration("TAR","wfk_k1_b"+str(lvl1)+"_s1")+\
       integration("TAR","wfk_k1_b"+str(lvl2)+"_s1")+\
       integration("TAR","wfk_k1_b"+str(lvl3)+"_s1"))/3
  averageTOR = (integration("TOR","wfk_k1_b"+str(lvl1)+"_s1")+\
       integration("TOR","wfk_k1_b"+str(lvl2)+"_s1")+\
       integration("TOR","wfk_k1_b"+str(lvl3)+"_s1"))/3
  averageLAI = (integration("LAI","wfk_k1_b"+str(lvl1)+"_s1")+\
       integration("LAI","wfk_k1_b"+str(lvl2)+"_s1")+\
       integration("LAI","wfk_k1_b"+str(lvl3)+"_s1"))/3
  averageLOI = (integration("LOI","wfk_k1_b"+str(lvl1)+"_s1")+\
       integration("LOI","wfk_k1_b"+str(lvl2)+"_s1")+\
       integration("LOI","wfk_k1_b"+str(lvl3)+"_s1"))/3
  averageTAI = (integration("TAI","wfk_k1_b"+str(lvl1)+"_s1")+\
       integration("TAI","wfk_k1_b"+str(lvl2)+"_s1")+\
       integration("TAI","wfk_k1_b"+str(lvl3)+"_s1"))/3
  averageTOI = (integration("TOI","wfk_k1_b"+str(lvl1)+"_s1")+\
       integration("TOI","wfk_k1_b"+str(lvl2)+"_s1")+\
       integration("TOI","wfk_k1_b"+str(lvl3)+"_s1"))/3


# Second order derivative (meV)
# LA mode
  D[0,k,0] = 2*(averageLAR-averageEQ)/\
          (h[k]**2)*(1.0/(4*Mc*omega[0]))*27.211383*1000+\
           2*(averageLAI-averageEQ)/\
          (h[k]**2)*(1.0/(4*Mc*omega[0]))*27.211383*1000
# LO mode
```

```
  D[0,k,1] = 2*(averageLOR-averageEQ)/\
           (h[k]**2)*(1.0/(4*Mc*omega[1]))*27.211383*1000+\
            2*(averageLOI-averageEQ)/\
           (h[k]**2)*(1.0/(4*Mc*omega[0]))*27.211383*1000
# TA mode
  D[0,k,2] = 2*(averageTAR-averageEQ)/\
           (h[k]**2)*(1.0/(4*Mc*omega[2]))*27.211383*1000+\
            2*(averageTAI-averageEQ)/\
           (h[k]**2)*(1.0/(4*Mc*omega[0]))*27.211383*1000
# TO mode
  D[0,k,3] = 2*(averageTOR-averageEQ)/\
           (h[k]**2)*(1.0/(4*Mc*omega[3]))*27.211383*1000+\
            2*(averageTOI-averageEQ)/\
           (h[k]**2)*(1.0/(4*Mc*omega[0]))*27.211383*1000
  k += 1

# ------------------------------------------------------------------------
# We will now use the Richardson Extrapolation to speed up the convergence
# ------------------------------------------------------------------------

l = 0
while l < len(h):
  k = 0
  while k < len(h):
    if (l-1) >= 0 and (k-1) >= 0:
#     LA mode
      D[l,k,0] = (D[l-1,k,0]-(D[l-1,k-1,0]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,0] == 0.0:
        D[l,k,0] = 0.0
#     LO mode
      D[l,k,1] = (D[l-1,k,1]-(D[l-1,k-1,1]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,1] == 0.0:
        D[l,k,1] = 0.0
#     TA mode
      D[l,k,2] = (D[l-1,k,2]-(D[l-1,k-1,2]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,2] == 0.0:
        D[l,k,2] = 0.0
#     TO mode
      D[l,k,3] = (D[l-1,k,3]-(D[l-1,k-1,3]*(1.0/(2**(2*l)))))/\
                 (1-(1.0/(2**(2*l))))
      if D[l-1,k-1,3] == 0.0:
        D[l,k,3] = 0.0
    k += 1
  l += 1


print "Richardson extrapolation of eigenenergies"
print Deigen

print "Richardson extrapolation of the first term of NDDW"
print Dfirst

print "Richardson extrapolation of the second term of NDDW"
print D
```

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

.

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

```
# ------------------------------------------------------
# This script only create the input file for the second
# NDDW term using another approach based on Tb and Sb
# ------------------------------------------------------

