МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КПІ ім. Ігоря Сікорського

Фізико-технічний інститут

$$\psi_k = \sum_\mu c_{\mu k}\chi_\mu \quad \text{(LCAO)}$$
$$\hat{F}\varphi_i = \varepsilon_i\varphi_i$$
$$S_{\mu\nu} = \int \chi_\mu^*(\mathbf{r})\chi_\nu(\mathbf{r})\,d\mathbf{r}$$

С. М. Пономаренко

Квантово-механічні методи обчислення

Використання PySCF

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

КИЇВ 2025

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

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
- H5py (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 XC 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
```

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:

code_1.py

```python
import pyscf
# Вивід 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_2, 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:

code_2.py

```python
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:

code_3.py

```
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:

code_4.py

```
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:

$$\hbar = 1$$

$$m_e = 1 \quad \text{(маса електрона)}$$
$$e = 1 \quad \text{(елементарний заряд)}$$
$$4\pi\varepsilon_0 = 1 \quad \text{(електрична константа)}$$

У цій системі:

- Енергія вимірюється в Hartree (Ha): 1 Ha = 27.211386 eV = 627.509 kcal/mol
- Відстань вимірюється в Bohr (a_0): 1 Bohr = 0.529177 Å
- Час вимірюється у a.u.: 1 a.u. = 2.4189×10^{-17} s

1.4.2. Конвертація одиниць

PySCF надає модуль для конвертації:

code_5.py

```python
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} Å')
```

1.4.3. Системні константи

Доступ до фізичних констант:

code_6.py

```python
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 ангстреми (Ångström), але можна вказати одиниці:

code_7.py

```python
# Координати в Ångström (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_α та N_β — кількість альфа та бета електронів:

code_8.py
```
# Атом H (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 O 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:

code_9.py
```
# 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	https://pyscf.org
GitHub репозиторій	https://github.com/pyscf/pyscf
Документація API	https://pyscf.org/pyscf_api_docs/pyscf.html
Приклади коду	https://github.com/pyscf/pyscf/tree/master/examples

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.
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.

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 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 **спін** (мультиплетність);
- **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 `Mole()` Цей спосіб є найзручнішим для простих систем, коли всі параметри можна вказати безпосередньо при створенні об'єкта:

MoleCreationMethod1.py

```
from pyscf import gto
```

```
# Простий спосіб
mol = gto.Mole(
    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.Mole()` 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:

MoleCreationMethod2.py

```
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:

SystemMainInfo.py

```
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 кіль-

кома 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'`).

AtomDefinitionWay1.py

```python
# 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)]`.

AtomDefinitionWay2.py

```python
# 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 H–H.

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.

IonsExample.py

```python
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 — 2 електрони (втратив один);
- Аніон Li^- — 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)
```

VerbosityExample.py

```python
# 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.

ValenceSplitBasisExample.py

```
# 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')

# З 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'а** — `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).

CorrelationConsistentBasisExample.py

```
# 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 експоненційним 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.

Def2BasisExample.py

```
# 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 (ECP)**, 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.

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()`.

CustomBasisExample.py

```
from pyscf import gto

# Визначення базису для H (тільки s-функції)
mol = gto.M(
```

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

```
    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.

CombiningBasesExample.py

```
# 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`.

BasisSetInfo.py

```
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.

SymmetryPointGroupsExample.py

```python
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.

UsingSymmetryExample.py

```python
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 орбіталей після розрахунку. Це особливо корисно для аналізу заповнених і віртуальних рівнів, побудови діаграм енергетичних рівнів та порівняння з експериментом.

OrbitalSymmetryExample.py

```
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[1];
- `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.

[1] **Незвідне представлення (irrep)** — фундаментальне поняття теорії груп, що описує, як функція (зокрема молекулярна орбіталь) трансформується під дією операцій симетрії молекули. Кожне ірредуцибельне представлення відповідає певному типу симетрії: наприклад, для групи C_{2v} це A_1, A_2, B_1, B_2, а для групи D_{2h} — A_g, B_{1g}, B_{2g}, B_{3g}, A_u, B_{1u}, B_{2u}, B_{3u}. Всі орбіталі, що належать до одного й того ж irrep, мають однакову симетрію і можуть змішуватися між собою, тоді як орбіталі різних `irrep` не взаємодіють.

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» (α) або «вниз» (β).

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 α- та β-електронів:

$$2S = N_\alpha - N_\beta \tag{2.1}$$

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 $(M = 4)$

SpinExamples.py

```python
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_α, N_β), 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.

PrintElectronConfig.py

```python
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 α- та β-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:

Таблиця 2.2. Основні спінові стани атомів другого періоду

Атом	Конфігурація	Терм	S	2S
Li	[He] $2s^1$	^{2}S	1/2	1
Be	[He] $2s^2$	^{1}S	0	0
B	[He] $2s^2\ 2p^1$	^{2}P	1/2	1
C	[He] $2s^2\ 2p^2$	^{3}P	1	2
N	[He] $2s^2\ 2p^3$	^{4}S	3/2	3
O	[He] $2s^2\ 2p^4$	^{3}P	1	2
F	[He] $2s^2\ 2p^5$	^{2}P	1/2	1
Ne	[He] $2s^2\ 2p^6$	^{1}S	0	0

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.

EnergyConfirmation2ndPeriod.py

```
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 квартету.

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 спін часто призводить до `SCF not converged`.
- При використанні `DFT` (наприклад, `scf.UKS`) параметр `spin` також впливає на спінову поляризацію густини.

2.5. Практичний приклад: комплексний аналіз атома

Розглянемо повний приклад аналізу атома з використанням всіх описаних концепцій:

AtomComplexAnalisys.py

```
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')
```

2.6. Корисні функції та методи

2.6.1. Інформація про атом

AtomInfoExample.py

```python
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}')
```

2.6.2. Маніпуляції з базисом

BasisManipulations.py

```python
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()}')
```

2.6.3. Енергетичні компоненти

EnergyComponents.py

```python
from pyscf import gto, scf
```

```python
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]}')
```

2.6.4. 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.

2.6.5. 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?

2.6.6. 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^+.
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:

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

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

Канонічні рівняння Хартрі-Фока мають вигляд:

$$\hat{f}\psi_i = \varepsilon_i \psi_i, \tag{3.2}$$

де $\hat{f}$ — оператор Фока:

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

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

3.1.2. Енергія Хартрі-Фока

Повна енергія системи у методі HF:

$$E_{HF} = \sum_{i=1}^{N} h_{ii} + \frac{1}{2} \sum_{i,j}^{N} (J_{ij} - K_{ij}) + V_{NN}, \tag{3.4}$$

де:

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

Віральне співвідношення. Для будь-якої стаціонарної хвильової функції (зокрема, у наближенні HF) повинно виконуватись **віральне співвідношення**:

$$2\langle T\rangle + \langle V\rangle = 0,$$

або еквівалентно:

$$\frac{\langle V\rangle}{\langle T\rangle} = -2.$$

Цей критерій є важливим тестом коректності збіжного SCF-розв'язку.

3.1.3. Матриця густини та енергетичні складові

Для практичної реалізації HF-методу у базисному представленні вводиться **матриця густини**:

$$D_{\mu\nu} = 2\sum_{i}^{\text{occ}} C_{\mu i} C_{\nu i}, \tag{3.5}$$

де $C_{\mu i}$ — коефіцієнти розкладу i-ї молекулярної орбіталі за базисом атомних орбіталей, а множник 2 враховує заповнення двома спінами (для RHF).

Оператор середнього значення будь-якої одноелектронної величини $\hat{O}$ можна записати у матричній формі:

$$\langle \hat{O}\rangle = \mathrm{Tr}(D\,O), \tag{3.6}$$

що значно спрощує обчислення середніх значень у базисі атомних орбіталей.

Одноелектронний гамільтоніан:

$$h_{\mu\nu} = T_{\mu\nu} + V_{\mu\nu}^{\text{nuc}}, \tag{3.7}$$

де $T_{\mu\nu}$ — інтеграли кінетичної енергії, а $V_{\mu\nu}^{\text{nuc}}$ — інтеграли притягання електрона до ядра.

Матриця Фока визначається як:

$$F_{\mu\nu} = h_{\mu\nu} + \sum_{\lambda\sigma} D_{\lambda\sigma}\left[(\mu\nu|\lambda\sigma) - \frac{1}{2}(\mu\lambda|\nu\sigma)\right]. \tag{3.8}$$

Тоді загальна енергія Гартрі–Фока може бути записана компактно:

$$E_{\text{HF}} = \sum_{\mu\nu} D_{\mu\nu}\left(h_{\mu\nu} + \frac{1}{2}F_{\mu\nu}\right). \tag{3.9}$$

У розгорнутому вигляді можна виділити окремі внески:

$$E_{\text{kin}} = \sum_{\mu\nu} D_{\mu\nu} T_{\mu\nu},$$
$$E_{\text{nuc}} = \sum_{\mu\nu} D_{\mu\nu} V_{\mu\nu}^{\text{nuc}},$$
$$E_{\text{ee}} = \frac{1}{2} \sum_{\mu\nu} D_{\mu\nu} (F_{\mu\nu} - h_{\mu\nu}),$$

і, відповідно, повна енергія:

$$E_{\text{tot}} = E_{\text{kin}} + E_{\text{nuc}} + E_{\text{ee}} + V_{NN}.$$

Таким чином, матриця густини D є центральним об'єктом, що зберігає всю інформацію про одноелектронні властивості системи. У коді `PySCF` вона створюється викликом `dm = mf.make_rdm1()`, а середні значення обчислюються як добутки D на відповідні матриці інтегралів.

3.1.4. Варіанти методу Хартрі-Фока

Restricted Hartree-Fock (RHF)

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

$$\psi_i^\alpha(\mathbf{r}) = \psi_i^\beta(\mathbf{r}) = \varphi_i(\mathbf{r}) \tag{3.10}$$

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:

$$\psi_i^\alpha(\mathbf{r}) \neq \psi_i^\beta(\mathbf{r}) \tag{3.11}$$

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:

$$\begin{cases} \psi_i^\alpha = \psi_i^\beta & \text{для спарених} \\ \psi_i^\alpha \neq \psi_i^\beta & \text{для неспарених} \end{cases} \tag{3.12}$$

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.

code_1.py

```python
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.

code_2.py

```
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.

code_3.py

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

Після завершення SCF-розрахунку `PySCF` надає доступ до одноелектронних інтегралів (`int1e_kin`, `int1e_nuc`) та до двоелектронної частини (`get_veff`). Це дозволяє безпосередньо обчислити всі компоненти енергії та перевірити віральне співвідношення.

HeHF.py

```python
from pyscf import gto, scf

# --- 1. Опис атома гелію ---
mol = gto.Mole()
mol.atom  = 'He 0 0 0'
mol.basis = 'cc-pVTZ'
mol.build()

# --- 2. Розрахунок RHF ---
mf = scf.RHF(mol)
E_tot = mf.kernel()

# --- 3. Матриця густини та інтеграли ---
dm = mf.make_rdm1()
T_int = mol.intor('int1e_kin')       # оператор кінетичної енергії
V_nuc_int = mol.intor('int1e_nuc')   # оператор потенціальної енергії ядро-електрон
vhf = mf.get_veff(mol, dm)           # ефективний потенціал (Кулон + обмін)

# --- 4. Енергетичні складові ---
E_kin = (dm * T_int).sum()           # <T>
E_nuc = (dm * V_nuc_int).sum()       # <V_nuc>
E_ee  = 0.5 * (dm * vhf).sum()       # <V_ee> (Кулон + обмін)
E_tot_check = E_kin + E_nuc + E_ee   # перевірка

# --- 5. Віральне співвідношення ---
V_total = E_nuc + E_ee
virial_ratio = V_total / E_kin

# --- 6. Вивід у вигляді таблиці ---
print("\n" + "="*60)
print(" "*15 + "ЕНЕРГЕТИЧНИЙ АНАЛІЗ АТОМА He")
print("="*60)
print(f"{'Компонента':<30} {'Значення (Ha)':>20}")
print("-"*60)
print(f"{'Кінетична енергія (T)':<30} {E_kin:>20.6f}")
print(f"{'Ядро-електрони (V_nuc)':<30} {E_nuc:>20.6f}")
print(f"{'Електрон-електрон (V_ee)':<30} {E_ee:>20.6f}")
print("-"*60)
print(f"{'Повна енергія (E_tot)':<30} {E_tot:>20.6f}")
```

```
print(f"{'Перевірка суми (E_sum)':<30} {E_tot_check:>20.6f}")
print("="*60)
print(f"\n{'Віральне співвідношення':<30} {'V/T':>20}")
print("-"*60)
print(f"{'Розраховане значення':<30} {virial_ratio:>20.3f}")
print(f"{'Теоретичне значення':<30} {-2.000:>20.3f}")
print("="*60 + "\n")
```

Коментар. Якщо розрахунок збіжний і фізично коректний, то відношення $\langle V\rangle/\langle T\rangle \approx -2.00 \pm 0.01$, що підтверджує виконання віральної теореми. Будь-яке суттєве відхилення (наприклад, -1.7 чи -2.3) свідчить про помилку збіжності або неадекватність базисного набору.

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 α та β орбіталям відрізнятися, але тут ця свобода не використовується.

code_5.py

