lyuhang428@gmail.com

Notations

eDFT	electronic DFT
cDFT	classical DFT
$\psi(\vec{r})$, $f(\vec{r})$	wavefunction
$\phi(\vec{r})$	basis function
$U(\vec{r})$	Coulomb potential
$u(r)$	Coulomb potential spherical expansion coefficient
$\rho(\vec{r})$	electron density (eDFT) ， particle number density (cDFT)

Contents

1. Mathematical background

1.1. Gaussian type basis function

1.1.1. Gaussian product theorem

$$
\begin{aligned}
&g_1(\vec{r}) = e^{-\alpha\ |\vec{r}-\boldsymbol{R_A}|^2} \\
&g_2(\vec{r}) = e^{-\beta\ |\vec{r}-\boldsymbol{R_B}|^2} \\
&g_1 \cdot g_2 = e^{-\frac{\alpha\beta}{\alpha+\beta}\ |\boldsymbol{R_A}-\boldsymbol{R_B}|^2} e^{-(\alpha+\beta)\ |\vec{r}-\frac{\alpha\boldsymbol{R_A}+\beta\boldsymbol{R_B}}{\alpha+\beta}|^2}
\end{aligned}
\tag{1.1}
$$

Define some parameters:

$$
\begin{cases}
K = e^{-\frac{\alpha\beta}{\alpha+\beta}\ |\boldsymbol{R_A}-\boldsymbol{R_B}|^2} \\
p = \alpha+\beta \\
\boldsymbol{R_C} = \frac{\alpha\boldsymbol{R_A}+\beta\boldsymbol{R_B}}{\alpha+\beta}
\end{cases}
\tag{1.2}
$$

When Gaussian function does not center at the origin, the form of integrand needs to modifies a little bit:

$$
\int_R^\infty e^{-\alpha\ |\vec{\boldsymbol{r}}-\vec{\boldsymbol{R}}|^2}(r-R)^2 \sin\theta dr d\theta d\varphi \Leftrightarrow \int_0^\infty e^{-\alpha\vec{\boldsymbol{r}}^2} r^2 dr d\theta d\varphi \tag{1.3}
$$

The center of Gaussian function does not matter if this integral is computed in Cartesian coordinate, because all variables go from $-\infty$ to ∞, which is symmetric range. But in the spherical coordinate, the variable r goes from 0 to ∞, which is not symmetric to the function center and gives solution of Gaussian error function, which is wrong.

The change of function center is translational, which does not change the integral (are under the curve).

But the product of two Gaussian functions give an extra term:

$$
\begin{aligned}
&g_1 \cdot g_2 = e^{-\frac{\alpha\beta}{\alpha+\beta}\ |\boldsymbol{R_A}-\boldsymbol{R_B}|^2} e^{-(\alpha+\beta)\ |\boldsymbol{r}-\boldsymbol{R_C}|^2} \\
&\int g_1 \cdot g_2\ d^3\boldsymbol{r} = K_{AB}^{\alpha\beta} \int_{R_C}^\infty e^{-p_{\alpha\beta}\ |\boldsymbol{r}-\boldsymbol{R_C}|^2}(r-R_C)^2 dr \int d\Omega \\
&\Leftrightarrow K_{AB}^{\alpha\beta} \int_0^\infty e^{-pr^2} r^2 dr \int d\Omega
\end{aligned}
\tag{1.4}
$$

Namely, when using Gaussian quadrature to compute the integral in the spherical coordinate, we don't need to take care of the new function center created by Gaussian product theorem, as the new function center can always be seen as the new coordinate origin, and only the pre-factor needs to be computed explicitly $K_{AB}^{\alpha\beta}$, which depends on the exponents and centers of each Gaussian functions involved in the product.

1.1.2. Pure basis function and Cartesian basis function

Primitive function (Cartesian form): $(x-A_x)^{l_x}(y-A_y)^{l_y}(z-A_z)^{l_z} e^{-\alpha\ |\vec{\boldsymbol{r}}-\vec{\boldsymbol{R}}|^2}$

Contracted function (Cartesian form): $\sum_i^n c_i N_i (x-A_x)^{l_{x,i}}(y-A_y)^{l_{y,i}}(z-A_z)^{l_{z,i}} e^{-\alpha_i\ |\vec{\boldsymbol{r}}-\vec{\boldsymbol{R}}|^2}$

In general all primitive functions in a contraction share one set of angular momentum: $l_{\alpha,i} == l_{\alpha,j};\ \alpha = x, y, x$. But different primitive functions may use different angular momentums in a spherical contraction that is created via explicit linear combination. For example $d_{x^2-y^2}$:

$$
d_{x^2-y^2} = \frac{\sqrt{3}}{2}(d_{xx} - d_{yy}) \tag{1.5}
$$

where two sets of angular momentums exist in the spherical form of $d_{x^2-y^2}$ function. But in practical calculations linear combinations are not performed a prior. Calculations are done in the Cartesian basis functions (such as compute the function values on the grid), and converted to spherical basis function values via transformation matrix after.

Basis function (atomic orbital), is the product of angular part and radial part: $\phi(\boldsymbol{r}) = Y_l^m(\theta,\phi)R(r)$, where the angular part is spherical harmonics, and the radial part is exponential.

Spherical harmonics are the angular part solutions to the Laplace equation in the spherical coordinate via separation of variables, which is in general complex:

$$\begin{aligned}
&\nabla^2 f = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial f}{\partial r}\right) + \frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial f}{\partial\theta}\right) + \frac{1}{r^2\sin^2\theta}\frac{\partial^2 f}{\partial\phi^2} = 0\\
&f(\vec{\boldsymbol{r}}) = R(r)Y_l^m(\theta,\phi)\\
&\begin{cases} \frac{1}{R}\frac{d}{dr}\left(r^2\frac{dR}{dr}\right) = \lambda \\ \frac{1}{Y}\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial Y}{\partial\theta}\right) + \frac{1}{Y}\frac{1}{\sin^2\theta}\frac{\partial^2 Y}{\partial\phi^2} = -\lambda \end{cases}
\end{aligned} \tag{1.6}$$

The separation of variables in above equations hints us that for a function in real space $\rho(\vec{r})$, we can decompose it into radial part and angular part, which is actually spherical expansion, where spherical harmonics are the basis functions and the radial part is the expansion coefficient. This expansion is valid even when the radial part and angular part are coupled.

Angular momentum operator:

$$\begin{aligned}
&\begin{cases} \hat{\boldsymbol{L}}^2 = -\hbar^2\left(\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial}{\partial\theta}\right) + \frac{1}{\sin^2\theta}\frac{\partial^2}{\partial\phi^2}\right) \\ \hat{\boldsymbol{L}}^2 Y_l^m = \hbar^2 l(l+1)Y_l^m \end{cases}\\
&\begin{cases} \hat{\boldsymbol{L}}_{\boldsymbol{z}} = -i\hbar\frac{\partial}{\partial\phi} \\ \hat{\boldsymbol{L}}_{\boldsymbol{z}} Y_l^m = m\hbar Y_l^m,\ m = -l, -l+1, \cdots, l \end{cases}\\
&\nabla^2 = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial}{\partial r}\right) - \frac{1}{\hbar^2 r^2}\hat{\boldsymbol{L}}^{\boldsymbol{2}}
\end{aligned} \tag{1.7}$$

Analytical formular for spherical harmonics:

$$\begin{aligned}
&Y_l^m(\theta,\phi) = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}}P_l^m(\cos\theta)e^{im\phi}\\
&\int_0^{2\pi}\int_0^{\pi}\left(Y_{l'}^{m'}\right)^* Y_l^m \sin\theta d\theta d\phi = \delta_{ll'}\delta_{mm'}
\end{aligned} \tag{1.8}$$

But spherical harmonics (and the linear combined real spherical harmonics) are never used in both Cartesian and pure form basis functions, instead we use the mathematically equivalent polynomials in the Cartesian coordinate (including the spherical basis function). The corresponding relation between spherical variables and their Cartesian counterparts can be found here https://en.wikipedia.org/wiki/Table_of_spherical_harmonics.

Formally:

$$\begin{aligned}
&\begin{cases} \phi(r,\theta,\varphi) = Y_l^m(\theta,\varphi)\ |\boldsymbol{r}|^l \sum_i c_i n_i e^{-\alpha_i\ |\boldsymbol{r}|^2} \\ \phi(r,\theta,\varphi;\boldsymbol{R}) = Y_l^m(\theta,\varphi)\ |\boldsymbol{r}-\boldsymbol{R}|^l \sum_i c_i n_i e^{-\alpha_i\ |\boldsymbol{r}-\boldsymbol{R}|^2} \end{cases}\\
&\begin{cases} \phi(x,y,z) = x^{l_x}y^{l_y}z^{l_z} \sum_i c_i n_i e^{-\alpha_i(x^2+y^2+z^2)} \\ \phi(x,y,z;a_x,a_y,a_z) = (x-a_x)^{l_x}(y-a_y)^{l_y}(z-a_z)^{l_z} \sum_i c_i n_i e^{-\alpha_i\left[(x-a_x)^2+(y-a_y)^2+(z-a_z)^2\right]} \end{cases}
\end{aligned} \tag{1.9}$$

But in practical calculations the spherical form is not used.

In the pure basis function we take the norm of a vector $|\vec{r} - \vec{\boldsymbol{R}}|$, while in the Cartesian form we DO NOT take the absolute value $\times\ |x - a_x|$.

Pure basis function has no redundance for arbitrary angular momentum l. For example for $l = 2$ there are 5 d-orbitals, for $l = 3$ there are 7 f-orbitals, and so on and so forth. We use spherical harmonics to build up the angular part of pure form basis function, and therefore no redundance exists.

But for the Cartesian basis function things become different. The angular part is represented as polynomial $x^{l_x} y^{l_y} z^{l_z}$. When the angular momentum $l \geq 2$, Cartesian form basis functions have redundance, and the higher the angular momentum the more redundances Cartesian form basis functions will have:

l	Cartesian	Pure	
2	xx, xy, xz, yy, yz, zz	$xy, yz, z^2, xz, x^2 - y^2$	$6d \to 5d$
3	$xxx, xxy, xxz, xyy, xyz,$ xzz, yyy, yyz, yzz, zzz	$y(3x^2 - y^2), xyz, yz^2, z^3, xz^2,$ $z(x^2 - y^2), x(x^2 - 3y^2)$	$10f \to 7f$
$\vdots$			

The ordering of the angular momentum in Cartesian form basis functions is usually lexicographic, (for pure form basis functions typically two ordering conventions are used, vide infra): $\{xx, xy, xz, yy, yz, zz\}$, $\{xxx, xxy, xxz, xyy, xyz, xzz, yyy, yyz, yzz, zzz\}$.

```python
for lx in range(lmax, 0-1, -1):
    for ly in range(lmax-lx, 0-1, -1):
        lz = lmax - lx - ly
        sx = lx * 'x'
        sy = ly * 'y'
        sz = lz * 'z'
        print(f"({lx}, {ly}, {lz})    ({sx}{sy}{sz})")
```

1.1.3. About Normalization

Normalization factor for Cartesian form basis function: [3]

$$\begin{aligned}
&\phi(x, y, z; \alpha, l_x, l_y, l_z, \boldsymbol{R}) = N(x - R_x)^{l_x} (y - R_y)^{l_y} (z - R_z)^{l_z} \ \mathrm{Exp}\{-\alpha\ |\boldsymbol{r} - \boldsymbol{R}|^2\} \\
&N = N(\alpha, l_x, l_y, l_z) = \left(\frac{2\alpha}{\pi}\right)^{\frac{3}{4}} (4\alpha)^{\frac{l_x + l_y + l_z}{2}} \frac{1}{\sqrt{(2l_x - 1)!!\ (2l_y - 1)!!\ (2l_z - 1)!!}} \\
&(-1)!! = 1
\end{aligned} \tag{1.10}$$

The normalization factor of pure form basis function:

$$\begin{aligned}
\phi(r, \theta, \varphi) &= N Y_l^m r^l e^{-\alpha r^2} \\
N &= \sqrt{\frac{2(2\alpha)^{l + \frac{3}{2}}}{\Gamma(l + \frac{3}{2})}}
\end{aligned} \tag{1.11}$$

N.B. Basis functions are NOT orthogonal and NOT normalized.

Basis functions localize at different centers are not orthogonal, that's why the overlap matrix S_{ij} is not diagonal. But anti-intuitively yet mathematically allowed is that basis functions are not necessarily normalized. As long as they are linearly independent, they are valid basis functions in mathematics. Different software adopt different normalization conventions. Besides the coefficients hard-coded in basis set files, primitive functions can also have:

1. Each primitive function multiplies its own normalization constant(Equation (1.10)). Leading to all primitive functions are normalized (but normalization of contraction function is not guaranteed).
2. Only the axis-aligned primitive functions (i.e. xx, yy, xxx, …) are normalized, and all other primitives are multiplied to the same normalization constant $N = \left(\frac{2\alpha}{\pi}\right)(4\alpha)^{\frac{l_{\text{tot}}}{2}}\sqrt{\frac{1}{(2l_{\text{tot}}-1)!!}}$.
3. Force all basis functions to be normalized by computing the seld-overlapping integral.

The diagonal elements in overlap matrix are not guaranteed to be unity!

The simplest case is to ignore normalization at all and do calculations directly. But it brings advantages to normalize basis functions/primitive functions in advance. The first is numerical stability, avoiding the occurrence of extremely small/large values (`1e-49, 1e+107`, etc.. Condition number of overlap matrix is better); And the second advantage is better interpretability if basis functions are normalized, which means the atomic orbitals are also normalized and are physically more reasonable. But no matter which convention is used, just to make sure to use the same convention throughout, especially in cases that basis function values on the grid are needed (such as computing the exchange-correlation potential numerically).

Normalization does not affect physical observables (or expectation values), but is does affect the construction of operator matrices. Assume $\langle\phi_\mu \mid \phi_\mu\rangle \equiv 1$, then the un-normalized basis function is normalized basis function times scaling factor:

$$\begin{aligned}
&\phi'_\mu = a_\mu \phi_\mu \\
&S'_{\mu\nu} = \langle\phi'_\mu \mid \phi'_\mu\rangle = a_\mu a_\nu S_{\mu\nu} \\
&\text{let}\ \ C'_{\mu i} = \frac{C_{\mu i}}{a_\mu} \\
&\Rightarrow f_i = \sum_\mu C_{\mu i}\phi_\mu = \sum_\mu \frac{C_{\mu,i}}{a_\mu} a_\mu \phi_\mu = \sum_\mu C'_{\mu,i}\phi'_\mu
\end{aligned} \tag{1.12}$$

viz. the wavefunction is unaffected.

