



$$\psi_k = \sum_{\mu} c_{\mu k} \chi_{\mu} \quad (\text{LCAO})$$

$$\hat{F} \varphi_i = \varepsilon_i \varphi_i$$

$$S_{\mu\nu} = \int \chi_{\mu}^*(\mathbf{r}) \chi_{\nu}(\mathbf{r}) d\mathbf{r}$$

С. М. Пономаренко

# Квантово-механічні методи обчислення Використання PySCF

$$\left[ -\frac{1}{2} \nabla_i^2 + \sum_{j \neq i} \frac{1}{r_{ij}} + V_{\text{nuc}}(i) \right] \varphi_i = \varepsilon_i \varphi_i$$
$$\Psi = \det \begin{pmatrix} \varphi_1(1) & \varphi_2(1) & \cdots & \varphi_N(1) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_1(N) & \varphi_2(N) & \cdots & \varphi_N(N) \end{pmatrix}$$

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені ІГОРЯ СІКОРСЬКОГО»

С. М. Пономаренко

# **Квантово-механічні методи обчислення Використання PySCF**

*Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського як  
навчальний посібник для здобувачів ступеня магістра за спеціальностями  
Е6 «Прикладна фізика та наноматеріали»*

КИЇВ  
КПІ ім. Ігоря Сікорського  
2025

# Зміст

---

<b>1</b>	<b>Знайомство з PySCF</b>	<b>7</b>
1.1	Що таке PySCF: можливості та переваги . . . . .	7
1.1.1	Основні можливості PySCF . . . . .	7
1.1.2	Переваги використання PySCF . . . . .	7
1.1.3	Порівняння з іншими програмами . . . . .	8
1.2	Встановлення PySCF . . . . .	8
1.2.1	Системні вимоги . . . . .	8
1.2.2	Встановлення в Linux . . . . .	9
1.2.3	Встановлення в Windows . . . . .	9
1.2.4	Встановлення в macOS . . . . .	9
1.2.5	Перевірка встановлення . . . . .	10
1.3	Базова структура програми на PySCF . . . . .	10
1.3.1	Загальна схема розрахунку . . . . .	10
1.3.2	Мінімальний приклад . . . . .	10
1.3.3	Структура для атома Гелію . . . . .	11
1.3.4	Розрахунок з аналізом результатів . . . . .	11
1.4	Система одиниць та конвенції . . . . .	11
1.4.1	Атомні одиниці . . . . .	11
1.4.2	Конвертація одиниць . . . . .	12
1.4.3	Системні константи . . . . .	12
1.4.4	Конвенції для атомів . . . . .	12
1.5	Документація та ресурси . . . . .	13
1.5.1	Офіційна документація . . . . .	13
1.5.2	Основні публікації . . . . .	13
1.6	Резюме . . . . .	14
<b>2</b>	<b>Базові об'єкти та поняття в PySCF</b>	<b>15</b>
2.1	Молекулярний об'єкт (Mole) . . . . .	15
2.1.1	Клас Mole — серце PySCF . . . . .	15
2.1.2	Створення об'єкта Mole . . . . .	15
2.1.3	Основні атрибути об'єкта Mole . . . . .	16
2.1.4	Визначення атома: різні способи . . . . .	17
2.1.5	Робота з іонами . . . . .	18
2.1.6	Налаштування вербальності виводу . . . . .	19

2.2	Базисні набори для атомів . . . . .	20
2.2.1	Типи базисних наборів . . . . .	20
2.2.2	Вибір базисного набору: рекомендації . . . . .	22
2.2.3	Власні базисні набори . . . . .	22
2.2.4	Комбінування базисів . . . . .	23
2.2.5	Інформація про базисний набір . . . . .	23
2.3	Симетрія в атомних розрахунках . . . . .	24
2.3.1	Точкові групи симетрії атомів . . . . .	24
2.3.2	Використання симетрії . . . . .	25
2.3.3	Симетрія орбіталей . . . . .	25
2.3.4	Коли вимикати симетрію . . . . .	26
2.4	Спін та мультиплетність . . . . .	27
2.4.1	Визначення спінового стану . . . . .	27
2.4.2	Розподіл альфа- та бета-електронів . . . . .	28
2.4.3	Вибір правильного спіну . . . . .	28
2.4.4	Енергетичне підтвердження спінових станів . . . . .	28
2.4.5	Практичні зауваження . . . . .	29
2.5	Налаштування параметрів конвергенції . . . . .	30
2.5.1	Основні параметри SCF . . . . .	30
2.5.2	Початкове наближення . . . . .	30
2.5.3	Робота з важкими випадками конвергенції . . . . .	31
2.5.4	Моніторинг конвергенції . . . . .	32
2.5.5	Збереження та завантаження результатів . . . . .	32
2.6	Практичний приклад: комплексний аналіз атома . . . . .	33
2.7	Корисні функції та методи . . . . .	35
2.7.1	Інформація про атом . . . . .	35
2.7.2	Маніпуляції з базисом . . . . .	36
2.7.3	Енергетичні компоненти . . . . .	36
2.7.4	Резюме . . . . .	37
2.7.5	Контрольні запитання . . . . .	37
2.7.6	Завдання для самостійної роботи . . . . .	37

### **3 Метод Хартрі-Фока для атомів 39**

3.1	Теоретичні основи методу Хартрі-Фока . . . . .	39
3.1.1	Рівняння Хартрі-Фока . . . . .	39
3.1.2	Варіанти методу Хартрі-Фока . . . . .	39
3.1.3	Енергія Хартрі-Фока . . . . .	40
3.2	Розрахунок атома Гідрогену . . . . .	40
3.2.1	Особливості одноелектронної системи . . . . .	41
3.2.2	Залежність від базисного набору . . . . .	41
3.2.3	Аналіз орбіталей . . . . .	42
3.2.4	Висновки . . . . .	43
3.3	Розрахунок атома Гелію . . . . .	43
3.3.1	Двоелектронна система . . . . .	43

3.3.2	Порівняння RHF та UHF . . . . .	44
3.3.3	Збуджені стани Гелію . . . . .	44
3.4	Атоми другого періоду (Li–Ne) . . . . .	46
3.4.1	Літій (Li, $Z=3$ ) . . . . .	46
3.4.2	Берилій (Be, $Z=4$ ) . . . . .	47
3.4.3	Бор–Неон: систематичне дослідження . . . . .	48
3.4.4	Енергії іонізації . . . . .	50
3.5	Аналіз енергій та орбіталей . . . . .	52
3.5.1	Енергетична діаграма орбіталей . . . . .	52
3.5.2	Розподіл електронної густини . . . . .	53
3.5.3	Порівняння електронних густин різних атомів . . . . .	58
3.6	Порівняння методів: RHF vs UHF vs ROHF . . . . .	61
3.6.1	Тестовий випадок: атом Вуглецю . . . . .	61
3.6.2	Теоретичний аналіз результатів . . . . .	63
3.6.3	Систематичне порівняння для всіх атомів . . . . .	63
3.7	Складні випадки та збіжність . . . . .	65
3.7.1	Перехідні метали . . . . .	65
3.7.2	Використання дробових заповнень . . . . .	67
3.7.3	Стратегії досягнення конвергенції . . . . .	68
3.8	Практичні завдання . . . . .	70
3.8.1	Завдання 1: Систематичне дослідження . . . . .	70
3.8.2	Завдання 2: Енергії іонізації . . . . .	70
3.8.3	Завдання 3: Спектроскопічні константи . . . . .	71
3.8.4	Завдання 4: Залежність від базису . . . . .	71
3.9	Резюме . . . . .	71
3.9.1	Ключові висновки . . . . .	71
<b>4</b>	<b>Теорія функціоналу густини (DFT)</b>	<b>73</b>
4.1	Основи теорії функціоналу густини . . . . .	73
4.1.1	Теореми Хоенберга–Кона . . . . .	73
4.1.2	Рівняння Кона–Шема . . . . .	73
4.1.3	Обмінно-кореляційна енергія . . . . .	74
4.1.4	Порівняння HF та DFT . . . . .	74
4.2	Функціонали обміну-кореляції . . . . .	74
4.2.1	Класифікація функціоналів: драбина Якова . . . . .	74
4.2.2	LDA та LSDA функціонали . . . . .	75
4.2.3	GGA функціонали . . . . .	76
4.2.4	Meta-GGA функціонали . . . . .	77
4.2.5	Гібридні функціонали . . . . .	78
4.2.6	Порівняння різних рівнів теорії . . . . .	80
4.2.7	Вибір функціоналу: рекомендації . . . . .	82
4.2.8	Практичний приклад: вплив функціоналу на властивості . . . . .	82
4.3	DFT розрахунки атомів . . . . .	84
4.3.1	Базова структура DFT розрахунку . . . . .	84

4.3.2	Систематичний розрахунок атомів другого періоду . . .	84
4.3.3	Розрахунок атомів перехідних металів . . . . .	85
4.3.4	Порівняння спінових станів . . . . .	86
4.3.5	Аналіз d-орбіталей перехідних металів . . . . .	88
4.3.6	Розрахунок важких атомів . . . . .	90
4.3.7	Числові сітки в DFT . . . . .	90
4.3.8	Паралелізація DFT розрахунків . . . . .	92
4.4	Порівняння HF та DFT результатів . . . . .	92
4.4.1	Систематичне порівняння енергій . . . . .	92
4.4.2	Порівняння енергій іонізації . . . . .	95
4.4.3	Електронна спорідненість . . . . .	97
4.4.4	Порівняння орбітальних енергій . . . . .	98
4.4.5	Забруднення спіном: HF vs DFT . . . . .	100
4.4.6	Час обчислень: HF vs DFT . . . . .	101
4.4.7	Загальні висновки HF vs DFT . . . . .	102
4.5	Вибір функціоналу для атомних систем . . . . .	103
4.5.1	Тестовий набір даних . . . . .	103
4.5.2	Рекомендації для різних елементів . . . . .	105
4.6	Практичні завдання . . . . .	105
4.6.1	Завдання 1: Систематичне дослідження . . . . .	105
4.6.2	Завдання 2: Функціональна залежність . . . . .	105
4.6.3	Завдання 3: Конвергенція до базисної межі . . . . .	106
4.6.4	Завдання 4: Перехідні метали . . . . .	106
4.7	Резюме . . . . .	106
4.7.1	Ключові висновки . . . . .	106
4.7.2	Типові помилки . . . . .	107
4.7.3	Корисні посилання . . . . .	107

## **5 Пост-Гартрі-Фоківські методи 108**

5.1	Вступ до електронної кореляції . . . . .	108
5.1.1	Що таке електронна кореляція? . . . . .	108
5.1.2	Типи електронної кореляції . . . . .	108
5.1.3	Кореляційна енергія атомів . . . . .	108
5.1.4	Ієрархія методів . . . . .	109
5.1.5	Важливість кореляції для різних властивостей . . . . .	110
5.1.6	Порівняння методів для He . . . . .	111
5.1.7	Концепція одно- та багатоконфігураційних методів . . . . .	112
5.1.8	Коли потрібні post-HF методи? . . . . .	114
5.2	Теорія збурень Møller-Plesset (MP2) . . . . .	114
5.2.1	Теоретичні основи MP2 . . . . .	114
5.2.2	Формула MP2 кореляційної енергії . . . . .	114
5.2.3	MP2 розрахунок атома Неону . . . . .	114
5.2.4	MP2 для відкритих оболонок (UMP2) . . . . .	115
5.2.5	Систематичне порівняння HF vs MP2 . . . . .	116

5.2.6	Залежність MP2 від базисного набору . . . . .	118
5.2.7	Переваги та недоліки MP2 . . . . .	120
5.2.8	Рекомендації по використанню MP2 . . . . .	120

<b>Література</b>	<b>122</b>
-------------------	------------

# 1

## Знайомство з PySCF

---

### 1.1. Що таке PySCF: можливості та переваги

PySCF (Python-based Simulations of Chemistry Framework) — це сучасний програмний пакет з відкритим вихідним кодом для квантово-хімічних розрахунків, написаний переважно на мові Python з критичними для продуктивності частинами на C. Розроблений групою під керівництвом професора Garnet Chan, PySCF надає потужний та гнучкий інструментарій для проведення *ab initio* розрахунків молекулярних та атомних систем.

**1.1.1. Основні можливості PySCF.** PySCF підтримує широкий спектр квантово-хімічних методів:

- *Метод Хартрі-Фока (HF)*: restricted (RHF), unrestricted (UHF), та restricted open-shell (ROHF) варіанти
- *Теорія функціоналу густини (DFT)*: з великою бібліотекою функціоналів обміну-кореляції через інтеграцію з `libxc`
- *Post-Hartree-Fock методи*: MP2, CCSD, CCSD(T), CASSCF, CASPT2, NEVPT2
- *Configuration Interaction (CI)*: CISD, full CI для малих систем
- *Явна кореляція*: F12 методи
- *Багаторівневі методи*: ONIOM, embedding
- *Періодичні системи*: через модуль PBC (Periodic Boundary Conditions)

### 1.1.2. Переваги використання PySCF.

**Відкритий код та безкоштовність.** PySCF розповсюджується під ліцензією Apache 2.0, що робить його повністю безкоштовним для академічного та комерційного використання. Доступ до вихідного коду дозволяє глибоко зрозуміти реалізацію методів та за потреби модифікувати програму.

**Гнучкість та розширюваність.** Завдяки використанню Python як основної мови, PySCF надзвичайно гнучкий. Користувачі можуть легко:

- Комбінувати різні методи в одному скрипті.

- Створювати власні workflow для складних розрахунків.
- Інтегрувати PySCF з іншими Python бібліотеками (NumPy, SciPy, matplotlib).
- Розробляти власні модулі та методи.

**Сучасна архітектура.** PySCF використовує об'єктно-орієнтований підхід, що робить код зрозумілим та легким для модифікації. Модульна структура дозволяє використовувати тільки потрібні компоненти.

**Активна розробка.** Проект активно розвивається, регулярно додаються нові функції та покращується продуктивність. Спільнота користувачів надає підтримку через GitHub та форуми.

**Продуктивність.** Незважаючи на використання Python, критичні обчислювальні частини написані на C та оптимізовані. PySCF ефективно використовує сучасні бібліотеки лінійної алгебри (BLAS, LAPACK) та може працювати на багатоядерних системах.

**1.1.3. Порівняння з іншими програмами.** У таблиці 1.1 наведено порівняння PySCF з іншими популярними квантово-хімічними пакетами.

Таблиця 1.1. Порівняння квантово-хімічних програм

Програма	Ліцензія	Мова	Гнучкість	Навчання
PySCF	Відкрита	Python/C	Висока	Середнє
Gaussian	Комерційна	Fortran	Низька	Легке
ORCA	Академічна	C++	Середня	Легке
Q-Chem	Комерційна	C/C++	Середня	Середнє
Psi4	Відкрита	C++/Python	Висока	Середнє

## 1.2. Встановлення PySCF

**1.2.1. Системні вимоги.** Для роботи з PySCF необхідно:

- Python 3.7 або новіший
- NumPy версії 1.13.0 або новішої
- SciPy версії 1.0.0 або новішої
- Н5ру (для збереження великих масивів даних)
- 4–8 ГБ оперативної пам'яті (мінімум)
- Сучасний процесор (бажано багатоядерний)

**1.2.2. Встановлення в Linux.** Найпростіший спосіб встановлення PySCF у Linux — використання `pip`:

```
# Оновлення pip
pip install --upgrade pip

# Встановлення PySCF
pip install pyscf
```

Для встановлення з додатковими можливостями:

```
# з підтримкою ХС функціоналів
pip install pyscf[geomopt,dftd3,dmrgscf]
```

Альтернативно, можна встановити з вихідного коду:

```
git clone https://github.com/pyscf/pyscf.git
cd pyscf
pip install -e .
```

**1.2.3. Встановлення в Windows.** Для Windows рекомендується використання Anaconda:

```
# Створення нового середовища
conda create -n pyscf_env python=3.10
conda activate pyscf_env

# Встановлення PySCF
conda install -c pyscf pyscf
```

Або через `pip` у PowerShell/Command Prompt:

```
pip install pyscf
```

**1.2.4. Встановлення в macOS.** Для macOS процес аналогічний до Linux:

```
# Встановлення через pip
pip3 install pyscf

# Або через Homebrew + pip
brew install python3
pip3 install pyscf
```

**1.2.5. Перевірка встановлення.** Після встановлення перевіримо, чи PySCF працює коректно:

```
import pyscf

# Вибір версії
print(pyscf.__version__)

# Простий тест
from pyscf import gto, scf

mol = gto.M(atom='H 0 0 0; H 0 0 0.74', basis='sto-3g')
mf = scf.RHF(mol)
energy = mf.kernel()
print(f'Енергія H2: {energy:.6f} Ha')
```

Якщо програма виводить версію та обчислює енергію молекули H<sub>2</sub>, встановлення виконано успішно.

## 1.3. Базова структура програми на PySCF

**1.3.1. Загальна схема розрахунку.** Типовий розрахунок у PySCF складається з трьох основних етапів:

1. **Створення молекулярного об'єкта** — визначення геометрії системи, базисного набору, заряду та спіну
2. **Вибір та налаштування методу** — вибір квантово-хімічного методу (HF, DFT, CC тощо)
3. **Виконання розрахунку** — запуск обчислень та отримання результатів

**1.3.2. Мінімальний приклад.** Розглянемо найпростіший приклад розрахунку атома Гідрогену:

```
from pyscf import gto, scf

# Етап 1: Створення молекулярного об'єкта
mol = gto.M(
    atom='H 0 0 0',      # Координати атома
    basis='sto-3g',      # Базисний набір
    spin=1               # 2S (1 неспарений електрон)
)

# Етап 2: Вибір методу (UHF для відкритої оболонки)
mf = scf.UHF(mol)

# Етап 3: Виконання розрахунку
energy = mf.kernel()

# Виведення результатів
print(f'Енергія атома H: {energy:.8f} Hartree')
print(f'Енергія атома H: {energy * 27.211386:.6f} eV')
```

**1.3.3. Структура для атома Гелію.** Для атома з двома електронами у замкненій оболонці:

```
from pyscf import gto, scf

# Атом Гелію
mol = gto.M(
    atom='He 0 0 0',
    basis='6-31g',
    spin=0,           # Всі електрони спарені
    charge=0          # Нейтральний атом
)

# RHF для замкненої оболонки
mf = scf.RHF(mol)
energy = mf.kernel()

print(f'Енергія He: {energy:.8f} Ha')
```

**1.3.4. Розрахунок з аналізом результатів.** Більш детальний приклад з виведенням додаткової інформації:

```
from pyscf import gto, scf

# Атом Літію
mol = gto.M(
    atom='Li 0 0 0',
    basis='cc-pvdz',
    spin=1           # Один неспарений електрон
)

# UHF розрахунок
mf = scf.UHF(mol)
mf.verbose = 4      # Детальний вивід
energy = mf.kernel()

# Аналіз результатів
print('\n=== Результати розрахунку ===')
print(f'Енергія: {energy:.8f} Ha')
print(f'Кількість базисних функцій: {mol.nao_nr()}')
print(f'Кількість електронів: {mol.nelectron}')
print(f'Спін: {mol.spin}')

# Заселеності Малікена
from pyscf import lo
pop = mf.mulliken_pop()
print('\nАналіз заселеностей Малікена:')
print(pop)

# Енергії орбіталей
print('\nЕнергії орбіталей (альфа):')
for i, e in enumerate(mf.mo_energy[0][:5]):
    print(f' MO {i+1}: {e:.6f} Ha ({e*27.211386:.4f} eV)')
```

## 1.4. Система одиниць та конвенції

**1.4.1. Атомні одиниці.** PySCF використовує атомну систему одиниць (atomic units, a.u.), де:

$$\begin{aligned}\hbar &= 1 \\ m_e &= 1 \quad (\text{маса електрона}) \\ e &= 1 \quad (\text{елементарний заряд}) \\ 4\pi\epsilon_0 &= 1 \quad (\text{електрична константа})\end{aligned}$$

У цій системі:

- Енергія вимірюється в Hartree (Ha):  $1 \text{ Ha} = 27.211386 \text{ eV} = 627.509 \text{ kcal/mol}$
- Відстань вимірюється в Bohr ( $a_0$ ):  $1 \text{ Bohr} = 0.529177 \text{ \AA}$
- Час вимірюється у а.о.:  $1 \text{ a.u.} = 2.4189 \times 10^{-17} \text{ s}$

#### 1.4.2. Конвертація одиниць. PySCF надає модуль для конвертації:

```
from pyscf import lib

# Конвертація енергії
energy_ha = -2.85516
energy_ev = energy_ha * lib.param.HARTREE2EV
energy_kcal = energy_ha * lib.param.HARTREE2KCAL

print(f'{energy_ha:.6f} Ha = {energy_ev:.4f} eV')
print(f'{energy_ha:.6f} Ha = {energy_kcal:.2f} kcal/mol')

# Конвертація відстані
dist_bohr = 2.0
dist_angstrom = dist_bohr * lib.param.BOHR

print(f'{dist_bohr:.2f} Bohr = {dist_angstrom:.4f} \AA')
```

#### 1.4.3. Системні константи. Доступ до фізичних констант:

```
from pyscf.data import nist

print(f'Швидкість світла: {nist.LIGHT_SPEED} a.u.')
print(f'Константа Планка: {nist.PLANCK} J.s')
print(f'Число Авогадро: {nist.AVOGADRO} mol^-1')
```

#### 1.4.4. Конвенції для атомів. При визначенні атомів у PySCF:

**Координати.** Координати задаються у формі рядка або списку. За замовчуванням використовуються ангстрєми ( $\text{\AA}$ ), але можна вказати одиниці:

```
# Координати в \AA (за замовчуванням)
mol = gto.M(atom='C 0 0 0')

# Явне вказання одиниць
mol = gto.M(atom='C 0 0 0', unit='Angstrom')

# Координати в Bohr
mol = gto.M(atom='C 0 0 0', unit='Bohr')
```

**Спін.** Параметр `spin` визначає  $2S = N_\alpha - N_\beta$ , де  $N_\alpha$  та  $N_\beta$  — кількість альфа та бета електронів:

```
# Атом Н (1 неспарений електрон): S=1/2, 2S=1
mol = gto.M(atom='H 0 0 0', spin=1)

# Атом He (всі спарені): S=0, 2S=0
mol = gto.M(atom='He 0 0 0', spin=0)

# Атом О у триплетному стані: S=1, 2S=2
mol = gto.M(atom='O 0 0 0', spin=2)
```

**Симетрія.** За замовчуванням PySCF використовує повну точкову симетрію системи:

```
# Автоматичне визначення симетрії
mol = gto.M(atom='Ne 0 0 0', symmetry=True)
print(f'Точкова група: {mol.groupname}')

# Вимкнення симетрії
mol = gto.M(atom='Ne 0 0 0', symmetry=False)
```

## 1.5. Документація та ресурси

### 1.5.1. Офіційна документація.

Ресурс	URL
Головний сайт	<a href="https://pyscf.org">https://pyscf.org</a>
GitHub репозиторій	<a href="https://github.com/pyscf/pyscf">https://github.com/pyscf/pyscf</a>
Документація API	<a href="https://pyscf.org/pyscf_api_docs/pyscf.html">https://pyscf.org/pyscf_api_docs/pyscf.html</a>
Приклади коду	<a href="https://github.com/pyscf/pyscf/tree/master/examples">https://github.com/pyscf/pyscf/tree/master/examples</a>

**1.5.2. Основні публікації.** Ключові статті для цитування при використанні PySCF:

1. PySCF: the Python-based simulations of chemistry framework / Q. Sun [et al.] // Wiley Interdisciplinary Reviews: Computational Molecular Science. — 2018. — Vol. 8, no. 1. — e1340. — DOI: [10.1002/wcms.1340](https://doi.org/10.1002/wcms.1340).
2. Recent developments in the PySCF program package / Q. Sun [et al.] // Journal of Chemical Physics. — 2020. — Vol. 153, no. 2. — P. 024109. — DOI: [10.1063/5.0006074](https://doi.org/10.1063/5.0006074).

## 1.6. Резюме

У цьому розділі ми познайомились з PySCF — потужним інструментом для квантово-хімічних розрахунків. Основні моменти:

- PySCF є сучасним, відкритим та гнучким програмним пакетом
- Встановлення здійснюється просто через `pip` або `conda`
- Базова структура програми включає три етапи: створення системи, вибір методу, виконання розрахунку
- PySCF використовує атомну систему одиниць
- Доступна велика кількість документації та навчальних матеріалів

У наступному розділі ми детально розглянемо базові об'єкти PySCF та навчимось налаштовувати параметри розрахунків для атомних систем.

# 2

## Базові об'єкти та поняття в PySCF

### 2.1. Молекулярний об'єкт (Mole)

**2.1.1. Клас Mole — серце PySCF.** Клас `gto.Mole` є центральним об'єктом у бібліотеці **PySCF** (Python-based Simulations of Chemistry Framework). Цей клас визначає і зберігає всю інформацію про атомну або молекулярну систему, яка необхідна для квантово-хімічних розрахунків. Саме з нього починається будь-який проєкт у PySCF, адже всі інші модулі (SCF, DFT, MP2, CCSD тощо) працюють із вже побудованим об'єктом **Mole**.

Об'єкт **Mole** містить такі основні дані:

- **Геометрію системи** — координати атомів у просторі (в ангстремах або борах);
- **Базисний набір** — набір функцій, на яких розкладаються молекулярні орбіталі;
- **Заряд системи та спин** (мультипліситет);
- **Інформацію про симетрію** (за потреби);
- **Одиниці вимірювання** (ангстреми, бори).

Таким чином, **Mole** виконує роль «серця» всієї програми — воно зберігає стан квантової системи і передає цю інформацію до інших підсистем для проведення обчислень.

**2.1.2. Створення об'єкта Mole.** Існує два основні способи створення молекулярного об'єкта: або за допомогою скороченого конструктора, або покроково. Обидва підходи рівноцінні за результатом, однак відрізняються стилем запису.

**Метод 1: Використання конструктора M()** Цей спосіб є найзручнішим для простих систем, коли всі параметри можна вказати безпосередньо при створенні об'єкта:

```
from pyscf import gto

# Простий спосіб
mol = gto.M(
    atom='Li 0 0 0',
```

```
basis='6-31g',  
charge=0,  
spin=1  
)
```

Тут:

- `atom='Li 0 0 0'` — визначає атом літію в координатах (0, 0, 0);
- `basis='6-31g'` — вказує базисний набір для розрахунків;
- `charge=0` — нейтральний атом;
- `spin=1` — означає, що система має  $2S = 1$ , тобто один неспарений електрон.

Функція `gto.M()` автоматично створює об'єкт, налаштовує всі параметри та викликає `.build()`.

**Метод 2: Покрокове налаштування** Цей спосіб зручний для складних систем, де потрібно задати багато параметрів або змінювати їх під час роботи:

```
from pyscf import gto  
  
# Створення порожнього об'єкта  
mol = gto.Mole()  
  
# Налаштування параметрів  
mol.atom = 'C 0 0 0'  
mol.basis = 'cc-pvdz'  
mol.charge = 0  
mol.spin = 2 # Два неспарені електрони  
  
# Завершення побудови (важливо!)  
mol.build()
```

**Важливо:** При використанні другого методу обов'язково викликати команду `mol.build()`, інакше PySCF не згенерує внутрішні структури (матриці, орбіталі, таблиці інтегралів тощо), необхідні для подальших розрахунків.

**2.1.3. Основні атрибути об'єкта Mole.** Після побудови об'єкта можна отримати детальну інформацію про систему. Ось приклад, який демонструє найпоширеніші атрибути:

```
from pyscf import gto  
  
mol = gto.M(atom='O 0 0 0', basis='6-31g', spin=2)  
  
# Інформація про систему  
print(f'Кількість електронів: {mol.nelectron}')
```

```
print(f'Заряд: {mol.charge}')
```

```
print(f'Спін (2S): {mol.spin}')
```

```
print(f'Кількість базисних функцій: {mol.nao_nr()}')
```

```
# Альфа та бета електрони
print(f'Альфа електронів: {mol.nelec[0]}')
print(f'Бета електронів: {mol.nelec[1]}')

# Атомна структура
print(f'Кількість атомів: {mol.natm}')
print(f'Заряди ядер: {mol.atom_charges()}')

# Базисний набір
print(f'Назва базису: {mol.basis}')
```

Ці атрибути особливо важливі для перевірки, чи правильно задані всі вхідні параметри перед запуском складних обчислень.

- `mol.nelectron` — кількість електронів у системі;
- `mol.charge` — сумарний заряд системи;
- `mol.spin` — подвоєне значення спіну ( $2S$ );
- `mol.nao_nr()` — кількість базисних орбіталей (число функцій, які будуть використані в обчисленнях);
- `mol.nelec` — кортеж (`n_alpha`, `n_beta`) з кількістю альфа- та бета-електронів;
- `mol.natm` — кількість атомів у системі;
- `mol.atom_charges()` — масив ядерних зарядів;
- `mol.basis` — опис обраного базисного набору.

Типовий вивід програми для атома кисню:

```
Кількість електронів: 8
Заряд: 0
Спін (2S): 2
Кількість базисних функцій: 18
Альфа електронів: 5
Бета електронів: 3
Кількість атомів: 1
Заряди ядер: [8.]
Назва базису: 6-31g
```

Такий вивід дозволяє переконатися, що система побудована коректно, а параметри відповідають фізичному змісту задачі. Наприклад, у цьому випадку маємо нейтральний атом кисню ( $Z = 8$ ,  $N_e = 8$ ), у якого спін  $S = 1$  (два неспарені електрони).

Після побудови об'єкта **Mole** його можна передавати до будь-яких методів PySCF — від найпростіших SCF до корельованих методів CCSD, CASCI та інших. Тому розуміння структури і властивостей цього класу є ключем до ефективного використання всієї бібліотеки.

**2.1.4. Визначення атома: різні способи.** Визначення атома або атомів — це перший крок при побудові квантово-хімічної моделі. У PySCF координати та типи атомів можна задавати кількома способами. Усі вони приводять до одного й того ж результату, тому вибір форми запису залежить лише від зручності.