```
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<S²> (UHF): {s2_uhf[0]:.6f}")
print("<S²> (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 $\equiv$ 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.

code_6.py

```python
from pyscf import gto, scf


def he_excited_state(config="1s2s", spin=0, basis="cc-pvdz"):
    """
    Розрахунок збуджених станів He (¹S, ³S) з використанням MOM
    """
    mol = gto.M(atom="He 0 0 0", basis=basis, spin=spin)

    # RHF для синглету, UHF для триплету
    if spin == 0:
        mf = scf.RHF(mol)
    else:
        mf = scf.UHF(mol)

    mf.verbose = 0
    mf.kernel()  # звичайний HF, щоб ініціалізуватись

    # MOM для стабілізації збудженого стану
    mom_solver = scf.addons.mom_occ(mf, mf.make_rdm1(), mf.mo_occ)
    mom_solver.verbose = 0
    mom_solver.max_cycle = 100
    e_exc = mom_solver.kernel()

    return e_exc


# 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("-" * 55)
print(f"1s² (¹S, основний): {e_gs:.6f} Ha  = {e_gs * 27.2114:.2f} eV")

# Збуджені стани
for spin, label in [(0, "1s2s (¹S)"), (2, "1s2s (³S)")]:
    e_exc = he_excited_state(spin=spin)
    print(f"{label:15s}: {e_exc: .6f} Ha  ({(e_exc - e_gs)*27.2114: .3f} eV вище основного)")

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

Коментар. Метод `MOM (Maximum Overlap Method)` — це модифікація самозгодженої процедури HF, у якій вибір зайнятих орбіталей на кожній ітерації здійснюється не за найменшими енергіями, а за критерієм **максимального перекриття** з орбіталями попередньої ітерації:

$$O_{ij} = \left| \langle \varphi_i^{(n)} \mid \varphi_j^{(n-1)} \rangle \right|^2 .$$

Таким чином, MOM «закріплює» бажану орбітальну конфігурацію (наприклад, $1s2s$) і не дозволяє хвильовій функції сповзати назад до основного

стану ($1s^2$), що часто трапляється у звичайному HF.

Попри те, що MOM стабілізує збуджені конфігурації, він залишається одно-детермінантним наближенням і не враховує електронної кореляції, тому не належить до post-HF методів. Отримані стани є лише самозгодженими розв'язками рівнянь HF, а не справжніми власними станами гамільтоніана. Для точного опису енергій і спектрів збудження слід застосовувати багатоконфігураційні або збурювальні підходи (`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 ($N_\alpha \neq N_\beta$, тобто густини ρ_α і ρ_β відрізняються).

Електронна конфігурація:

$$\text{Li: } 1s^2 2s^1.$$

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` є природним вибором. `UHF` дозволяє альфа- та бета-орбіталям відрізнятися, тобто хвильова функція не змушена мати однакову просторову форму для різних спінів.

Альтернативою є метод `ROHF (Restricted Open-Shell Hartree-Fock)`, у якому

- усі замкнені орбіталі мають однакові просторові функції для α та β спінів;
- відкриті орбіталі (неспарені) можуть мати різну зайнятість, але однакову форму.

ROHF забезпечує правильне значення $\langle S^2 \rangle$ та зберігає спінову симетрію, проте формулювання рівнянь Фока є складнішим. UHF, навпаки, простіший у реалізації, але може призводити до *spin contamination*. У практиці обидва методи дають близькі енергії для атома Li, проте ROHF вважається формально більш коректним.

code_7.py

```
from pyscf import gto, scf
```

```python
# --- 1. Визначення молекули ---
mol = gto.M(
    atom="Li 0 0 0",
    basis="cc-pvdz",
    spin=1,          # неспарений електрон (S=1/2)
    symmetry=True,
)

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

# --- 2. UHF ---
uhf = scf.UHF(mol)
E_uhf = uhf.kernel()

print("=== UHF ===")
print(f"Енергія Li (UHF): {E_uhf:.8f} Ha")
print(f"Енергія Li (UHF): {E_uhf * 27.211386:.4f} eV")

s2_uhf = uhf.spin_square()
print(f"<S²> = {s2_uhf[0]:.6f}  (очікується 0.75)\n")

# --- 3. ROHF ---
rohf = scf.ROHF(mol)
E_rohf = rohf.kernel()

print("=== ROHF ===")
print(f"Енергія Li (ROHF): {E_rohf:.8f} Ha")
print(f"Енергія Li (ROHF): {E_rohf * 27.211386:.4f} eV")

s2_rohf = rohf.spin_square()
print(f"<S²> = {s2_rohf[0]:.6f}  (очікується 0.75)\n")

# --- 4. Порівняння ---
ΔE = (E_uhf - E_rohf) * 27.211386
print(f"Різниця (UHF - ROHF): {ΔE:.6f} eV")

# --- 5. Коментар ---
# UHF дозволяє різні орбіталі для α і β спінів,
# тому виникає спінове забруднення (<S²> > 0.75).
# ROHF зберігає правильну спінову симетрію, але менш гнучкий варіаційно.
```

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

$$\text{Be: } 1s^2 2s^2.$$

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).

code_8.py

```
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("  Конфігурація: [He] 2s²")
print("  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 (≈ 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:	$[\mathrm{He}]\,2s^1$	^2S
Be:	$[\mathrm{He}]\,2s^2$	^1S
B:	$[\mathrm{He}]\,2s^2 2p^1$	^2P
C:	$[\mathrm{He}]\,2s^2 2p^2$	^3P
N:	$[\mathrm{He}]\,2s^2 2p^3$	^4S
O:	$[\mathrm{He}]\,2s^2 2p^4$	^3P
F:	$[\mathrm{He}]\,2s^2 2p^5$	^2P
Ne:	$[\mathrm{He}]\,2s^2 2p^6$	^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.

code_9.py

```
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, "□S", "[He] 2s² 2p³"),
    ("O", 2, "³P", "[He] 2s² 2p□"),
    ("F", 1, "²P", "[He] 2s² 2p□"),
    ("Ne", 0, "¹S", "[He] 2s² 2p□"),
]

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} {'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: 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 Be→B, N→O, F→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.

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.

code_11.py

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


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. Енергії іонізації

Іонізаційна енергія I є однією з фундаментальних характеристик атома, що визначає мінімальні енерговитрати на видалення одного електрона із нейтрального атома в основному стані:

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

де $E(A)$ — повна енергія нейтрального атома, а $E(A^+)$ — енергія його однозарядного катіона. Ця величина безпосередньо вимірюється експериментально (у електронвольтах) і є важливим тестом якості будь-якого квантово-хімічного методу.

У межах методу Гартрі–Фока іонізаційна енергія може бути визначена двома способами.

Метод ΔSCF. Найбільш пряме обчислення полягає у незалежному знаходженні повних енергій нейтрального атома і катіона:

$$I_{\Delta\mathrm{SCF}} = E(A^+) - E(A).$$

Такий підхід враховує релаксацію орбіталей після видалення електрона, тому дає точніші результати, ніж прості наближення.

Теорема Купманса. У межах наближення «заморожених орбіталей» (*frozen orbital approximation*) повна енергія системи вважається незмінною при видаленні одного електрона. У цьому випадку зміна енергії дорівнює орбітальній енергії найвищої зайнятої молекулярної орбіталі:

$$I_{\mathrm{Koopmans}} \approx -\varepsilon_{\mathrm{HOMO}}.$$

Це твердження відоме як **теорема Купманса**. Воно дозволяє оцінити енергії іонізації без повторного самозгодження, хоча і не враховує релаксаційні та кореляційні ефекти.

Порівняння підходів. Для легких атомів (Li, Be, B, C, N, O, F, Ne) метод Купманса правильно відтворює тенденцію зростання енергії іонізації уздовж періоду, але систематично недооцінює абсолютні значення. Метод ΔSCF наближається до експерименту краще, оскільки включає релаксацію орбіталей.

Приклад: атом Літію. Для Li (конфігурація $1s^2 2s^1$):

$$I_{\text{Koopmans}} = -\varepsilon_{2s}, \quad I_{\Delta\text{SCF}} = E(\text{Li}^+) - E(\text{Li}), \quad I_{\text{exp}} = 5.39 \text{ eV}.$$

Енергія Купманса дещо завищена, а результат ΔSCF ближчий до експериментального, що демонструє вплив орбітальної релаксації.

KoopmansTheoremCheck.py

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


def test_koopmans_theorem(atoms_list, basis="cc-pvtz"):
    """
    Перевірка теореми Купманса:
    порівняння ΔSCF, -εHOMO та експериментальних іонізаційних енергій.
    """

    # 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(f"\nПеревірка теореми Купманса (базис: {basis})")
    print("=" * 110)
    print(
        f"{'Атом':6s} {'IE(ΔSCF)':>12s} {'-εHOMO':>12s} {'Різниця':>12s} "
        f"{'%ΔSCF→HOMO':>12s} {'Exp.':>10s} {'%ΔSCF→Exp':>12s} {'%HOMO→Exp':>12s}"
    )
    print("-" * 110)

    results = {
        "atoms": [],
        "ie_scf": [],
        "ie_koop": [],
        "ie_exp": [],
        "diff_scf_koop": [],
        "diff_scf_exp": [],
        "diff_koop_exp": [],
    }

    for symbol, spin_n, spin_c in atoms_list:
        # Нейтральний атом
        mol = gto.M(atom=f"{symbol} 0 0 0", basis=basis, spin=spin_n, verbose=0)
        mf = scf.UHF(mol) if spin_n != 0 else scf.RHF(mol)
        mf.conv_tol = 1e-11
        e_neutral = mf.kernel()

        # HOMO енергія
        if spin_n == 0:
            n_occ = mol.nelectron // 2
            eps_homo = mf.mo_energy[n_occ - 1]
        else:
            n_alpha = mol.nelec[0]
            eps_homo = mf.mo_energy[0][n_alpha - 1]
```

```
        ie_koopmans = -eps_homo * 27.211386  # eV

        # Катион
        mol_cat = gto.M(atom=f"{symbol} 0 0 0", basis=basis, charge=1, spin=spin_c, verbose=0)
        mf_cat = scf.UHF(mol_cat) if spin_c != 0 else scf.RHF(mol_cat)
        mf_cat.conv_tol = 1e-11
        e_cation = mf_cat.kernel()

        ie_scf = (e_cation - e_neutral) * 27.211386  # eV

        # Порівняння
        diff_scf_koop = ie_koopmans - ie_scf
        diff_scf_koop_percent = abs(diff_scf_koop / ie_scf) * 100

        ie_exp = exp_ie.get(symbol, np.nan)
        diff_scf_exp_percent = abs((ie_scf - ie_exp) / ie_exp) * 100 if not np.isnan(ie_exp) else
        ↪  np.nan
        diff_koop_exp_percent = abs((ie_koopmans - ie_exp) / ie_exp) * 100 if not np.isnan(ie_exp)
        ↪  else np.nan

        print(
            f"{symbol:6s} {ie_scf:12.4f} {ie_koopmans:12.4f} {diff_scf_koop:12.4f} "
            f"{diff_scf_koop_percent:12.2f} {ie_exp:10.2f} "
            f"{diff_scf_exp_percent:12.2f} {diff_koop_exp_percent:12.2f}"
        )

        # Збереження
        results["atoms"].append(symbol)
        results["ie_scf"].append(ie_scf)
        results["ie_koop"].append(ie_koopmans)
        results["ie_exp"].append(ie_exp)
        results["diff_scf_koop"].append(diff_scf_koop)
        results["diff_scf_exp"].append(diff_scf_exp_percent)
        results["diff_koop_exp"].append(diff_koop_exp_percent)

    print("=" * 110)

    mean_diff = np.nanmean(np.abs(results["diff_scf_koop"]))
    mean_err_scf = np.nanmean(results["diff_scf_exp"])
    mean_err_koop = np.nanmean(results["diff_koop_exp"])

    print(f"\nСередня |ΔSCF - Koopmans| різниця: {mean_diff:.3f} eV")
    print(f"Середня похибка ΔSCF від експерименту: {mean_err_scf:.2f}%")
    print(f"Середня похибка -εHOMO від експерименту: {mean_err_koop:.2f}%")

    # --- Графік ---
    fig, ax = plt.subplots(figsize=(10, 6))
    x = np.arange(len(results["atoms"]))
    width = 0.25

    ax.bar(x - width, results["ie_exp"], width, label="Експеримент", color="gray", alpha=0.6)
    ax.bar(x, results["ie_scf"], width, label="IE (ΔSCF)", color="blue", alpha=0.7)
    ax.bar(x + width, results["ie_koop"], width, label="-εHOMO", color="red", alpha=0.7)

    ax.set_xlabel("Атом", fontsize=12)
    ax.set_ylabel("Енергія іонізації (eV)", fontsize=12)
    ax.set_title("Перевірка теореми Купманса", fontsize=14)
    ax.set_xticks(x)
    ax.set_xticklabels(results["atoms"])
    ax.legend()
    ax.grid(True, alpha=0.3, axis="y")

    plt.tight_layout()
    plt.savefig("koopmans_theorem_test.pdf")
```

```
    plt.show()

    return results


# --- Тестування ---
atoms_test = [
    ("Li", 1, 0),
    ("Be", 0, 1),
    ("B", 1, 0),
    ("C", 2, 1),
    ("N", 3, 2),
    ("O", 2, 3),
    ("F", 1, 2),
    ("Ne", 0, 1),
]

results_koop = test_koopmans_theorem(atoms_test)
```

Коментар. Розрахунок виводить таблицю порівняння енергій іонізації ($-\varepsilon_{\text{HOMO}}$, ΔSCF, експеримент) для атомів другого періоду та відносну похибку у відсотках:

$$\delta = 100 \times \frac{I_{\text{calc}} - I_{\text{exp}}}{I_{\text{exp}}}.$$

Аналіз цих відхилень дозволяє оцінити якість базисного набору і межі застосовності теореми Купманса.

