Class Taylor3D

- class that stores a Taylor expansion of a function in 3d and defines some arithmetic related to it.

* ⇒ Reminder of static methods

Example :-

```
class Person(object):
    def __init__(self, name, age):
        self.name = name
        self.age = age

    @classmethod
    def frombirthyear(cls, name, year):
        return cls(name, date.today().year - year)

    @staticmethod
    def isAdult(age):
        return age > 18
```

Usage :-

```
person1 = Person("Mayank", 21)
person2 = Person.frombirthyear("Mayank", 1992)
```

↳ The class method is accessible only by a class and not its instance

→ `print(Person.isAdult(22))`

↳ a staticmethod is like a utility function

→ not used to instantiate object but add functionality to the class.

→ Property of a class and not its instance.

TODO - ① What happens if we try to access a class / static method using an instance?

② What happens if we don't use the necessary decorators?

```
@staticmethod
def makeindexPowerYlm(Lmax):
    Analyzes the spherical harmonics and powers for a given Lmax.
```

Definition of spherical harmonics :-

→ A harmonic is a function that satisfies Laplace's equation :-

$$\nabla^2 f = 0.$$

→ Spherical harmonics are an infinite set of harmonic functions defined on the surface of a sphere.

→ We write ∇^2 in spherical co-ordinates :-

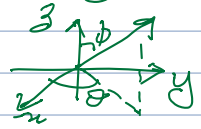
$$\nabla^2 = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial}{\partial r}\right) + \frac{1}{r^2\sin^2\phi}\frac{\partial^2}{\partial\theta^2} + \frac{1}{r^2\sin\phi}\frac{\partial}{\partial\phi}\left(\sin\phi\frac{\partial}{\partial\phi}\right)$$

and we write $f(\vec{r}) = f(r,\theta,\phi)$

$= R(r)\,\theta(\theta)\,\phi(\phi)$

$\phi \rightarrow$ polar angle

$\theta \rightarrow$ azimuthal angle



then we get

$$\frac{\theta(\theta)\phi(\phi)}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial}{\partial r}(R(r)\right) + \frac{1}{r^2\sin^2\phi}R(r)\phi(\phi)\frac{\partial^2}{\partial\theta^2}\theta(\theta)$$

$$+\frac{1}{r^2\sin\phi}R(r)\theta(\theta)\frac{\partial}{\partial\phi}\left(\sin\phi\frac{\partial}{\partial\phi}\phi(\phi)\right)$$

$$= 0$$

dividing throughout by $R(r)\theta(\theta)\phi(\phi)$:-

$$\frac{1}{r^2R(r)}\frac{\partial}{\partial r}\left(r^2\frac{\partial R(r)}{\partial r}\right) + \frac{1}{r^2\sin^2\phi\,\theta(\theta)}\frac{\partial^2}{\partial\theta^2}\theta(\theta)$$

$$+\frac{1}{r^2\sin\phi\,\phi(\phi)}\frac{\partial}{\partial\phi}\left(\sin\phi\frac{\partial\phi(\phi)}{\partial\phi}\right) = 0$$

$$= \frac{1}{r^2R(r)}\frac{\partial}{\partial r}R(r)\cdot 2r + \frac{1}{r^2R(r)}r^2\frac{\partial^2R(r)}{\partial r^2} + \frac{1}{r^2\sin^2\phi\,\theta(\theta)}\frac{\partial^2\theta(\theta)}{\partial\theta^2}$$

$$+\frac{1}{r^2\sin\phi\,\phi(\phi)}\sin\phi\frac{\partial^2\phi(\phi)}{d\phi^2} + \frac{1}{r^2\sin\phi\cdot\phi(\phi)}\cdot\frac{\partial\phi(\phi)}{\partial\phi}\cos\phi = 0$$

