lyuhang428@gmail.com

符号约定

eDFT	电子密度泛函理论
cDFT	经典密度泛函理论
$\psi(\vec{r})$, $f(\vec{r})$	波函数
$\phi(\vec{r})$	基函数
$U(\vec{r})$	库伦势能
$u(r)$	库伦势球谐展开系数
$\rho(\vec{r})$	电子密度(eDFT)，粒子数密度(cDFT)

目录

1. 数学基础

1.1. 高斯型基函数

1.1.1. 高斯乘积定理

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

定义一些参数:

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

当高斯函数的中心不再是原点时, 被积函数的形式需要稍作调整:

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

如果将该积分在直角坐标下计算, 高斯函数的中心在哪里无所谓, 因为所有的积分变量的积分范围为负无穷到正无穷, 积分区间对于任何中心都对称. 而在球坐标下, 积分变量r的积分范围为零到正无穷, 该区间对函数中心不再是对称的, 最终会给出高斯误差函数的错误结果.

函数中心改变不过是平移变换, 平移变换不改变积分结果（曲线下面积）.

但是如果是两个高斯函数乘在一起, 需要额外考虑由于高斯乘积定理产生出的额外的常数项:

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

也就是说, 在使用高斯求积计算球坐标下的积分时, 无需考虑由高斯乘积定理带来的新的函数中心, 因为两条高斯函数相乘后的新的函数中心永远可以视为新的原点. 唯一需要显示计算的是前置系数$K_{AB}^{\alpha\beta}$, 该系数又依赖于两条高斯函数各自的指数和中心.

1.1.2. 纯基函数与笛卡尔基函数

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

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

通常来说同一组收缩中的所有组分使用同一套角动量：$l_{\alpha,i} == l_{\alpha,j};\ \alpha = x,y,x.$ 当显式线性组合成球谐基函数时不同组分会使用不同的角动量. 例如 $d_{x^2-y^2}$:

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

即，球谐形式的 $d_{x^2-y^2}$ 组分中有两组角动量. 但是实操中不会在事先就进行线性组合，而是在笛卡尔基函数上进行操作（例如计算基函数在网格上的函数值），最后通过转换矩阵得到球谐基函数.

基函数(原子轨道), 是角度部分与径向部分的乘积: $\phi(\boldsymbol{r})=Y_l^m(\theta,\phi)R(r)$. 其中角度部分是球谐函数, 而径向部分是指数函数. 对于角度部分, 球谐函数是拉普拉斯方程在球坐标下经过变量分离后角度部分的解, 通常是复的:

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

上式中的变量分离提示我们对于一个实空间函数 $\rho(\vec{\boldsymbol{r}})$，我们可以将其拆解成径向部分和角度部分，这实际上就是球谐展开，此时球谐函数作为基函数，而径向部分则是展开系数. 即使径向变量和角度变量耦合，这种展开也可以进行.

角动量算符:

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

球谐函数解析表达式:

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

但是无论是笛卡尔基函数还是球谐基函数，都不会直接使用球谐函数（包括线性组合后的实球谐函数），二是使用笛卡尔坐标下的多项式等价表示（包括球谐基函数）. 对应关系见 https://en.wikipedia.org/wiki/Table_of_spherical_harmonics.

形式上:

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

注意球谐函数形式在实际计算中不会使用.

在纯基函数(pure basis function)表达式中取的是向量的模$|\vec{\boldsymbol{r}}-\vec{\boldsymbol{R}}|$；而在笛卡尔基函数表达式**不能**取绝对值 $\otimes\ |x-a_x|$.

纯基函数对任意角动量 l 不存在冗余, 例如对于$l=2$共有 5 条 d 轨道, 对于$l=3$共有 7 条 f 轨道, 以此类推. 因为在构造时纯基函数的角度部分直接取的是球谐函数, 自然不存在冗余.

而对于笛卡尔基函数, 情况有所不同. 此时的角度部分是多项式$x^{l_x}y^{l_y}z^{l_z}$. 当角动量$l\geq 2$时, 笛卡尔基函数相较于纯基函数存在冗余, 且角动量越高冗余越多:

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	$y(3x^2-y^2), xyz, yz^2, z^3, xz^2,$ $z(x^2-y^2), x(x^2-3y^2)$	$10f \to 7f$
⋮			