# Find the eigenvector from the dynamical matrice

import numpy as np

# Location of the Dynamical matrix file
L_file = '/home/Samuel/Dropbox/WorkDiam/2L3_Global/Dyanmical_mat.dat'
# Carbon mass in a.u.
Mc=21894.16693

def read_dynamical_matrix(datafile):
  with open(datafile,'r') as f:
```

```
    raw = list()
    for line in f:
      if line[0]=='#': continue
      parts=line.split()
      if not parts: continue
      valR=float(parts[4]) #real
      valI=float(parts[5]) #imaginary
      if np.abs(valR) < 0.000000001:
        valR=0.0
      if np.abs(valI) < 0.000000001:
        valI=0.0
      raw.append(complex(valR,valI))
    dynmat = np.array(raw)
    dynmat = dynmat.reshape(6,6)
  return dynmat

dynmat=read_dynamical_matrix(L_file)
[eigval,eigvect]=np.linalg.eig(dynmat)
eigvect=np.transpose(eigvect)

print "eigenvalues at 2/3L :"
for n in eigval:
  a = np.sqrt(n/Mc)
  print a

print "eigenvectors at 2/3L :"
for n in eigvect: print n


# Create the xcart needed for abinit input file

h = np.array([0.005])

rprim = np.array([6.70346805,6.70346805,6.70346805])


print "# Equilibrium position"
print " xcart1 0.0 0.0  0.0                  # 0.0 0.0 0.0 "
print "        1.67586701 1.67586701 1.67586701 # 0.25 0.25 0.25"
print " "
print "# LA mode (+h)"
j = 2

for disp in h:
# -------
# LA mode
# -------
  i = 1
# Atome K
  SBX1 = np.array([eigvect.real[i][0],eigvect.real[i][1],\
         eigvect.real[i][2]])*eigvect.real[i][0]*disp
  SBY1 = np.array([eigvect.real[i][0],eigvect.real[i][1],\
         eigvect.real[i][2]])*eigvect.real[i][1]*disp
  SBZ1 = np.array([eigvect.real[i][0],eigvect.real[i][1],\
         eigvect.real[i][2]])*eigvect.real[i][2]*disp
  SBX2 = np.array([eigvect.real[i][3],eigvect.real[i][4],\
         eigvect.real[i][5]])*eigvect.real[i][3]*disp
  SBY2 = np.array([eigvect.real[i][3],eigvect.real[i][4],\
         eigvect.real[i][5]])*eigvect.real[i][4]*disp
  SBZ2 = np.array([eigvect.real[i][3],eigvect.real[i][4],\
         eigvect.real[i][5]])*eigvect.real[i][5]*disp

# Calculs translation
  TrX  = np.array([1,0,0])*disp
  TrY  = np.array([0,1,0])*disp
  TrZ  = np.array([0,0,1])*disp

# --------------------------------
# Calcul du terme d2Vhxc/dSBdTB
# Atomes K
# --------------------------------

# VHXC(x,y)
  print " xcart%.0f %.10f %.10f %.10f "%(j,rprim[0]*0.0+(SBX1[0]+TrX[0]),\
         rprim[1]*0.0+(SBX1[1]+TrX[1]),rprim[2]*0.0+(SBX1[2]+TrX[2]))
  print "        %.10f  %.10f  %.10f  "%(rprim[0]*0.25+(SBX2[0]+TrX[0]),\
         rprim[1]*0.25+(SBX2[1]+TrX[1]),rprim[2]*0.25+(SBX2[2]+TrX[2]))
  print ""
  j += 1

# VHXC(-x,y)
  print " xcart%.0f %.10f %.10f %.10f "%(j,rprim[0]*0.0+(-SBX1[0]+TrX[0]),\
        rprim[1]*0.0+(-SBX1[1]+TrX[1]),rprim[2]*0.0+(-SBX1[2]+TrX[2]))
  print "        %.10f  %.10f  %.10f  "%(rprim[0]*0.25+(-SBX2[0]+TrX[0]),\
```

```
        rprim[1]*0.25+(-SBX2[1]+TrX[1]),rprim[2]*0.25+(-SBX2[2]+TrX[2]))
  print " "
  j += 1

# VHXC(x,-y)
  print " xcart%.0f %.10f %.10f %.10f "%(j,rprim[0]*0.0+(SBX1[0]-TrX[0]),\
       rprim[1]*0.0+(SBX1[1]-TrX[1]),rprim[2]*0.0+(SBX1[2]-TrX[2]))
  print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(SBX2[0]-TrX[0]),\
       rprim[1]*0.25+(SBX2[1]-TrX[1]),rprim[2]*0.25+(SBX2[2]-TrX[2]))
  print ""
  j += 1