$$\Rightarrow \frac{1}{R(r)}\left[\frac{2}{r}\frac{\partial}{\partial r}R(r) + \frac{\partial^2R(r)}{\partial r^2}\right] + \frac{1}{r^2\sin^2\phi}\left[\frac{1}{\theta(\theta)}\frac{\partial^2\theta(\theta)}{\partial\theta^2}\right.$$

$$\left.+\frac{\sin^2\phi}{\phi(\phi)}\frac{\partial^2\phi(\phi)}{\partial\phi^2} + \frac{\cos\phi\sin\phi}{\phi(\phi)}\frac{\partial\phi(\phi)}{\partial\phi}\right]$$

$$= 0$$

$$\Rightarrow \frac{1}{R(r)} \frac{2r^2\sin^2\phi}{\alpha} \frac{\partial}{\partial r} R(r) + \frac{r^2\sin^2\phi}{R(r)} \frac{\partial^2 R(r)}{\partial r^2}$$

$$+ \underbrace{\frac{1}{\Theta(\theta)} \frac{\partial^2 \Theta(\theta)}{\partial \theta^2}}_{\uparrow} + \frac{\sin^2\phi}{\Phi(\phi)} \frac{\partial^2 \Phi(\phi)}{\partial \phi^2} + \frac{\cos\phi\sin\phi}{\Phi(\phi)} \frac{\partial \Phi(\phi)}{\partial \phi} = 0 \quad - \text{(I)}$$

The second term is only θ-dependent while the other two terms are not θ-dependent, so we must have :-

$$\frac{1}{\Theta(\theta)} \frac{\partial^2 \Theta(\theta)}{\partial \theta^2} = -m^2 \quad \text{where } m \text{ is some constant} \quad - \text{(II)}$$

which gives

$$\frac{\partial^2 \Theta(\theta)}{\partial \theta} = -m^2 \Theta(\theta)$$

$$\Rightarrow \Theta(\theta) = A_m e^{im\theta} \quad \text{where } m = -\infty, \dots, \infty$$

This means for the remaining two terms in (I), we must have

$$\frac{r^2}{R(r)} \frac{\partial^2 R(r)}{\partial r^2} + \frac{2r}{R(r)} \frac{\partial}{\partial r} R(r) + \frac{1}{\sin^2\phi} \left[\frac{\sin^2\phi}{\Phi(\phi)} \frac{\partial^2 \Phi(\phi)}{\partial \phi^2} + \frac{\cos\phi\sin\phi}{\Phi(\phi)} \frac{\partial \Phi(\phi)}{\partial \phi} - m^2 \right]$$

$$= 0$$

(III)

→ The radial and the angular part must be the negatives of the same constant, which we write as :-

$$\frac{r^2}{R(r)} \frac{\partial^2 R(r)}{\partial r^2} + \frac{2r}{R(r)} \frac{\partial R(r)}{\partial r} = l(l+1) \quad \text{(IV)} \qquad l > 0$$

$$\Rightarrow \frac{\partial^2 R(r)}{\partial r^2} + 2r \frac{\partial R(r)}{\partial r} = l(l+1) R(r) \quad \text{(V)}$$

This is the Cauchy-Euler equation, which can be solved through a power series of the form :-

$$R(r) = \sum_{n=0}^{\infty} a_n r^{n+c} \quad - \text{(VI)}$$

putting (VI) into (V), we get :-

$$\sum_{n=0}^{\infty} \left[(n+c)(n+c+1) - l(l+1) \right] a_n r^{n+c} = 0$$

This must hold for all powers of r
for the $n=0$ term, we have

$$a_0 r^c \left[c(c+1) - l(l+1) \right] = 0$$

which for non-zero a_0 gives

$$\underline{c(c+1) - l(l+1) = 0}$$

This is true only if $c = l, (-l-1)$
but if we use $c = l$ or $(-l-1)$, then we see that
for all other n, $a_n = 0$. $n \neq 0$.