But operator matrices change, as their constructions depend on the choice of coordinate system and basis vectors.

$$\begin{aligned}
h'_{\mu\nu} &= a_\mu a_\nu h_{\mu\nu} \\
(\mu\nu|\lambda\sigma)' &= a_\mu a_\nu a_\lambda a_\sigma (\mu\nu|\lambda\sigma)
\end{aligned} \tag{1.13}$$

$$\begin{aligned}
&S'_{\mu\nu} = a_\mu a_\nu S_{\mu\nu} \\
&S' = \int \begin{pmatrix} a_1 & & & \\ & a_2 & & \\ & & \ddots & \\ & & & a_N \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_N \end{pmatrix} \begin{pmatrix} \phi_1 & \phi_2 & \cdots & \phi_N \end{pmatrix} \begin{pmatrix} a_1 & & & \\ & a_2 & & \\ & & \ddots & \\ & & & a_N \end{pmatrix} d^3\vec{\boldsymbol{r}} \\
&= ASA
\end{aligned} \tag{1.14}$$

$$\begin{aligned}
C'_{\mu i} &= \frac{C_{\mu i}}{a_\mu} \\
C' &= \begin{pmatrix} \frac{1}{a_1} & & & \\ & \frac{1}{a_2} & & \\ & & \ddots & \\ & & & \frac{1}{a_N} \end{pmatrix} \begin{pmatrix} C_{11} & C_{12} & \cdots & C_{1N} \\ C_{21} & C_{22} & \cdots & C_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ C_{N1} & C_{N2} & \cdots & C_{NN} \end{pmatrix} = A^{-1}C
\end{aligned} \tag{1.15}$$

Coefficient matrix changes accordingly.

Formalism of Kohn-Sham equation is unchanged, so is its eigenvalues:

$$\begin{aligned}
&F'C' = AFA \cdot A^{-1}C = AFC \equiv ASC\varepsilon \\
&S'C'\varepsilon = ASC\varepsilon \\
&\Rightarrow F'C' = S'C'\varepsilon
\end{aligned} \tag{1.16}$$

Density matrix after transformation becomes (n_i is the occupation number):

$$\begin{aligned}
&P_{\mu\nu} = \sum_i^{\text{occ}} n_i C_{\mu i} C_{\nu i} = a_\mu a_\nu \sum_i n_i C'_{\mu i} C'_{\nu i} = a_\mu a_\nu P'_{\mu\nu} \\
&P'_{\mu\nu} = \frac{1}{a_\mu} P_{\mu\nu} \frac{1}{a_\nu} \\
&\Rightarrow P' = A^{-1} P A^{-1}
\end{aligned} \tag{1.17}$$

Expectation values are invariant because of trace invariance:

$$\langle \hat{O} \rangle = \mathrm{Tr}\{P'O'\} = \mathrm{Tr}\{A^{-1}PA^{-1}AOA\} \Leftrightarrow \mathrm{Tr}\{OAA^{-1}P\} \equiv \mathrm{Tr}\{PO\} \tag{1.18}$$

The normalization of wavefunction is retained, which is the constraint of Lagrange multiplier:

$$\begin{aligned}
&\langle f_i \mid f_i \rangle = \sum_\mu \sum_\nu \langle C_{\mu i}\phi_\mu \mid C_{\nu i}\phi_\nu \rangle \\
&= \sum_{\mu\nu} C_{\mu i} C_{\nu i} S_{\mu\nu} \equiv 1 \\
&\Leftrightarrow C^T S C \equiv I \\
&(C')^T S' C' = (A^{-1}C)^T ASAA^{-1}C = C^T S C \equiv I
\end{aligned} \tag{1.19}$$

1.2. Gauss quadrature

N.B This section is NOT strict mathematical definition, theorem, axiom, etc.

Numerical integration involves taking the weighted average of the function values of a function $f(x)$ at discrete points within the interval $[a, b]$ as an approximation to the integral (based on the fundamental idea of the mean value theorem for integrals):

$$\begin{aligned}
&a \le x_0 \le x_1 \le \cdots \le x_n \le b \\
&\int_a^b f(x)dx \approx \sum_{i=0}^n A_i f(x_i)
\end{aligned} \tag{1.20}$$

where $\{A_i\}$ are coefficients (weights), and $\{x_i\}$ are quadrature roots.

The simplest trapezoidal method employs equally spaced nodes with equal weights at each point, which needs a significant number of discrete points to achieve satisfactory numerical accuracy. To attain higher accuracy with fewer nodes, the selection of nodes and the determination of weights are crucial.

If there exists $x_i \in [a, b]$ and coefficients $\{A_i\}$ such that $\int_a^b f(x)dx \approx \sum_{\{i=0\}}^n A_i f_i$ has algebraic precision of degree $2n + 1$, then the nodes $\{x_i\}$ are called a Gaussian points, A_i is called a Gaussian coefficient, and the quadrature formula is called a Gaussian-type quadrature formula.

Typically the Gaussian points are the zeros of various orthogonal polynomials in their respective orthogonal intervals. The coefficients (weights) have defined formulae to compute, and the weighting functions of various orthogonal polynomials need to be considered when performing numerical quadrature:

$$\int_a^b f(x)\omega(x)dx \approx \sum_{i=1}^{n} f(x_i)w_i \tag{1.21}$$

1.2.1. Orthogonal polynomials

If the weighted inner product of two arbitrary polynomials in a family of polynomial is zero, this family of polynomial is orthogonal polynomial:

$$\langle p_i \cdot p_j \rangle = \int_a^b \omega(x)p_i(x)p_j(x)dx = N_{ij}\delta_{ij} \tag{1.22}$$

Trigonometric functions $\{1, \cos x, \sin x, \cos 2x, \sin 2x, \cdots\}$ form orthogonal functions in the interval of $[-\pi, \pi]$ (not polynomials).

- If $\{\phi_k\}_{k=0}^{\infty}$ are weighted orthogonal in the interval of $[a, b]$, and H_n is the linear space consisting of all polynomials of degree not exceeding n, then $\{\phi_i \mid i = 0, 1, \cdots, n\}$ constitute a basis for H_n. And for all $p(x) \in H_n$, $p(x) = \sum_{j=0}^{n} c_j\phi_j$.
- Orthogonal polynomial satisfies three-term recurrence relation:

$$\begin{aligned}
&\phi_{-1} = 0\\
&\phi_0 = 1\\
&\phi_{n+1} = (x-\alpha_n)\phi_n - \beta_n\phi_{n-1},\ n = 0, 1, 2, \cdots\\
&\alpha_n = \frac{(x\phi_n, \phi_n)}{(\phi_n, \phi_n)}\\
&\beta_n = \frac{(\phi_n, \phi_n)}{(\phi_{n-1}, \phi_{n-1})}
\end{aligned} \tag{1.23}$$

Three-term recurrence relation is an efficient way of computing orthogonal polynomial.

- If $\{\phi_n\}_0^{\infty}$ are weighted orthogonal in the interval $[a, b]$, then ϕ_n has n zeros in this interval.

1.2.2. Some commonly used orthogonal polynomials

- Legendre quadrature

Integral interval $[-1, 1]$, weighting function $\omega(x) = 1$, $P_0 = 1$, $P_1 = x$, $P_n = \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2-1)^n$.

1. $$\int_{-1}^{1} P_nP_m dx = \begin{cases} 0, & m \neq n\\ \frac{2}{2n+1}, & m = n \end{cases} \tag{1.24}$$
2. $$P_n(-x) = (-1)^n P_n(x) \tag{1.25}$$
3. $$(n+1)P_{n+1} = (2n+1)xP_n - nP_{n-1},\ n = 1, 2, \cdots \tag{1.26}$$
4. $P_n(x)$ has n different zeros in the interval $[-1, 1]$.

Quadrature formula: $\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} f(x_i)w_i$

- Laguerre quadrature

Integral interval $[0, \infty)$, weighting function $\omega(x) = e^{-x}$, $L_n = e^x\frac{d^n}{dx^n}(x^ne^{-x})$.

1. $$\int_0^{\infty} e^{-x}L_nL_m dx = \begin{cases} 0, & m \neq n\\ (n!)^2, & m = n \end{cases} \tag{1.27}$$

2.
$$\begin{aligned} &L_0 = 1 \\ &L_1 = 1 - x \\ &L_{n+1} = (1 + 2n - x)L_n - n^2 L_{n-1},\ n = 1, 2, \cdots \end{aligned} \tag{1.28}$$

Quadrature formula: $\int_0^\infty f(x)e^{-x}dx \approx \sum_{i=1}^n f(x_i)w_i$

- Hermite quadrature

Interval $(-\infty, \infty)$, weighting function $\omega(x) = e^{-x^2}$, $H_n = (-1)^n e^{x^2} \frac{d^n}{dx^n}\left(e^{-x^2}\right)$.

1.
$$\int_{-\infty}^{\infty} e^{-x^2} H_n H_m = \begin{cases} 0,\ m \neq n \\ 2^n n! \sqrt{\pi},\ m = n \end{cases} \tag{1.29}$$

2.
$$\begin{aligned} &H_0 = 1 \\ &H_1 = 2x \\ &H_{n+1} = 2xH_n - 2nH_{n-1},\ n = 1, 2, \cdots \end{aligned} \tag{1.30}$$

Quadrature formula: $\int_{-\infty}^{\infty} f(x)e^{-x^2}dx \approx \sum_{i=1}^n f(x_i)w_i$

- Chebyshev quadrature of the first kind

Interval $[-1, 1]$, weighting funtion $\omega(x) = \frac{1}{\sqrt{1-x^2}}$

1.
$$\begin{aligned} &T_n(x) = \cos(n \arccos x) \quad |x| \leq 1 \\ &T_0 = 1 \\ &T_1 = x \\ &T_{n+1} = 2xT_n - T_{n-1},\ n = 1, 2, \cdots \end{aligned} \tag{1.31}$$

2.
$$\int_{-1}^{1} \frac{T_m T_n}{\sqrt{1 - x^2}} dx = \begin{cases} 0,\ n \neq m \\ \frac{\pi}{2},\ n = m \neq 0 \\ \pi, n = m = 0 \end{cases} \tag{1.32}$$

3.
$$T_n(-x) = (-1)^n T_n(x) \tag{1.33}$$

4. Zeros of T_n

$$x_k = \cos\left(\frac{2k - 1}{2n}\pi\right)\ k = 1, 2, \cdots, n \tag{1.34}$$

Quadrature formula: $\int_{-1}^{1} \frac{f(x)}{\sqrt{1-x^2}}dx \approx \sum_{i=1}^n f(x_i)w_i$

- Chebyshev quadrature of the second kind

Interval $[-1, 1]$, weighting function $\omega(x) = \sqrt{1 - x^2}$

1.
$$\begin{aligned} &U_n = \frac{\sin[(n + 1) \arccos x]}{\sqrt{1 - x^2}} \\ &U_0 = 1 \\ &U_1 = 2x \\ &U_{n+1} = 2xU_n - U_{n-1},\ n = 1, 2, \cdots \end{aligned} \tag{1.35}$$

2.
$$\int_{-1}^{1} U_n U_m \sqrt{1 - x^2} = \begin{cases} 0,\ m \neq n \\ \frac{\pi}{2},\ m = n \end{cases} \tag{1.36}$$

Quadrature formula: $\int_{-1}^{1} f(x)\sqrt{1 - x^2}dx \approx \sum_{i=1}^n f(x_i)w_i$

Summary

Type	Interval	Weighting function
Legendre	$[-1,1]$	$\omega(x)=1$
Laguerre	$[0,\infty)$	$\omega(x)=e^{-x}$
Hermite	$(-\infty,\infty)$	$\omega(x)=e^{-x^2}$
Chebyshev I	$(-1,1)$	$\omega(x)=\frac{1}{\sqrt{1-x^2}}$
Chebyshev II	$[-1,1]$	$\omega(x)=\sqrt{1-x^2}$

Example

- $\int_{-1}^{1} 1+x+x^2+x^3+x^4+x^5dx$

```
f = lambda x: 1+x+x**2+x**3+x**4+x**5

# Gauss-Legendre quadraure
import numpy.polynomial.legendre as L
# order 5 needs only 3-order roots, as 2x3-1=5, but here we use 5 roots
res = L.leggauss(5) # tuple, res[0] is roots, res[1] is weights
quad = np.sum(f(res[0]) * res[1]) # 3.0666666666666673, exact value 46/15

# Gauss-ChebyshevII quadrature
n = 10
z = np.arange(1, n+1)
x = np.cos(z / (n+1) * np.pi)
w = np.pi / (n+1) * np.sin(z/(n+1) * np.pi)**2
quad = np.sum(f(x) / np.sqrt(1-x**2) * w) # big error!
```

When using Gauss-Chebyshev quadrature, it is important to note that if the target integral is $\int_{-1}^{1} f(x)\sqrt{1-x^2}dx$, then for a $2n-1$ degree function $f(x)$, only n nodes are required to exactly calculate the integral. However, for $\int f(x)dx$, using Chebyshev quadrature requires dividing by the weighting function first: $f(x) \to \frac{f(x)}{\sqrt{1-x^2}}$, where the target function $\frac{f(x)}{\sqrt{1-x^2}}$ is not a polynomial of finite degree! The integral result obtained with only a few nodes will have significant errors.

1.2.3. Change of variable

Many mathematical physical functions are absolute integrable functions defined in the infinite/semi-infinite intervals $f \in \mathbb{L}^2$. But usually Laguerre or Hermite quadrature is not used when computing this kind of functions.