**Спосіб 1: Рядок** Цей варіант є найпростішим і найчастіше використовується для одиночних атомів або малих систем. Формат: `<атом> x y z`, де координати задаються в ангстремах за замовчуванням (або в борах, якщо `unit='Bohr'`).

```
# Один атом
mol = gto.M(atom='Ne 0 0 0', basis='def2-svp')

# Можна використовувати атомний номер
mol = gto.M(atom='10 0 0 0', basis='def2-svp') # 10 = Ne
```

Як видно, PySCF дозволяє вказувати елемент як за його хімічним символом, так і за атомним номером. У другому випадку він автоматично розпізнає елемент періодичної системи.

**Спосіб 2: Список або кортеж** Якщо потрібно задати кілька атомів, або якщо координати отримуються програмно (наприклад, з масиву), зручно використовувати структуру даних типу `list` або `tuple`. Кожен атом описується як пара: `[ім'я, (x, y, z)]`.

```
# Використання списку
mol = gto.M(
    atom=[['Na', (0.0, 0.0, 0.0)]],
    basis='aug-cc-pvdz'
)

# Для декількох атомів (наприклад, для порівняння)
mol = gto.M(
    atom=[
        ['H', (0.0, 0.0, 0.0)],
        ['He', (5.0, 0.0, 0.0)] # далеко один від одного
    ],
    basis='sto-3g'
)
```

Така форма є більш універсальною — вона дозволяє легко зчитувати геометрію з файлів, генерувати координати циклом або будувати складні молекули. Наприклад, молекула водню  $H_2$  може бути задана як:

```
atom = [['H', (0, 0, 0)], ['H', (0, 0, 0.74)]]
```

де відстань у 0.74 Å відповідає типовій довжині зв'язку Н–Н.

**2.1.5. Робота з іонами.** Клас `Mole` дозволяє легко моделювати заряджені системи — катіони та аніони. Для цього достатньо змінити параметр `charge`, який визначає сумарний заряд системи:

$$Q = Z_{\text{ядер}} - N_{\text{електронів}}$$

Крім того, слід відповідно скоригувати **spin**, який задає подвоєне значення спіну  $2S$ . Для нейтральних атомів значення спіну зазвичай відповідає їхній електронній конфігурації.

```
from pyscf import gto, scf

# Нейтральний атом Li
li_atom = gto.M(atom='Li 0 0 0', basis='6-31g', charge=0, spin=1)

# Катіон Li+ (втрачено 1 електрон)
li_cation = gto.M(atom='Li 0 0 0', basis='6-31g', charge=1, spin=0)

# Аніон Li- (додано 1 електрон)
li_anion = gto.M(atom='Li 0 0 0', basis='6-31g', charge=-1, spin=1)

print(f'Li: {li_atom.nelectron} електронів')
print(f'Li+: {li_cation.nelectron} електронів')
print(f'Li-: {li_anion.nelectron} електронів')
```

У цьому прикладі можна побачити, як зміна заряду впливає на кількість електронів:

- Нейтральний атом **Li** має 3 електрони;
- Катіон **Li<sup>+</sup>** — 2 електрони (втратив один);
- Аніон **Li<sup>-</sup>** — 4 електрони (отримав додатковий).

Такі прості зміни параметрів дозволяють вивчати іонізаційні енергії, електронну спорідненість, а також порівнювати властивості нейтральних і заряджених систем.

**2.1.6. Налаштування вербальності виводу.** PySCF має вбудований механізм керування докладністю текстового виводу (тобто "балакучістю" програми). Це зручно, коли потрібно або бачити повну діагностику, або, навпаки, приховати проміжні повідомлення при пакетних розрахунках.

Рівень керується параметром **verbose**:

0 = тихо, 4 = стандартно, 9 = детально (debug)

```
# verbose контролює кількість виводу
# 0 = мінімум, 4 = максимум (за замовчуванням), 9 = дебаг

mol = gto.M(
    atom='F 0 0 0',
    basis='cc-pvdz',
    verbose=4 # Детальний вивід
)

# Або налаштувати пізніше
mol.verbose = 0 # Тихий режим
mol.build()
```

У практиці рекомендується:

- Використовувати **verbose=4** для навчальних цілей, щоб бачити хід побудови базису та інтегралів;
- Використовувати **verbose=0** або **1** при серійному запуску розрахунків на кластері, щоб уникнути перевантаження логів;
- Для діагностики чи налагодження коду — **verbose=7-9**, що дає максимально деталізований вивід.

Таким чином, параметр **verbose** дозволяє зручно регулювати ступінь деталізації повідомлень під час обчислень, роблячи роботу з PySCF більш контрольованою і зручною як для навчання, так і для автоматизованих досліджень.

## 2.2. Базисні набори для атомів

**2.2.1. Типи базисних наборів.** У методах квантової хімії атомні орбіталі представлені не аналітичними розв'язками рівняння Шредінгера, а певними апроксимаціями — **базисними функціями**. Сукупність цих функцій для всіх атомів системи називається **базисним набором (basis set)**.

Базис задає, скільки функцій використовується для опису кожної орбіталі (1s, 2p, 3d, ...), та яку форму вони мають (наприклад, гаусівські чи слейтерівські орбіталі). Від вибору базисного набору залежить баланс між швидкістю розрахунку та точністю енергії.

У PySCF базис задається через параметр **basis='...'**, і можна використовувати як стандартні бібліотечні набори, так і власні, створені вручну.

### Мінімальні базиси

Мінімальний базис — це найпростіший опис, коли на кожну атомну орбіталь використовується рівно одна базисна функція. Наприклад, для атома вуглецю (1s, 2s, 2p) це всього п'ять функцій.

```
# STO-3G - найпростіший базис
mol = gto.M(atom='C 0 0 0', basis='sto-3g')
print(f'Кількість базисних функцій: {mol.nao_nr()}')
# Вивід: 5 (1s, 2s, 2px, 2py, 2pz)
```

STO-3G означає, що кожна орбіталь Слейтера апроксимується сумою трьох гаусівських функцій. Такі базиси часто використовують для тестів і навчальних цілей, але вони не забезпечують точних енергій — похибка порівняно з великими базисами може сягати десятків кДж/моль.

### Валентно-розчеплені базиси

Більш точні обчислення вимагають **розщеплення валентних орбіталей**, щоб кожна з них могла гнучко адаптуватися під різні типи хімічних зв'язків. Так з'являються базиси типу 6-31G, 6-311G тощо.

```
# Сімейство Pople
mol_631g = gto.M(atom='N 0 0 0', basis='6-31g')
mol_6311g = gto.M(atom='N 0 0 0', basis='6-311g')

# 3 поляризаційними функціями
mol_631gd = gto.M(atom='N 0 0 0', basis='6-31g*')      # d на важких атомах
mol_631gdp = gto.M(atom='N 0 0 0', basis='6-31g**')     # d на важких, p на H

print(f'6-31g: {mol_631g.nao_nr()} функцій')
print(f'6-31g*: {mol_631gd.nao_nr()} функцій')
print(f'6-31g**: {mol_631gdp.nao_nr()} функцій')
```

Позначення **6-31G** читається так:

- 6 гаусів використовуються для опису внутрішніх орбіталей (core);
- валентна частина описується двома групами функцій: 3 і 1.

Додаткові символи мають такі значення:

- \* — додає поляризаційні *d*-функції на важких атомах;
- \*\* — додає *d*-функції на важких і *p*-функції на водні атоми.

Ці функції дозволяють орбіталям деформуватися під дією хімічного оточення — завдяки цьому значно поліпшується опис зв'язків та дипольних моментів.

## Кореляційно-послідовні базиси

Для кореляційних методів (MP2, CCSD, CI тощо) часто використовують **сімейство Dunning'a** — **cc-pVXZ**, де X = D, T, Q, 5, 6 (double-, triple-, quadruple-, quintuple-zeta). Ці базиси систематично покращують опис хвильової функції й дозволяють проводити екстраполяцію до межі повного базису (CBS-limit).

```
# cc-pVXZ сімейство (X = D, T, Q, 5, 6)
mol_dz = gto.M(atom='O 0 0 0', basis='cc-pvdz')      # Double-zeta
mol_tz = gto.M(atom='O 0 0 0', basis='cc-pvtz')      # Triple-zeta
mol_qz = gto.M(atom='O 0 0 0', basis='cc-pvqz')      # Quadruple-zeta

print(f'cc-pVDZ: {mol_dz.nao_nr()} функцій')
print(f'cc-pVTZ: {mol_tz.nao_nr()} функцій')
print(f'cc-pVQZ: {mol_qz.nao_nr()} функцій')

# Augmented версії (з дифузними функціями)
mol_adz = gto.M(atom='O 0 0 0', basis='aug-cc-pvdz')
print(f'aug-cc-pVDZ: {mol_adz.nao_nr()} функцій')
```

Префікс **aug-** означає додавання **дифузних функцій** — функцій з малим експоненціальним коефіцієнтом, що описують електрони, віддалені від ядра. Такі базиси обов'язкові для аніонів, збуджених станів та слабких взаємодій (ван-дер-ваальсових систем).

## def2 базиси

Базиси типу **def2** — це сучасні розробки групи Карла Ахріхса, оптимізовані для широкого спектру елементів (до лантаноїдів) і дуже добре збалансовані для методів DFT.

```
# Сімейство Ahlrichs
mol_svp = gto.M(atom='Si 0 0 0', basis='def2-svp')
mol_tzvp = gto.M(atom='Si 0 0 0', basis='def2-tzvp')
mol_qzvp = gto.M(atom='Si 0 0 0', basis='def2-qzvp')

print(f'def2-SVP: {mol_svp.nao_nr()} функцій')
print(f'def2-TZVP: {mol_tzvp.nao_nr()} функцій')
print(f'def2-QZVP: {mol_qzvp.nao_nr()} функцій')
```

Позначення: - **SVP** — Split-Valence Polarized (валентно-розщеплений, з поляризацією); - **TZVP** — Triple-Zeta, більш гнучкий опис; - **QZVP** — Quadruple-Zeta, для високоточної роботи.

Базиси **def2** часто використовуються разом з **ефективними псевдопотенціалами (ЕСР)**, які враховують релятивістські ефекти у важких елементах.

**2.2.2. Вибір базисного набору: рекомендації.** Вибір базису — це завжди компроміс між швидкістю й точністю. Загальні рекомендації наведено в Таблиці 2.1.

Таблиця 2.1. Рекомендовані базисні набори для різних задач

Задача	Базис	Коментар
Тестові розрахунки	sto-3g, 3-21g	Дуже швидко, але грубо
Якісні результати	6-31g*, 6-31g**	Оптимальний баланс точність/швидкість
Точні енергії	cc-pVTZ, cc-pVQZ	Для екстраполяції до межі базису
Аніони, збуджені стани	aug-cc-pVDZ	Потрібні дифузні функції
Важкі атоми	def2-TZVP	Добре враховує релятивістські ефекти
Дослідницькі розрахунки	cc-pVDZ	Надійний старт для систем середнього розміру

**2.2.3. Власні базисні набори.** Іноді потрібно створити власний базис (наприклад, для навчання або тестування гібридних моделей). PySCF дозволяє явно задавати коефіцієнти та експоненти базисних функцій через `gto.basis.parse()`.

```

from pyscf import gto

# Визначення базису для H (тільки s-функції)
mol = gto.M(
    atom='H 0 0 0',
    basis={
        'H': gto.basis.parse('''
            H      S
                13.00773    0.019685
                1.962079    0.137977
                0.444529    0.478148
            H      S
                0.1219492   1.000000
        ''')
    })

print(f'Власний базис: {mol.nao_nr()} функцій')

```

У цьому прикладі явно визначено два набори *s*-функцій із різними коефіцієнтами. Так можна створювати спеціалізовані базиси для навчання або розширення стандартних наборів.

**2.2.4. Комбінування базисів.** PySCF підтримує **гібридні базиси** — різні для різних атомів у тій самій системі. Це корисно, коли потрібно зменшити обчислювальні витрати, наприклад, використовуючи спрощений базис для водню, а розширений — для важчих атомів.

```

# Для складних систем можна використовувати різні базиси
# (хоча для атомів це рідко потрібно)
mol = gto.M(
    atom='''
        H 0 0 0
        He 5 0 0
    ''',
    basis={
        'H': 'sto-3g',
        'He': 'cc-pvdz'
    })

```

У результаті PySCF автоматично комбінує обидва базиси при побудові молекулярних орбіталей.

**2.2.5. Інформація про базисний набір.** Іноді потрібно дослідити, які саме функції використовуються у вибраному базисі — їхні типи, кількість та мітки. Для цього можна скористатися внутрішніми структурами об'єкта `Mole`.

```

from pyscf import gto

```

```
mol = gto.M(atom='C 0 0 0', basis='6-31g')

# Детальна інформація
print('Базисні функції по типу:')
for atom_idx in range(mol.natm):
    symbol = mol.atom_symbol(atom_idx)
    print(f'\nАтом {symbol}:')

    # Кількість функцій кожного типу
    basis_atom = mol._basis[symbol]
    for shell in basis_atom:
        l_quantum = shell[0] # Азимутальне квантове число
        n_contractions = len(shell[1:])

        shell_type = ['s', 'p', 'd', 'f', 'g'][l_quantum]
        print(f' {shell_type}-тип: {n_contractions} contracted')

# Діапазони базисних функцій для атома
ao_labels = mol.ao_labels()
print(f'\nМітки базисних функцій:')
for i, label in enumerate(ao_labels):
    print(f'{i}: {label}')
```

---

Так можна отримати повну структуру базису — скільки функцій кожного типу (*s*, *p*, *d*), які індекси мають їхні атомні орбіталі, та як вони нумеруються в PySCF. Ця інформація важлива при аналізі орбіталей, побудові щільності або інтегралів.

## 2.3. Симетрія в атомних розрахунках

Симетрія відіграє ключову роль у квантово-хімічних розрахунках. Вона дозволяє:

- скоротити обчислювальні витрати (менше інтегралів та менші матриці),
- розпізнавати вироджені орбіталі та їх симетрійні властивості,
- аналізувати структуру електронних станів через іррепи (незвідні представлення),
- уникати «зайвих» розв'язків при самоузгодженні.

У PySCF симетрія автоматично враховується при побудові молекулярного об'єкта, якщо активувати опцію `symmetry=True`. Для атомів сферична симетрія апроксимується підгрупами типу **D2h**, що сумісні з декартовими координатами.

**2.3.1. Точкові групи симетрії атомів.** Ізольований атом у строгому сенсі має повну сферичну симетрію *SO(3)*. Однак у PySCF, через використання декартових базисних функцій, використовується одна з підгруп, зазвичай **D2h**. Це дає змогу використовувати блокову структуру матриць і автоматично класифікувати орбіталі за іррепами.

---

```
from pyscf import gto
```

```
# Автоматичне визначення симетрії
mol = gto.M(
    atom='Ne 0 0 0',
    basis='cc-pvdz',
    symmetry=True # Автоматичне визначення точкової групи
)

print(f'Точкова група: {mol.groupname}')
print(f'Топологічна група: {mol.topgroup}')

# Для атома результат, як правило: D2h або подібна підгрупа
```

### Пояснення:

- `groupname` — ідентифікатор підгрупи (наприклад, `D2h`, `C2v`, `Cs`);
- `topgroup` — вища група симетрії, до якої належить дана підгрупа.

**2.3.2. Використання симетрії.** Симетрія допомагає прискорити розрахунок, оскільки гамільтоніан та інші оператори блокуються відповідно до іррепів. Тобто інтеграли між функціями з різних симетрій не обчислюються — що значно зменшує обсяг роботи.

```
from pyscf import gto, scf

# З симетрією (швидше, менше пам'яті)
mol_sym = gto.M(
    atom='Ar 0 0 0',
    basis='cc-pvdz',
    symmetry=True
)
mf_sym = scf.RHF(mol_sym)
e_sym = mf_sym.kernel()

# Без симетрії
mol_nosym = gto.M(
    atom='Ar 0 0 0',
    basis='cc-pvdz',
    symmetry=False
)
mf_nosym = scf.RHF(mol_nosym)
e_nosym = mf_nosym.kernel()

print(f'З симетрією: {e_sym:.8f} Ha')
print(f'Без симетрії: {e_nosym:.8f} Ha')
print(f'Різниця: {abs(e_sym - e_nosym):.2e} Ha')
```

**Коментар:** Різниця в енергіях практично нульова (обидва методи еквівалентні з фізичної точки зору), але симетрійний розрахунок потребує значно менше часу і пам'яті.

**2.3.3. Симетрія орбіталей.** PySCF автоматично визначає симетрійні властивості молекулярних орбіталей після розрахунку. Це особливо корисно для аналізу заповнених і віртуальних рівнів, побудови діаграм енергетичних рівнів та порівняння з експериментом.

```
from pyscf import gto, scf

mol = gto.M(
    atom='N 0 0 0',
    basis='cc-pvdz',
    spin=3, # Основний стан 4S
    symmetry=True
)

mf = scf.UHF(mol)
mf.kernel()

# Орбітальна симетрія
if mol.symmetry:
    orbsym = scf.hf_symm.get_orbsym(mol, mf.mo_coeff[0])
    from pyscf.symm import label_orb_symm

    labels = label_orb_symm(mol, mol.irrep_name, mol.symm_orb,
                           mf.mo_coeff[0])

    print('\nСиметрія заповнених орбіталей (альфа):')
    for i in range(mol.nelec[0]):
        print(f' MO {i+1}: {labels[i]}')
```

#### Коментарі:

- `mol.irrep_name` — список назв іррепів (наприклад, `Ag`, `B1u` тощо);
- `mol.symm_orb` — матриці симетричних орбіталей;
- `label_orb_symm()` — функція, що відображає орбіталі у відповідні іррепи;
- `get_orbsym()` — отримує симетрії молекулярних орбіталей з урахуванням побудованих коефіцієнтів.

Це дає змогу, наприклад, визначити які орбіталі мають однакову симетрію, а які не взаємодіють між собою.

**2.3.4. Коли вимикати симетрію.** Хоча симетрія зазвичай корисна, існують ситуації, коли її слід **вимкнути**:

- При моделюванні **збуджених станів**, які порушують симетрію основного стану.
- При побудові **поверхонь потенційної енергії (PES)**, де геометрія змінюється і симетрія зникає.
- Якщо виникають **проблеми з конвергенцією** — часто через жорсткі симетрійні обмеження.
- Коли необхідно вручну **змінювати орбіталі** або аналізувати змішування між різними симетріями.

У таких випадках достатньо задати:

---

```
mol = gto.M(atom='...', basis='...', symmetry=False)
```

---

**Підсумок:** Використання симетрії — це не лише «оптимізація швидкості», а й **фізично обґрунтований підхід**, що дозволяє зрозуміти структуру

електронних рівнів, виродження та природу хімічних зв'язків.

## 2.4. Спін та мультиплетність

**2.4.1. Визначення спінового стану.** У квантовій хімії поняття спіну має фундаментальне значення. Кожен електрон характеризується спіном  $s = \frac{1}{2}$ , тобто він може перебувати в одному з двох можливих спінових станів — “вгору” ( $\alpha$ ) або “вниз” ( $\beta$ ).

Для багаточастинкової системи повний спін  $S$  визначається як векторна сума спінів усіх електронів. У більшості практичних розрахунків нас цікавить тільки величина  $S$  (а не його напрям).

В PySCF параметр `spin` задає величину  $2S$ , тобто різницю між кількістю  $\alpha$ - та  $\beta$ -електронів:

$$2S = N_{\alpha} - N_{\beta} \quad (2.1)$$

Звідси:

$$S = \frac{N_{\alpha} - N_{\beta}}{2}, \quad M = 2S + 1$$

де  $M$  — мультиплетність (кількість можливих орієнтацій спіну системи в магнітному полі).

- $S = 0 \Rightarrow$  синглет ( $M = 1$ )
- $S = \frac{1}{2} \Rightarrow$  дублет ( $M = 2$ )
- $S = 1 \Rightarrow$  триплет ( $M = 3$ )
- $S = \frac{3}{2} \Rightarrow$  квартет ( $M = 4$ )

```
from pyscf import gto

# Атом H: S=1/2, 2S=1, дублет (M=2)
h = gto.M(atom='H 0 0 0', basis='cc-pvdz', spin=1)

# Атом He: S=0, 2S=0, синглет (M=1)
he = gto.M(atom='He 0 0 0', basis='cc-pvdz', spin=0)

# Атом O: основний стан 3P, S=1, 2S=2, триплет (M=3)
o = gto.M(atom='O 0 0 0', basis='cc-pvdz', spin=2)

# Атом N: основний стан 4S, S=3/2, 2S=3, квартет (M=4)
n = gto.M(atom='N 0 0 0', basis='cc-pvdz', spin=3)

print(f'H: {h.nelec}, 2S={h.spin}, M={h.spin+1}')
print(f'He: {he.nelec}, 2S={he.spin}, M={he.spin+1}')
print(f'O: {o.nelec}, 2S={o.spin}, M={o.spin+1}')
print(f'N: {n.nelec}, 2S={n.spin}, M={n.spin+1}')
```

Цей код створює атомні молекули з різними значеннями параметра `spin`. В об'єкті `mol.nelec` PySCF зберігає кортеж  $(N_{\alpha}, N_{\beta})$ , що дозволяє контролювати спінову структуру системи.

**2.4.2. Розподіл альфа- та бета-електронів.** У багатоспінових системах важливо знати, як електрони розподілені за спіновими підрівнями. В PySCF ця інформація визначається автоматично на основі параметра `spin`, однак її можна перевірити вручну.

```
from pyscf import gto

def print_electron_config(atom_symbol, charge=0, spin=0):
    mol = gto.M(
        atom=f'{atom_symbol} 0 0 0',
        basis='sto-3g',
        charge=charge,
        spin=spin
    )

    n_alpha, n_beta = mol.nelec
    print(f'{atom_symbol} (charge={charge}, 2S={spin}):')
    print(f'  Всього електронів: {mol.nelectron}')
    print(f'  α-електронів: {n_alpha}')
    print(f'  β-електронів: {n_beta}')
    print(f'  Неспарених: {n_alpha - n_beta}\n')

# Приклади
print_electron_config('Li', charge=0, spin=1)    # Li (2s1)
print_electron_config('C', charge=0, spin=2)     # C (3P)
print_electron_config('O', charge=0, spin=2)     # O (3P)
print_electron_config('O', charge=-1, spin=1)    # O- (дублет)
```

Ця функція показує, як саме PySCF обчислює кількість  $\alpha$ - та  $\beta$ -електронів. Наприклад, для нейтрального атома кисню ( $Z = 8$ ) маємо 8 електронів. При  $2S = 2$  отримаємо:

$$N_{\alpha} = 5, \quad N_{\beta} = 3$$

тобто два неспарені електрони — саме це відповідає триплетному стану ( $^3P$ ).

**2.4.3. Вибір правильного спіну.** Необхідно завжди задавати правильний спін для основного стану атома або молекули, інакше SCF-процедура може не збігатися або дати фізично некоректний результат.

Правильне значення  $S$  визначається на основі **\*\*правил Хунда\*\***: 1. Електрони займають орбіталі так, щоб сумарний спін  $S$  був максимальним. 2. Для даного  $S$  максимізується орбітальний момент  $L$ . 3. Для менш ніж напівзаповненої оболонки найнижчий рівень має  $J = |L - S|$ , а для більш ніж напівзаповненої —  $J = L + S$ .

Для атомів другого періоду ці правила дають такі основні стани:

**2.4.4. Енергетичне підтвердження спінових станів.** Нижче наведено код, який обчислює енергії атомів другого періоду для їхніх правильних спінових станів. Для відкрито-оболонкових систем (`spin > 0`) використовується UHF (неспарені електрони), а для синглетів (`spin = 0`) — RHF.

Таблиця 2.2. Основні спінові стани атомів другого періоду

Атом	Конфігурація	Терм	S	2S
Li	[He] 2s <sup>1</sup>	<sup>2</sup> S	1/2	1
Be	[He] 2s <sup>2</sup>	<sup>1</sup> S	0	0
B	[He] 2s <sup>2</sup> 2p <sup>1</sup>	<sup>2</sup> P	1/2	1
C	[He] 2s <sup>2</sup> 2p <sup>2</sup>	<sup>3</sup> P	1	2
N	[He] 2s <sup>2</sup> 2p <sup>3</sup>	<sup>4</sup> S	3/2	3
O	[He] 2s <sup>2</sup> 2p <sup>4</sup>	<sup>3</sup> P	1	2
F	[He] 2s <sup>2</sup> 2p <sup>5</sup>	<sup>2</sup> P	1/2	1
Ne	[He] 2s <sup>2</sup> 2p <sup>6</sup>	<sup>1</sup> S	0	0

```

from pyscf import gto, scf

# Правильні спінові стани для другого періоду
atoms_2nd_period = [
    ('Li', 1), ('Be', 0), ('B', 1), ('C', 2),
    ('N', 3), ('O', 2), ('F', 1), ('Ne', 0)
]

print('Основні стани атомів другого періоду:\n')
for symbol, spin in atoms_2nd_period:
    mol = gto.M(
        atom=f'{symbol} 0 0 0',
        basis='6-31g',
        spin=spin
    )
    mf = scf.UHF(mol) if spin > 0 else scf.RHF(mol)
    mf.verbose = 0
    energy = mf.kernel()

    mult = spin + 1
    print(f'{symbol:2s}: 2S={spin}, M={mult}, E={energy:12.6f} Ha')

```

Результати дозволяють перевірити, що обраний спіновий стан дійсно відповідає мінімуму енергії для даного атома. Якщо спробувати змінити **spin**, наприклад для атома азоту задати **spin=1**, отримаємо енергію вищу на кілька десятків міліГартрі — тобто триплет виявиться збудженим станом відносно правильного к्वартету.

#### 2.4.5. Практичні зауваження.

- У PySCF спін задається на всю систему, тому при моделюванні молекул потрібно враховувати сумарний спін усіх атомів.
- Для молекул із непарним числом електронів (odd number of electrons) **spin** завжди має бути непарним.
- Неправильно заданий спіл часто призводить до **SCF not converged**.
- При використанні DFT (наприклад, **scf.UKS**) параметр **spin** також впливає на спінову поляризацію густини.

## 2.5. Налаштування параметрів конвергенції

Метод самозгодженого поля (SCF — Self-Consistent Field) є основним чисельним кроком у розрахунках Гартрі–Фока (HF) та функціоналу густини (DFT). Його мета — знайти хвильову функцію, яка узгоджена з власним потенціалом електронної густини. Однак SCF-процедура є ітераційною, і її збіжність може бути неочевидною, особливо для відкрито-оболонкових систем, перехідних металів або нестандартних базисів.

Нижче наведено параметри та прийоми, які дозволяють контролювати й покращувати збіжність у PySCF.

**2.5.1. Основні параметри SCF.** Основні параметри визначають точність, стратегію прискорення та межі ітераційного процесу. У PySCF об'єкт `mf = scf.UHF(mol)` (чи `RHF/ROHF`) має низку атрибутів, що дозволяють тонко налаштовувати алгоритм SCF.

```
from pyscf import gto, scf

mol = gto.M(atom='Fe 0 0 0', basis='def2-svp', spin=4)
mf = scf.UHF(mol)

# Критерій конвергенції (допустима різниця енергії між ітераціями)
mf.conv_tol = 1e-10 # За замовчуванням 1e-9

# Максимальна кількість ітерацій SCF
mf.max_cycle = 100 # За замовчуванням 50

# Використання DIIS (Pulay mixing) для прискорення збіжності
mf.diis = True # Активовано за замовчуванням
mf.diis_space = 8 # Розмір DIIS-простору (типово 6-8)

# Level shift – підйом незайнятих орбіталей для стабілізації
mf.level_shift = 0.5 # У Hartree, типово 0.3-1.0 для важких випадків

energy = mf.kernel()
```

### Коментар:

- `conv_tol` — контролює точність енергії. Для атомів перехідних металів бажано знижувати до  $10^{-10}$ .
- `max_cycle` — якщо SCF не збігається за цей ліміт, програма припиняє обчислення.
- **DIIS** (Direct Inversion in the Iterative Subspace) — найпоширеніший метод прискорення SCF.
- `level_shift` — зменшує змішування зайнятих і віртуальних орбіталей на ранніх ітераціях, стабілізуючи розв'язок.

**2.5.2. Початкове наближення.** Початкова матриця густини визначає, з якого стану починається ітераційний процес. Від правильного вибору `init_guess` часто залежить, чи збіжиться SCF взагалі.

```

from pyscf import gto, scf

mol = gto.M(atom='Cr 0 0 0', basis='cc-pvdz', spin=6)
mf = scf.UHF(mol)

# Метод 1: Hcore (за замовчуванням)
mf.init_guess = 'hcore' # Використання Гамільтоніана ядра

# Метод 2: Мінімальний базис (розширене наближення)
mf.init_guess = 'minao'

# Метод 3: Атомні густини (оптимально для атомів та іонів)
mf.init_guess = 'atom'

# Метод 4: 1e guess (використання лише одноелектронних інтегралів)
mf.init_guess = '1e'

# Метод 5: З попереднього файлу (для restart)
# mf.init_guess = 'chkfile'
# mf.chkfile = 'previous_calc.chk'

energy = mf.kernel()

```

### Рекомендації:

- Для ізольованих атомів і відкритих систем — **atom**.
- Для молекул середнього розміру — **minao**.
- Для restart або більших базисів після попереднього розрахунку — **chkfile**.

**2.5.3. Робота з важкими випадками конвергенції.** Деякі системи, особливо перехідні метали або радикали, можуть давати розбіжність SCF навіть за коректних параметрів. У таких випадках застосовують комбінацію стабілізаційних технік.

```

from pyscf import gto, scf
import numpy as np

mol = gto.M(atom='Mn 0 0 0', basis='def2-tzvp', spin=5)
mf = scf.UHF(mol)

# --- Стратегія 1: Level shift
mf.level_shift = 0.3
mf.max_cycle = 100

try:
    energy = mf.kernel()
except:
    print('Не конвергувало з level shift')

# --- Стратегія 2: Newton-Raphson SCF (другий порядок)
if not mf.converged:
    mf = mf.newton() # Перехід до SCF другого порядку
    energy = mf.kernel()

# --- Стратегія 3: Fractional occupation (часткове заселення)
from pyscf import scf
mf = scf.UHF(mol)
mf = scf.addons.frac_occ(mf)

```

```
energy = mf.kernel()

print(f'Фінальна енергія: {energy:.8f} Ha')
print(f'Конвергувало: {mf.converged}')
```

### Коментар:

- **Level shift** допомагає усунути коливання енергії при близьких за енергією орбіталях.
- **Newton-Raphson SCF** — більш точний, але важчий чисельно метод; ефективний для металів.
- **Fractional occupation** дозволяє часткове заселення орбіталей, корисно для вирівнювання густини при дегенерації.