3.5.3. Електронна густина та її інтерпретація

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 побуду-

вати одноосний «профіль густини» та простежити, як електронна густина спадає при віддаленні від ядра.

Для кожної точки z обчислюється матриця базисних функцій $\varphi_i(\mathbf{r})$, і далі густина:

$$\rho(\mathbf{r}) = \sum_{ij} \chi_i(\mathbf{r}) D_{ij} \chi_j(\mathbf{r}),$$

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.

code_12.py

```python
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 $\rho(0,0,z)$ 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 напрямах 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.

code_13.py

```
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)

# Кути 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 4πr²ρ(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 ρ(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 θ, φ) 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.

3.5.4. 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.

3.5.5. Зв'язок із електронними оболонками

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

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

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.

3.5.6. 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?

3.5.7. Методичні рекомендації

1. Змінюючи базис (наприклад, `STO-3G`, `6-31G`, `cc-pVTZ`), можна дослідити, як збільшується точність відтворення радіального профілю.
2. 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 α- і β-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 α- і β-електронів дозволено різні. Цей метод придатний для відкритих оболонок (наприклад, атомів з неспареними електронами), однак може страждати від *спінового забруднення (spin contamination).*
- `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.

Енергетичне порівняння

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

Орбітальні енергії

`UHF` формує два набори орбіталей — для α- і β-електронів. Тому орбітальні енергії можуть відрізнятись:

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

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

3.6.1. Спінове забруднення

У системах з відкритими оболонками метод `UHF` часто використовується як простіше наближення, що дозволяє займати незалежні орбіталі електронам зі спінами α та β. Однак відсутність суворого обмеження на спінову симетрію призводить до важливого недоліку — **спінового забруднення**.

Математична суть явища

`UHF`-хвильова функція не є власним станом оператора $\hat{S}^2$, хоч і залишається власним станом $\hat{S}_z$:

$$\hat{S}_z \Psi_{\text{UHF}} = M_S \Psi_{\text{UHF}}, \qquad \hat{S}^2 \Psi_{\text{UHF}} \neq S(S+1)\Psi_{\text{UHF}}.$$

Отже, очікуване значення

$$\langle S^2 \rangle = \langle \Psi_{\text{UHF}} | \hat{S}^2 | \Psi_{\text{UHF}} \rangle$$

зазвичай перевищує істинне $S(S+1)$, тобто:

$$\Delta S^2 = \langle S^2 \rangle - S(S+1) > 0.$$

Фізичне тлумачення

UHF-хвильова функція може бути подана як суперпозиція чистих спінових станів:

$$\Psi_{\text{UHF}} = c_0\Psi_S + c_1\Psi_{S+1} + c_2\Psi_{S+2} + \cdots,$$

де присутність коефіцієнтів $c_i \neq 0$ для $i > 0$ означає змішування різних мультиплетів. Наприклад, для триплетного стану ($S = 1$) теоретичне значення $\langle S^2\rangle = 2.0$, але якщо розрахунок UHF дає 2.25, це означає, що хвильова функція містить домішку п'ятичленного ($S = 2$) стану.

Вплив на результати

- **Енергія.** `UHF` може давати занадто низьку енергію через змішування спінів і штучне зменшення кулонівського відштовхування.
- **Електронна густина.** Спінова густина стає несиметричною, особливо поблизу атомів з неспареними електронами.
- **Магнітні властивості.** Похибки у спіновій густині ведуть до помилкових магнітних моментів і спотворених мультиплетних розщеплень.

Як обчислюється спінове забруднення

Програми квантово-хімічних розрахунків (зокрема `PySCF`, `Gaussian`, `ORCA`) обчислюють спінове забруднення не безпосередньо через оператор $\hat{S}^2$, а через перекриття між орбіталями спінів α і β.

Основна формула для середнього значення $\langle S^2\rangle$ у наближенні `UHF` має вигляд:

$$\boxed{\langle S^2\rangle = \frac{(N_\alpha - N_\beta)^2}{4} + \frac{N_\alpha + N_\beta}{2} - \sum_{i=1}^{N_\alpha}\sum_{j=1}^{N_\beta} |\langle\varphi_i^\alpha|\varphi_j^\beta\rangle|^2}$$

де:

- N_α, N_β — кількість електронів зі спінами α та β відповідно;
- φ_i^α, φ_j^β — молекулярні(атомні) орбіталі для спінів α і β;
- $\langle\varphi_i^\alpha|\varphi_j^\beta\rangle$ — перекриття між орбіталями різних спінів.

Якщо орбіталі α і β однакові, тобто $\varphi_i^\alpha = \varphi_i^\beta$, то:

$$\langle S^2\rangle = \frac{(N_\alpha - N_\beta)^2}{4},$$

і для синглету ($N_\alpha = N_\beta$) це дорівнює нулю — спінове забруднення відсутнє.

Інтерпретація

Якщо орбіталі α і β подібні, перекриття $S_{\alpha\beta}$ велике, тому Σ наближається до N_β, і $\langle S^2\rangle$ близьке до теоретичного $S(S+1)$ — тобто забруднення мінімальне.

Якщо ж орбіталі сильно відрізняються (наприклад, при розриві зв'язку або в радикалах), Σ зменшується, і $\langle S^2 \rangle$ зростає — з'являється суттєве спінове забруднення.

Діагностика та критерії

Значення $\langle S^2 \rangle$ зазвичай друкується у вихідних даних програми. Практичні межі:

$$\Delta S^2 = \langle S^2 \rangle - S(S+1),$$

$$\begin{cases} \Delta S^2 < 0.05 & \text{— незначне забруднення, прийнятно;} \\ 0.05 < \Delta S^2 < 0.10 & \text{— помірне, бажано перевірити;} \\ \Delta S^2 > 0.10 & \text{— суттєве, слід перейти до ROHF або проєкційних методів.} \end{cases}$$

Приклад чисельного розрахунку

Для системи з $N_\alpha = 5$, $N_\beta = 4$ (очікувано $S = \frac{1}{2}$, $\langle S^2 \rangle = 0.75$):

- якщо $\Sigma = 3.9$, тоді

$$\langle S^2 \rangle = \frac{1}{4} + \frac{9}{2} - 3.9 = 0.85,$$

$$\Delta S^2 = 0.85 - 0.75 = 0.10,$$

тобто забруднення перебуває на межі суттєвого;

- якщо $\Sigma = 3.4$, тоді

$$\langle S^2 \rangle = \frac{1}{4} + \frac{9}{2} - 3.4 = 1.35,$$

$$\Delta S^2 = 1.35 - 0.75 = 0.60,$$

що свідчить про сильне спінове забруднення.

Як зменшити спінове забруднення

1. **ROHF:** обмежена відкрита схема Гартрі–Фока, що зберігає спінову симетрію.
2. **Спін-проекційні методи:** після варіації застосовується оператор проєкції

$$\Psi_{\text{proj}} = \hat{P}_S \Psi_{\text{UHF}},$$

який видаляє небажані компоненти.

3. **Методи після HF:** MP2, CISD, CCSD частково компенсують спінове забруднення через багатоконфігураційність.

3.6.2. 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`.

code_15.py

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

# Орбітальні енергії
print("\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}")
```

Щоб узагальнити різницю між методами UHF та ROHF, виконаємо серію розрахунків для кількох відкритооболонкових атомів перших двох періодів. Це дозволить оцінити енергетичну різницю ΔE і побачити, чи зберігає ROHF спінову симетрію без втрати точності.

code_16.py

```
from pyscf import gto, scf

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("=" * 120)
print(f"{'Атом':4s} {'Спін':4s} {'E(UHF), Ha':15s} {'E(ROHF), Ha':15s} {'ΔE, mHa':10s}
↪  {'⟨S²⟩UHF':10s} {'ΔS²':8s} {'Оцінка':40s}")
print("-" * 120)

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.conv_tol = 1e-10
    e_uhf = mf_uhf.kernel()

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

    # Спінові характеристики
    s2_uhf, s_uhf = mf_uhf.spin_square()
    s2_rohf, s_rohf = mf_rohf.spin_square()

    # Теоретичне значення S(S+1)
    S = spin / 2
    S2_expected = S * (S + 1)

    delta_e = (e_uhf - e_rohf) * 1000  # mHa
    delta_s2 = s2_uhf - S2_expected

    # Класифікація за критеріями
    if delta_s2 < 0.05:
```

```
        quality = "незначне забруднення, прийнятно"
    elif delta_s2 < 0.10:
        quality = "помірне, бажано перевірити"
    else:
        quality = "суттєве забруднення, бажано ROHF або проєкційні методи"

    print(f"{symbol:4s} {spin:4d} {e_uhf:15.8f} {e_rohf:15.8f} {delta_e:10.4f} {s2_uhf:10.4f}
    ↪ {delta_s2:8.4f} {quality:40s}")

print("=" * 120)
print("\nПримітка:")
print("UHF часто дає нижчу енергію, але може мати спінове забруднення (ΔS²).")
print("ROHF зберігає спінову симетрію, але іноді трохи вищу енергію.")
```

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 mHa 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 спіновою 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 mHa.
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 простих систем ітераційна процедура самозгодженого поля (SCF) може не досягати стабільного рішення. Найчастіше проблеми збіжності виникають для перехідних металів, відкритих оболонок та систем із виродженими

орбіталями. У цьому розділі розглянемо основні джерела труднощів та методи їх подолання.

3.7.1. Причини поганої збіжності

Типові джерела нестабільності SCF:

- сильна електронна кореляція у незаповнених d- і f-оболонках;
- мала різниця енергій між орбіталями (виродження);
- невдалий початковий вектор коефіцієнтів (погане стартове наближення);
- занадто жорсткі або занадто м'які критерії збіжності.

Через ці фактори енергія може осцилювати, DIIS — розходитись, а результат — бути нефізичним.

3.7.2. Перехідні метали

Перехідні елементи (Fe, Co, Ni, Cr, Mn тощо) — класичні приклади систем із поганою збіжністю SCF. Причини:

- близькість енергетичних рівнів $3d$ і $4s$-орбіталей;
- можливість декількох спінових станів, близьких за енергією;
- наявність кількох локальних мінімумів функціоналу енергії.

У PySCF це проявляється як:

- осциляції енергії між ітераціями;
- розбігання DIIS;
- стрибки значення $\langle S^2 \rangle$ між кроками.

Нижче наведено приклад універсальної функції для стабільного розрахунку атомів перехідних металів із урахуванням практичних прийомів:

- використовується UHF (для врахування спіну);
- застосовується зсув рівнів (`level_shift`);
- розширюється DIIS-простір (`diis_space`);
- у разі потреби вмикається уточнення Ньютона–Рафсона;
- контролюється спін-забруднення через $\langle S^2 \rangle$.

code_18.py

```
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] 3d□ 4s¹, □S
mf_cr, e_cr = transition_metal_calculation("Cr", spin=6)

# Mn: [Ar] 3d□ 4s², □S
mf_mn, e_mn = transition_metal_calculation("Mn", spin=5)

# Fe: [Ar] 3d□ 4s², □D
mf_fe, e_fe = transition_metal_calculation("Fe", spin=4)
```

Такий підхід забезпечує стабільність навіть для важких атомів, де стандартний SCF часто не сходиться.

3.7.3. Вироджені орбіталі та дробові заповнення

Якщо в системі наявні вироджені або майже вироджені орбіталі, SCF може “коливатись”, не вибираючи між ними. Для таких випадків ефективним є застосування *дробових заповнень орбіталей* (*fractional occupations*), що згладжують різницю між рівнями енергії.

Ідея полягає у введенні ефективного теплового розмазування Фермі–Дірака:

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

де f_i — часткове заповнення орбіталі i, ε_i — її енергія, μ — хімічний потенціал, а kT — параметр розмазування.

Цей підхід дозволяє SCF уникнути нестійких перестрибувань між виродженими рівнями.

У PySCF для цього достатньо викликати `scf.addons.frac_occ(mf)`. Методика добре працює для атомів (V, Cr), радикалів та малих кластерів перехідних металів.

code_19.py

```python
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")
```

3.7.4. Стратегія досягнення збіжності SCF

Для надійного отримання фізично коректного рішення рекомендується послідовна стратегія стабілізації SCF — від простих прийомів до складніших:

1. стандартний UHF;
2. зсув рівнів (`level_shift`);
3. збільшення DIIS-простору (`diis_space`);
4. атомне початкове наближення (`init_guess='atom'`);
5. уточнення за методом Ньютона–Рафсона;

6. дробові заповнення (`frac_occ`) для вироджених станів.

code_20.py

```
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("\n✗ Жодна стратегія не спрацювала.")
    return None, None


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

Фізичний зміст прийомів.