Thus at $n = 0$, we have two possible solutions for $R(r)$

$$R(r) = a_0 r^l$$

$$R(r) = a_0 r^{-l-1}$$

Thus, the general solution is :-

$$R(r) = A_l r^l + \frac{B}{-l-1} r^{-l-1}.$$

Again, (I) gives form (II) :-

$$l(l+1) - \frac{m^2}{\sin^2\phi} + \frac{\cos\phi}{\sin\phi} \cdot \frac{1}{\phi(\phi)} \frac{\partial \phi(\phi)}{\partial \phi} + \frac{1}{\phi(\phi)} \frac{\partial^2 \phi(\phi)}{\partial \phi} = 0$$

$$\Rightarrow \phi'' + \frac{\cos\phi}{\sin\phi} \phi' + \left[l(l+1) - \frac{m^2}{\sin^2\phi} \right] \phi = 0$$

This is called the associated Legendre differential equations, for which the solutions are of the form :-

$$P_l^m(\cos\phi) \quad \text{where } m = 0, 1, 2, \ldots, l.$$

The general solution of $f(\vec{r})$ then becomes :-

$$f(\vec{r}) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \left(A_l r^l + B_{-l-1} r^{-l-1} \right) P_l^m(\cos\phi)\, e^{im\theta}$$

$$\Rightarrow f(r, \theta, \phi) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \left(A_l r^l + B_{-l-1} r^{-l-1} \right) Y_l^m(\theta, \phi)$$

where $Y_l^m(\theta, \phi) = P_l^m(\cos\phi)\, e^{im\theta}$ $\quad \phi \in [0, \pi]$, $\theta \in [0, 2\pi]$

are the spherical harmonics.

$P_l^m(\cos\phi)$ are the associated Legendre polynomials
↦ They have a well-defined form → no need to memorize now.

So, we have :-

$$Y_l^m(\theta, \phi) = P_l^m(\cos\phi)\, e^{im\theta}$$

$$l \in [0, \infty]$$

$$-l \leq m \leq l$$

for every allowed value of l there are $(2l+1)$ allowed values of m.

D.R. Trinkle, 2008, Phys. Rev. B :-

→ The normalized spherical harmonics are given by :-

$$Y_l^m(\theta, \phi) = e^{im\theta} P_l^m(\cos\phi) \sqrt{\frac{2l+1}{4\pi} \frac{(l-m)!}{(l+m)!}}$$

→ Reverse your notations to match with that of the paper. According to the paper

$$Y_l^m(\theta, \phi) = e^{im\phi} P_l^m(\cos\theta) \sqrt{\frac{2l+1}{4\pi} \frac{(l-m)!}{(l+m)!}}$$

$\phi \in [0, 2\pi]$
$\theta \in [0, \pi]$
az. angle
polar angle.

→ Consult GM meeting notebook of 24th Oct, 2018, for the detailed explanation of the math and for understanding the makePowYlm function. Also, see the Jupyter notebook where this function was been dissected.

Functions use in GFcalc → function TaylorExpandJumps

→ understanding powerup :-

⇒ we have $l_{max} = 4$

→ Say we enter a vector $u = [u_x, u_y, u_z]$

⇒ We want to understand what upow stores :-

upow

$$\begin{bmatrix} [u_x^0, u_y^0, u_z^0] \\ [u_x^1, u_y^0, u_z^0], \\ [u_x^0, u_y^1, u_z^0], \\ \vdots \end{bmatrix}$$

ind2pow

$$\begin{bmatrix} [0,0,0], \\ [1,0,0], \\ [0,1,0], \\ [0,0,1], \\ [2,0,0], \\ [0,2,0], \\ [0,0,2], \\ [1,1,0], \\ [0,1,1], \\ [1,0,1], \\ \vdots \end{bmatrix}$$

Functions and quantities defined in --initTaylor3Dindexing-- (cls, Lmax):

cls.Lmax = Lmax (the maximum power of q we want to Taylor expand to)

cls.NYlm, cls.Npower, cls.pow2ind, cls.ind2pow, cls.Ylm2ind, cls.ind2Ylm, cls.powLrange = makeindexPowerYlm(Lmax)

cls.Ylmpow = cls.makeYlmpow() → see the note on STF expansions

cls.powYlm = cls.makepowYlm()

cls.Lproj = cls.makeLprojections()

cls.directmult = cls.makedirectmult()

cls.powercoeff = cls.makepowercoeff.

