Goal - To understand the Taylor expansion of $e^{-i\vec{q}\cdot d\vec{x}}$ in a spherical harmonics basis.

First → where does this Taylor expansion come from?

* → Since w^0 is singular, our method of calculating its pseudo-inverse is to take it to Fourier space, invert it, and the inverse transform to get g^0.

Step 1 → Find eigenvalues and eigenvectors

First we write the normal Fourier transform of w^0 as follows:-

$$\hat{w}_{ij}(\vec{q}) = \sum_{\vec{R}} w^0_{ij}(\vec{R})\, e^{i\vec{q}\cdot\vec{R}}$$

where i, j are the initial and final vacancy sites and $\vec{R}$ is the lattice vector that seperates them.

now, we can write the eigenvalue equation for $\hat{w}(\vec{q})$ as:-

$$\hat{w}(\vec{q})\, s^\alpha(\vec{q}) = \gamma^\alpha(\vec{q}) s^\alpha(\vec{q})$$ where γ^α is the α^{th} eigenvalue, s^α is the α^{th} eigenvector.

for a particular component of the eigenvectors, we can write:-

$$\sum_b \hat{w}_{ab}(\vec{q})\, s^\alpha_b(\vec{q}) = \gamma^\alpha(\vec{q})\, s^\alpha_a(\vec{q})$$

$$\Rightarrow \sum_b \sum_{\vec{R}} w^0_{ij}(\vec{R})\, e^{-i\vec{q}\cdot\vec{R}}\, s^\alpha_b(\vec{q}) = \gamma^\alpha(\vec{q})\, s^\alpha_a(\vec{q})$$

at $\vec{q} = 0$, we have :-

$$\sum_b \sum_{\vec{R}} w^0_{ij}(\vec{R})\, s^\alpha_b = \gamma^\alpha s^\alpha_a \quad - (1a) \leftarrow$$

$$\Rightarrow \frac{1}{N}\sum_{\vec{x},\vec{x}'} \sum_b w_{\vec{x}a,\vec{x}'b}\, s^\alpha_b = \gamma^\alpha s^\alpha_a \quad - (1b) \qquad \sum_{x'b} w_{\vec{x}a,\vec{x}'b}\, s^\alpha_b = \gamma^\alpha s^\alpha_a$$

Dividing by N, because the initial state is now not at the origin, but at other translationally equivalent sites.

⁂ The paper has equation (1b) in equation (36).
The code implements equation (1a).
→ Remember that in our jumpnetworks, the initial state is always at the origin.

Step 2 → creating the new basis for representing the Fourier space matrix.

Note – since $\omega_{ij}(\vec{k})$ is symmetric, so is $\hat{\omega}_{ij}(\vec{q}=0)$ and hence the eigenvectors s^α are all orthogonal.

Also, there will be one zero eigenvalue, which corresponds to the equilibrium distribution.

There are a total N_{sites} eigenvalue-vector pairs.

We write :-

$$\phi^{\alpha}_{\vec{q},\vec{x}a} = \frac{1}{\sqrt{N}} s^{\alpha}_{a} e^{i\vec{q}\cdot(\vec{x}+\vec{u}_a)}$$ as our new basis vectors.

→ Next – Refer to Notebook 2, as to how we can write the inverse transform in this new basis.

$$\phi^{\alpha}_{\vec{q}}(\vec{x}+u_a) = \frac{1}{\sqrt{N}} s^{\alpha}_{a} e^{i\vec{q}\cdot(\vec{x}+\vec{u}_a)}$$ where $\vec{q}$ are vectors in the first Brillouin zone, as allowed by the BVK periodic boundary conditions.

the indices $(\alpha, \vec{q})$ together determine the vectors, i.e.,

$\bar{\phi}^{\alpha_1}_{\vec{q}_1}$, $\bar{\phi}^{\alpha_2}_{\vec{q}_2}$ ... are the vectors,

and the positions $(\vec{x}+\vec{u}_a)$ determine their "components", so that, we can determine the inner product as :-

$$\langle \bar{\phi}^{\alpha_1}_{q_1} \cdot \bar{\phi}^{\alpha_2}_{q_2} \rangle = \sum_{xa} \phi^{\alpha_1,*}_{q_1}(\vec{x}+\vec{u}_a)\, \phi^{\alpha_2}_{q_2}(\vec{x}+\vec{u}_a)$$

Analogous to :- $\langle a\, b \rangle = \sum_j a_j b_j$

then, we have :-

$$\langle \bar{\phi}^{\alpha_1}_{q_1}, \bar{\phi}^{\alpha_2}_{q_2} \rangle = \frac{1}{N} \sum_{x\alpha} s^{\alpha_1 *}_{a} e^{-i\vec{q}_1(\vec{x}+\vec{u}_a)} s^{\alpha_2}_{a} e^{i\vec{q}_2\cdot(\vec{x}+\vec{u}_a)}$$

$$= \frac{1}{N} \sum_{\vec{x}a} s^{\alpha_1}_{a} s^{\alpha_2}_{a} e^{i\vec{x}\cdot(\vec{q}_2-\vec{q}_1)} e^{i\vec{u}_a\cdot(\vec{q}_2-\vec{q}_1)}$$

$$= \gamma \langle \bar{\phi}^{\alpha_1}_{q_1}, \bar{\phi}^{\alpha_2}_{q_2} \rangle$$

$$= \frac{1}{N} \sum_{a} e^{i\vec{u}_a(\vec{q}_2-\vec{q}_1)} s^{\alpha_1}_{a} s^{\alpha_2}_{a} \sum_{\vec{x}} e^{i\vec{x}\cdot(\vec{q}_2-\vec{q}_1)}$$

now,

$$\sum_{\vec{x}} e^{i\vec{x}\cdot(\vec{q}_2-\vec{q}_1)} = \sum_{\vec{x}} e^{i(\vec{x}+\vec{R})\cdot(\vec{q}_2-\vec{q}_1)}$$ (we're only shifting the origin, nothing else.)

$$= e^{i\vec{R}\cdot(\vec{q}_2-\vec{q}_1)} \sum_{\vec{x}} e^{i\vec{x}\cdot(\vec{q}_2-\vec{q}_1)}$$

$$\Rightarrow \sum_{\vec{x}} e^{i\vec{x}\cdot(\vec{q}_2-\vec{q}_1)} = e^{i\vec{R}\cdot(\vec{q}_2-\vec{q}_1)} \sum_{\vec{x}} e^{i\vec{x}\cdot(\vec{q}_2-\vec{q}_1)}$$

$$\Rightarrow e^{i\vec{R}\cdot(\vec{q}_2-\vec{q}_1)} = 1 \quad \text{or} \quad \sum_{x} e^{i\vec{x}\cdot(\vec{q}_2-\vec{q}_1)} = 0$$

which means either $\vec{q}_2$ and $\vec{q}_1$ must both be reciprocal lattice vectors, or $\vec{q}_2 = \vec{q}_1$

But since the reciprocal lattice vectors are in the 1st B.Z., they must satisfy $q_2 = q_1$, ie

$$\sum_{\vec{x}} e^{i\vec{x}\cdot(\vec{q}_2-\vec{q}_1)} = \delta(\vec{q}_1-\vec{q}_2)$$

$$\therefore \langle \bar{\phi}^{\alpha_1}_{\vec{q}_1} \bar{\phi}^{\alpha_2}_{\vec{q}_2} \rangle = \frac{1}{N} \sum_{a} e^{i\vec{u}_a\cdot(\vec{q}_2-\vec{q}_1)} s^{\alpha_1}_a s^{\alpha_2}_a \delta(\vec{q}_2-\vec{q}_1)$$

$$= \frac{1}{N}\delta(\vec{q}_2-\vec{q}_1) \sum_a s^{\alpha_1}_a s^{\alpha_2}_a = \frac{1}{N}\delta(\vec{q}_2-\vec{q}_1)\delta_{\alpha_1\alpha_2}$$