Taking Hermite quadrature as an example, since its integration interval is the entire real domain, its zeros distribute in the interval $-\infty < x_i < \infty$. If a function can be written exactly as a polynomial function multiplied by a Gaussian function, then using Hermite quadrature is a very natural choice: $F(x)=\sum_j c_j x^j e^{-x^2}$. However, if it cannot be written in this form, the quadrature requires dividing by the Gaussian weighting function: $\int f(x)dx = \int \frac{f(x)}{e^{-x^2}} e^{-x^2} dx$. Since the zeros of Hermite polynomials are distributed throughout the entire real domain, the denominator e^{-x^2} may be quite small, leading to numerical instability in $\frac{f(x)}{e^{-x^2}}$. Additionally, when we need to "make up" the required quadrature formula by dividing by the Gaussian weighting function, the integral kernel $\frac{f(x)}{e^{-x^2}}$ is undoubtedly no longer a polynomial (an infinite series). Finite-term summation can only obtain approximate results, and Hermite quadrature has lost its superiority. Increasing the number of terms can improve accuracy, but it will encounter numerical stability issues.

But, it can still be used (and provide pretty good result). For example: $f(x)=\frac{1}{x^2+1}e^{-x^2}+(x-1)^3e^{-5(x+1)^2}$

```
import numpy.polynomial.hermite as H

# exact -43 sqrt(pi/5) / 5 + e pi Erfc[1] = -5.4736295302356037725
f = lambda x: (1/(x**2+1)) * np.exp(-x**2) + (x-1)**3 * np.exp(-5 * (x+1)**2)
res = H.hermgauss(128)
quad = np.sum(f(res[0]) / np.exp(-res[0]**2) * res[1]) # -5.47362953023580089962, AbsErr =
1e-13
# N.B. np.exp(-res[0]**2) min=1e-102
```

Another method for numerically calculating improper integrals is through the use of Legendre quadrature. One of the advantages of Legendre quadrature lies in its simplest weighting function, $\omega(x) = 1$. Additionally, since its integration interval is $[-1, 1]$ and its zeros are also confined within this range, numerically there will be no occurrence of illy small or large values. Similarly, due to the boundedness of its effective interval, variable transformation is required to map the bounded interval to a (pseudo-)unbounded interval.

For the improper integral of an absolute integrable function on a semi-infinite interval, numerical quadrature cannot handle infinite boundaries. However, considering that $f|_{x=\infty} = 0$, the calculation of the improper integral can be approximated as the calculation of a definite integral with a very large upper bound:

```
N[Integrate[Exp[-x^2], {x, 0, Infinity}] - Integrate[Exp[-x^2], {x, 0, 10^4}]];
Out[]= 0.
```

To map $(-1, 1)$ to $(0, \text{Big})$, define (one possible) mapping function:

$$\begin{aligned} & x \in (-1,1) \mapsto T(x) \in (0, \text{Big}) \\ & \text{left} = 1e-3 \\ & \text{right} = 1e+4 \\ & \alpha = \ln\left(\frac{\text{right}}{\text{left}}\right) \\ & T(x) = \text{left}\ \left(e^{\alpha\frac{x+1}{2}} - 1\right) \\ & T'(x) = \frac{\alpha}{2}\ \text{left}\ e^{\alpha\frac{x+1}{2}} \end{aligned} \tag{1.37}$$

and the calculation of integral $\int_0^\infty F(x)dx$ can be approximated as:

$$\begin{aligned} & \int_0^\infty F(x)dx \approx \int_0^{\text{Big}} F[T(x)]dT(x) \Leftrightarrow \int_{-1}^{1} F[T(x)]T'(x)dx \\ & \approx \sum_{i=1}^{n} F[T(x_i)]\underbrace{(T'(x_i))}_{\text{Jacobian}} w_i \end{aligned} \tag{1.38}$$

where $T'(x)$ is the resulting Jacobian determinant due to the change of variable (1D scalar).

Example

- Compute $\int_0^\infty \frac{1}{x^2+1}e^{-x^2} + (x-1)^3 e^{-5(x+1)^2}$ via Legendre quadrature + change of variable

```
left = 1e-3
right = 1e+4
alpha = np.log(right/left)
f = lambda x: (1/(x**2+1)) * np.exp(-x**2) + (x-1)**3 * np.exp(-5 * (x+1)**2) # integrand
T = lambda x: left * (np.exp(alpha * (x+1)/2) - 1.) # mapping function
jacobian = lambda x: alpha/2. * left * np.exp((x+1)/2 * alpha) # Jacobian det
```

```
res = L.leggauss(128)
x_mapped = T(res[0])
fnvals = f(x_mapped)
quad = np.sum(fnvals * res[1] * jacobian(res[0])) # 0.67116241937370002546, AbsErr = 1e-12
# exact 1/50 (36/e^2 + 25 e pi Erfc[1] - 43 sqrt{5 pi} Erfc[sqrt{5}]) =
0.67116241937195611168
```

Similarly, the same trick can be applied to Chebyshev quadrature of the second kind. Different from the Legendre quadrature, performing Chebyshev quadrature needs to think about the weighting function. But its quadrature roots and weights are very easy to compute, and the roots can be obtained from arithmetic sequence:

$$\begin{aligned} z &= 1, 2, \cdots, N \quad \text{equally spaced grid} \\ x_i &= \cos\left(\frac{z_i}{N+1}\pi\right) \in (1, -1) \quad \text{root} \\ w_i &= \frac{\pi}{N+1}\sin^2\left(\frac{z_i}{N+1}\pi\right) \quad \text{weight} \end{aligned} \tag{1.39}$$

Mapping function suggested by Becke: $x \mapsto r = r(x) = r_m \frac{1+x}{1-x} \in (\text{Big}, 0)$, where r_m is the element Bragg-Slater radius[1], [4]. This design on one hand makes Chebyshev quadrature possible to deal with integrals in semi-infinite interval, and on other hand it is dense near the nucleus and sparse away from it, just like how electron density behaves.

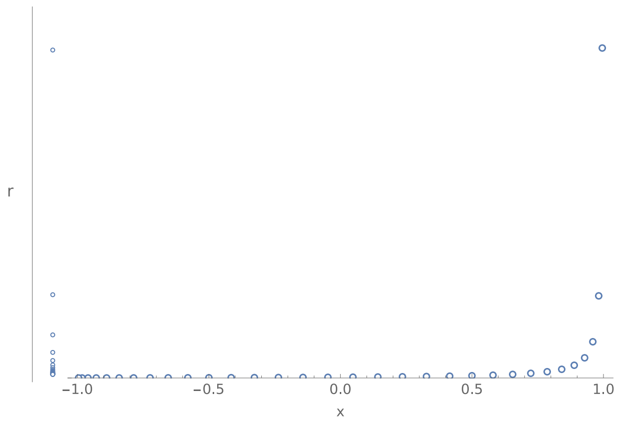


Figure 1: Becke mapping function.

$z \in [1, N]$, $x \in (1, -1)$, $r \in (\infty, 0)$. The final variable $\{r_i | i = 1, 2, \cdots, N\}$ is a function of the original variable $\{z_i\}$, so all functions with r as a variable are functions of the variable z through the chain rule. This brings a benefit: if the derivative of $f(r)$ with respect to r needs to be calculated through finite difference, the problem can be transformed into calculating the derivative of $f(r)$ with respect to z through the chain rule, while $\Delta z = 1$. If Legendre quadrature is used, the original variable z is not on a uniform grid, and finite difference is not easy to operate on non-uniform grids

Chebyshev quadrature requires division by the weighting function. Taking into account the Jacobian determinant brought about by change of variable, these factors can ultimately be incorporated into the weights:

$$
\begin{aligned}
&\frac{dr}{dx}|_{x_i} = 2r_m \frac{1}{(1-x_i)^2} \\
&\int_0^\infty f(r) r^2 dr \approx \sum_{i=1}^{N} \frac{f[r(x_i)] r_i^2}{\sqrt{1-x_i^2}} \frac{\pi}{N+1} \sin^2\left(\frac{z_i}{N+1}\pi\right) 2r_m \frac{1}{(1-x_i)^2} \\
&= \sum_{i=1}^{N} f(r_i) r_i^2 \frac{\pi}{N+1} \sin^2\left(\frac{z_i}{N+1}\pi\right) \frac{2r_m}{\sqrt{1-x_i^2}(1-x_i)^2} \\
&\Rightarrow w_i = \frac{\pi}{N+1} \sin^2\left(\frac{z_i}{N+1}\pi\right) \frac{2r_m}{\sqrt{1-x_i^2}(1-x_i)^2}
\end{aligned}
\tag{1.40}
$$

Note that the roots are arranged in a reverse order: $\{x_i\} \in (1, -1)$, mapped variables $\{r_i\} \in (\text{Big}, 0)$ are also in the reverse order. The weights $\{w_i\}$ need to correspond to this.

Example

- $\int_0^\infty \frac{1}{r^2+1} e^{-r^2} + (r-1)^3 e^{-5(r+1)^2} r^2 dr$. Compute the radial part integral for a 3D function (1D problem)

```
def gaussCheby2(norder:int=32, rm:float=1.) -> tuple[np.ndarray, np.ndarray, np.ndarray]:
    z = np.arange(1, norder+1)
    x = np.cos(z / (norder + 1) * np.pi) # z ∈ [1, norder] ↦ x ∈ (1, -1)
    r = rm * (1 + x) / (1 - x)           # x ∈ (1, -1) ↦ r ∈ (∞, 0)
    w = np.pi / (norder + 1) * np.sin(z / (norder + 1) * np.pi)**2 / np.sqrt(1 - x**2) * 2
* rm / (1 - x)**2
    return x, r, w

f = lambda x: (1/(x**2+1)) * np.exp(-x**2) + (x-1)**3 * np.exp(-5 * (x+1)**2)

x, r, w = gaussCheby2(128)
quad = np.sum(f(r) * r**2 * w) # 0.21457600129729723; AbsErr = 1e-14
# exact 1/250(277/e^5 - 125e pi Erfc[1] + sqrt{pi} (125 - 301 Erfc[sqrt{5}])) =
0.2145760012973228
```

1.3. Numeric quadrature on the sphere

1.3.1. Real spherical harmonics

Spherical harmonics are eigenfunctions of angular momentum operator ($\hat{L}^2$) and its z-component operator ($\hat{L}_z$):

$$
\begin{aligned}
\hat{L}^2 Y_l^m(\theta, \varphi) &= l(l+1)\hbar^2 Y_l^m(\theta, \varphi) \\
\hat{L}_z Y_l^m(\theta, \varphi) &= m\hbar Y_l^m(\theta, \varphi)
\end{aligned}
\tag{1.41}
$$

Spherical harmonics are usually complex:

$$
\begin{aligned}
Y_l^m(\theta, \phi) &= (-1)^m N_{lm} P_l^m(\cos\theta) e^{im\phi} \\
N_{lm} &= \sqrt{\frac{2l+1}{4\pi} \frac{(l-m)!}{(l+m)!}}
\end{aligned}
\tag{1.42}
$$

$(-1)^m$ is Condon-Shortley phase and P_l^m is associated-Legendre polynomial.

Spherical harmonics are orthonormal on unit sphere:

$$
\int_0^\pi \int_0^{2\pi} Y_l^m \, Y_{l'}^{m'} \sin\theta d\theta d\varphi = \delta_{ll'} \delta_{mm'} = \langle Y_l^m \mid Y_{l'}^{m'} \rangle_\Omega
\tag{1.43}
$$

For the convenience of practical calculations, and since the linear superposition of solutions to a differential equation remains a solution to the original equation (superposition principle), it is common in practice to perform linear combinations on spherical harmonics to eliminate the imaginary part, resulting in what are known as real spherical harmonics. Both complex and real spherical harmonics are typically functions of the angles (θ, φ): $Y_l^m(\Omega) = Y_l^m(\theta, \varphi)$. The angular variables can be transformed into Cartesian coordinates through inverse trigonometric functions, so $Y_l^m(\theta, \varphi) \equiv Y_l^m(x, y, z)$.

The construction of real spherical harmonics is not unique, but orthonormality must be fulfilled: $\int_0^{2\pi} \int_0^{\pi} \left(Y_{l'}^{m'}\right)^* Y_l^m \sin\theta d\theta d\phi = \delta_{ll'}\delta_{mm'}$.

Below are a few possible constructions:

1.
$$Y_l^m = \begin{cases} \frac{(-1)^m}{\sqrt{2}} \left\{ Y_l^m + (Y_l^m)^\dagger \right\} & m > 0 \\ Y_l^0 \quad m = 0 \\ \frac{(-1)^m}{i\sqrt{2}} \left\{ Y_l^{|m|} - \left(Y_l^{|m|}\right)^\dagger \right\} & m < 0 \end{cases} \tag{1.44}$$

2.
$$Y_l^m = \begin{cases} \sqrt{2}(-1)^m \mathfrak{I}\left[Y_l^{|m|}\right] & m < 0 \\ Y_l^0 \quad m = 0 \\ \sqrt{2}(-1)^m \mathfrak{R}[Y_l^m] & m > 0 \end{cases} \tag{1.45}$$

And possible implementations:

1.
```python
# Copied from PyDFT: [https://github.com/ifilot/pydft].
# no Condon-Shortley phase
def rsh(l, m, theta, phi):
    if m < 0:
        val = np.sqrt(2) * np.imag(sph_harm_y(l, np.abs(m), phi, theta)) # ifilot
swaps theta and phi
    elif m > 0:
        val = np.sqrt(2) * np.real(sph_harm_y(l, m, phi, theta))
    else:
        val = np.real(sph_harm_y(l, m, phi, theta))

    return val
```

2.
```python
# https://scipython.com/blog/visualizing-the-real-forms-of-the-spherical-harmonics/
# https://en.wikipedia.org/wiki/Spherical_harmonics#Real_form
def rsh(l:int, m:int, theta:float, phi:float) -> float:
    if abs(m) > l:
        raise ValueError('magnetic quantum number m should not be larger than
angular qunautm number l')

    phase = -1 if abs(m) % 2 else 1

    if m > 0:
        return phase * np.sqrt(2.) * sph_harm_y(l, m, theta, phi).real
    elif m < 0:
        return phase * np.sqrt(2.) * sph_harm_y(l, -m, theta, phi).imag
    elif m == 0:
        return sph_harm_y(l, 0, theta, phi).real # a+0j, forcefully remove null
imaginary part
```

3.
```
# https://docs.abinit.org/theory/spherical_harmonics/
def rsh2(l:int, m:int, theta:float, phi:float) -> float:
    if abs(m) > l:
        raise ValueError('magnetic quantum number m should not be larger than
angular qunautm number l')

    phase = 1 if (abs(m) % 2 == 0) else -1 # (-1)^m

    if m > 0:
        return phase / np.sqrt(2.) * (sph_harm_y(l, m, theta, phi) + sph_harm_y(l,
m, theta, phi).conjugate())
    elif m < 0:
        return phase / np.sqrt(2.) / 1j * (sph_harm_y(l, -m, theta, phi) -
sph_harm_y(l, -m, theta, phi).conjugate())
    elif m == 0:
        return sph_harm_y(l, 0, theta, phi)
```

4. C++ std library implemented Legendre polynomial, which is ready to use:

```
double rsh(const int l, const int m, const double theta, const double phi) noexcept
{
    assert((l >= 0) && (std::abs(m) <= l));
    const    int        mm = (m < 0 ? -m : m);
    const double     phase = (mm & 1) ? -1. : 1.;
    const double         P = std::sph_legendre(l, mm, theta);
    constexpr double root2 = 1.41421356237309514547462185873882845044l4;

    if (m == 0) return P;
    if (m > 0)  return phase * root2 * P * std::cos(mm * phi);
    return phase * root2 * P * std::sin(mm * phi);
}
```

1.3.2. Lebedev quadrature

Lebedev quadrature is the most commonly used angular quadrature scheme[5].