笛卡尔基函数角动量的排序通常采用字典顺序（lexicographic, 纯基函数的顺序常用的有两种，见下文）: $\{xx, xy, xz, yy, yz, zz\}$, $\{xxx, xxy, xxz, xyy, xyz, xzz, yyy, yyz, yzz, zzz\}$.

```
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. 归一化问题

笛卡尔基函数的归一化系数: [3]

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

纯基函数的归一化系数:

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

基函数既不正交，也不归一！

位于不同中心的基函数不正交，这也是为什么会存在重叠矩阵 S_{ij} 的原因. 然而更反直觉但数学上允许的是，基函数本身不需要归一，只要这些函数满足线性无关的条件，就是数学上允许的基函数. 不同的计算软件有不同的归一化规定，笛卡尔基函数的每一组分(primitive function)除了基组文件规定的系数外，还存在:

1. 每一组分都乘上各自的归一化系数(式 (1.10)). 即所有组分都是归一化的（但是缩并后的基函数不保证归一化）.
2. 以轴对齐组分（axis-aligned, i.e. xx, yy, xxx, ...）为基准，所有组分乘上相同的常数 $N = \left(\frac{2\alpha}{\pi}\right)(4\alpha)^{\frac{l_{\mathrm{tot}}}{2}}\sqrt{\frac{1}{(2l_{\mathrm{tot}}-1)!!}}$. 即只有轴对齐的组分是归一化的.
3. 计算自重叠积分后，强制每一条基函数归一化.

重叠矩阵的对角元素，不保证全部为 1!

最简单的情况，是根本不管归一化，直接进行计算. 但是提前对基函数/组分进行归一化操作有其好处，其一是确保数值稳定性，防止出现极小/极大值（`1e-49`, `1e+17`, etc.. 重叠矩阵条件数更好）；其二是归一化的基函数可解释性更好. 但是无论采用何种归一化约定（甚至是根本不归一化），**确保在整个计算过程中使用相同的约定**. 特别是需要基函数在空间网格点上的函数值的情形（例如数值计算交换相关势）.

归一化约定不影响物理可观测量(observable. 或期望值 expectation value)，但是会影响算符矩阵的构建. 假设 $\langle \phi_\mu \mid \phi_\mu \rangle \equiv 1$, 那么未归一化的基函数是归一化基函数乘上缩放系数:

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

即波函数不受影响.

但是算符的矩阵形式发生改变，因为算符的矩阵表示取决于基向量的选择:

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

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

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

系数矩阵随之变化.

Kohn-Sham 方程形式不变，且特征值不变:

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

密度矩阵变换后为(n_i为占据数)：

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

期望值保持不变（trace invariant）：

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

波函数归一性（拉格朗日乘子法的约束条件）保持不变：

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

1.2. 高斯求积

注：本节内容不是严格的数学定义.

数值积分是将函数 $f(x)$ 在区间 $[a,b]$ 上的离散点处的函数值做加权平均作为积分近似值（基本思想为积分中值定理）：

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

其中 $\{A_i\}$ 为系数（权重），$\{x_i\}$ 为求积节点.

最简单的梯形法使用的是等分节点，各点处的权重相同，需要非常多的离散点才能达到满意的数值精度. 为了用更少的节点获得更高的精度，节点的选择和权重的确定很重要.

如果存在节点 $x_i \in [a,b]$ 以及系数 $\{A_i\}$ 使得 $\int_a^b f(x)dx \approx \sum_{i=0}^{n} A_i f_i$ 具有 $2n+1$ 次代数精度，节点 $\{x_i\}$ 称为**高斯点**，A_i 称为**高斯系数**, 求积公式称为**高斯型求积公式**.

通常高斯点是各种正交多项式在各自正交区间上的零点，系数（权重）有专门的公式进行计算，且进行数值求积时需要考虑各种正交多项式的权重函数：

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

1.2.1. 正交多项式

如果一族多项式中的任意两个不同的多项式的带权内积为零，那么这族多项式是正交多项式:

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

三角函数 $\{1, \cos x, \sin x, \cos 2x, \sin 2x, \cdots\}$ 在区间 $[-\pi, \pi]$ 上构成正交函数族（不是多项式）.

- 如果 $\{\phi_k\}_{k=0}^{\infty}$ 在区间 $[a,b]$ 上带权 ω 正交，H_n 是所有次数不超过 n 的多项式组成的线性空间，那么 $\{\phi_i \mid i = 0, 1, \cdots, n\}$ 构成 H_n 的一组基. 即对所有 $p(x) \in H_n$, $p(x) = \sum_{j=0}^{n} c_j\phi_j$.
- 正交多项式满足三项递推关系：

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

三项递推关系是高效计算正交多项式的手段.

- 如果 $\{\phi_n\}_0^\infty$ 在区间 $[a, b]$ 带权函数 ω 正交, 那么 ϕ_n 在区间 $[a, b]$ 内有 n 个不同零点.

1.2.2. 常用正交多项式

- 勒让德求积

积分区间 $[-1, 1]$, 权重函数 $\omega(x) = 1$, $P_0 = 1$, $P_1 = x$, $P_n = \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2 - 1)^n$.

1. $$\int_{-1}^{1} P_n P_m dx = \begin{cases} 0, & m \neq n \\ \frac{2}{2n+1}, & m = n \end{cases} \tag{1.24}$$

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. $P_n(x)$ 在区间 $[-1, 1]$ 内有 n 个不同的零点.

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

- 拉瓜尔求积

积分区间 $[0, \infty)$, 权重函数 $\omega(x) = e^{-x}$, $L_n = e^x \frac{d^n}{dx^n}(x^n e^{-x})$.

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

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

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

- 厄米求积

区间 $(-\infty, \infty)$, 权重函数 $\omega(x) = e^{-x^2}$, $H_n = (-1)^n e^{x^2}\frac{d^n}{dx^n}\left(e^{-x^2}\right)$.

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

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

- 切比雪夫求积(第一类)

区间 $[-1, 1]$, 权重函数 $\omega(x) = \frac{1}{\sqrt{1-x^2}}$

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

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. T_n 的 n 个零点
$$x_k = \cos\left(\frac{2k-1}{2n}\pi\right)\ k = 1, 2, \cdots, n \tag{1.34}$$

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

- 切比雪夫求积(第二类)

区间 $[-1, 1]$, 权重函数 $\omega(x) = \sqrt{1-x^2}$

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

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

总结

类别	区间	权重函数
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	$[-1, 1]$	$\omega(x) = \sqrt{1-x^2}$

例子

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

```
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!
```

使用高斯-切比雪夫求积时需要注意，如果目标积分为 $\int_{-1}^{1} f(x)\sqrt{1-x^2}dx$, 那么对于 $2n-1$ 次函数 $f(x)$, 仅需 n 个节点即可准确计算积分. 但是对于 $\int f(x)dx$ 而言，使用切比雪夫求积需要先除以权重函数: $f(x) \to \frac{f(x)}{\sqrt{1-x^2}}$, 而此时的目标函数 $\frac{f(x)}{\sqrt{1-x^2}}$ 不是有限次数多项式！仅靠几个节点得到的积分结果误差很大.

1.2.3. 变量变换

数学物理中的很多函数是定义在无界/半无界区间上的平方可积函数 $f \in \mathbb{L}^2$. 然而计算这些函数的积分时通常使用的不是拉瓜尔或厄米积分.

以厄米求积为例，由于其积分区间是整个实数域，其零点的分布为 $-\infty < x_i < \infty$. 如果某函数恰好可以写成一个多项式函数乘以高斯函数那么使用厄米求积是非常自然的选择: $F(x) = \sum_j c_j x^j e^{-x^2}$. 但是如果不能写成这种形式，求积时需要除以高斯权重函数: $\int f(x)dx = \int \frac{f(x)}{e^{-x^2}} e^{-x^2} dx$. 由于厄米多项式的零点分布在整个实数域，分母 e^{-x^2} 可能相当小并导致 $f(x)/e^{-x^2}$ 数值不稳定. 另外，当我们需要通过除以高斯权重函数"凑出"要求的求积公式时，积分核 $f(x)/e^{-x^2}$毫无疑问已经不是多项式了(无穷级数)，有限项求和只能得到近似结果，厄米求积已经失去优越性. 增加项数可以提高精度，但是又会遇到数值稳定性问题.

但是，仍然可以使用(而且效果相当好). 例如: $f(x) = \frac{1}{x^2+1} e^{-x^2} + (x-1)^3 e^{-5(x+1)^2}$