Already there in the Jupyter notebook 'initindexing quantities' - look it up.

→ See the Jupyter notebook on Lproj

See the Jupyter Notebook

(I) NYlm, Npower, pow2ind, ind2pow, Ylm2ind, ind2Ylm, powLrange.

(1) → NYlm = (Lmax+1) ** 2

↦ these are the no. of spherical harmonics basis functions $Y_m^l(\vec{q})$ that we will need, in order to represent homogeneous polynomials upto order Lmax.

For example, if we have Lmax = 4, then, we'll need $Y_m^l(\vec{q})$ for

$l = 0, 1, 2, 3, 4$

for $l = 0$, $m = 0 \to 1$ spherical harmonic function $Y_0^0(\vec{q})$

$l = 1$, $m = -1, 0, 1 \to 3$ " " " Y_{-1}^1, Y_0^1, Y_1^1

$l = 2$, $m = -2, -1, 0, 1, 2 \to 5$ " " " $Y_{-2}^2, Y_{-1}^2, Y_0^2, Y_1^2, Y_2^2$

. . . .

for any given l, we need $(2l+1)$ spherical harmonic functions.

→ Now, this is because, if we have a homogeneous polynomial of order l, we can represent that polynomial in the basis of spherical harmonics Y_m^l (with the basis being the functions:-

$$\left(Y_{-l}^l, Y_{-l+1}^l, \cdots, Y_0^l \cdots Y_{l-1}^l, Y_l^l\right)$$

→ What we would like to do is express the homogeneous polynomials that arise out of the expansions of $(\vec{q}\cdot\vec{\delta x})^l$, in the spherical harmonics basis.

→ Now, when we have a defined Lmax, then in total we'll need

$$\sum_{l=0}^{Lmax} (2l+1)$$ spherical harmonics basis functions.

$$= 2\sum_{l=0}^{L_{max}} l + \sum_{l=0}^{L_{max}} 1$$

$$= \cancel{2}\cdot \frac{L_{max}(L_{max}+1)}{\cancel{2}} + L_{max} + 1$$

$$= L_{max}^2 + 2L_{max} + 1 = (L_{max}+1)^2$$

number of SH basis functions. This is what is stored in the NYlm variable.

(2) → Npower → these are the total no. of combinations of (l_1, l_2, l_3) in $(q_x dx_x)^{l_1} (q_y dx_y)^{l_2} (q_z dx_z)^{l_3}$ such that $l_1 + l_2 + l_3 = l$ for $l = 0, 1, \ldots, L_{max}$

This means for a given L_{max}, we'll have

Npower = NYlm + ((Lmax+1) Lmax (2Lmax−1))/6 no of total such combinations.

For example: say $L_{max} = 2$

	l_1	l_2	l_3	NYlm +=	Npower +=
$l = 0$	0	0	0	1	1
$l = 1$	1	0	0		3
	0	1	0	3	
	0	0	1		
$l = 2$	2	0	0	5	6
	0	2	0		
	0	0	2		
	1	1	0		
	0	1	1		
	1	0	1		

Npower = 10

$$= 9 + ((2+1) \times 2 \times 1)/6$$

$$= 9 + \frac{6}{6} = 10$$

(3) pow2ind and ind2pow and powlrange.