- **Level shift** — підвищує енергії віртуальних орбіталей, запобігаючи коливанням.
- **DIIS-простір** — збільшення пам'яті ітерацій покращує апроксимацію поля.
- **Atom guess** — старт з атомних орбіталей ближчий до реального розв'язку.

- **Newton–Raphson** — забезпечує квадратичну збіжність поблизу мінімуму.
- **Fractional occupations** — стабілізують вироджені стани, забезпечуючи плавний перехід.

Практичні рекомендації.

- Для великих систем на перших етапах можна зменшити точність: `conv_tol=1e-6`.
- Якщо енергія осцилює — увімкніть `level_shift=0.5`.
- Якщо `UHF` не сходиться — використайте `init_guess='atom'` або вимкніть симетрію (`symmetry=False`).
- Завжди перевіряйте фізичність результату через $\langle S^2 \rangle$.

Таким чином, навіть якщо стандартна процедура SCF не збігається, послідовне застосування описаних прийомів практично гарантує стабільне та фізично обґрунтоване рішення.

3.8. Практичні завдання

3.8.1. Завдання 1: Систематичне дослідження

code_21.py

```
"""
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

code_22.py

```
"""
ЗАВДАННЯ 2: Обчисліть першу та другу енергії іонізації
для атомів Li, Be, B, C, N, O, F, Ne. Порівняйте з
експериментальними значеннями.
IE□ = E(A□) - E(A)
IE□ = E(A²□) - E(A□)
Використайте базис cc-pvtz
"""
# Ваш код тут
```

3.8.3. Завдання 3: Спектроскопічні константи

code_23.py

```
"""
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

code_24.py

```
"""
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. Теореми Хоенберга–Кона

Теорія функціоналу густини базується на двох фундаментальних теоремах, доведених Хоенбергом 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:

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

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

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

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:

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

Ефективний потенціал визначається як:

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

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:

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

4.1.3. Обмінно-кореляційна енергія

Повна енергія в DFT:

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

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

- Різницю між точною кінетичною енергією та кінетичною енергією невзаємодіючої 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 τ
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:

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

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

Обмінна частина (Slater/Dirac):

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

Кореляційна частина: Використовуються параметризації (VWN, PW92).

LSDA для спін-поляризованих систем

Для систем з різними ρ_α та ρ_β:

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

code_1.py

```python
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:

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

Популярні GGA функціонали

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

code_2.py

```
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:

code_3.py

```
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)

code_4.py

```
mf = dft.UKS(mol)
mf.xc = 'bp86'
energy_bp86 = mf.kernel()
print(f'Енергія C (BP86): {energy_bp86:.8f} Ha')
```

Порівняння GGA функціоналів

code_5.py

```
from pyscf import gto, dft

# 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 τ або лапласіан густини $\nabla^2\rho$:

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

де кінетична густина орбіталей Кона-Шема:

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

TPSS (Tao-Perdew-Staroverov-Scuseria)

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

code_6.py

```
from pyscf import gto, dft

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

# TPSS meta-GGA
mf = dft.UKS(mol)
mf.xc = "tpss"
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} mHa")
```

M06-L (Minnesota 06 Local)

Високопараметризований meta-GGA функціонал для широкого спектру задач:

code_7.py

```
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:

code_8.py

```
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:

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

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

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

Найпопулярніший гібридний функціонал у хімії:

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

code_9.py

```python
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("\nВнесок точного обміну: 20%")
print("Це робить розрахунок повільнішим, але точнішим")
```

PBE0 (PBE hybrid)

Неемпіричний гібрид з 25

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

code_10.py

```python
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):

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

code_11.py

```python
from pyscf import gto, dft

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

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

code_12.py

```python
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

code_13.py

```python
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")
```

4.2.7. Вибір функціоналу: рекомендації

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, ωB97X-D	З дисперсією
Високоспінові стани	SCAN, TPSSh	Кращий баланс
Загальні розрахунки	PBE0	Універсальний вибір
Максимальна точність	CCSD(T)	Але дуже повільно

4.2.8. Практичний приклад: вплив функціоналу на властивості

code_14.py

```
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} {'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} {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:

code_15.py

```python
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

code_16.py

```python
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} {'<S²>':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} {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:

code_17.py

```
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, □F
e_v, s2_v = transition_metal_dft("V", spin=3, functional="pbe")

# Cr: [Ar] 3d□ 4s¹, □S (виняток!)
e_cr, s2_cr = transition_metal_dft("Cr", spin=6, functional="pbe")

# Mn: [Ar] 3d□ 4s², □S
e_mn, s2_mn = transition_metal_dft("Mn", spin=5, functional="pbe")

# Fe: [Ar] 3d□ 4s², □D
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:

code_18.py

```
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} {'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} {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} ({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

code_19.py

```
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} {'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("\nОрбітальні енергії (eV):")
    print(f"{'MO':5s} {'α-енергія':12s} {'β-енергія':12s} {'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} 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:

code_20.py

```
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 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_au_nr = heavy_atom_calculation("Au", spin=1, relativistic=False)
e_au_r = heavy_atom_calculation("Au", spin=1, relativistic=True)
rel_effect = (e_au_r - e_au_nr) * 627.509  # kcal/mol
print(f"Релятивістський ефект: {rel_effect:.2f} kcal/mol")

# 5d метал: Платина
print("\nПлатина (Pt):")
e_pt_nr = heavy_atom_calculation("Pt", spin=2, relativistic=False)
e_pt_r = heavy_atom_calculation("Pt", spin=2, relativistic=True)
rel_effect = (e_pt_r - e_pt_nr) * 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:

code_21.py

```
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} {'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} {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:
            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

code_22.py

```python
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

code_23.py

```python
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 (mHa)", 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

code_24.py

```
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 IE
    """

    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, спін нейтрального, спін катіона, IE 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

code_25.py

```
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, спін нейтрального, спін 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

code_26.py

```
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} {'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

code_27.py

```
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 слабке 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

code_28.py

```
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")
```

4.4.7. Загальні висновки HF vs DFT

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	HOMO ≈ -IE (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

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 використаємо стандартні атомні властивості:

code_29.py

```python
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  # Спрощене 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}")
```

4.5.2. Рекомендації для різних елементів

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, ωB97X-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

4.6. Практичні завдання

4.6.1. Завдання 1: Систематичне дослідження

code_30.py

```
"""
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

code_31.py

```
"""
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
```

4.6.3. Завдання 3: Конвергенція до базисної межі

code_32.py

```
"""
ЗАВДАННЯ 3: Для атома Неону розрахуйте енергію з функціоналом
PBE0 та базисами cc-pVDZ, cc-pVTZ, cc-pVQZ, cc-pV5Z.
Екстраполюйте енергію до базисної межі (CBS) за формулою:
E(X) = E_CBS + A/X³
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

code_33.py

```
"""
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 α/β?
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 Хоенберга–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, спін, 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 $\geqslant 3$.

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:

$$E_{\text{corr}} = E_{\text{exact}} - E_{\text{HF}} \tag{5.1}$$

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
- Диелектронних 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 (Ha)

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. Концепція одно- та багатоконфігураційних методів

Одноконфігураційні методи Базуються на одному детермінанті Слейтера (HF) і додають кореляцію через збудження:

У найпростішому випадку хвильова функція є одним детермінантом:

$$|\Psi_{\mathrm{SR}}\rangle = |\Phi_0\rangle = |\varphi_1\varphi_2 \dots \varphi_N|. \tag{5.2}$$

Далі на нього накладають кореляційні поправки — не у вигляді нових детермінантів у самій хвильовій функції, а через операторні або збурювальні члени у виразі для енергії. Це дає «однореференсні» методи:

$$E = E_{\mathrm{HF}} + E_{\mathrm{corr}},$$

де E_{corr} визначається з теорії збурень (MP2, MP3...), або з експоненційного кластерного розкладу

$$|\Psi_{\mathrm{CC}}\rangle = e^{\hat{T}}|\Phi_0\rangle, \quad \hat{T} = \hat{T}_1 + \hat{T}_2 + \dots,$$

у методі CCSD тощо.

До однодетермінантних методів належать наступні:

- MP2, MP3, MP4 — теорія збурень
- CCSD, CCSD(T) — coupled cluster
- CISD, QCISD — configuration interaction

Добре працюють для основних станів, де один детермінант домінує у хвильовій функції.

Багатоконфігураційні методи Коли один детермінант не здатен описати систему (наприклад, при розриві хімічного зв'язку), хвильова функція має вигляд:

$$|\Psi_{\mathrm{MR}}\rangle = \sum_I c_I |\Phi_I\rangle. \tag{5.3}$$

Кожен детермінант $|\Phi_I\rangle$ утворюється перестановкою заповнених і вільних орбіталей у певному активному просторі. Наприклад, у методі CASSCF(4,4) ми розглядаємо всі можливі розподіли 4 електронів по 4 активних орбіталях. Це породжує набір детермінантів:

$$\{|\Phi_I\rangle\} = \{|(2s)^2(2p_x)^2|,\ |(2s)^1(2p_x)^1(2p_y)^2|,\ \dots\},$$

а хвильова функція є їхньою суперпозицією з коефіцієнтами c_I, які обчислюються варіаційно.

До багатоконфігураційних належать методи:

- CASSCF — вибір активного простору
- CASPT2 — додавання динамічної кореляції до CASSCF
- MRCI — multireference CI

Необхідні для: розриву зв'язків, збуджених станів, діелектронних систем.

code_3.py

```
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()
```

5.1.5. 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:

$$\hat{H} = \hat{H}_0 + \lambda \hat{V} \tag{5.4}$$

де $\hat{H}_0$ — оператор Фока, а $\hat{V}$ — різниця між точним гамільтоніаном та HF.

Енергія розкладається в ряд за степенями λ:

$$E = E^{(0)} + E^{(1)} + E^{(2)} + E^{(3)} + \ldots \tag{5.5}$$

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):

$$E^{(2)} = -\sum_{i<j}^{\text{occ}} \sum_{a<b}^{\text{virt}} \frac{|\langle ij||ab\rangle|^2}{\varepsilon_i + \varepsilon_j - \varepsilon_a - \varepsilon_b} \tag{5.6}$$

де:
- i, j — заповнені орбіталі
- a, b — віртуальні орбіталі
- $\langle ij||ab\rangle$ — антисиметризовані двоелектронні інтеграли
- ε — орбітальні енергії

5.2.3. MP2 розрахунок атома Неону

code_4.py

```
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")
    mymp2 = mp.MP2(mf)
    mymp2.verbose = 4
    e_mp2_corr, t2 = mymp2.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}% 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} mHa")
    print(f"  MP2 похибка: {error_mp2:8.4f} mHa")
    print(f"  Покращення: {abs(error_hf - error_mp2):8.4f} mHa")
    print(f"  MP2 відновлює {(1 - error_mp2 / error_hf) * 100:.1f}% 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:

code_5.py

```
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"<S²> (UHF): {s2_hf:.6f} (очікується {expected_s2:.6f})")

    # UMP2
    mymp2 = mp.UMP2(mf)
    mymp2.verbose = 0
    e_mp2_corr, t2 = mymp2.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

code_6.py