# VHXC(-x,-y)
  print " xcart%.0f %.10f %.10f %.10f "%(j,rprim[0]*0.0+(-SBX1[0]-TrX[0]),\
       rprim[1]*0.0+(-SBX1[1]-TrX[1]),rprim[2]*0.0+(-SBX1[2]-TrX[2]))
  print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(-SBX2[0]-TrX[0]),\
       rprim[1]*0.25+(-SBX2[1]-TrX[1]),rprim[2]*0.25+(-SBX2[2]-TrX[2]))
  print ""
  j += 1
# VHXC(x,y)
  print " xcart%.0f %.10f %.10f %.10f "%(j,rprim[0]*0.0+(SBY1[0]+TrY[0]),\
        rprim[1]*0.0+(SBY1[1]+TrY[1]),rprim[2]*0.0+(SBY1[2]+TrY[2]))
  print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(SBY2[0]+TrY[0]),\
        rprim[1]*0.25+(SBY2[1]+TrY[1]),rprim[2]*0.25+(SBY2[2]+TrY[2]))
  print ""
  j += 1

# VHXC(-x,y)
  print " xcart%.0f %.10f %.10f %.10f "%(j,rprim[0]*0.0+(-SBY1[0]+TrY[0]),\
       rprim[1]*0.0+(-SBY1[1]+TrY[1]),rprim[2]*0.0+(-SBY1[2]+TrY[2]))
  print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(-SBY2[0]+TrY[0]),\
        rprim[1]*0.25+(-SBY2[1]+TrY[1]),rprim[2]*0.25+(-SBY2[2]+TrY[2]))
  print " "
  j += 1

# VHXC(x,-y)
  print " xcart%.0f %.10f %.10f %.10f "%(j,rprim[0]*0.0+(SBY1[0]-TrY[0]),\
       rprim[1]*0.0+(SBY1[1]-TrY[1]),rprim[2]*0.0+(SBY1[2]-TrY[2]))
  print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(SBY2[0]-TrY[0]),\
       rprim[1]*0.25+(SBY2[1]-TrY[1]),rprim[2]*0.25+(SBY2[2]-TrY[2]))
  print ""
  j += 1

# VHXC(-x,-y)
  print " xcart%.0f %.10f %.10f %.10f "%(j,rprim[0]*0.0+(-SBY1[0]-TrY[0]),\
       rprim[1]*0.0+(-SBY1[1]-TrY[1]),rprim[2]*0.0+(-SBY1[2]-TrY[2]))
  print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(-SBY2[0]-TrY[0]),\
       rprim[1]*0.25+(-SBY2[1]-TrY[1]),rprim[2]*0.25+(-SBY2[2]-TrY[2]))
  print ""
  j += 1
# VHXC(x,y)
  print " xcart%.0f %.10f %.10f %.10f "%(j,rprim[0]*0.0+(SBZ1[0]+TrZ[0]),\
        rprim[1]*0.0+(SBZ1[1]+TrZ[1]),rprim[2]*0.0+(SBZ1[2]+TrZ[2]))
  print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(SBZ2[0]+TrZ[0]),\
        rprim[1]*0.25+(SBZ2[1]+TrZ[1]),rprim[2]*0.25+(SBZ2[2]+TrZ[2]))
  print ""
  j += 1

# VHXC(-x,y)
  print " xcart%.0f %.10f %.10f %.10f "%(j,rprim[0]*0.0+(-SBZ1[0]+TrZ[0]),\
       rprim[1]*0.0+(-SBZ1[1]+TrZ[1]),rprim[2]*0.0+(-SBZ1[2]+TrZ[2]))
  print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(-SBZ2[0]+TrZ[0]),\
        rprim[1]*0.25+(-SBZ2[1]+TrZ[1]),rprim[2]*0.25+(-SBZ2[2]+TrZ[2]))
  print " "
  j += 1

# VHXC(x,-y)
  print " xcart%.0f %.10f %.10f %.10f "%(j,rprim[0]*0.0+(SBZ1[0]-TrZ[0]),\
       rprim[1]*0.0+(SBZ1[1]-TrZ[1]),rprim[2]*0.0+(SBZ1[2]-TrZ[2]))
  print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(SBZ2[0]-TrZ[0]),\
       rprim[1]*0.25+(SBZ2[1]-TrZ[1]),rprim[2]*0.25+(SBZ2[2]-TrZ[2]))
  print ""
  j += 1