Since $\omega_{ij}(\vec{q}=0)$ is a symmetric matrix, the eigenvectors s^{α_1} s^{α_2} are orthogonal.

$$\therefore \langle \phi^{\alpha_1}_{q_1} \phi^{\alpha_2}_{q_2} \rangle = \delta_{\alpha_1\alpha_2}\delta(\vec{q}_2-\vec{q}_1)$$

with these basis vectors, we can write:

$$\omega^{\alpha\beta}(\vec{q}_1,\vec{q}_2) = \sum_{\substack{x_1 a \\ x_2 b}} \phi^{\beta *}_{q_2}(\vec{x}_2+\vec{u}_b)\, \omega^0_{\vec{x}_1 a,\vec{x}_2 b}\, \phi^{\alpha}_{q_1}(\vec{x}_1+\vec{u}_a)$$

$$\Rightarrow \omega^{\alpha\beta}(\vec{q}_1,\vec{q}_2) = \frac{1}{N} \sum_{\substack{x_1 a \\ x_2 b}} s^{\beta}_b e^{-i\vec{q}_2\cdot(\vec{x}_2+\vec{u}_b)} \omega^0_{\vec{x}_1 a,\vec{x}_2 b} s^{\alpha}_a e^{i\vec{q}_1\cdot(\vec{x}_1+\vec{u}_a)}$$

$$\Rightarrow \omega^{\alpha\beta}(\vec{q}_1,\vec{q}_2) = \frac{1}{N} \sum_{\substack{x_1 a \\ x_2 b}} e^{i\vec{q}_1\cdot(\vec{x}_1+\vec{u}_a)} s^{\alpha}_a \omega^0_{\vec{x}_1 a,\vec{x}_2 b} s^{\beta}_b e^{-i\vec{q}_2\cdot(\vec{x}_2+\vec{u}_b)}$$

$$= \frac{N}{N} \sum_{a,x_2 b} e^{i\vec{q}_1\cdot\vec{u}_a} s^{\alpha}_a \omega^0_{0a,\vec{x}_2 b} s^{\beta}_b e^{-i\vec{q}_2\cdot(\vec{x}_2+\vec{u}_b)}$$

$$= \sum_{a,\vec{x}_2 b} e^{i\vec{q}_1\cdot\vec{u}_a} s^{\alpha}_a \omega^0_{0a,\vec{x}_2 b} s^{\beta}_b e^{-i\vec{q}_2\cdot(\vec{x}_2+\vec{u}_b)}$$

$$\omega_{\alpha\beta}(\vec{q}_1,\vec{q}_2) = \sum_{a,\vec{x}_2 b} e^{i\vec{q}_1\cdot\vec{u}_a} s_a^\alpha \omega^0_{0a,\vec{x}_2 b} s_b^\beta e^{-i\vec{q}_2\cdot(\vec{x}_2+\vec{u}_b)}$$

now, $\omega^0_{0a,\vec{x}_2 b} = \omega^0_{\vec{x}_2 b,\,0a}$ since ω^0 is symmetric.

the symmetry is of the form $\omega_{0a,x_2 b} = \omega_{x_2 b,\,0a}$

$$\hat{\omega}_{ab}(\vec{q}) = \sum_{x_2} \omega^0_{0a,\vec{x}_2 b} e^{-i\vec{q}\cdot\vec{x}_2}$$

$$\hat{\omega}_{ba}(\vec{q}) = \sum_{x_2} \omega^0_{0b,\vec{x}_2 a} e^{-i\vec{q}\cdot\vec{x}_2}$$

$$\hat{\omega}_{ab}(\vec{q}) = \sum_{x_2} \omega^0_{0a,\vec{x}_2 b} e^{-i\vec{q}\cdot\vec{x}_2}$$

$$\Rightarrow \hat{\omega}_{ab}(\vec{q}) = \sum_{x_2} \omega^0_{\vec{x}_2 b,\,0a} e^{-i\vec{q}\cdot\vec{x}_2}$$

at $\vec{q}=0$, $\hat{\omega}_{ab}(\vec{q}=0) = \sum_{x_2} \omega^0_{0a,\vec{x}_2 b}$

$$\hat{\omega}_{ba}(\vec{q}=0) = \sum_{x_2} \omega^0_{0b,\vec{x}_2 a}$$

$$\sum_b \hat{\omega}_{ab}(\vec{q}=0)\, s_b^\alpha = \gamma^\alpha s_a^\alpha$$

$$\Rightarrow \sum_b \left(\sum_{x_2} \omega^0_{0a,\vec{x}_2 b}\right) s_b^\alpha = \gamma^\alpha s_a^\alpha$$

$$\Rightarrow \frac{1}{N}\sum_b \sum_{x_1,x_2} \left(\omega^0_{\vec{x}_1 a,\vec{x}_2 b}\right) s_b^\alpha = \gamma^\alpha s_a^\alpha$$

$$\Rightarrow \frac{1}{N} \sum_{x_1,x_2 b} \omega^0_{\vec{x}_1 a,\vec{x}_2 b}\, s_b^\alpha = \gamma^\alpha s_a^\alpha$$

See Almatt and Lidiard to see how they showed these are orthogonal. In Almatt and Lidiard, they deal with the ω^0 matrix's eigenvalues.

* Ask Dallas about this later. This is important for the orthogonality of the new basis vectors $\bar{\Phi}_q^\alpha$ — Eq 1

$$\phi_q^{\alpha}(\vec{x}+\vec{u}_a) = \frac{1}{\sqrt{N}} e^{i\vec{q}\cdot(\vec{x}+\vec{u}_a)} s_a^{\alpha}$$

$$\langle \phi_q^{\alpha} \phi_{q'}^{\beta} \rangle = \sum_{xa} \phi_q^{\alpha *}(\vec{x}+\vec{u}_a)\, \phi_{q'}^{\beta}(\vec{x}+\vec{u}_a)$$

$$= \frac{1}{N} \sum_{xa} e^{-i\vec{q}\cdot(\vec{x}+\vec{u}_a)} s_a^{\alpha} e^{i\vec{q}'\cdot(\vec{x}+\vec{u}_a)} s_a^{\beta}$$

$$= \frac{1}{N} \sum_{xa} e^{i\vec{x}\cdot(\vec{q}'-\vec{q})} e^{i\vec{u}_a\cdot(\vec{q}'-\vec{q})} s_a^{\alpha} s_a^{\beta}$$

$$= \frac{1}{N} \sum_{a} e^{i\vec{u}_a\cdot(\vec{q}'-\vec{q})} s_a^{\alpha} s_a^{\beta} \sum_{\vec{x}} e^{i\vec{x}\cdot(\vec{q}'-\vec{q})}$$

$$= \frac{1}{N} \cdot N\, \delta(\vec{q}-\vec{q}') \sum_{a} s_a^{\alpha} s_a^{\beta}$$

$$= \delta(\vec{q}-\vec{q}') \sum_{a} s_a^{\alpha} s_a^{\beta}$$

$\Rightarrow \langle \phi_q^{\alpha} \phi_{q'}^{\beta} \rangle = \delta(\vec{q}-\vec{q}')\, \delta_{\alpha\beta}$, which establishes the orthogonality of the basis vectors.