```
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 (mHa)", 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:

code_7.py

```
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} {'E_corr, mHa':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} {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

5.3. Coupled Cluster методи (CCSD, CCSD(T))

5.3.1. Теоретичні основи Coupled Cluster

Coupled Cluster (CC) — один з найточніших методів для врахування електронної кореляції. На відміну від CI, CC є size-extensive (правильно

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

масштабується для великих систем).

Хвильова функція у CC записується через експоненціальний оператор збудження:

$$|\Psi_{CC}\rangle = e^{\hat{T}}|\Phi_0\rangle \tag{5.7}$$

де $|\Phi_0\rangle$ — HF детермінант, а $\hat{T}$ — оператор кластерів:

$$\hat{T} = \hat{T}_1 + \hat{T}_2 + \hat{T}_3 + \ldots \tag{5.8}$$

CCSD (Coupled Cluster Singles and Doubles) Враховує тільки $\hat{T}_1$ (одиночні збудження) та $\hat{T}_2$ (подвійні збудження):

$$\hat{T}_1 = \sum_{i}^{\text{occ}} \sum_{a}^{\text{virt}} t_i^a \, \hat{a}_a^\dagger \hat{a}_i \tag{5.9}$$

$$\hat{T}_2 = \frac{1}{4} \sum_{ij}^{\text{occ}} \sum_{ab}^{\text{virt}} t_{ij}^{ab} \, \hat{a}_a^\dagger \hat{a}_b^\dagger \hat{a}_j \hat{a}_i \tag{5.10}$$

CCSD(T) — «золотий стандарт» Додає пертурбативну корекцію для потрійних збуджень $\hat{T}_3$:

$$E_{CCSD(T)} = E_{CCSD} + E_{(T)} \tag{5.11}$$

де $E_{(T)}$ обчислюється за формулами четвертого порядку теорії збурень.

5.3.2. CCSD розрахунок атома Гелію

code_8.py

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


def ccsd_helium_detailed(basis="cc-pvqz"):
    """
    Детальний CCSD розрахунок для He
    """

    mol = gto.M(atom="He 0 0 0", basis=basis, spin=0, verbose=0)
```

```
print(f"\nCCSD розрахунок атома Гелію (базис: {basis})")
print("=" * 70)

# Крок 1: HF
print("\nКрок 1: Hartree-Fock")
mf = scf.RHF(mol)
mf.verbose = 4
mf.conv_tol = 1e-12
e_hf = mf.kernel()

print(f"\nHF енергія: {e_hf:.12f} Ha")

# Крок 2: CCSD
print("\nКрок 2: CCSD розрахунок")
mycc = cc.CCSD(mf)
mycc.verbose = 4
mycc.conv_tol = 1e-10

e_ccsd_corr, t1, t2 = mycc.kernel()
e_ccsd = e_hf + e_ccsd_corr

print(f"\nCCSD кореляція: {e_ccsd_corr:.12f} Ha")
print(f"CCSD повна енергія: {e_ccsd:.12f} Ha")

# Аналіз амплітуд
print(f"\nАналіз амплітуд:")
print(f"  T1 амплітуди: {t1.shape}")
print(f"  T2 амплітуди: {t2.shape}")
print(f"  |T1|_max = {np.max(np.abs(t1)):.6f}")
print(f"  |T2|_max = {np.max(np.abs(t2)):.6f}")
print(f"  ||T1||_2 = {np.linalg.norm(t1):.6f}")
print(f"  ||T2||_2 = {np.linalg.norm(t2):.6f}")

# Для He T1 має бути дуже малим (симетрія)
if np.max(np.abs(t1)) < 1e-6:
    print("  T1 ≈ 0 (як і очікувалось для замкненої оболонки)")

# Крок 3: CCSD(T)
print("\nКрок 3: (T) корекція")
e_t = mycc.ccsd_t()
e_ccsdt = e_ccsd + e_t

print(f"\n(T) корекція: {e_t:.12f} Ha")
print(f"CCSD(T) повна енергія: {e_ccsdt:.12f} Ha")

# Порівняння з експериментом
e_exp = -2.903724377  # Ha (високоточне значення)

print(f"\nПорівняння з експериментом:")
print(f"Експеримент: {e_exp:.12f} Ha")
print(f"HF похибка:     {(e_hf - e_exp) * 1000:10.6f} mHa")
print(f"CCSD похибка:   {(e_ccsd - e_exp) * 1000:10.6f} mHa")
print(f"CCSD(T) похибка:{(e_ccsdt - e_exp) * 1000:10.6f} mHa")

# Розбиття кореляції
print(f"\nРозбиття кореляційної енергії:")
total_corr = e_ccsdt - e_hf
print(f"Повна кореляція: {total_corr * 1000:.6f} mHa (100%)")
print(
    f"CCSD внесок:     {e_ccsd_corr * 1000:.6f} mHa "
    f"({abs(e_ccsd_corr / total_corr) * 100:.2f}%)"
)
print(f"(T) внесок:      {e_t * 1000:.6f} mHa ({abs(e_t / total_corr) * 100:.2f}%)")
```

```
    return e_hf, e_ccsd, e_ccsdt


e_hf_he, e_ccsd_he, e_ccsdt_he = ccsd_helium_detailed()
```

5.3.3. CCSD для відкритих оболонок (UCCSD)

code_9.py

```
from pyscf import gto, scf, cc


def uccsd_calculation(symbol, spin, basis="cc-pvtz"):
    """
    UCCSD розрахунок для відкритої оболонки
    """

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

    print(f"\nUCCSD розрахунок {symbol} (2S={spin}, базис: {basis})")
    print("=" * 70)

    # UHF
    mf = scf.UHF(mol)
    mf.verbose = 0
    mf.conv_tol = 1e-11
    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
    spin_cont = s2_hf - expected_s2
    print(f"<S²> (UHF): {s2_hf:.6f} (очікується {expected_s2:.6f})")
    print(f"Забруднення спіном: {spin_cont:.6f}")

    # UCCSD
    print("\nUCCSD розрахунок...")
    mycc = cc.UCCSD(mf)
    mycc.verbose = 4
    mycc.conv_tol = 1e-9

    e_ccsd_corr, t1, t2 = mycc.kernel()
    e_ccsd = e_hf + e_ccsd_corr

    print(f"\nUCCSD кореляція: {e_ccsd_corr:.10f} Ha")
    print(f"UCCSD повна енергія: {e_ccsd:.10f} Ha")

    # CCSD виправляє забруднення спіном
    # (але <S²> все ще не є точним оператором)

    # Аналіз T1 діагностики
    t1_alpha, t1_beta = t1
    t1_diag = np.linalg.norm(t1_alpha) / np.sqrt(mol.nelec[0])

    print(f"\nT1 діагностика: {t1_diag:.6f}")
    if t1_diag < 0.02:
        print("  T1 < 0.02: одноконфігураційний характер")
    elif t1_diag < 0.05:
        print("  0.02 < T1 < 0.05: слабка багатоконфігураційність")
    else:
        print("  T1 > 0.05: сильна багатоконфігураційність!")
        print("  Можливо потрібен CASSCF/MRCI")
```

```
    # UCCSD(T)
    if mol.nelectron <= 10:  # (T) дуже повільно
        print("\n(T) корекція...")
        e_t = mycc.ccsd_t()
        e_ccsdt = e_ccsd + e_t

        print(f"(T) корекція: {e_t:.10f} Ha")
        print(f"UCCSD(T) повна енергія: {e_ccsdt:.10f} Ha")

        return e_hf, e_ccsd, e_ccsdt
    else:
        return e_hf, e_ccsd, None


# Приклади
uccsd_calculation("Li", spin=1, basis="cc-pvdz")
uccsd_calculation("C", spin=2, basis="cc-pvdz")
uccsd_calculation("N", spin=3, basis="cc-pvdz")
```

5.3.4. Порівняння ієрархії методів

code_10.py

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


def method_hierarchy_comparison(symbol, spin, basis="cc-pvtz"):
    """
    Порівняння HF → MP2 → CCSD → CCSD(T) → FCI
    """

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

    # Перевірка розміру (FCI експоненційно зростає)
    if mol.nelectron > 10:
        print(f"Атом {symbol} занадто великий для FCI")
        return

    print(f"\nІєрархія методів для {symbol} (базис: {basis})")
    print("=" * 80)

    methods = {}

    # HF
    print("HF розрахунок...")
    if spin == 0:
        mf = scf.RHF(mol)
    else:
        mf = scf.UHF(mol)

    mf.verbose = 0
    mf.conv_tol = 1e-11
    e_hf = mf.kernel()
    methods["HF"] = e_hf

    # MP2
    print("MP2 розрахунок...")
    if spin == 0:
        mymp2 = mp.MP2(mf)
    else:
        mymp2 = mp.UMP2(mf)
```

```
mymp2.verbose = 0
e_mp2_corr, _ = mymp2.kernel()
methods["MP2"] = e_hf + e_mp2_corr

# CCSD
print("CCSD розрахунок...")
if spin == 0:
    mycc = cc.CCSD(mf)
else:
    mycc = cc.UCCSD(mf)

mycc.verbose = 0
mycc.conv_tol = 1e-10
e_ccsd_corr, _, _ = mycc.kernel()
methods["CCSD"] = e_hf + e_ccsd_corr

# CCSD(T)
print("CCSD(T) розрахунок...")
e_t = mycc.ccsd_t()
methods["CCSD(T)"] = methods["CCSD"] + e_t

# FCI (точний у даному базисі)
print("FCI розрахунок...")
if spin == 0:
    myfci = fci.FCI(mf)
else:
    myfci = fci.FCI(mf)

myfci.verbose = 0
e_fci = myfci.kernel()[0]
methods["FCI"] = e_fci

# Результати
print("\n" + "=" * 80)
print(
    f"{'Метод':12s} {'Енергія, Ha':18s} {'Відносно HF, mHa':20s} "
    f"{'% FCI кореляції':18s}"
)
print("-" * 80)

e_corr_fci = e_fci - e_hf

for method in ["HF", "MP2", "CCSD", "CCSD(T)", "FCI"]:
    e = methods[method]
    rel = (e - e_hf) * 1000

    if method == "HF":
        percent = 0.0
    else:
        percent = abs((e - e_hf) / e_corr_fci) * 100

    print(f"{method:12s} {e:18.10f} {rel:20.6f} {percent:18.2f}")

print("=" * 80)

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

# Абсолютні енергії
method_names = list(methods.keys())
energies = [methods[m] for m in method_names]
colors = ["blue", "orange", "green", "red", "purple"]

x = np.arange(len(method_names))
bars = ax1.bar(x, energies, color=colors, alpha=0.7)
```

```
    ax1.set_xticks(x)
    ax1.set_xticklabels(method_names)
    ax1.set_ylabel("Енергія (Ha)", fontsize=12)
    ax1.set_title(f"Повні енергії {symbol}", fontsize=14)
    ax1.grid(True, alpha=0.3, axis="y")

    # Значення на стовпчиках
    for bar, e in zip(bars, energies):
        height = bar.get_height()
        ax1.text(
            bar.get_x() + bar.get_width() / 2.0,
            height,
            f"{e:.6f}",
            ha="center",
            va="bottom",
            fontsize=9,
        )

    # Кореляційна енергія (відносно HF)
    corr_energies = [(methods[m] - e_hf) * 1000 for m in method_names[1:]]

    ax2.bar(range(len(corr_energies)), corr_energies, color=colors[1:], alpha=0.7)
    ax2.set_xticks(range(len(corr_energies)))
    ax2.set_xticklabels(method_names[1:])
    ax2.set_ylabel("Кореляційна енергія (mHa)", fontsize=12)
    ax2.set_title(f"Кореляційні енергії {symbol}", fontsize=14)
    ax2.grid(True, alpha=0.3, axis="y")

    plt.tight_layout()
    plt.savefig(f"{symbol}_method_hierarchy.pdf")
    plt.show()

    return methods


# Приклади (тільки для малих атомів через FCI)
methods_he = method_hierarchy_comparison("He", spin=0, basis="cc-pvdz")
methods_be = method_hierarchy_comparison("Be", spin=0, basis="cc-pvdz")
methods_c = method_hierarchy_comparison("C", spin=2, basis="cc-pvdz")
```

5.3.5. T1 діагностика та багатоконфігураційний характер

T1 діагностика — індикатор того, наскільки система є одноконфігураційною:

$$T_1 = \frac{\|t_1\|}{\sqrt{N_{\text{elec}}}} \tag{5.12}$$

code_11.py

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


def t1_diagnostic_survey(atoms_list, basis="cc-pvdz"):
    """
    T1 діагностика для серії атомів
    """

    print(f"\nT1 діагностика (базис: {basis})")
    print("=" * 70)
    print(f"{'Атом':6s} {'Спін':4s} {'T1 діаг.':12s} {'Оцінка':30s}")
```

```
    print("-" * 70)

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

        # Пропускаємо дуже великі атоми
        if mol.nelectron > 18:
            print(f"{symbol:6s} {spin:4d} (пропущено - занадто великий)")
            continue

        # HF
        if spin == 0:
            mf = scf.RHF(mol)
            mycc = cc.CCSD(mf)
        else:
            mf = scf.UHF(mol)
            mycc = cc.UCCSD(mf)

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

        # CCSD
        mycc.verbose = 0
        mycc.conv_tol = 1e-8

        try:
            e_corr, t1, t2 = mycc.kernel()

            # Обчислення T1 діагностики
            if spin == 0:
                t1_norm = np.linalg.norm(t1)
                n_elec = mol.nelectron
            else:
                t1_alpha, t1_beta = t1
                t1_norm = np.linalg.norm(t1_alpha)
                n_elec = mol.nelec[0]

            t1_diag = t1_norm / np.sqrt(n_elec)

            # Інтерпретація
            if t1_diag < 0.02:
                assessment = "Добре (одноконфігураційний)"
            elif t1_diag < 0.05:
                assessment = "Прийнятно (слабка МК)"
            else:
                assessment = "Погано (потрібен CASSCF)"

            print(f"{symbol:6s} {spin:4d} {t1_diag:12.6f} {assessment:30s}")

        except:
            print(f"{symbol:6s} {spin:4d} {'ПОМИЛКА':12s} {'Не конвергувало':30s}")

    print("=" * 70)
    print("\nЛегенда:")
    print("  T1 < 0.02: одноконфігураційний (CCSD надійний)")
    print("  0.02 < T1 < 0.05: слабка багатоконфігураційність")
    print("  T1 > 0.05: сильна МК (краще CASSCF/MRCI)")


# Тестування різних атомів
atoms_test = [
    ("He", 0),
    ("Be", 0),
    ("C", 2),
```

```
    ("N", 3),
    ("O", 2),
    ("F", 1),
    ("Ne", 0),
    ("Mg", 0),
]

t1_diagnostic_survey(atoms_test)
```

5.3.6. Обчислювальна складність та масштабування

Таблиця 5.3. Порівняння обчислювальної складності методів

Метод	Масштабування	Пам'ять	Disk
HF	$\mathcal{O}(N^4)$	$\mathcal{O}(N^2)$	Мало
MP2	$\mathcal{O}(N^5)$	$\mathcal{O}(N^4)$	$\mathcal{O}(N^4)$
CCSD	$\mathcal{O}(N^6)$	$\mathcal{O}(N^4)$	$\mathcal{O}(N^4)$
CCSD(T)	$\mathcal{O}(N^7)$	$\mathcal{O}(N^4)$	$\mathcal{O}(N^4)$
CCSDT	$\mathcal{O}(N^8)$	$\mathcal{O}(N^6)$	$\mathcal{O}(N^6)$
FCI	$\mathcal{O}(e^N)$	$\mathcal{O}(e^N)$	$\mathcal{O}(e^N)$

де N — характерний розмір системи (базисних функцій або електронів).

