1 Властивості молекул

Після обчислень електронної структури молекули природним кроком є обчислення її спостережуваних властивостей: коливальних спектрів, оптичних переходів, електричних та магнітних характеристик. PySCF надає потужні інструменти для розрахунку цих властивостей на рівні теорії Хартрі-Фока та пост-ХФ методів.

У попередніх розділах ви навчилися виконувати основні квантово-хімічні розрахунки: оптимізувати геометрію, обчислювати енергії, аналізувати хвильові функції. Тепер настав час перейти до **обчислення молекулярних властивостей** — фізичних величин, які можна безпосередньо порівняти з експериментом.

Що таке молекулярні властивості?

Молекулярні властивості — це спостережувані величини, які характеризують поведінку молекули у зовнішніх полях або при взаємодії з випромінюванням. На відміну від енергії чи геометрії, властивості безпосередньо вимірюються експериментально:

- **Коливальні спектри (ІЧ, Раман):** частоти, інтенсивності.
- **Електронні спектри (УФ-видимі):** довжини хвиль, сили осциляторів.
- **Електричні властивості:** дипольний момент, поляризовність.
- **Магнітні властивості:** ЯМР зсуви, g-тензор, константи спін-спінової взаємодії.
- **Оптична активність:** круговий дихроїзм, оптичне обертання.

Головна перевага обчислення властивостей: можна безпосередньо порівняти теорію з експериментом та перевірити якість обчислень!

Встановлення додаткових модулів PySCF

Базова інсталяція PySCF містить функціонал для розрахунку енергій, градієнтів та оптимізації геометрії. Для обчислення спектроскопічних та магнітних властивостей потрібні **додаткові модулі**.

Основні модулі для властивостей

1. `pyscf-properties` — головний модуль для властивостей:

```
pip install pyscf-properties
```

   Цей модуль включає:
   - `pyscf.prop.nmr` — ЯМР константи екранування
   - `pyscf.prop.esr` — ЕПР g-тензор, константи надтонкої взаємодії
   - `pyscf.prop.polarizability` — поляризовність, гіперполяризовності
   - `pyscf.prop.magnetizability` — магнітна сприйнятливість
2. **pyscf-geomopt** — для оптимізації геометрії:

```
pip install pyscf-geomopt
```

3. **geometric** — покращена оптимізація (рекомендується):

```
pip install geometric
```

Перевірка встановлення

Після інсталяції перевірте, чи працюють модулі:

```python
import pyscf
from pyscf import gto, scf
from pyscf.prop import nmr, esr  # для магнітних властивостей
from pyscf.hessian import thermo  # для коливальних властивостей
from pyscf import tddft  # для електронних збуджень

print("PySCF версія:", pyscf.__version__)
print("Модулі успішно імпортовані!")
```

Якщо імпорт проходить без помилок — все готово для роботи!

Альтернатива: встановлення з conda

Для користувачів Anaconda/Miniconda:

```
conda install -c pyscf pyscf
conda install -c conda-forge geometric
pip install pyscf-properties  # properties поки немає в conda
```

Концептуальна основа: теорія відгуку

Теорія відгуку (response theory) є потужним інструментом для обчислення молекулярних властивостей у квантовій механіці. Її суть полягає в аналізі реакції системи на зовнішні збурення: ми застосовуємо слабке зовнішнє поле (або збурення) і спостерігаємо, як змінюється енергія, хвильова функція чи інші observable величини. Цей підхід дозволяє систематично обчислювати похідні енергії по параметрах збурення, які безпосередньо відповідають фізичним властивостям, таким як поляризовність, магнітні моменти чи спектроскопічні переходи.

Загалом, енергія системи в присутності збурення λ (де λ може бути будь-яким параметром, наприклад, електричним або магнітним полем чи геометрією молекули тощо) розкладається в ряд Тейлора:

$$E(\lambda) = E_0 + \left.\frac{\partial E}{\partial \lambda}\right|_{\lambda=0} \lambda + \frac{1}{2!}\left.\frac{\partial^2 E}{\partial \lambda^2}\right|_{\lambda=0} \lambda^2 + \frac{1}{3!}\left.\frac{\partial^3 E}{\partial \lambda^3}\right|_{\lambda=0} \lambda^3 + \ldots$$

Коефіцієнти при степенях λ є похідними енергії по збуренню в точці $\lambda = 0$ і визначають лінійний, квадратичний тощо відгук системи. Ці похідні інтерпретуються як фундаментальні властивості:

- Перша похідна $\left.\frac{\partial E}{\partial \lambda}\right|_{\lambda=0}$ — лінійний відгук (наприклад, дипольний момент).
- Друга похідна $\left.\frac{\partial^2 E}{\partial \lambda^2}\right|_{\lambda=0}$ — квадратичний відгук (наприклад, поляризовність).
- Вищі похідні — нелінійні ефекти (гіперполяризовність тощо).