$$\langle \phi_{\vec{q}}^{\alpha} \phi_{\vec{q}'}^{\beta} \rangle = \delta(\vec{q}-\vec{q}')\, \delta_{\alpha\beta}$$

$$\hat{w}_{\alpha\beta}(\vec{q},\vec{q}') = \frac{1}{N} \sum_{xa,\vec{x}'b} \phi_q^{\alpha}(\vec{x}+\vec{u}_a)\, w^0_{\vec{x}a,\vec{x}'b}\, \phi_{q'}^{*\beta}(\vec{x}'+\vec{u}_b)$$

$$\hat{w}_{\alpha\beta}(\vec{q},\vec{q}') = \frac{1}{N} \sum_{xa,\vec{x}'b} e^{i\vec{q}\cdot(\vec{x}+\vec{u}_a)} s_a^{\alpha}\, w^0_{\vec{x}a,\vec{x}'b}\, s_b^{\beta} e^{-i\vec{q}'\cdot(\vec{x}'+\vec{u}_b)}$$

$$\hat{w}_{\alpha\beta}(\vec{q},\vec{q}') = \frac{1}{N} \sum_{xa,\vec{x}'b} e^{i\vec{q}\cdot(\vec{x}+\vec{u}_a)} s_a^{\alpha}\, w^0_{\vec{x}a,\vec{x}'b}\, s_b^{\beta} e^{-i\vec{q}'\cdot(\vec{x}'+\vec{u}_b)} \quad (1)$$

$$= \frac{1}{N} \sum_{\substack{\vec{x}a \\ \vec{x}b}} e^{i\vec{q}\cdot(\vec{x}'+\vec{u}_b)} s_a^{\alpha}\, w^0_{\vec{x}'b,\vec{x}a}\, s_b^{\beta} e^{-i\vec{q}'\cdot(\vec{x}+\vec{u}_a)} \quad (2)$$

consideration of (1) and (2) implies that

$$\hat{w}_{\alpha\beta}(\vec{q},\vec{q}') = \frac{1}{N} \delta(\vec{q}-\vec{q}') \sum_{\substack{\vec{x}a \\ \vec{x}b}} s_a^{\alpha} w^0_{\vec{x}a,\vec{x}b} s_b^{\beta} e^{i\vec{q}\cdot(\vec{x}+\vec{u}_a-\vec{x}'-\vec{u}_b)}$$

$$= \delta(\vec{q}-\vec{q}') \sum_{\substack{\vec{x}a \\ \vec{x}'b}} s_a^{\alpha} w^0_{\vec{x}a,\vec{x}'b} s_b^{\beta} e^{i\vec{q}\cdot(\vec{u}_a-\vec{x}-\vec{u}_b)}$$

$$\hat{w}_{\alpha\beta}(\vec{q},\vec{q}') = \delta(\vec{q}-\vec{q}') \sum_{\substack{\vec{x}a \\ \vec{x}'b}} s_a^\alpha \, w^0_{\vec{x}a,\vec{x}'b} \, s_b^\beta \, e^{-i\vec{q}\cdot(\vec{x}+\vec{u}_b-\vec{u}_a)}$$

$$\Rightarrow w^0_{\vec{x}a,\vec{x}'b} = \frac{1}{N}\sum_{\vec{q},\vec{q}'} e^{-i\vec{q}\cdot(\vec{x}+\vec{u}_a)} \left(\sum_{\alpha\beta} s_a^\alpha \, w_{\alpha\beta}(\vec{q},\vec{q}') \, s_b^\beta\right) e^{i\vec{q}'\cdot(\vec{x}'+\vec{u}_b)}$$

(Revise on why these integrals are always carried over the 1st BZ.)

$$= \frac{1}{N}\sum_{\vec{q}\vec{q}'}\sum_{\alpha\beta} e^{-i\vec{q}\cdot(\vec{x}+\vec{u}_a)} s_a^\alpha \, w_{\alpha\beta}(\vec{q}\;\;) \delta(\vec{q}-\vec{q}') \, s_b^\beta \, e^{i\vec{q}'\cdot(\vec{x}'+\vec{u}_b)}$$

$$= \frac{1}{N}\sum_{\vec{q}}\sum_{\alpha\beta} e^{-i\vec{q}\cdot(\vec{x}+\vec{u}_a)} s_a^\alpha \, w_{\alpha\beta}(q) \, s_b^\beta \, e^{i\vec{q}\cdot(\vec{x}'+\vec{u}_b)}$$

where the sum is over the allowed points in the Brillouin zone due to periodic boundary conditions.

$$= \frac{1}{N}\cdot\frac{1}{\Delta\vec{q}}\sum_{\vec{q}} \Delta\vec{q}\; e^{-i\vec{q}\cdot(\vec{x}+\vec{u}_a)} \sum_{\alpha\beta} s_a^\alpha \, w_{\alpha\beta}(\vec{q}) \, s_b^\beta \, e^{i\vec{q}\cdot(\vec{x}'+\vec{u}_b)}$$

$$= \frac{1}{N}\frac{1}{\Delta\vec{q}}\int_{BZ} d\vec{q}\; e^{-i\vec{q}\cdot(\vec{x}+\vec{u}_a)} \sum_{\alpha\beta} s_a^\alpha \, w_{\alpha\beta}(\vec{q}) \, s_b^\beta \, e^{i\vec{q}\cdot(\vec{x}'+\vec{u}_b)}$$

$\Delta\vec{q}$ → volume allowed per k-point $= \frac{(2\pi)^3}{V}$

$$= \frac{1}{N} V \int_{BZ} \frac{d\vec{q}}{(2\pi)^3} e^{-i\vec{q}\cdot\delta x} \sum_{\alpha\beta} s_a^\alpha \, w_{\alpha\beta}(\vec{q}) \, s_b^\beta$$

$$= V_0 \int_{BZ} \frac{d\vec{q}}{(2\pi)^3} e^{-i\vec{q}\cdot\delta\vec{x}} \sum_{\alpha\beta} s_a^\alpha \, w_{\alpha\beta}(\vec{q}) \, s_b^\beta = w^0_{\vec{x}a,\vec{x}b}$$

$$\Rightarrow w^0_{\vec{x}a,\vec{x}'b} = V_0 \int_{BZ} \frac{d\vec{q}}{(2\pi)^3} e^{-i\vec{q}\cdot\delta\vec{x}} \sum_{\alpha\beta} s_a^\alpha \, w_{\alpha\beta}(\vec{q}) \, s_b^\beta$$

$$\Rightarrow w^0_{\vec{x}a,\vec{x}'b} = V_0 \int_{BZ} \frac{d\vec{q}}{(2\pi)^3} e^{-i\vec{q}\cdot\delta\vec{x}} \sum_{\alpha\beta} s_a^\alpha \, w_{\alpha\beta}(\vec{q}) \, s_b^\beta \quad - \boxed{1}$$

Similarly:-

$$g^0_{\vec{x}a,\vec{x}'b} = V_0 \int_{BZ} \frac{d\vec{q}}{(2\pi)^3} e^{-i\vec{q}\cdot\delta\vec{x}} \sum_{\alpha\beta} s^\alpha_a \, \omega^{-1}_{\alpha\beta}(q) \, s^\beta_b \quad - (II)$$

we need to calculate eqn (II). To do that, we'll block invert $\omega_{\alpha\beta}(\vec{q})$, ie :-

$$\bar{\omega}(\vec{q}) = \left(\begin{array}{c|c} DD(q) & RD(q) \\ \hline DR(q) & DD(q) \end{array} \right)$$

To find the elements of this, we use the general equation for the transform :-

$$\omega_{\alpha\beta}(\vec{q}) = \sum_{a,\vec{x}'b} e^{-i\vec{q}\cdot\delta\vec{x}} \, s^\alpha_a \, \omega^0_{0a,\vec{x}b} \, s^\beta_b$$

now $DD(\vec{q}) = \omega_{00}(\vec{q}) = \sum_{a,\vec{x}'b} e^{-i\vec{q}\cdot\delta\vec{x}} \, s^0_a \, \omega^0_{0a,\vec{x}'b} \, s^0_b$

$$= \sum_{a,\vec{x}'b} e^{-i\vec{q}\cdot\delta\vec{x}} \, (p^{0,v}_a)^{1/2} \, \omega^0_{0a,\vec{x}'b} \, (p^{0,v}_b)^{1/2}$$

$$\Rightarrow DD(\vec{q}) = \sum_{a,\vec{x}'b} e^{-i\vec{q}\cdot\delta x} \, (p^{0,v}_a \, p^{0,v}_b)^{1/2} \, \omega^0_{0a,\vec{x}'b} \rightarrow$$ but before that, we'll first figure out how $\vec{s}^\alpha, r^\alpha$ are calculated in the code.