5.4. Configuration Interaction (CI) та Full CI

5.4.1. Основи Configuration Interaction

Configuration Interaction (CI) — варіаційний метод, де хвильова функція представлена як лінійна комбінація детермінантів Слейтера:

$$|\Psi_{CI}\rangle = c_0|\Phi_0\rangle + \sum_{i}^{\text{occ}}\sum_{a}^{\text{virt}} c_i^a|\Phi_i^a\rangle + \sum_{i<j}\sum_{a<b} c_{ij}^{ab}|\Phi_{ij}^{ab}\rangle + \ldots \tag{5.13}$$

де:

- $|\Phi_0\rangle$ — HF детермінант (основна конфігурація)
- $|\Phi_i^a\rangle$ — одиночно збуджені детермінанти (S)
- $|\Phi_{ij}^{ab}\rangle$ — подвійно збуджені детермінанти (D)
- $|\Phi_{ijk}^{abc}\rangle$ — потрійно збуджені детермінанти (T)
- і т.д.

Варіанти CI методу:

- **CIS** (CI Singles) — тільки одиночні збудження (для збуджених станів)
- **CISD** (CI Singles and Doubles) — S + D

- **CISDT** — S + D + T
- **Full CI** — всі можливі збудження (точний у межах базису)

5.4.2. Full CI — точний розв'язок

Full CI включає всі можливі детермінанти і дає точну хвильову функцію та енергію в межах обраного базису:

$$E_{FCI} = \min_{\{c_i\}} \langle \Psi_{FCI} | \hat{H} | \Psi_{FCI} \rangle \tag{5.14}$$

Проблема: Кількість детермінантів зростає як:

$$N_{det} = \binom{N_{orb}}{N_\alpha} \times \binom{N_{orb}}{N_\beta} \tag{5.15}$$

Для 10 електронів у 20 орбіталях: $N_{det} \approx 10^{10}$ детермінантів!

FullCI.py

```python
from pyscf import gto, scf, fci, ci
import numpy as np

def fci_calculation_detailed(symbol, spin, basis='cc-pvdz'):
    """
    Детальний Full CI розрахунок
    """

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

    # Перевірка розміру
    n_det_approx = np.math.factorial(mol.nao_nr()) / (
        np.math.factorial(mol.nelec[0]) *
        np.math.factorial(mol.nao_nr() - mol.nelec[0])
    )

    if n_det_approx > 1e8:
        print(f'Система занадто велика для FCI!')
        print(f'Приблизна кількість детермінантів: {n_det_approx:.2e}')
        return

    print(f'\nFull CI розрахунок {symbol} (базис: {basis})')
    print('='*70)

    print(f'Розмір задачі:')
    print(f'  Електронів: {mol.nelectron}')
    print(f'  Базисних функцій: {mol.nao_nr()}')
    print(f'  α-електронів: {mol.nelec[0]}')
    print(f'  β-електронів: {mol.nelec[1]}')
    print(f'  Приблизна кількість детермінантів: {n_det_approx:.2e}')

    # HF розрахунок
    print('\nHF розрахунок...')
    if spin == 0:
        mf = scf.RHF(mol)
```

```
    else:
        mf = scf.UHF(mol)

    mf.verbose = 0
    mf.conv_tol = 1e-12
    e_hf = mf.kernel()

    print(f'HF енергія: {e_hf:.12f} Ha')

    # FCI розрахунок
    print('\nFull CI розрахунок...')
    myfci = fci.FCI(mf)
    myfci.verbose = 4

    e_fci, ci_vec = myfci.kernel()

    print(f'\nFCI енергія: {e_fci:.12f} Ha')
    print(f'Кореляційна енергія: {(e_fci - e_hf)*1000:.8f} mHa')

    # Аналіз хвильової функції
    print(f'\nАналіз хвильової функції:')

    # Вага HF конфігурації
    if spin == 0:
        # Для RHF: HF конфігурація є першим елементом
        c0_squared = ci_vec[0]**2
        print(f'Вага HF конфігурації: {c0_squared:.6f} '
              f'({c0_squared*100:.2f}%)')

    # Ентропія CI вектора (міра багатоконфігураційності)
    ci_vec_flat = ci_vec.ravel()
    ci_squared = ci_vec_flat**2
    ci_squared = ci_squared[ci_squared > 1e-10]  # відкидаємо дуже малі

    entropy = -np.sum(ci_squared * np.log(ci_squared))
    print(f'Ентропія CI вектора: {entropy:.6f}')

    if entropy < 0.1:
        print('  → Сильно одноконфігураційна система')
    elif entropy < 1.0:
        print('  → Помірна багатоконфігураційність')
    else:
        print('  → Сильна багатоконфігураційність')

    # Найважливіші конфігурації
    n_important = np.sum(ci_squared > 0.01)
    print(f'Конфігурацій з вагою > 1%: {n_important}')

    return e_hf, e_fci

# Приклади (тільки малі системи!)
fci_calculation_detailed('He', spin=0, basis='cc-pvdz')
fci_calculation_detailed('Be', spin=0, basis='cc-pvdz')
fci_calculation_detailed('Li', spin=1, basis='6-31g')
```

5.4.3. CISD розрахунки

CISD — практичний варіант CI, але не є size-extensive:

CISDcalc.py

```
from pyscf import gto, scf, ci

def cisd_calculation(symbol, spin, basis='cc-pvtz'):
```

```
    """
    CISD розрахунок
    """

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

    print(f'\nCISD розрахунок {symbol} (базис: {basis})')
    print('='*70)

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

    mf.verbose = 0
    mf.conv_tol = 1e-11
    e_hf = mf.kernel()

    print(f'HF енергія: {e_hf:.10f} Ha')

    # CISD
    print('\nCISD розрахунок...')
    if spin == 0:
        myci = ci.CISD(mf)
    else:
        myci = ci.UCISD(mf)

    myci.verbose = 4
    e_cisd, civec = myci.kernel()

    print(f'\nCISD енергія: {e_cisd:.10f} Ha')
    print(f'CISD кореляція: {(e_cisd - e_hf)*1000:.6f} mHa')

    return e_hf, e_cisd

cisd_calculation('C', spin=2)
cisd_calculation('Ne', spin=0)
```

5.4.4. Порівняння CI та CC методів

CIvsCD.py

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

def compare_ci_cc(symbol='Be', spin=0, basis='cc-pvdz'):
    """
    Порівняння CI та CC методів
    """

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

    # Перевірка розміру для FCI
```

```
if mol.nelectron > 10:
    print('Атом занадто великий для FCI')
    include_fci = False
else:
    include_fci = True

print(f'\nПорівняння методів для {symbol} (базис: {basis})')
print('='*70)

results = {}

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

mf.verbose = 0
mf.conv_tol = 1e-11
e_hf = mf.kernel()
results['HF'] = e_hf

print(f'HF: {e_hf:.10f} Ha')

# CISD
print('CISD...')
if spin == 0:
    myci = ci.CISD(mf)
else:
    myci = ci.UCISD(mf)

myci.verbose = 0
e_cisd, _ = myci.kernel()
results['CISD'] = e_cisd

print(f'CISD: {e_cisd:.10f} Ha')

# CCSD
print('CCSD...')
if spin == 0:
    mycc = cc.CCSD(mf)
else:
    mycc = cc.UCCSD(mf)

mycc.verbose = 0
e_ccsd_corr, _, _ = mycc.kernel()
e_ccsd = e_hf + e_ccsd_corr
results['CCSD'] = e_ccsd

print(f'CCSD: {e_ccsd:.10f} Ha')

# CCSD(T)
print('CCSD(T)...')
e_t = mycc.ccsd_t()
e_ccsdt = e_ccsd + e_t
results['CCSD(T)'] = e_ccsdt

print(f'CCSD(T): {e_ccsdt:.10f} Ha')

# FCI
if include_fci:
    print('FCI...')
    myfci = fci.FCI(mf)
    myfci.verbose = 0
    e_fci, _ = myfci.kernel()
```

```
        results['FCI'] = e_fci

        print(f'FCI: {e_fci:.10f} Ha')

    # Порівняння
    print('\n' + '='*70)
    print(f'{"Метод":12s} {"Енергія, Ha":18s} {"Кореляція, mHa":18s}')
    print('-'*70)

    for method in results.keys():
        e = results[method]
        corr = (e - e_hf) * 1000
        print(f'{method:12s} {e:18.10f} {corr:18.6f}')

    print('='*70)

    # Аналіз
    if include_fci:
        e_corr_fci = results['FCI'] - e_hf

        print('\nВідсоток відновленої кореляції (відносно FCI):')
        for method in ['CISD', 'CCSD', 'CCSD(T)']:
            e_corr = results[method] - e_hf
            percent = abs(e_corr / e_corr_fci) * 100
            print(f'{method:12s}: {percent:6.2f}%')

        # Графік
        fig, ax = plt.subplots(figsize=(10, 6))

        methods_list = list(results.keys())
        energies = [results[m] for m in methods_list]
        corr_energies = [(results[m] - e_hf) * 1000
                         for m in methods_list]

        x = np.arange(len(methods_list))
        colors = ['blue', 'orange', 'green', 'red', 'purple']

        bars = ax.bar(x, corr_energies, color=colors[:len(methods_list)],
                      alpha=0.7)
        ax.set_xticks(x)
        ax.set_xticklabels(methods_list)
        ax.set_ylabel('Кореляційна енергія (mHa)', fontsize=12)
        ax.set_title(f'Порівняння методів для {symbol}', fontsize=14)
        ax.grid(True, alpha=0.3, axis='y')

        # Додавання значень
        for bar, e in zip(bars, corr_energies):
            height = bar.get_height()
            ax.text(bar.get_x() + bar.get_width()/2., height,
                    f'{e:.4f}', ha='center', va='bottom', fontsize=10)

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

    return results

# Приклади
results_be = compare_ci_cc('Be', spin=0, basis='cc-pvdz')
results_c = compare_ci_cc('C', spin=2, basis='6-31g')
```

5.4.5. Size-extensivity проблема CI

Важлива відмінність між CI та CC:

$$E_{CI}(2A) \neq 2 \times E_{CI}(A) \tag{5.16}$$

$$E_{CC}(2A) = 2 \times E_{CC}(A) \tag{5.17}$$

SizeExtensivityExample.py

```
from pyscf import gto, scf, ci, cc

def demonstrate_size_extensivity():
    """
    Демонстрація size-extensivity
    """

    print('\nДемонстрація size-extensivity')
    print('='*70)

    # Один атом He
    mol_1he = gto.M(atom='He 0 0 0', basis='sto-3g', verbose=0)
    mf_1he = scf.RHF(mol_1he)
    mf_1he.verbose = 0
    e_hf_1he = mf_1he.kernel()

    # CISD для 1 He
    myci_1he = ci.CISD(mf_1he)
    myci_1he.verbose = 0
    e_cisd_1he, _ = myci_1he.kernel()

    # CCSD для 1 He
    mycc_1he = cc.CCSD(mf_1he)
    mycc_1he.verbose = 0
    e_ccsd_corr_1he, _, _ = mycc_1he.kernel()
    e_ccsd_1he = e_hf_1he + e_ccsd_corr_1he

    print(f'Один атом He:')
    print(f'  HF:   {e_hf_1he:.10f} Ha')
    print(f'  CISD: {e_cisd_1he:.10f} Ha')
    print(f'  CCSD: {e_ccsd_1he:.10f} Ha')

    # Два атоми He (далеко один від одного)
    mol_2he = gto.M(
        atom='He 0 0 0; He 0 0 100',  # 100 Bohr відстань
        basis='sto-3g',
        verbose=0
    )
    mf_2he = scf.RHF(mol_2he)
    mf_2he.verbose = 0
    e_hf_2he = mf_2he.kernel()