- Example[6]

$\int_\Omega f(x,y,z) d\Omega = \int_\Omega 1 + x + y^2 + x^2 y + x^4 + y^5 + x^2 y^2 z^2$

```
import numpy as np
from scipy.integrate import lebedev_rule

f = lambda x,y,z: 1 + x + y**2 + x**2 * y + x**4 + y**5 + x**2 * y**2 * z**2

res = lebedev_rule(17) # 110 angular pts
quad = np.sum(f(*res[0]) * res[1]) # 19.388114662154152
# exact 216pi/35 = 19.388114662154152
```

The quadrature result can be verified via Mathematica ($r = 1$ on unit sphere):

```
f[x_,y_,z_]:=1+x+y^2+x^2 y+x^4+y^5+x^2 y^2 z^2;
Integrate[(f[x,y,z]/.{x->Sin[\[Theta]] Cos[\[Phi]],y->Sin[\[Theta]] Sin[\[Phi]],z-
>Cos[\[Theta]]}) Sin[\[Theta]],{\[Theta],0,Pi},{\[Phi],0,2 Pi}]
Out[]=216Pi/35
```

1.4. Numeric solution to Poisson's equation

1.4.1. First order functional derivative

Analogous to the derivative of a function: $f'(x) = \frac{df(x)}{dx}$, $df = f'dx$. The first-order functional derivative is:

$$\begin{aligned} &F[f(x)] \\ &F'[f(x)] = \frac{\delta F[f(x)]}{\delta f(x)} \\ &\delta F = F[f(x) + \delta f(x)] - F[f(x)] = \underbrace{\int \left(\frac{\delta F}{\delta f}\right) \delta f dx}_{\text{contributions from all x}} + \mathcal{O}(\delta^2 f) \end{aligned} \tag{1.46}$$

Unlike the derivative of a function, there is no formula for the functional derivative. The derivative result depends on the specific form of the functional. The coefficient of δf in the integral is the functional derivative.

Example

- Compute the first-order functional derivative of $F[n(x)] = \int_0^1 n^2 dx$:

$$\begin{aligned} &\delta F = \int_0^1 (n + \delta n)^2 - n^2 dx \\ &\delta F = \int_0^1 2n\delta n dx \\ &\Rightarrow \frac{\delta F}{\delta n} = 2n \end{aligned} \tag{1.47}$$

Example

- First order derivative of integral-type functional:

Assume δn is fixed at the boundaries: $\delta n = 0$, therefore $\frac{\partial f}{\partial n'} \delta n \mid_{\text{boundary}} = 0$

$$
\begin{aligned}
&F[n]=\int f(n,n',x)dx\\
&F[n+\delta n]=\int f\left(n+\delta n,\frac{d}{dx}(n+\delta n),x\right)dx\\
&\text{Taylor expansion to the first term}\\
&\Rightarrow F[n+\delta n]=\int f(n,n',x)+\left\{\frac{\partial f}{\partial n}\delta n+\frac{\partial f}{\partial n'}(\delta n)'\right\}dx\\
&F[n+\delta n]-F[n]=\delta F=\int\frac{\partial f}{\partial n}\delta n+\frac{\partial f}{\partial n'}(\delta n)'\ dx\\
&\text{do integration by parts to the second term 对第二项进行分部积分}\\
&\Rightarrow\delta F=\int\frac{\partial f}{\partial n}\delta n\ dx+\int\frac{\partial f}{\partial n'}d(\delta n)=\int\frac{\partial f}{\partial n}\delta n\ dx+\frac{\partial f}{\partial n'}\delta n\ |_{\text{边界}}-\int\delta n\ d\left(\frac{\partial f}{\partial n'}\right)\\
&\Rightarrow\delta F=\int\frac{\partial f}{\partial n}\delta n\ dx-\int\delta n\ \frac{d}{dx}\left(\frac{\partial f}{\partial n'}\right)dx\\
&\delta F=\int\frac{\delta F[n]}{\delta n}\ \delta n\ dx=\int\left\{\frac{\partial f}{\partial n}-\frac{d}{dx}\left(\frac{\partial f}{\partial n'}\right)\right\}\delta n\ dx\\
&\therefore\frac{\delta F[n]}{\delta n}=\frac{\partial f}{\partial n}-\frac{d}{dx}\left(\frac{\partial f}{\partial n'}\right)
\end{aligned}
\tag{1.48}
$$

And this is Euler-Lagrange equation.

The functional derivative of Coulomb energy $E[n(\vec{\boldsymbol{r}})]_{\text{Coul}}$ w.r.t. electron density gives Coulomb potential $v[n(\vec{\boldsymbol{r}})]_{\text{Coul}}=\frac{\delta E[n]}{\delta n}$:

$$
\begin{aligned}
&E_{\text{Ha}}[n]=\frac{1}{2}\int\int\frac{n(\vec{\boldsymbol{r}})n(\vec{\boldsymbol{r}}')}{|\vec{\boldsymbol{r}}-\vec{\boldsymbol{r}}'|}d^3\vec{\boldsymbol{r}}d^3\vec{\boldsymbol{r}}'\\
&\delta E_{\text{Ha}}[n]=E_{\text{Ha}}[n+\delta n]-E_{\text{Ha}}[n]\\
&E_{\text{Ha}}[n+\delta n]=\frac{1}{2}\int\int\frac{\{n(\vec{\boldsymbol{r}})+\delta n(\vec{\boldsymbol{r}})\}\{n(\vec{\boldsymbol{r}}')+\delta n(\vec{\boldsymbol{r}}')\}}{|\vec{\boldsymbol{r}}-\vec{\boldsymbol{r}}'|}d^3\vec{\boldsymbol{r}}d^3\vec{\boldsymbol{r}}'\\
&\approx\frac{1}{2}\int\int\frac{n(\vec{\boldsymbol{r}})n(\vec{\boldsymbol{r}}')}{|\vec{\boldsymbol{r}}-\vec{\boldsymbol{r}}'|}+\underbrace{\left\{\frac{n(\vec{\boldsymbol{r}})\delta n(\vec{\boldsymbol{r}}')}{|\vec{\boldsymbol{r}}-\vec{\boldsymbol{r}}'|}+\frac{n(\vec{\boldsymbol{r}}')\delta n(\vec{\boldsymbol{r}})}{|\vec{\boldsymbol{r}}-\vec{\boldsymbol{r}}'|}\right\}}_{\vec{\boldsymbol{r}}\leftrightarrow\vec{\boldsymbol{r}}'}d^3\vec{\boldsymbol{r}}d^3\vec{\boldsymbol{r}}'\\
&=E_{\text{Ha}}[n]+\int\int\frac{n(\vec{\boldsymbol{r}}')\delta n(\vec{\boldsymbol{r}})}{|\vec{\boldsymbol{r}}-\vec{\boldsymbol{r}}'|}d^3\vec{\boldsymbol{r}}d^3\vec{\boldsymbol{r}}'\\
&\Rightarrow\delta E_{\text{Ha}}[n]=\int\int\frac{n(\vec{\boldsymbol{r}}')\delta n(\vec{\boldsymbol{r}})}{|\vec{\boldsymbol{r}}-\vec{\boldsymbol{r}}'|}d^3\vec{\boldsymbol{r}}d^3\vec{\boldsymbol{r}}'\\
&\Leftrightarrow\int\left\{\int\frac{n(\vec{\boldsymbol{r}}')}{|\vec{\boldsymbol{r}}-\vec{\boldsymbol{r}}'|}d^3\vec{\boldsymbol{r}}'\right\}\delta n(\vec{\boldsymbol{r}})d^3\vec{\boldsymbol{r}}\\
&\therefore\delta E_{\text{Ha}}[n(\vec{\boldsymbol{r}})]=\int\frac{\delta E_{\text{Ha}}[n(\vec{\boldsymbol{r}})]}{\delta n(\vec{\boldsymbol{r}})}\delta n(\vec{\boldsymbol{r}})d^3\vec{\boldsymbol{r}}\\
&\Rightarrow v_{\text{Ha}}(\vec{\boldsymbol{r}})=\frac{\delta E_{\text{Ha}}[n(\vec{\boldsymbol{r}})]}{\delta n(\vec{\boldsymbol{r}})}=\int\frac{n(\vec{\boldsymbol{r}}')}{|\vec{\boldsymbol{r}}-\vec{\boldsymbol{r}}'|}d^3\vec{\boldsymbol{r}}'
\end{aligned}
\tag{1.49}
$$

1.4.2. Poisson's equation

An important identity:

$$\nabla^2_{\vec{r}} \frac{1}{|\vec{r} - \vec{r}'|} = -4\pi\delta(\vec{r} - \vec{r}') \tag{1.50}$$

where $\frac{1}{|\vec{r}-\vec{r}'|}$ is Green's function. Subscript in $\nabla^2_{\vec{r}}$ indicates applying Laplacian to $\vec{r}$.

Equation (1.49) is the convolution of the Coulomb kernel function with the electron density:

$$\begin{aligned} &(f * g)(t) = \int f(\tau)g(-\tau + t)d\tau \\ &f = n(\vec{r}) \\ &g = \frac{1}{|\vec{r}|} \\ &\Rightarrow (f * g)(\vec{r}) = \int n\left(\vec{r}'\right)\frac{1}{|-\vec{r}' + \vec{r}|}d^3\vec{r}' = \int \frac{n(\vec{r}')}{|\vec{r} - \vec{r}'|}d^3\vec{r}' \end{aligned} \tag{1.51}$$

By plugging Equation (1.50) into Equation (1.49) we get:

$$\begin{aligned} &\nabla^2_{\vec{r}} v_{\text{Ha}}(\vec{r}) = \nabla^2 \int \frac{n\left(\vec{r}'\right)}{|\boldsymbol{r} - \boldsymbol{r}'|}d^3\vec{r}' = \int n\left(\vec{r}'\right)\nabla^2 \frac{1}{|\vec{r} - \vec{r}'|}d^3\vec{r}' \\ &= \int n\left(\vec{r}'\right) \cdot -4\pi\delta(\vec{r} - \vec{r}')d^3\vec{r}' = -4\pi n(\vec{r}) \end{aligned} \tag{1.52}$$

And we get the integral form and derivative form of Coulomb potential, which are mathematically equivalent:

$$\begin{cases} v_{\text{Ha}}(\vec{r}) = \int \frac{n(\vec{r}')}{|\vec{r}-\vec{r}'|}d^3\vec{r}' \\ \nabla^2 v_{\text{Ha}}(\vec{r}) = -4\pi n(\vec{r}) \end{cases} \tag{1.53}$$

The derivative form is the Poisson equation. Solving Poisson equation is to calculate the Coulomb potential at given electron density.

1.4.3. Spherical expansion

Since (real) spherical harmonics are orthonormal on the unit sphere ($\theta \in (0, \pi)$, $\phi \in (0, 2\pi)$, $r \equiv 1$), they constitute a complete basis set on the unit sphere. Any absolute integrable function on the unit sphere can be expressed as a linear combination of spherical harmonics:

$$f(\theta, \phi) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} a_{lm} Y_l^m(\theta, \varphi), \quad l \in \mathbb{N} \tag{1.54}$$

Basis function in the spherical coordinate is the product of spherical harmonics and Gaussian functions (N.B. primitive gaussian normalized, contracted gaussian normalization not guaranteed)：

$$\phi(r, \theta, \varphi) = Y_l^m(\theta, \varphi)\ |\vec{r} - \vec{\boldsymbol{R}}|^l \sum_i c_i n_i e^{-\alpha_i\ |\vec{r}-\vec{\boldsymbol{R}}|^2} \tag{1.55}$$

The spherical form is equivalet to the Cartesian form:

$$\phi(x, y, z) = (x - R_x)^{l_x}\left(y - R_y\right)^{l_y}(z - R_z)^{l_z} \sum_i c_i n_i e^{-\alpha_i\ |\vec{r}-\vec{\boldsymbol{R}}|^2} \tag{1.56}$$

The radial part and angular part are coupled and cannot be explicitly separated, but the radial part can be extracted via spherical expansion:

$$\begin{aligned}
&|\phi(r,\theta,\varphi)|^2 \equiv |\phi(x,y,z)|^2 = \sum_{lm} \rho_{lm}(r) Y_l^m(\theta,\varphi) \\
&\rho_{lm}(r) = \int Y_l^m(\theta,\varphi)^* \ |\phi(r,\theta,\varphi)|^2 \ d\Omega \\
&\rho_{lm}(r_i) \approx \sum_j^M |\phi(r_i,\Omega_j)|^2 \ Y_l^m(\Omega_j)^* w_j
\end{aligned} \tag{1.57}$$

Ω_j in Equation (1.57) is the sampling point on the unit sphere (built via Lebedev quadrature), and w_j is the quadrature weight.

The equation is valid only when the summation in Equation (1.57) goes to infinity; unless the function to be expanded is itself spherical harmonics or power of spherical harmonics, in which case the expansion is finite and the expansion coefficient can be determined by Clebsch-Gordan coefficient. In common cases, Becke suggested l to be half the Lebedev order. For example, `nleb=29 (302 angular pts)`, `lmax=14`, `nlm=225` [2].