Further doubt regarding Q1 :-

we have $\sum_b \hat{\omega}_{ab}(\vec{q}=0) \, s^\alpha_b = r^\alpha s^\alpha_a$ I don't understand how this is symmetric.

$$= \sum_{\vec{x}'b} \omega^0_{0a,\vec{x}'b} \, s^\alpha_b = r^\alpha s^\alpha_a$$

Now, let us consider the equation :-

$$\sum_{\vec{x}'b} \omega^0_{0a,\vec{x}'b} \, a^\alpha_{\vec{x}'b} = r^\alpha a^\alpha_{\vec{x}'a}$$

now, since ω^0 is symmetric, the eigenvectors $\vec{a}^\alpha$ are orthogonal

→ Prove that $\{s^\alpha\}$ form a complete and orthogonal vector basis in state (or site) space.

$$\omega^0_{0a,\vec{x}b} = \omega^0_{ab}(\vec{x})$$

$$\hat{\omega}_{ab}(\vec{q}) = \sum_{\vec{x}} \omega^0_{ab}(\vec{x}) \, e^{i\vec{q}\cdot\vec{x}}$$

Then, at $\vec{q}=0$, $\hat{\omega}_{ab}(\vec{q}=0) = \sum_{\vec{x}} \omega^0_{ab}(\vec{x})$

now, if $(\gamma^\alpha, \bar{s}^\alpha)$ is the α^{th} eigenvalue - eigenvector pair of $\hat{w}(\vec{q}=0)$

$$\sum_b \hat{w}_{ab}(\vec{q}=0)\, \bar{s}_b^\alpha = \gamma^\alpha s_a^\alpha \quad -(1)$$

$$\Rightarrow \sum_b \left(\sum_n w^0_{ab}(\vec{n}) \right) \bar{s}_b^\alpha = \gamma^\alpha s_a^\alpha$$

$$\Rightarrow \sum_n \sum_b w^0_{0a,\vec{n}b}\, \bar{s}_b^\alpha = \gamma^\alpha s_a^\alpha$$

$$\Rightarrow \frac{1}{N} \sum_{\vec{x},\vec{x}'b} w^0_{\vec{x}a,\vec{x}'b}\, \bar{s}_b^\alpha = \gamma^\alpha s_a^\alpha \quad - \underline{\text{eq}^n\ 36}$$

but for s_a^α to be a complete and orthogonal set of vectors, we must have, in equation (1) :-

$$\hat{w}_{ab}(\vec{q}=0) = \hat{w}_{ba}(\vec{q}=0) \quad (\text{since at } \vec{q}=0,\ \hat{w}(q) \text{ is real})$$

which means :-

$$\sum_{\vec{x}} w^0_{ab}(\vec{x}) = \sum_{\vec{x}} w^0_{ba}(\vec{x})$$

$$\Rightarrow \sum_{\vec{x}} w^0_{0a,\vec{x}b} = \sum_{\vec{x}} w^0_{0b,\vec{x}a}$$

→ All of the terms on the left are symmetrized transition rates from the a^{th} site, while all of the terms on the right are transitions from the b^{th} site.
How can they be equal in general? → See Page 15

Mapping of Taylor expansions on a spherical harmonic basis.

we have, for all the blocks, Taylor series expansions of the form :-

$$e^{-i\vec{q}\cdot\delta\vec{x}} = 1 - i\vec{q}\cdot\delta\vec{x} + \frac{i^2}{2!}(\vec{q}\cdot\delta\vec{x})^2 + \frac{i^3}{3!}(\vec{q}\cdot\delta\vec{x})^3 + \frac{i^4}{4!}(\vec{q}\cdot\delta\vec{x})^4 + \ldots.$$

where $\delta\vec{x} \equiv \delta\vec{x}_{ij}$ is a jump displacement. See the equations of the jumps for more clarity.

$$\Rightarrow e^{-i\vec{q}\cdot\delta\vec{x}} = 1 - iq\left(\hat{q}_x \delta x_x + \hat{q}_y \delta x_y + q_z \delta x_z\right)$$

$$- \frac{1}{2} q^2 \left(q_x \delta x_x + q_y \delta x_y + q_z \delta x_z\right)^2$$

$$- \frac{i}{6} q^3 \left(q_x \delta x_x + q_y \delta x_y + q_z \delta x_z\right)^3$$

$$+ \frac{1}{24} q^4 (q_x \delta x_x + q_y \delta x_y + q_z \delta x_z)^4 + \cdots$$

ie, we can write :-

$$e^{-i \vec{q} \cdot \delta \vec{x}} = \sum_{n=0}^{4} \frac{i^n}{n!} q^n (q_x \delta x_x + q_y \delta x_y + q_z \delta x_z)^n \quad - (I)$$

Now, let us consider the term for $n=2$, ie

$$T_2 = \frac{-1}{2} q^2 (q_x \delta x_x + q_y \delta x_y + q_z \delta x_z)^2$$

$$\Rightarrow T_2 = -\frac{1}{2} q^2 (q_x^2 \delta x_x^2 + q_y^2 \delta x_y^2 + q_z^2 \delta x_z^2 + 2 q_x q_y \delta x_x \delta x_y + 2 q_y q_z \delta x_y \delta x_z + 2 q_z q_x \delta x_z \delta x_x)$$

which we can also write as :-

$$T_2 = -\frac{1}{2} q^2 \sum_{n_1+n_2+n_3=2} q_x^{n_1} q_y^{n_2} q_z^{n_3} C_{n_1 n_2 n_3, ij}$$

where 'ij' denotes that the coefficients are dependent on the jump distances between states i and j

Thus, we can write the entire Taylor series as :-

$$e^{-i \vec{q} \cdot \delta \vec{x}_{ij}} = \sum_{n=0}^{4} \frac{i^n}{n!} q^n \sum_{n_1+n_2+n_3=n} C_{n_1 n_2 n_3, ij} q_x^{n_1} q_y^{n_2} q_z^{n_3} \quad - (II)$$

$$e^{-i \vec{q} \cdot \delta \vec{x}_{ij}} = \sum_{n=0}^{4} \frac{i^n}{n!} q^n \sum_{n_1+n_2+n_3=n} C_{n_1 n_2 n_3, ij} q_x^{n_1} q_y^{n_2} q_z^{n_3} \quad - (III)$$

In general, we can also write this, for a given L_{max} as :-

$$e^{-i \vec{q} \cdot \delta \vec{x}_{ij}} = \sum_{n=0}^{4} \frac{i^n}{n!} q^n \sum_{l=0}^{L_{max}} \sum_{n_1+n_2+n_3=l} C^{l, ij}_{n_1 n_2 n_3} q_x^{n_1} q_y^{n_2} q_z^{n_3} \quad - (IV)$$

so that when $l \neq n$ $\quad C^{l, ij}_{n_1 n_2 n_3} = 0$

clearly here L_{max} must be ≥ 4.

ie, for each n, we'll have an array of the coefficients $C^{l}_{ij}[n_1, n_2 n_3]$ so that :-

$$\left\{ C^{l, ij}_{n_1 n_2 n_3} \right\}_{l=0}^{L_{max}} = 0 \quad \forall \; l \neq n$$

Now, how many elements will we have in each such array?

→ We consider $L_{max} = 4$, like in the code.