```
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
```

另一种数值计算反常积分的方法是使用勒让德求积. 勒让德求积的优势之一是其具有最简单的权重函数($\omega(x) = 1$)，另外由于其积分区间为 $[-1, 1]$, 其零点也被限制在该范围内，数值上不会出现病态小或病态大值. 同样由于其有效区间有界，需要通过变量变换将有界区间映射到（伪）无界区间.

对于半无界区间平方可积函数的反常积分，数值求积无法处理无穷边界，但是考虑到 $f|_{x=\infty} = 0$, 反常积分的计算可以近似成上限很大的定积分的计算:

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

为了将 $(-1, 1)$ 映射到 $(0, \mathrm{Big})$, 定义（一种可能的）映射函数:

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

那么积分 $\int_0^\infty F(x)dx$ 的计算可以近似为:

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

其中 $T'(x)$ 是由于变量变换产生的雅可比行列式（1D scalar）.

例子

- 通过勒让德求积+变量映射计算 $\int_0^\infty \frac{1}{x^2+1}e^{-x^2} + (x-1)^3 e^{-5(x+1)^2}$

```
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
```

同理，相同的手段可以套用在切比雪夫求积(第二类). 与勒让德求积相比，切比雪夫求积需要考虑权重函数，但是其优势在于节点和权重的计算很简单，**且节点的分布可以从等差数列得到**:

$$
\begin{aligned}
&z = 1, 2, \cdots, N \quad \text{等距网格} \\
&x_i = \cos\left(\frac{z_i}{N+1}\pi\right) \in (1,-1) \quad \text{节点} \\
&w_i = \frac{\pi}{N+1}\sin^2\left(\frac{z_i}{N+1}\pi\right) \quad \text{权重}
\end{aligned}
\tag{1.39}
$$

使用 Becke 建议的映射函数: $x \mapsto r = r(x) = r_m \frac{1+x}{1-x} \in (\mathrm{Big}, 0)$. 其中 r_m 是元素的 Bragg-Slater 半径[1], [4]. 一方面使得切比雪夫求积可以应用于半无界区间的积分，另一方面该映射函数在原子核处密集，远离原子核处稀疏，符合电子密度的分布特点.

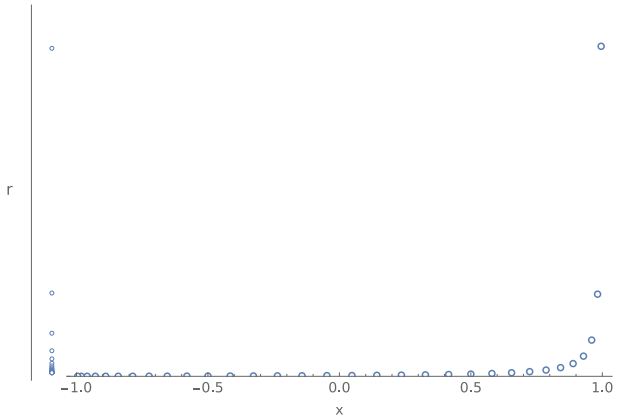


Figure 1: Becke mapping function.

$z \in [1, N]$, $x \in (1, -1)$, $r \in (\infty, 0)$. 最终变量$\{r_i | i = 1, 2, \cdots, N\}$是最初变量$\{z_i\}$的函数，所有以$r$为变量的函数经过链式法则都是变量$z$的函数. 这带来一个好处：如果需要通过有限差分计算$f(r)$对r的导数，可以通过链式法则将问题转变为计算$f(r)$对z的导数，而$\Delta z = 1$. 如果使用勒让德求积，原始变量z不是均匀网格，有限差分在非均匀网格上不好操作.

切比雪夫求积需要除以权重函数. 再考虑上变量变换带来的雅可比行列式，最终可以把这些因素全部包含进权重中：

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

注意节点是倒序排列：$\{x_i\} \in (1, -1)$. 求积时节点倒序，映射 $\{r_i\} \in (\text{Big}, 0)$ 也是倒序，权重 $\{w_i\}$ 需要与之对应.

例子

- $\int_0^\infty \frac{1}{r^2+1} e^{-r^2} + (r-1)^3 e^{-5(r+1)^2} r^2 dr$. 对一个三维函数的径向部分求积（即 1D 问题）

```
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. 球谐函数与数值求积

1.3.1. 实球谐函数

球谐函数是角动量算符($\hat{L}^2$)及其 z-分量 ($\hat{L}_z$)的特征函数:

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

球谐函数通常是复的:

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

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

且在单位球面上正交归一:

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

为了实际计算的便利性, 且微分方程解的线性叠加仍是原微分方程的解, 在实际使用中通常对球谐函数进行线性组合以消除虚部, 得到的就是所谓的实球谐函数(real spherical harmonics). 无论是复球谐函数还是实球谐函数, 他们通常都是角度(θ,φ)的函数: $Y_l^m(\Omega) = Y_l^m(\theta,\varphi)$. 角度变量可以通过反三角函数转化为直角坐标, 因此$Y_l^m(\theta,\varphi) \equiv Y_l^m(x,y,z)$.

