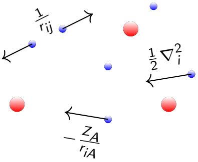
Hartree–Fock em `Python`

Yuri Alexandre Aoto

Universidade Federal do ABC, UFABC

SeedMol 2022

Sistema de N_e elétrons em um campo formado por M núcleos



$$
\begin{aligned}
\hat{H} &= -\frac{1}{2}\sum_{i=1}^{N_e}\nabla_i^2 - \sum_{i=1}^{N_e}\sum_{A=1}^{M}\frac{Z_A}{r_{iA}} + \sum_{i=1}^{N_e}\sum_{j=i+1}^{N_e}\frac{1}{r_{ij}} \\
&= \sum_{i=1}^{N_e}\left(-\frac{1}{2}\nabla_i^2 - \sum_{A=1}^{M}\frac{Z_A}{r_{iA}}\right) + \sum_{i=1}^{N_e}\sum_{j=i+1}^{N_e}\frac{1}{r_{ij}} \\
&= \sum_{i}\hat{h}_i + \sum_{j<i}\frac{1}{r_{ij}}
\end{aligned}
$$

A função de onda do estado fundamental

$$\hat{H}\Psi = E\Psi$$

$$E = \frac{\langle\Psi|\hat{H}|\Psi\rangle}{\langle\Psi|\Psi\rangle} = \frac{\int \Psi^* \hat{H}\Psi d\mathbf{x}}{\int \Psi^*\Psi d\mathbf{x}}$$

$$E = \min_{\text{todas } \Phi} \frac{\langle\Phi|\hat{H}|\Phi\rangle}{\langle\Phi|\Phi\rangle}$$

Função de onda tentativa, o "*ansatz*"

$$\Psi_{HF} = |\phi_1\phi_2\cdots\phi_{N_e}\rangle = \frac{1}{\sqrt{N_e!}}\begin{vmatrix} \phi_1(\mathbf{x}_1) & \phi_2(\mathbf{x}_1) & \cdots & \phi_{N_e}(\mathbf{x}_1) \\ \phi_1(\mathbf{x}_2) & \phi_2(\mathbf{x}_2) & \cdots & \phi_{N_e}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\mathbf{x}_{N_e}) & \phi_2(\mathbf{x}_{N_e}) & \cdots & \phi_{N_e}(\mathbf{x}_{N_e}) \end{vmatrix}$$

$$E_{HF} = \min \frac{\langle\phi_1\phi_2\cdots\phi_{N_e}|\hat{H}|\phi_1\phi_2\cdots\phi_{N_e}\rangle}{\langle\phi_1\phi_2\cdots\phi_{N_e}|\phi_1\phi_2\cdots\phi_{N_e}\rangle}$$

$$E = \sum_{a=1}^{N_e}\int \phi_a^*(\mathbf{x})\hat{h}\phi_a(\mathbf{x})d\mathbf{x} + \frac{1}{2}\sum_{a=1}^{N_e}\sum_{b=1}^{N_e}\iint \phi_a^*(\mathbf{x}_1)\phi_b^*(\mathbf{x}_2)\frac{1}{r_{12}}\phi_a(\mathbf{x}_1)\phi_b(\mathbf{x}_2)d\mathbf{x}_1 d\mathbf{x}_2 - \iint \phi_a^*(\mathbf{x}_1)\phi_b^*(\mathbf{x}_2)\frac{1}{r_{12}}\phi_b(\mathbf{x}_1)\phi_a(\mathbf{x}_2)d\mathbf{x}_1 d\mathbf{x}_2$$

$$\int \phi_i\phi_j d\mathbf{x} = \delta_{ij} = \begin{cases} 1 & \text{se } i = j \\ 0 & \text{se } i \neq j \end{cases}$$

Qual critério cada ϕ_j deve satisfazer?

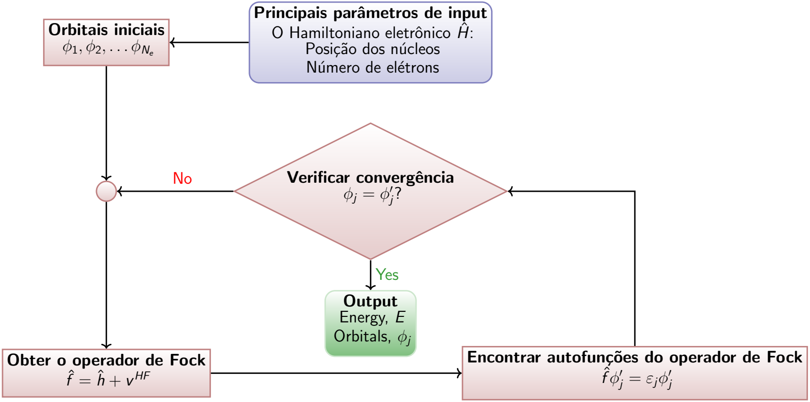
$$\hat{f}\phi_j = \varepsilon_j \phi_j$$

$$\hat{f}\phi_j = \hat{h}\phi_j + v^{HF}\phi_j$$

$$\hat{h}\phi_j(\mathbf{x}) = -\frac{1}{2}\nabla^2\phi_j(\mathbf{x}) - \sum_{A=1}^{M}\frac{Z_A}{|\mathbf{x} - R_A|}\phi_j(\mathbf{x})$$