Then for $l = 0$

n_1	n_2	n_3
0	0	0

$l = 1$

n_1	n_2	n_3
0	0	1
0	1	0
1	0	0

$l = 2$

$2\ 0\ 0$
$0\ 2\ 0$
$0\ 0\ 2$
$1\ 1\ 0$
$0\ 1\ 1$
$1\ 0\ 1$

$l = 3$

$3\ 0\ 0$
$0\ 3\ 0$
$0\ 0\ 3$
$2\ 1\ 0$
$1\ 2\ 0$
$0\ 1\ 2$
$0\ 2\ 1$
$1\ 0\ 2$
$2\ 0\ 1$
$1\ 1\ 1$

$l = 4$ 15 terms

Thus, for every n, the array $\{C^{l,ij}_{n_1 n_2 n_3}\}_{l=0}^{l_{max}}$ has 35 terms, and they are non-zero for only when $l = n$.

now, (1) we define an array for every jump, ij as :-

Taylor. pexp = [1, dx_x, dx_y, dx_z, dx_x^2, dx_y^2, dx_z^2, $dx_x dx_y$, $dx_x dx_z$, $dx_y dx_z$, dx_x^3, ... 35 terms]

and (2) another array

Taylor. powercoeff = $\left[[C^{l}_{n_1 n_2 n_3}]_{l=0 \text{ to } l_{max}}^{n=0}, [C^{l}_{n_1 n_2 n_3}]_{l=0 \text{ to } l_{max}}^{n=1}, \cdots [C^{l}_{n_1 n_2 n_3}]_{l=0 \text{ to } l_{max}}^{n=4} \right]$

so that $C^{l,ij}_{n_1 n_2 n_3} = C^{l}_{n_1 n_2 n_3} * (dx_x^{n_1} dx_y^{n_2} dx_z^{n_3})_{ij}$

and $[C^{l}_{n_1 n_2 n_3}]_{l=0 \text{ to } l_{max}}^{n=k} = 0 \quad \forall\ l \neq k$ (ie, $n_1 + n_2 + n_3 = l \neq k$),

In the array Taylor. powercoeff, $[C^{l}_{n_1 n_2 n_3}]_{n=k}$ are zero when $l \neq k$, and are the binomial constants of $(a+b+c)^k$ when $l = k$.

(3) We define another array

pre = $\left[\frac{i^n}{n!}\right]_{n=0}^{4}$

now if we do :-

fore[n] * Taylor.powercoeff[n] * Taylor.perexp

and look at equation (IV) :-

$$e^{-i\vec{q}\cdot\delta\vec{x}_{ij}} = \sum_{n=0}^{4} \frac{i^n}{n!} q^n \sum_{l=0}^{Lmax} \sum_{n_1+n_2+n_3=l} C^{l,n,ij}_{n_1 n_2 n_3} q_x^{n_1} q_y^{n_2} q_z^{n_3} \quad - (IV)$$

we get all of the coefficients of the terms that appear in the inner sum, ie,

$$\frac{i^n}{n!} \sum_{l=0}^{Lmax} \sum_{n_1+n_2+n_3=l}$$, ie

$$\frac{i^n}{n!}\left[C^{0,n,ij}_{000}, C^{1,n,ij}_{100}, C^{1,n,ij}_{010}, C^{1,n,ij}_{001}, C^{2,n,ij}_{200}, C^{2,n,ij}_{020}, \ldots \text{35 terms} \right]$$

where $C^{l,n,ij}_{n_1 n_2 n_3} = \delta_{ln} \times \left(\delta x_x^{ij\,n_1}\, \delta x_y^{ij\,n_2}\, \delta x_z^{ij\,n_3}\right) \times b_{n_1 n_2 n_3}$

$b_{n_1 n_2 n_3}$ → binomial constant

Taylor.coeff list contains these arrays for $n = 0, 1, 2, \ldots, Lmax$, truncated, after the last non-zero term → which is found from Taylor.powrange. This means that when $l > n$, these $C^{l,n,ij}_{n_1 n_2 n_3}$ are going to be zero anyway, so we truncate them.

Thus,

(C[n][2])[:, i, j]

0 1 2 3

$$= \frac{i^n}{n!}\left[C^{0,n,ij}_{000}, C^{1,n,ij}_{100}, C^{1,n,ij}_{010}, C^{1,n,ij}_{001}, C^{2,n,ij}_{200}, C^{2,n,ij}_{020}, \ldots \text{last non-zero term} \right]$$

$C^{0,n,ij}_{000} = 1$

powrange[n]

Next, how do we construct the entire Taylor series :-

self.omega_Taylor = sum(symmrate * expansion for symmrate, expansion in zip(self.symmrates, self.TaylorJumps)

expansion = [csum(0, 0, 1 x N x N), csum(1, 1, 4 x N x N), csum(2, 2, 10 x N x N),

csum(3, 3, 20 x N x N), csum(4, 4, 35 x N x N)]

symmrate * expansion

= [csum(0, 0, symmrate x (1 x N x N)), csum(1, 1, symmrate x (4 x N x N),
csum(2, 2, symmrate x (10 x N x N)),
csum(3, 3, symmrate x (20 x N x N)),
csum(4, 4, symmrate x (35 x N x N))]

sum(symmrate * expansion)

c[n][2]

= [csum(0, 0, $\sum_{jt}$ symmrate x (1 x N x N)), csum(1, 1, $\sum_{jt}$ symmrate x (4 x N x N)),
csum(2, 2, $\sum_{jt}$ symmrate x (10 x N x N)), csum(3, 3, $\sum_{jt}$ symmrate x (20 x N x N)),
csum(4, 4, $\sum_{jt}$ symmrate x (35 x N x N))] = self.omega_Taylor

Now, let's see how the eigenvalues and vectors are calculated.

def DiagGamma(): → diagonalize the Fourier transform at the Gamma point.

omega = self.omega_Taylor.

gammacoeff = None

for (n, l, coeff) in omega.coeff_list:

if n < 0: error

if n == 0:

if l != 0: error (angular dependence at n=0 ⇒ error).

gammacoeff = coeff[0]

if gammacoeff is None:

gammacoeff = np.zeros((self.N, self.N), dt=complex)

v, vr = LA.eigh(gammacoeff)

return $-x$, $\forall x$.

gammacoeff

$$= \sum_{jt} \text{symmrate}_{jt} \times (N \times N)$$

First, let us see if we pictured the summation correctly.

→ This is in PowerExpansion module, functions 'sumcoeff'.

@classmethod (Revise class methods after this).

```
def sumcoeff(cls, a, b, α=1, β=1, inplace=False):

    acoeff = getattr(a, 'coeffList', a)
    bcoeff = getattr(b, 'coeffList', b)

    # scalarproductcoeff → see later.

    if not inplace:

        c = [(an, almax, α*apow) for an, almax, apow in acoeff]

    else

        c = acoeff.

    for bn, blmax, bpow in bcoeff:
        cpow = β * bpow.
        matched = False
        for coeffindex, cmatch in enumerate(c):
            if cmatch[0] == bn
                matched = True
                break
        if not matched:
            c.append((bn, blmax, cpow))

        else:
            clmax0 = cmatch[1]  # assume we get a match the first time.
            if blmax > clmax0:
                cpow[:cls.powrange[clmax0]] += cmatch[2]

            c[coeffindex] = (bn, blmax, cpow)
```

Remember that (an, almax) and (bn, blmax) are equal in TaylorJumps.