1.4.4. Solving Poisson's equation via spherical expansion and finite difference

Do spherical expansion on electronic Coulomb potential $U(\vec{\boldsymbol{r}})$ and electron density $\rho(\vec{\boldsymbol{r}})$:

$$\begin{aligned}
U(\vec{\boldsymbol{r}}) &= U(r,\theta,\varphi) = \sum_{lm} \frac{u_{lm}(r)}{r} Y_l^m(\theta,\varphi) \\
\rho(\vec{\boldsymbol{r}}) &= \rho(r,\theta,\varphi) = \sum_{lm} \rho_{lm}(r) Y_l^m(\theta,\varphi)
\end{aligned} \tag{1.58}$$

plug in Equation (1.53)

$$\begin{aligned}
&\nabla^2 = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 \frac{\partial}{\partial r}\right) - \frac{L^2}{r^2} \\
&\nabla^2 \sum_{lm} \frac{u_{lm}}{r} Y_l^m = -4\pi \sum_{lm} \rho_{lm} Y_l^m \\
&\text{lhs} = \sum_{lm} \left\{ \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 \frac{\partial}{\partial r}\right) - \frac{L^2}{r^2} \right\} \frac{u}{r} Y \\
&= \sum_{lm} \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 \frac{\partial}{\partial r}\frac{u}{r}\right) Y - \frac{1}{r^2}\frac{u}{r} L^2 Y \\
&= \sum_{lm} \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 \frac{u' r - u}{r^2}\right) Y - \frac{u}{r^3} l(l+1) Y \\
&= \sum_{lm} \frac{1}{r^2}(u'' r + u' - u') Y - \frac{u}{r^3} l(l+1) Y \\
&\Rightarrow \sum_{lm} \frac{u''_{lm}}{r} Y_l^m - \frac{u_{lm}}{r^3} l(l+1) Y_l^m = \sum_{lm} -4\pi \rho_{lm}(r) Y_l^m \\
&\Rightarrow \frac{d^2}{dr^2} u_{lm} - \frac{l(l+1)}{r^2} u_{lm} = -4\pi r \rho_{lm}
\end{aligned} \tag{1.59}$$

For each pair of (l,m) we get one 2nd-order ODE:

$$\frac{d^2}{dr^2} u_{lm}(r) - \frac{l(l+1)}{r^2} u_{lm}(r) = -4\pi r \rho_{lm}(r) \tag{1.60}$$

The problem is simplified from solving one 2nd-order PDE to solving a set of 2nd-order ODE.

Uniform grid $\{z_i | i = 1, 2, \cdots, N\}$ is converted to the zeros of Chebyshev of the second kind: $\Big\{x_i = \cos\frac{z_i}{N+1}\pi \mid i = 1, 2, \cdots, N\Big\} \in (1, -1)$, which is then mapped to semi-infinite interval: $\Big\{r_i = r_m\frac{1+x_i}{1-x_i} \mid i = 1, 2, \cdots, N\Big\} \in (\mathrm{Big}, 0)$.

2nd-order derivative function needs at least two boundary conditions, which are determinied by (z_0, x_0, r_0) and $(z_{N+1}, x_{N+1}, r_{N+1})$ (Dirichlet condition).

By using chain rule, the radial part of Coulomb potential is a function of scalar variable r: $u_{lm} = u_{lm}(r)$, and therefore is a function of uniform grid $\{z_i\}$: $u_{lm} = u_{lm}[r(z)]$:

$$
\begin{aligned}
&\frac{d^2u(r)}{dr^2} = \frac{d}{dr}\frac{du}{dr} = \frac{d}{dr}\left(\frac{du}{dz}\frac{dz}{dr}\right) \\
&= \frac{d}{dr}\left(\frac{du}{dz}\right)\frac{dz}{dr} + \frac{du}{dz}\frac{d}{dr}\left(\frac{dz}{dr}\right) \\
&= \frac{d}{dz}\frac{dz}{dr}\left(\frac{du}{dz}\right)\frac{dz}{dr} + \frac{du}{dz}\frac{d^2z}{dr^2} \\
&= \frac{d}{dz}\left(\frac{du}{dz}\right)\left(\frac{dz}{dr}\right)^2 + \frac{du}{dz}\frac{d^2z}{dr^2} \\
&\Rightarrow \frac{d^2u}{dr^2} = \frac{d^2u}{dz^2}\left(\frac{dz}{dr}\right)^2 + \frac{du}{dz}\frac{d^2z}{dr^2} \\
&\frac{d^2u_{lm}}{dz^2}\left(\frac{dz}{dr}\right)^2 + \frac{du_{lm}}{dz}\frac{d^2z}{dr^2} - \frac{l(l+1)}{r^2}u_{lm} = -4\pi r\rho_{lm} \\
&\Rightarrow \left\{\left(\frac{dz}{dr}\right)^2\frac{d^2}{dz^2} + \frac{d^2z}{dr^2}\frac{d}{dz} - \frac{l(l+1)}{r^2}\right\}u_{lm} = -4\pi r\rho_{lm}
\end{aligned}
\tag{1.61}
$$

Equation above is equivalent to $\boldsymbol{A}\vec{\boldsymbol{x}} = \vec{\boldsymbol{b}}$.

$\left(\frac{dz}{dr}\right)^2$ and $\frac{d^2z}{dr^2}$ are known:

$$
\begin{aligned}
\left(\frac{dz}{dr}\right)^2 &= (N+1)^2\frac{r_m}{\pi^2 r(r_m + r)^2} \\
\left(\frac{d^2z}{dr^2}\right) &= (N+1)\frac{r_m^2(3r + r_m)}{2\pi(r\cdot r_m)^{3/2}(r + r_m)^2}
\end{aligned}
\tag{1.62}
$$

$\{z\}$ is uniform grid, and finite difference discretization (fdd) is naturally applied to calculate $\frac{d^2u_{lm}}{dz^2}$ and $\frac{du_{lm}}{dz}$.

As suggested by ref[2], we use 7th-order fdd:

- First order derivative fdd

Sampling point	Derivative formula	Matrix representation
−3,-2,-1,0,1,2,3	$\frac{df}{dx} \approx \frac{-f_{i-3}+9f_{i-2}-45f_{i-1}+45f_{i+1}-9f_{i+2}+f_{i+3}}{60dx}$	$\{-1\ \ 9\ \ -45\ \ 0\ \ 45\ \ -9\ \ 1\}$

- Second order derivative fdd

Sampling point	Derivative formula	Matrix representation
−3,-2,-1,0,1,2,3	$\frac{d^2f}{dx^2} \approx \frac{2f_{i-3}-27f_{i-2}+270f_{i-1}-490f_i+270f_{i+1}-27f_{i+2}+2f_{i+3}}{180dx^2}$	$\{2\ \ -27\ \ 270\ \ -490\ \ 270\ \ -27\ \ 2\}$

For the discrete function space $\{f_i | i = 0, 1, 2, 3, 4, \cdots\}$, the seventh-order central finite difference formula can only calculate up to f'_3. At the boundary, an asymmetric finite difference formula is used to calculate the differential of the boundary points. Note that constructing a differential operator matrix through finite differences is not a good method for numerically solving differential equations with initial conditions, as the constructed differential operator is likely to be singular, leading to the inability to solve the corresponding linear equation system of the differential equation. However, solving differential equations with boundary conditions through a differential operator matrix is quite suitable, as it only requires providing two boundary conditions in the first and last rows of the differential operator matrix.

- Asymmetric finite difference at boundaries

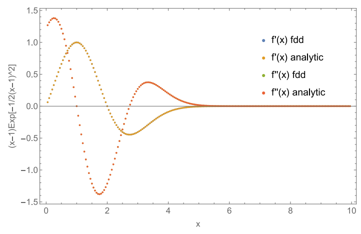
For function $\vec{f} = \{f_1, f_2, f_3, f_4, \cdots, f_{N-3}, f_{N-2}, f_{N-1}, f_N\}^T$, the first- and second-order derivative at $f_4 \cdots f_{N-3}$ can be calculated by 7th-order central fdd; for derivatives at (f_2, f_3) 与 (f_{N-2}, f_{N-1}) can be calculated by:

$$
\begin{aligned}
D^1 &= \begin{cases}
\frac{df}{dx}\Big|_{x_2} = \frac{-3f_1-10f_2+18f_3-6f_4+f_5}{12dx} \\
\frac{df}{dx}\Big|_{x_3} = \frac{3f_1-30f_2-20f_3+60f_4-15f_5+2f_6}{60dx} \\
\\
\frac{df}{dx}\Big|_{x_{N-2}} = \frac{-2f_{N-5}+15f_{N-4}-60f_{N-3}+20f_{N-2}+30f_{N-1}-3f_N}{60dx} \\
\frac{df}{dx}\Big|_{x_{N-1}} = \frac{-f_{N-4}+6f_{N-3}-18f_{N-2}+10f_{N-1}+3f_N}{12dx}
\end{cases} \\
D^2 &= \begin{cases}
\frac{d^2f}{dx^2}\Big|_{x_2} = \frac{11f_1-20f_2+6f_3+4f_4-f_5}{12dx^2} \\
\frac{d^2f}{dx^2}\Big|_{x_3} = \frac{-f_1+16f_2-30f_3+16f_4-f_5}{12dx^2} \\
\\
\frac{d^2f}{dx^2}\Big|_{x_{N-2}} = \frac{-f_{N-4}+16f_{N-3}-30f_{N-2}+16f_{N-1}-f_N}{12dx^2} \\
\frac{d^2f}{dx^2}\Big|_{x_{N-1}} = \frac{-f_{N-4}+4f_{N-3}+6f_{N-2}-20f_{N-1}+11f_N}{12dx^2}
\end{cases}
\end{aligned}
\tag{1.63}
$$

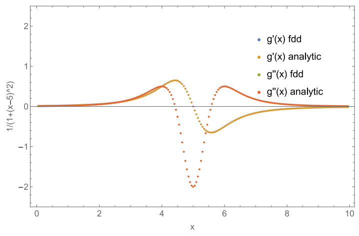
At the boundaries (f_1, f_N) we use boundary conditions: $\begin{cases} f(x_1)=y_1 \\ f(x_N)=y_N \end{cases}$, instead of calculating their derivatives.

Example

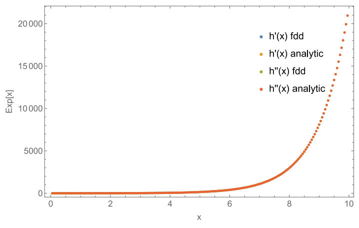
Exponential decay



Lorentian



Exponential



2. Molecular DFT calculation

Kohn-Sham equation:

$$
\begin{aligned}
&FS = SC\varepsilon \\
&F = \underbrace{T + V_{\text{ext}}}_{\text{analytic}} + \underbrace{U}_{\substack{\text{analytic}\\\text{or}\\\text{numeric}}} + \underbrace{K + C}_{\text{numeric}}
\end{aligned}
\tag{2.1}
$$

Gird is used to numerically calculate exchange-correlation functional. Coulomb potential can also be numerically calculated on the grid.

Notations :

`natom`	number of atoms

`nbf`	number of basis function
`nlm`	number of spherical pair
`nrad`	number of radial shells/grids
`nang`	number of angular grids
`natgrid`	number of atomic grids, `natgrid = nrad x nang`
`ngrid`	number of total grids, `ngrid = natom x nrad x nang = natom x natgrid`
`wi, wj`	weight of radial point/grid i, angular point/grid j
`wa`	Becke weight (atomic weight)
`i, j, k/lm`	indices for radial grid, angular grid, spherical pair `((0,0), (1,-1), (1,0), ..., (l,l))`

2.1. Atomic grid

Based on previously mentioned Gauss quadrature, spherical expansion, and quadrature on the unit sphere, we can build atomically centered Gauss-Chebyshev-Lebedev grid.

The angular grid is constructed on the Lebedev sphere sampling points. The radial grid is constructed using the second kind of Chebyshev quadrature. At each layer of the radial grid, the angular grid is scaled by the radius of that layer, and then translated with the atomic position as the origin. This results in a layer of atomic grid. There are ultimately `nrad` layers of atomic grids, with the grid closer to the atomic nucleus being denser. The total number of grids is `natgrid=nrad x nang`. Each grid has its own Cartesian coordinates, and the atomic grid can be represented in matrix form:

$$\text{nrad}\{\overbrace{\begin{pmatrix} \{x_{0,0}\ \ y_{0,0}\ \ z_{0,0}\} & \{x_{0,1}\ \ y_{0,1}\ \ z_{0,1}\} & \cdots & \{x_{0,\text{nang-1}}\ \ y_{0,\text{nang-1}}\ \ z_{0,\text{nang-1}}\} \\ \{x_{1,0}\ \ y_{1,0}\ \ z_{1,0}\} & \{x_{1,1}\ \ y_{1,1}\ \ z_{1,1}\} & \cdots & \{x_{1,\text{nang-1}}\ \ y_{1,\text{nang-1}}\ \ z_{1,\text{nang-1}}\} \\ \vdots & \vdots & \ddots & \vdots \\ \{x_{\text{nrad-1},0}\ \ y_{\text{nrad-1},0}\ \ z_{\text{nrad-1},0}\} & \{x_{\text{nrad-1},1}\ \ y_{\text{nrad-1},1}\ \ z_{\text{nrad-1},1}\} & \cdots & \{x_{\text{nrad-1},\text{nang-1}}\ \ y_{\text{nrad-1},\text{nang-1}}\ \ z_{\text{nrad-1},\text{nang-1}}\} \end{pmatrix}}^{\text{nang}} \tag{2.2}$$

3D function values on the grid are organized in matrix with shape `(nrad, nang)` : $f(\{\vec{r}_{ij}\}) = f(\{x_{ij}, y_{ij}, z_{ij}\}) = A_{ij}$.