$$\begin{aligned} v^{HF}(\mathbf{x}_1)\phi_j(\mathbf{x}_1) &= \sum_{a=1}^{N_e}\left\{J_a(\mathbf{x}_1) - K_a(\mathbf{x}_1)\right\}\phi_j(\mathbf{x}_1) \\ &= \sum_{a=1}^{N_e}\left\{\left(\int d\mathbf{x}_2 \phi_a^*(\mathbf{x}_2)\frac{1}{r_{12}}\phi_a(\mathbf{x}_2)\right)\phi_j(\mathbf{x}_1)\right. \\ &\qquad \left. - \left(\int d\mathbf{x}_2 \phi_a^*(\mathbf{x}_2)\frac{1}{r_{12}}\phi_j(\mathbf{x}_2)\right)\phi_a(\mathbf{x}_1)\right\} \end{aligned}$$

O algoritmo: ideia geral



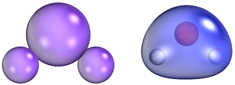
Como escolher os orbitais ϕ_i? Expandir em uma base χ_i

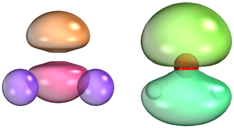
χ_i : Conhecidos e definidos no começo de um cálculo.
Exemplos: 6-31G, 6-311G(d,p), cc-pVDZ, etc

$$\phi_j = \sum_{i=1}^{n} C_{ij}\chi_i$$

Exemplos:

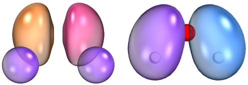
$\phi = 0.7\chi_{2s} + 0.6\chi_{1s_{H_1}} + 0.6\chi_{1s_{H_2}}$



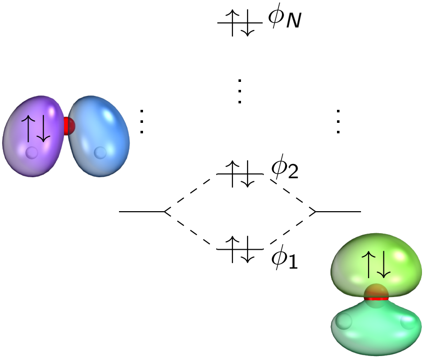


$\phi = 0.7\chi_{2p_z} + 0.6\chi_{1s_{H_1}} + 0.6\chi_{1s_{H_2}}$

$\phi = 0.7\chi_{2p_x} + 0.6\chi_{1s_{H_1}} - 0.6\chi_{1s_{H_2}}$



Hartree-Fock restrito de camada fechada



Hartree-Fock restrito de camada fechada

$$
\begin{aligned}
v^{HF}(\mathbf{x}_1)\phi_j(\mathbf{x}_1) &= \sum_{a=1}^{N_e} \left\{ J_a(\mathbf{x}_1) - K_a(\mathbf{x}_1) \right\} \phi_j(\mathbf{x}_1) \\
&= \sum_{a=1}^{N} \left\{ J_{a_\alpha}(\mathbf{x}_1) - K_{a_\alpha}(\mathbf{x}_1) \right\} \phi_j(\mathbf{x}_1) \\
&\quad + \sum_{a=1}^{N} \left\{ J_{a_\beta}(\mathbf{x}_1) - K_{a_\beta}(\mathbf{x}_1) \right\} \phi_j(\mathbf{x}_1) \\
&= \sum_{a=1}^{N} \left\{ J_{a_\alpha}(\mathbf{x}_1)\phi_j(\mathbf{x}_1) - K_{a_\alpha}(\mathbf{x}_1)\phi_j(\mathbf{x}_1) \right. \\
&\qquad \left. + J_{a_\beta}(\mathbf{x}_1)\phi_j(\mathbf{x}_1) - K_{a_\beta}(\mathbf{x}_1)\phi_j(\mathbf{x}_1) \right\}
\end{aligned}
$$

Hartree-Fock restrito de camada fechada

$$J_{a_\alpha}(\mathbf{x}_1)\phi_j(\mathbf{x}_1) = \left(\int d\mathbf{x}_2 \phi^*_{a_\alpha}(\mathbf{x}_2)\frac{1}{r_{12}}\phi_{a_\alpha}(\mathbf{x}_2)\right)\phi_j(\mathbf{x}_1)$$

$$J_{a_\beta}(\mathbf{x}_1)\phi_j(\mathbf{x}_1) = \left(\int d\mathbf{x}_2 \phi^*_{a_\beta}(\mathbf{x}_2)\frac{1}{r_{12}}\phi_{a_\beta}(\mathbf{x}_2)\right)\phi_j(\mathbf{x}_1)$$

$$K_{a_\alpha}(\mathbf{x}_1)\phi_j(\mathbf{x}_1) = \left(\int d\mathbf{x}_2 \phi^*_{a_\alpha}(\mathbf{x}_2)\frac{1}{r_{12}}\phi_j(\mathbf{x}_2)\right)\phi_{a_\alpha}(\mathbf{x}_1)$$

$$K_{a_\beta}(\mathbf{x}_1)\phi_j(\mathbf{x}_1) = \left(\int d\mathbf{x}_2 \phi^*_{a_\beta}(\mathbf{x}_2)\frac{1}{r_{12}}\phi_j(\mathbf{x}_2)\right)\phi_{a_\beta}(\mathbf{x}_1)$$