→ only this part is executed in omega-Taylor

```
    else:
        coeff = cmatch[2]

        coeff[:cls.powrange[blmax]] += cpow.
```

↖ is this necessary? should'nt they be of the same dimension? Yes, but probably this

c.sort (key = cls.__sortkey) → look at how the sorting happens.
return c.

was written earlier.

```
def __sortkey(cls, entry):
    return (entry[0] + entry[1] / (cls.lmax + 1))
```

* write down the addition part again:-

```
def sumcoeff(cls, a, b, α=1, β=1):
    c = [an, almax, α*acoeff for an, almax, acoeff in a]
    for bn, blmax, bcoeff in b:
        cpow = β * bcoeff
        for cindex, cmatch in enumerate(c):
            if cmatch[0] == bn:
                break
        coeff = cmatch[2]
        coeff += cpow
    return c.sort(cls.__sortkey)
```

Now, let us see how the eigenvalues of the Fourier transform are found at $q = 0$:-

we have

self.omegaTaylor = [csum(0, 0, $\sum_{jt}$ symrate × (1×N×N)), csum(1, 1, $\sum_{jt}$ symrate × (4×N×N)),
csum(2, 2, $\sum_{jt}$ symrate × (10×N×N)),
csum(3, 3, $\sum_{jt}$ symrate × (20×N×N)),
csum(4, 4, $\sum_{jt}$ symrate × (35×N×N))]

terms for q^0 in eq IV; terms for q^1 in eq IV; terms for q^2 in eq IV; terms for q^3 in eq IV; terms for q^4 in eq IV

now what are the terms for $q = 0$

we have $w(q) = \sum_R w_{ij}(R)\, e^{-i\vec{q}\cdot\delta\vec{x}_{ij}}$

$= \sum_R w_{ij}(\vec{R}) \left(1 - \vec{q}\cdot\delta\vec{x} + \frac{i^2}{2!}(\vec{q}\cdot\delta\vec{x})^2 + \frac{i^3}{3!}(\vec{q}\cdot\delta\vec{x})^3 + \frac{i^4}{4!}(\vec{q}\cdot\delta\vec{x})^4 + \dots\right)$

when $q = 0$

$$\omega(\vec{q}) = \sum_{\vec{R}} \omega_{ij}(\vec{R})$$

but $\sum_{\vec{R}} \omega_{ij}(\vec{R})$ are also the terms corresponding to $q = 0$

ie, the first $\vec{q}$ of omega-Taylor.

ie, omega-Taylor[0][2] $= \sum_{\vec{R}} \omega_{ij}(\vec{R}) = \hat{\omega}(\vec{q} = 0)$

$$\Rightarrow \omega_{ij}(\vec{q}=0) = \sum_{\vec{x}} \omega_{0i,\vec{x}j} = \hat{\omega}(\vec{q}=0)$$

$$\Rightarrow \omega_{ij}(\vec{q}=0) = \sum_{\vec{x}} \omega_{0i,\vec{x}j}$$

$$\Rightarrow \omega_{ij}(\vec{q}=0) \frac{1}{N} \sum_{\vec{x}} \sum_{\vec{x}'} \omega_{\vec{x}i,\vec{x}'j}$$

similarly $\omega_{ji}(\vec{q}=0) = \sum_{\vec{x}} \omega_{0j,\vec{x}i}$

and $\omega_{ij}(\vec{q}=0) = \sum_{\vec{x}} \omega_{0i,\vec{x}j}$

} how are these two equal?

$$\omega_{0i,\vec{x}j} = (P_i^{0,0})^{1/2} W_{0i,\vec{x}j} (P_j^{0,0})^{-1/2}$$

$$\omega_{0j,\vec{x}i} = (P_j^{0,0})^{1/2} W_{0j,\vec{x}i} (P_i^{0,0})^{-1/2}$$

by detailed balance

$$P_i^{0,0} W_{0i,\vec{x}j} = P_j^{0,0} W_{\vec{x}j,0i} = P_j^{0,0} W_{0j,-\vec{x}i}$$

$$= P_j^{0,0} W_{0j,\vec{x}i}$$

only if we have a Bravais lattice with inversion symmetry. what about when we don't? what about dumbbells?

The rates corresponding to different jump types get added, so we need only worry about the inversion symmetry issue.

Now, $P_i^{0,0} W_{0i,\vec{x}j} = P_j^{0,0} W_{0j,\vec{x}i}$ ← assuming inversion symmetry

$$\Rightarrow (P_i^{0,0})^{1/2} W_{0i,\vec{x}j} (P_j^{0,0})^{-1/2} = (P_j^{0,0})^{1/2} W_{0j,\vec{x}i} (P_i^{0,0})^{-1/2}$$

$$\Rightarrow \omega_{0i,\vec{x}j} = \omega_{0j,\vec{x}i}$$

$$\therefore \sum_{\vec{x}} w_{0i,\vec{x}j} = \sum_{\vec{x}} w_{0j,\vec{x}i}$$

$$\Rightarrow \hat{w}_{ij}(\vec{q}=0) = w_{ji}(\vec{q}=0)$$

↳ which is why $\hat{w}$ is symmetric.

Now, let us see how the diagonal jumps are added into self.omega_Taylor.

Line 350 in Gfcalc:

self.omega_Taylor += self.escape - 350

self.escape = -np.diag([sum(self.SEjumps[i,J] * pretrans / pre[wi] * np.exp(betamu[wi] - BET) for J, pretrans, BET in zip(count, preT, betamuT)) for i, wi in enumerate(self.invmap)]) / self.mxrate.

Let us try to understand SEjumps :-

def FourierTransformJumps(self, jumpnetwork, N, kpts):

FTjumps = zeros((len(jumpnetwork), self.Nkpt, N, N), dt=complex)

SEjumps = zeros((N, len(jumpnetwork)), dt=int)

for J, jumplist in enumerate(jumpnetwork):

for i,j, dr in jumplist:

FTjumps[J, :, i, j] = exp(1j * dot(kpts, dr))

SEjumps[i, J] += 1

↳ no. of jumps coming out of a state.

return FTjumps, SEjumps.

→ Ok, so what is self.escape?

N x N diagonal matrix, containing the negative of the sum of all the escape rates out of a particular site.

→ Ok, now, what is self.omegaTaylor_t = self.escape?
- Adds in the missing diagonal jump terms.
- now remember that self.escape is an np array.
- see how this addition takes place.

* scalarproductcoeff, tensorproductcoeff, numpy tensordot, how to define axes and what does it mean?

✓

look in the jupyter notebook tensordot_example to understand how numpy.tensordot works.

def DiagGamma(self, omega=None)

if omega = None: omega = self.omega-Taylor.
gammacoeff = None
for n, l, coeff in omega.coeffslist:

if n == 0:
gammacoeff = -coeff[0] → this is the N x N matrix containing $\Rightarrow \hat{W}_{ij}(\vec{q}=0) = \sum_{\vec{R}} W^{0}_{ij}(\vec{R})$

r, vr = LA.eigh(gammacoeff)

return -r, vr. → now → the jumpnetwork does not contain diagonal terms right?
→ what about those?
↳ we'll see how this is used.

What does it mean to compute tensordot product along specified axes.
⇒ Look at the appropriate Jupyter notebook to understand how tensordot works.