    # CISD для 2 He
    myci_2he = ci.CISD(mf_2he)
    myci_2he.verbose = 0
    e_cisd_2he, _ = myci_2he.kernel()

    # CCSD для 2 He
    mycc_2he = cc.CCSD(mf_2he)
    mycc_2he.verbose = 0
    e_ccsd_corr_2he, _, _ = mycc_2he.kernel()
    e_ccsd_2he = e_hf_2he + e_ccsd_corr_2he

    print(f'\nДва атоми He (відокремлені):')
    print(f'  HF:   {e_hf_2he:.10f} Ha')
    print(f'  CISD: {e_cisd_2he:.10f} Ha')
    print(f'  CCSD: {e_ccsd_2he:.10f} Ha')
```

```
    # Перевірка size-extensivity
    print(f'\nПеревірка size-extensivity:')
    print(f'  2×E(1 He):')
    print(f'    HF:   {2*e_hf_1he:.10f} Ha')
    print(f'    CISD: {2*e_cisd_1he:.10f} Ha')
    print(f'    CCSD: {2*e_ccsd_1he:.10f} Ha')

    print(f'\n  Похибка E(2 He) - 2×E(1 He):')
    err_hf = (e_hf_2he - 2*e_hf_1he) * 1000
    err_cisd = (e_cisd_2he - 2*e_cisd_1he) * 1000
    err_ccsd = (e_ccsd_2he - 2*e_ccsd_1he) * 1000

    print(f'    HF:   {err_hf:.6f} mHa (має бути ☒0)')
    print(f'    CISD: {err_cisd:.6f} mHa (не size-extensive!)')
    print(f'    CCSD: {err_ccsd:.6f} mHa (має бути ☒0)')

    print('\nВисновок: CCSD є size-extensive, CISD - ні')

demonstrate_size_extensivity()
```

5.4.6. Коли використовувати CI vs CC

Таблиця 5.4. Порівняння CI та CC методів

Критерій	**CI**	**CC**
Варіаційність	Так	Ні
Size-extensivity	Ні (окрім FCI)	Так
Точність	CISD $<$ CCSD	CCSD(T) ☒ FCI
Збуджені стани	EOM-CI, CIS	EOM-CC
Швидкість	CISD швидше CCSD	Повільніше
Стабільність	Варіаційна межа	Може розходитись
Багатоконфігураційність	Природно (FCI)	Потребує MRCC

Рекомендації:

- **Використовуйте FCI** для малих систем як еталон
- **Використовуйте CCSD(T)** для точних розрахунків одноконфігураційних систем
- **Використовуйте CISD** для швидких оцінок кореляції
- **Для багатоконфігураційних систем** використовуйте CASSCF/CASPT2

5.5. CASSCF — багатоконфігураційний метод

5.5.1. Ідея Complete Active Space Self-Consistent Field

CASSCF (Complete Active Space SCF) — багатоконфігураційний метод, який поєднує:

- **Активний простір** — набір орбіталей, де проводиться Full CI

- **SCF оптимізацію** — орбіталі та CI коефіцієнти оптимізуються одночасно

Орбіталі розділяються на три групи:

1. **Core (остов)** — завжди подвійно заповнені, не беруть участь у кореляції
2. **Active (активні)** — електрони розподілені по всіх можливих способах (Full CI)
3. **Virtual (віртуальні)** — завжди порожні

$$|\Psi_{CASSCF}\rangle = \sum_I c_I |\Phi_I^{\text{active}}\rangle \otimes |\text{core}\rangle \tag{5.18}$$

5.5.2. Позначення CASSCF(n,m)

CASSCF(n,m) означає:

- n — кількість електронів в активному просторі
- m — кількість орбіталей в активному просторі

Приклади:

- **Be**: CASSCF(4,8) — 4 валентні електрони у 8 орбіталях (2s, 2p, 3s, 3p)
- **C**: CASSCF(4,4) — 4 валентні електрони у 4 орбіталях (2s, 2p)
- **Cr**: CASSCF(6,5) — 6 електронів у 5 d-орбіталях

5.5.3. CASSCF розрахунок атома Берилію

Берилій має близькі за енергією конфігурації $2s^2$ та $2p^2$, що робить його класичним прикладом для CASSCF:

CASSCFBe.py

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


def casscf_beryllium_detailed(basis="cc-pvtz"):
    """
    Детальний CASSCF розрахунок для Be
    """

    mol = gto.M(atom="Be 0 0 0", basis=basis, spin=0, symmetry=True, verbose=0)

    print(f"\nCAS розрахунок атома Берилію (базис: {basis})")
    print("=" * 70)

    # Крок 1: HF розрахунок
    print("Крок 1: RHF розрахунок")
    mf = scf.RHF(mol)
    mf.verbose = 4
    mf.conv_tol = 1e-12
    e_hf = mf.kernel()

    print(f"\nRHF енергія: {e_hf:.10f} Ha")
    print(f"Електронна конфігурація HF: 1s² 2s²")

    # Крок 2: Вибір активного простору
    print("\nКрок 2: Визначення активного простору")
    print("Активний простір: CASSCF(4,8)")
```

```
    print("  4 електрони: валентні 2s² 2p□")
    print("  8 орбіталей: 2s, 2p (3 орб), 3s, 3p (3 орб)")

    # CASSCF(4,8): 4 електрони у 8 орбіталях
    ncas = 8  # активних орбіталей
    nelecas = 4  # активних електронів

    mc = mcscf.CASSCF(mf, ncas, nelecas)
    mc.verbose = 4
    mc.conv_tol = 1e-10

    # Крок 3: CASSCF розрахунок
    print("\nКрок 3: CASSCF оптимізація")
    e_casscf = mc.kernel()[0]

    print(f"\nCASCF(4,8) енергія: {e_casscf:.10f} Ha")
    print(f"Статична кореляція: {(e_casscf - e_hf) * 1000:.6f} mHa")

    # Аналіз хвильової функції
    print("\nКрок 4: Аналіз хвильової функції")

    # Природні орбіталі та заселеності
    natocc, natorb = mc.cas_natorb()

    print("\nЗаселеності природних орбіталей (активний простір):")
    for i, occ in enumerate(natocc):
        print(f"  Орбіталь {i + 1}: {occ:.6f}")

    # Аналіз конфігурацій
    print("\nАналіз CI коефіцієнтів:")

    # Для детального аналізу можна використати fcisolver
    ci = mc.ci

    # Вага основної конфігурації
    # (потребує додаткового аналізу CI вектора)

    # Entanglement entropy
    s_entropy = 0
    for occ in natocc:
        if occ > 1e-10 and occ < 2 - 1e-10:
            s_entropy += -occ / 2 * np.log(occ / 2) - (2 - occ) / 2 * np.log(
                (2 - occ) / 2
            )

    print(f"\nОднопартичкова ентропія: {s_entropy:.6f}")
    if s_entropy < 0.5:
        print("  → Слабка статична кореляція")
    elif s_entropy < 1.5:
        print("  → Помірна статична кореляція")
    else:
        print("  → Сильна статична кореляція")

    return e_hf, e_casscf, natocc


e_hf_be, e_cas_be, occ_be = casscf_beryllium_detailed()
```

5.5.4. CASSCF для перехідних металів

Для перехідних металів d-орбіталі часто близькі за енергією, потребуючи багатоконфігураційного підходу:

CASSCFTransitionMetals.py

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


def casscf_transition_metal(symbol, spin, nelecas, ncas, basis="def2-svp"):
    """
    CASSCF для перехідного металу
    """

    mol = gto.M(
        atom=f"{symbol} 0 0 0",
        basis=basis,
        spin=spin,
        symmetry=False,  # Часто простіше без симетрії
        verbose=0,
    )

    print(f"\nCASCF розрахунок {symbol} (2S={spin})")
    print(f"Активний простір: CASSCF({nelecas},{ncas})")
    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")

    s2_hf = mf.spin_square()[0]
    expected_s2 = spin * (spin + 2) / 4
    print(f"<S²> (UHF): {s2_hf:.6f} (очікується {expected_s2:.6f})")

    # CASSCF
    print(f"\nCASCF({nelecas},{ncas}) розрахунок...")
    mc = mcscf.CASSCF(mf, ncas, nelecas)
    mc.verbose = 4
    mc.conv_tol = 1e-8
    mc.max_cycle_macro = 100

    # Для важких випадків
    mc.fcisolver.max_cycle = 100
    mc.fcisolver.conv_tol = 1e-8

    try:
        e_casscf = mc.kernel()[0]

        print(f"\nCASCF енергія: {e_casscf:.10f} Ha")
        print(f"Статична кореляція: {(e_casscf - e_hf) * 1000:.6f} mHa")

        # Природні орбіталі
        natocc, natorb = mc.cas_natorb()

        print("\nЗаселеності природних орбіталей (активний простір):")
        for i, occ in enumerate(natocc):
            if occ > 0.01:  # тільки значні заселеності
                print(f"  Орбіталь {i + 1}: {occ:.4f}")

        # Оцінка багатоконфігураційності
        # Strongly occupied (> 1.98) та weakly occupied (< 0.02)
        n_strong = np.sum(natocc > 1.98)
        n_weak = np.sum(natocc < 0.02)
        n_fractional = ncas - n_strong - n_weak
```

```
        print(f"\nАналіз заселеностей:")
        print(f"  Сильно заповнені (>1.98): {n_strong}")
        print(f"  Дробові (0.02-1.98): {n_fractional}")
        print(f"  Порожні (<0.02): {n_weak}")

        if n_fractional > ncas / 2:
            print("  → Сильна багатоконфігураційність!")

        return e_hf, e_casscf, natocc

    except Exception as e:
        print(f"\nПомилка: {str(e)}")
        print("Спробуйте інший активний простір або початкове наближення")
        return e_hf, None, None


# Приклади

# Cr: [Ar] 3d⁵ 4s¹
# CASSCF(6,6): 6 електронів у 6 орбіталях (5×3d + 1×4s)
e_hf_cr, e_cas_cr, occ_cr = casscf_transition_metal(
    "Cr", spin=6, nelecas=6, ncas=6, basis="def2-svp"
)

# Fe: [Ar] 3d⁶ 4s²
# CASSCF(8,5): 8 електронів у 5 d-орбіталях
e_hf_fe, e_cas_fe, occ_fe = casscf_transition_metal(
    "Fe", spin=4, nelecas=8, ncas=5, basis="def2-svp"
)
```

5.5.5. Вибір активного простору

Правильний вибір активного простору критичний для CASSCF:

Таблиця 5.5. Рекомендації по вибору активного простору

Система	Активний простір	Коментар
Be	(4,8): 2s,2p,3s,3p	Враховує $2s^2 \boxtimes 2p^2$
C	(4,4): 2s,2p	Мінімальний валентний
O	(6,6): 2s,2p + корел.	З кореляційними орбіталями
3d метали	(n,5): 3d	Тільки d-орбіталі
3d метали	(n+2,6): 3d,4s	d + s орбіталі
Лантаноїди	(n,7): 4f	f-орбіталі

Принципи вибору:

1. Включити орбіталі, близькі за енергією
2. Включити орбіталі з значною хімічною активністю
3. Більший простір → точніше, але експоненційно повільніше
4. Перевірити заселеності природних орбіталей
5. Якщо заселеності ⊠ 2 або ⊠ 0, можна виключити з активного простору

5.5.6. State-averaged CASSCF

Для розрахунку декількох станів одночасно (наприклад, різних мультиплетів):

StateAvaragedCASSCF.py

```python
from pyscf import gto, scf, mcscf


def sa_casscf_carbon():
    """
    State-averaged CASSCF для різних станів Карбону
    """

    mol = gto.M(
        atom="C 0 0 0",
        basis="cc-pvtz",
        spin=2,  # Для триплету
        symmetry=True,
        verbose=0,
    )

    print("\nState-Averaged CASSCF для C")
    print("Стани: ³P (основний) та ¹D (збуджений)")
    print("=" * 70)

    # HF
    mf = scf.UHF(mol)
    mf.verbose = 0
    e_hf = mf.kernel()

    print(f"UHF енергія: {e_hf:.10f} Ha")

    # SA-CASSCF(4,4): усереднення по декількох станах
    mc = mcscf.CASSCF(mf, 4, 4)
    mc.verbose = 4

    # State-averaging для 2 станів з рівними вагами
    mc.state_average_([0.5, 0.5])

    # Або для різних симетрій:
    # mc = mc.state_average_mix_([
    #     mcscf.state_average_(mc, [0.5, 0.5]),  # стани симетрії 1
    # ])

    mc.conv_tol = 1e-9
    e_sa = mc.kernel()[0]

    print(f"\nSA-CASSCF енергія (усереднена): {e_sa:.10f} Ha")

    # Енергії окремих станів
    print("\nЕнергії окремих станів:")
    for i, e_state in enumerate(mc.e_states):
        print(f"  Стан {i + 1}: {e_state:.10f} Ha ({e_state * 27.211386:.4f} eV)")

    # Різниця енергій
    if len(mc.e_states) > 1:
        delta_e = (mc.e_states[1] - mc.e_states[0]) * 27.211386
        print(f"\nРізниця енергій збудження: {delta_e:.4f} eV")


sa_casscf_carbon()
```

5.5.7. CASPT2 — динамічна кореляція поверх CASSCF

CASSCF враховує тільки статичну кореляцію. Для повної точності потрібно додати динамічну кореляцію через CASPT2:

CASPT2.py