Hartree-Fock restrito de camada fechada

$$
\begin{aligned}
v^{HF}(\mathbf{x}_1)\phi_j(\mathbf{x}_1) &= \sum_{a=1}^{N} \{J_{a_\alpha}(\mathbf{x}_1)\phi_j(\mathbf{x}_1) - K_{a_\alpha}(\mathbf{x}_1)\phi_j(\mathbf{x}_1) \\
&\quad + J_{a_\beta}(\mathbf{x}_1)\phi_j(\mathbf{x}_1) - K_{a_\beta}(\mathbf{x}_1)\phi_j(\mathbf{x}_1)\} \\
&= \sum_{a=1}^{N} \{2J_a(\mathbf{x}_1) - K_a(\mathbf{x}_1)\}\,\phi_j(\mathbf{x}_1)
\end{aligned}
$$

$$
\hat{f} = \hat{h} + v^{HF} = \hat{h} + \sum_{a=1}^{N} 2J_a - K_a
$$

Como escolher os orbitais ϕ_i? Expandir em uma base χ_i

$$N = \frac{N_e}{2} = \text{número de orbitais espaciais}$$

$$\Psi_{HF} = |(\phi_1\alpha)(\phi_1\beta)(\phi_2\alpha)(\phi_2\beta)\cdots(\phi_N\alpha)(\phi_N\beta)\rangle$$

$$\phi_1, \phi_2, \ldots, \phi_N \to \Psi_{HF}$$

$$\phi_j = \sum_{i=1}^{n} C_{ij}\chi_i \qquad C_{ij} \in \mathbb{R}, \chi_i(\mathbf{x}) \in \mathbb{R}$$

$$C_{ij},\ i \in \{1, 2, \ldots, n\},\ j \in \{1, \ldots, N\} \to \Psi_{HF}$$

$$\begin{pmatrix} C_{11} & \cdots & C_{1N} \\ C_{21} & \cdots & C_{2N} \\ \vdots & \ddots & \vdots \\ C_{n1} & \cdots & C_{nN} \end{pmatrix}$$

A equação de Fock em uma base: Equações de Roothaan

$$\hat{f}\phi_j = \varepsilon_j \phi_j$$

$$\hat{f}\sum_{i=1}^{n} C_{ij}\chi_i = \varepsilon_j \sum_{i=1}^{n} C_{ij}\chi_i$$

$$\int \chi_k \left(\hat{f}\sum_{i=1}^{n} C_{ij}\chi_i \right) d\mathbf{x} = \int \chi_k \left(\varepsilon_j \sum_{i=1}^{n} C_{ij}\chi_i \right) d\mathbf{x}$$

$$\sum_{i=1}^{n} \left(\int \chi_k \hat{f} \chi_i d\mathbf{x} \right) C_{ij} = \varepsilon_j \sum_{i=1}^{n} \left(\int \chi_k \chi_i d\mathbf{x} \right) C_{ij}$$

$$\sum_{i=1}^{n} F_{ki} C_{ij} = \varepsilon_j \sum_{i=1}^{n} S_{ki} C_{ij}$$

A equação de Fock em uma base: Equações de Roothaan

$$\sum_{i=1}^{n} F_{ki} C_{ij} = \varepsilon_j \sum_{i=1}^{n} S_{ki} C_{ij}$$

$$\mathbf{FC}_j = \varepsilon_j \mathbf{SC}_j$$

$$\begin{pmatrix} F_{11} & F_{12} & \cdots & F_{1n} \\ F_{21} & F_{22} & \cdots & F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ F_{n1} & F_{n2} & \cdots & F_{nn} \end{pmatrix} \begin{pmatrix} C_{1j} \\ C_{2j} \\ \vdots \\ C_{nj} \end{pmatrix} = \varepsilon_j \begin{pmatrix} S_{11} & S_{12} & \cdots & S_{1n} \\ S_{21} & S_{22} & \cdots & S_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ S_{n1} & S_{n2} & \cdots & S_{nn} \end{pmatrix} \begin{pmatrix} C_{1j} \\ C_{2j} \\ \vdots \\ C_{nj} \end{pmatrix}$$

A equação de Fock em uma base: Equações de Roothaan

$$\sum_{i=1}^{n} F_{ki} C_{ij} = \varepsilon_j \sum_{i=1}^{n} S_{ki} C_{ij}$$

$$\mathbf{FC} = \mathbf{SC}\varepsilon$$

$$\begin{pmatrix} F_{11} & F_{12} & \cdots & F_{1n} \\ F_{21} & F_{22} & \cdots & F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ F_{n1} & F_{n2} & \cdots & F_{nn} \end{pmatrix} \begin{pmatrix} C_{11} & \cdots & C_{1N} \\ C_{21} & \cdots & C_{2N} \\ \vdots & \ddots & \vdots \\ C_{n1} & \cdots & C_{nN} \end{pmatrix}$$

$$= \begin{pmatrix} S_{11} & S_{12} & \cdots & S_{1n} \\ S_{21} & S_{22} & \cdots & S_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ S_{n1} & S_{n2} & \cdots & S_{nn} \end{pmatrix} \begin{pmatrix} C_{11} & \cdots & C_{1N} \\ C_{21} & \cdots & C_{2N} \\ \vdots & \ddots & \vdots \\ C_{n1} & \cdots & C_{nN} \end{pmatrix} \begin{pmatrix} \varepsilon_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \varepsilon_N \end{pmatrix}$$