These three quantities are constructed with the following loop:

```
ind2pow = np.zeros ((Npower, 3), dt = int)
pow2ind = -np.ones ((Lmax+1, Lmax+1, Lmax+1), dt = int)

powlrange = np.zeros ( Lmax+2 , dt= int)

powlrange [-1] = 0
```

```
ind = 0
for l in range (lmax + 1):
    for n1 in range (l + 1):
        for n2 in range (l+1-n1):
            n3 = l - n1 - n2
            pow2ind [n1, n2, n3] = ind
            ind2pow [ind, 0],
            ind2pow [ind, 1], ind2pow [ind, 2] = n1, n2, n3
            ind += 1
    powrange [l] = ind
```

→ for l in range (lmax + 1)
for n1, n2 n3 in ([n1, n2, n3] for n1 in range (l+1) for n2 in range l+1) for n3 in range (l+1) if n1+n2+n3 ≤ l)

note that we'll have a lot of # -1s left over. See the #notebook for how this array looks

Ylm2ind, ind2Ylm and powercoeff are also there in the notebook.

(3) Now comes the new part

```
cls.Ylmpow = cls.makeYlmpow()
```

for now just remember this :- → see the note on SH expansions and wikipedia for more details.

Ylmpow [ind, powind] gives the contribution of

x: ind2pow [powind] [0] y: ind2pow [powind] [1] z: ind2pow [powind] [2]

to $Y_m^l(\vec{r})$ where $\vec{r} = x\hat{i} + y\hat{j} + z\hat{k}$

where Ylm2ind [l, m] = ind. → look up a mathematical functions handbook to verify the coefficients.

→ cls.powYlm = makepowYlm (cls) → class method.

makepowYlm (cls):

→ This function constructs the expansion of the powers in terms of the spherical harmonics.

→ Uses recursion relationship instead of direct calculations.

→ Alternatively, this can be done using Gaussian Quadrature.

```
powYlm = zeros ((cls.Npower, cls.NYlm), dtype = complex)
Cp = zeros ((cls.Lmax, 2 * cls.Lmax - 1))
Cm = zeros ((cls.Lmax, 2 * cls.Lmax - 1))
Sp = zeros ((cls.Lmax, 2 * cls.Lmax - 1))
Sm = zeros ((cls.Lmax, 2 * cls.Lmax - 1))

for l, m in [(l, m) for l in range (cls.Lmax) for m in range (-l, l+1)]:
    Cp[l, m] = sqrt ((l-m+1)(l+m+1) / ((2l+1)(2l+3)))
    Sp[l, m] = 0.5 sqrt ((l+m+1)(l+m+2) / ((2l+1)(2l+3)))
```

```
if l > 0:
    Cm[l,m] = sqrt((l-m)(l-m-1)/((2l-1)(2l+1)))
    Sm[l,m] = 0.5*sqrt((l-m)(l-m-1)/((2l-1)(2l+1)))
```

→ l=0, taken care of

```
powYlm[cls.pow2ind[0,0,0], cls.Ylm2ind[0,0]] = np.sqrt(4*np.pi)
```

→ n0+n1+n2 = 0 → taken care of

```
for n0, n1, n2 in ((n0,n1,n2) for n0 in range(cls.lmax+1)
                              for n1 in range(cls.lmax+1)
                              for n2 in range(cls.lmax+1)
                              if 0 < n0+n1+n2 <= cls.lmax):
```