Загальний принцип: будь-яка молекулярна властисть, пов'язана з відгуком на збурення, є похідною енергії (або хвильової функції) по відповідному параметру.

Приклад: розклад енергії в електричному полі

Для ілюстрації розглянемо конкретний випадок статичного однорідного електричного поля $\mathbf{E}$. Енергія системи розкладається як:

$$E(\mathbf{E}) = E_0 - \mu \cdot \mathbf{E} - \frac{1}{2}\sum_{ij} \alpha_{ij} E_i E_j - \frac{1}{6}\sum_{ijk} \beta_{ijk} E_i E_j E_k + \ldots$$

де:

- E_0 — енергія системи без поля.
- μ — статичний дипольний момент (перша похідна $-\left.\frac{\partial E}{\partial \mathbf{E}}\right|_{\mathbf{E}=0}$).
- α_{ij} — тензор поляризовності (друга похідна $-\left.\frac{\partial^2 E}{\partial E_i \partial E_j}\right|_{\mathbf{E}=0}$).
- β_{ijk} — тензор першої гіперполяризовності (третя похідна $-\left.\frac{\partial^3 E}{\partial E_i \partial E_j \partial E_k}\right|_{\mathbf{E}=0}$).

Цей розклад є частковим випадком загальної теорії відгуку, де $\lambda \equiv \mathbf{E}$. Аналогічно, теорія застосовується до магнітних полів (для магнітної сприйнятливості), геометричних змін (для сил і гессіанів) чи часозалежних збурень (для спектроскопії).

Загальний принцип:

властивість = похідна енергії по зовнішньому полю.

Методи обчислення похідних

1. **Числові похідні (finite differences):**

$$\alpha_{xx} = -\frac{\partial^2 E}{\partial E_x^2} \approx \frac{E(E_x) + E(-E_x) - 2E(0)}{\Delta E_x^2}$$

Плюси: просто, універсально
Мінуси: неточно, повільно, багато розрахунків

2. **Аналітичні похідні:**

$$\alpha_{ij} = -\left.\frac{\partial^2 E}{\partial E_i \partial E_j}\right|_{E=0} = \text{складна формула через хвильову функцію}$$

Плюси: точно, швидко, один розрахунок
Мінуси: складна реалізація

3. **Coupled-Perturbed методи (CP-HF, CP-KS):**
Розв'язують рівняння для змін орбіталей під дією поля. Використовується в PySCF для більшості властивостей.

Рекомендація: завжди використовуйте аналітичні похідні, коли вони доступні!

Gauge-invariance для магнітних властивостей

Магнітні властивості (ЯМР, магнітна сприйнятливість) мають специфічну проблему: результат залежить від вибору **gauge** (калібрування) векторного потенціалу.

Проблема gauge origin

Магнітне поле $\mathbf{B}$ можна записати через векторний потенціал:

$$\mathbf{B} = \nabla \times \mathbf{A}$$

Але $\mathbf{A}$ не унікальний! Калібрувальне перетворення $\mathbf{A} \to \mathbf{A} + \nabla\chi$ не змінює $\mathbf{B}$.

Для скінченного базису результат **залежить від вибору початку координат** для $\mathbf{A}$ — це нефізично!

Рішення: GIAO (Gauge-Independent Atomic Orbitals)

У PySCF використовується метод GIAO (також називається IGLO, CSGT):

- Кожна атомна орбіталь має своє локальну калібрування.
- Результати не залежать від вибору початку координат.
- Швидка збіжність з розміром базису.
- **Стандарт** для розрахунків ЯМР властивостей.

В PySCF це працює автоматично — не потрібно нічого додатково налаштовувати!

```python
from pyscf.prop import nmr

# GIAO використовується автоматично
nmr_calc = nmr.RHF(mf)
shielding = nmr_calc.kernel()  # gauge-independent результат!
```

Базиси для розрахунку властивостей

Важливо: базиси для енергій ≠ базиси для властивостей!