A equação de Fock em uma base: Equações de Roothaan

$$
\begin{aligned}
F_{ij} &= \int \chi_i \hat{f} \chi_j d\mathbf{x} \\
&= \int \chi_i \left(\hat{h} + v^{HF} \right) \chi_j d\mathbf{x} \\
&= \int \chi_i \hat{h} \chi_j d\mathbf{x} + \int \chi_i v^{HF} \chi_j d\mathbf{x} \\
&= h_{ij} + v_{ij}^{HF}
\end{aligned}
$$

$$
\mathbf{F} = \mathbf{h} + \mathbf{v}^{\mathbf{HF}}
$$

A equação de Fock em uma base: Equações de Roothaan

$$
\begin{aligned}
v_{ij}^{HF} &= \int \chi_i v^{HF} \chi_j d\mathbf{x} \\
&= \int \chi_i \left(\sum_{a=1}^{N} 2J_a - K_a \right) \chi_j d\mathbf{x} \\
&= \sum_{a=1}^{N} 2 \int \chi_i J_a \chi_j d\mathbf{x} - \int \chi_i K_a \chi_j d\mathbf{x}
\end{aligned}
$$

A equação de Fock em uma base: Equações de Roothaan

$$
\begin{aligned}
\int \chi_i J_a \chi_j d\mathbf{x} &= \int \chi_i \left\{ \int \phi_a(\mathbf{x}_2) \frac{1}{r_{12}} \phi_a(\mathbf{x}_2) d\mathbf{x}_2 \right\} \chi_j d\mathbf{x}_1 \\
&= \int \chi_i \left\{ \left(\sum_{k=1}^{n} C_{ka} \chi_k(\mathbf{x}_2) \right) \frac{1}{r_{12}} \left(\sum_{l=1}^{n} C_{la} \chi_l(\mathbf{x}_2) \right) d\mathbf{x}_2 \right\} \chi_j d\mathbf{x}_1 \\
&= \sum_{k=1}^{n} \sum_{l=1}^{n} C_{ka} C_{la} \int \chi_i \left\{ \int \chi_k(\mathbf{x}_2) \frac{1}{r_{12}} \chi_l(\mathbf{x}_2) d\mathbf{x}_2 \right\} \chi_j d\mathbf{x}_1 \\
&= \sum_{k=1}^{n} \sum_{l=1}^{n} C_{ka} C_{la} \iint \chi_i(\mathbf{x}_1) \chi_j(\mathbf{x}_1) \frac{1}{r_{12}} \chi_k(\mathbf{x}_2) \chi_l(\mathbf{x}_2) d\mathbf{x}_1 d\mathbf{x}_2 \\
&= \sum_{k=1}^{n} \sum_{l=1}^{n} C_{ka} C_{la} (ij|kl) \\
&= \sum_{k=1}^{n} \sum_{l=1}^{n} C_{ka} C_{la} g_{ijkl}
\end{aligned}
$$

A equação de Fock em uma base: Equações de Roothaan

$$
\begin{aligned}
\int \chi_i K_a \chi_j d\mathbf{x} &= \int \chi_i \left\{ \int \phi_a(\mathbf{x}_2) \frac{1}{r_{12}} \chi_j(\mathbf{x}_2) d\mathbf{x}_2 \right\} \phi_a d\mathbf{x}_1 \\
&= \int \chi_i \left\{ \left(\sum_{k=1}^{n} C_{ka} \chi_k(\mathbf{x}_2) \right) \frac{1}{r_{12}} \chi_j(\mathbf{x}_2)) d\mathbf{x}_2 \right\} \left(\sum_{l=1}^{n} C_{la} \chi_l \right) d\mathbf{x}_1 \\
&= \sum_{k=1}^{n} \sum_{l=1}^{n} C_{ka} C_{la} \int \chi_i \left\{ \int \chi_k(\mathbf{x}_2) \frac{1}{r_{12}} \chi_j(\mathbf{x}_2) d\mathbf{x}_2 \right\} \chi_l d\mathbf{x}_1 \\
&= \sum_{k=1}^{n} \sum_{l=1}^{n} C_{ka} C_{la} \iint \chi_i(\mathbf{x}_1) \chi_l(\mathbf{x}_1) \frac{1}{r_{12}} \chi_k(\mathbf{x}_2) \chi_j(\mathbf{x}_2) d\mathbf{x}_1 d\mathbf{x}_2 \\
&= \sum_{k=1}^{n} \sum_{l=1}^{n} C_{ka} C_{la} (il|kj) \\
&= \sum_{k=1}^{n} \sum_{l=1}^{n} C_{ka} C_{la} g_{ilkj}
\end{aligned}
$$