Embedding the radial Jacobian into radial weight, and the vector outer product with angular weight gives another matrix representation:

$$w_{\text{atgrid}} = \begin{pmatrix} r_0^2 w_0^{\text{rad}} \\ r_1^2 w_1^{\text{rad}} \\ \vdots \\ r_{\text{nrad-1}}^2 w_{\text{nrad-1}}^{\text{rad}} \end{pmatrix} \cdot \begin{pmatrix} w_0^{\text{ang}} & w_1^{\text{ang}} & \cdots & w_{\text{nang-1}}^{\text{ang}} \end{pmatrix} \tag{2.3}$$

Traversing the radial grid, the angular integral is calculated at each layer of the radial grid through the Lebedev quadrature, and then multiplied by the radial weight w_i and the radial Jacobian determinant r_i^2 and summed up. This allows for the numerical calculation of the integral of a three-dimensional function over the entire real space:

$$\begin{aligned} &\int_0^\infty r^2 dr \iint_\Omega f(r,\theta,\varphi) \sin\theta d\Omega \approx \sum_i^{\text{nrad}} w_i r_i^2 \sum_j^{\text{nang}} A_{ij} w_j \\ &\int \Leftarrow w_i r_i^2 A_{ij} w_j \end{aligned} \tag{2.4}$$

The last line assumes Einstein summation convention is used.

A computationally more efficient way is doing element-wise matrix multiplication between function values A_{ij} and weight w_{atgrid}, and finally sum up all elements.

Example

- Calculate the carbon 2px orbital self-overlap integral at sto-3g basis set: $\phi(x, y, z; ax, ay, az) = \sum_i^3 c_i n_i(x - ax)e^{-\alpha_i[(x-ax)^2+(y-ay)^2+(z-az)^2]}$.

The atomic position does not affect the calculation results, because for the atomic grid of that specific atom, its position serves as the coordinate origin. Therefore, regardless of where the atom is actually located, it is equivalent to the atom being at the coordinate origin.

Below is one possible way of doing it:

```
center = np.array([5, 1., 2.])
expns = np.array([2.9412494, 0.6834831, 0.2222899])
coefs = np.array([0.15591627, 0.60768372, 0.39195739])
nrfcs = (2. * expns / np.pi)**0.75 * (4 * expns)**0.5
f = lambda x, y, z: (x - center[0]) * coefs[0] * nrfcs[0] * np.exp(-expns[0] * ((x -
center[0])**2 + (y - center[1])**2 + (z - center[2])**2)) + \
                    (x - center[0]) * coefs[1] * nrfcs[1] * np.exp(-expns[1] * ((x -
center[0])**2 + (y - center[1])**2 + (z - center[2])**2)) + \
                    (x - center[0]) * coefs[2] * nrfcs[2] * np.exp(-expns[2] * ((x -
center[0])**2 + (y - center[1])**2 + (z - center[2])**2))

xrad, rrad, wrad = gaussCheby2(75, 0.35 * 1.88972)
xleb, wleb = lebedev_rule(29) # 302 pts

xcoords = np.outer(rrad, xleb[0]) + center[0] # scaling + translation
ycoords = np.outer(rrad, xleb[1]) + center[1]
zcoords = np.outer(rrad, xleb[2]) + center[2]
A = f(xcoords, ycoords, zcoords)**2 # (nrad, nang) fnvals mat

atweight = np.outer(wrad * rrad**2, wleb) # (nrad, nang) atweight mat

quad = np.sum(A * atweight) # <2px, 2px>
print(abs(quad - 1.)) # 2.7e-8
```

Operator matrix element on the atomic grid can be calculated using the same trick.

If operator's values, except for derivative operators, on the atomic grid are known by any means, this operator can be represented in matrix form: $\hat{O} = O(\vec{r}) \approx O(\{\vec{r}_{ij}\}) = O_{ij}$.

Basis function values on the atomic grid can be represented in matrix form as well: $\{\phi_\mu(\vec{r}) \mid \mu = 1, 2, \cdots\} \approx \{\phi^\mu_{ij} \mid \mu = 1, 2, \cdots\}$.

And the matrix element is nothing but an integral:

$$\begin{aligned} O_{\mu\nu} &= \langle \phi_\mu \mid \hat{O} \mid \phi_\nu \rangle = \int \underbrace{\phi_\mu(\vec{r}) O(\vec{r}) \phi_\nu(\vec{r})}_{f(\vec{r})} d^3\vec{r} \\ &\approx \sum_i^{\text{nrad}} r_i^2 w_i \sum_j^{\text{nang}} \underbrace{\phi^\mu_{ij} \phi^\nu_{ij} O_{ij}}_{A_{ij}} w_j \\ O_{\mu\nu} &\Leftarrow w_i r_i^2 \{\phi^\mu_{ij} \phi^\nu_{ij} O_{ij}\} w_j \end{aligned} \tag{2.5}$$

Basis function ϕ_μ centers on atom C, ϕ_ν centers on atom B, to compute the matrix element on atom A's grid only the discrete values of ϕ_μ, ϕ_ν, $\hat{O}$ on A's grid are needed. The centers of basis functions are not necessarily aligned with the center of grid.

2.2. Becke molecular grid[1]

For polyatomic molecule, all positions in the space are affected by all atoms simultaneously, but some atoms may have larger influence and some atoms' influence are negligible. Namely, atoms have weights in different positions in the space.

Introducing atomic weighting function (different from w_im and w_j): $w_a(\vec{r})$, and requires $\sum_a^{\text{natom}} w_a(\vec{r}) \equiv 1$. So that the summation of weights of all atoms in any position in space gives unity.

Performing integral in a `natom` system:

$$\begin{aligned} \int f(\vec{r}) d^3\vec{r} &= \int \left\{ \sum_a^{\text{natom}} w_a(\vec{r}) \equiv 1 \right\} f(\vec{r}) d^3\vec{r} \\ &= \sum_a^{\text{natom}} \int \underbrace{w_a(\vec{r}) f(\vec{r})}_{f_a(\vec{r})} d^3\vec{r} \end{aligned} \tag{2.6}$$

The integral over the whole molecule is factored into integrals over each atomic component:

$$\int f(\vec{r}) d^3\vec{r} = \sum_a^{\text{natom}} \underbrace{\left\{ \sum_i^{\text{nrad}} r_i^2 w_i \sum_j^{\text{nang}} \left(w_{ij}^a f_{ij} \right) w_j \right\}}_{\text{atgrid}} \tag{2.7}$$

The introduction of atomic weighting function $\{w_a(\vec{r}) \mid a = 1, 2, \cdots, \text{natom}\}$ brings boundaries in space that are based on atoms, which requires:

1. Cannot be hard, atomic weights near boundaries should transition smoothly (not step function).
2. Boundaries cannot be too smooth, atomic contributions should be distinguishable (switching function).

In short, we need a boundary blurred Voronoi cell.

Define elliptical coordinate: $\mu_{ij} = \frac{r_i - r_j}{R_{ij}} \in [-1, 1]$, where r_i is the distance between a point and atom i, r_j is the distance between point and atom j, and R_{ij} is the inter-nuclear distance.

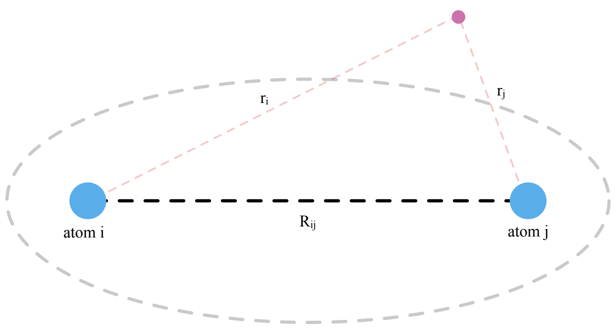


Figure 2: Confocal elliptical coordinate system.

Define cutoff profile: $s(\mu_{ij})$, which requires：

1. $s(-1) = 1$
2. $s(1) = 0$
3. $\frac{ds}{d\mu}|_{-1} = \frac{ds}{d\mu}|_1 = 0$

It should behave like step function but without jumping, and should be continuous near the nucleus (no cusp). Definition of function that meets all these requirements is not unique.

Define function $f(\mu)$:

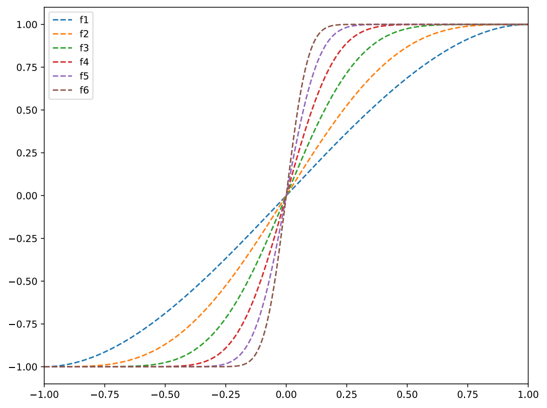
1. $f(-1) = -1$
2. $f(1) = 1$
3. $\frac{df}{d\mu}|_{-1} = \frac{df}{d\mu}|_1 = 0$

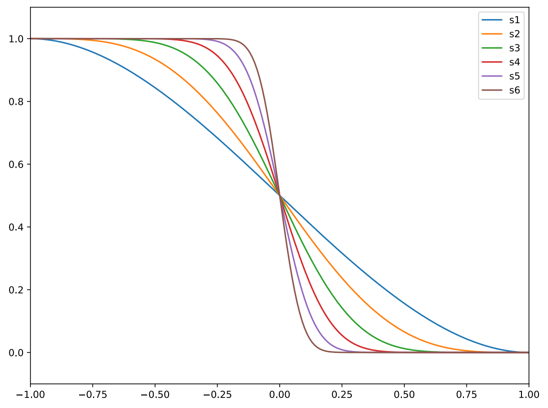
and $s(\mu)$ is obtained from $f(\mu)$: $s(\mu) = \frac{1}{2}[1 - f(\mu)]$.

One function that satisfies all the criteria: $p(\mu) = f(\mu) = \frac{3}{2}\mu - \frac{1}{2}\mu^3$. But this function is too smooth and boundaries are not well distinguished. Repeating iteration on $p(\mu)$ can systematically increase its separation capability:

$$\begin{aligned}
f_1(\mu) &= p(\mu) \\
f_2(\mu) &= p[p(\mu)] \\
f_3(\mu) &= p\{p[p(\mu)]\} \\
&\vdots \\
s_k(\mu) &= \frac{1}{2}[1 - f_k(\mu)]
\end{aligned} \tag{2.8}$$

When the iteration is very high, the limit of s_k is step function.





And finally the atomic weighting function is obtained:

$$\begin{aligned}
w_n(\vec{\boldsymbol{r}}) &= P_n(\vec{\boldsymbol{r}}) / \sum_{m}^{\text{natom}} P_m(\vec{\boldsymbol{r}}) \\
P_i(\vec{\boldsymbol{r}}) &= \prod_{j \neq i} s(\mu_{ij})
\end{aligned} \tag{2.9}$$

Core idea of Becke grid: deal with atomic grid first, then loop over all atoms.

2.3. Building up Coulomb potential

2.3.1. Building up Coulomb potential analytically via electron repulsion integral

Same as wavefunction-based method:

$$J_{\mu\nu} = \sum_{\lambda\sigma} P_{\lambda\sigma}(\mu\nu|\lambda\sigma) \tag{2.10}$$

Coulomb potential is calculated by analytically evaluating tensor contraction between ERI and electron density.

2.3.2. Building up Coulomb potential numerically via solving Poisson's equation[2]

Problem to be solved:

$$
\begin{aligned}
U(\vec{r}) &= \sum_{lm} \frac{u_{lm}(r)}{r} Y_{lm}(\Omega) \\
\rho(\vec{r}) &= \sum_{lm} \rho_{lm}(r) Y_{lm}(\Omega) \\
\left\{ \left(\frac{dz}{dr}\right)^2 \frac{d^2}{dz^2} + \frac{d^2 z}{dr^2}\frac{d}{dz} - \frac{l(l+1)}{r^2} \right\} & u_{lm} = -4\pi r \rho_{lm}
\end{aligned}
\tag{2.11}
$$

which is equivalent to solving a set of linear equations: $A\vec{x} = \vec{b}$. Both second-order derivative operator and first-order derivative are heptadiagonal matrices, $\frac{l(l+1)}{r^2}$ is diagonal, and $\frac{dz}{dr}$is constant.

2nd-order ODE needs two boundary conditions, there are `nrad` radial grids so the shape of matrix A is `(nrad+2, nrad+2)`. ρ_{lm} is spherical expansion coefficient, which has shape `(nrad+2, )`, where the first and last elements are boundary conditions for $r \to \infty$ and $r \to 0$ (Dirichlet boundary condition). Note that $\{r_i\}$ is reversely ordered.

- Boundary conditions

When $r \to \infty$ ' any charge distributions can be seen as point changes no matter how complex they actually are, so $\lim_{r\to\infty} U(\vec{r}) = U(r) = \frac{q}{r} = \sum_{lm} \frac{u_{lm}}{r} Y_{lm}$. The potential now is a function of scalar r only, therefore only the coefficient of Y_{00} is non-zero: $\frac{q}{r} = \frac{u_{00}}{r}\sqrt{\frac{1}{4\pi}} \Rightarrow u_{00} = \sqrt{4\pi} q$.

When $r \to 0$, $u_{lm} = 0$.

N.B. q is partial charge, not atomic number.

```
mweights = np.array([np.outer(rrad[iatom]**2 * wrad[iatom], wleb).ravel() * watom[iatom]
for iatom in range(natom)]): # (natom, natgrid)
    qn = np.sum(rho[iatom] * mweights[iatom]) # partial charge for atom i
```

- Spherical basis

Define spherical basis $Y_{lm,j} = Y_{kj}$ `shape (nlm, nang)`, which is shared by all atoms and the angular sampling points are the same as Lebedev quadrature points.

```
def _build_spherical_basis(lm:list[(int,int)], x:np.ndarray, y:np.ndarray, z:np.ndarray) -
> np.ndarray:
    r'''
    Arguments
    ---------
    lm : list[(int,int)]
        Angular number. [(0,0), (1,-1), (1,0), ..., (lmax,lmax)]
    x, y, z : np.ndarray:
        Cartesian coordinates of lebedev sampling points on unit sphere

    Return
    ------
    y_kj : np.ndarray (nlm, nang)
    '''
    nlm = len(lm)
    nang = len(x)
    y_kj = np.zeros((nlm,nang))
    for ilm, (l,m) in enumerate(lm):
        y_kj[ilm] = compute_real_spherical_harmonics(l, m, theta=np.arccos(z),
phi=np.arctan2(y,x))
    return y_kj
```

- Spherical expansion on electron density

Using the orthogonality of spherical harminics:

$$\begin{aligned}\rho_{lm}(r) &= \iint_{\Omega} Y_{lm}(\Omega)\rho(\vec{\boldsymbol{r}})\sin\theta d\Omega \\ \rho_{ik} &= \sum_{j} \underbrace{\left(\rho_{ij} w^{a}_{ij}\right)}_{\rho^{a}_{ij}} \underbrace{\left(Y_{kj} w_{j}\right)^{T}}_{\text{row-wise product}}\end{aligned} \tag{2.12}$$

ρ_{ij} is eletron density values on the grid of atom A in matrix form, multiplied by A's Becke weight on its grid w^{a}_{ij} we get ρ^{a}_{ij}. By doing matrix multiplication with spherical basis the electron density expansion coefficients are obtained ρ_{ik} `shape (nrad, nlm)`

```
rho_ik[iatom] = (watom[iatom] * rho[iatom]).reshape((nrad, nang)) @ (y_kj * wleb).T
```

- Coulomb potential on atomic grid

After getting ρ^{a}_{ik}, by using fdd and solving linear equations the Coulomb potential expansion coefficients on atomic grid are obtained: $\rho^{a}_{ik} \to u^{a}_{ik}$.