RECAP

self.omegaTaylor
= sum(symmrate * expansion for symmrate, expansion in zip(symmrates, self.TaylorJumps))

$= \sum_{jt} Symmrate_{jt} \times expansion_{jt}$

$$= \sum_{jt} \text{Taylor3D}_{jt}\Big(C\Big[(0,0,\text{symrate}_{jt}\times(1\times N\times N)_{jt}),(1,1,\text{symrate}_{jt}(4\times N\times N)_{jt})$$

$$(2,2,\text{symrate}_{jt}(10\times N\times N)_{jt}),(3,3,\text{symrate}_{jt}(20\times N\times N)_{jt}),(4,4,\text{symrate}_{jt}(35\times N\times N)_{jt})\Big]\Big)$$

$$=$$

$$\text{Taylor3D}\Big(\text{coeff list}\Big[(0,0,\sum_{jt}\text{symrate}_{jt}\times(1\times N\times N)_{jt}),(1,1,\sum_{jt}\text{symrate}_{jt}(4\times N\times N)_{jt}),$$

$$(2,2,\sum_{jt}\text{symrate}_{jt}(10\times N\times N)_{jt}),(3,3,\sum_{jt}\text{symrate}_{jt}(20\times N\times N)_{jt}),$$

and $$(4,4,\sum_{t}\text{symrate}_{jt}(35\times N\times N)_{jt})\Big]\Big)$$

$$-\gamma, \vec{v}\gamma = \text{la.eigh}(\text{omega\_Taylor.coeff list}[0][2][0])$$

$$= \hat{\omega}_{ab}(\vec{q}=0) = \sum_{\vec{R}} \omega_{0a,\vec{R}b}$$

$\vec{v}\gamma$ → The set of vector $\{\vec{\xi}^{\alpha}\}$

$-\gamma$ → The set of eigenvalues $\vec{\gamma}^{\alpha}$ } → $\alpha = 0, \ldots, N\text{sites}-1$

Q. to ask Dallas? → Why should there be as many zero eigenvalues as there are disconnected jumpnetworks?
Can there not be a case that while one subset of states reach equilibrium, another disconnected subset of states have not?

Step 2

- Recast omega_Taylor in the new basis involving the eigenvectors at $q=0$.

** Before this - Go through the Jupyter notebook to understand how np.tensordot works.

line 364

self.omega_Taylor_rotate = (self.omega_Taylor.ldot(self.vγ.T)).γdot(self.vγ)

Note, la.eigh returns $\vec{v}\gamma$ → which is an $N\times N$ matrix, with the columns being the eigenvalues.

$\vec{v}\gamma.T$ → rows are now the eigenvalues.

Transformation rule of a second order tensor:-

If we have an orthogonal matrix Q,

then $T'_{ij} = Q_{pi} Q_{qj} T_{pq} \Rightarrow T'_{ij} = Q^T T Q$

So, applied to $\omega_{ab}(\vec{q})$

$$\omega_{\alpha\beta}(q) = \sum_{ab} Q_{\alpha a} Q_{\beta b} \omega_{ab}(\vec{q})$$
$$= \sum_{ab} S^{\alpha}_{a} \omega_{ab}(\vec{q}) S^{\beta}_{b}$$

where we have the orthogonal matrix $Q \equiv vy$ such that

$vy[\alpha][a] = S^{\alpha}_{a}$ the a^{th} component of the α^{th} eigenvectos.

$\therefore \tilde{\omega}(\vec{q}) = vy^T \hat{\omega}(\vec{q}) vy$ ← columns are vectors indexed to eigenvalues with components as states.

$\tilde{\omega}(\vec{q})$ indexed by eigenvalues; $\hat{\omega}(\vec{q})$ indexed by states

Now, let us see this how the left and right dots have been implemented.

First, we do the math:-

$$\tilde{\omega}(\vec{q}) = vy^T \hat{\omega}(\vec{q}) vy$$

$$\tilde{\omega}_{\alpha\beta}(\vec{q}) = \sum_{ab} S^{\alpha}_{a} \hat{\omega}_{ab}(\vec{q}) S^{\beta}_{b}$$

$$= \sum_{ab} \sum_{R} S^{\alpha}_{a} \omega_{ab}(\vec{R}) e^{-i\vec{q}\cdot\delta\vec{x}} S^{\beta}_{b}$$

$$= \sum_{ab} S^{\alpha}_{a} S^{\beta}_{b} \sum_{R} \omega_{ab}(\vec{R}) \left[1 - \vec{q}\cdot\delta\vec{x} + \frac{i^2}{2}(\vec{q}\cdot\delta\vec{x})^2 - \frac{i^3}{3!}(\vec{q}\cdot\delta\vec{x})^3 + \frac{i^4}{4!}(\vec{q}\cdot\delta x)^4 + \cdots\right]$$

$$\Rightarrow \tilde{\omega}_{\alpha\beta}(\vec{q}) = \sum_{ab} \sum_{\vec{R}} S^{\alpha}_{a} \omega_{ab}(\vec{R}) S^{\beta}_{b} - \sum_{ab} \sum_{\vec{R}} S^{\alpha}_{a} \omega_{ab}(\vec{R}) S^{\beta}_{b} \vec{q}\cdot\delta\vec{x}$$
$$+ \frac{i^2}{2} \sum_{ab} \sum_{\vec{R}} S^{\alpha}_{a} \omega_{ab}(\vec{R}) S^{\beta}_{b} (\vec{q}\cdot\delta\vec{x})^2$$
$$- \frac{i^3}{3!} \sum_{ab} \sum_{\vec{R}} S^{\alpha}_{a} \omega_{ab}(\vec{R}) S^{\beta}_{b} (\vec{q}\cdot\delta\vec{x})^3$$
$$+ i^4 \sum \sum S^{\alpha}_{a} \omega_{ab}(\vec{R}) S^{\beta}_{b} (\vec{q}\cdot\delta\vec{x})^4$$

$$= \sum_{ab} \sum_{\vec{R}} s_a^{\alpha} w_{ab}(\vec{R}) s_b^{\beta} - q \sum_{ab} \sum_{\vec{R}} s_a^{\alpha} w_{ab}(\vec{R}) s_b^{\beta} \left(\hat{q}_x \delta x_m + \hat{q}_y \delta x_y + \hat{q}_z \delta x_z \right)$$

$$+ \frac{i^2}{2} q^2 \sum_{ab} \sum_{\vec{R}} s_a^{\alpha} w_{ab}(\vec{R}) s_b^{\beta} \left(\hat{q}_x^2 \delta x_m^2 + \hat{q}_y^2 \delta x_y^2 + \hat{q}_z^2 \delta x_z^2 \right.$$

$$+ 2 \hat{q}_x \hat{q}_y \delta x_m \delta x_y + 2 \hat{q}_y \hat{q}_z \delta x_y \delta x_z$$

$$\left. + 2 \hat{q}_x \hat{q}_z \delta x_x \delta x_z \right)$$

$$+ \ldots$$

The coefficients of $\hat{q}_x$, $\hat{q}_y$, $\hat{q}_z$ terms for q^0, q^1, q^2, ... are stored, without being summed up over the sites, in:-

```
self.omega_Taylor =
Taylor3D( coefflist [ (0, 0, Σ_jt symmrate_jt x (1 x N_s x N_s)), (1, 1, Σ_jt symmrate x (4 x N_s x N_s))
                    , (2, 2, Σ_jt symmrate_jt x (10 x N_s x N_s)),
                      (3, 3, Σ_jt symmrate_jt (20 x N_s x N_s)),
                      (4, 4, Σ_jt symmrate_jt (35 x N_s x N_s))] )
```

where $N_s \equiv N_{sites}$, the no. of basis sites.

Note that :-

$$\left[\sum_{jt} \text{symmrate}_{jt} \times (1 \times N_s \times N_s) \right] [0] [a] [b]$$

$$= \sum_{\vec{R}} w_{ab}(\vec{R}) = \hat{w}_{ab}(\vec{q}=0)$$

line 364

```
self.omega_Taylor_rotate = (self.omega_Taylor.ldot(vr.T)).rdot(vr)
```

In powerExpansion module, class Taylor3d :-

```
def ldot(self, c):
    return type(self)(self.tensorproductcoeff(c, self, leftmultiply=True))

def rdot(self, c):
    return type(self)(self.tensorproductcoeff(c, self, leftmultiply=False))
```

So how does tensorproductcoeff work? → returns a new coefficient list.
→ Note – here we study the functions purely in the context of how it

handles the coefficient list and the

```
@classmethod
def tensorproductcoeff(cls, c, a, leftmultiply=True):
    acoeff = getattr(a, "coeff list", a)
    ca = []
    for an, alman, apow in acoeff:
        cmult = c
        if leftmultiply:
            shape = (apow.shape[0],) + cmult.shape[:-1] + apow.shape[2:]
            # apow → powlrange[alman], N, N
            # when c = vr, shape = (powlrange[alman], N, N)
            mat = zeros(shape, dt=complex)
            for p in range(powlrange[alman]):
                mat[p] = np.tensordot(cmult, apow[p], axes=1)
            ca.append((an, alman, mat))
        else:
            mat = tensordot(apow, cmult, axes=(-1, 0))
            ca.append((an, alman, mat))
    return ca
```

keep recalling that here c = vr or vr.T

In this block, cmult = c = vr.T

In this block, c = vr.