**2.5.4. Моніторинг конвергенції.** PySCF дозволяє відстежувати стан збіжності через callback-функції. Це корисно для аналізу енергетичного профілю і контролю похідних.

```
from pyscf import gto, scf

mol = gto.M(atom='V 0 0 0', basis='cc-pvdz', spin=3)
mf = scf.UHF(mol)

# Callback для моніторингу збіжності
def monitor(envs):
    cycle = envs['cycle']
    e_tot = envs['e_tot']
    norm_gorb = envs['norm_gorb']
    print(f'Cycle {cycle:2d}: E={e_tot:.8f}, |grad|={norm_gorb:.2e}')

mf.callback = monitor
energy = mf.kernel()

# Після завершення:
print(f'\nФінальна енергія: {energy:.8f} Ha')
print(f'Число ітерацій: {mf.iterations}')
print(f'Норма градієнта: {mf.scf_summary["norm_gorb"]:.2e}')
```

### Пояснення:

- **envs** — словник зі станом SCF (цикл, енергія, градієнт тощо).
- **norm\_gorb** — норма матриці похідних (градієнт SCF).
- Моніторинг дозволяє оцінювати стабільність збіжності та спостерігати коливання енергії.

**2.5.5. Збереження та завантаження результатів.** Після завершення SCF-обчислень результати можна зберегти у файл для подальшого використання. Це дозволяє виконувати розрахунки з більшими базисами, використовуючи попередню густину.

```

from pyscf import gto, scf, lib

mol = gto.M(atom='Cu 0 0 0', basis='def2-tzvp', spin=1)
mf = scf.UHF(mol)

# Збереження у checkpoint файл
mf.chkfile = 'cu_atom.chk'
energy = mf.kernel()

# Завантаження результатів з файлу
mol2 = lib.chkfile.load_mol('cu_atom.chk')
mf2 = scf.UHF(mol2)
mf2.__dict__.update(lib.chkfile.load('cu_atom.chk', 'scf'))

print('Завантажена енергія:', mf2.e_tot)

# Використання як початкового наближення для нового базису
mol3 = gto.M(atom='Cu 0 0 0', basis='def2-qzvp', spin=1)
mf3 = scf.UHF(mol3)
mf3.init_guess = 'chkfile'
mf3.chkfile = 'cu_atom.chk'
energy3 = mf3.kernel()

```

### Поради:

- Для повторних розрахунків з більшими базисами або у DFT — `chkfile` з попереднього SCF значно пришвидшує збіжність.
- Можна зберігати SCF-хвильову функцію атома і використовувати її як ініціалізацію для молекули.
- Формат `.chk` — це двійковий HDF5-файл, який містить усі орбіталі, густини, енергії тощо.

## 2.6. Практичний приклад: комплексний аналіз атома

Розглянемо повний приклад аналізу атома з використанням всіх описаних концепцій:

```

from pyscf import gto, scf, lib
import numpy as np

def analyze_atom(symbol, charge=0, spin=None, basis='cc-pvdz'):
    """
    Комплексний аналіз атома

    Parameters:
    -----
    symbol : str
        Символ атома
    charge : int
        Заряд (0 для нейтрального)
    spin : int
        2S (якщо None, визначається автоматично)
    basis : str
        Базисний набір
    """

```

```

print(f'\n{"="*60}')
print(f'Аналіз атома {symbol} (charge={charge}, basis={basis})')
print(f'{"="*60}\n')

# Створення молекулярного об'єкта
mol = gto.M(
    atom=f'{symbol} 0 0 0',
    basis=basis,
    charge=charge,
    spin=spin if spin is not None else 0,
    symmetry=True,
    verbose=4
)

# Інформація про систему
print(f'Кількість електронів: {mol.nelectron}')
print(f'α-електронів: {mol.nelec[0]}')
print(f'β-електронів: {mol.nelec[1]}')
print(f'Спін (2S): {mol.spin}')
print(f'Мультиплетність: {mol.spin + 1}')
print(f'Базисних функцій: {mol.nao_nr()}')
print(f'Точкова група: {mol.groupname}\n')

# Вибір методу SCF
if mol.spin == 0:
    mf = scf.RHF(mol)
    method_name = 'RHF'
else:
    mf = scf.UHF(mol)
    method_name = 'UHF'

# Налаштування параметрів
mf.conv_tol = 1e-10
mf.max_cycle = 100
mf.init_guess = 'atom'

# Розрахунок
print(f'Запуск {method_name} розрахунку...\n')
energy = mf.kernel()

if not mf.converged:
    print('Не конвергувало! Спроба Newton-Raphson...')
    mf = mf.newton()
    energy = mf.kernel()

# Результати
print(f'\n{"="*60}')
print(f'РЕЗУЛЬТАТИ')
print(f'{"="*60}')
print(f'Енергія: {energy:.10f} Ha')
print(f'Енергія: {energy * 27.211386:.6f} eV')
print(f'Конвергувало: {mf.converged}')

# Аналіз орбіталей
print(f'\nОрбітальні енергії ({method_name}):')

if mol.spin == 0:
    # RHF
    print('\nЗаповнені орбіталі:')
    n_occ = mol.nelec[0]
    for i in range(n_occ):
        print(f' MO {i+1:2d}: {mf.mo_energy[i]:10.6f} Ha '
              f'({mf.mo_energy[i]*27.211386:8.4f} eV)')

    print('\nВіртуальні орбіталі (перші 5):')

```

```

    for i in range(n_occ, min(n_occ+5, len(mf.mo_energy))):
        print(f' MO {i+1:2d}: {mf.mo_energy[i]:10.6f} Ha '
              f'({mf.mo_energy[i]*27.211386:8.4f} eV)')
    else:
        # UHF
        print('\nАльфа-орбіталі (заповнені):')
        n_alpha = mol.nelec[0]
        for i in range(n_alpha):
            print(f' α-MO {i+1:2d}: {mf.mo_energy[0][i]:10.6f} Ha '
                  f'({mf.mo_energy[0][i]*27.211386:8.4f} eV)')

        print('\nБета-орбіталі (заповнені):')
        n_beta = mol.nelec[1]
        for i in range(n_beta):
            print(f' β-MO {i+1:2d}: {mf.mo_energy[1][i]:10.6f} Ha '
                  f'({mf.mo_energy[1][i]*27.211386:8.4f} eV)')

        # Заселеності Малікена
        print(f'\n{"="*60}')
        print('АНАЛІЗ ЗАСЕЛЕНОСТЕЙ (Mulliken)')
        print(f'{"="*60}')

        pop = mf.mulliken_pop()

        # Дипольний момент (для нейтральних атомів = 0)
        dip = mf.dip_moment(unit='Debye')
        print(f'\nДипольний момент: ({dip[0]:.4f}, {dip[1]:.4f}, '
              f'{dip[2]:.4f}) Debye')
        print(f'|μ| = {np.linalg.norm(dip):.6f} Debye')

        # Спінова густина (для відкритих систем)
        if mol.spin > 0:
            print(f'\nСпінова густина:')
            s = mf.spin_square()
            print(f' <S²> = {s[0]:.4f}')
            print(f' <S²> (очікуване) = {mol.spin*(mol.spin+2)/4:.4f}')
            print(f' Забруднення спіном: {s[0] - mol.spin*(mol.spin+2)/4:.4f}')

    return mf, energy

# Приклади використання
if __name__ == '__main__':
    # Атом Li (основний стан)
    mf_li, e_li = analyze_atom('Li', spin=1, basis='cc-pvdz')

    # Атом C (триплет)
    mf_c, e_c = analyze_atom('C', spin=2, basis='6-31g*')

    # Катіон O+
    mf_o_plus, e_o_plus = analyze_atom('O', charge=1, spin=3,
                                       basis='aug-cc-pvdz')

```

## 2.7. Корисні функції та методи

### 2.7.1. Інформація про атом.

```

from pyscf import gto
from pyscf.data import elements

mol = gto.M(atom='Zn 0 0 0', basis='def2-svp')

```

```
# Основні атомні характеристики
symbol = mol.atom_symbol(0)
charge = mol.atom_charge(0)
z = gto.charge(symbol)

print(f'Атом: {symbol}')
print(f'Атомний номер: {z}')
print(f'Заряд ядра: {charge}')
print(f'Атомна маса: {elements.MASSES[charge]:.4f} а.о.м.')
print(f'Ковалентний радіус: {elements.COV_RADII[charge]:.2f} Å')

# Кількість електронів (нейтральний атом: ne = Z)
nelectron = z - mol.charge
print(f'Електронів: {nelectron}')
```

---

### 2.7.2. Маніпуляції з базисом.

```
import numpy as np
from pyscf import gto

mol = gto.M(atom='Si 0 0 0', basis='6-31g*')

# Отримання базисних функцій
print('Базисні функції (AO labels):')
for i, label in enumerate(mol.ao_labels(fmt=False)):
    atom_id, atom_symbol, shell_type, *rest = label
    print(f'{i:3d}: Атом {atom_symbol}, тип {shell_type}')

# Кількість функцій кожного типу
from collections import Counter
ao_types = [label[2] for label in mol.ao_labels(fmt=False)]
count = Counter(ao_types)

print(f'\nРозподіл по типах:')
for shell_type, num in sorted(count.items()):
    print(f' {shell_type}: {num} функцій')

# Перекривання базисних функцій
s = mol.intor('int1e_ovlp')
print(f'\nМатриця перекривання: {s.shape}')
print(f'Діагональні елементи (норми): {np.diag(s)[:5]}')

# Загальна кількість AO
print(f'\nЗагальна кількість AO: {mol.nao_nr()}')
```

---

### 2.7.3. Енергетичні компоненти.

```
from pyscf import gto, scf

mol = gto.M(atom='Ne 0 0 0', basis='cc-pvtz', symmetry=True)
mf = scf.RHF(mol)
energy = mf.kernel()

# Детальні компоненти енергії
print('Енергетичні компоненти:')
print(f' Електрон-ядерна: {mf.energy_nuc():.8f} Ha')

# Енергетичні вкладення через методи mf
```

```

e_elec, e_coul_xc = mf.energy_elec()
print(f' Електронна (E_elec): {e_elec:.8f} Ha')
print(f' Кулон+обмін/кореляція: {e_coul_xc:.8f} Ha') # для DFT це Coulomb+XC

print(f' Повна енергія: {energy:.8f} Ha')

# Матриці Фока та густини
h1e = mf.get_hcore() # Одноелектронний гамільтоніан
dm = mf.make_rdm1() # Матриця густини
vhf = mf.get_veff() # Хартрі-Фок / ефективний потенціал

print(f'\nРозміри матриць:')
print(f' h1e: {h1e.shape}')
print(f' dm: {dm.shape}')
print(f' vhf: {vhf.shape}')

# Орбітальні енергії
mo_energy = mf.mo_energy
print(f'\nЕнергії МО (перші 5): {mo_energy[:5]}')

```

#### 2.7.4. Резюме.

- **Клас Mole** — центральний об'єкт, що містить всю інформацію про систему.
- **Базисні набори** — від мінімальних (STO-3G) до великих (cc-pVQZ); вибір залежить від задачі та компромісу точність/вартість.
- **Симетрія** — прискорює розрахунки, але може обмежувати гнучкість при спонтанному руйнуванні симетрії.
- **Спін** — критичний параметр для правильного опису основного стану відкрито-оболонкових систем.

#### 2.7.5. Контрольні запитання.

1. Яка різниця між методами створення об'єкта Mole через `gto.M()` та покроково?
2. Чому для атома Карбону у основному стані використовується `spin=2`?
3. Коли слід використовувати дифузні функції (aug- базиси)?
4. Що означає параметр `conv_tol` і як він впливає на точність?
5. Які методи початкового наближення найкраще підходять для атомів?

#### 2.7.6. Завдання для самостійної роботи.

1. Створіть функцію, яка автоматично визначає правильний спін для атомів першого та другого періодів.
2. Порівняйте енергії атома Неону з різними базисними наборами (STO-3G, 6-31G, cc-pVDZ, cc-pVTZ) та побудуйте графік залежності енергії від розміру базису.
3. Дослідіть вплив симетрії на швидкість розрахунку для атома Аргону: порівняйте час виконання з `symmetry=True` та `symmetry=False`.
4. Обчисліть енергію іонізації атома Літію як різницю енергій Li та Li<sup>+</sup>.
5. Для атома Заліза (Fe) з різними спіновими станами (2S = 2, 4, 6) знайдіть, який стан має найнижчу енергію.

У наступному розділі ми детально розглянемо метод Хартрі-Фока та його застосування до розрахунків атомів.

# 3

## Метод Хартрі-Фока для атомів

### 3.1. Теоретичні основи методу Хартрі-Фока

**3.1.1. Рівняння Хартрі-Фока.** Метод Хартрі-Фока (HF) є наближеним методом розв'язання рівняння Шредінгера для багатоелектронних систем. Основна ідея полягає у представленні багатоелектронної хвильової функції у вигляді єдиного детермінанта Слейтера:

$$\Psi(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_N) = \frac{1}{\sqrt{N!}} \begin{vmatrix} \psi_1(\mathbf{r}_1) & \psi_2(\mathbf{r}_1) & \cdots & \psi_N(\mathbf{r}_1) \\ \psi_1(\mathbf{r}_2) & \psi_2(\mathbf{r}_2) & \cdots & \psi_N(\mathbf{r}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1(\mathbf{r}_N) & \psi_2(\mathbf{r}_N) & \cdots & \psi_N(\mathbf{r}_N) \end{vmatrix} \quad (3.1)$$

де  $\psi_i(\mathbf{r})$  — спін-орбіталі, які є добутком просторової частини та спінової функції.

Канонічні рівняння Хартрі-Фока мають вигляд:

$$\hat{f}\psi_i = \varepsilon_i\psi_i \quad (3.2)$$

де  $\hat{f}$  — оператор Фока:

$$\hat{f} = \hat{h} + \sum_{j=1}^N (\hat{J}_j - \hat{K}_j) \quad (3.3)$$

Тут  $\hat{h}$  — одноелектронний оператор (кінетична енергія + притягання до ядра),  $\hat{J}_j$  — кулонівський оператор,  $\hat{K}_j$  — обмінний оператор.

### 3.1.2. Варіанти методу Хартрі-Фока.

#### Restricted Hartree-Fock (RHF)

RHF використовується для систем з замкненими оболонками, де всі електрони спарені:

$$\psi_i^\alpha(\mathbf{r}) = \psi_i^\beta(\mathbf{r}) = \varphi_i(\mathbf{r}) \quad (3.4)$$

Підходить для: He, Be, Ne, Mg, Ar, Ca, Zn (у синглетних станах).

### Unrestricted Hartree-Fock (UHF)

UHF дозволяє різні просторові орбіталі для альфа та бета спінів:

$$\psi_i^\alpha(\mathbf{r}) \neq \psi_i^\beta(\mathbf{r}) \quad (3.5)$$

Підходить для: всіх відкритих систем (H, Li, B, C, N, O, F та їх іони).

### Restricted Open-shell Hartree-Fock (ROHF)

ROHF — компроміс між RHF та UHF. Спарені електрони описуються однаковими орбіталями, неспарені — різними:

$$\begin{cases} \psi_i^\alpha = \psi_i^\beta & \text{для спарених} \\ \psi_i^\alpha \neq \psi_i^\beta & \text{для неспарених} \end{cases} \quad (3.6)$$

**3.1.3. Енергія Хартрі-Фока.** Повна енергія системи у методі HF:

$$E_{HF} = \sum_{i=1}^N h_{ii} + \frac{1}{2} \sum_{i,j}^N (J_{ij} - K_{ij}) + V_{NN} \quad (3.7)$$

де:

- $h_{ii}$  — одноелектронні інтеграли
- $J_{ij}$  — кулонівські інтеграли
- $K_{ij}$  — обмінні інтеграли
- $V_{NN}$  — енергія міжядерного відштовхування (= 0 для атомів)

### 3.2. Розрахунок атома Гідрогену

Атом Гідрогену є найпростішим квантовим об'єктом, який можна описати в рамках не лише хімії, а й фундаментальної квантової механіки. Його Гамільтоніан має вигляд:

$$\hat{H} = -\frac{1}{2}\nabla^2 - \frac{1}{r},$$

де  $-\frac{1}{2}\nabla^2$  — оператор кінетичної енергії електрона, а  $-\frac{1}{r}$  — потенціал кулонівської взаємодії між електроном і ядром. Ця система має аналітичне рішення, і енергія основного стану дорівнює

$$E_1 = -\frac{1}{2} \text{ Ha.}$$

Для атома Гідрогену це рішення можна отримати навіть чисельно в PySCF, що дозволяє перевірити точність обраного базисного набору та методів квантово-хімічних розрахунків.

**3.2.1. Особливості одноелектронної системи.** Атом Гідрогену є одноелектронною системою, тому метод Гартрі-Фока (HF) не містить жодних апроксимацій, окрім обмежень базисного набору. У звичайних багатоелектронних атомах HF наближає взаємодію електронів середнім потенціалом, але для H відсутнє електрон-електронне відштовхування, тож метод дає *точну* хвильову функцію для обраного базису.

```
from pyscf import gto, scf
import numpy as np

# Створення атома Гідрогену
mol = gto.M(
    atom='H 0 0 0',      # координати ядра
    basis='sto-3g',       # мінімальний базис
    spin=1,               # один неспарений електрон
    verbose=4             # рівень деталізації виводу
)

print(f'Кількість електронів: {mol.nelectron}')
print(f'Базисних функцій: {mol.nao_nr()}')

# Розрахунок методом UHF (необмежений Гартрі-Фок)
mf = scf.UHF(mol)
energy = mf.kernel()

print(f'\nЕнергія H (STO-3G): {energy:.8f} Ha')
print(f'Енергія H (STO-3G): {energy * 27.211386:.6f} eV')

# Теоретичне значення
print(f'Теоретична енергія: -0.5 Ha')
print(f'Похибка базису: {abs(energy + 0.5):.6f} Ha')
```

Отримане значення енергії наближається до  $-0.5$  Ha, але точність залежить від базису. STO-3G — це мінімальний базис, де кожна орбіталь апроксимується трьома гаусовими функціями, тому результат має невелику похибку (близько  $10^{-3}$  Ha).

**3.2.2. Залежність від базисного набору.** Базисний набір визначає якість апроксимації хвильової функції. Чим більший набір, тим ближче розрахована енергія до аналітичного результату. Для атома Гідрогену ця збіжність особливо показова, бо ми можемо порівняти з точним розв'язком.

У коді нижче порівнюються кілька популярних базисів, від найменшого STO-3G до розширених кореляційно-узгоджених наборів cc-pV5Z. Для кожного базису обчислюється енергія та похибка відносно  $-0.5$  Ha.

```
from pyscf import gto, scf
import matplotlib.pyplot as plt

basis_sets = ['sto-3g', '3-21g', '6-31g', '6-311g',
              'cc-pvdz', 'cc-pvtz', 'cc-pvqz', 'cc-pv5z']

energies = []
n_basis = []
```

```

print('Базис          N_bas    Енергія (Ha)    Похибка (mHa)')
print('-' * 60)

for basis in basis_sets:
    try:
        mol = gto.M(atom='H 0 0 0', basis=basis, spin=1, verbose=0)
        mf = scf.UHF(mol)
        e = mf.kernel()

        n = mol.nao_nr()
        error = (e + 0.5) * 1000 # похибка в міліГартрі

        energies.append(e)
        n_basis.append(n)

        print(f'{basis:15s} {n:3d}    {e:12.8f}    {error:8.4f}')
    except:
        print(f'{basis:15s} --- недоступний')

# Побудова графіка збіжності
plt.figure(figsize=(10, 6))
plt.plot(n_basis, energies, 'o-', linewidth=2, markersize=8)
plt.axhline(y=-0.5, color='r', linestyle='--', label='Точне значення')
plt.xlabel('Кількість базисних функцій', fontsize=12)
plt.ylabel('Енергія (Ha)', fontsize=12)
plt.title('Збіжність енергії атома H від базисного набору', fontsize=14)
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.savefig('h_atom_basis_convergence.pdf')
plt.show()

```

З графіка видно, що зі збільшенням кількості базисних функцій енергія швидко наближається до точного значення  $-0.5$  Ha. Набори типу **cc-pVQZ** або **cc-pV5Z** практично дають збіжність до повного базисного ліміту (CBS limit).

**3.2.3. Аналіз орбіталей.** Хоча в атома Гідрогену існує лише одна заповнена орбіталь (1s), PySCF дозволяє вивести енергетичні рівні для всіх функцій базису, а також дослідити матрицю густини. Цей підхід зручний для демонстрації структури HF-розрахунку.

```

from pyscf import gto, scf
import numpy as np

mol = gto.M(atom='H 0 0 0', basis='cc-pvtz', spin=1)
mf = scf.UHF(mol)
energy = mf.kernel()

# Орбітальні енергії
print('\nОрбітальні енергії (альфа-спін):')
print('-' * 50)
for i, (e, label) in enumerate(zip(mf.mo_energy[0], mol.ao_labels())):
    occ = '(occ)' if i < mol.nelec[0] else '(virt)'
    print(f'{i+1:2d}. {label:20s}: {e:10.6f} Ha {occ}')

print(f'\nЕнергія 1s орбіталі: {mf.mo_energy[0][0]:.8f} Ha')

```

```
print(f'Теоретична енергія: -0.50000000 Ha')

# Матриця густини
dm = mf.make_rdm1()
print(f'\nМатриця густини (альфа): {dm[0].shape}')
print(f'Слід dm: {np.trace(dm[0]):.1f} (має дорівнювати 1)')
```

Матриця густини (**dm**) відображає заповнення орбіталей. Її слід дорівнює кількості електронів (тут — 1). Такі розрахунки є основою для подальшого аналізу електронної густини, побудови орбіталей і візуалізації електронної хмари.

### 3.2.4. Висновки.

- Для атома Гідрогену метод Гартрі–Фока є **точним**, бо немає взаємодії між електронами.
- Основна похибка виникає через обмеження базисного набору.
- Зі збільшенням кількості базисних функцій енергія швидко збігається до точного значення  $-0.5$  Ha.
- PySCF дозволяє досліджувати вплив базису, аналізувати орбіталі, матриці густини та підготовлює ґрунт для подальшого розгляду багатоелектронних систем.

## 3.3. Розрахунок атома Гелію

**3.3.1. Двоелектронна система.** Атом гелію є фундаментальним прикладом для демонстрації методів **Хартрі–Фока (HF)**. Це перша система, де проявляється **електрон–електронна кореляція**, відсутня в атомі водню.

Два електрони гелію мають протилежні спіни й заповнюють одну й ту саму орбіталь  $1s$ , тому це *замкнена оболонка* зі спіном  $S = 0$  (синглет,  $M = 1$ ).

```
from pyscf import gto, scf

# Атом He (основний стан  $1S$ )
mol = gto.M(
    atom='He 0 0 0',
    basis='cc-pvtz',
    spin=0, # Замкнена оболонка
    verbose=4
)

print(f'Електронна конфігурація:  $1s^2$ ')
print(f'Кількість електронів: {mol.nelectron}')
```

```
# RHF для замкненої оболонки
mf = scf.RHF(mol)
energy = mf.kernel()

print(f'\nЕнергія He (HF): {energy:.8f} Ha')
print(f'Енергія He (HF): {energy * 27.211386:.6f} eV')
```

```
# Експериментальна енергія He: -2.90372 Ha
exp_energy = -2.90372
correlation_energy = exp_energy - energy

print(f'\nЕкспериментальна енергія: {exp_energy:.6f} Ha')
print(f'Кореляційна енергія: {correlation_energy:.6f} Ha')
print(f'Відносна похибка HF: {abs(correlation_energy/exp_energy)*100:.2f}%')
```

**Коментар.** Метод RHF нехтує кореляцією між електронами, тобто вважає, що хвильова функція є добутком одноелектронних орбіталей. Тому енергія HF завжди *вище* (менш від'ємна), ніж точна. Для гелію похибка складає близько  $\sim 1.7\%$ . Ця різниця — це **кореляційна енергія**:

$$E_{\text{corr}} = E_{\text{exact}} - E_{\text{HF}}$$

**3.3.2. Порівняння RHF та UHF.** Оскільки гелій має замкнену оболонку ( $N_\alpha = N_\beta$ ), обидва методи — **RHF** (restricted HF) і **UHF** (unrestricted HF) — дають ідентичні результати. UHF дозволяє  $\alpha$  та  $\beta$  орбіталям відрізнятися, але тут ця свобода не використовується.

```
from pyscf import gto, scf

mol = gto.M(atom='He 0 0 0', basis='6-31g', spin=0)

# RHF розрахунок
mf_rhf = scf.RHF(mol)
mf_rhf.verbose = 0
e_rhf = mf_rhf.kernel()

# UHF розрахунок
mf_uhf = scf.UHF(mol)
mf_uhf.verbose = 0
e_uhf = mf_uhf.kernel()

print(f'RHF енергія: {e_rhf:.10f} Ha')
print(f'UHF енергія: {e_uhf:.10f} Ha')
print(f'Різниця: {abs(e_rhf - e_uhf):.2e} Ha')

# Перевірка симетрії спіну
s2_uhf = mf_uhf.spin_square()
print(f'\n<S^2> (UHF): {s2_uhf[0]:.6f}')
print(f'<S^2> (точно): 0.000000')
```

**Коментар.** UHF може порушувати спінову симетрію (*spin contamination*), особливо для відкрито-оболонкових систем. Тут же  $S^2 = 0$ , тобто спінова симетрія зберігається, і RHF  $\boxtimes$  UHF.

**3.3.3. Збуджені стани Гелію.** У збуджених станах один електрон переходить на більш високий рівень (наприклад,  $2s$ ), а система може існувати в двох спінових конфігураціях:

- **Синглет:**  $S = 0$  — антисиметрична за спіном, симетрична за просторовою координатою.

- **Триплет:**  $S = 1$  — симетрична за спіном, антисиметрична за просторовою частиною.

У PySCF ці стани можна моделювати за допомогою відповідного значення параметра `spin` та методу `ROHF` для частково заповнених оболонок.

```
from pyscf import gto, scf

def he_excited_state(config='1s2s', spin=0, basis='cc-pvdz'):
    """
    Розрахунок збуджених станів He

    config: '1s2s' (синглет або триплет)
    spin: 0 для синглету, 2 для триплету
    """
    mol = gto.M(atom='He 0 0 0', basis=basis, spin=spin)

    if spin == 0:
        # Синглет: 1s² → 1s2s ¹S
        mf = scf.RHF(mol)
    else:
        # Триплет: 1s² → 1s2s ³S
        mf = scf.ROHF(mol)

    # Для збуджених станів можна використати MOM (maximum overlap method)
    mf = scf.addons.mom_occ(mf, mf.make_rdm1(), mf.get_occ())
    mf.verbose = 0
    energy = mf.kernel()

    return energy

# Основний стан
mol_gs = gto.M(atom='He 0 0 0', basis='cc-pvdz', spin=0)
mf_gs = scf.RHF(mol_gs)
mf_gs.verbose = 0
e_gs = mf_gs.kernel()

print('Стани атома Гелію (cc-pVDZ):')
print('-' * 50)
print(f'1s² (¹S, основний): {e_gs:.6f} Ha')
print(f'    = {e_gs * 27.211386:.2f} eV')

# Примітка: для збуджених станів
# краще використовувати методи типу TD-DFT або CASSCF
```

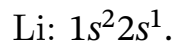
**Коментар.** Метод MOM (Maximum Overlap Method) дозволяє стабілізувати SCF для збуджених станів, зберігаючи певну орбітальну конфігурацію. Проте результати залишаються наближеними. Для точного опису необхідно застосовувати багатоконфігураційні підходи (наприклад, CASSCF, CI, TD-DFT).

#### Підсумок:

- Гелій — базовий тест для валідації HF-методу.
- Різниця між RHF і експериментом демонструє значення кореляційної енергії.
- Збуджені стани потребують спеціальних методів (MOM, CASSCF, TD-DFT).

### 3.4. Атоми другого періоду (Li–Ne)

**3.4.1. Літій (Li,  $Z=3$ ).** Атом Літію — перший багатоелектронний атом, у якому проявляється **неспарений електрон** та **спінова поляризація**. Електронна конфігурація:



Два перші електрони утворюють замкнену оболонку ( $1s$ ), а третій — одинокий у підоболонці  $2s$ , що зумовлює мультиплетність  $2S$  (спін  $S = \frac{1}{2}$ ).

Через наявність неспареного електрона метод **UHF** (Unrestricted Hartree–Fock) є природним вибором. UHF дозволяє альфа- та бета-орбіталям відрізнитися, тобто хвильова функція не змушена мати однакову просторову форму для різних спінів.

```
from pyscf import gto, scf

# Li: [He] 2s1, основний стан 2S
mol = gto.M(
    atom='Li 0 0 0',
    basis='cc-pvdz',
    spin=1, # один неспарений електрон (S=1/2)
    symmetry=True
)

print('Літій (Li):')
print(f' Конфігурація: [He] 2s1')
print(f' Основний стан: 2S')
print(f' Електронів: {mol.nelectron}')

# UHF розрахунок
mf = scf.UHF(mol)
energy = mf.kernel()

print(f'\nЕнергія Li: {energy:.8f} Ha')
print(f'\nЕнергія Li: {energy * 27.211386:.4f} eV')

# Аналіз заселеностей
print('\nЗаселеності Малікена:')
pop = mf.mulliken_pop()

# Орбітальний аналіз
print('\nАльфа-орбіталі (заповнені):')
for i in range(mol.nelec[0]):
    print(f' α-MO {i+1}: {mf.mo_energy[0][i]:10.6f} Ha')

print('\nБета-орбіталі (заповнені):')
for i in range(mol.nelec[1]):
    print(f' β-MO {i+1}: {mf.mo_energy[1][i]:10.6f} Ha')

# Спінова густина
s2 = mf.spin_square()
print(f'\n<S2> = {s2[0]:.6f} (очікується 0.75)')
```

#### Коментарі.

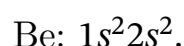
- Значення  $\langle S^2 \rangle \approx 0.75$  свідчить про правильний спіновий стан подвійності (мультиплетність  $2S + 1 = 2$ ).