But the Coulomb potential obtained at this point is "local". As shown in Figure 3, the target point is not on the grid of atom a, but the contribution of atom a to the Coulomb potential at this point is greater than that of atom b, which generates this point. If the contribution of atom a is not considered, the Coulomb potential at this point will be greatly underestimated. Since this point is not a grid point of atom a, the Coulomb potential of atom a at this point cannot be obtained by solving the Poisson equation, and therefore interpolation calculation is required.

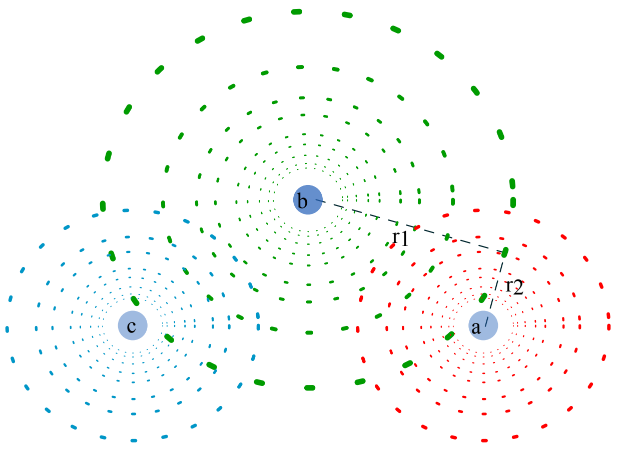


Figure 3: Numerically build Hartree potential on the grids.

$u_{lm}(r)$ is a function of scalar r, intuitively we would like to do interpolation on r. However r is non-uniform grid, and the large the r the larger the spacing, which leads to very inaccurate interpolated results at large r.

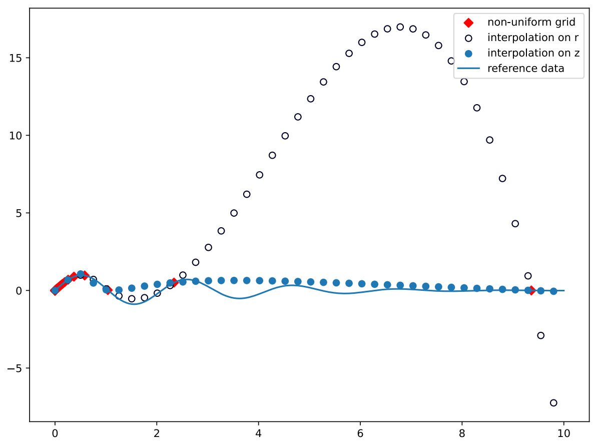


Figure 4: Cubic spline interpolation.

- Construction Coulomb potential on the whole grid via interpolation

As suggested in [2], the interpolation is performed on uniform grid $\{z\}$. Note that z is the same for all atoms while r is scaled differently for different elements, this is another advantage of using Chebyshev radial grid.

```
for ilm, (l, _) in ennumerate(lm):
    u_ik[iatom] = poisson_solver(rho_ik[iatom, :, ilm], l, rrad[iatom], qn) # (nrad, nlm)
    u_ik_splines[iatom].append(CubicSpline(z, u_ik[iatom][:, ilm]))
```

After constructing an interpolator for each pair of (l, m), interpolation needs to be performed on all grids. Thus, the constructed Coulomb potential is no longer "local" but "global". Therefore, it is necessary to calculate the distances from all atoms to all grids:

```
def _get_distance(grid_global:np.ndarray, xyz:np.ndarray) -> np.ndarray:
'''
    Arguments
    ---------
    grid_global : np.ndarray (natom,nradxnang,3)
    xyz : np.ndarray (natom,3)

    Return
    ------
    dist : np.ndarray (natom,natomxnradxnang)
    '''
    grid_total = np.concatenate(grid_global) # (natomxnradxnang,3)
    natom = len(xyz)
    ngrid = len(grid_total)
    dist  = np.zeros((natom,ngrid))
    for iatom in range(natom):
        _disp = grid_total[:,:3] - xyz[iatom]
        _dist = np.linalg.norm(_disp, axis=1)
        dist[iatom]  = _dist

    return dist

 zz = [(nrad + 1) / np.pi * np.arccos((dist[iatom] - rms[iatom]) / (dist[iatom] +
rms[iatom])) for iatom in range(natom)] # r = r(z) z = z(r) for interpolation
```

and calculate the contribution of atom i to all grids via cubic spline interpolation:

```
for ilm, (l, _) in enumerate(lm):
  u_ik[iatom] = poisson_solver(rho_ik[iatom], ...) # (nrad, nlm)
  u_ik_splines[iatom].append(CubicSpline(z, u_ik[iatom][:,ilm])) # (nlm, )
  u_ik_interp[iatom][ilm] = u_ik_splines[iatom][ilm](zz[iatom]) # (nlm, ngrid)
```

Re-constructing the global Coulomb potential generated by atom i by inverse spherical expansion: $U^a_{\text{ngrid}} = \sum_k u^{\text{a interp}}_{k,\text{ngrid}}/\text{dist}^a \cdot y^a_{k,\text{ngrid}}$, where $y^a_{k,\text{ ngrid}}$ needs to compute solid angles from atom i to all grids, and then compute the spherical harmonics values of all these solid angles at all (l, m) pairs:

```
def _get_solid_angle(grid_global:np.ndarray, xyz:np.ndarray) -> tuple[np.ndarray,
np.ndarray]:
    '''
    Arguments
    ---------
    grid_global : np.ndarray (natom,nradxnang,3)
    xyz : np.ndarray (natom,3)

    Return
    ------
    theta : np.ndarray (natom,natomxnradxnang)
    phi : np.ndarray (natom,natomxnradxnang)
    '''
    grid_total = np.concatenate(grid_global) # (natomxnradxnang,3)
    natom = len(xyz)
    ngrid = len(grid_total)
    theta = np.zeros((natom,ngrid))
    phi   = np.zeros((natom,ngrid))
    for iatom in range(natom):
        _disp = grid_total[:,:3] - xyz[iatom]
        _disp_normalized = _disp / _dist[:,None]
        _theta = np.arccos(_disp_normalized[:,2])
        _phi = np.arctan2(_disp_normalized[:,1], _disp_normalized[:,0])
        theta[iatom] = _theta
        phi[iatom]   = _phi

    return theta, phi

def _get_spherical_vals(lm:list[(int,int)], theta:np.ndarray, phi:np.ndarray) ->
np.ndarray:
    '''
    Arguments
    ---------
    lm : list[(int,int)]
    theta : np.ndarray (natom, natomxnradxnang)
    phi : np.ndarray (natom, natomxnradxnang)

    Return
    ------
    ylm : np.ndarray (natom, nlm, natomxnradxnang)
    '''
    natom, ngrid = theta.shape
    nlm = len(lm)
    ylm = np.zeros((natom,nlm,ngrid)) # (natom,natomxnradxnang)

    for iatom in range(natom):
        for ilm, (l,m) in enumerate(lm):
```

```python
            ylm[iatom,ilm] = rsh(l, m, theta=theta[iatom], phi=phi[iatom])

    return ylm
```

And finally:

```python
u_ij = np.zeros((ngrid,))

for iatom in range(natom):
    for ilm, (l, _) in enumerate(lm):
        u_ik[iatom] = poisson_solver(rho_ik[iatom], ...) # (nrad, nlm)
        u_ik_splines[iatom].append(CubicSpline(z, u_ik[iatom][:,ilm])) # (nlm, )
        u_ik_interp[iatom][ilm] = u_ik_splines[iatom][ilm](dist) # (nlm, ngrid)
    u_ij += np.sum((u_ik_interp[iatom] / dist[iatom]) * ylm[iatom], axis=0)
```

The interpolated Coulomb potential loops over all atoms, and takes contributions from all atoms on the whole grid into consideration. Thus what we get here is the global Coulomb potential.

`u_ij shape (ngrid = natomxnradxnang, )`. Do so to facilitate the subsequent evaluation of operator matrix element.

- Coulomb potential matrix element

Equation (2.5) computes matrix element on atomic grid. To conveniently calculate matrix elements on the whole molecular grid, flatten all quantities represented in matrix form: `u_ij -> shape (natomxnradxnang,)`.

The calculation of matrix element involves calculation of basis function values on the grid. Organize basis function values as multi-dimensional array: $\left\{\phi^{\mu}_{a,ij} \mid \mu = 1, 2, \cdots, \text{nbf} \mid a = 1, 2, \cdots, \text{natom}\right\}$. Flatten it atom-wisely as a matrix with shape `(nbf x ngrid)` : $\{\phi_{\mu,\text{ ngrid}} \mid \mu = 1, 2, \cdots, \text{nbf}\}$.

And combine all kinds of weights and Jacobian and flatten them:

```python
# watom (natom, natgrid)
mweights = [np.outer(rrad[iatom]**2 * wrad[iatom], wleb).ravel() * watom[iatom] for iatom
in range(natom)] # (natom, natgrid)
mweights = np.flatten(mweights) # (natomxnradxnang, )
```

And the Coulomb potential matrix can be evaluated by:

$$
\begin{aligned}
&J_{\mu\nu} = \langle \phi_\mu \mid U \mid \phi_\nu \rangle \\
&\begin{cases} \phi_\mu \to (\text{ngrid}, \ ) \\ \phi_\nu \to (\text{ngridm}, \ ) \\ U \to \text{u\_ij } (\text{ngrid}, \ ) \end{cases} \\
&\langle \phi_\mu \mid U \mid \phi_\nu \rangle = \int r^2 dr \iint \phi_\mu(\vec{r}) U(\vec{r}) \phi_\nu(\vec{r}) \sin\theta d\Omega \\
&\approx \sum_a^{\text{natom}} w_a \sum_i^{\text{nrad}} (r_i^a)^2 w_i^a \sum_j^{\text{nang}} \phi_\mu(r_i^a, \Omega_j) \phi_\nu(r_i^a, \Omega_j) U(r_i^a, \Omega_j) w_j \\
&= \sum_a \sum_i \sum_j \underbrace{\left\{ w_a w_i^a (r_i^a)^2 w_j \right\}}_{\text{mweights}} \phi^\mu_{a,ij} \phi^\nu_{a,ij} U_{a,ij} \\
&= \sum_a \sum_{ij}^{\text{natgrid}} W_{a,\text{ natgrid}} \phi^\mu_{a,\text{ natgrid}} \phi^\nu_{a,\text{ natgrid}} U_{a,\text{ natgrid}} \\
&= \sum_{aij}^{\text{ngrid}} W_{\text{ngrid}} \phi^\mu_{\text{ngrid}} \phi^\nu_{\text{ngrid}} U_{\text{ngrid}} \Rightarrow J_{\mu\nu}
\end{aligned}
\tag{2.13}
$$

One possible implementation:

```
for mu in range(nbf):
    for nu in range(nbf):
        Juv[mu,nu]  = np.sum(aos_vals[mu].ravel() * u_ij * aos_vals[nu].ravel() *
mweights)
```

And we have numerically calculated the Coulomb potential operator matrix.

2.4. Numeric quadrature of exchange-correlation functional

Exchange-correlation operator matrix elements are usually evaluated numerically via quadrature, as the mathematical expression for exchange-correlation functional is complicated and the integral over real space is pretty hard to compute analytically.

Exchange-correlation functional is functional of electron density, so we calculate the electron density on the grid:

$$
\begin{aligned}
\rho(\boldsymbol{r}) &= \sum_i^{\text{nocc}} 2\ |f_i|^2 \\
&= \sum_i^{\text{nocc}} 2 \sum_\mu C_{\mu i} \phi_\mu \sum_\nu C_{\nu i} \phi_\nu \\
&= \sum_{\mu\nu} \left(\sum_i^{\text{nocc}} 2 C_{\mu i} C_{\nu i} \right) \phi_\mu \phi_\nu \\
&= \sum_{\mu\nu} P_{\mu\nu} \phi_\mu \phi_\nu
\end{aligned}
\tag{2.14}
$$

in matrix form:

```
rho = np.zeros((ngrid,))
for mu in range(nbf):
    for nu in range(nbf):
        rho += 2 * aos_vals[mu] * Puv[mu,nu] * aos_vals[nu]
```

Integrating electron density over the whole real space reproduce the total number of electrons: ne $= \int \rho(\boldsymbol{r}) d^3\boldsymbol{r} =$ `np.sum(rho * mweights)`.