(ie, iterating over $n_0, n_1, n_2 : n_1+n_2+n_3 = l, \forall l \le lmax$)

```
    ind = cls.pow2ind[n0, n1, n2]
    lmax = n0+n1+n2
    if n2 > 0:                                   (the index of the previous (n0, n1, n2))
        indlow = cls.pow2ind[n0, n1, n2-1]
        for l,m in ((l,m) for l in range(lmax) for m in range(-l, l+1)):
            plm = powYlm[indlow, cls.Ylm2ind[l,m]]
            powYlm[ind, cls.Ylm2ind[l+1,m]] += Cp[l,m] * plm

            if l > 0 and -l < m < l:
                powYlm[ind, cls.Ylm2ind[l-1,m]] += Cm[l,m] * plm

    elif n1 > 0:
        indlow = cls.pow2ind[n0, n1-1, n2]
        for l,m in ((l,m) for l in range(lmax) for m in range(-l, l+1)):
            plm = powYlm[indlow, cls.Ylm2ind[l,m]]
            powYlm[ind, cls.Ylm2ind[l+1,m+1]] += 1j * Sp[l,m] * plm
            powYlm[ind, cls.Ylm2ind[l+1,m-1]] += 1j * Sp[l,-m] * plm

            if m < l-1:
                powYlm[ind, cls.Ylm2ind[l-1,m+1]] += 1j * Sm[l,m] * plm

            if m > -l+1:
                powYlm[ind, cls.Ylm2ind[l-1,m-1]] += 1j * Sm[l,-m] * plm

    elif n0 > 0:
        indlow = cls.pow2ind[n0-1, n1, n2]
        for l,m in ((l,m) for l in range(lmax) for m in range(-l, l+1)):
            plm = powYlm[indlow, cls.Ylm2ind[l,m]]
            powYlm[ind, cls.Ylm2ind[l+1,m+1]] += -Sp[l,m] * plm
            powYlm[ind, cls.Ylm2ind[l+1,m-1]] += Sp[l,-m] * plm

            if m < l-1:
                powYlm[ind, cls.Ylm2ind[l-1,m+1]] += Sm[l,m] * plm

            if m > -l+1:
                powYlm[ind, cls.Ylm2ind[l-1,m-1]] += Sm[l,-m] * plm
```

return powYlm

Okay First, in order to understand this, we need to understand how we can find the coefficients of the projection of a function onto a spherical harmonics basis.

$$f(\theta,\phi) = \sum_{l=0}^{L_{max}} \sum_{m=-l}^{l} C_m^l \, Y_m^l(\theta,\phi)$$

consider the following homogeneous polynomial in our calculation

$$f(\hat{q}) = \sum_{n_1+n_2+n_3=4} a_{n_1} \hat{q}_x^{n_1} \, a_{n_2} \hat{q}_y^{n_2} \, a_{n_3} q_3^{n_3}$$

$$\Rightarrow f(\hat{q}) = \sum_{n_1+n_2+n_3=4} C_{n_1 n_2 n_3} \, q_x^{n_1} \, q_y^{n_2} \, q_3^{n_3}$$

$$\Rightarrow f(\hat{q}_x, \hat{q}_y, \hat{q}_3) = \sum_{n_1+n_2+n_3=4} C_{n_1 n_2 n_3} \, q_x^{n_1} \, q_y^{n_2} \, q_3^{n_3}$$

where we have written $f(\theta,\phi) \equiv f(\hat{q}_x, \hat{q}_y, \hat{q}_3)$

where $\hat{q}_x, \hat{q}_y, \hat{q}_3$ are the components of the unit vector $\hat{q}$.

$$f(\hat{q}_x, \hat{q}_y, \hat{q}_3) = \sum_{n_1+n_2+n_3=4} C_{n_1 n_2 n_3} \, \hat{q}_x^{n_1} \, \hat{q}_y^{n_2} \, \hat{q}_3^{n_3} = f(\hat{q})$$

Now, we want to expand this in the spherical harmonics basis :-

$$f(\hat{q}) = \sum_{l=0}^{L_{max}} \sum_{m=-l}^{l} C_m^l \, Y_m^l(\hat{q})$$

In order to do this, we find the coefficients C_m^l.