# VHXC(-x,-y)
  print " xcart%.0f %.10f %.10f %.10f "%(j,rprim[0]*0.0+(-SBZ1[0]-TrZ[0]),\
       rprim[1]*0.0+(-SBZ1[1]-TrZ[1]),rprim[2]*0.0+(-SBZ1[2]-TrZ[2]))
  print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(-SBZ2[0]-TrZ[0]),\
       rprim[1]*0.25+(-SBZ2[1]-TrZ[1]),rprim[2]*0.25+(-SBZ2[2]-TrZ[2]))
  print ""
  j += 1

# -------
# 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]))
```

```
  print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(-SBY2[0]-TrY[0]),\
        rprim[1]*0.25+(-SBY2[1]-TrY[1]),rprim[2]*0.25+(-SBY2[2]-TrY[2]))
  print ""
  j += 1
# VHXC(x,y)
  print " xcart%.0f %.10f %.10f %.10f "%(j,rprim[0]*0.0+(SBZ1[0]+TrZ[0]),\
         rprim[1]*0.0+(SBZ1[1]+TrZ[1]),rprim[2]*0.0+(SBZ1[2]+TrZ[2]))
  print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(SBZ2[0]+TrZ[0]),\
         rprim[1]*0.25+(SBZ2[1]+TrZ[1]),rprim[2]*0.25+(SBZ2[2]+TrZ[2]))
  print ""
  j += 1

# VHXC(-x,y)
  print " xcart%.0f %.10f %.10f %.10f "%(j,rprim[0]*0.0+(-SBZ1[0]+TrZ[0]),\
        rprim[1]*0.0+(-SBZ1[1]+TrZ[1]),rprim[2]*0.0+(-SBZ1[2]+TrZ[2]))
  print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(-SBZ2[0]+TrZ[0]),\
         rprim[1]*0.25+(-SBZ2[1]+TrZ[1]),rprim[2]*0.25+(-SBZ2[2]+TrZ[2]))
  print " "
  j += 1
# VHXC(x,-y)
  print " xcart%.0f %.10f %.10f %.10f "%(j,rprim[0]*0.0+(SBZ1[0]-TrZ[0]),\
        rprim[1]*0.0+(SBZ1[1]-TrZ[1]),rprim[2]*0.0+(SBZ1[2]-TrZ[2]))
  print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(SBZ2[0]-TrZ[0]),\
        rprim[1]*0.25+(SBZ2[1]-TrZ[1]),rprim[2]*0.25+(SBZ2[2]-TrZ[2]))
  print ""
  j += 1

# VHXC(-x,-y)
  print " xcart%.0f %.10f %.10f %.10f "%(j,rprim[0]*0.0+(-SBZ1[0]-TrZ[0]),\
        rprim[1]*0.0+(-SBZ1[1]-TrZ[1]),rprim[2]*0.0+(-SBZ1[2]-TrZ[2]))
  print "       %.10f %.10f %.10f "%(rprim[0]*0.25+(-SBZ2[0]-TrZ[0]),\
        rprim[1]*0.25+(-SBZ2[1]-TrZ[1]),rprim[2]*0.25+(-SBZ2[2]-TrZ[2]))
  print ""
  j += 1
```

Code 16: Python scripts use to create ABINIT input files

.

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

```
# ------------------------------------------------------------------
# This python script is used to compute the contribution of the
# L point to the Gamma eigenenergies using finite difference.
# ------------------------------------------------------------------
import os
import numpy as np

def read_eigenenergies(datafile):
  flag1 = False
  flag2 = False
  with open(datafile,'r') as f:
    for line in f:
      if flag2:
        eigenenergies = np.append([eigenenergies],[map(float,line.split())])
        break
      if flag1:
        eigenenergies = map(float,line.split())
        flag2 = True
      if "kpt= 0.0000 0.0000 0.0000" in line:
        flag1 = True
  return eigenenergies

def use_cut3d(datafile,out,gamma,band):
  if datafile[len(datafile)-4:len(datafile)] == "VHXC":
    os.system("rm cut3d.files")
    with open("cut3d.files","a") as files:
      files.write(str(datafile)+"\n")
      files.write("1 \n")
      files.write("5 \n") #3D formatted data
                          #(output the bare 3D data - one column)
      files.write(str(out)+"\n")
      files.write("0 \n")
    os.system("cut3d < cut3d.files ")
  if datafile[len(datafile)-3:len(datafile)] == "WFK":
    os.system("rm wfk.files")
    with open("wfk.files","a") as files:
      files.write(str(datafile)+"\n")
      files.write("1 \n")
      files.write("0 \n")
```