实球谐函数的构造不唯一, 但是必须满足正交归一性: $\int_0^{2\pi}\int_0^{\pi}\left(Y_{l'}^{m'}\right)^* Y_l^m \sin\theta d\theta d\phi = \delta_{ll'}\delta_{mm'}$.

下面是几种可能的构造方法:

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

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

可能的代码实现:

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.

```
# 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++标准库中定义了勒让德多项式，可以拿来直接用：

```
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 = 1.4142135623730951454746218587388284504414;

    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. 列别杰夫求积

相较于径向求积有多种正交多项式可供选择，角度（球面）求积最常用的是列别杰夫求积[5].

- **例子**[6]

计算在单位球面上的积分: $\int_\Omega f(x,y,z)d\Omega = \int_\Omega 1+x+y^2+x^2y+x^4+y^5+x^2y^2z^2$

```
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
```

通过 Mathematica 验证一下（单位球面 $r=1$）:

```
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. 泊松方程数值解

1.4.1. 泛函一阶导数

类比于函数的导数 $f'(x)=\frac{df(x)}{dx}$, $df=f'dx$. 泛函的一阶导数为:

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

与函数导数不同，泛函导数不存在公式. 导数结果取决于泛函的具体形式. 提取积分中 δf 的系数即为泛函导数.

例子

- 计算泛函 $F[n(x)]=\int_0^1 n^2dx$ 的一阶导数:

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

例子

- 积分型泛函的一阶导数:

一般认为 δn 在端点（或边界）处固定: $\delta n=0$，因此 $\frac{\partial f}{\partial n'}\delta n\,|_{\text{边界}}=0$

$$
\begin{aligned}
&F[n]=\int f(n,n',x)dx\\
&F[n+\delta n]=\int f\left(n+\delta n,\frac{d}{dx}(n+\delta n),x\right)dx\\
&\text{泰勒展开到一次项}\\
&\Rightarrow F[n+\delta n]=\int f(n,n',x)+\left\{\frac{\partial f}{\partial n}\delta n+\frac{\partial f}{\partial n'}(\delta n)'\right\}dx\\
&F[n+\delta n]-F[n]=\delta F=\int \frac{\partial f}{\partial n}\delta n+\frac{\partial f}{\partial n'}(\delta n)'\ dx\\
&\text{对第二项进行分部积分}\\
&\Rightarrow \delta F=\int \frac{\partial f}{\partial n}\delta n\ dx+\int \frac{\partial f}{\partial n'}d(\delta n)=\int \frac{\partial f}{\partial n}\delta n\ dx+\frac{\partial f}{\partial n'}\delta n\ |_{\text{边界}}-\int \delta n\ d\left(\frac{\partial f}{\partial n'}\right)\\
&\Rightarrow \delta F=\int \frac{\partial f}{\partial n}\delta n\ dx-\int \delta n\ \frac{d}{dx}\left(\frac{\partial f}{\partial n'}\right)\ dx\\
&\delta F=\int \frac{\delta F[n]}{\delta n}\ \delta n\ dx=\int\left\{\frac{\partial f}{\partial n}-\frac{d}{dx}\left(\frac{\partial f}{\partial n'}\right)\right\}\ \delta n\ dx\\
&\therefore \frac{\delta F[n]}{\delta n}=\frac{\partial f}{\partial n}-\frac{d}{dx}\left(\frac{\partial f}{\partial n'}\right)
\end{aligned}
\tag{1.48}
$$

这是欧拉-拉格朗日方程的形式.

库伦势能 $E[n(\vec{\boldsymbol{r}})]_{\text{Coul}}$ 对电子密度的泛函导数得到库伦势 $v[n(\vec{\boldsymbol{r}})]_{\text{Coul}}=\frac{\delta E[n]}{\delta n}$:

$$
\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. 泊松方程

重要等式:

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

其中 $\frac{1}{|\vec{\boldsymbol{r}}-\vec{\boldsymbol{r}}'|}$ 是格林函数(Green's function). 角标 $\nabla_{\vec{\boldsymbol{r}}}^2$ 指的是对变量 $\vec{\boldsymbol{r}}$ 施加拉普拉斯算符.

式 (1.49) 本质上是库伦核函数对电子密度的卷积:

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

将式 (1.50) 带入式 (1.49) 可以得到:

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

由此我们得到**库伦势的积分与微分表达式**，二者等价:

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

其中库伦势的微分形式即为"泊松方程". 求解泊松方程就是在给定电子密度时计算库伦势.

1.4.3. 球谐展开

由于（实）球谐函数在单位球面上正交归一 ($\theta\in(0,\pi)$, $\phi\in(0,2\pi)$, $r\equiv 1$)，球谐函数在单位球面上构成完备集，单位球面上的平方可积函数都可以表为球谐函数的线性组合:

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

球坐标下积函数是球谐函数和高斯函数线性组合的乘积 (N.B. primitive gaussian normalized, contracted gaussian normalization not guaranteed!)：

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

数学上与笛卡尔基函数等价:

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

基函数径向部分和角度部分耦合在一起无法显式分离, 但是可以利用球谐展开将笛卡尔基函数的径向部分剥离出来:

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

式 (1.57) 中的Ω_j是单位球面上的采样点 (通过列别杰夫求积构建), w_j 是求积权重.