```
from pyscf import gto, scf, mcscf, mrpt


def caspt2_calculation(symbol="Be", spin=0, nelecas=4, ncas=8, basis="cc-pvdz"):
    """
    CASSCF + CASPT2 розрахунок
    """

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

    print(f"\nCASPT2 розрахунок {symbol}")
    print(f"Активний простір: ({nelecas},{ncas})")
    print("=" * 70)

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

    mf.verbose = 0
    e_hf = mf.kernel()

    print(f"HF енергія: {e_hf:.10f} Ha")

    # CASSCF
    mc = mcscf.CASSCF(mf, ncas, nelecas)
    mc.verbose = 0
    mc.conv_tol = 1e-9
    e_casscf = mc.kernel()[0]

    print(f"CASSCF енергія: {e_casscf:.10f} Ha")
    print(f"Статична кореляція: {(e_casscf - e_hf) * 1000:.6f} mHa")

    # CASPT2
    print("\nCASPT2 розрахунок...")
    try:
        # В PySCF NEVPT2 більш стабільний ніж CASPT2
        from pyscf.mrpt import NEVPT

        nevpt2 = NEVPT(mc)
        e_corr_pt2 = nevpt2.kernel()
        e_caspt2 = e_casscf + e_corr_pt2

        print(f"NEVPT2 кореляція: {e_corr_pt2 * 1000:.6f} mHa")
        print(f"CASSCF+NEVPT2 енергія: {e_caspt2:.10f} Ha")

        # Розбиття кореляції
        total_corr = e_caspt2 - e_hf
        static_corr = e_casscf - e_hf
        dynamic_corr = e_corr_pt2

        print(f"\nРозбиття кореляційної енергії:")
        print(f"  Повна: {total_corr * 1000:.6f} mHa")
        print(
            f"  Статична (CASSCF): {static_corr * 1000:.6f} mHa "
            f"({abs(static_corr / total_corr) * 100:.1f}%)"
        )
```

```
        print(
            f"  Динамічна (NEVPT2): {dynamic_corr * 1000:.6f} mHa "
            f"({abs(dynamic_corr / total_corr) * 100:.1f}%)"
        )

        return e_hf, e_casscf, e_caspt2

    except Exception as e:
        print(f"CASPT2/NEVPT2 недоступний: {str(e)}")
        return e_hf, e_casscf, None


# Приклади
caspt2_calculation("Be", spin=0, nelecas=4, ncas=8)
caspt2_calculation("C", spin=2, nelecas=4, ncas=4)
```

5.5.8. Переваги та недоліки CASSCF

Переваги:

- Правильно описує статичну кореляцію
- Може розрахувати декілька станів одночасно
- Варіаційний метод
- Підходить для розриву зв'язків, збуджених станів
- Дає природні орбіталі та заселеності

Недоліки:

- Вибір активного простору не завжди очевидний
- Експоненційне масштабування з розміром активного простору
- Не враховує динамічну кореляцію (потрібен CASPT2)
- Може бути проблеми з конвергенцією
- Потребує досвіду для правильного використання

Коли використовувати CASSCF:

- T1 діагностика > 0.05 (сильна багатоконфігураційність)
- Перехідні метали з близькими d-орбіталями
- Збуджені стани атомів
- Діелектронні системи
- Відкрито-оболонкові синглети

5.5.9. 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)

5.5.10. Важливість кореляції для різних властивостей

Таблиця 5.6. 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

code_12.py

```
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}% 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)
```

5.5.11. Порівняння методів для He

Розглянемо найпростішу багатоелектронну систему — атом Гелію:

code_13.py

```
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} {'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()
```

5.6. Практичні завдання

5.6.1. Завдання 1: Порівняння методів для He

```
"""
ЗАВДАННЯ 1: Для атома Гелію розрахуйте енергії методами
HF, MP2, CCSD, CCSD(T) та FCI з базисами cc-pVDZ, cc-pVTZ,
cc-pVQZ.

1) Побудуйте графік збіжності кожного методу до базисної межі
2) Екстраполюйте до CBS за формулою E(X) = E_CBS + A/X³
3) Порівняйте з експериментальним значенням -2.90372 Ha
4) Яка частка кореляційної енергії відновлюється кожним методом?

Базиси: cc-pVDZ, cc-pVTZ, cc-pVQZ
"""

# Ваш код тут
```

Завдання 2: T1 діагностика для другого періоду

```
"""
ЗАВДАННЯ 2: Розрахуйте T1 діагностику для всіх атомів
другого періоду (Li-Ne) з базисом cc-pVDZ.

1) Для яких атомів T1 > 0.02 (багатоконфігураційні)?
2) Чи є кореляція між T1 та положенням у періоді?
3) Порівняйте T1 для основного та збудженого спінового
   стану Карбону (³P vs ¹D)
4) Побудуйте графік залежності T1 від атомного номера

Базис: cc-pVDZ
"""

# Ваш код тут
```

Завдання 3: Size-extensivity тест

```
"""
ЗАВДАННЯ 3: Дослідіть size-extensivity для атомів Ne.

Розрахуйте методами HF, MP2, CISD, CCSD:
1) Енергію одного атома Ne
2) Енергію двох атомів Ne на відстані 100 Bohr
3) Обчисліть похибку: E(2×Ne) - 2×E(Ne)

Питання:
- Який метод є size-extensive?
- Яка похибка для CISD (у mHa)?
- Чому MP2 є size-extensive, а CISD - ні?

Базис: cc-pVTZ
"""

# Ваш код тут
```

Завдання 4: CASSCF для перехідного металу

```
"""
ЗАВДАННЯ 4: CASSCF аналіз атома Титану (Ti).

Ti: [Ar] 3d² 4s², основний стан ³F

1) Розрахуйте UHF, CCSD та CASSCF(4,5) енергії
   (4 електрони у 5 d-орбіталях)
2) Проаналізуйте заселеності природних орбіталей
3) Обчисліть статичну кореляцію (CASSCF - HF)
4) Спробуйте різні активні простори: (4,5), (4,6), (6,11)
   Який найкращий?

Базис: def2-SVP
"""

# Ваш код тут
```

Завдання 5: Енергії збудження

```
"""
ЗАВДАННЯ 5: Розрахуйте енергію збудження ³P → ¹D для
атома Карбону.

Використайте методи:
1) ΔSCF (окремі розрахунки для кожного стану)
2) State-averaged CASSCF
3) EOM-CCSD (якщо доступно)

Порівняйте з експериментальним значенням 1.26 eV.

Який метод дає найточніший результат?

Базис: aug-cc-pVTZ
"""
```

```
# Ваш код тут
```

Завдання 6: Кореляційна енергія vs Z

```
"""
ЗАВДАННЯ 6: Дослідіть залежність кореляційної енергії
від атомного номера.

Розрахуйте MP2 кореляційну енергію для:
- Благородних газів: He, Ne, Ar, Kr
- У абсолютних величинах (mHa)
- У відсотках від повної енергії

Побудуйте графіки:
1) |E_corr| vs Z
2) |E_corr|/|E_HF| vs Z

Яка тенденція спостерігається?

Базис: cc-pVTZ
"""

# Ваш код тут
```

5.7. Резюме

У цьому розділі ми детально вивчили Post-Hartree-Fock методи для врахування електронної кореляції:

5.7.1. Основні методи та їх характеристики

Таблиця 5.7. Порівняння Post-HF методів

Метод	Складність	Size-ext.	Варіац.	Точність	МК?
MP2	$O(N^5)$	Так	Ні	Помірна	Ні
MP3/MP4	$O(N^6/N^7)$	Так	Ні	Добра	Ні
CISD	$O(N^6)$	Ні	Так	Помірна	Ні
CCSD	$O(N^6)$	Так	Ні	Добра	Ні
CCSD(T)	$O(N^7)$	Так	Ні	Відмінна	Ні
CASSCF	$O(e^{n_{act}})$	Так	Так	Статична	Так
CASPT2	$O(e^{n_{act}})$	Так	Ні	Відмінна	Так
Full CI	$O(e^N)$	Так	Так	Точна	Так

МК = Багатоконфігураційний

5.7.2. Ключові висновки

1. Електронна кореляція
- Становить 1-5% від повної енергії, але критична для точних розрахунків
- Поділяється на динамічну (короткодіючу) та статичну (близькі стани)
- HF і DFT не враховують кореляцію повністю

2. MP2 — найпростіший post-HF метод
- Швидкий ($O(N^5)$), size-extensive
- Добре працює для замкнених оболонок з великим HOMO-LUMO gap
- Відновлює 80-95% кореляційної енергії
- Не виправляє забруднення спіном UHF
- Потребує великих базисів для збіжності

3. CCSD(T) — "золотий стандарт"
- Найточніший одноконфігураційний метод
- Size-extensive, відновлює >99% кореляції
- Повільний ($O(N^7)$), але надійний
- Виправляє забруднення спіном
- Обмежений розміром системи (до 20-30 атомів)

4. T1 діагностика
- T1 < 0.02: одноконфігураційна система (CCSD надійний)
- 0.02 < T1 < 0.05: слабка багатоконфігураційність
- T1 > 0.05: потрібен CASSCF/MRCI

5. CI методи
- Варіаційні (енергія завжди вище точної)
- Full CI — точний у межах базису, але непрактичний
- CISD не є size-extensive (на відміну від CCSD)
- Корисні для аналізу хвильової функції

6. CASSCF — для багатоконфігураційних систем
- Необхідний коли T1 > 0.05 або близькі стани
- Вибір активного простору критичний
- Враховує статичну кореляцію
- Потребує CASPT2/NEVPT2 для динамічної кореляції
- Ідеальний для перехідних металів, збуджених станів

Таблиця 5.8. Вибір методу залежно від задачі

Задача	Рекомендований метод	Альтернатива
Швидка оцінка кореляції	MP2	DFT
Точні енергії (1 kcal/mol)	CCSD(T)	—
Замкнені оболонки	CCSD(T), MP2	DFT
Відкриті оболонки	UCCSD(T)	ROHF-CCSD
Енергії збудження	EOM-CCSD, CASSCF	TD-DFT
Перехідні метали	CASSCF, CASPT2	DFT+U
Багатоконфігураційні	CASSCF/CASPT2	MRCI
Еталонні розрахунки	CCSD(T)/CBS	Full CI
Великі системи	MP2, DFT	DLPNO-CCSD(T)

5.7.3. Рекомендації по використанню

5.7.4. Типові помилки

1. **Використання MP2 для систем з малим gap** — може давати нефізичні результати
2. **Забування перевірки T1 діагностики** — CCSD може бути ненадійним
3. **Занадто малий активний простір у CASSCF** — не врахує всю статичну кореляцію
4. **Занадто великий активний простір** — неможливо розрахувати
5. **Недостатній базис** — post-HF методи дуже чутливі до базису
6. **Ігнорування size-extensivity** — CISD не підходить для енергій дисоціації
7. **CASSCF без CASPT2** — відсутня динамічна кореляція

5.7.5. Практичні поради

1. Завжди починайте з HF розрахунку та перевірте конвергенцію
2. Для нових систем спочатку протестуйте на малому базисі
3. Перевіряйте T1 діагностику після CCSD
4. Для CASSCF: починайте з малого активного простору, збільшуйте поступово
5. Аналізуйте природні орбіталі для вибору активного простору
6. Використовуйте симетрію коли можливо
7. Для екстраполяції до CBS потрібні принаймні 3 базиси
8. Зберігайте проміжні результати (checkpoint файли)

Таблиця 5.9. Орієнтовний час розрахунку (відносно HF)

Метод	He (DZ)	Ne (TZ)	Ar (DZ)	Пам'ять
HF	1×	1×	1×	N^2
MP2	2×	5×	10×	N^4
CCSD	10×	50×	200×	N^4
CCSD(T)	20×	200×	1000×	N^4
CASSCF(4,8)	5×	—	—	$e^{n_{act}}$

5.7.6. Обчислювальні ресурси

Примітка: реальний час залежить від якості початкового наближення та конвергенції

5.7.7. Подальше вивчення

Для поглибленого розуміння post-HF методів рекомендуємо:

- **Книги:**
  - T. Helgaker et al., "Molecular Electronic-Structure Theory"
  - I. Shavitt, R. J. Bartlett, "Many-Body Methods in Chemistry and Physics"
  - R. J. Bartlett, M. Musiał, "Coupled-cluster theory in quantum chemistry"
- **Огляди:**
  - CCSD(T): Reviews in Computational Chemistry, Vol. 14
  - CASSCF: Chemical Reviews 2012, 112, 108
  - Post-HF methods: Wiley Interdisciplinary Reviews: Computational Molecular Science
- **Документація PySCF:**
  - https://pyscf.org/user/mp.html — MP2, MP3
  - https://pyscf.org/user/cc.html — Coupled Cluster
  - https://pyscf.org/user/mcscf.html — CASSCF, CASPT2
  - https://pyscf.org/user/fci.html — Full CI

5.7.8. Що далі?

У наступному розділі ми розглянемо аналіз результатів квантово-хімічних розрахунків: орбітальні енергії, густини, заселеності, спектроскопічні властивості та візуалізацію даних.

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. — 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. — URL: https://nznano.blogspot.com/2018/03/simple-quantum-chemistry-hartree-fock.html.