→ take vr.T, dot it with every N x N matrix in apow.
→ get new matrix, with the same shape as apow.

To see what this does, do the following :-

→ take a = [10 x 5 x 5] → we try with N = 5 and powlrange[alman] = 10

→ then take b = 5 x 5 → it, alman = 2

→ this is representative of vr.

→ then do → tensordot (a, b, axes = (-1, 0))

→ compare with :-

```
mat = zeros((10, 5, 5))
for i in range 10:
    mat[i, :, :] = dot(a[i, :, :], b)
```

see if we get the same things.

Try to figure out how tensordot works and how these are equivalent. How would you have done it?

In general, if we have $a = (X, N, N)$
$b = (N, N)$

then if c = tensordot (a, b, axes = (-1, 0)), then :-

$c \rightarrow (X, N, N)$ a.shape[-1] goes away and b.shape[0] goes away

So, we'll have :-

```
for i in range (X):
    for j in range (N):
        for k in range (N):   # b's remaining axes.
            # now the sum reduction
            for l in range (b.shape[0]):  # or a.shape[-1] → both must be the same
                c[i,j,k] += a[i,j,l] * b[k,l]
```

a's remaining axes

this means that

$C[i,j,k] = a[i,j,l] * b[k,l]$ in Einstein notation.

$\therefore c[i] = a[i] \cdot b$

In any matrix element multiplication like this, if you see repeated indices, try to think of it in terms of numpy.tensordot.

So that means

self.omega_Taylor_rotate = $\vec{v}_r$.omega_Taylor.$\vec{v}_r$
instance

where, the Taylor3D's omega_Taylor_rotate has every $N \times N$ matrix transformed by $\vec{v}_r$.

i.e self.omega_Taylor_rotate.coeff_list

$$= \Big[\Big(0, 0, \sum_{jt} \text{symmrate}_{jt} * \big(1x\ \vec{v}_\delta . (N\times N) . \vec{v}_r\big)_{jt} \Big) \quad \text{one matrix for } q^0,\ 3 \text{ for } q^1.$$

$$, \Big(1, 1, \sum_{jt} \text{symmrate}_{jt} * \big(4x\ \vec{v}_\delta . (N\times N) . \vec{v}_r\big)_{jt} \Big)$$

$$\Big(2, 2, \sum_{jt} \text{symmrate}_{jt} * \big(10x\ \vec{v}_\delta . (N\times N) . \vec{v}_r\big)_{jt} \Big) \quad \text{1 for } q^0,\ 3 \text{ for } q^1,\ 6 \text{ for } q^2$$

except the last 6, all are zeros.

$$\Big(3, 3, \sum_{jt} \text{symmrate}_{jt} * \big(20x\ \vec{v}_\delta . (N\times N) . \vec{v}_r\big)_{jt} \Big)$$

$$\Big(4, 4, \sum_{jt} \text{symmrate}_{jt} * \big(35x\ \vec{v}_\delta . (N\times N) . \vec{v}_r\big)_{jt} \Big) \Big]$$

Step 3

Now, we must extract the blocks of $\tilde{w}(\vec{q}) \equiv \left[\tilde{w}_{\alpha\beta}(q)\right]$

$$\tilde{w}_{\alpha\beta}(\vec{q}) = \left[\begin{array}{c|c} DD(q) & DR(q) \\ \hline RD(q) & RR(q) \end{array}\right]$$

$DD(q)$ = 1×1 matrix = $\tilde{w}_{00}(\vec{q})$ or $\tilde{w}_{0:Ndiff,\,0:Ndiff}(\vec{q})$

Why? Ask Dallas

First, in order to make python objects indexable, so that we can get/set the 'x^{th}' element of that object, like we do for an array as arr[x], we must define the following methods :-

```
__getitem__
__setitem__
__delitem__
```

Let us see what they are and how they are implemented for a Taylor3D object :-

① `__getitem__` → used to retrieve something from an object, based on a key we enter.

In the Taylor3D class :-

```
def __getitem__(self, key):
    if type(key) is not tuple:
```

```
        keyt = (key,)
    else:
        keyt = key
    return type(self)([(n, l, c[(slice(0, None, None),) + keyt])
                       for n, l, c in self.coeffList], nodeepcopy=True)
```

understand this advanced indexing method from the numpy reference manual.

How this works: Csee the Jupyter notebook for more details.
Once blocks are extracted, they are also 'reduced' - projected onto the Ylm space and then rid of all the zero terms (like q^0 and q^1 in coeffList[2][2])

To do this, we call the following methods in the Taylor3D class:-

```
def (reduce):
    self.reducecoeff (self.coeffList, inplace = True)
    self.collectcoeff (self.coeffList, inplace = True)
    return self
```

Now, let's see how these two methods work:-

(1) reducecoeff

- Before this, it is good to recall the quantities defined in __initTaylor3Dindexing__(cls, Lmax). Look for the Notability note on the PowerExpansion module in the 'Onsager Cod' subject name. Also, refer to the corresponding Jupyter notebooks.

- A little bit of recall
what do we have now:-

① omega_Taylor =

self.omega_Taylor_rotate.coeffList

$$= \Big[\Big(0, 0, \sum_{jt} \text{symmrate}_{jt} * (1 \times \vec{v}_\sigma . (N \times N) . \vec{v}_r)_{jt} \Big) \quad \text{one matrix for } q^0,$$

$$, \Big(1, 1, \sum_{jt} \text{symmrate}_{jt} * (4 \times \vec{v}_\sigma . (N \times N) . \vec{v}_r)_{jt} \Big) \quad \text{3 for } q^1.$$

$$\Big(2, 2, \sum_{jt} \text{symmrate}_{jt} * (10 \times \vec{v}_\sigma . (N \times N) . \vec{v}_r)_{jt} \Big) \quad \text{1 for } q^0,\ 3 \text{ for } q^1,\ 6 \text{ for } q^2$$

except the last 6, all are zeros.

$$\Big(3, 3, \sum_{jt} \text{symmrate}_{jt} * (20 \times \vec{v}_\sigma . (N \times N) . \vec{v}_r)_{jt} \Big)$$

$$\Big(4, 4, \sum_{jt} \text{symmrate}_{jt} * (35 \times \vec{v}_\sigma . (N \times N) . \vec{v}_r)_{jt} \Big) \Big]$$

② Then we did :-

oT_dd, oT_dr, oT_rd, oT_D, etaV = BlockRotateOmegaTaylor (self.omega_Taylor_rotate)

Here, we use advanced indexing in the __getitem__ class to extract the requisite blocks :-

```
elif BlockRotateOmegaTaylor(omega_Taylor_rotate):
    oT_dd = omega_Taylor_rotate[0:ND, 0:ND].copy()
    oT_dr =          "          [0:ND, ND:]     "
    oT_rd =          "          [ND:, 0:ND]     "
    oT_rr =          "          [ND:, ND:]      "
```

Once we have extracted these blocks, we now reduce them with :-

```
for t in [oT_dd, oT_dr, oT_rd, oT_rr] : t.reduce
```

Now, we take a detour into the reduce function :-