A equação de Fock em uma base: Equações de Roothaan

$$
\begin{aligned}
v_{ij}^{HF} &= \sum_{a=1}^{N} 2\int \chi_i J_a \chi_j d\mathbf{x} - \int \chi_i K_a \chi_j d\mathbf{x} \\
&= \sum_{a=1}^{N} 2\left(\sum_{k=1}^{n}\sum_{l=1}^{n} C_{ka} C_{la} g_{ijkl}\right) - \left(\sum_{k=1}^{n}\sum_{l=1}^{n} C_{ka} C_{la} g_{ilkj}\right) \\
&= \sum_{k=1}^{n}\sum_{l=1}^{n}\left(\sum_{a=1}^{N} 2C_{ka}C_{la}\right) g_{ijkl} - \frac{1}{2}\sum_{k=1}^{n}\sum_{l=1}^{n}\left(\sum_{a=1}^{N} 2C_{ka}C_{la}\right) g_{ilkj} \\
&= \sum_{k=1}^{n}\sum_{l=1}^{n} P_{kl} g_{ijkl} - \frac{1}{2}\sum_{k=1}^{n}\sum_{l=1}^{n} P_{kl} g_{ilkj} \\
&= \sum_{k=1}^{n}\sum_{l=1}^{n} P_{kl}\left(g_{ijkl} - \frac{1}{2} g_{ilkj}\right)
\end{aligned}
$$

A equação de Fock em uma base: Equações de Roothaan

$$P_{kl} = 2\sum_{a=1}^{N} C_{ka}C_{la}$$

$$\mathbf{P} = \begin{pmatrix} P_{11} & P_{12} & \cdots & P_{1n} \\ P_{21} & P_{22} & \cdots & P_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{n1} & P_{n2} & \cdots & P_{nn} \end{pmatrix} = 2\begin{pmatrix} C_{11} & \cdots & C_{1N} \\ C_{21} & \cdots & C_{2N} \\ \vdots & \ddots & \vdots \\ C_{n1} & \cdots & C_{nN} \end{pmatrix} \begin{pmatrix} C_{11} & C_{21} & \cdots & C_{n1} \\ \vdots & \vdots & \ddots & \vdots \\ C_{1N} & C_{2N} & \cdots & C_{nN} \end{pmatrix}$$

$$\mathbf{P} = 2\mathbf{C}\mathbf{C}^{T}$$

A equação de Fock em uma base: Equações de Roothaan

$$\mathbf{FC} = \mathbf{SC}\varepsilon$$

$$\sum_{i=1}^{n} F_{ki} C_{ij} = \sum_{i=1}^{n} S_{ki} C_{ij} \varepsilon_j$$

$$F_{ij} = h_{ij} + v_{ij}^{HF} = h_{ij} + \sum_{k=1}^{n} \sum_{l=1}^{n} P_{kl} \left(g_{ijkl} - \frac{1}{2} g_{ilkj} \right)$$

$$S_{ij} = \int \chi_i(\mathbf{x}) \chi_j(\mathbf{x}) d\mathbf{x} \qquad h_{ij} = \int \chi_i(\mathbf{x}) \hat{h} \chi_j(\mathbf{x}) d\mathbf{x}$$

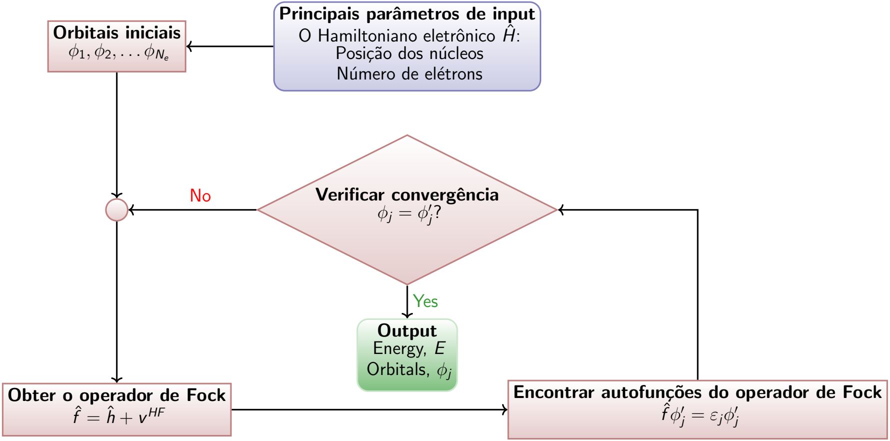
$$g_{ijkl} = (ij|kl) = \iint \chi_i(\mathbf{x}_1) \chi_j(\mathbf{x}_1) \frac{1}{r_{12}} \chi_k(\mathbf{x}_2) \chi_l(\mathbf{x}_2) d\mathbf{x}_1 d\mathbf{x}_2$$