```
      files.write(str(gamma)+" \n") # Gamma point
      files.write(str(band)+" \n") # Band nb 6
      files.write("0 \n")
      files.write("0 \n")
      files.write("2 \n") # real 3D data one column
      files.write(str(out)+" \n")
      files.write("0 \n")
    os.system("cut3d < wfk.files ")

def integration(VHXC,wfk):
  psi_file=open(wfk,'r')
  v_file=open(VHXC,'r')
  S=0.0; N=0.0
  while True:
    psi_l=psi_file.readline()
    v_l=v_file.readline()
    if not psi_l or not v_l: break
    psi=float(psi_l)
    v=float(v_l)
    S+=psi*v*psi
    N+=psi*psi
  psi_file.close()
  v_file.close()
  S=S/N
  return S

# The base name for the supercell files are assumed to be named "supercell"
# and the primitive as assumed to be named "primitive"


# Value and number of the displacement
h = np.array([0.005])

# Initialisation values
k = 0

# Frequency (computed in DFPT using ABINIT)
omega = np.array([0.003915559,0.006088587,0.002163719,0.00589085])

# Carbon mass in a.u.
Mc = 21894.16693

# ------------------------------------------------------------
# Compute the second part the the NDDW term on a primitive cell
# ------------------------------------------------------------

# Level: Bandes
# HOMO
#lvl1 = 2 ; lvl2 = 3 ; lvl3 = 4
# LUMO
lvl1 = 5 ; lvl2 = 6 ; lvl3 = 7


# Prepare equilibrium files
use_cut3d("primitiveo_DS1_VHXC","EQ",0,0)
use_cut3d("primitiveo_DS1_WFK","wfk",1,lvl1)
use_cut3d("primitiveo_DS1_WFK","wfk",1,lvl2)
use_cut3d("primitiveo_DS1_WFK","wfk",1,lvl3)
averageEQ = (integration("EQ","wfk_k1_b"+str(lvl1)+"_s1")+\
       integration("EQ","wfk_k1_b"+str(lvl2)+"_s1")+\
       integration("EQ","wfk_k1_b"+str(lvl3)+"_s1"))/3

k = 0
eigen = [0.0,0.0]
while k < 2:

# Transform binary files into .txt files using cut3d
  use_cut3d("primitiveo_DS"+str(12*k+2)+"_VHXC","VX++",0,0)
  use_cut3d("primitiveo_DS"+str(12*k+3)+"_VHXC","VX-+",0,0)
  use_cut3d("primitiveo_DS"+str(12*k+4)+"_VHXC","VX+-",0,0)
  use_cut3d("primitiveo_DS"+str(12*k+5)+"_VHXC","VX--",0,0)
  use_cut3d("primitiveo_DS"+str(12*k+6)+"_VHXC","VY++",0,0)
  use_cut3d("primitiveo_DS"+str(12*k+7)+"_VHXC","VY-+",0,0)
  use_cut3d("primitiveo_DS"+str(12*k+8)+"_VHXC","VY+-",0,0)
  use_cut3d("primitiveo_DS"+str(12*k+9)+"_VHXC","VY--",0,0)
  use_cut3d("primitiveo_DS"+str(12*k+10)+"_VHXC","VZ++",0,0)
  use_cut3d("primitiveo_DS"+str(12*k+11)+"_VHXC","VZ-+",0,0)
  use_cut3d("primitiveo_DS"+str(12*k+12)+"_VHXC","VZ+-",0,0)
  use_cut3d("primitiveo_DS"+str(12*k+13)+"_VHXC","VZ--",0,0)

# Perform the integration int(psi_0*VHXC*psi_0)
  averageVX1 = (integration("VX++","wfk_k1_b"+str(lvl1)+"_s1")+\
       integration("VX++","wfk_k1_b"+str(lvl2)+"_s1")+\
       integration("VX++","wfk_k1_b"+str(lvl3)+"_s1"))/3
```