- Наявність різниці між енергіями альфа- та бета-орбіталей демонструє спінову асиметрію.
- Метод UHF може частково порушувати симетрію (так звана *spin contamination*), але для Li це незначно.

Енергія, отримана методом HF, для Li становить близько  $-7.432$  Ha у базисі *cc-pVDZ*, тоді як експериментальна енергія основного стану (повна) — близько  $-7.478$  Ha. Таким чином, кореляційна похибка становить близько  $0.046$  Ha (приблизно  $1.25$  eV).

Цей приклад є першим, де проявляється *кореляція електронів*, яку HF не враховує. В подальших розділах буде показано, як методи MP2, CI та CC враховують ці ефекти.

**3.4.2. Берилій (Be,  $Z=4$ ).** Атом Берилію має електронну конфігурацію



Тут усі орбіталі парні, тому спінова поляризація відсутня, і зручно використовувати **RHF** (Restricted Hartree–Fock). Обидва електрони у підоболонці  $2s$  мають протилежні спіни, тож система має мультиплетність  $1S$  (синглетний стан).

```
from pyscf import gto, scf

# Be: [He] 2s2, основний стан 1S
mol = gto.M(
    atom='Be 0 0 0',
    basis='cc-pvtz',
    spin=0,          # замкнена оболонка
    symmetry=True
)

print('Берилій (Be):')
print(f' Конфігурація: [He] 2s2')
print(f' Основний стан: 1S')

# RHF розрахунок
mf = scf.RHF(mol)
energy = mf.kernel()

print(f'\nЕнергія Be: {energy:.8f} Ha')

# Порівняння з експериментом
exp_be = -14.6674 # Ha (експериментальна повна енергія)
print(f'Експериментальна: {exp_be:.4f} Ha')
print(f'Кореляційна енергія: {exp_be - energy:.4f} Ha')
```

### Обговорення.

- Для Be HF-енергія виходить приблизно  $-14.573$  Ha (у базисі *cc-pVTZ*), тоді як експериментальна —  $-14.667$  Ha.
- Різниця близько  $0.094$  Ha ( $\approx 2.6$  eV) — це **кореляційна енергія**, тобто внесок взаємодії електронів, не врахований у HF.

- У Берилія ефекти електронної кореляції вже досить значні, адже електрони в орбіталі  $2s$  взаємодіють один з одним.
- Цей випадок є гарною ілюстрацією межі застосування методу Гартрі-Фока.

#### Висновки для атомів Li і Be.

- Li демонструє появу неспареного електрона і спінової поляризації (UHF необхідний).
- Be — перший приклад, де **електронна кореляція** дає помітну похибку енергії.
- PySCF дозволяє наочно дослідити вплив базису, спіну, симетрії та методів на точність енергії.

**3.4.3. Бор–Неон: систематичне дослідження.** Після розгляду окремих атомів Літію та Берилію доцільно перейти до систематичного аналізу всіх атомів другого періоду (від Бору до Неону). У цих атомах поступово заповнюється  $2p$ -підрівень, і змінюється як спінова мультиплетність, так і форма електронної густини. Метод Гартрі-Фока дозволяє побачити, як змінюється енергія системи при збільшенні числа електронів, а також як проявляється спінова поляризація у відкритих оболонках.

#### Електронні конфігурації.

Li:	$[\text{He}] 2s^1$	$^2S$
Be:	$[\text{He}] 2s^2$	$^1S$
B:	$[\text{He}] 2s^2 2p^1$	$^2P$
C:	$[\text{He}] 2s^2 2p^2$	$^3P$
N:	$[\text{He}] 2s^2 2p^3$	$^4S$
O:	$[\text{He}] 2s^2 2p^4$	$^3P$
F:	$[\text{He}] 2s^2 2p^5$	$^2P$
Ne:	$[\text{He}] 2s^2 2p^6$	$^1S$

Як видно, кількість неспарених електронів збільшується від Бору до Нітрогену, а потім зменшується до Неону, що зумовлює зміну спінового стану та мультиплетності.

**Мета дослідження.** Провести розрахунок енергій Гартрі-Фока для атомів другого періоду в однаковому базисі **cc-pVDZ**, оцінити правильність спінового стану (через  $\langle S^2 \rangle$ ) та проаналізувати систематичні тенденції.

---

```
from pyscf import gto, scf
import numpy as np
```

```

# Дані про атоми другого періоду
atoms_data = [
    ('Li', 1, '2S', '[He] 2s1'),
    ('Be', 0, '1S', '[He] 2s2'),
    ('B', 1, '2P', '[He] 2s2 2p1'),
    ('C', 2, '3P', '[He] 2s2 2p2'),
    ('N', 3, '4S', '[He] 2s2 2p3'),
    ('O', 2, '3P', '[He] 2s2 2p4'),
    ('F', 1, '2P', '[He] 2s2 2p5'),
    ('Ne', 0, '1S', '[He] 2s2 2p6')
]

basis = 'cc-pvdz'
print(f'Розрахунок атомів 2-го періоду (базис: {basis})')
print('=' * 70)
print(f'{"Атом":4s} {"Спін":4s} {"Терм":4s} {"E(HF), Ha":15s} '
      f'{"E(HF), eV":12s}')
print('-' * 70)

energies = {}

for symbol, spin, term, config in atoms_data:
    mol = gto.M(
        atom=f'{symbol} 0 0 0',
        basis=basis,
        spin=spin,
        symmetry=True,
        verbose=0
    )

    # Вибір методу SCF
    if spin == 0:
        mf = scf.RHF(mol)
    else:
        mf = scf.UHF(mol)

    mf.verbose = 0
    mf.conv_tol = 1e-10
    energy = mf.kernel()
    energies[symbol] = energy

    e_ev = energy * 27.211386
    print(f'{symbol:4s} {spin:4d} {term:4s} {energy:15.8f} {e_ev:12.2f}')

    # Додаткова інформація для відкритих оболонок
    if spin > 0:
        s2 = mf.spin_square()
        expected_s2 = spin * (spin + 2) / 4
        print(f'<S2> = {s2[0]:.4f} (очікується {expected_s2:.4f})')

print('=' * 70)

# Збереження результатів
np.savez('second_period_hf.npz', **energies)

```

### Коментарі до коду.

- Для кожного атома автоматично обирається тип SCF: RHF (для замкнених оболонок,  $S = 0$ ) або UHF (для відкритих).
- Параметр `spin` у PySCF означає різницю між кількістю альфа- та бета-електронів:  $\text{spin} = N_\alpha - N_\beta = 2S$ .

- Розрахунок  $\langle S^2 \rangle$  дозволяє перевірити правильність спінового стану. Для ідеального випадку значення має збігатися з теоретичним  $S(S + 1)$ .
- У файлі `second_period_hf.npz` зберігаються всі енергії для подальшого аналізу або побудови графіків.

### Очікувані тенденції.

1. Повна енергія атома зменшується (стає більш негативною) із зростанням атомного номера  $Z$ .
2. Енергія спостерігає стрибки на межі заповнення оболонок: при переходах  $\text{Be} \rightarrow \text{B}$ ,  $\text{N} \rightarrow \text{O}$ ,  $\text{F} \rightarrow \text{Ne}$ .
3. Спінова мультиплетність відображає заповнення  $2p$ -орбіталей згідно з **правилом Гунда** — максимальний спін у середині підоболонки (для  $N$ ).
4. Значення  $\langle S^2 \rangle$  для UHF повинні бути близькі до теоретичних, але можуть мати невеликі відхилення через *spin contamination*.

**Фізичне узагальнення.** Цей розрахунок демонструє фундаментальну властивість методу Гартрі-Фока: він добре описує загальну структуру енергетичних рівнів і тенденції в періодичній таблиці, але не враховує електронну кореляцію, через що абсолютні значення енергії мають систематичну похибку.

**Подальші кроки.** Наступним логічним етапом є додавання пост-Гартрі-Фок методів (MP2, CI, CCSD), щоб показати, як кореляція електронів уточнює енергії і дозволяє досягати експериментальної точності.

**3.4.4. Енергії іонізації.** Іонізаційна енергія є однією з фундаментальних характеристик атома, що описує енерговитрати на видалення одного електрона. У квантово-хімічних розрахунках її визначають як різницю повних енергій катіона  $E(A^+)$  і нейтрального атома  $E(A)$ :

$$I = E(A^+) - E(A)$$

де енергії  $E$  обчислюються в межах методу Гартрі-Фока або його узагальнень. Для ізольованих атомів другого періоду така процедура дозволяє отримати якісно правильну послідовність енергій іонізації, навіть попри те, що метод HF систематично недооцінює абсолютні значення через відсутність кореляційних ефектів.

```
from pyscf import gto, scf

def ionization_energy(symbol, charge_neutral=0, spin_neutral=None,
                       spin_cation=None, basis='cc-pvtz'):
    """
    Розрахунок енергії іонізації атома
```

```

IE = E(A+) - E(A)
"""
# Нейтральний атом
mol_neutral = gto.M(
    atom=f'{symbol} 0 0 0',
    basis=basis,
    charge=charge_neutral,
    spin=spin_neutral,
    verbose=0
)

if spin_neutral == 0:
    mf_neutral = scf.RHF(mol_neutral)
else:
    mf_neutral = scf.UHF(mol_neutral)

mf_neutral.verbose = 0
e_neutral = mf_neutral.kernel()

# Катіон
mol_cation = gto.M(
    atom=f'{symbol} 0 0 0',
    basis=basis,
    charge=charge_neutral + 1,
    spin=spin_cation,
    verbose=0
)

if spin_cation == 0:
    mf_cation = scf.RHF(mol_cation)
else:
    mf_cation = scf.UHF(mol_cation)

mf_cation.verbose = 0
e_cation = mf_cation.kernel()

# Енергія іонізації
ie_ha = e_cation - e_neutral
ie_ev = ie_ha * 27.211386

return ie_ev, e_neutral, e_cation

# Розрахунок IE для атомів 2-го періоду
atoms_ie = [
    ('Li', 1, 0), # Li → Li+
    ('Be', 0, 1), # Be → Be+
    ('B', 1, 0), # B → B+
    ('C', 2, 1), # C → C+
    ('N', 3, 2), # N → N+
    ('O', 2, 3), # O → O+
    ('F', 1, 2), # F → F+
    ('Ne', 0, 1), # Ne → Ne+
]

# Експериментальні значення (eV)
exp_ie = {
    'Li': 5.39, 'Be': 9.32, 'B': 8.30, 'C': 11.26,
    'N': 14.53, 'O': 13.62, 'F': 17.42, 'Ne': 21.56
}

print('Енергії іонізації атомів 2-го періоду')
print('=' * 65)
print(f'{"Атом":4s} {"IE(HF), eV":12s} {"IE(exp), eV":12s} {"Похибка, eV":12s}')
print('-' * 65)

```

```
for symbol, spin_n, spin_c in atoms_ie:
    ie_calc, e_n, e_c = ionization_energy(symbol, 0, spin_n, spin_c)
    ie_experimental = exp_ie[symbol]
    error = ie_calc - ie_experimental

    print(f'{symbol:4s} {ie_calc:12.2f} {ie_experimental:12.2f} '
          f'{error:12.2f}')

print('=' * 65)
```

Результатом буде таблиця іонізаційних енергій у електронвольтах. Порівнюючи їх з експериментальними, можна оцінити якість базисного набору та межі методу Гартрі-Фока. Для більш точних результатів використовують після-НФ методи, наприклад MP2 або CCSD, які враховують електронну кореляцію.

## 3.5. Аналіз енергій та орбіталей

**3.5.1. Енергетична діаграма орбіталей.** Енергетичні діаграми орбіталей є наочним способом представлення енергетичного спектру молекулярних орбіталей (МО), що виникають у результаті розв'язання рівнянь Гартрі-Фока.

```
from pyscf import gto, scf
import matplotlib.pyplot as plt
import numpy as np

def plot_orbital_diagram(symbol, spin, basis='cc-pvdz'):
    """Побудова діаграми орбітальних енергій"""

    mol = gto.M(
        atom=f'{symbol} 0 0 0',
        basis=basis,
        spin=spin,
        verbose=0
    )

    if spin == 0:
        mf = scf.RHF(mol)
    else:
        mf = scf.UHF(mol)

    mf.verbose = 0
    mf.kernel()

    fig, ax = plt.subplots(figsize=(8, 10))

    if spin == 0:
        # RHF: одна колонка орбіталей
        energies = mf.mo_energy * 27.211386 # eV
        n_occ = mol.nelec[0]

        for i, e in enumerate(energies[:n_occ+5]):
            color = 'blue' if i < n_occ else 'red'
            label = mol.ao_labels()[i] if i < len(mol.ao_labels()) else ''
            ax.hlines(e, 0.2, 0.8, colors=color, linewidth=2)
            ax.text(0.85, e, f'{label}', fontsize=10, va='center')
```

```

# Стрілки для електронів
if i < n_occ:
    ax.arrow(0.35, e+0.3, 0, -0.25, head_width=0.05,
             head_length=0.1, fc='black', ec='black')
    ax.arrow(0.65, e-0.3, 0, 0.25, head_width=0.05,
             head_length=0.1, fc='black', ec='black')
else:
    # UHF: дві колонки (альфа та бета)
    e_alpha = mf.mo_energy[0] * 27.211386
    e_beta = mf.mo_energy[1] * 27.211386
    n_alpha, n_beta = mol.nelec

    # Альфа орбіталі
    for i, e in enumerate(e_alpha[:n_alpha+3]):
        color = 'blue' if i < n_alpha else 'red'
        ax.hlines(e, 0.1, 0.4, colors=color, linewidth=2)
        if i < n_alpha:
            ax.arrow(0.25, e+0.3, 0, -0.25, head_width=0.04,
                     head_length=0.1, fc='black', ec='black')

    # Бета орбіталі
    for i, e in enumerate(e_beta[:n_beta+3]):
        color = 'blue' if i < n_beta else 'red'
        ax.hlines(e, 0.6, 0.9, colors=color, linewidth=2)
        if i < n_beta:
            ax.arrow(0.75, e-0.3, 0, 0.25, head_width=0.04,
                     head_length=0.1, fc='black', ec='black')

    ax.text(0.25, min(e_alpha[:5])-2, 'α', fontsize=14, ha='center')
    ax.text(0.75, min(e_beta[:5])-2, 'β', fontsize=14, ha='center')

ax.set_xlim(0, 1)
ax.set_ylabel('Енергія (eV)', fontsize=12)
ax.set_title(f'Діаграма орбіталей {symbol}', fontsize=14)
ax.set_xticks([])
ax.grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.savefig(f'{symbol}_orbital_diagram.pdf')
plt.show()

# Приклади
plot_orbital_diagram('C', spin=2) # Вуглець
plot_orbital_diagram('Ne', spin=0) # Неон

```

**3.5.2. Розподіл електронної густини.** Розподіл електронної густини  $\rho(\mathbf{r})$  є однією з ключових величин у квантовій хімії. Саме ця функція визначає, де саме в просторі «перебувають» електрони атома або молекули, і, отже, пояснює більшість хімічних і спектроскопічних властивостей системи.

Згідно з однодетермінантним наближенням Гартрі–Фока, електронна густина визначається як

$$\rho(\mathbf{r}) = \sum_i^{\text{occ}} |\psi_i(\mathbf{r})|^2,$$

де  $\psi_i(\mathbf{r})$  — орбіталі, а сума береться по всіх зайнятих орбіталях.

У програмному пакеті **PySCF** густина може бути обчислена як на сітці, так і в окремих точках простору, що дозволяє візуалізувати просторовий розподіл електронів.

Обчислення густини вздовж осі  $z$ 

У наведеному прикладі реалізовано функцію `plot_density_profile`, що обчислює електронну густину вздовж осі  $z$  для атома. Це дозволяє побудувати одноосний «профіль густини» та простежити, як електронна густина спадає при віддаленні від ядра.

- Молекулярний об'єкт створюється через `gto.M`, де вказується атом, базис та спінова мультиплічність.
- Виконується розрахунок за методом Гартрі-Фока (RHF або UHF залежно від спіну).
- Для кожної точки  $z$  обчислюється матриця базисних функцій  $\varphi_i(\mathbf{r})$ , і далі густина:

$$\rho(\mathbf{r}) = \sum_{ij} \varphi_i(\mathbf{r}) D_{ij} \varphi_j(\mathbf{r}),$$

де  $D_{ij}$  — елементи матриці щільності.

Результат зображається графічно: густина  $\rho(0,0,z)$  як функція відстані від ядра. Для нейтральних атомів крива має різкий максимум біля ядра, який швидко спадає експоненційно.

```
from pyscf import gto, scf
from pyscf.tools import cubegen
import numpy as np
import matplotlib.pyplot as plt

def plot_density_profile(symbol, spin, basis='cc-pvdz'):
    """Радіальний розподіл густини"""

    mol = gto.M(
        atom=f'{symbol} 0 0 0',
        basis=basis,
        spin=spin,
        verbose=0
    )

    if spin == 0:
        mf = scf.RHF(mol)
    else:
        mf = scf.UHF(mol)

    mf.verbose = 0
    mf.kernel()

    # Розрахунок густини вздовж осі z
    r_points = np.linspace(0, 5, 200) # Bohr
    density = []

    dm = mf.make_rdm1()

    for r in r_points:
        coords = np.array([[0, 0, r]])
        rho = mol.eval_gto('GTOval_sph', coords)

        if spin == 0:
            dens = np.einsum('pi,ij,pj->p', rho, dm, rho)[0]
        else:
            dens_a = np.einsum('pi,ij,pj->p', rho, dm[0], rho)[0]
```

```

dens_b = np.einsum('pi,ij,pj->p', rho, dm[1], rho)[0]
dens = dens_a + dens_b

density.append(dens)

# Побудова графіка
plt.figure(figsize=(10, 6))
plt.plot(r_points, density, linewidth=2)
plt.xlabel('Відстань від ядра (Bohr)', fontsize=12)
plt.ylabel('Електронна густина (e/Bohr³)', fontsize=12)
plt.title(f'Радіальний розподіл густини {symbol}', fontsize=14)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig(f'{symbol}_density.pdf')
plt.show()

return r_points, density

# Приклад
r, rho = plot_density_profile('Ne', spin=0)

```

**Інтерпретація результатів.** На побудованому графіку  $\rho(r)$  спостерігаються чіткі області локалізації електронів: внутрішні електрони формують центральний пік, тоді як зовнішні оболонки проявляються у вигляді плавних плечей. Для важчих атомів густина стає більш «зосередженою» біля ядра через сильніше притягання кулонівським потенціалом.

**Практична порада.** При виборі базисного набору (cc-pvdz, 6-31G\*\*, тощо) слід мати на увазі, що форма профілю густини істотно залежить від якості базису: мінімальні базиси дають лише грубу картину, тоді як розширені базиси (cc-pVTZ) відтворюють експоненційний спад більш точно.

### Сферично усереднена густина

Для атомів, що мають сферичну симетрію, зручно розглядати усереднену по всіх напрямках густину:

$$\rho(r) = \frac{1}{4\pi} \int \rho(\mathbf{r}) d\Omega,$$

де  $r = |\mathbf{r}|$  та  $d\Omega$  — елемент тілесного кута.

Ця функція показує, як змінюється густина лише від радіуса  $r$ , незалежно від напрямку, і є основою для побудови радіальної функції розподілу

$$P(r) = 4\pi r^2 \rho(r),$$

що описує, яка частка електронів перебуває у сферичному шарі товщини  $dr$  на відстані  $r$  від ядра.

```

from pyscf import gto, scf
import numpy as np

```

```

import matplotlib.pyplot as plt

def spherical_averaged_density(symbol, spin, basis='cc-pvdz'):
    """Сферично усереднена електронна густина"""

    mol = gto.M(
        atom=f'{symbol} 0 0 0',
        basis=basis,
        spin=spin,
        verbose=0
    )

    if spin == 0:
        mf = scf.RHF(mol)
    else:
        mf = scf.UHF(mol)

    mf.verbose = 0
    mf.kernel()

    dm = mf.make_rdm1()

    # Радіальні точки
    r_points = np.linspace(0.01, 8, 300)

    # Куты для інтегрування (метод Монте-Карло)
    n_angles = 100
    theta = np.random.uniform(0, np.pi, n_angles)
    phi = np.random.uniform(0, 2*np.pi, n_angles)

    density_sph = []

    for r in r_points:
        rho_avg = 0

        for t, p in zip(theta, phi):
            x = r * np.sin(t) * np.cos(p)
            y = r * np.sin(t) * np.sin(p)
            z = r * np.cos(t)

            coords = np.array([x, y, z])
            ao_value = mol.eval_gto('GTOval_sph', coords)

            if spin == 0:
                rho_point = np.einsum('pi,ij,pj->p',
                                       ao_value, dm, ao_value)[0]
            else:
                rho_a = np.einsum('pi,ij,pj->p',
                                   ao_value, dm[0], ao_value)[0]
                rho_b = np.einsum('pi,ij,pj->p',
                                   ao_value, dm[1], ao_value)[0]
                rho_point = rho_a + rho_b

            rho_avg += rho_point

        rho_avg /= n_angles
        density_sph.append(rho_avg)

    # Радіальна функція розподілу  $4\pi r^2 \rho(r)$ 
    radial_dist = 4 * np.pi * r_points**2 * np.array(density_sph)

    # Графіки
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

    # Густина  $\rho(r)$ 

```

```

ax1.plot(r_points, density_sph, linewidth=2)
ax1.set_xlabel('r (Bohr)', fontsize=12)
ax1.set_ylabel('ρ(r) (e/Bohr³)', fontsize=12)
ax1.set_title(f'Електронна густина {symbol}', fontsize=14)
ax1.grid(True, alpha=0.3)
ax1.set_yscale('log')

# Радіальна функція розподілу
ax2.plot(r_points, radial_dist, linewidth=2, color='red')
ax2.set_xlabel('r (Bohr)', fontsize=12)
ax2.set_ylabel('4πr²ρ(r)', fontsize=12)
ax2.set_title(f'Радіальна функція розподілу {symbol}',
              fontsize=14)
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig(f'{symbol}_spherical_density.pdf')
plt.show()

# Перевірка нормування
dr = r_points[1] - r_points[0]
total_electrons = np.sum(radial_dist) * dr
print(f'\nІнтеграл 4πr²ρ(r)dr = {total_electrons:.2f}')
print(f'Очікується: {mol.nelectron} електронів')

return r_points, density_sph, radial_dist

# Приклади для різних атомів
r, rho, rdf = spherical_averaged_density('He', spin=0)
r, rho, rdf = spherical_averaged_density('C', spin=2)
r, rho, rdf = spherical_averaged_density('Ne', spin=0)

```

**Методика обчислення.** Для кожного радіуса  $r$  вибираються випадкові напрями (кутові координати  $\theta, \varphi$ ) за методом Монте-Карло. У цих напрямках обчислюється густина  $\rho(x, y, z)$ , після чого береться середнє значення:

$$\rho(r) \approx \frac{1}{N} \sum_{k=1}^N \rho(r, \theta_k, \varphi_k).$$

Цей підхід забезпечує добру точність навіть при невеликій кількості напрямів  $N \sim 100$ .

**Радіальна функція розподілу.** На другому графіку зображується  $4\pi r^2 \rho(r)$  — це функція, інтеграл від якої по  $r$  дає повну кількість електронів:

$$\int_0^\infty 4\pi r^2 \rho(r) dr = N_e.$$

Таким чином, можна перевірити нормування хвильової функції та коректність чисельного інтегрування.

**Приклад інтерпретації.** Для атома гелію максимум  $4\pi r^2 \rho(r)$  спостерігається приблизно при  $r \approx 0.3$  Bohr, що відповідає найбільш ймовірній відстані електрона від ядра у  $1s$ -стані. Для вуглецю або неону виникають додаткові піки, пов'язані з електронами на  $2s$  та  $2p$  оболонках.

**Перевірка нормування.** Наприкінці виконання функції виводиться значення

$$\int 4\pi r^2 \rho(r) dr,$$

яке має збігатися з кількістю електронів у системі, що є корисним тестом правильності побудованої густини.

**3.5.3. Порівняння електронних густин різних атомів.** Ми розглянули, як отримати електронну густину  $\rho(\mathbf{r})$  та сферично усереднену густину  $\rho(r)$  для окремого атома. Однак для повного розуміння періодичних закономірностей важливо порівняти, як змінюється форма та протяжність електронної хмари при переході від легких до важчих елементів.

### Фізична мотивація

Порівняння електронних густин дозволяє:

- побачити, як із зростанням атомного номера  $Z$  ядро сильніше притягує електрони;
- проаналізувати зміну розмірів атома та ефективного радіуса;
- виявити закономірності заповнення електронних оболонок;
- візуально оцінити зв'язок між структурою густини та положенням елемента в періодичній системі.

У програмі нижче реалізовано функцію порівняння радіальних профілів густини для кількох атомів одночасно. Вона виконує незалежні розрахунки SCF для кожного атома, будує  $\rho(r)$  та радіальну функцію  $4\pi r^2 \rho(r)$ , і виводить результати на одному графіку для порівняння.

### Коментар до реалізації

Ключові кроки програми:

1. Створення об'єкта **Mole** для кожного атома з базисом **cc-pvdz** або **cc-pvtz**.
2. Проведення SCF-розрахунку типу RHF або UHF залежно від спіну.
3. Побудова масиву радіальних точок уздовж осі  $z$ :

$$r \in [0, 6] \text{ Bohr}$$

4. Обчислення електронної густини як:

$$\rho(r) = \sum_{i,j} \varphi_i(r) D_{ij} \varphi_j(r),$$

де  $\varphi_i$  — атомні орбіталі,  $D_{ij}$  — елементи матриці щільності.

5. Для кожного атома будується:

- крива  $\rho(r)$  — логарифмічний масштаб показує різке спадання густини;

- радіальний розподіл  $4\pi r^2 \rho(r)$  — для аналізу оболонкової структури.

```

from pyscf import gto, scf
import numpy as np
import matplotlib.pyplot as plt

def compare_densities(atoms_list, basis='cc-pvdz'):
    """Порівняння електронної густини різних атомів"""

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

    colors = plt.cm.tab10(np.linspace(0, 1, len(atoms_list)))

    for (symbol, spin), color in zip(atoms_list, colors):
        mol = gto.M(
            atom=f'{symbol} 0 0 0',
            basis=basis,
            spin=spin,
            verbose=0
        )

        if spin == 0:
            mf = scf.RHF(mol)
        else:
            mf = scf.UHF(mol)

        mf.verbose = 0
        mf.kernel()

        dm = mf.make_rdm1()

        # Густина вздовж осі z
        r_points = np.linspace(0.01, 6, 200)
        density = []

        for r in r_points:
            coords = np.array([[0, 0, r]])
            ao = mol.eval_gto('GTOval_sph', coords)

            if spin == 0:
                rho = np.einsum('pi,ij,pj->p', ao, dm, ao)[0]
            else:
                rho = (np.einsum('pi,ij,pj->p', ao, dm[0], ao)[0] +
                       np.einsum('pi,ij,pj->p', ao, dm[1], ao)[0])

            density.append(rho)

        # Графіки
        ax1.plot(r_points, density, linewidth=2,
                 label=f'{symbol} (Z={mol.atom_charge(0)})',
                 color=color)

        # Радіальна функція
        radial = 4 * np.pi * r_points**2 * np.array(density)
        ax2.plot(r_points, radial, linewidth=2,
                 label=f'{symbol}', color=color)

    ax1.set_xlabel('r (Bohr)', fontsize=12)
    ax1.set_ylabel('ρ(r) (e/Bohr³)', fontsize=12)
    ax1.set_title('Електронна густина', fontsize=14)
    ax1.set_yscale('log')
    ax1.legend()
    ax1.grid(True, alpha=0.3)

```

```

ax2.set_xlabel('r (Bohr)', fontsize=12)
ax2.set_ylabel('4πr²ρ(r)', fontsize=12)
ax2.set_title('Радіальний розподіл', fontsize=14)
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('atoms_density_comparison.pdf')
plt.show()

# Порівняння атомів благородних газів
noble_gases = [
    ('He', 0), ('Ne', 0), ('Ar', 0)
]
compare_densities(noble_gases, basis='cc-pvtz')

# Порівняння атомів 2-го періоду
second_period = [
    ('Li', 1), ('C', 2), ('N', 3), ('O', 2), ('F', 1)
]
compare_densities(second_period, basis='6-31g*')

```

## Аналіз результатів

На графіку для **благородних газів** (He, Ne, Ar) чітко видно:

- з ростом  $Z$  максимум густини зміщується ближче до ядра — електрони сильніше притягуються кулонівським полем;
- протяжність хвоста  $\rho(r)$  зменшується, атоми стають компактнішими;
- радіальні криві  $4\pi r^2 \rho(r)$  демонструють послідовне збільшення кількості максимумів, що відповідає заповненню нових оболонок.

Для **атомів другого періоду** (Li – F) спостерігається:

- різке зростання центральної густини у міру збільшення  $Z$ ;
- поява другорядного максимуму в  $4\pi r^2 \rho(r)$ , що відповідає електронам  $2s$ -оболонки;
- звуження зовнішньої частини електронної хмари.

## Зв'язок із електронними оболонками

Максимуми радіальної функції розподілу  $4\pi r^2 \rho(r)$  відповідають областям найбільшої ймовірності перебування електронів певних орбіталей:

$$\begin{aligned}
 1s &\rightarrow \text{перший максимум,} & 2s, 2p &\rightarrow \text{другий максимум,} \\
 & & 3s, 3p, 3d &\rightarrow \text{третій тощо.}
 \end{aligned}$$

Таким чином, форма  $4\pi r^2 \rho(r)$  дає прямий зв'язок між результатами квантово-хімічних розрахунків і традиційною атомною структурою, знайомою студентам з курсу атомної фізики. Візуальний аналіз таких графіків дозволяє простежити закономірності побудови періодичної системи Менделєєва з точки зору електронної густини.