A equação de Fock em uma base: Equações de Roothaan

$$E = \sum_{a=1}^{N_e} \int \phi_a \hat{h} \phi_a d\mathbf{x} + \frac{1}{2} \sum_{a=1}^{N_e} \sum_{b=1}^{N_e} \iint \phi_a(\mathbf{x}_1)\phi_b(\mathbf{x}_2) \frac{1}{r_{12}} \phi_a(\mathbf{x}_1)\phi_b(\mathbf{x}_2) d\mathbf{x}_1 d\mathbf{x}_2 - \iint \phi_a(\mathbf{x}_1)\phi_b(\mathbf{x}_2) \frac{1}{r_{12}} \phi_b(\mathbf{x}_1)\phi_a(\mathbf{x}_2) d\mathbf{x}_1 d\mathbf{x}_2$$

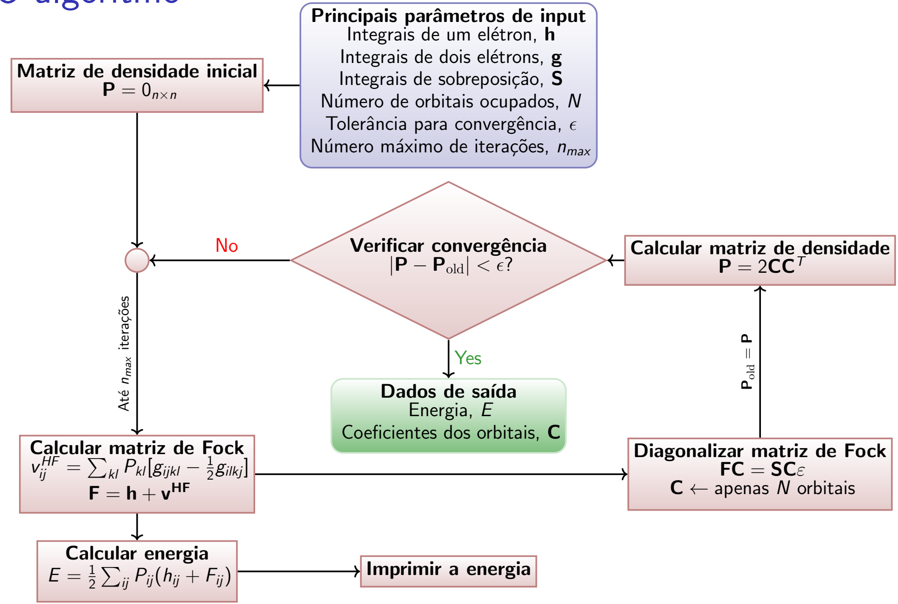
$$= \frac{1}{2} \sum_{ij} P_{ij}(h_{ij} + F_{ij})$$

O algoritmo



Principais parâmetros de input
O Hamiltoniano eletrônico $\hat{H}$:
Posição dos núcleos
Número de elétrons
Orbitais iniciais
$\phi_1, \phi_2, \ldots \phi_{N_e}$
No
Verificar convergência
$\phi_j = \phi'_j$?
Yes
Output
Energy, E
Orbitals, ϕ_j
Obter o operador de Fock
$\hat{f} = \hat{h} + v^{HF}$
Encontrar autofunções do operador de Fock
$\hat{f}\phi'_j = \varepsilon_j \phi'_j$

O algoritmo



Principais parâmetros de input
Integrais de um elétron, $\mathbf{h}$
Integrais de dois elétrons, $\mathbf{g}$
Integrais de sobreposição, $\mathbf{S}$
Número de orbitais ocupados, N
Tolerância para convergência, ϵ
Número máximo de iterações, n_{max}
Matriz de densidade inicial
$\mathbf{P} = 0_{n\times n}$
Até n_{max} iterações
No
Verificar convergência
$|\mathbf{P} - \mathbf{P}_{\text{old}}| < \epsilon$?
Calcular matriz de densidade
$\mathbf{P} = 2\mathbf{C}\mathbf{C}^T$
$\mathbf{P}_{\text{old}} = \mathbf{P}$
Yes
Dados de saída
Energia, E
Coeficientes dos orbitais, $\mathbf{C}$
Calcular matriz de Fock
$v_{ij}^{HF} = \sum_{kl} P_{kl}[g_{ijkl} - \frac{1}{2}g_{ilkj}]$
$\mathbf{F} = \mathbf{h} + \mathbf{v}^{\mathbf{HF}}$
Diagonalizar matriz de Fock
$\mathbf{FC} = \mathbf{SC}\varepsilon$
$\mathbf{C} \leftarrow$ apenas N orbitais
Calcular energia
$E = \frac{1}{2}\sum_{ij} P_{ij}(h_{ij} + F_{ij})$
Imprimir a energia

Numpy Arrays

https://numpy.org/doc/stable/user/absolute_beginners.html

NumPy: the absolute basics for beginners

Welcome to the absolute beginner's guide to NumPy! If you have comments or suggestions, please don't hesitate to reach out!

Welcome to NumPy!

NumPy (**Numerical Python**) is an open source Python library that's used in almost every field of science and engineering. It's the universal standard for working with numerical data in Python, and it's at the core of the scientific Python and PyData ecosystems. NumPy users include everyone from beginning coders to experienced researchers doing state-of-the-art scientific and industrial research and development. The NumPy API is used extensively in Pandas, SciPy, Matplotlib, scikit-learn, scikit-image and most other data science and scientific Python packages.