式 (1.57) 需要展开到无穷项才使等式成立；除非被展开函数本身就是球谐函数或球谐函数的幂，此时展开项有限，且展开系数可以通过 Clebsch-Gordan 系数解析确定. 对于一般情况, Becke 建议 l 取列别杰夫求积阶数的一半. 例如 `nleb=29 (302 angular pts)`, `lmax=14`, `nlm=225` [2].

1.4.4. 通过球谐展开和有限差分数值求解泊松方程

对电子库伦势 $U(\vec{\boldsymbol{r}})$ 和电子密度 $\rho(\vec{\boldsymbol{r}})$ 进行球谐展开:

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

带入式 (1.53)

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

每确定一组 (l,m) 得到一个二阶常微分方程:

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

问题就从求解一个二阶偏微分方程变为求解一组二阶常微分方程.

均匀网格 $\{z_i | i = 1,2,\cdots,N\}$ 通过余弦变换转变为切比雪夫求积(第二类)的节点: $\left\{ x_i = \cos\frac{z_i}{N+1}\pi \mid i = 1,2,\cdots,N \right\} \in (1,-1)$, 再通过变换映射到半无穷区间上: $\left\{ r_i = r_m \frac{1+x_i}{1-x_i} \mid i = 1,2,\cdots,N \right\} \in (\text{Big}, 0)$.

二阶微分方程需要至少两个边界条件，由 (z_0, x_0, r_0) 以及 $(z_{N+1}, x_{N+1}, r_{N+1})$ 确定 (Dirichlet condition).

根据链式法则，库伦势的径向部分是关于 r 的标量函数: $u_{lm} = u_{lm}(r)$, 因此也是关于均匀网格 $\{z_i\}$ 的标量函数: $u_{lm} = u_{lm}[r(z)]$:

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

上式等价于求解线性方程组: $\boldsymbol{A}\vec{\boldsymbol{x}} = \vec{\boldsymbol{b}}$.

$\left(\frac{dz}{dr}\right)^2$ 以及 $\frac{d^2z}{dr^2}$ 是已知量:

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

$\{z\}$ 是等距网格, 可以很自然地使用有限差分求解 $\frac{d^2u_{lm}}{dz^2}$ 和 $\frac{du_{lm}}{dz}$.

根据 ref[2] 的建议，使用 7 阶有限差分:

- **一阶导数有限差分**

采样点	微分公式	矩阵表示
−3,-2,-1,0,1,2,3	$\frac{df}{dx} \approx \frac{-f_{i-3}+9f_{i-2}-45f_{i-1}+45f_{i+1}-9f_{i+2}+f_{i+3}}{60dx}$	$\{-1\ \ 9\ \ -45\ \ 0\ \ 45\ \ -9\ \ 1\}$

- **二阶导数有限差分**

采样点	微分公式	矩阵表示
−3,-2,-1,0,1,2,3	$\frac{d^2f}{dx^2} \approx \frac{2f_{i-3}-27f_{i-2}+270f_{i-1}-490f_i+270f_{i+1}-27f_{i+2}+2f_{i+3}}{180dx^2}$	$\{2\ \ -27\ \ 270\ \ \ -490\ \ 270\ \ -27\ \ 2\}$

对于离散函数空间 $\{f_i | i = 0, 1, 2, 3, 4, \cdots,\}$, 七阶中心差分公式只能计算到 f_3'. 在边界处使用非对称的有限差分公式计算边界点的微分. 注意通过有限差分构建微分算符矩阵不是数值求解带**初始条件**的微分方程的好方法, 因为构建的微分算符很可能是奇异的, 导致微分方程对应的线性方程组解不出来. 但是通过微分算符矩阵求解带**边界条件**的微分方程问题很合适, 只需要在微分算符矩阵首尾两行需要提供两个边界条件即可.

- **边界非对称有限差分**

对于函数 $\vec{f}=\{f_1,f_2,f_3,f_4,\cdots,f_{N-3},f_{N-2},f_{N-1},f_N\}^T$, $f_4\cdots f_{N-3}$ 处的一二阶导数可由七阶中心差分计算, 对于 (f_2,f_3) 与 (f_{N-2},f_{N-1}) 处的微分可由下式计算:

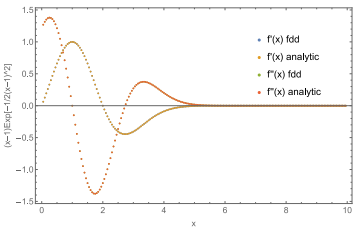
$$
D^1=\begin{cases}
\frac{df}{dx}|_{x_2} &= \frac{-3f_1-10f_2+18f_3-6f_4+f_5}{12dx}\\
\frac{df}{dx}|_{x_3} &= \frac{3f_1-30f_2-20f_3+60f_4-15f_5+2f_6}{60dx}\\
&\\
\frac{df}{dx}|_{x_{N-2}} &= \frac{-2f_{N-5}+15f_{N-4}-60f_{N-3}+20f_{N-2}+30f_{N-1}-3f_N}{60dx}\\
\frac{df}{dx}|_{x_{N-1}} &= \frac{-f_{N-4}+6f_{N-3}-18f_{N-2}+10f_{N-1}+3f_N}{12dx}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

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

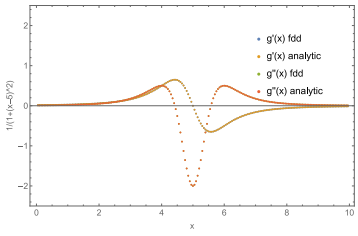
对于端点(边界) (f_1,f_N) 处使用边界条件: $\begin{cases}f(x_1)=y_1\\f(x_N)=y_N\end{cases}$, 而非计算其微分.

例子

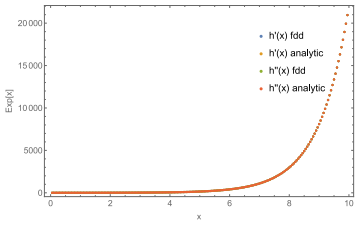
Exponential decay



Lorentian



Exponential



2. 分子 DFT 计算

Kohn-Sham 方程:

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

构建网格的目的在于数值计算交换相关泛函. 库伦势的计算同样可以基于网格数值计算.

一些符号约定:

`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. 原子网格

利用之前关于高斯求积，球谐展开，球面求积的技术，**以原子为中心构建** Gauss-Chebyshev-Lebedev **网格**.

角度网格通过列别杰夫球面采样点构建. 通过第二类切比雪夫求积构建径向网格，在每一层径向网格处，以该层半径对角度网格进行缩放，然后以原子位置为原点进行平移. 这样就得到了一层原子网格，最终共有 `nrad` 层原子网格，越接近原子核网格越密集，总网格数为 `natgrid=nrad x nang`. 每个网格有自己的笛卡尔坐标，原子网格可以通过矩阵形式表示:

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

三维函数在网格上的函数值则是 `(nrad, nang)` 维度的矩阵: $f(\{\vec{r}_{ij}\}) = f(\{x_{ij}, y_{ij}, z_{ij}\}) = A_{ij}$.

将径向雅可比行列式并入径向权重，通过与角度网格的权重做向量外积将总权重也组织成矩阵形式:

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

遍历径向网格，在每一层径向网格通过列别杰夫求积计算角度积分，然后乘以径向权重 w_i 和径向雅可比行列式 r_i^2 并求和，就可以数值计算三维函数对整个实空间的积分:

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

其中最后一行是爱因斯坦求和约定.

更有效的方法是将函数值矩阵 A_{ij} 和权重矩阵 w_{atgrid} 逐元素相乘然后求和.

例子

- 计算 sto-3g 基组下碳原子 `2px` 轨道的自重叠积分: $\phi(x,y,z;ax,ay,az) = \sum_i^3 c_i n_i (x-ax) e^{-\alpha_i[(x-ax)^2+(y-ay)^2+(z-az)^2]}$.

事实上原子的位置不影响计算结果，因为对于该原子的原子网格而言，原子的位置就是坐标原点，所以无论原子实际位于哪里，等价于原子位于坐标原点.

计算代码如下:

```
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
```

利用相同的方法可以**在原子网格上计算算符矩阵元**.

如果某算符（微分算符除外）在原子网格上的函数值（通过某些手段）已知, 其可表示成矩阵形式: $\hat{O} = O(\vec{r}) \approx O(\{\vec{r}_{ij}\}) = O_{ij}$.

基函数在原子网格上的函数值也可以表示成矩阵形式: $\{\phi_\mu(\vec{r}) \mid \mu = 1, 2, \cdots\} \approx \{\phi_{ij}^\mu \mid \mu = 1, 2, \cdots\}$.

矩阵元不过是一个积分:

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

基函数 ϕ_μ 的中心在原子 C, ϕ_ν 的中心在原子 B, 要计算原子 A 的网格上的矩阵元只需要计算 ϕ_μ, ϕ_ν, $\hat{O}$ 在原子 A 网格上的函数值. 基函数的中心不需要与网格中心一致.

2.2. Becke 分子网格[1]

对于多原子分子, 空间中任意点同时受到所有原子的影响，区别在于有的原子影响很大，有的原子的影响可忽略. 即，原子在空间各点处有权重.

引入原子权重函数（不同于 w_i, w_j !）: $w_a(\vec{r})$. 且 $\sum_a^{\text{natom}} w_a(\vec{r}) \equiv 1$. 即在空间中任意一点各原子的权重相加为一.

在 `natom` 系统中对某函数进行积分:

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

这样就把对整个分子的积分转化为对组成分子的各个原子的积分:

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

权重函数 $\{w_a(\vec{r}) \mid a = 1, 2, \cdots, \text{natom}\}$ 的引入相当于在空间中按原子划分边界，要求:

1. 不能是刚性边界，边界处原子权重应光滑过渡 (not step function)
2. 边界不能过于光滑，否则无法区分原子的贡献 (switching function)

是一种边界“模糊”的沃罗诺伊原胞 (fuzzy Voronoi cells).

定义椭圆坐标: $\mu_{ij} = \frac{r_i - r_j}{R_{ij}} \in [-1, 1]$. 其中 r_i 是空间点到原子 i 的距离，r_j 是空间点到原子 j 的距离，R_{ij} 是核间距.

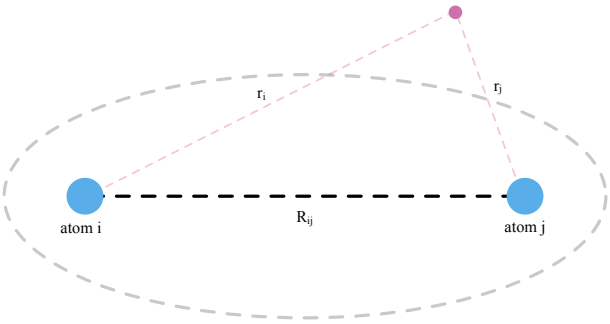


Figure 2: Confocal elliptical coordinate system.

定义 cutoff profile: $s(\mu_{ij})$. 要求:

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

即要求其类似于阶梯函数，但是没有阶跃，且在原子核处连续(no cusp). 满足条件的 $s(\mu)$ 的定义不唯一.

在定义一个函数 $f(\mu)$, 满足:

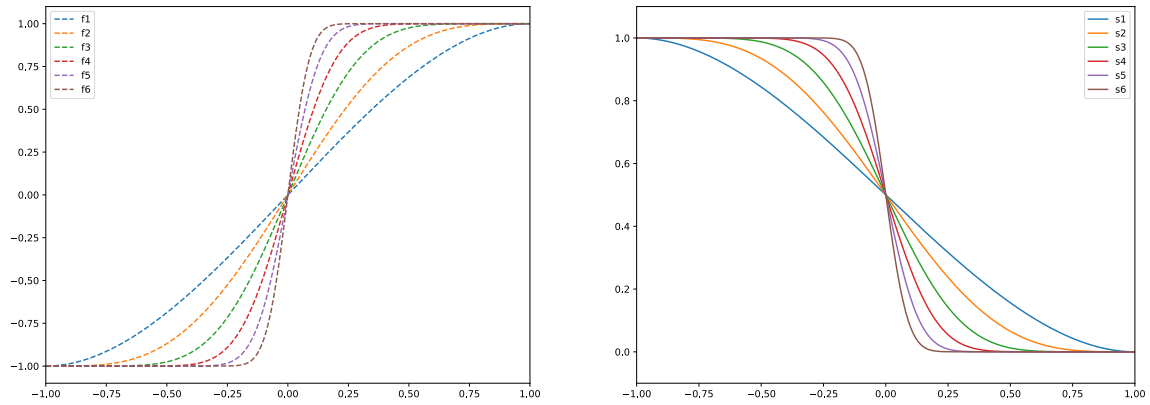
1. $f(-1) = -1$
2. $f(1) = 1$
3. $\frac{df}{d\mu}|_{-1} = \frac{df}{d\mu}|_1 = 0$

$s(\mu)$ 通过 $f(\mu)$ 得到: $s(\mu) = \frac{1}{2}[1 - f(\mu)]$.

一种满足上述要求的函数: $p(\mu) = f(\mu) = \frac{3}{2}\mu - \frac{1}{2}\mu^3$. 但是该函数过于光滑，不能很好的区分边界. 对 $p(\mu)$ 进行迭代可以系统地提高区分度:

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

当迭代次数非常高时 s_k 的极限即为阶梯函数.



最后权重函数定义为:

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

Becke **网格核心思想：先处理原子网格(带权)，再遍历所有原子**.

2.3. 库伦势的构建

2.3.1. 通过电子排斥积分解析构建

同波函数方法:

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

通过解析计算电子排斥积分 ERI 和密度矩阵的张量乘法得到库伦势.

2.3.2. 通过数值求解泊松方程构建[2]

本节结合小节 1.4 一同食用效果更佳.

待求解的问题:

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

上式等价于求解线性方程组: $A\vec{x} = \vec{b}$. 其中二阶微分算符和一阶微分算符都是七对角矩阵, $\frac{l(l+1)}{r^2}$ 是对角矩阵, z 对 r 的导数是常数.

二阶 ODE 需要两个边界条件，共有 `nrad` 径向网格, 所以矩阵 A 的维度为 `(nrad+2, nrad+2)`. ρ_{lm} 通过对电子密度球谐展开得到，维度为 `(nrad+2, )`. 首尾两个元素分别是 $r \to \infty$ 和 $r \to 0$ 时的边界条件 (Dirichlet boundary condition). 注意 $\{r_i\}$ 是倒序排列的.

- **边界条件**

当 $r \to \infty$，无论多么复杂的电荷分布都可以看作是点电荷，所以 $\lim_{r\to\infty} U(\vec{r}) = U(r) = \frac{q}{r} = \sum_{lm} \frac{u_{lm}}{r} Y_{lm}$. 此时电势仅仅是标量 r 的函数，所以求和中只有 Y_{00} 的系数不为零：$\frac{q}{r} = \frac{u_{00}}{r}\sqrt{\frac{1}{4\pi}} \Rightarrow u_{00} = \sqrt{4\pi}q$.

当 $r \to 0$, $u_{lm} = 0$

N.B. 这里的 q 是通过电子密度积分后得到 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
```