## Практичні завдання

1. Побудуйте на одному графіку  $\rho(r)$  для Li, Na, K. Як змінюється радіус атома?
2. Знайдіть положення максимумів  $4\pi r^2 \rho(r)$  для атомів He, Ne, Ar. До яких оболонок вони належать?
3. Порівняйте радіальні функції для атомів C і O. Як змінюється густина зовнішньої оболонки?

## Методичні рекомендації

1. Для легких атомів (H, He, Li) можна порівняти результати з аналітичними розв'язками рівняння Шредінгера.
2. Змінюючи базис (наприклад, ST0-3G, 6-31G, cc-pVTZ), можна дослідити, як збільшується точність відтворення радіального профілю.
3. Для відкритих оболонок (атомів з неспареними електронами) потрібно враховувати спінову поляризацію — використовується UHF.

Таким чином, наведені приклади демонструють практичний зв'язок між теоретичними поняттями електронної густини і їх чисельною реалізацією в пакеті PySCF.

### 3.6. Порівняння методів: RHF vs UHF vs ROHF

У квантово-хімічних розрахунках вибір типу наближення Гартрі-Фока залежить від спінового стану системи. Методи **RHF**, **UHF** та **ROHF** відрізняються тим, як вони поводяться із спіновими функціями електронів — тобто, чи допускають різні орбіталі для  $\alpha$ - і  $\beta$ -електронів.

- **RHF (Restricted Hartree-Fock)** — усі електрони спарені, кожна просторова орбіталь заповнена двома електронами з протилежними спінами. Підходить лише для синглетних закритих оболонок.
- **UHF (Unrestricted Hartree-Fock)** — орбіталі для  $\alpha$ - і  $\beta$ -електронів дозволено різні. Цей метод придатний для відкритих оболонок (наприклад, атомів з неспареними електронами), однак може страждати від *спінового забруднення*.
- **ROHF (Restricted Open-Shell Hartree-Fock)** — компроміс: спільні орбіталі для парних електронів і різні — лише для неспарених. Це забезпечує правильний спіновий симетрійний стан без забруднення.

**3.6.1. Тестовий випадок: атом Вуглецю.** Вуглець має конфігурацію  $1s^2 2s^2 2p^2$  і основний стан  $^3P$  (триплет). Отже, система має два неспарені електрони ( $S = 1$ ), а отже  $\langle S^2 \rangle = S(S + 1) = 2.0$ .

Далі наведено приклад прямого порівняння результатів методів UHF і ROHF для атома C у триплетному стані з використанням базису cc-pVTZ.

```

from pyscf import gto, scf

# Атом C: [He] 2s2 2p2, основний стан 3P
basis = 'cc-pvtz'

print('Порівняння методів для атома Вуглецю (3P)')
print('='*70)

# Молекула
mol = gto.M(
    atom='C 0 0 0',
    basis=basis,
    spin=2, # Триплет
    symmetry=True,
    verbose=0
)

# UHF
print('\n1. Unrestricted Hartree-Fock (UHF)')
print('='*70)
mf_uhf = scf.UHF(mol)
mf_uhf.verbose = 4
e_uhf = mf_uhf.kernel()

s2_uhf = mf_uhf.spin_square()
print(f'\nЕнергія (UHF): {e_uhf:.8f} Ha')
print(f'<S2> (UHF): {s2_uhf[0]:.6f} (очікується 2.0)')
print(f'Забруднення спіном: {s2_uhf[0] - 2.0:.6f}')

# ROHF
print('\n2. Restricted Open-shell Hartree-Fock (ROHF)')
print('='*70)
mf_rohf = scf.ROHF(mol)
mf_rohf.verbose = 4
e_rohf = mf_rohf.kernel()

s2_rohf = mf_rohf.spin_square()
print(f'\nЕнергія (ROHF): {e_rohf:.8f} Ha')
print(f'<S2> (ROHF): {s2_rohf[0]:.6f} (очікується 2.0)')

# Порівняння
print('\n' + '='*70)
print('ПОРІВНЯННЯ')
print('='*70)
print(f'ΔE (UHF-ROHF): {(e_uhf - e_rohf)*1000:.4f} mHa')
print(f'ΔE (UHF-ROHF): {(e_uhf - e_rohf)*627.509:.4f} kcal/mol')

print(f'\nЗабруднення спіном:')
print(f'  UHF: {s2_uhf[0] - 2.0:.6f}')
print(f'  ROHF: {s2_rohf[0] - 2.0:.6f}')

# Орбітальні енергії
print(f'\nОрбітальні енергії (Ha):')
print(f'{"Орбіталь":15s} {"UHF (α)":12s} {"UHF (β)":12s} {"ROHF":12s}')
print('='*70)

n_show = 5
for i in range(n_show):
    label = mol.ao_labels()[i] if i < len(mol.ao_labels()) else f'MO{i+1}'
    e_uhf_a = mf_uhf.mo_energy[0][i]
    e_uhf_b = mf_uhf.mo_energy[1][i]
    e_rohf_i = mf_rohf.mo_energy[i]

```

```
print(f'{label:15s} {e_uhf_a:12.6f} {e_uhf_b:12.6f} {e_rohf_i:12.6f}')
```

### 3.6.2. Теоретичний аналіз результатів.

#### Спінове забруднення

Для триплетного стану очікується:

$$\langle S^2 \rangle = S(S + 1) = 1(1 + 1) = 2.0.$$

Якщо результат UHF дає  $\langle S^2 \rangle > 2.0$ , це свідчить про *спінове забруднення* — хвильова функція містить домішку інших спінових станів (наприклад,  $^5S$ ). Це відбувається через те, що UHF не примушує  $\alpha$ - і  $\beta$ -орбіталі бути ортогональними, тому спінова симетрія порушується.

**РОНФ**, на відміну від цього, зберігає точне спінове квантове число, тому  $\langle S^2 \rangle$  збігається з теоретичним значенням.

#### Енергетичне порівняння

UHF зазвичай дає трохи нижчу енергію, ніж РОНФ, оскільки має більше варіаційної свободи. Різниця  $\Delta E = E_{\text{UHF}} - E_{\text{РОНФ}}$  зазвичай незначна (менше кількох мілігаусів), однак у системах з сильним спіновим змішуванням може бути суттєвою.

#### Орбітальні енергії

UHF формує два набори орбіталей — для  $\alpha$ - і  $\beta$ -електронів. Тому орбітальні енергії можуть відрізнятися:

$$\epsilon_i^{(\alpha)} \neq \epsilon_i^{(\beta)}.$$

У РОНФ усі парні орбіталі спільні, тож орбітальні енергії чітко впорядковані відповідно до симетрії та спіну.

**3.6.3. Систематичне порівняння для всіх атомів.** Щоб узагальнити різницю між методами UHF та РОНФ, виконаємо серію розрахунків для кількох відкритооболонкових атомів перших двох періодів. Це дозволить оцінити енергетичну різницю  $\Delta E = E_{\text{UHF}} - E_{\text{РОНФ}}$  і побачити, чи зберігає РОНФ спінову симетрію без втрати точності.

```
from pyscf import gto, scf
import numpy as np

atoms_open_shell = [
    ('H', 1), ('Li', 1), ('B', 1), ('C', 2),
    ('N', 3), ('O', 2), ('F', 1)]
```

```

]

basis = '6-31g*'

print(f'Порівняння UHF vs ROHF (базис: {basis})')
print('='*80)
print(f'{"Атом":4s} {"Спін":4s} {"E(UHF), Ha":15s} {"E(ROHF), Ha":15s} '
      f'{"ΔE, mHa":10s}')
print('-'*80)

for symbol, spin in atoms_open_shell:
    mol = gto.M(
        atom=f'{symbol} 0 0 0',
        basis=basis,
        spin=spin,
        verbose=0
    )

    # UHF
    mf_uhf = scf.UHF(mol)
    mf_uhf.verbose = 0
    mf_uhf.conv_tol = 1e-10
    e_uhf = mf_uhf.kernel()

    # ROHF
    mf_rohf = scf.ROHF(mol)
    mf_rohf.verbose = 0
    mf_rohf.conv_tol = 1e-10
    e_rohf = mf_rohf.kernel()

    delta = (e_uhf - e_rohf) * 1000 # mHa

    print(f'{symbol:4s} {spin:4d} {e_uhf:15.8f} {e_rohf:15.8f} {delta:10.4f}')

print('='*80)
print('\nПримітка: UHF зазвичай дає нижчу енергію, але з')
print('        забрудненням спіну. ROHF зберігає чисту')
print('        спінову симетрію.')

```

**Інтерпретація результатів.** Як правило, для легких атомів (H, Li, B) енергії UHF і ROHF майже збігаються. Для атомів із більшою кількістю незапарених електронів (C, N, O, F) різниця у кілька мілігаусів відображає більшу гнучкість UHF, що дозволяє частково знизити енергію ціною забруднення спіном ( $\langle S^2 \rangle > S(S + 1)$ ).

Таким чином, ROHF — це більш строгий метод із правильною спіноюю симетрією, а UHF — енергетично вигідніший, але менш «фізично чистий». У практичних обчисленнях ROHF часто слугує базовою точкою для подальших кореляційних методів (MP2, CCSD, CASSCF).

## Практичне значення

Порівняння методів RHF, UHF і ROHF дозволяє студентам зрозуміти:

- коли можна використовувати RHF (синглети закритих оболонок);
- коли необхідно застосовувати UHF (відкриті оболонки, магнітні системи);
- у яких випадках варто віддати перевагу ROHF (спінова чистота, мінімізація забруднення).

## Контрольне завдання

1. Повторіть розрахунок для атомів O (спін = 2) і N (спін = 3). Порівняйте  $\langle S^2 \rangle$  для UHF і ROHF.
2. Для кожного атома побудуйте різницю енергій UHF–ROHF у мілігаусах.
3. Проаналізуйте, як спінове забруднення зростає зі збільшенням кількості неспарених електронів.

### 3.7. Складні випадки та збіжність

Розрахунки у квантовій хімії не завжди проходять гладко. Навіть для, здавалося б, простих атомів або малих молекул можуть виникати проблеми збіжності — коли ітераційний процес самозгодженого поля (SCF) не може досягти стабільного рішення. Особливо це стосується перехідних металів, систем із сильною кореляцією електронів або випадків із виродженими орбіталями. У цьому розділі наведено практичні прийоми, які допомагають стабілізувати розрахунок, а також приклади програмного коду на базі бібліотеки PySCF.

Основні джерела труднощів:

- сильна електронна кореляція в незаповнених *d*- та *f*-оболонках;
- мала різниця енергій між орбіталями (виродження);
- вибір невдалого початкового наближення хвильової функції;
- надто жорсткі або надто м'які критерії збіжності.

У наступних підрозділах ми розглянемо конкретні стратегії для подолання таких труднощів.

**3.7.1. Перехідні метали.** Перехідні метали є одними з найпроблемніших систем для збіжності SCF. Причини — сильна кореляція електронів *d*-оболонки, множинність близьких за енергією станів спіну та значна роль релятивістичних ефектів. Навіть у межах методу Гартрі–Фока функціонал енергії має кілька локальних мінімумів, і звичайна ітераційна процедура може «застрягнути» в некоректному стані.

У PySCF це часто проявляється у вигляді:

- осциляцій енергії між ітераціями,
- дивергентного DIIS,
- різкої зміни спінового моменту між кроками.

Типовий приклад — атоми Fe, Co, Ni або їхні оксиди, де різниця між конфігураціями високого й низького спіну становить лише декілька сотих гартрі.

```
from pyscf import gto, scf

mol = gto.M(atom='Co 0 0 0', basis='def2-svp', spin=3)
mf = scf.UHF(mol).run()
print('Converged?', mf.converged)
```

Перехідні метали є класичним прикладом систем, де звичайні методи SCF часто не конвергують або дають фізично некоректний результат. Це зумовлено тим, що у таких атомів одночасно заповнюються  $3d$ - та  $4s$ -орбіталі, енергії яких близькі. У результаті електрон може “перестрибувати” між орбіталями під час ітерацій, порушуючи стабільність процесу.

У прикладі нижче подано універсальну функцію для обчислення енергії атомів перехідних металів із урахуванням практичних порад:

- використовується метод UHF (неспарені електрони важливі для коректного опису спіну);
- уведено **level shift**, що зміщує енергетичний спектр і полегшує збіжність;
- застосовано розширений DIIS-простір (метод прискорення збіжності);
- при необхідності переходять до Ньютона–Рафсона для уточнення рішення;
- результати контролюються через значення  $\langle S^2 \rangle$ .

Після кожного розрахунку порівнюється очікуване значення спіну зі знайденим, а також обчислюється “забруднення спіном” — показник, наскільки рішення відхиляється від ідеального стану.

```
from pyscf import gto, scf

def transition_metal_calculation(symbol, spin, basis='def2-svp'):
    """
    Розрахунок атома перехідного металу
    Часто потребує спеціальних налаштувань
    """

    print(f'\nРозрахунок {symbol} (2S={spin})')
    print('='*60)

    mol = gto.M(
        atom=f'{symbol} 0 0 0',
        basis=basis,
        spin=spin,
        symmetry=False, # Іноді краще без симетрії
        verbose=0
    )

    mf = scf.UHF(mol)

    # Налаштування для важких випадків
    mf.conv_tol = 1e-8
    mf.max_cycle = 200
    mf.level_shift = 0.5 # Level shift допомагає конвергенції
    mf.diis_space = 12
    mf.init_guess = 'atom'

    print('Спроба 1: UHF з level shift...')
    mf.verbose = 4
    energy = mf.kernel()

    if not mf.converged:
        print('\nНе конвергувало! Спроба 2: Newton-Raphson...')
        mf = mf.newton()
        mf.max_cycle = 50
```

```

energy = mf.kernel()

if mf.converged:
    s2 = mf.spin_square()
    expected_s2 = spin * (spin + 2) / 4

    print(f'\nРезультати:')
    print(f'  Енергія: {energy:.8f} Ha')
    print(f'  <S²>: {s2[0]:.4f} (очікується {expected_s2:.4f})')
    print(f'  Забруднення: {s2[0] - expected_s2:.4f}')
else:
    print('\nНЕ ВДАЛОСЯ ДОСЯГТИ КОНВЕРГЕНЦІЇ!')

return mf, energy if mf.converged else None

# Приклади перехідних металів
# Sc: [Ar] 3d¹ 4s², ²D
mf_sc, e_sc = transition_metal_calculation('Sc', spin=1)

# Ti: [Ar] 3d² 4s², ³F
mf_ti, e_ti = transition_metal_calculation('Ti', spin=2)

# Cr: [Ar] 3d⁵ 4s¹, ⁷S
mf_cr, e_cr = transition_metal_calculation('Cr', spin=6)

# Mn: [Ar] 3d⁵ 4s², ⁶S
mf_mn, e_mn = transition_metal_calculation('Mn', spin=5)

# Fe: [Ar] 3d⁶ 4s², ⁵D
mf_fe, e_fe = transition_metal_calculation('Fe', spin=4)

```

Такий підхід забезпечує стабільність навіть для важких атомів — наприклад, Cr, Mn або Fe, де стандартний SCF часто не сходиться.

**3.7.2. Використання дробових заповнень.** Ще одна причина проблем зі збіжністю — наявність вироджених або майже вироджених орбіталей. У таких випадках система “вагається”, яку саме конфігурацію обрати, і стандартний SCF може осцилювати без досягнення стабільного мінімуму.

Один із ефективних підходів — застосування *дробових заповнень орбіталей* (англ. *fractional occupations*). Ідея полягає у тому, що замість жорсткого заповнення 0 або 1 для кожної орбіталі вводиться часткове заповнення, яке імітує теплове розмазування Фермі–Дірака:

$$f_i = \frac{1}{1 + e^{(\varepsilon_i - \mu)/kT}},$$

де  $f_i$  — ефективне заповнення  $i$ -ї орбіталі,  $\varepsilon_i$  — її енергія,  $\mu$  — хімічний потенціал, а  $kT$  — параметр “розмазування”.

Цей підхід згладжує різкі переходи між виродженими орбіталями і дозволяє SCF знайти стабільне рішення.

Нижче наведено приклад для атома ванадію, у якого типово спостерігаються проблеми збіжності через близькість енергетичних рівнів 3d і 4s.

```
from pyscf import gto, scf

# Важкий випадок: атом з близькими за енергією орбіталями
mol = gto.M(
    atom='V 0 0 0',
    basis='cc-pvdz',
    spin=3,
    verbose=0
)

print('Розрахунок V з дробовими заповненнями')
print('='*60)

# Стандартний UHF може не конвергувати
mf = scf.UHF(mol)
mf.verbose = 0
mf.max_cycle = 100

try:
    energy = mf.kernel()
    if not mf.converged:
        raise RuntimeError('Не конвергувало')
except:
    print('Стандартний UHF не конвергував')

# Використання дробових заповнень (smearing)
print('\nСпроба з дробовими заповненнями...')

mf = scf.UHF(mol)
mf = scf.addons.frac_occ(mf)
mf.verbose = 4
energy = mf.kernel()

if mf.converged:
    print(f'\nУспішно! Енергія: {energy:.8f} Ha')
else:
    print('\nВсе одно не конвергувало')
```

Як видно, якщо звичайний UHF не конвергує, можна скористатися `scf.addons.frac_occ(mf)`, що вмикає дробові заповнення. Методика добре працює для окремих атомів, радикалів і навіть для малих кластерів перехідних металів.

**3.7.3. Стратегії досягнення конвергенції.** Проблема збіжності — одна з найчастіших у практиці квантово-хімічних розрахунків. Ітераційна процедура самозгодженого поля (SCF) повинна забезпечити стабільне розв’язання рівнянь Гартрі–Фока, однак у реальних системах — особливо для відкритих оболонок, перехідних металів або вироджених станів — енергія може коливатися, DIIS може втрачати стабільність, а розрахунок сходиться до нефізичного мінімуму.

Для таких випадків доцільно застосовувати *послідовну стратегію стабілізації SCF*, у якій різні методи збільшення стійкості перевіряються крок за кроком — від простих до складніших.

**Основні прийоми стабілізації:**

1. стандартний UHF;
2. зсув рівнів (`level_shift`), що зменшує вплив віртуальних орбіталей;
3. збільшення DIIS-простору (`diis_space`);
4. атомне початкове наближення (`init_guess='atom'`);
5. уточнення за методом Ньютона-Рафсона;
6. дробові заповнення (`frac_occ`) при вироджених станах.

Нижче подано узагальнений алгоритм, який автоматично перебирає ці стратегії для пошуку збіжного рішення.

```
from pyscf import gto, scf

def convergence_strategies(symbol, spin, basis='def2-svp'):
    """
    Покрокова стабілізація SCF для складних систем
    """
    mol = gto.M(atom=f'{symbol} 0 0 0', basis=basis, spin=spin, verbose=0)

    strategies = [
        ('Стандартний UHF', {}),
        ('UHF з level shift', {'level_shift': 0.3}),
        ('UHF з більшим DIIS', {'diis_space': 15}),
        ('UHF з атомним guess', {'init_guess': 'atom'}),
        ('Метод Ньютона-Рафсона', {'newton': True}),
        ('Дробові заповнення', {'frac_occ': True}),
    ]

    print(f'=== Тестування стратегій конвергенції для {symbol} ===')

    for name, params in strategies:
        print(f'\n→ {name}')
        mf = scf.UHF(mol)
        mf.max_cycle = 100
        mf.conv_tol = 1e-9

        for k, v in params.items():
            if k in ('newton', 'frac_occ'):
                continue
            setattr(mf, k, v)

        try:
            if params.get('newton'):
                mf = mf.newton()
            if params.get('frac_occ'):
                mf = scf.addons.frac_occ(mf)
            energy = mf.kernel()
            if mf.converged:
                print(f'Успіх: E = {energy:.8f} Ha')
                return mf, energy
            else:
                print('Не конвергувало')
        except Exception as e:
            print(f'Помилка: {e}')
    print('\nЖодна стратегія не спрацювала.')
    return None, None

# Приклад: перехідний метал
mf, e = convergence_strategies('Co', spin=3)
```

**Фізичний сенс прийомів.**

- **Level shift.** Тимчасово підвищує енергії віртуальних орбіталей, запобігаючи коливанням поля.
- **DIIS-простір.** Збільшення пам'яті історії ітерацій покращує апроксимацію самозгодженого стану.
- **Atom guess.** Використовує атомні орбіталі як старт, що ближче до фізичного рішення.
- **Newton–Raphson.** Прискорює збіжність у зоні поблизу мінімуму функціоналу енергії.
- **Fractional occupations.** Згладжують виродження, дозволяючи системі знайти стабільний середній стан.

#### Практичні рекомендації.

- Для великих систем зменшіть точність (`conv_tol=1e-6`) для попереднього тесту.
- Якщо спостерігаються осциляції енергії — додайте `level_shift=0.5`.
- Якщо UHF не сходиться — спробуйте `init_guess='atom'` або `symmetry=False`.
- Завжди перевіряйте фізичність розв'язку через  $\langle S^2 \rangle$ .

Таким чином, навіть якщо стандартний SCF не збігається, послідовне застосування описаних методів практично гарантує знаходження стабільного фізично коректного рішення.

## 3.8. Практичні завдання

### 3.8.1. Завдання 1: Систематичне дослідження.

```
"""
ЗАВДАННЯ 1: Розрахуйте енергії всіх атомів першого періоду
(H, He) з різними базисними наборами. Побудуйте графік
збіжності енергії до базисної межі.
```

```
Базиси: sto-3g, 6-31g, cc-pvdz, cc-pvtz, cc-pvqz, cc-pv5z
"""
```

```
from pyscf import gto, scf
import matplotlib.pyplot as plt
```

```
# Ваш код тут
```

### 3.8.2. Завдання 2: Енергії іонізації.

```
"""
ЗАВДАННЯ 2: Обчисліть першу та другу енергії іонізації
для атомів Li, Be, B, C, N, O, F, Ne. Порівняйте з
експериментальними значеннями.
```

```
IE1 = E(A+) - E(A)
IE2 = E(A2+) - E(A+)
```

```
Використайте базис cc-pvtz
```

```
"""
# Ваш код тут
```

### 3.8.3. Завдання 3: Спектроскопічні константи.

```
"""
ЗАВДАННЯ 3: Для атома Карбону розрахуйте енергетичну
різницю між основним триплетним станом ( $^3P$ ) та
збудженими станами ( $^1D$  та  $^1S$ ).
```

```
Підказка: використайте різні спінові конфігурації
"""
```

```
# Ваш код тут
```

### 3.8.4. Завдання 4: Залежність від базису.

```
"""
ЗАВДАННЯ 4: Дослідіть, як додавання дифузних функцій
впливає на енергію та дипольну поляризованість атома
Кисню ( $O$ ).
```

```
Порівняйте: cc-pvdz vs aug-cc-pvdz, cc-pvtz vs aug-cc-pvtz
"""
```

```
# Ваш код тут
```

## 3.9. Резюме

У цьому розділі ми детально вивчили метод Хартрі-Фока для атомних систем:

- **Теоретичні основи** — рівняння HF, детермінант Слейтера, оператор Фока
- **Варіанти методу** — RHF для замкнених оболонок, UHF для відкритих, ROHF як компроміс
- **Одноелектронні системи** — H атом як тестовий випадок
- **Багатоелектронні атоми** — He та атоми другого періоду
- **Аналіз результатів** — орбітальні енергії, заселеності, спінова густина
- **Складні випадки** — перехідні метали, стратегії конвергенції

### 3.9.1. Ключові висновки.

1. Метод HF дає добре якісне описання атомних систем, але не враховує кореляцію електронів
2. Вибір між RHF/UHF/ROHF залежить від спінової структури системи
3. UHF страждає від забруднення спіном, але зазвичай дає нижчу енергію
4. Для важких атомів (перехідні метали) потрібні спеціальні техніки конвергенції

5. Якість результатів сильно залежить від вибору базисного набору

У наступному розділі ми розглянемо теорію функціоналу густини (DFT), яка часто дає кращі результати для багатоелектронних систем.

# 4

## Теорія функціоналу густини (DFT)

### 4.1. Основи теорії функціоналу густини

**4.1.1. Теорема Хюенберга–Кона.** Теорія функціоналу густини базується на двох фундаментальних теоремах, доведених Хюенбергом та Коном у 1964 році:

**Теорема 1 (Теорема існування).** Зовнішній потенціал  $v_{\text{ext}}(\mathbf{r})$  (а отже, і повна енергія системи) однозначно визначається електронною густиною основного стану  $\rho(\mathbf{r})$  з точністю до адитивної константи.

Це означає, що електронна густина містить всю інформацію про систему:

$$E[\rho] = T[\rho] + V_{ee}[\rho] + \int v_{\text{ext}}(\mathbf{r})\rho(\mathbf{r})d\mathbf{r} \quad (4.1)$$

**Теорема 2 (Варіаційний принцип).** Існує універсальний функціонал енергії  $F[\rho]$ , який для будь-якої пробної густини  $\tilde{\rho}(\mathbf{r})$  задовольняє:

$$E_0 \leq E[\tilde{\rho}] = F[\tilde{\rho}] + \int v_{\text{ext}}(\mathbf{r})\tilde{\rho}(\mathbf{r})d\mathbf{r} \quad (4.2)$$

де  $E_0$  — точна енергія основного стану.

**4.1.2. Рівняння Кона–Шема.** Кон та Шем (1965) запропонували практичний підхід до DFT, замінивши взаємодіючу систему еквівалентною невзаємодіючою системою з такою ж густиною:

$$\left[-\frac{1}{2}\nabla^2 + v_{\text{eff}}(\mathbf{r})\right]\psi_i(\mathbf{r}) = \varepsilon_i\psi_i(\mathbf{r}) \quad (4.3)$$

Ефективний потенціал визначається як:

$$v_{\text{eff}}(\mathbf{r}) = v_{\text{ext}}(\mathbf{r}) + v_H(\mathbf{r}) + v_{xc}(\mathbf{r}) \quad (4.4)$$

де:

- $v_{\text{ext}}(\mathbf{r})$  — зовнішній потенціал (від ядер)

- $v_H(\mathbf{r}) = \int \frac{\rho(\mathbf{r}')}{\|\mathbf{r}-\mathbf{r}'\|} d\mathbf{r}'$  — потенціал Хартрі
- $v_{xc}(\mathbf{r}) = \frac{\delta E_{xc}[\rho]}{\delta \rho(\mathbf{r})}$  — обмінно-кореляційний потенціал

Електронна густина обчислюється через орбіталі Кона–Шема:

$$\rho(\mathbf{r}) = \sum_{i=1}^N |\psi_i(\mathbf{r})|^2 \quad (4.5)$$

#### 4.1.3. Обмінно-кореляційна енергія. Повна енергія в DFT:

$$E[\rho] = T_s[\rho] + V_{ext}[\rho] + J[\rho] + E_{xc}[\rho] \quad (4.6)$$

де  $E_{xc}[\rho]$  — обмінно-кореляційна енергія, яка містить:

- Різницю між точною кінетичною енергією та кінетичною енергією незалежної системи
- Некласичну частину електрон-електронного відштовхування (обмін + кореляція)

Точний вигляд  $E_{xc}[\rho]$  невідомий, тому використовуються наближення (функціонали).

#### 4.1.4. Порівняння HF та DFT.

Рис. 4.1. Порівняння методів Хартрі-Фока та DFT

Аспект	Hartree-Fock	DFT
Базова змінна	Хвильова функція	Електронна густина
Обмін	Точний (нелокальний)	Наближений (локальний)
Кореляція	Відсутня	Включена наближено
Масштабування	$\mathcal{O}(N^4)$	$\mathcal{O}(N^3)$
Точність для атомів	Добра якісно	Часто краща кількісно
Збуджені стани	Можливі	Складно (TD-DFT)

## 4.2. Функціонали обміну-кореляції

**4.2.1. Класифікація функціоналів: драбина Якова.** Функціонали DFT класифікуються за "драбиною Якова" (Jacob's ladder), запропонованою Пердью:

1. **LDA/LSDA** (Local Density Approximation) — залежать тільки від  $\rho(\mathbf{r})$
2. **GGA** (Generalized Gradient Approximation) — залежать від  $\rho(\mathbf{r})$  та  $\nabla\rho(\mathbf{r})$
3. **Meta-GGA** — додатково залежать від  $\nabla^2\rho$  або кінетичної густини  $\tau$
4. **Hybrid** — включають частку точного обміну HF
5. **Double-hybrid** — включають і HF обмін, і MP2 кореляцію

### 4.2.2. LDA та LSDA функціонали.

#### Локальне наближення густини

LDA базується на моделі однорідного електронного газу:

$$E_{xc}^{LDA}[\rho] = \int \rho(\mathbf{r}) \varepsilon_{xc}(\rho(\mathbf{r})) d\mathbf{r} \quad (4.7)$$

де  $\varepsilon_{xc}(\rho)$  — обмінно-кореляційна енергія на електрон в однорідному газі густини  $\rho$ .