The NumPy library contains multidimensional array and matrix data structures (you'll find more information about this in later sections). It provides **ndarray**, a homogeneous n-dimensional array object, with methods to efficiently operate on it. NumPy can be used to perform a wide variety of mathematical operations on arrays. It adds powerful data structures to Python that guarantee efficient calculations with arrays and matrices and it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices.

Learn more about NumPy here!

Transposing and reshaping a matrix

This section covers `arr.reshape()`, `arr.transpose()`, `arr.T`

It's common to need to transpose your matrices. NumPy arrays have the property `T` that allows you to transpose a matrix.

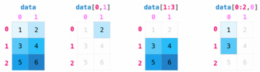
Creating matrices

You can pass Python lists of lists to create a 2-D array (or "matrix") to represent them in NumPy.

```
>>> data = np.array([[1, 2], [3, 4], [5, 6]])
>>> data
array([[1, 2],
       [3, 4],
       [5, 6]])
```

Indexing and slicing operations are useful when you're manipulating matrices:

```
>>> data[0, 1]
2
>>> data[1:3]
array([[3, 4],
       [5, 6]])
>>> data[0:2, 0]
array([1, 3])
```



Dimensões da array

https://numpy.org/doc/stable/reference/generated/numpy.shape.html

numpy.shape

`numpy.shape(a)` [source]

Return the shape of an array.

Parameters: a : *array_like*

Input array.

Returns: shape : *tuple of ints*

The elements of the shape tuple give the lengths of the corresponding array dimensions.

See also

`len`

`len(a)` is equivalent to `np.shape(a)[0]` for N-D arrays with `N>=1`.

`ndarray.shape`

Equivalent array method.

Examples

```
>>> np.shape(np.eye(3))
(3, 3)
>>> np.shape([[1, 3]])
(1, 2)
>>> np.shape([0])
(1,)
>>> np.shape(0)
()
```

Array com zeros

https://numpy.org/doc/stable/reference/generated/numpy.zeros.html

numpy.zeros

`numpy.zeros(shape, dtype=float, order='C', *, like=None)`

Return a new array of given shape and type, filled with zeros.

Parameters: **shape** : *int or tuple of ints*

Shape of the new array, e.g., `(2, 3)` or `2`.

dtype : *data-type, optional*

The desired data-type for the array, e.g., `numpy.int8`. Default is `numpy.float64`.

order : *{'C', 'F'}, optional, default: 'C'*

Whether to store multi-dimensional data in row-major (C-style) or column-major (Fortran-style) order in memory.

like : *array_like, optional*

Reference object to allow the creation of arrays which are not NumPy arrays. If an array-like passed in as `like` supports the `__array_function__` protocol, the result will be defined by it. In this case, it ensures the creation of an array object compatible with that passed in via this argument.

New in version 1.20.0.

Returns: **out** : *ndarray*

Array of zeros with the given shape, dtype, and order.

$\mathbf{P} = 2\mathbf{C}\mathbf{C}^T$

https://numpy.org/doc/stable/reference/generated/numpy.matmul.html

numpy.matmul

`numpy.matmul(x1, x2, /, out=None, *, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj, axes, axis]) = <ufunc 'matmul'>`

Matrix product of two arrays.

Parameters: **x1, x2** : *array_like*

Input arrays, scalars not allowed.

out : *ndarray, optional*

A location into which the result is stored. If provided, it must have a shape that matches the signature *(n,k),(k,m)->(n,m)*. If not provided or None, a freshly-allocated array is returned.

****kwargs**

For other keyword-only arguments, see the ufunc docs.

New in version 1.16: Now handles ufunc kwargs

Returns: **y** : *ndarray*

The matrix product of the inputs. This is a scalar only when both x1, x2 are 1-d vectors.

The `@` operator can be used as a shorthand for `np.matmul` on ndarrays.

```
>>> x1 = np.array([2j, 3j])
>>> x2 = np.array([2j, 3j])
>>> x1 @ x2
(-13+0j)
```

$\mathbf{P} = 2\mathbf{C}\mathbf{C}^T$

https://numpy.org/doc/stable/reference/generated/numpy.ndarray.T.html

numpy.ndarray.T

attribute

`ndarray.T`

The transposed array.

Same as `self.transpose()`.

See also

`transpose`

Examples

```
>>> x = np.array([[1.,2.],[3.,4.]])
>>> x
array([[ 1.,  2.],
       [ 3.,  4.]])
>>> x.T
array([[ 1.,  3.],
       [ 2.,  4.]])
>>> x = np.array([1.,2.,3.,4.])
>>> x
array([ 1.,  2.,  3.,  4.])
>>> x.T
array([ 1.,  2.,  3.,  4.])
```

Previous
numpy.ndarray.dtype

Next
numpy.ndarray.real

$|\mathbf{P} - \mathbf{P}_{\text{old}}| < \epsilon$

https://numpy.org/doc/stable/reference/generated/numpy.allclose.html

numpy.allclose

`numpy.allclose(a, b, rtol=1e-05, atol=1e-08, equal_nan=False)` [source]

Returns True if two arrays are element-wise equal within a tolerance.