- **球谐基**

定义球谐基 $Y_{lm,j} = Y_{kj}$ `shape (nlm, nang)`. 所有原子共用一套, 角度采样点取列别杰夫求积采样点.

```
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
```

- **电子密度球谐展开**

利用球谐函数的正交归一性:

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

ρ_{ij} 是电子密度在原子 A 网格上的值(矩阵形式)，乘以原子 A 在自己网格上的 Becke 权重 w_{ij}^a 得到 ρ_{ij}^a. 通过矩阵乘法球谐基得到电子密度球谐展开系数 ρ_{ik} `shape (nrad, nlm)`

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

- **原子网格上的库伦势**

得到 ρ_{ik}^a 后通过有限差分和求解线性方程组的得到**原子网格上库伦势的球谐展开系数**：$\rho_{ik}^a \to u_{ik}^a$.

此时得到的库伦势是“局域”的. 如 Figure 3 所示，目标点不在原子 a 的网格上，但是原子 a 对该点的库伦势的贡献大于产生该点的原子 b. 如果不考虑原子 a 的贡献，最终该点处的库伦势会被大大低估. 又因为该点不是原子 a 的网格点，无法通过求解泊松方程得到原子 a 在该点处的库伦势，需要通过插值计算.

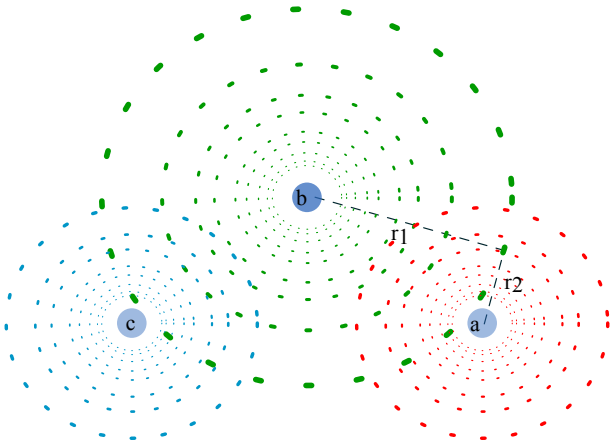


Figure 3: Numerically build Hartree potential on the grids.

由于 $u_{lm}(r)$ 是标量 r 的函数，很自然地想到对 r 进行插值. 但是 r 是非均匀网格，r 越大网格间距越大，会导致在大 r 处三次样条插值结果很不准确.

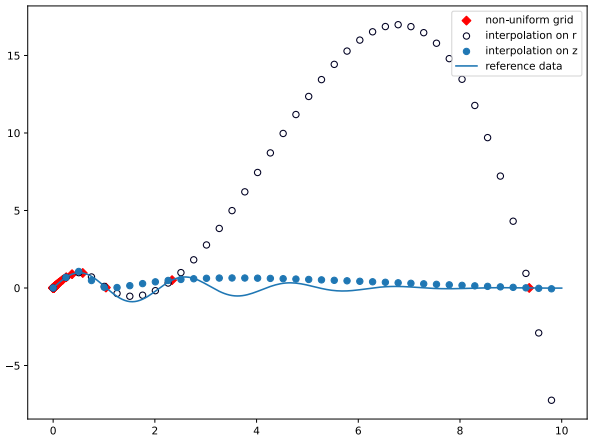


Figure 4: Cubic spline interpolation.