In the code, this is broken down a step further as follows :-

First, we consider a sum of homogeneous polynomials of degrees upto L_{max} as :-

$$f(\hat{q}) = \sum_{l=0}^{L_{max}} q^l \sum_{n_1+n_2+n_3=l} C_{n_1 n_2 n_3}^{l} \, \hat{q}_x^{n_1} \, \hat{q}_y^{n_2} \, \hat{q}_3^{n_3}$$

We'll take each of the terms in the inner sum and express them in SH basis as follows :-

$$q_x^{n_1} \, q_y^{n_2} \, q_3^{n_3} = \sum_{l=0}^{L_{max}} \sum_{m=-l}^{l} C_m^l \, Y_m^l, \quad n_1+n_2+n_3 = l$$

and store them as powYlm [pow2ind [n_1, n_2, n_3], Ylm2ind [l, m]] = C_m^l

$\therefore$ powYlm [pow2ind $[n_1, n_2, n_3]$, Ylm2ind $[l, m]$] = $(l,m)^{th}$ coefficient C_{lm}^{l} in $q_x^{n_1} q_y^{n_2} q_z^{n_3} = \sum_{l=0}^{Lmax} \sum_{m=-l}^{l} C_{lm}^{l} Y_{lm}^{l}$, for $n_1+n_2+n_3 = l$.

This is done using a recursive algorithm in makepowYlm.

→ See the note on SH expansions to see how these work.

(2) cls.Lproj = cls.makeLprojections()

```
def makeLprojections(cls):
    Defines a series of projections for each l component - see
    Appendix A again.
    projL = np.zeros((cls.Lmax+2, cls.Npower, cls.Npower))
    projLYlm = zeros((cls.Lmax+2, cls.NYlm, cls.NYlm), dt=complex)
    for l,m in [(l,m) for l in range(Lmax+1) for m in range(-l, l+1)]:
        lm = cls.Ylm2ind[l,m]
        projLYlm[l, lm, lm] = 1
        projLYlm[-1, lm, lm] = 1
    for l in range(cls.Lmax+2):
        projL = tensordot(Ylmpow, tensordot(projLYlm[l], powYlm,
                                            axes=(0,1)),
                                  axes=(0,0)).real
    return projL
```

First, let us see what the projection matrix is:

we have $P^{n_1,n_2,n_3}_{n_1',n_2',n_3'} := \sum_{l=0}^{Lmax} \sum_{m=-l}^{l} C^{lm}_{n_1,n_2,n_3} E^{lm}_{n_1',n_2',n_3'}$

note that this involves repeated sums, so this can easily be done with tensordot.

First, given that Ylmpow [Ylm2ind $[l,m]$, pow2ind $[n_1,n_2,n_3]$] = $C^{lm}_{n_1,n_2,n_3}$

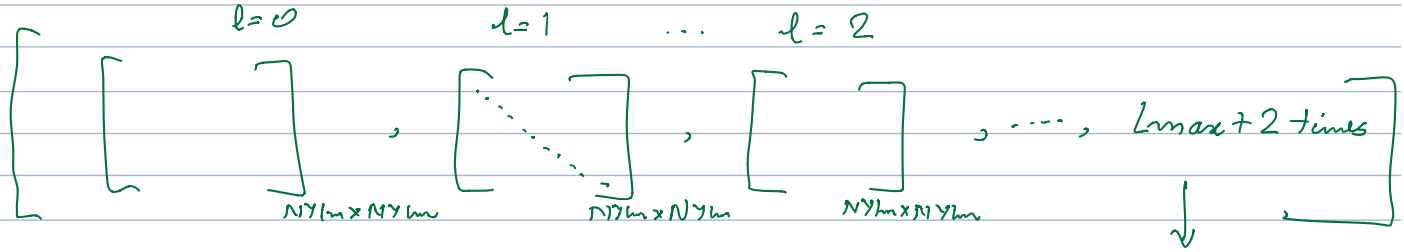
and powYlm [pow2ind $[n_1,n_2,n_3]$, Ylm2ind $[l,m]$] = $E^{lm}_{n_1,n_2,n_3}$

how would you proceed to construct the $P^{n_1,n_2,n_3}_{n_1',n_2',n_3'}$ matrix

↳ did it in Jupyter.