The tolerance values are positive, typically very small numbers. The relative difference (*rtol* * abs(*b*)) and the absolute difference *atol* are added together to compare against the absolute difference between *a* and *b*.

NaNs are treated as equal if they are in the same place and if `equal_nan=True`. Infs are treated as equal if they are in the same place and of the same sign in both arrays.

Parameters:

a, b : *array_like*

Input arrays to compare.

rtol : *float*

The relative tolerance parameter (see Notes).

atol : *float*

The absolute tolerance parameter (see Notes).

equal_nan : *bool*

Whether to compare NaN's as equal. If True, NaN's in *a* will be considered equal to NaN's in *b* in the output array.

New in version 1.10.0.

Returns:

allclose : *bool*

Returns True if the two arrays are equal within the given tolerance; False otherwise.

$\mathbf{FC} = \mathbf{SC}\varepsilon$

$$\mathbf{FC}_j = \varepsilon_j \mathbf{SC}_j$$

$$\begin{pmatrix} F_{11} & F_{12} & \cdots & F_{1n} \\ F_{21} & F_{22} & \cdots & F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ F_{n1} & F_{n2} & \cdots & F_{nn} \end{pmatrix} \begin{pmatrix} C_{1j} \\ C_{2j} \\ \vdots \\ C_{nj} \end{pmatrix} = \varepsilon_j \begin{pmatrix} S_{11} & S_{12} & \cdots & S_{1n} \\ S_{21} & S_{22} & \cdots & S_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ S_{n1} & S_{n2} & \cdots & S_{nn} \end{pmatrix} \begin{pmatrix} C_{1j} \\ C_{2j} \\ \vdots \\ C_{nj} \end{pmatrix}$$

$$\mathbf{FC} = \mathbf{SC}\varepsilon$$

$$\begin{pmatrix} F_{11} & F_{12} & \cdots & F_{1n} \\ F_{21} & F_{22} & \cdots & F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ F_{n1} & F_{n2} & \cdots & F_{nn} \end{pmatrix} \begin{pmatrix} C_{11} & C_{12} & \cdots & C_{1n} \\ C_{21} & C_{22} & \cdots & C_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ C_{n1} & C_{n2} & \cdots & C_{nn} \end{pmatrix}$$

$$= \begin{pmatrix} S_{11} & S_{12} & \cdots & S_{1n} \\ S_{21} & S_{22} & \cdots & S_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ S_{n1} & S_{n2} & \cdots & S_{nn} \end{pmatrix} \begin{pmatrix} C_{11} & C_{12} & \cdots & C_{1n} \\ C_{21} & C_{22} & \cdots & C_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ C_{n1} & C_{n2} & \cdots & C_{nn} \end{pmatrix} \begin{pmatrix} \varepsilon_1 & 0 & \cdots & 0 \\ 0 & \varepsilon_2 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \varepsilon_n \end{pmatrix}$$

https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.eigh.html

scipy.linalg.eigh

scipy.linalg.**eigh**(*a, b=None, lower=True, eigvals_only=False, overwrite_a=False, overwrite_b=False, turbo=True, eigvals=None, type=1, check_finite=True, subset_by_index=None, subset_by_value=None, driver=None*) [source]

Solve a standard or generalized eigenvalue problem for a complex Hermitian or real symmetric matrix.

Find eigenvalues array `w` and optionally eigenvectors array `v` of array `a`, where `b` is positive definite such that for every eigenvalue λ (i-th entry of w) and its eigenvector `vi` (i-th column of `v`) satisfies:

```
              a @ vi = λ * b @ vi
vi.conj().T @ a @ vi = λ
vi.conj().T @ b @ vi = 1
```

In the standard problem, `b` is assumed to be the identity matrix.

Parameters: **a** : ***(M, M) array_like***

A complex Hermitian or real symmetric matrix whose eigenvalues and eigenvectors will be computed.

b : ***(M, M) array_like, optional***

A complex Hermitian or real symmetric definite positive matrix in. If omitted, identity matrix is assumed.

Returns: **w** : ***(N,) ndarray***

The N (1<=N<=M) selected eigenvalues, in ascending order, each repeated according to its multiplicity.

v : ***(M, N) ndarray***

(if `eigvals_only == False`)

Examples

```
>>> from scipy.linalg import eigh
>>> A = np.array([[6, 3, 1, 5], [3, 0, 5, 1], [1, 5, 6, 2], [5, 1, 2, 2]])
>>> w, v = eigh(A)
>>> np.allclose(A @ v - v @ np.diag(w), np.zeros((4, 4)))
True
```

Ortogonalização da base

$$\mathbf{U}^T\mathbf{S}\mathbf{U} = \mathbf{s} \iff \mathbf{S}\mathbf{U} = \mathbf{U}\mathbf{s}$$

$$\mathbf{X} = \mathbf{U}\mathbf{s}^{-1/2}$$

Se

$$\mathbf{F}' = \mathbf{X}^T\mathbf{F}\mathbf{X}$$

e

$$\mathbf{C}' = \mathbf{X}^{-1}\mathbf{C} \quad (\mathbf{C} = \mathbf{X}\mathbf{C}')$$

então

$$\mathbf{F}'\mathbf{C}' = \mathbf{C}'\varepsilon$$