By using the same trick as before, the exchange-correlation potential matrix element and energy density matrix element are obtained. Once electron density on the grid is calculated, the exchange-correlation potential and energy density can be computed by calling `libxc`:

```
#include "xtensor.hpp"
#include "xc.h"
// link library `-lxc`

xc_func_type funcx, funcc;
xc_func_init(&funcx, X_id, XC_UNPOLARIZED); // init exchange
xc_func_init(&funcc, C_id, XC_UNPOLARIZED); // init correlation

auto rho = compute_rho();

xt::xtensor<double, 1> vx = xt::zeros<double>({ngrid}); // exchange potential
xt::xtensor<double, 1> ex = xt::zeros<double>({ngrid}); // exchange energy density
xt::xtensor<double, 1> vc = xt::zeros<double>({ngrid});
xt::xtensor<double, 1> ec = xt::zeros<double>({ngrid});

xc_lda_vxc(&funcx, ngrid, rho.data(), vx.data());
xc_lda_exc(&funcx, ngrid, rho.data(), ex.data());
xc_lda_vxc(&funcc, ngrid, rho.data(), vc.data());
xc_lda_exc(&funcc, ngrid, rho.data(), ec.data());

for (auto ibf=0; ibf < nbf; ++ibf) {
    for (auto jbf=0; jbf < nbf; ++jbf) {
        Kuv(ibf,  jbf) = xt::sum(aos_vals_mu * vx * aos_vals_nu * mweights)();
        Cuv(ibf,  jbf) = xt::sum(aos_vals_mu * vc * aos_vals_nu * mweights)();
        Exuv(ibf, jbf) = xt::sum(aos_vals_mu * ex * aos_vals_nu * mweights)();
        Ecuv(ibf, jbf) = xt::sum(aos_vals_mu * ec * aos_vals_nu * mweights)();
        }
    }
}

xc_func_end(&funcx); // free exchange
xc_func_end(&funcc); // free correlation
```

2.5. Fock matrix

Summing up all operator matrices gives Fock matrix:

$$F = \underbrace{T + V_{\text{ext}}}_{\text{analytic}} + \underbrace{U}_{\substack{\text{analytic}\\\text{or}\\\text{numeric}}} + \underbrace{K + C}_{\text{numeric}} \tag{2.15}$$

The energy components can be calculated by using density matrix: $\langle \hat{O} \rangle = \text{Tr}\{PO\}$. And the energy decomposition:

```
kin_e         = np.trace(Puv @ tij.T)
ext_e         = np.trace(Puv @ vij.T)
hartree_e     = np.trace(Puv @ Juv.T) / 2 # double counting
exchange_e    = np.trace(Puv @ Exuv.T) # energy density matrix DOES NOT appear in Fock
```

```
correlation_e = np.trace(Puv @ Ecuv.T)
etot = kin_e + ext_e + hartree_e + exchange_e + correlation_e + nuc_repulsion
```

2.6. Transformation from Cartesian basis function to pure basis function

The ordering of Cartesian basis function is typically lexicographic, for details see Section 1.1.2. While there are two commonly used ordering conventions for the pure basis functions:

1. HORTON/ORCA convention. see this link

   $m = 0, 1, -1, 2, -2, \cdots, l, -l$. C stands for cos $\because m > 0$, S stands for sin $\because m < 0$.

l	Ordering
0	C_{00}
1	$C_{10} = z, C_{11} = x, S_{11} = y$
2	$C_{20}, C_{21}, S_{21}, C_{22}, S_{22}$
3	$C_{30}, C_{31}, S_{31}, C_{32}, S_{32}, C_{33}, S_{33}$
4	$C_{40}, C_{41}, S_{41}, C_{42}, S_{42}, C_{43}, S_{43}, C_{44}, S_{44}$

2. libint/CCA convention $m = -l, -l + 1, \cdots, l - 1, l$.

The pure basis function is linear combination of Cartesian basis function. In practical calculations, the results (such as integrals, function values, etc.) are first obtained in Cartesian form, and then by matrix multiplication with transform matrix the results in pure form are obtained.

Transformation matrices under the HORTON convention can be found at https://github.com/theochem/iodata/blob/main/tools/harmonics.py.

For example, matrix form of basis function in Cartesian form on the grid: `aos_vals (nbf, ngrid)`. The transformation matrix is block diagonal:

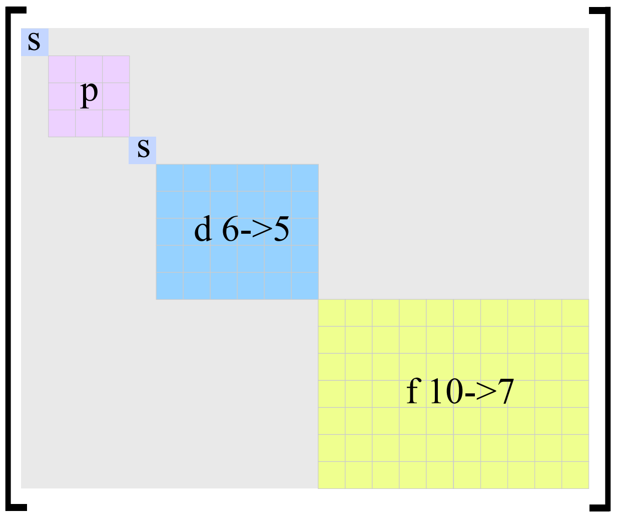


Figure 5: Cartesian to pure transformation matrix.

Matrix multiplying with `aos_vals` gives pure basis function values on the grid: `spsdf 21 -> 17`.

The same idea applies to the operator matrices. Note that the transformation matrices are highly sparse.

Below lists some transformation matrices up to `lmax=3` in HORTON convention:

$$
\begin{aligned}
&C_{00} = 1\\
&\begin{pmatrix} C_{10}\\ C_{11}\\ S_{11}\end{pmatrix} = \begin{pmatrix} \cdot & \cdot & 1\\ 1 & \cdot & \cdot\\ \cdot & 1 & \cdot \end{pmatrix} \cdot \begin{pmatrix} x\\ y\\ z\end{pmatrix}\\
&\begin{pmatrix} C_{20}\\ C_{21}\\ S_{21}\\ C_{22}\\ S_{22}\end{pmatrix} = \begin{pmatrix} -\frac{1}{2} & \cdot & \cdot & -\frac{1}{2} & \cdot & 1\\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot\\ \frac{\sqrt{3}}{2} & \cdot & \cdot & -\frac{\sqrt{3}}{2} & \cdot & \cdot\\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \cdot \begin{pmatrix} xx\\ xy\\ xz\\ yy\\ yz\\ zz\end{pmatrix}\\
&\begin{pmatrix} C_{30}\\ C_{31}\\ S_{31}\\ C_{32}\\ S_{32}\\ C_{33}\\ S_{33}\end{pmatrix} = \begin{pmatrix} \cdot & \cdot & -\frac{3\sqrt{5}}{10} & \cdot & \cdot & \cdot & \cdot & -\frac{3\sqrt{5}}{10} & \cdot & 1\\ -\frac{\sqrt{6}}{4} & \cdot & \cdot & -\frac{\sqrt{30}}{20} & \cdot & \frac{\sqrt{30}}{5} & \cdot & \cdot & \cdot & \cdot\\ \cdot & -\frac{\sqrt{30}}{20} & \cdot & \cdot & \cdot & \cdot & -\frac{\sqrt{6}}{4} & \cdot & \frac{\sqrt{30}}{5} & \cdot\\ \cdot & \cdot & \frac{\sqrt{3}}{2} & \cdot & \cdot & \cdot & \cdot & -\frac{\sqrt{3}}{2} & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot\\ \frac{\sqrt{10}}{4} & \cdot & \cdot & -\frac{3\sqrt{2}}{4} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \frac{3\sqrt{2}}{4} & \cdot & \cdot & \cdot & \cdot & -\frac{\sqrt{10}}{4} & \cdot & \cdot & \cdot \end{pmatrix} \cdot \begin{pmatrix} xxx\\ xxy\\ xxz\\ xyy\\ xyz\\ xzz\\ yyy\\ yyz\\ yzz\\ zzz\end{pmatrix}
\end{aligned}
\tag{2.16}
$$

3. Appendix

- **密度矩阵**

Fock 矩阵的特征向量组成 LCAO 分子轨道的系数:

$$
\begin{aligned}
&FC = SC\varepsilon\\
&C_{N\times N} = \left(\overrightarrow{C_1}\ \overrightarrow{C_2}\ \cdots\ \overrightarrow{C_N}\right)
\end{aligned}
\tag{3.1}
$$

$$
\begin{aligned}
&\psi_i = \sum_\mu C_{\mu,i}\phi_\mu\\
&|\psi_i|^2 = \left(\sum_\mu C_{\mu,i}\phi_\mu\right)\left(\sum_\nu C_{\nu,i}\phi_\nu\right)^*\\
&\rho = \sum_i 2\ |\psi_i|^2 = \sum_\mu\sum_\nu\left(\sum_i 2C_{\mu,i}C^*_{\nu,i}\right)\phi_\mu\phi_\nu \Rightarrow \sum_{\mu,\nu} P_{\mu,\nu}\phi_\mu\phi_\nu
\end{aligned}
\tag{3.2}
$$

$P_{\mu,\nu} = \sum_i^{N/2} 2C_{\mu,i}C_{\nu,i}$ 即为密度矩阵.

通过矩阵乘法计算

```
Puv = 2 * C[:,:ne//2] @ C[:,:ne//2].T
```

$$
\begin{aligned}
\text{ne} &= \sum_{i}^{\text{occ}} 2 \int |\psi_i(\boldsymbol{r})|^2 \ d^3\boldsymbol{r} \\
&= \sum_{i}^{\text{occ}} \int 2\left(\sum_{\mu} |C_{\mu,i}\phi_{\mu}|\right)\left(\sum_{\nu} C_{\nu,i}\phi_{\nu}\right) d^3\boldsymbol{r} \\
&= \sum_{\mu}\sum_{\nu} 2 \sum_{i}^{\text{occ}} C_{\mu,i}C_{\nu,i} \int \phi_{\mu}^{\dagger}\phi_{\nu} d^3\boldsymbol{r} \\
&= \sum_{\mu\nu} P_{\mu,\nu} \langle \phi_{\mu} \mid \phi_{\nu} \rangle \\
&\Leftrightarrow \sum_{\mu\nu} P_{\mu,\nu} S_{\mu,\nu} \\
&\equiv \text{Tr}\{PS^T\}
\end{aligned}
\tag{3.3}
$$

直接計算“密度矩陣” $P_{\mu,\nu}$ 的跡不會給出電子數, 因爲**基函數不正交**.

- **拉普拉斯算符**

球坐标下的拉普拉斯算符:

$$
\begin{aligned}
\nabla^2 &= \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial}{\partial r}\right) - \frac{L^2}{r^2} \\
L^2 &= -\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial}{\partial\theta}\right) - \frac{1}{\sin\theta}\frac{\partial^2}{\partial\varphi^2}
\end{aligned}
\tag{3.4}
$$

- **广义对角化**

1. Symmetric orthogonalization

$$
\begin{aligned}
&FC = SC\varepsilon \\
&C = S^{-\frac{1}{2}}C' \\
&FS^{-\frac{1}{2}}C' = SS^{-\frac{1}{2}}C'\varepsilon \\
&FS^{-\frac{1}{2}}C' = S^{\frac{1}{2}}C'\varepsilon \\
&\left(S^{-\frac{1}{2}}FS^{-\frac{1}{2}}\right)C' = C'\varepsilon \\
&\Rightarrow F'C' = C'\varepsilon
\end{aligned}
\tag{3.5}
$$

$$
\begin{aligned}
&P^{-1}SP = s \\
&S^{-\frac{1}{2}} = Ps^{-\frac{1}{2}}P^{-1} \\
\because\ &S^{-\frac{1}{2}}S^{\frac{1}{2}} = Ps^{-\frac{1}{2}}P^{-1}Ps^{\frac{1}{2}}P^{-1} \\
&= Ps^{-\frac{1}{2}}s^{\frac{1}{2}}P^{-1} = I
\end{aligned}
\tag{3.6}
$$

2. Canonical orthogonalization …

- Slater **交换泛函**

$$
\begin{aligned}
&E_x[n(\boldsymbol{r})] = \int -\frac{3}{4}\left(\frac{3}{\pi}\right)^{\frac{1}{3}} n^{\frac{4}{3}}(\boldsymbol{r}) d^3\boldsymbol{r} \\
&V_x[n(\boldsymbol{r})] = \frac{\delta E_x[n]}{\delta n} = -\left(\frac{3}{\pi}\right)^{\frac{1}{3}} n^{\frac{1}{3}} \\
&E_x[n] = \int n\varepsilon_x d^3\boldsymbol{r} \\
&\Rightarrow \varepsilon_x[n(\boldsymbol{r})] = -\frac{3}{4}\left(\frac{3}{\pi}\right)^{\frac{1}{3}} n^{\frac{1}{3}}(\boldsymbol{r}) = -\frac{3}{4}V_x
\end{aligned}
\tag{3.7}
$$

交换泛函的能量:

$$
\begin{aligned}
E_{\text{exchange}} &= \int V_x n d^3\boldsymbol{r} = \int V_x \sum_i^{\text{occ}} \sum_{\mu,\nu} 2C_{\mu,i}C_{\nu,i}\phi_\mu\phi_\nu \\
&= \sum_{\mu,\nu} P_{\mu,\nu} \int \phi_\mu V_x \phi_\nu d^3\boldsymbol{r} \\
&= \sum_{\mu,\nu} P_{\mu,\nu} V^x_{\mu,\nu} \\
&= \mathrm{Tr}\{\mathrm{P\ @\ V.T}\}
\end{aligned} \tag{3.8}
$$

最后乘上 $-\frac{3}{4}$ 得到交换能量.

在 `libxc` 中编号为 1.

- **谱方法, 范德蒙德矩阵**

Bibliography

[1] A. D. Becke, “A multicenter numerical integration scheme for polyatomic molecules,” The Journal of Chemical Physics, vol. 88, no. 4, pp. 2547–2553, 1988, doi: 10.1063/1.454033.

[2] A. D. Becke, “A multicenter numerical integration scheme for polyatomic molecules,” The Journal of Chemical Physics, vol. 88, no. 4, pp. 2547–2553, 1988, doi: 10.1063/1.454033.

[3] S. Obara and A. Saika, “Efficient recursive computation of molecular integrals over Cartesian Gaussian functions,” The Journal of Chemical Physics, vol. 84, no. 7, pp. 3963–3974, 1986, doi: 10.1063/1.450106.

[4] J. C. Slater, “Atomic Radii in Crystals,” The Journal of Chemical Physics, vol. 41, no. 10, pp. 3199–3204, 1964, doi: 10.1063/1.1725697.

[5] V. I. Lebedev, “Values of the nodes and weights of quadrature formulas of Gauss–Markov type for a sphere from the ninth to seventeenth order of accuracy that are invariant with respect to an octahedron group with inversion,” Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki, vol. 15, no. 1, pp. 48–54, 1975.

[6] C. H. Beentjes, “Quadrature on a spherical surface,” Working note available on the website http://people. maths. ox. ac. uk/beentjes/Essays, 2015.