```
  averageVX2 = (integration("VX-+","wfk_k1_b"+str(lvl1)+"_s1")+\
        integration("VX-+","wfk_k1_b"+str(lvl2)+"_s1")+\
        integration("VX-+","wfk_k1_b"+str(lvl3)+"_s1"))/3
  averageVX3 = (integration("VX+-","wfk_k1_b"+str(lvl1)+"_s1")+\
        integration("VX+-","wfk_k1_b"+str(lvl2)+"_s1")+\
        integration("VX+-","wfk_k1_b"+str(lvl3)+"_s1"))/3
  averageVX4 = (integration("VX--","wfk_k1_b"+str(lvl1)+"_s1")+\
        integration("VX--","wfk_k1_b"+str(lvl2)+"_s1")+\
        integration("VX--","wfk_k1_b"+str(lvl3)+"_s1"))/3
  averageVY1 = (integration("VY++","wfk_k1_b"+str(lvl1)+"_s1")+\
        integration("VY++","wfk_k1_b"+str(lvl2)+"_s1")+\
        integration("VY++","wfk_k1_b"+str(lvl3)+"_s1"))/3
  averageVY2 = (integration("VY-+","wfk_k1_b"+str(lvl1)+"_s1")+\
        integration("VY-+","wfk_k1_b"+str(lvl2)+"_s1")+\
        integration("VY-+","wfk_k1_b"+str(lvl3)+"_s1"))/3
  averageVY3 = (integration("VY+-","wfk_k1_b"+str(lvl1)+"_s1")+\
        integration("VY+-","wfk_k1_b"+str(lvl2)+"_s1")+\
        integration("VY+-","wfk_k1_b"+str(lvl3)+"_s1"))/3
  averageVY4 = (integration("VY--","wfk_k1_b"+str(lvl1)+"_s1")+\
        integration("VY--","wfk_k1_b"+str(lvl2)+"_s1")+\
        integration("VY--","wfk_k1_b"+str(lvl3)+"_s1"))/3
  averageVZ1 = (integration("VZ++","wfk_k1_b"+str(lvl1)+"_s1")+\
        integration("VZ++","wfk_k1_b"+str(lvl2)+"_s1")+\
        integration("VZ++","wfk_k1_b"+str(lvl3)+"_s1"))/3
  averageVZ2 = (integration("VZ-+","wfk_k1_b"+str(lvl1)+"_s1")+\
        integration("VZ-+","wfk_k1_b"+str(lvl2)+"_s1")+\
        integration("VZ-+","wfk_k1_b"+str(lvl3)+"_s1"))/3
  averageVZ3 = (integration("VZ+-","wfk_k1_b"+str(lvl1)+"_s1")+\
        integration("VZ+-","wfk_k1_b"+str(lvl2)+"_s1")+\
        integration("VZ+-","wfk_k1_b"+str(lvl3)+"_s1"))/3
  averageVZ4 = (integration("VZ--","wfk_k1_b"+str(lvl1)+"_s1")+\
        integration("VZ--","wfk_k1_b"+str(lvl2)+"_s1")+\
        integration("VZ--","wfk_k1_b"+str(lvl3)+"_s1"))/3

# Second order derivative (meV)
# LA mode
  eigen[k] = (averageVX1-averageVX2-averageVX3+averageVX4+\
         averageVY1-averageVY2-averageVY3+averageVY4+
         averageVZ1-averageVZ2-averageVZ3+averageVZ4)/\
          (4*(h[0]**2))*(1.0/(4*Mc*omega[1]))*27.211383*1000

  k += 1

print eigen
```

Code 17: Python scriptsuse 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 densitÃ© figÃ© Ã un systeme qui est isolant dans un calcul de phonon entraine qu'on observe un changement de valeur propre des phonons qui donne le NDDW.

BUG remarquÃ© par Paul sur le TR sym.

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

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-symmetrie (64 points Q, grille de 4x4x4 pour les tests):

BUG remarquÃ© par Xavier dans la routine thmeig.F90. Le facteur de phase a Ã©tÃ© calculÃ© deux fois.

Il y a en effet deux faÃ§on de dÃ©finir la matrice dynamique (avec l la cellule unitÃ© considÃ©rÃ©e) (voir eq 5.56 these de Paul):

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

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

$$D^I_{\alpha\kappa,\beta\kappa'}(Q) = e^{iq(R_\kappa - R_{\kappa'})} D^{II}_{\alpha\kappa,\beta\kappa'}(Q) \tag{130}$$

Si on a la relation suivante:

$$\sum_{\beta\kappa'} D^I_{\alpha\kappa,\beta\kappa'}(Q)\xi_{\beta\kappa'}(q) = M_\kappa\omega^2\xi_{\alpha\kappa}(q) \tag{131}$$

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

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

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