#### Обмінна частина (Slater/Dirac):

$$\varepsilon_x^{LDA}(\rho) = -\frac{3}{4} \left( \frac{3}{\pi} \right)^{1/3} \rho^{1/3} \quad (4.8)$$

**Кореляційна частина:** Використовуються параметризації (VWN, PW92).

#### LSDA для спін-поляризованих систем

Для систем з різними  $\rho_\alpha$  та  $\rho_\beta$ :

$$E_{xc}^{LSDA}[\rho_\alpha, \rho_\beta] = \int \rho \varepsilon_{xc}(\rho_\alpha, \rho_\beta) d\mathbf{r} \quad (4.9)$$

```
from pyscf import gto, dft

# Приклад LDA розрахунку атома Ne
mol = gto.M(
    atom='Ne 0 0 0',
    basis='cc-pvdz',
    spin=0
)

# LDA функціонал (VWN для кореляції)
mf = dft.RKS(mol)
mf.xc = 'lda,vwn' # або просто 'lda'
mf.verbose = 4
energy_lda = mf.kernel()

print(f'\nЕнергія Ne (LDA): {energy_lda:.8f} Ha')

# Порівняння з HF
from pyscf import scf
mf_hf = scf.RHF(mol)
mf_hf.verbose = 0
energy_hf = mf_hf.kernel()

print(f'Енергія Ne (HF): {energy_hf:.8f} Ha')
print(f'Різниця (LDA-HF): {(energy_lda-energy_hf)*1000:.2f} mHa')
```

**4.2.3. GGA функціонали.** GGA функціонали враховують не тільки локальну густину, але й її градієнт:

$$E_{xc}^{GGA}[\rho] = \int f(\rho(\mathbf{r}), \nabla\rho(\mathbf{r}))d\mathbf{r} \quad (4.10)$$

### Популярні GGA функціонали

**PBE (Perdew-Burke-Ernzerhof, 1996)** Найпопулярніший неемпіричний GGA функціонал:

```
from pyscf import gto, dft

mol = gto.M(atom='C 0 0 0', basis='cc-pvtz', spin=2)

mf = dft.UKS(mol)
mf.xc = 'pbe' # або 'pbe,pbe'
energy_pbe = mf.kernel()

print(f'Енергія C (PBE): {energy_pbe:.8f} Ha')
```

**BLYP (Becke88 + Lee-Yang-Parr)** Комбінація обміну Becke88 та кореляції LYP:

```
mf = dft.UKS(mol)
mf.xc = 'blyp' # або 'b88,Lyp'
energy_blyp = mf.kernel()

print(f'Енергія C (BLYP): {energy_blyp:.8f} Ha')
```

### BP86 (Becke88 + Perdew86)

```
mf = dft.UKS(mol)
mf.xc = 'bp86'
energy_bp86 = mf.kernel()

print(f'Енергія C (BP86): {energy_bp86:.8f} Ha')
```

### Порівняння GGA функціоналів

```
from pyscf import gto, dft
import numpy as np

# Тестування на атомі Кисню
mol = gto.M(
    atom='O 0 0 0',
    basis='def2-tzvp',
    spin=2
)

gga_functionals = ['pbe', 'blyp', 'bp86', 'pw91', 'pberev']
```

```

print('Порівняння GGA функціоналів для атома O (³P)')
print('='*60)
print(f'{"Функціонал":12s} {"Енергія, Ha":15s} {"Відносно PBE, mHa":20s}')
print('='*60)

energies = {}

for xc in gga_functionals:
    mf = dft.UKS(mol)
    mf.xc = xc
    mf.verbose = 0
    mf.conv_tol = 1e-10

    try:
        energy = mf.kernel()
        energies[xc] = energy

        if xc == 'pbe':
            e_ref = energy
            rel = 0.0
        else:
            rel = (energy - e_ref) * 1000

        print(f'{xc:12s} {energy:15.8f} {rel:20.4f}')
    except:
        print(f'{xc:12s} --- помилка розрахунку')

print('='*60)

```

**4.2.4. Meta-GGA функціонали.** Meta-GGA функціонали включають додаткову інформацію про систему: кінетичну густину  $\tau$  або лапласіан густини  $\nabla^2\rho$ :

$$E_{xc}^{\text{meta-GGA}}[\rho] = \int f(\rho, \nabla\rho, \tau) d\mathbf{r} \quad (4.11)$$

де кінетична густина орбіталей Кона-Шема:

$$\tau(\mathbf{r}) = \frac{1}{2} \sum_{i=1}^N |\nabla\psi_i(\mathbf{r})|^2 \quad (4.12)$$

### TPSS (Tao-Perdew-Staroverov-Scuseria)

TPSS — один з перших успішних meta-GGA функціоналів (2003):

```

from pyscf import gto, dft

mol = gto.M(
    atom='Fe 0 0 0',
    basis='def2-tzvp',
    spin=4 # ⁵D основний стан
)

# TPSS meta-GGA
mf = dft.UKS(mol)
mf.xc = 'tpss'

```

#### 4. Теорія функціоналу густини (DFT)

---

```
mf.verbose = 4
energy_tpss = mf.kernel()

print(f'\nЕнергія Fe (TPSS): {energy_tpss:.8f} Ha')

# Порівняння з PBE
mf_pbe = dft.UKS(mol)
mf_pbe.xc = 'pbe'
mf_pbe.verbose = 0
energy_pbe = mf_pbe.kernel()

print(f'Енергія Fe (PBE): {energy_pbe:.8f} Ha')
print(f'Різниця (TPSS-PBE): {(energy_tpss-energy_pbe)*1000:.2f} мHa')
```

---

### M06-L (Minnesota 06 Local)

Високопараметризований meta-GGA функціонал для широкого спектру задач:

```
from pyscf import gto, dft

mol = gto.M(atom='Ni 0 0 0', basis='def2-svp', spin=2)

mf = dft.UKS(mol)
mf.xc = 'm06l' # або 'm06-L'
mf.verbose = 4

try:
    energy_m06l = mf.kernel()
    print(f'\nЕнергія Ni (M06-L): {energy_m06l:.8f} Ha')
except:
    print('M06-L може бути недоступний у вашій версії PySCF')
    print('Встановіть: pip install pyscf[geomopt]')
```

---

### SCAN (Strongly Constrained and Appropriately Normed)

Сучасний meta-GGA, що задовольняє всі відомі точні умови:

```
from pyscf import gto, dft

mol = gto.M(atom='C 0 0 0', basis='def2-qzvp', spin=2)

mf = dft.UKS(mol)
mf.xc = 'scan'
energy_scan = mf.kernel()

print(f'Енергія C (SCAN): {energy_scan:.8f} Ha')
```

---

**4.2.5. Гібридні функціонали.** Гібридні функціонали комбінують DFT обмін з точним (HF) обміном:

$$E_{xc}^{\text{hybrid}} = a \cdot E_x^{\text{HF}} + (1 - a) \cdot E_x^{\text{DFT}} + E_c^{\text{DFT}} \quad (4.13)$$

де  $a$  — частка HF обміну (зазвичай 0.2–0.3).

**B3LYP (Becke 3-parameter Lee-Yang-Parr)**

Найпопулярніший гібридний функціонал у хімії:

$$E_{xc}^{B3LYP} = E_x^{LDA} + 0.2(E_x^{HF} - E_x^{LDA}) + 0.72 \cdot E_x^{B88} + 0.81 \cdot E_c^{LYP} + 0.19 \cdot E_c^{VWN} \quad (4.14)$$

```
from pyscf import gto, dft

mol = gto.M(
    atom='N 0 0 0',
    basis='6-311+g(d,p)',
    spin=3
)

# B3LYP розрахунок
mf = dft.UKS(mol)
mf.xc = 'b3lyp'
mf.verbose = 4
energy_b3lyp = mf.kernel()

print(f'\nЕнергія N (B3LYP): {energy_b3lyp:.8f} Ha')

# Аналіз компонентів
print(f'\nВнесок точного обміну: 20%')
print(f'Це робить розрахунок повільнішим, але точнішим')
```

**PBE0 (PBE hybrid)**

Неемпіричний гібрид з 25

$$E_{xc}^{PBE0} = 0.25 \cdot E_x^{HF} + 0.75 \cdot E_x^{PBE} + E_c^{PBE} \quad (4.15)$$

```
from pyscf import gto, dft

mol = gto.M(atom='O 0 0 0', basis='aug-cc-pvtz', spin=2)

mf = dft.UKS(mol)
mf.xc = 'pbe0'
energy_pbe0 = mf.kernel()

print(f'Енергія O (PBE0): {energy_pbe0:.8f} Ha')
```

**CAM-B3LYP (Coulomb-Attenuated Method)**

Функціонал з дальньодіючою корекцією (range-separated):

$$\frac{1}{r_{12}} = \frac{\alpha + \beta \cdot \text{erf}(\mu r_{12})}{r_{12}} + \frac{1 - [\alpha + \beta \cdot \text{erf}(\mu r_{12})]}{r_{12}} \quad (4.16)$$

```
from pyscf import gto, dft

mol = gto.M(atom='F 0 0 0', basis='cc-pvqz', spin=1)
```

#### 4. Теорія функціоналу густини (DFT)

```
mf = dft.UKS(mol)
mf.xc = 'camb3lyp' # або 'cam-b3lyp'
energy_camb3lyp = mf.kernel()

print(f'Енергія F (CAM-B3LYP): {energy_camb3lyp:.8f} Ha')
```

### M06-2X та інші Minnesota функціонали

Родина M06 з різною часткою HF обміну:

- M06-L: 0% HF (meta-GGA)
- M06: 27% HF
- M06-2X: 54% HF (для кінетики, слабких взаємодій)
- M06-HF: 100% HF

```
from pyscf import gto, dft

mol = gto.M(atom='Ar 0 0 0', basis='def2-tzvp', spin=0)

# Порівняння M06 функціоналів
m06_functionals = {
    'm06l': 'M06-L (0% HF)',
    'm06': 'M06 (27% HF)',
    'm062x': 'M06-2X (54% HF)'
}

print('Порівняння M06 функціоналів для Ar')
print('='*60)

for xc, name in m06_functionals.items():
    mf = dft.RKS(mol)
    mf.xc = xc
    mf.verbose = 0

    try:
        energy = mf.kernel()
        print(f'{name:20s}: {energy:.8f} Ha')
    except:
        print(f'{name:20s}: недоступний')
```

#### 4.2.6. Порівняння різних рівнів теорії.

```
from pyscf import gto, scf, dft
import numpy as np
import matplotlib.pyplot as plt

def compare_functionals(symbol, spin, basis='cc-pvtz'):
    """
    Систематичне порівняння функціоналів
    """

    mol = gto.M(
        atom=f'{symbol} 0 0 0',
        basis=basis,
        spin=spin,
        verbose=0
    )
```

```

# Список методів для порівняння
methods = [
    ('HF', 'scf'),
    ('LDA', 'lda'),
    ('PBE', 'pbe'),
    ('BLYP', 'blyp'),
    ('TPSS', 'tpss'),
    ('B3LYP', 'b3lyp'),
    ('PBE0', 'pbe0'),
    ('CAM-B3LYP', 'camb3lyp'),
]

energies = []
labels = []

print(f'\nПорівняння методів для {symbol} (базис: {basis})')
print('='*70)
print(f'{"Метод":15s} {"Енергія, Ha":15s} {"Відносно HF, mHa":20s}')
print('-'*70)

for name, method in methods:
    try:
        if method == 'scf':
            if spin == 0:
                mf = scf.RHF(mol)
            else:
                mf = scf.UHF(mol)
        else:
            if spin == 0:
                mf = dft.RKS(mol)
            else:
                mf = dft.UKS(mol)
            mf.xc = method

        mf.verbose = 0
        mf.conv_tol = 1e-10
        energy = mf.kernel()

        if mf.converged:
            energies.append(energy)
            labels.append(name)

            if name == 'HF':
                e_ref = energy
                rel = 0.0
            else:
                rel = (energy - e_ref) * 1000

            print(f'{name:15s} {energy:15.8f} {rel:20.4f}')
        else:
            print(f'{name:15s} не конвергувало')
    except Exception as e:
        print(f'{name:15s} помилка: {str(e)[:30]}')

print('='*70)

# Графік
if len(energies) > 1:
    fig, ax = plt.subplots(figsize=(10, 6))

    x = np.arange(len(labels))
    colors = ['red' if l == 'HF' else
              'blue' if l in ['LDA', 'PBE', 'BLYP', 'TPSS'] else
              'green' for l in labels]

```

```

bars = ax.bar(x, energies, color=colors, alpha=0.7)
ax.set_xticks(x)
ax.set_xticklabels(labels, rotation=45, ha='right')
ax.set_ylabel('Енергія (Ha)', fontsize=12)
ax.set_title(f'Порівняння методів для атома {symbol}',
             fontsize=14)
ax.grid(True, alpha=0.3, axis='y')

# Легенда
from matplotlib.patches import Patch
legend_elements = [
    Patch(facecolor='red', alpha=0.7, label='HF'),
    Patch(facecolor='blue', alpha=0.7, label='Pure DFT'),
    Patch(facecolor='green', alpha=0.7, label='Hybrid DFT')
]
ax.legend(handles=legend_elements, loc='best')

plt.tight_layout()
plt.savefig(f'{symbol}_functionals_comparison.pdf')
plt.show()

return energies, labels

# Тестування на різних атомах
energies_c, labels_c = compare_functionals('C', spin=2)
energies_ne, labels_ne = compare_functionals('Ne', spin=0)
energies_fe, labels_fe = compare_functionals('Fe', spin=4,
                                             basis='def2-svp')

```

Таблиця 4.1. Рекомендовані функціонали для різних задач

Задача	Функціонал	Коментар
Швидкі розрахунки	PBE	Добра точність/швидкість
Енергії атомізації	B3LYP, PBE0	Стандарт у хімії
Перехідні метали	TPSSh, M06	Краще для d-електронів
Слабкі взаємодії	M06-2X, $\omega$ B97X-D	З дисперсією
Високоспінові стани	SCAN, TPSSh	Кращий баланс
Загальні розрахунки	PBE0	Універсальний вибір
Максимальна точність	CCSD(T)	Але дуже повільно

#### 4.2.7. Вибір функціоналу: рекомендації.

#### 4.2.8. Практичний приклад: вплив функціоналу на властивості.

```

from pyscf import gto, scf, dft

def analyze_functional_effect(symbol, charge, spin_n, spin_c,
                             basis='aug-cc-pvtz'):
    """
    Аналіз впливу функціоналу на енергію іонізації
    """

```

```

functionals = ['pbe', 'blyp', 'b3lyp', 'pbe0', 'cam-b3lyp']

print(f'\nЕнергії іонізації {symbol} (базис: {basis})')
print('='*70)
print(f'{"Метод":15s} {"E(A), Ha":15s} {"E(A+), Ha":15s} '
      f'{"IE, eV":10s}')
print('-'*70)

# HF референс
mol_n = gto.M(atom=f'{symbol} 0 0 0', basis=basis,
              charge=charge, spin=spin_n, verbose=0)
mol_c = gto.M(atom=f'{symbol} 0 0 0', basis=basis,
              charge=charge+1, spin=spin_c, verbose=0)

if spin_n == 0:
    mf_n = scf.RHF(mol_n)
else:
    mf_n = scf.UHF(mol_n)

if spin_c == 0:
    mf_c = scf.RHF(mol_c)
else:
    mf_c = scf.UHF(mol_c)

mf_n.verbose = 0
mf_c.verbose = 0

e_n_hf = mf_n.kernel()
e_c_hf = mf_c.kernel()
ie_hf = (e_c_hf - e_n_hf) * 27.211386

print(f'{"HF":15s} {e_n_hf:15.8f} {e_c_hf:15.8f} {ie_hf:10.4f}')

# DFT функціонали
for xc in functionals:
    if spin_n == 0:
        mf_n = dft.RKS(mol_n)
    else:
        mf_n = dft.UKS(mol_n)

    if spin_c == 0:
        mf_c = dft.RKS(mol_c)
    else:
        mf_c = dft.UKS(mol_c)

    mf_n.xc = xc
    mf_c.xc = xc
    mf_n.verbose = 0
    mf_c.verbose = 0

    try:
        e_n = mf_n.kernel()
        e_c = mf_c.kernel()
        ie = (e_c - e_n) * 27.211386

        print(f'{xc.upper():15s} {e_n:15.8f} {e_c:15.8f} '
              f'{"ie:10.4f"}')
    except:
        print(f'{xc.upper():15s} помилка розрахунку')

print('='*70)

```

# Приклади

```

analyze_functional_effect('C', 0, 2, 1)  # C → C+
analyze_functional_effect('O', 0, 2, 3)  # O → O+

```

```
analyze_functional_effect('Ne', 0, 0, 1) # Ne → Ne+
```

### 4.3. DFT розрахунки атомів

**4.3.1. Базова структура DFT розрахунку.** DFT розрахунки в PySCF використовують класи RKS (Restricted Kohn-Sham) та UKS (Unrestricted Kohn-Sham), аналогічні до RHF/UHF:

```
from pyscf import gto, dft

# Замкнена оболонка (RKS)
mol_he = gto.M(atom='He 0 0 0', basis='cc-pvtz', spin=0)
mf_he = dft.RKS(mol_he)
mf_he.xc = 'pbe0'
e_he = mf_he.kernel()

print(f'He (RKS/PBE0): {e_he:.8f} Ha')

# Відкрита оболонка (UKS)
mol_li = gto.M(atom='Li 0 0 0', basis='cc-pvtz', spin=1)
mf_li = dft.UKS(mol_li)
mf_li.xc = 'pbe0'
e_li = mf_li.kernel()

print(f'Li (UKS/PBE0): {e_li:.8f} Ha')
```

### 4.3.2. Систематичний розрахунок атомів другого періоду.

```
from pyscf import gto, dft
import numpy as np

def dft_second_period(functional='pbe', basis='def2-tzvp'):
    """
    Розрахунок всіх атомів другого періоду
    """

    atoms_data = [
        ('Li', 1, '2S'), ('Be', 0, '1S'), ('B', 1, '2P'),
        ('C', 2, '3P'), ('N', 3, '4S'), ('O', 2, '3P'),
        ('F', 1, '2P'), ('Ne', 0, '1S')
    ]

    print(f'\nАтоми 2-го періоду ({functional.upper()}/{basis})')
    print('='*80)
    print(f'{"Атом":4s} {"Терм":6s} {"2S":3s} {"Енергія, Ha":15s} '
          f'{"Енергія, eV":12s} {"<S^2>":8s}')
    print('='*80)

    energies = {}

    for symbol, spin, term in atoms_data:
        mol = gto.M(
            atom=f'{symbol} 0 0 0',
            basis=basis,
            spin=spin,
            symmetry=True,
            verbose=0
```

```

    )

    if spin == 0:
        mf = dft.RKS(mol)
    else:
        mf = dft.UKS(mol)

    mf.xc = functional
    mf.conv_tol = 1e-10
    energy = mf.kernel()

    energies[symbol] = energy
    e_ev = energy * 27.211386

    if spin == 0:
        s2 = 0.0
    else:
        s2_result = mf.spin_square()
        s2 = s2_result[0]

    print(f'{symbol:4s} {term:6s} {spin:3d} {energy:15.8f} '
          f'{e_ev:12.2f} {s2:8.4f}')

print('='*80)

return energies

# Розрахунки з різними функціоналами
energies_pbe = dft_second_period('pbe')
energies_b3lyp = dft_second_period('b3lyp')
energies_pbe0 = dft_second_period('pbe0')

```

**4.3.3. Розрахунок атомів перехідних металів.** Атоми перехідних металів — складна задача через багато близьких за енергією станів:

```

from pyscf import gto, dft

def transition_metal_dft(symbol, spin, functional='tpss',
                        basis='def2-tzvp'):
    """
    DFT розрахунок атома перехідного металу
    """

    print(f'\nРозрахунок {symbol} (2S={spin}, {functional.upper()}')
    print('='*60)

    mol = gto.M(
        atom=f'{symbol} 0 0 0',
        basis=basis,
        spin=spin,
        symmetry=False, # Часто краще без симетрії
        verbose=0
    )

    mf = dft.UKS(mol)
    mf.xc = functional

    # Налаштування для важких випадків
    mf.conv_tol = 1e-8
    mf.max_cycle = 200
    mf.diis_space = 12

```

```
# Для перехідних металів часто потрібен level shift
if symbol in ['Cr', 'Mn', 'Fe', 'Co', 'Ni']:
    mf.level_shift = 0.3

mf.verbose = 4
energy = mf.kernel()

if mf.converged:
    s2 = mf.spin_square()
    expected_s2 = spin * (spin + 2) / 4

    print(f'\nРезультати:')
    print(f'  Енергія: {energy:.8f} Ha')
    print(f'  <S²>: {s2[0]:.4f} (очікується {expected_s2:.4f})')
    print(f'  Забруднення спіном: {s2[0] - expected_s2:.4f}')

    # Заселеності d-орбіталей
    from pyscf import lo
    pop = mf.mulliken_pop()

    return energy, s2[0]
else:
    print('\nНе конвергувало!')
    return None, None

# Приклади 3d металів
# Sc: [Ar] 3d¹ 4s², ²D
e_sc, s2_sc = transition_metal_dft('Sc', spin=1, functional='pbe')

# Ti: [Ar] 3d² 4s², ³F
e_ti, s2_ti = transition_metal_dft('Ti', spin=2, functional='pbe')

# V: [Ar] 3d³ 4s², ⁴F
e_v, s2_v = transition_metal_dft('V', spin=3, functional='pbe')

# Cr: [Ar] 3d⁵ 4s¹, ⁷S (виняток!)
e_cr, s2_cr = transition_metal_dft('Cr', spin=6, functional='pbe')

# Mn: [Ar] 3d⁵ 4s², ⁶S
e_mn, s2_mn = transition_metal_dft('Mn', spin=5, functional='pbe')

# Fe: [Ar] 3d⁶ 4s², ⁵D
e_fe, s2_fe = transition_metal_dft('Fe', spin=4, functional='pbe')
```

**4.3.4. Порівняння спінових станів.** Для деяких атомів важливо порівняти різні спінові стани:

```
from pyscf import gto, dft
import matplotlib.pyplot as plt

def compare_spin_states(symbol, spin_list, functional='pbe0',
                        basis='def2-tzvp'):
    """
    Порівняння енергій різних спінових станів
    """

    print(f'\nПорівняння спінових станів {symbol}')
    print(f'Функціонал: {functional.upper()}, базис: {basis}')
    print('='*70)
    print(f'{"2S":5s} {"Mult":5s} {"Енергія, Ha":15s} '
```

```

    f'{"Відносна, kcal/mol":20s}')
print('-'*70)

energies = []
spins = []

for spin in spin_list:
    mol = gto.M(
        atom=f'{symbol} 0 0 0',
        basis=basis,
        spin=spin,
        verbose=0
    )

    if spin == 0:
        mf = dft.RKS(mol)
    else:
        mf = dft.UKS(mol)

    mf.xc = functional
    mf.conv_tol = 1e-10
    mf.max_cycle = 150

    try:
        energy = mf.kernel()

        if mf.converged:
            energies.append(energy)
            spins.append(spin)

            mult = spin + 1

            if len(energies) == 1:
                e_ref = energy
                rel = 0.0
            else:
                rel = (energy - e_ref) * 627.509 # kcal/mol

            marker = ' ← найнижча' if energy == min(energies) else ''
            print(f'{spin:5d} {mult:5d} {energy:15.8f} '
                  f'{rel:20.4f}{marker}')
        except:
            print(f'{spin:5d} {spin+1:5d} не конвергувало')

print('-'*70)

# Графік
if len(energies) > 1:
    fig, ax = plt.subplots(figsize=(10, 6))

    # Відносні енергії в kcal/mol
    e_min = min(energies)
    rel_energies = [(e - e_min) * 627.509 for e in energies]

    ax.plot(spins, rel_energies, 'o-', markersize=10, linewidth=2)
    ax.axhline(y=0, color='gray', linestyle='--', alpha=0.5)

    ax.set_xlabel('2S', fontsize=12)
    ax.set_ylabel('Відносна енергія (kcal/mol)', fontsize=12)
    ax.set_title(f'Спінові стани {symbol} '
                  f'({functional.upper()})', fontsize=14)
    ax.grid(True, alpha=0.3)

    # Підписи мультиплетностей
    for s, e in zip(spins, rel_energies):

```

```

        ax.text(s, e+1, f'M={s+1}', ha='center', fontsize=9)

    plt.tight_layout()
    plt.savefig(f'{symbol}_spin_states_{functional}.pdf')
    plt.show()

    return energies, spins

# Приклад: Карбон (різні спінові стани)
# Основний стан C: 3P (триплет, 2S=2)
# Збуджені: 1D (синглет, 2S=0), 1S (синглет, 2S=0)
energies_c, spins_c = compare_spin_states('C', [0, 2, 4],
                                          functional='pbe0')

# Залізо (різні спінові стани)
# Fe може бути у низькоспіновому, середньо- та високоспіновому станах
energies_fe, spins_fe = compare_spin_states('Fe', [0, 2, 4, 6],
                                          functional='tpss',
                                          basis='def2-svp')

```

#### 4.3.5. Аналіз d-орбіталей перехідних металів.

```

from pyscf import gto, dft, lo
import numpy as np

def analyze_d_orbitals(symbol, spin, functional='pbe',
                      basis='def2-tzvp'):
    """
    Аналіз заселеності d-орбіталей
    """

    mol = gto.M(
        atom=f'{symbol} 0 0 0',
        basis=basis,
        spin=spin,
        verbose=0
    )

    mf = dft.UKS(mol)
    mf.xc = functional
    mf.verbose = 0
    energy = mf.kernel()

    if not mf.converged:
        print(f'Не конвергувало для {symbol}')
        return

    print(f'\nАналіз d-орбіталей {symbol} ({functional.upper()})')
    print('='*70)

    # Заселеності Малікена
    pop, chg = mf.mulliken_pop()

    # Пошук d-орбіталей
    ao_labels = mol.ao_labels(fmt=False)

    d_orbitals_alpha = []
    d_orbitals_beta = []

    for i, label in enumerate(ao_labels):
        atom_id, atom_symbol, shell_type, *rest = label
        if shell_type.startswith('3d'):

```

```

# Альфа заселеність
dm_alpha = mf.make_rdm1()[0]
s = mol.intor('int1e_ovlp')
pop_alpha = (dm_alpha @ s)[i, i]

# Бета заселеність
dm_beta = mf.make_rdm1()[1]
pop_beta = (dm_beta @ s)[i, i]

d_orbitals_alpha.append((shell_type, pop_alpha))
d_orbitals_beta.append((shell_type, pop_beta))

if d_orbitals_alpha:
    print('\nЗаселеності d-орбіталей:')
    print(f'{"Орбіталь":12s} {"α":10s} {"β":10s} {"Сума":10s} '
          f'{"Спін":10s}')
    print('- '*70)

    for (orb_a, pop_a), (orb_b, pop_b) in zip(d_orbitals_alpha,
                                              d_orbitals_beta):
        total = pop_a + pop_b
        spin_dens = pop_a - pop_b
        print(f'{"orb_a":12s} {"pop_a":10.4f} {"pop_b":10.4f} '
              f'{"total":10.4f} {"spin_dens":10.4f}')

# Загальна заселеність d-оболонки
total_d_alpha = sum(p for _, p in d_orbitals_alpha)
total_d_beta = sum(p for _, p in d_orbitals_beta)

print('- '*70)
print(f'{"Разом":12s} {"total_d_alpha":10.4f} '
      f'{"total_d_beta":10.4f} '
      f'{"total_d_alpha+total_d_beta":10.4f} '
      f'{"total_d_alpha-total_d_beta":10.4f}')

# Орбітальні енергії
print(f'\nОрбітальні енергії (eV):')
print(f'{"MO":5s} {"α-енергія":12s} {"β-енергія":12s} '
      f'{"Заповнення":15s}')
print('- '*70)

n_alpha, n_beta = mol.nelec

for i in range(min(10, len(mf.mo_energy[0]))):
    e_alpha = mf.mo_energy[0][i] * 27.211386
    e_beta = mf.mo_energy[1][i] * 27.211386

    occ_a = 'occ' if i < n_alpha else 'virt'
    occ_b = 'occ' if i < n_beta else 'virt'

    print(f'{i+1:5d} {e_alpha:12.4f} {e_beta:12.4f} '
          f'α:{occ_a:5s} β:{occ_b:5s}')

# Приклади
analyze_d_orbitals('Sc', spin=1)
analyze_d_orbitals('Ti', spin=2)
analyze_d_orbitals('V', spin=3)
analyze_d_orbitals('Cr', spin=6)
analyze_d_orbitals('Mn', spin=5)
analyze_d_orbitals('Fe', spin=4)
analyze_d_orbitals('Co', spin=3)
analyze_d_orbitals('Ni', spin=2)
analyze_d_orbitals('Cu', spin=1)
analyze_d_orbitals('Zn', spin=0)

```

**4.3.6. Розрахунок важких атомів.** Для важких атомів (4d, 5d, 4f, 5f) важливі релятивістські ефекти:

```
from pyscf import gto, dft

def heavy_atom_calculation(symbol, spin, functional='pbe',
                           basis='def2-svp', relativistic=False):
    """
    Розрахунок важкого атома з/без релятивістських ефектів
    """

    mol = gto.M(
        atom=f'{symbol} 0 0 0',
        basis=basis,
        spin=spin,
        verbose=0
    )

    mf = dft.UKS(mol)
    mf.xc = functional

    if relativistic:
        # X2C (exact 2-component) релятивістський гамільтоніан
        mf = mf.x2c()
        print(f'\nРозрахунок {symbol} з X2C релятивістськими '
              f'корекціями')
    else:
        print(f'\nРозрахунок {symbol} (нерелятивістський)')

    mf.conv_tol = 1e-9
    mf.max_cycle = 150
    energy = mf.kernel()

    print(f'Енергія: {energy:.8f} Ha')

    return energy

# Порівняння релятивістських ефектів
print('Порівняння релятивістських ефектів')
print('='*70)

# 5d метал: Золото
print('\nЗолото (Au):')
e_aunr = heavy_atom_calculation('Au', spin=1, relativistic=False)
e_aur = heavy_atom_calculation('Au', spin=1, relativistic=True)
rel_effect = (e_aur - e_aunr) * 627.509 # kcal/mol
print(f'Релятивістський ефект: {rel_effect:.2f} kcal/mol')

# 4f метал: Гадоліній
print('\nГадоліній (Gd):')
e_gdnr = heavy_atom_calculation('Gd', spin=8, basis='def2-svp',
                                relativistic=False)
e_gdr = heavy_atom_calculation('Gd', spin=8, basis='def2-svp',
                                relativistic=True)
rel_effect = (e_gdr - e_gdnr) * 627.509
print(f'Релятивістський ефект: {rel_effect:.2f} kcal/mol')
```