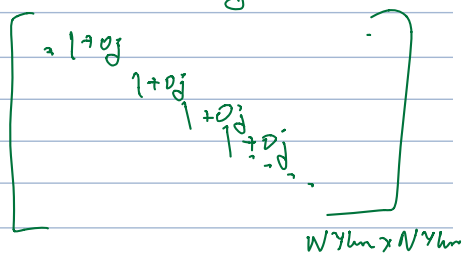
Now, let's see how Dallas has done it

First → what is projLYlm



for each given l, we'll have
Ylm2ind[l,m] for m = -l to l
only those diagonal elements will be 1 in the matrix, corresponding to these indices.
So for example, say l=2 and m=1
and Ylm2in [2,-1] = 20
Then
projLYlm [2, 20, 20] = 1+0j

the last one is special, because it will have all diag elements = 1+0j



So, this basically stores:

projLYlm [l, i, i] = 1+0j if Ylm2ind[l,m] = i for any m = -l to l

except for the last element, where all the diagonal elements are 0+0j

→ Okay, so now what is projL

for l in range (cls.Lmax + 2):

projL[l] = np.tensordot (cls.YlmPow,
np.tensordot(projLYlm[l], cls.powYlm, axes=(1,1)),
axes = (0,0)).real

First let us work out the inner tensordot.

we have projLYlm[l].shape = (25, 25)
and powYlm.shape = (35, 25)

tdot = tensordot (projLYlm[l], powYlm, axes = (1,1))
∴ tdot.shape = (25, 25)

→ Now let us work out this tensordot by hand

C = np.zeros((cls.Lmax+2, projLYlm[l].shape[0], cls.powYlm.shape[0]), dt=complex)

```
for l in range (cls.lmax+2):
    for i in range projL[Ylm[l].shape[0]:   ← we are eliminating the 1st axis
        for j in range powYlm.shape[0]:     ← " " " " " "
            for k in range [25]:            ← this is the length of the 1st axis in both
                C[i,j] += projL[Ylm[l][i,k] * powYlm[j,k]
```

in the jupyter notebook, this variable is named tdot_test.

↳ we sum up along the 1st axis.

Ylmpow[k, i] → contribution of i^{th} (n_0, n_1, n_2) combination to the expansion of the k^{th} (l,m) pair., ie, to the power expansion of Y_m^l.

$$= C^{lm(k)}_{n_0 n_1 n_2 (i)}$$

and C (in the jupyter notebook):-

tdot_test[l][k,j] = gives the contribution of the k^{th} (l,m) combination to the j^{th} (n_0, n_1, n_2) combination.

$$= E^{lm(k)}_{n_0 n_1 n_2 (j)}$$

∴ looking at the explicit form of the tensordot in the jupyter notebook, we can see that it gives us:-

$$\sum_k C^{lm(k)}_{n_0 n_1 n_2 (i)} E^{lm(k)}_{n_0 n_1 n_2 (j)}$$ for a given l in

projL[l] ≡ cls.Lproj[l]

$$\therefore \text{cls.Lproj}[l_1][i,j] = \sum_k C^{lm(k)}_{n_0 n_1 n_2 (i)} E^{lm(k)}_{(n_0 n_1 n_2) j} \delta(l, l_1)$$

This is the projection matrix that has been described in Appendix A of the 2017 paper.

(III) makedirectmult

cls.directmult = cls.makedirectmult()

def makedirectmult(cls):

→ returns directmult[p][p'] → the power index (in pow2ind) that arises when when ind2pow[p] (xyz) and ind2pow[p'] (xyz) are multiplied.

→ The code is fairly easy to understand → just go through it.

directmult [p0, p1] = pow2ind [ind2pow [p0] + ind2pow [p1]]

if the sum is less than Lmax.