- **插值构建整个网格上的库伦势**

如[2] 建议的那样，使用均匀网格 $\{z\}$ 进行插值. 注意到对所有原子 $\{z\}$ 是一样的（$\{r\}$的缩放因子取决于具体的元素）, 这是使用切比雪夫径向网格的另一个优势.

```
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]))
```

为每一组 (l,m) 构建插值器后，需要对**所有**网格进行插值，这样构建的就不再是“局域”的库伦势，而是“全局”库伦势. 因此需要计算所有原子到所有网格的距离：

```
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
```

然后通过三次样条插值计算原子 i 对所有网格点的库伦势贡献:

```
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)
```

再通过反球谐展开重构出原子 i 产生的全局库伦势：$U^a_{\text{ngrid}} = \sum_k u^{\text{a interp}}_{k,\text{ngrid}}/\text{dist}^a \cdot y^a_{k,\text{ngrid}}$. 其中$y^a_{k,\text{ ngrid}}$ 需要计算原子 i 到所有网格点处固体角，然后计算这些固体角在各 (l, m) 处的球谐函数值:

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

    return ylm
```

最终计算：

```
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)
```

插值重构出的库伦势遍历了所有原子，**考虑了所有原子在整个网格上的库伦势的贡献**. 因此这里得到是**全局库伦势**.

`u_ij shape (ngrid = natomxnradxnang, )`. 这样做为了方便后续计算算符矩阵元.

- **库伦矩阵元**

式 (2.5) 在原子网格上计算矩阵元. 为了方便计算算符在整个分子网格上的矩阵元，将矩阵表示的各种量展平：`u_ij -> shape (natomxnradxnang,)`.

矩阵元的计算需要计算基函数在网格上的函数值. 先将基函数组织成多维数组：$\left\{\phi_{a,ij}^{\mu} \mid \mu = 1,2,\cdots,\mathrm{nbf} \mid a = 1,2,\cdots,\mathrm{natom}\right\}$. 按照原子拉平，组织成 `(nbf x ngrid)` 维度矩阵：$\left\{\phi_{\mu,\,\mathrm{ngrid}} \mid \mu = 1,2,\cdots,\mathrm{nbf}\right\}$.

同时将所有的权重以及径向雅可比行列式组织到一起并拉平：

```
# 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, )
```

这样就可以计算库伦算符矩阵:

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

代码表示:

```
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)
```

这样，就通过数值方法得到了库伦算符矩阵.

2.4. 交换相关泛函数值求积

交换相关算符矩阵元的计算通常通过数值求积获得，因为交换泛函和相关泛函的表达式与基函数在实空间的解析积分极难计算.

交换相关泛函是电子密度的泛函，因此首先需要计算电子密度在分子网格上的值:

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

矩阵形式:

```
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]
```

通过电子密度在整个实空间上的积分可以重构出电子数: $\mathrm{ne} = \int \rho(\boldsymbol{r}) d^3\boldsymbol{r} =$ `np.sum(rho * mweights)`.

通过使用相同的手段，就可以得到交换相关泛函势矩阵元和能量密度矩阵元. 给定电子密度后，交换相关势和能量密度直接通过调用 `libxc` 计算:

```
#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 矩阵

各种算符矩阵计算完毕后直接将他们加起来就得到 Fock 矩阵:

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

利用密度矩阵计算各种能量组分的期望值: $\langle \hat{O} \rangle = \mathrm{Tr}\{PO\}$. 最终得到能量分解:

```
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. 笛卡尔基函数矩阵变换到纯基函数

笛卡尔基函数的排序通常按照字典顺序，详情见小节 1.1.2. 而纯基函数的排序通常有两种:

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 $m = -l, -l+1, \cdots, l-1, l$.

纯基函数通过笛卡尔基函数线性组合得到，具体计算中则是先在笛卡尔基函数下得到结果（积分，函数值等），然后通过转换矩阵乘法得到纯基函数下的结果.

HORTON 约定下的转换矩阵见 https://github.com/theochem/iodata/blob/main/tools/harmonics.py.

例如，笛卡尔基函数(CCA ordering)在分子网格上的函数值组织成矩阵形式：`aos_vals (nbf, ngrid)`. 根据基函数的角动量 l 构建块对角矩阵:

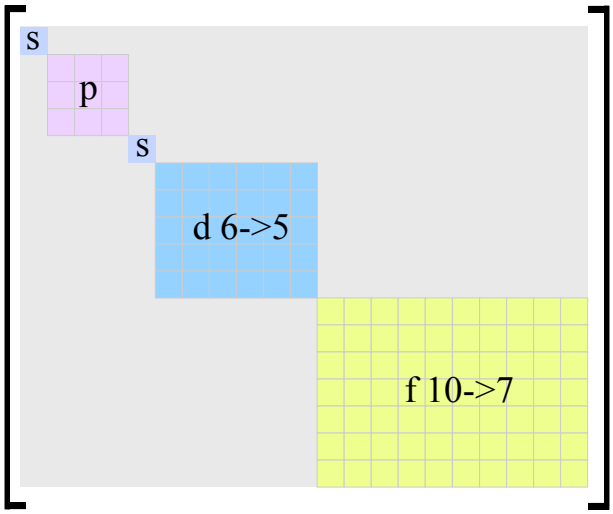


Figure 5: Cartesian to pure transformation matrix.

然后与 `aos_vals` 做矩阵乘法即得到纯基函数在网格上的函数值: `spsdf 21 -> 17`.

至于算符矩阵同理. 注意转换矩阵高度稀疏.

下面列出 `lmax=3` 的转换矩阵 (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
```

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

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

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

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

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:

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

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.