**4.3.7. Числові сітки в DFT.** Точність DFT розрахунків залежить від якості числової сітки для інтегрування:

```
from pyscf import gto, dft
```

```

import numpy as np

def test_grid_quality(symbol, spin, functional='pbe',
                     basis='cc-pvtz'):
    """
    Тестування впливу якості сітки на результати
    """

    mol = gto.M(
        atom=f'{symbol} 0 0 0',
        basis=basis,
        spin=spin,
        verbose=0
    )

    # Різні рівні сіток
    grids = [
        (1, 'грубо'),
        (2, 'середньо'),
        (3, 'добре (за замовчуванням)'),
        (4, 'дуже добре'),
        (5, 'ультра добре')
    ]

    print(f'\nВплив якості сітки на енергію {symbol}')
    print(f'Функціонал: {functional.upper()}, базис: {basis}')
    print('='*70)
    print(f'{"Рівень":8s} {"Опис":30s} {"Енергія, Ha":15s} '
          f'{"ΔE, μHa":10s}')
    print('-'*70)

    energies = []

    for level, description in grids:
        if spin == 0:
            mf = dft.RKS(mol)
        else:
            mf = dft.UKS(mol)

        mf.xc = functional
        mf.grids.level = level
        mf.verbose = 0
        mf.conv_tol = 1e-11

        energy = mf.kernel()
        energies.append(energy)

        if len(energies) == 1:
            e_ref = energy
            delta = 0.0
        else:
            delta = (energy - e_ref) * 1e6 # microHartree

        print(f'{level:8d} {description:30s} {energy:15.8f} '
              f'{delta:10.2f}')

    print('='*70)

    # Оцінка збіжності
    if len(energies) >= 3:
        conv = abs(energies[-1] - energies[-2]) * 1e6
        print(f'\nЗбіжність (рівні 4→5): {conv:.4f} μHa')
        if conv < 1.0:
            print('Сітка рівня 4 достатня для точних розрахунків')
        else:

```

## 4. Теорія функціоналу густини (DFT)

---

```
print('Для високої точності використовуйте рівень 5')

# Тестування
test_grid_quality('C', spin=2)
test_grid_quality('Fe', spin=4, basis='def2-svp')
test_grid_quality('Kr', spin=0)
```

---

### 4.3.8. Паралелізація DFT розрахунків.

---

```
from pyscf import gto, dft
import os

# Налаштування кількості потоків
os.environ['OMP_NUM_THREADS'] = '4' # 4 потоки

mol = gto.M(
    atom='Br 0 0 0',
    basis='def2-tzvp',
    spin=1,
    verbose=4
)

mf = dft.UKS(mol)
mf.xc = 'pbe0'

# DFT розрахунки автоматично використовують паралелізацію
# для обчислення інтегралів та Фок-матриць
energy = mf.kernel()

print(f'\nЕнергія Br: {energy:.8f} Ha')
print(f'Використано потоків: {os.environ.get("OMP_NUM_THREADS")}')


```

---

## 4.4. Порівняння HF та DFT результатів

### 4.4.1. Систематичне порівняння енергій.

---

```
from pyscf import gto, scf, dft
import numpy as np
import matplotlib.pyplot as plt

def comprehensive_comparison(atoms_list, basis='cc-pvtz'):
    """
    Детальне порівняння HF та DFT для списку атомів
    """
    methods = {
        'HF': None,
        'LDA': 'lda',
        'PBE': 'pbe',
        'BLYP': 'blyp',
        'B3LYP': 'b3lyp',
        'PBE0': 'pbe0'
    }

    results = {method: [] for method in methods}
    atom_symbols = []

    print(f'\nПорівняння методів (базис: {basis})')
```

---

```

print('='*90)
print(f'{"Атом":6s} {"HF":15s} {"LDA":15s} {"PBE":15s} '
      f'{"BLYP":15s} {"B3LYP":15s} {"PBE0":15s}')
print('='*90)

for symbol, spin in atoms_list:
    mol = gto.M(
        atom=f'{symbol} 0 0 0',
        basis=basis,
        spin=spin,
        verbose=0
    )

    energies_row = []

    # HF
    if spin == 0:
        mf = scf.RHF(mol)
    else:
        mf = scf.UHF(mol)
    mf.verbose = 0
    mf.conv_tol = 1e-10
    e_hf = mf.kernel()
    results['HF'].append(e_hf)
    energies_row.append(e_hf)

    # DFT функціонали
    for method in ['LDA', 'PBE', 'BLYP', 'B3LYP', 'PBE0']:
        if spin == 0:
            mf = dft.RKS(mol)
        else:
            mf = dft.UKS(mol)

        mf.xc = methods[method]
        mf.verbose = 0
        mf.conv_tol = 1e-10

        try:
            energy = mf.kernel()
            results[method].append(energy)
            energies_row.append(energy)
        except:
            results[method].append(np.nan)
            energies_row.append(np.nan)

    atom_symbols.append(symbol)

    # Виведення рядка
    row_str = f'{symbol:6s}'
    for e in energies_row:
        if not np.isnan(e):
            row_str += f' {e:15.8f}'
        else:
            row_str += f' {"N/A":15s}'
    print(row_str)

print('='*90)

# Аналіз різниць
print('\nРізниці відносно HF (mHa):')
print('='*90)
print(f'{"Атом":6s} {"LDA-HF":12s} {"PBE-HF":12s} '
      f'{"BLYP-HF":12s} {"B3LYP-HF":12s} {"PBE0-HF":12s}')
print('='*90)

```

```

for i, symbol in enumerate(atom_symbols):
    row_str = f'{symbol:6s}'
    e_hf = results['HF'][i]

    for method in ['LDA', 'PBE', 'BLYP', 'B3LYP', 'PBE0']:
        e_dft = results[method][i]
        if not np.isnan(e_dft):
            diff = (e_dft - e_hf) * 1000
            row_str += f' {diff:12.4f}'
        else:
            row_str += f' {"N/A":12s}'
    print(row_str)

print('='*90)

# Графік
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

# Абсолютні енергії
x = np.arange(len(atom_symbols))
width = 0.14

for i, method in enumerate(methods.keys()):
    offset = (i - 2.5) * width
    energies = results[method]
    ax1.bar(x + offset, energies, width, label=method)

ax1.set_xlabel('Атом', fontsize=12)
ax1.set_ylabel('Енергія (Ha)', fontsize=12)
ax1.set_title('Абсолютні енергії', fontsize=14)
ax1.set_xticks(x)
ax1.set_xticklabels(atom_symbols)
ax1.legend()
ax1.grid(True, alpha=0.3, axis='y')

# Різниці відносно HF
for i, method in enumerate(['LDA', 'PBE', 'BLYP', 'B3LYP', 'PBE0']):
    offset = (i - 2) * width
    diffs = [(results[method][j] - results['HF'][j]) * 1000
              for j in range(len(atom_symbols))]
    ax2.bar(x + offset, diffs, width, label=method)

ax2.axhline(y=0, color='black', linestyle='--', linewidth=0.8)
ax2.set_xlabel('Атом', fontsize=12)
ax2.set_ylabel('ΔE відносно HF (мHa)', fontsize=12)
ax2.set_title('Різниці енергій', fontsize=14)
ax2.set_xticks(x)
ax2.set_xticklabels(atom_symbols)
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('hf_vs_dft_comparison.pdf')
plt.show()

return results

# Атоми другого періоду
atoms_2nd = [
    ('Li', 1), ('Be', 0), ('B', 1), ('C', 2),
    ('N', 3), ('O', 2), ('F', 1), ('Ne', 0)
]

results = comprehensive_comparison(atoms_2nd, basis='cc-pvtz')

```

## 4.4.2. Порівняння енергій іонізації.

```

from pyscf import gto, scf, dft
import numpy as np
import matplotlib.pyplot as plt

def compare_ionization_energies(atoms_data, basis='aug-cc-pvtz'):
    """
    Порівняння розрахункових та експериментальних ІЕ
    """

    methods = ['HF', 'LDA', 'PBE', 'B3LYP', 'PBE0']

    print(f'\nЕнергії іонізації (eV), базис: {basis}')
    print('='*90)
    print(f'{"Атом":6s} {"Експ.":10s} {"HF":10s} {"LDA":10s} '
          f'{"PBE":10s} {"B3LYP":10s} {"PBE0":10s}')
    print('-'*90)

    results = {method: [] for method in methods}
    experimental = []
    atom_symbols = []

    for symbol, spin_n, spin_c, ie_exp in atoms_data:
        atom_symbols.append(symbol)
        experimental.append(ie_exp)

        # Нейтральний атом
        mol_n = gto.M(
            atom=f'{symbol} 0 0 0',
            basis=basis,
            spin=spin_n,
            verbose=0
        )

        # Катіон
        mol_c = gto.M(
            atom=f'{symbol} 0 0 0',
            basis=basis,
            charge=1,
            spin=spin_c,
            verbose=0
        )

        ie_values = [ie_exp]

        for method in methods:
            # Нейтральний
            if method == 'HF':
                mf_n = scf.UHF(mol_n) if spin_n > 0 else scf.RHF(mol_n)
                mf_c = scf.UHF(mol_c) if spin_c > 0 else scf.RHF(mol_c)
            else:
                mf_n = dft.UKS(mol_n) if spin_n > 0 else dft.RKS(mol_n)
                mf_c = dft.UKS(mol_c) if spin_c > 0 else dft.RKS(mol_c)

            xc_dict = {'LDA': 'lda', 'PBE': 'pbe',
                      'B3LYP': 'b3lyp', 'PBE0': 'pbe0'}
            mf_n.xc = xc_dict[method]
            mf_c.xc = xc_dict[method]

            mf_n.verbose = 0
            mf_c.verbose = 0

            e_n = mf_n.kernel()

```

```

e_c = mf_c.kernel()

ie = (e_c - e_n) * 27.211386 # eV
results[method].append(ie)
ie_values.append(ie)

# Виведення
row_str = f'{symbol:6s}'
for ie in ie_values:
    row_str += f' {ie:10.4f}'
print(row_str)

print('='*90)

# Статистика похибок
print('\nСередні абсолютні похибки (MAE, eV):')
print('-'*50)

for method in methods:
    errors = [abs(results[method][i] - experimental[i])
              for i in range(len(experimental))]
    mae = np.mean(errors)
    print(f'{method:10s}: {mae:8.4f} eV')

# Графік
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

# Порівняння IE
x = np.arange(len(atom_symbols))
width = 0.14

colors = ['black', 'red', 'blue', 'green', 'orange', 'purple']

for i, (method, color) in enumerate(zip(['Експ.'] + methods,
                                         colors)):
    offset = (i - 2.5) * width
    if method == 'Експ.':
        values = experimental
    else:
        values = results[method]

    ax1.bar(x + offset, values, width, label=method, color=color,
            alpha=0.8)

ax1.set_xlabel('Атом', fontsize=12)
ax1.set_ylabel('Енергія іонізації (eV)', fontsize=12)
ax1.set_title('Порівняння енергій іонізації', fontsize=14)
ax1.set_xticks(x)
ax1.set_xticklabels(atom_symbols)
ax1.legend()
ax1.grid(True, alpha=0.3, axis='y')

# Похибки
for method, color in zip(methods, colors[1:]):
    errors = [results[method][i] - experimental[i]
              for i in range(len(experimental))]
    ax2.plot(atom_symbols, errors, 'o-', label=method,
             color=color, linewidth=2, markersize=8)

ax2.axhline(y=0, color='black', linestyle='--', linewidth=1)
ax2.set_xlabel('Атом', fontsize=12)
ax2.set_ylabel('Похибка (eV)', fontsize=12)
ax2.set_title('Похибки відносно експерименту', fontsize=14)
ax2.legend()
ax2.grid(True, alpha=0.3)

```

```

plt.tight_layout()
plt.savefig('ionization_energies_comparison.pdf')
plt.show()

# Дані: (символ, спин нейтрального, спин катіона, ІЕ експ.)
atoms_ie = [
    ('Li', 1, 0, 5.39),
    ('Be', 0, 1, 9.32),
    ('B', 1, 0, 8.30),
    ('C', 2, 1, 11.26),
    ('N', 3, 2, 14.53),
    ('O', 2, 3, 13.62),
    ('F', 1, 2, 17.42),
    ('Ne', 0, 1, 21.56)
]

compare_ionization_energies(atoms_ie)

```

#### 4.4.3. Електронна спорідненість.

```

from pyscf import gto, scf, dft

def electron_affinity_comparison(atoms_data, basis='aug-cc-pvqz'):
    """
    Порівняння електронної спорідненості
    EA = E(A) - E(A-)
    """

    methods = ['HF', 'LDA', 'PBE', 'B3LYP', 'PBE0']

    print(f'\nЕлектронна спорідненість (eV), базис: {basis}')
    print('='*90)
    print(f'{"Атом":6s} {"Експ.":10s} {"HF":10s} {"LDA":10s} '
          f'{"PBE":10s} {"B3LYP":10s} {"PBE0":10s}')
    print('-'*90)

    for symbol, spin_n, spin_a, ea_exp in atoms_data:
        # Нейтральний атом
        mol_n = gto.M(
            atom=f'{symbol} 0 0 0',
            basis=basis,
            spin=spin_n,
            verbose=0
        )

        # Аніон
        mol_a = gto.M(
            atom=f'{symbol} 0 0 0',
            basis=basis,
            charge=-1,
            spin=spin_a,
            verbose=0
        )

        ea_values = [ea_exp]

        for method in methods:
            if method == 'HF':
                mf_n = scf.UHF(mol_n) if spin_n > 0 else scf.RHF(mol_n)
                mf_a = scf.UHF(mol_a) if spin_a > 0 else scf.RHF(mol_a)
            else:

```

```
mf_n = dft.UKS(mol_n) if spin_n > 0 else dft.RKS(mol_n)
mf_a = dft.UKS(mol_a) if spin_a > 0 else dft.RKS(mol_a)

xc_dict = {'LDA': 'lda', 'PBE': 'pbe',
           'B3LYP': 'b3lyp', 'PBE0': 'pbe0'}
mf_n.xc = xc_dict[method]
mf_a.xc = xc_dict[method]

mf_n.verbose = 0
mf_a.verbose = 0
mf_a.level_shift = 0.5 # Для аніонів часто потрібно

try:
    e_n = mf_n.kernel()
    e_a = mf_a.kernel()

    ea = (e_n - e_a) * 27.211386 # eV
    ea_values.append(ea)
except:
    ea_values.append(np.nan)

# Виведення
row_str = f'{symbol:6s}'
for ea in ea_values:
    if not np.isnan(ea):
        row_str += f' {ea:10.4f}'
    else:
        row_str += f' {"N/A":10s}'
print(row_str)

print('='*90)
print('\nПримітка: EA > 0 означає, що аніон стабільний')

# Дані: (символ, спин нейтрального, спин аніона, EA експ.)
atoms_ea = [
    ('B', 1, 2, 0.28),
    ('C', 2, 3, 1.26),
    ('O', 2, 1, 1.46),
    ('F', 1, 0, 3.40),
    ('Cl', 1, 0, 3.61),
    ('Br', 1, 0, 3.36)
]

electron_affinity_comparison(atoms_ea)
```

#### 4.4.4. Порівняння орбітальних енергій.

```
from pyscf import gto, scf, dft
import matplotlib.pyplot as plt

def compare_orbital_energies(symbol, spin, basis='cc-pvtz'):
    """
    Порівняння орбітальних енергій HF vs DFT
    """

    mol = gto.M(
        atom=f'{symbol} 0 0 0',
        basis=basis,
        spin=spin,
        verbose=0
    )
```

```

methods = {
    'HF': None,
    'LDA': 'lda',
    'PBE': 'pbe',
    'B3LYP': 'b3lyp',
    'PBE0': 'pbe0'
}

orbital_energies = {}

for method, xc in methods.items():
    if method == 'HF':
        mf = scf.UHF(mol) if spin > 0 else scf.RHF(mol)
    else:
        mf = dft.UKS(mol) if spin > 0 else dft.RKS(mol)
        mf.xc = xc

    mf.verbose = 0
    mf.kernel()

    if spin == 0:
        orbital_energies[method] = mf.mo_energy * 27.211386
    else:
        orbital_energies[method] = mf.mo_energy[0] * 27.211386

# Графік
fig, ax = plt.subplots(figsize=(12, 8))

n_orb = min(10, len(orbital_energies['HF']))
x = np.arange(n_orb)
width = 0.16

colors = ['red', 'blue', 'green', 'orange', 'purple']

for i, (method, color) in enumerate(zip(methods.keys(), colors)):
    offset = (i - 2) * width
    energies = orbital_energies[method][:n_orb]
    ax.bar(x + offset, energies, width, label=method,
           color=color, alpha=0.8)

# Лінія HOMO
if spin == 0:
    n_occ = mol.nelectron // 2
else:
    n_occ = mol.nelec[0]

ax.axvline(x=n_occ-0.5, color='black', linestyle='--',
           linewidth=2, label='HOMO/LUMO')

ax.set_xlabel('Орбіталь', fontsize=12)
ax.set_ylabel('Енергія (eV)', fontsize=12)
ax.set_title(f'Орбітальні енергії {symbol}', fontsize=14)
ax.set_xticks(x)
ax.set_xticklabels([f'MO{i+1}' for i in x])
ax.legend()
ax.grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.savefig(f'{symbol}_orbital_energies_comparison.pdf')
plt.show()

# Таблиця
print(f'\nОрбітальні енергії {symbol} (eV)')
print('='*80)
print(f'{"MO":5s} {"HF":12s} {"LDA":12s} {"PBE":12s} '

```

```

        f'{"B3LYP":12s} {"PBE0":12s}')
print('-'*80)

for i in range(n_orb):
    row_str = f'{i+1:5d}'
    for method in methods.keys():
        e = orbital_energies[method][i]
        row_str += f' {e:12.4f}'

    if i == n_occ - 1:
        row_str += ' ← HOMO'
    elif i == n_occ:
        row_str += ' ← LUMO'

    print(row_str)

print('='*80)

# Приклади
compare_orbital_energies('C', spin=2)
compare_orbital_energies('Ne', spin=0)
compare_orbital_energies('O', spin=2)

```

#### 4.4.5. Забруднення спіном: HF vs DFT.

```

from pyscf import gto, scf, dft

def spin_contamination_analysis(atoms_list, basis='cc-pvtz'):
    """
    Порівняння забруднення спіном у HF та DFT
    """

    print(f'\nЗабруднення спіном <S²> (базис: {basis})')
    print('='*80)
    print(f'{"Атом":6s} {"2S":4s} {"<S²> очік.":12s} {"HF":12s} '
          f'{"LDA":12s} {"PBE":12s} {"B3LYP":12s}')
    print('-'*80)

    for symbol, spin in atoms_list:
        if spin == 0:
            continue # Тільки відкриті системи

        mol = gto.M(
            atom=f'{symbol} 0 0 0',
            basis=basis,
            spin=spin,
            verbose=0
        )

        expected_s2 = spin * (spin + 2) / 4

        s2_values = [expected_s2]

        # HF
        mf_hf = scf.UHF(mol)
        mf_hf.verbose = 0
        mf_hf.kernel()
        s2_hf = mf_hf.spin_square()[0]
        s2_values.append(s2_hf)

        # DFT
        for xc in ['lda', 'pbe', 'b3lyp']:

```

```

mf = dft.UKS(mol)
mf.xc = xc
mf.verbose = 0
mf.kernel()
s2 = mf.spin_square()[0]
s2_values.append(s2)

# Виведення
row_str = f'{symbol:6s} {spin:4d}'
for s2 in s2_values:
    row_str += f' {s2:12.6f}'

# Забруднення
contamination = s2_hf - expected_s2
if abs(contamination) > 0.01:
    row_str += ' ← помітне забруднення HF'

print(row_str)

print('='*80)
print('\nПримітка: Чисті DFT (LDA, PBE) не мають забруднення')
print('          Гібридні DFT (B3LYP) мають слабе забруднення')

# Атоми з відкритими оболонками
atoms_open = [
    ('H', 1), ('Li', 1), ('B', 1), ('C', 2),
    ('N', 3), ('O', 2), ('F', 1), ('Na', 1)
]

spin_contamination_analysis(atoms_open)

```

#### 4.4.6. Час обчислень: HF vs DFT.

```

from pyscf import gto, scf, dft
import time

def timing_comparison(symbol, spin, basis='def2-tzvp'):
    """
    Порівняння часу виконання HF vs DFT
    """

    mol = gto.M(
        atom=f'{symbol} 0 0 0',
        basis=basis,
        spin=spin,
        verbose=0
    )

    methods = {
        'HF': None,
        'LDA': 'lda',
        'PBE': 'pbe',
        'B3LYP': 'b3lyp',
        'PBE0': 'pbe0'
    }

    print(f'\nЧас виконання для {symbol} (базис: {basis})')
    print('='*60)
    print(f'{"Метод":10s} {"Час, с":10s} {"Відносно HF":15s}')
    print('-'*60)

    times = {}

```

```

for method, xc in methods.items():
    if method == 'HF':
        mf = scf.UHF(mol) if spin > 0 else scf.RHF(mol)
    else:
        mf = dft.UKS(mol) if spin > 0 else dft.RKS(mol)
        mf.xc = xc

mf.verbose = 0

# Вимірювання часу
start = time.time()
mf.kernel()
elapsed = time.time() - start

times[method] = elapsed

if method == 'HF':
    t_ref = elapsed
    rel = 1.0
else:
    rel = elapsed / t_ref

print(f'{method:10s} {elapsed:10.4f} {rel:15.3f}x')

print('='*60)

# Тестування
timing_comparison('C', spin=2, basis='cc-pvtz')
timing_comparison('Fe', spin=4, basis='def2-svp')
timing_comparison('Kr', spin=0, basis='def2-tzvp')

```

Таблиця 4.2. Підсумкове порівняння HF та DFT методів

Критерій	Hartree-Fock	DFT
Точність енергій	Добра якісно	Краща кількісно
Енергії іонізації	Систематично завищені	Ближче до експерименту
Електронна спорідненість	Погана для аніонів	Значно краща
Орбітальні енергії	НОМО $\approx$ -ІЕ (теорема Купманса)	Відхилення від теореми
Забруднення спіном	Присутнє (UHF)	Відсутнє (чисті DFT)
Швидкість	Середня	Швидше (чисті DFT) Повільніше (гібриди)
Перехідні метали	Складно конвергує	Зазвичай краще
Дисперсійні взаємодії	Відсутні	Потрібні корекції
Систематичність	Добра	Залежить від функціоналу

#### 4.4.7. Загальні висновки HF vs DFT. Рекомендації:

- Використовуйте HF для швидких якісних оцінок та як початок для post-HF методів

- Використовуйте чисті DFT (PBE) для великих систем, перехідних металів
- Використовуйте гібридні DFT (B3LYP, PBE0) для найкращого балансу точності та швидкості
- Для аніонів обов'язково використовуйте дифузні функції (aug-базиси)
- Для важких атомів враховуйте релятивістські ефекти

## 4.5. Вибір функціоналу для атомних систем

**4.5.1. Тестовий набір даних.** Для оцінки якості функціоналів використовуємо стандартні атомні властивості:

```
from pyscf import gto, scf, dft
import numpy as np

# Експериментальні дані (eV)
experimental_data = {
    'Li': {'IE1': 5.39, 'EA': 0.62},
    'C': {'IE1': 11.26, 'EA': 1.26},
    'N': {'IE1': 14.53, 'EA': -0.07},
    'O': {'IE1': 13.62, 'EA': 1.46},
    'F': {'IE1': 17.42, 'EA': 3.40},
    'Ne': {'IE1': 21.56, 'EA': None},
}

def benchmark_functional(functional, basis='aug-cc-pvtz'):
    """
    Тестування функціоналу на наборі атомів
    """

    print(f'\nТестування функціоналу {functional.upper()}')
    print('='*80)

    mae_ie = []
    mae_ea = []

    for symbol, data in experimental_data.items():
        # Визначення спінів (спрощено)
        spins = {
            'Li': (1, 0), 'C': (2, 1), 'N': (3, 2),
            'O': (2, 3), 'F': (1, 2), 'Ne': (0, 1)
        }
        spin_n, spin_c = spins[symbol]

        # Нейтральний атом
        mol_n = gto.M(atom=f'{symbol} 0 0 0', basis=basis,
                      spin=spin_n, verbose=0)

        if functional.lower() == 'hf':
            mf_n = scf.UHF(mol_n) if spin_n > 0 else scf.RHF(mol_n)
        else:
            mf_n = dft.UKS(mol_n) if spin_n > 0 else dft.RKS(mol_n)
            mf_n.xc = functional

        mf_n.verbose = 0
        e_n = mf_n.kernel()

        # Катіон (IE)
        mol_c = gto.M(atom=f'{symbol} 0 0 0', basis=basis,
```

```

        charge=1, spin=spin_c, verbose=0)

    if functional.lower() == 'hf':
        mf_c = scf.UHF(mol_c) if spin_c > 0 else scf.RHF(mol_c)
    else:
        mf_c = dft.UKS(mol_c) if spin_c > 0 else dft.RKS(mol_c)
        mf_c.xc = functional

    mf_c.verbose = 0
    e_c = mf_c.kernel()

    ie_calc = (e_c - e_n) * 27.211386
    ie_exp = data['IE1']
    error_ie = abs(ie_calc - ie_exp)
    mae_ie.append(error_ie)

    print(f'{symbol:4s} IE: calc={ie_calc:7.3f} eV, '
          f'exp={ie_exp:7.3f} eV, error={error_ie:6.3f} eV')

    # Аніон (EA) - якщо є дані
    if data['EA'] is not None and data['EA'] > 0:
        spin_a = spin_n + 1 # Спрошене припущення

        mol_a = gto.M(atom=f'{symbol} 0 0 0', basis=basis,
                      charge=-1, spin=spin_a, verbose=0)

        if functional.lower() == 'hf':
            mf_a = scf.UHF(mol_a) if spin_a > 0 else scf.RHF(mol_a)
        else:
            mf_a = dft.UKS(mol_a) if spin_a > 0 else dft.RKS(mol_a)
            mf_a.xc = functional

        mf_a.verbose = 0
        mf_a.level_shift = 0.5

        try:
            e_a = mf_a.kernel()
            ea_calc = (e_n - e_a) * 27.211386
            ea_exp = data['EA']
            error_ea = abs(ea_calc - ea_exp)
            mae_ea.append(error_ea)

            print(f'      EA: calc={ea_calc:7.3f} eV, '
                  f'exp={ea_exp:7.3f} eV, error={error_ea:6.3f} eV')
        except:
            print(f'      EA: не конвергувало')

    print('='*80)
    print(f'MAE (IE): {np.mean(mae_ie):.3f} eV')
    if mae_ea:
        print(f'MAE (EA): {np.mean(mae_ea):.3f} eV')

    return np.mean(mae_ie), np.mean(mae_ea) if mae_ea else None

# Тестування різних функціоналів
functionals_to_test = ['HF', 'LDA', 'PBE', 'BLYP', 'B3LYP', 'PBE0']

results = {}
for func in functionals_to_test:
    mae_ie, mae_ea = benchmark_functional(func)
    results[func] = {'IE': mae_ie, 'EA': mae_ea}

# Підсумкова таблиця
print('\n\nПідсумкові MAE (eV):')
print('='*50)

```

```
print(f'{"Функціонал":12s} {"IE":10s} {"EA":10s}')
print('-'*50)
for func, res in results.items():
    ea_str = f"{res['EA']:.3f}" if res['EA'] else "N/A"
    print(f'{"func":12s} {"res["IE"]":10.3f} {"ea_str":10s}')
print('-'*50)
```

Таблиця 4.3. Рекомендовані функціонали для різних груп елементів

Група елементів	Рекомендований	Альтернатива
H, He	HF, PBE0	Будь-який
Li–Ne (2 період)	PBE0, B3LYP	PBE, $\omega$ B97X-D
Na–Ar (3 період)	PBE0, B3LYP	TPSSH
3d метали (Sc–Zn)	TPSSH, M06	PBE, B3LYP
4d метали (Y–Cd)	TPSSH, PBE	M06
5d метали (La–Hg)	PBE, TPSSH	+ релятивістські
Лантаноїди	PBE, SCAN	+ SOC
Актиноїди	PBE	+ SOC + DFT+U

#### 4.5.2. Рекомендації для різних елементів.

### 4.6. Практичні завдання

#### 4.6.1. Завдання 1: Систематичне дослідження.

ЗАВДАННЯ 1: Розрахуйте енергії всіх атомів третього періоду (Na–Ar) з функціоналами PBE, B3LYP та PBE0. Порівняйте результати з HF. Побудуйте графіки.

Базис: def2-TZVP

# Ваш код тут

#### 4.6.2. Завдання 2: Функціональна залежність.

ЗАВДАННЯ 2: Для атома Заліза (Fe) порівняйте енергії різних спінових станів ( $2S = 0, 2, 4, 6$ ) використовуючи функціонали: LDA, PBE, TPSS, B3LYP, PBE0.

Який спіновий стан є найнижчим для кожного функціоналу?  
Який функціонал дає правильний основний стан?

Базис: def2-SVP

# Ваш код тут

## 4. Теорія функціоналу густини (DFT)

### 4.6.3. Завдання 3: Конвергенція до базисної межі.

```
""
ЗАВДАННЯ 3: Для атома Неону розрахуйте енергію з функціоналом
PBE0 та базисами cc-pVDZ, cc-pVTZ, cc-pVQZ, cc-pV5Z.
```

```
Екстраполуйте енергію до базисної межі (CBS) за формулою:
 $E(X) = E_{CBS} + A/X^3$ 
```

```
де X = 2, 3, 4, 5 для DZ, TZ, QZ, 5Z.
```

```
Порівняйте з експериментальною енергією Ne: -128.547 eV
```

```
""
```

```
# Ваш код тут
```

### 4.6.4. Завдання 4: Перехідні метали.

```
""
ЗАВДАННЯ 4: Розрахуйте всі 3d перехідні метали (Sc--Zn)
з функціоналом TPSSh. Проаналізуйте заселеності d-орбіталей.
```

```
Для яких атомів:
```

- 1) Всі d-орбіталі рівномірно заселені?
- 2) Є виражена асиметрія  $\alpha/\beta$ ?
- 3) Найбільше забруднення спіном?

```
Базис: def2-TZVP
```

```
""
```

```
# Ваш код тут
```

## 4.7. Резюме

У цьому розділі ми детально вивчили теорію функціоналу густини та її застосування до атомних систем:

- **Теоретичні основи** — теореми Хюенберга–Кона, рівняння Кона–Шема
- **Функціонали** — від простих LDA до складних гібридних і meta-GGA
- **Практичні розрахунки** — атоми різних періодів, перехідні метали
- **Порівняння з HF** — енергії, орбіталі, спин, швидкість
- **Вибір методу** — рекомендації для різних задач

### 4.7.1. Ключові висновки.

1. DFT зазвичай дає кращі результати для атомних енергій порівняно з HF
2. Гібридні функціонали (B3LYP, PBE0) — золота середина між точністю та швидкістю
3. Для перехідних металів краще використовувати meta-GGA (TPSS) або спеціалізовані функціонали (M06)
4. Вибір базису критичний: для аніонів потрібні дифузні функції

5. Чисті DFT не мають забруднення спіном на відміну від UHF
6. Якість числової сітки важлива для точних розрахунків
7. Для важких атомів необхідні релятивістські корекції

#### 4.7.2. Типові помилки.

1. **Неправильний спін** — завжди перевіряйте основний стан атома
2. **Недостатній базис** — для точних енергій використовуйте triple-zeta або більше
3. **Забування дифузних функцій** — критично для аніонів та збуджених станів
4. **Ігнорування симетрії** — може уповільнити розрахунок
5. **Погана конвергенція** — використовуйте level shift, змініть початкове наближення
6. **Неправильна сітка** — для точних результатів використовуйте grids.level  $\geq 3$ .

#### 4.7.3. Корисні посилання.

- **Libxc** — бібліотека DFT функціоналів: <https://www.tddft.org/programs/libxc/>
- **NIST** — експериментальні дані атомів: <https://physics.nist.gov/PhysRefData/>
- **Basis Set Exchange** — база даних базисних наборів: <https://www.basissetexchange.org/>

У наступному розділі ми розглянемо Post-Hartree-Fock методи (MP2, CCSD, CASSCF), які дозволяють досягти ще вищої точності для атомних систем, враховуючи електронну кореляцію явно.

# 5

## Пост-Гартрі-Фоківські методи

---

### 5.1. Вступ до електронної кореляції

**5.1.1. Що таке електронна кореляція?** Електронна кореляція — це різниця між точною енергією системи та енергією, отриманою методом Хартрі-Фока:

$$E_{\text{corr}} = E_{\text{exact}} - E_{\text{HF}} \quad (5.1)$$

Метод Хартрі-Фока не враховує миттєву кореляцію рухів електронів, оскільки кожен електрон рухається в усередненому полі всіх інших електронів. Насправді ж електрони "уникають" один одного через кулонівське відштовхування.

### 5.1.2. Типи електронної кореляції.

**Динамічна кореляція** Пов'язана з миттєвими флуктуаціями електронної густини. Проявляється на коротких відстанях між електронами. Може бути враховані методами:

- Теорія збурень Møller-Plesset (MP2, MP3, MP4)
- Coupled Cluster (CCSD, CCSD(T))
- Configuration Interaction (CISD, QCISD)

**Статична (нединамічна) кореляція** Виникає, коли кілька конфігурацій близькі за енергією. Важлива для:

- Розриву хімічних зв'язків
  - Збуджених станів
  - Діелектронних систем
  - Перехідних металів з близькими d-орбіталями
- Методи: CASSCF, CASPT2, MRCI.

**5.1.3. Кореляційна енергія атомів.** Для атомів кореляційна енергія становить 1–5% від повної енергії, але вона критична для точних розрахунків:

Таблиця 5.1. Кореляційна енергія атомів (На)

Атом	$E_{HF}$	$E_{exact}$	$E_{corr}$	% від $E_{HF}$
He	-2.8617	-2.9037	-0.0420	1.5%
Be	-14.573	-14.667	-0.094	0.6%
Ne	-128.547	-128.937	-0.390	0.3%
Ar	-526.817	-527.540	-0.723	0.1%

**5.1.4. Ієрархія методів.** Post-HF методи утворюють ієрархію за точністю та обчислювальною складністю:

1. **HF** — базовий рівень, без кореляції
2. **MP2** —  $\mathcal{O}(N^5)$  — найпростіша кореляція
3. **MP3, MP4** —  $\mathcal{O}(N^6), \mathcal{O}(N^7)$  — вищі порядки теорії збурень
4. **CCSD** —  $\mathcal{O}(N^6)$  — надійна динамічна кореляція
5. **CCSD(T)** —  $\mathcal{O}(N^7)$  — "золотий стандарт" квантової хімії
6. **Full CI** —  $\mathcal{O}(e^N)$  — точний розв'язок (у межах базису)

```

from pyscf import gto, scf, mp, cc, ci
import numpy as np

def correlation_energy_demo(symbol, spin, basis='cc-pvdz'):
    """
    Демонстрація кореляційної енергії
    """

    mol = gto.M(
        atom=f'{symbol} 0 0 0',
        basis=basis,
        spin=spin,
        verbose=0
    )

    print(f'\nКореляційна енергія атома {symbol} (базис: {basis})')
    print('='*70)

    # HF розрахунок
    if spin == 0:
        mf = scf.RHF(mol)
    else:
        mf = scf.UHF(mol)

    mf.verbose = 0
    mf.conv_tol = 1e-10
    e_hf = mf.kernel()

    print(f'HF енергія:           {e_hf:.8f} Ha')

    # MP2
    if spin == 0:
        mp2 = mp.MP2(mf)
    else:
        mp2 = mp.UMP2(mf)

    mp2.verbose = 0
    e_mp2, t2 = mp2.kernel()
    e_total_mp2 = e_hf + e_mp2

```

```

print(f'MP2 кореляція:      {e_mp2:.8f} Ha')
print(f'MP2 повна енергія:  {e_total_mp2:.8f} Ha')

# CCSD (якщо атом не дуже великий)
if mol.nelectron <= 10:
    if spin == 0:
        mycc = cc.CCSD(mf)
    else:
        mycc = cc.UCCSD(mf)

    mycc.verbose = 0
    e_ccsd, t1, t2 = mycc.kernel()
    e_total_ccsd = e_hf + e_ccsd

    print(f'CCSD кореляція:      {e_ccsd:.8f} Ha')
    print(f'CCSD повна енергія:  {e_total_ccsd:.8f} Ha')

# CCSD(T)
e_t = mycc.ccsd_t()
e_total_ccsdt = e_total_ccsd + e_t

print(f'(T) корекція:          {e_t:.8f} Ha')
print(f'CCSD(T) повна енергія:{e_total_ccsdt:.8f} Ha')
else:
    print('CCSD пропущено (атом занадто великий для демо)')

print('='*70)

# Аналіз
print(f'\nАналіз:')
print(f'Кореляція складає {abs(e_mp2/e_hf)*100:.2f}% від HF енергії')

if mol.nelectron <= 10:
    print(f'MP2 відновлює {abs(e_mp2/e_ccsd)*100:.1f}% '
          f'f'CCSD кореляції')

# Приклади
correlation_energy_demo('He', spin=0)
correlation_energy_demo('Be', spin=0)
correlation_energy_demo('C', spin=2)
correlation_energy_demo('Ne', spin=0)

```

### 5.1.5. Важливість кореляції для різних властивостей.

Рис. 5.1. Вплив електронної кореляції на різні властивості

Властивість	Важливість кореляції	Рекомендований метод
Абсолютні енергії	Помірна	MP2, DFT
Енергії іонізації	Висока	CCSD(T)
Електронна спорідненість	Дуже висока	CCSD(T) + дифузні
Енергії збудження	Висока	EOM-CCSD, TD-DFT
Відстані між станами	Висока	CASPT2, MRCI
Дисоціація	Критична	CASSCF, MRCI
Слабкі взаємодії	Критична	CCSD(T), MP2-F12

**5.1.6. Порівняння методів для He.** Розглянемо найпростішу багатоелектронну систему — атом Гелію:

```
from pyscf import gto, scf, mp, cc, fci
import numpy as np
import matplotlib.pyplot as plt

def helium_correlation_study():
    """
    Детальне дослідження кореляції у атомі He
    """

    # Різні базисні набори
    basis_sets = ['cc-pvdz', 'cc-pvtz', 'cc-pvqz', 'cc-pv5z']

    results = {
        'HF': [],
        'MP2': [],
        'CCSD': [],
        'CCSD(T)': [],
        'FCI': []
    }

    print('Збіжність до базисної межі для He')
    print('='*80)
    print(f'{"Базис":12s} {"HF":15s} {"MP2":15s} {"CCSD":15s} '
          f'{"CCSD(T)":15s} {"FCI":15s}')
    print('-'*80)

    for basis in basis_sets:
        mol = gto.M(atom='He 0 0 0', basis=basis, verbose=0)

        # HF
        mf = scf.RHF(mol)
        mf.verbose = 0
        mf.conv_tol = 1e-12
        e_hf = mf.kernel()
        results['HF'].append(e_hf)

        # MP2
        mymp2 = mp.MP2(mf)
        mymp2.verbose = 0
        e_mp2_corr, _ = mymp2.kernel()
        e_mp2 = e_hf + e_mp2_corr
        results['MP2'].append(e_mp2)

        # CCSD
        mycc = cc.CCSD(mf)
        mycc.verbose = 0
        e_ccsd_corr, _, _ = mycc.kernel()
        e_ccsd = e_hf + e_ccsd_corr
        results['CCSD'].append(e_ccsd)

        # CCSD(T)
        e_t = mycc.ccsd_t()
        e_ccsdt = e_ccsd + e_t
        results['CCSD(T)'].append(e_ccsdt)

        # FCI (точний у даному базисі)
        myfci = fci.FCI(mf)
        e_fci = myfci.kernel()[0]
        results['FCI'].append(e_fci)
```

```

print(f'{basis:12s} {e_hf:15.10f} {e_mp2:15.10f} '
      f'{e_ccsd:15.10f} {e_ccsdT:15.10f} {e_fci:15.10f}')

print('='*80)

# Експериментальне значення
e_exp = -2.90372 # Ha
print(f'\nЕкспериментальна енергія He: {e_exp:.10f} Ha')

# Похибки для найбільшого базису
print(f'\nПохибки (cc-pv5z):')
for method in results.keys():
    error = (results[method][-1] - e_exp) * 1000 # mHa
    print(f'{method:10s}: {error:8.4f} mHa')

# Графік збіжності
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Абсолютні енергії
x = np.arange(len(basis_sets))
for method, energies in results.items():
    ax1.plot(x, energies, 'o-', label=method, linewidth=2,
             markersize=8)

ax1.axhline(y=e_exp, color='red', linestyle='--', linewidth=2,
            label='Експеримент')
ax1.set_xticks(x)
ax1.set_xticklabels(basis_sets, rotation=45)
ax1.set_xlabel('Базисний набір', fontsize=12)
ax1.set_ylabel('Енергія (Ha)', fontsize=12)
ax1.set_title('Збіжність енергії He', fontsize=14)
ax1.legend()
ax1.grid(True, alpha=0.3)

# Кореляційна енергія
for method in ['MP2', 'CCSD', 'CCSD(T)', 'FCI']:
    corr_energies = [results[method][i] - results['HF'][i]
                     for i in range(len(basis_sets))]
    ax2.plot(x, corr_energies, 'o-', label=method, linewidth=2,
             markersize=8)

ax2.set_xticks(x)
ax2.set_xticklabels(basis_sets, rotation=45)
ax2.set_xlabel('Базисний набір', fontsize=12)
ax2.set_ylabel('Кореляційна енергія (Ha)', fontsize=12)
ax2.set_title('Кореляційна енергія He', fontsize=14)
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('he_correlation_convergence.pdf')
plt.show()

helium_correlation_study()

```

### 5.1.7. Концепція одно- та багатоконфігураційних методів.

**Одноконфігураційні методи** Базуються на одному детермінанті Слейтера (HF) і додають кореляцію через збудження:

- MP2, MP3, MP4 — теорія збурень
- CCSD, CCSD(T) — coupled cluster

- CISD, QCISD — configuration interaction

Добрі для: основного стану з одним домінуючим детермінантом.

**Багатоконфігураційні методи** Використовують лінійну комбінацію кількох детермінантів:

- CASSCF — вибір активного простору
- CASPT2 — додавання динамічної кореляції до CASSCF
- MRCI — multireference CI

Необхідні для: розриву зв'язків, збуджених станів, діелектронних систем.

```
from pyscf import gto, scf, mcscf

def multiconfigurational_demo():
    """
    Демонстрація багатоконфігураційного характеру
    """

    # Атом Be: [He] 2s2
    # Близькі за енергією 2s2 та 2p2

    mol = gto.M(
        atom='Be 0 0 0',
        basis='cc-pvtz',
        spin=0,
        verbose=0
    )

    print('Багатоконфігураційний характер Be')
    print('='*60)

    # RHF
    mf = scf.RHF(mol)
    mf.verbose = 0
    e_hf = mf.kernel()

    print(f'RHF енергія: {e_hf:.8f} Ha')

    # CASSCF(4,8): 4 електрони у 8 орбіталях (2s, 2p, 3s, 3p)
    mc = mcscf.CASSCF(mf, 8, 4)
    mc.verbose = 0
    e_casscf = mc.kernel()[0]

    print(f'CASSCF(4,8) енергія: {e_casscf:.8f} Ha')
    print(f'Статична кореляція: {(e_casscf - e_hf)*1000:.4f} mHa')

    # Аналіз конфігурацій
    print(f'\nОсновні конфігурації (вага > 1%):')

    # Отримання CI коефіцієнтів
    ci_coeff = mc.ci

    # Для детального аналізу потрібен додатковий код
    print('(детальний аналіз потребує додаткової обробки)')

    print('='*60)

multiconfigurational_demo()
```

**5.1.8. Коли потрібні post-HF методи?**

1. Спектроскопічна точність — похибка  $< 1$  kcal/mol (0.04 eV)
2. Порівняння близьких станів — різниці енергій між мультиплетами
3. Електронна спорідненість — HF і DFT часто не достатньо
4. Еталонні розрахунки — для валідації DFT функціоналів
5. Системи з сильною кореляцією — перехідні метали, f-елементи

**5.2. Теорія збурень Møller-Plesset (MP2)**

**5.2.1. Теоретичні основи MP2.** Теорія збурень Møller-Plesset розділяє гамільтоніан на незбурену частину (HF) та збурення:

$$\hat{H} = \hat{H}_0 + \lambda \hat{V} \quad (5.2)$$

де  $\hat{H}_0$  — оператор Фока, а  $\hat{V}$  — різниця між точним гамільтоніаном та HF.

Енергія розкладається в ряд за степенями  $\lambda$ :

$$E = E^{(0)} + E^{(1)} + E^{(2)} + E^{(3)} + \dots \quad (5.3)$$

де:

- $E^{(0)} + E^{(1)} = E_{HF}$  — енергія Хартрі-Фока
- $E^{(2)}$  — MP2 корекція (перший порядок кореляції)
- $E^{(3)}, E^{(4)}$  — вищі порядки (MP3, MP4)

**5.2.2. Формула MP2 кореляційної енергії.** Для замкненої оболонки (RMP2):

$$E^{(2)} = - \sum_{i < j}^{\text{occ}} \sum_{a < b}^{\text{virt}} \frac{|\langle ij || ab \rangle|^2}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b} \quad (5.4)$$

де:

- $i, j$  — заповнені орбіталі
- $a, b$  — віртуальні орбіталі
- $\langle ij || ab \rangle$  — антисиметризовані двоелектронні інтеграли
- $\epsilon$  — орбітальні енергії

**5.2.3. MP2 розрахунок атома Неону.**

```
from pyscf import gto, scf, mp
import numpy as np

def mp2_neon_calculation(basis='cc-pvtz'):
    """
    Детальний MP2 розрахунок атома Ne
    """
```

```

mol = gto.M(
    atom='Ne 0 0 0',
    basis=basis,
    spin=0,
    verbose=0
)

print(f'\nMP2 розрахунок атома Неону (базис: {basis})')
print('='*70)

# Крок 1: HF розрахунок
print('\nКрок 1: Hartree-Fock розрахунок')
mf = scf.RHF(mol)
mf.verbose = 4
mf.conv_tol = 1e-12
e_hf = mf.kernel()

print(f'\nHF енергія: {e_hf:.10f} Ha')
print(f'HF енергія: {e_hf * 27.211386:.6f} eV')

# Крок 2: MP2 розрахунок
print('\nКрок 2: MP2 кореляція')
mump2 = mp.MP2(mf)
mump2.verbose = 4
e_mp2_corr, t2 = mump2.kernel()

e_total_mp2 = e_hf + e_mp2_corr

print(f'\nMP2 кореляційна енергія: {e_mp2_corr:.10f} Ha')
print(f'MP2 повна енергія: {e_total_mp2:.10f} Ha')
print(f'MP2 повна енергія: {e_total_mp2 * 27.211386:.6f} eV')

# Аналіз
print('\nАналіз:')
print(f'Кореляція становить {abs(e_mp2_corr/e_hf)*100:.3f}% '
      f'від HF енергії')

# Порівняння з експериментом
e_exp = -128.937 # Ha (експериментальна енергія Ne)
error_hf = (e_hf - e_exp) * 1000
error_mp2 = (e_total_mp2 - e_exp) * 1000

print(f'\nПорівняння з експериментом ({e_exp:.6f} Ha):')
print(f'  HF похибка: {error_hf:8.4f} мHa')
print(f'  MP2 похибка: {error_mp2:8.4f} мHa')
print(f'  Покращення: {abs(error_hf - error_mp2):8.4f} мHa')
print(f'  MP2 відновлює {(1 - error_mp2/error_hf)*100:.1f}% '
      f'похибки HF')

# Інформація про розміри
print(f'\nОбчислювальні деталі:')
print(f'  Заповнених орбіталей: {mol.nelectron // 2}')
print(f'  Віртуальних орбіталей: {mol.nao_nr() - mol.nelectron // 2}')
print(f'  Розмір t2 амплітуд: {t2.shape}')

return e_hf, e_total_mp2, e_mp2_corr

e_hf, e_mp2, e_corr = mp2_neon_calculation()

```

**5.2.4. MP2 для відкритих оболонок (UMP2).** Для систем з неспареними електронами використовується UMP2:

```
from pyscf import gto, scf, mp
```

```
def ump2_calculation(symbol, spin, basis='cc-pvtz'):
    """
    UMP2 розрахунок для відкритої оболонки
    """

    mol = gto.M(
        atom=f'{symbol} 0 0 0',
        basis=basis,
        spin=spin,
        verbose=0
    )

    print(f'\nUMP2 розрахунок {symbol} (2S={spin})')
    print('='*70)

    # UHF
    mf = scf.UHF(mol)
    mf.verbose = 0
    mf.conv_tol = 1e-10
    e_hf = mf.kernel()

    print(f'UHF енергія: {e_hf:.10f} Ha')

    # Забруднення спіном
    s2_hf = mf.spin_square()[0]
    expected_s2 = spin * (spin + 2) / 4
    print(f'<S^2> (UHF): {s2_hf:.6f} (очікується {expected_s2:.6f})')

    # UMP2
    mump2 = mp.UMP2(mf)
    mump2.verbose = 0
    e_mp2_corr, t2 = mump2.kernel()

    e_total = e_hf + e_mp2_corr

    print(f'\nUMP2 кореляція: {e_mp2_corr:.10f} Ha')
    print(f'UMP2 повна енергія: {e_total:.10f} Ha')

    # MP2 не виправляє забруднення спіном
    # (для цього потрібні інші методи)

    return e_hf, e_total, e_mp2_corr

# Приклади
e_hf_li, e_mp2_li, _ = ump2_calculation('Li', spin=1)
e_hf_c, e_mp2_c, _ = ump2_calculation('C', spin=2)
e_hf_n, e_mp2_n, _ = ump2_calculation('N', spin=3)
e_hf_o, e_mp2_o, _ = ump2_calculation('O', spin=2)
```

### 5.2.5. Систематичне порівняння HF vs MP2.

```
from pyscf import gto, scf, mp
import numpy as np
import matplotlib.pyplot as plt

def hf_vs_mp2_comparison(atoms_list, basis='cc-pvtz'):
    """
    Систематичне порівняння HF та MP2 для списку атомів
    """

    results = {
```

```

    'atoms': [],
    'e_hf': [],
    'e_mp2': [],
    'e_corr': [],
    'corr_percent': []
}

print(f'\nПорівняння HF та MP2 (базис: {basis})')
print('='*80)
print(f'{"Атом":6s} {"Спін":4s} {"E(HF), Ha":15s} {"E(MP2), Ha":15s} '
      f'{"E_corr, mHa":12s} {"% від HF":10s}')
print('='*80)

for symbol, spin in atoms_list:
    mol = gto.M(
        atom=f'{symbol} 0 0 0',
        basis=basis,
        spin=spin,
        verbose=0
    )

    # HF
    if spin == 0:
        mf = scf.RHF(mol)
        mp_method = mp.MP2
    else:
        mf = scf.UHF(mol)
        mp_method = mp.UMP2

    mf.verbose = 0
    mf.conv_tol = 1e-10
    e_hf = mf.kernel()

    # MP2
    mymp2 = mp_method(mf)
    mymp2.verbose = 0
    e_mp2_corr, _ = mymp2.kernel()
    e_mp2 = e_hf + e_mp2_corr

    # Зберігаємо результати
    results['atoms'].append(symbol)
    results['e_hf'].append(e_hf)
    results['e_mp2'].append(e_mp2)
    results['e_corr'].append(e_mp2_corr)

    corr_percent = abs(e_mp2_corr / e_hf) * 100
    results['corr_percent'].append(corr_percent)

    print(f'{symbol:6s} {spin:4d} {e_hf:15.8f} {e_mp2:15.8f} '
          f'{e_mp2_corr*1000:12.4f} {corr_percent:10.4f}')

print('='*80)

# Статистика
avg_corr = np.mean(results['corr_percent'])
print(f'\nСередній % кореляції: {avg_corr:.4f}%')

# Графіки
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

x = np.arange(len(results['atoms']))

# Абсолютні енергії
ax1.plot(x, results['e_hf'], 'o-', label='HF',
         linewidth=2, markersize=8, color='blue')

```

```

ax1.plot(x, results['e_mp2'], 's-', label='MP2',
         linewidth=2, markersize=8, color='red')
ax1.set_xticks(x)
ax1.set_xticklabels(results['atoms'])
ax1.set_xlabel('Атом', fontsize=12)
ax1.set_ylabel('Енергія (Ha)', fontsize=12)
ax1.set_title('HF vs MP2 енергії', fontsize=14)
ax1.legend()
ax1.grid(True, alpha=0.3)

# Кореляційна енергія
ax2.bar(x, np.array(results['e_corr'])*1000, color='green',
        alpha=0.7)
ax2.set_xticks(x)
ax2.set_xticklabels(results['atoms'])
ax2.set_xlabel('Атом', fontsize=12)
ax2.set_ylabel('Кореляційна енергія (мHa)', fontsize=12)
ax2.set_title('MP2 кореляційна енергія', fontsize=14)
ax2.grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.savefig('hf_vs_mp2_comparison.pdf')
plt.show()

return results

# Атоми другого періоду
atoms_2nd = [
    ('He', 0), ('Li', 1), ('Be', 0), ('B', 1),
    ('C', 2), ('N', 3), ('O', 2), ('F', 1), ('Ne', 0)
]

results = hf_vs_mp2_comparison(atoms_2nd)

```

**5.2.6. Залежність MP2 від базисного набору.** MP2 сильно залежить від базису, особливо потрібні функції високих  $l$ :

```

from pyscf import gto, scf, mp
import matplotlib.pyplot as plt

def mp2_basis_convergence(symbol, spin):
    """
    Дослідження збіжності MP2 від базису
    """

    basis_sets = ['cc-pvdz', 'cc-pvtz', 'cc-pvqz', 'cc-pv5z']

    print(f'\nЗбіжність MP2 енергії {symbol} від базису')
    print('='*70)
    print(f'{"Базис":12s} {"N_bas":6s} {"E(HF)":15s} {"E(MP2)":15s} '
          f'{"E_corr, мHa":12s}')
    print('='*70)

    e_hf_list = []
    e_mp2_list = []
    e_corr_list = []
    n_bas_list = []

    for basis in basis_sets:
        mol = gto.M(
            atom=f'{symbol} 0 0 0',

```

```

        basis=basis,
        spin=spin,
        verbose=0
    )

    # HF
    if spin == 0:
        mf = scf.RHF(mol)
        mp_method = mp.MP2
    else:
        mf = scf.UHF(mol)
        mp_method = mp.UMP2

    mf.verbose = 0
    mf.conv_tol = 1e-11
    e_hf = mf.kernel()

    # MP2
    mymp2 = mp_method(mf)
    mymp2.verbose = 0
    e_corr, _ = mymp2.kernel()
    e_mp2 = e_hf + e_corr

    n_bas = mol.nao_nr()

    e_hf_list.append(e_hf)
    e_mp2_list.append(e_mp2)
    e_corr_list.append(e_corr)
    n_bas_list.append(n_bas)

    print(f'{basis:12s} {n_bas:6d} {e_hf:15.8f} {e_mp2:15.8f} '
          f'{e_corr*1000:12.4f}')

print('='*70)

# Екстраполяція до CBS
#  $E(X) = E_{CBS} + A/X^3$ 
X = np.array([2, 3, 4, 5]) # D, T, Q, 5

# CBS для HF (швидша збіжність)
A_hf = (e_hf_list[-1]*5**3 - e_hf_list[-2]*4**3) / (5**3 - 4**3)
e_hf_cbs = e_hf_list[-1] - A_hf / 5**3

# CBS для кореляції
A_corr = (e_corr_list[-1]*5**3 - e_corr_list[-2]*4**3) / (5**3 - 4**3)
e_corr_cbs = e_corr_list[-1] - A_corr / 5**3

e_mp2_cbs = e_hf_cbs + e_corr_cbs

print(f'\nЕкстраполяція до CBS:')
print(f'E(HF/CBS): {e_hf_cbs:.10f} Ha')
print(f'E(MP2/CBS): {e_mp2_cbs:.10f} Ha')
print(f'E_corr(CBS): {e_corr_cbs*1000:.4f} mHa')

# Графік
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Повна енергія
ax1.plot(n_bas_list, e_hf_list, 'o-', label='HF',
         linewidth=2, markersize=8)
ax1.plot(n_bas_list, e_mp2_list, 's-', label='MP2',
         linewidth=2, markersize=8)
ax1.axhline(y=e_hf_cbs, color='blue', linestyle='--',
            alpha=0.5, label='HF CBS')
ax1.axhline(y=e_mp2_cbs, color='orange', linestyle='--',

```

```

        alpha=0.5, label='MP2 CBS')
ax1.set_xlabel('Кількість базисних функцій', fontsize=12)
ax1.set_ylabel('Енергія (Ha)', fontsize=12)
ax1.set_title(f'Збіжність енергії {symbol}', fontsize=14)
ax1.legend()
ax1.grid(True, alpha=0.3)

# Кореляційна енергія
ax2.plot(n_bas_list, np.array(e_corr_list)*1000, 'o-',
        linewidth=2, markersize=8, color='green')
ax2.axhline(y=e_corr_cbs*1000, color='green', linestyle='--',
        alpha=0.5, label='CBS')
ax2.set_xlabel('Кількість базисних функцій', fontsize=12)
ax2.set_ylabel('Кореляційна енергія (mHa)', fontsize=12)
ax2.set_title(f'Збіжність кореляції {symbol}', fontsize=14)
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig(f'{symbol}_mp2_basis_convergence.pdf')
plt.show()

```

# Приклади

```
mp2_basis_convergence('Ne', spin=0)
```

```
mp2_basis_convergence('C', spin=2)
```

### 5.2.7. Переваги та недоліки MP2.

#### Переваги:

- Відносно швидкий ( $\mathcal{O}(N^5)$ ) порівняно з іншими post-HF
- Добре відновлює динамічну кореляцію
- Size-consistent (правильне масштабування для великих систем)
- Доступний для великих атомів/молекул
- Добрий базис для вищих методів (MP3, MP4)

#### Недоліки:

- Не виправляє забруднення спіном UHF
- Може розходитись для систем з малим gap HOMO-LUMO
- Погано працює для сильно корельованих систем
- Потребує великих базисів для точних результатів
- Не варіаційний (може давати енергію нижчу за точну)

### 5.2.8. Рекомендації по використанню MP2.

Таблиця 5.2. Коли використовувати MP2

Добре для:	Погано для:
Замкнені оболонки	Системи з малим HOMO-LUMO gap
Якісні оцінки кореляції	Розрив зв'язків
Великі системи	Збуджені стани
Слабкі взаємодії	Перехідні стани
Відносні енергії	Діелектронні системи

# Література

---

## Основна література

1. *Jensen F.* Introduction to Computational Chemistry. — 3rd ed. — Wiley, 2017. — 661 p. — ISBN 1118825993.
2. *Levine I. N.* Quantum Chemistry. — 7th ed. — Pearson, 2014. — 714 p. — ISBN 978-0321803450.

## Додаткові посилання

1. [Basis Sets](https://gaussian.com/basissets/). — URL: <https://gaussian.com/basissets/>.
2. *Ho M., Hernández-Perez J. M.* [Evaluation of Gaussian Molecular Integrals. I Overlap Integrals](#) // The Mathematica Journal. — 2012. — Vol. 14.
3. *Ho M., Hernández-Perez J. M.* [Evaluation of Gaussian Molecular Integrals. II. Kinetic-Energy Integrals](#) // The Mathematica Journal. — 2013. — Vol. 15.
4. *Ho M., Hernández-Perez J. M.* [Evaluation of Gaussian Molecular Integrals. III. Nuclear-Electron Attraction Integrals](#) // The Mathematica Journal. — 2014. — Vol. 16.
5. [Simple Quantum Chemistry: Hartree-Fock in Python](https://nznano.blogspot.com/2018/03/simple-quantum-chemistry-hartree-fock.html). — URL: <https://nznano.blogspot.com/2018/03/simple-quantum-chemistry-hartree-fock.html>.