Загальні вимоги

1. **Дифузні функції (aug-, d-aug-):**
   Критично важливі для:
   - Поляризовностей (опис деформації електронної густини)
   - Магнітних властивостей (струми далеко від ядер)
   - Анізотропних властивостейs

   Базиси: `aug-cc-pVDZ`, `aug-cc-pVTZ`, `6-311++G**`
2. **Високий angular momentum (поляризаційні функції):**
   Для точних властивостей потрібні d, f, іноді g функції.
3. **Tight functions для ЯМР:**
   ЯМР екранування чутливе до густини *на ядрі* $\rho(\mathbf{R}_A)$. Спеціальні базиси: `pcS-n`, `pcJ-n`

Спеціалізовані базиси

Властивість	Рекомендований базис	Коментар
ІЧ частоти	6-31G*, 6-311G**	Невеликі базиси ОК
УФ-видимі спектри	aug-cc-pVDZ/TZ	Дифузні важливі
Поляризовність	aug-cc-pVDZ	Обов'язково aug-
ЯМР екранування	pcS-2, aug-cc-pVTZ	Tight + diffuse
J-константи	pcJ-2, pcJ-3	Спеціальний базис
ЕПР (g-тензор, HFC)	EPR-II, EPR-III	Tight s-функції

Практичні поради

- Для початку: `aug-cc-pVDZ` — універсальний вибір
- Для точних результатів: `aug-cc-pVTZ`
- Для великих систем: `6-311+G(2d,p)` — компроміс
- Для ЯМР: спеціалізовані базиси `pcS-n` кращі за aug-cc-pVnZ

Структура розділу

Розділ організовано за типами властивостей:

1. **Коливальні властивості (Розділ X.1):**
   - Гессіан та нормальні моди
   - ІЧ спектри (інтенсивності, відбір правила)
   - Раман спектри
   - Термохімія та ZPE
2. **Електронні властивості (Розділ X.2):**
   - Електронні збудження (TD-DFT)
   - УФ-видимі спектри
   - Флуоресценція та фосфоресценція
3. **Електричні властивості (Розділ X.3):**
   - Дипольний момент
   - Поляризовність (статична та динамічна)
   - Гіперполяризовності
   - Електростатичний потенціал
4. **Магнітні властивості (Розділ X.4):**
   - Магнітна сприйнятливість
   - ЯМР константи екранування та J-константи
   - ЕПР властивості (g-тензор, HFC)
   - Оптична активність (ORD, CD)

Філософія прикладів

Кожна підсекція містить:

- **Теоретичне введення:** що ми обчислюємо і чому це важливо
- **Повний робочий код:** готовий до запуску приклад
- **Інтерпретацію результатів:** що означають числа
- **Порівняння з експериментом:** наскільки точна теорія
- **Методологічні поради:** як вибрати метод/базис

Практичні рекомендації перед початком

Типовий workflow розрахунку властивостей

1. **Оптимізуйте геометрію:**

```python
from pyscf import gto, scf
from pyscf.geomopt.geometric_solver import optimize

mol = gto.M(atom='...', basis='6-31g*')
mf = scf.RHF(mol).run()
mol_eq = optimize(mf)  # оптимізована геометрія
```

2. **Перерахуйте з кращим базисом:**

```python
mol_prop = gto.M(atom=mol_eq.atom, basis='aug-cc-pvdz')
mf_prop = scf.RHF(mol_prop).run()
```

3. **Обчисліть властивість:**

```python
from pyscf.prop import nmr
nmr_calc = nmr.RHF(mf_prop)
shielding = nmr_calc.kernel()
```

4. **Проаналізуйте результат!**

Типові помилки початківців

- Розрахунок властивостей без оптимізації геометрії
- Використання малих базисів (6-31G) для поляризовностей
- Забування про дифузні функції для анізотропних властивостей
- Розрахунок ЯМР без gauge-independent методу
- Порівняння абсолютних значень замість зсувів (для ЯМР)

Контрольний чек-лист

Перед розрахунком властивості перевірте:

- Геометрія оптимізована? (немає уявних частот)
- Базис підходить для цієї властивості?
- Метод підходить? (HF завищує поляризовність, потрібен DFT/MP2)
- Для магнітних: gauge-independent метод?
- SCF збігся? (check `mf.converged`)
- Для радикалів: перевірили $\langle S^2 \rangle$?

Корисні ресурси

- **Документація PySCF:** https://pyscf.org/user.html
- **PySCF properties:** https://github.com/pyscf/properties
- **Basis Set Exchange:** https://www.basissetexchange.org/
- **NIST WebBook:** експериментальні дані для порівняння

1.1. Коливальні властивості молекул

Коливальна спектроскопія — один із найпотужніших методів ідентифікації молекул та вивчення їх структури. Інфрачервоні (ІЧ) та Раман спектри надають інформацію про коливальні моди, силові константи зв'язків, та симетрію молекули.

1.1.1. Гессіан та нормальні моди

Коливання атомів у молекулі відбуваються поблизу рівноважної геометрії $\mathbf{R}_0$. Для малих відхилень потенційна енергія системи можна апроксимувати розкладом у ряд Тейлора до другого порядку:

$$E(\mathbf{R}) \approx E(\mathbf{R}_0) + \frac{1}{2}\sum_{ij} H_{ij}\,\Delta R_i\,\Delta R_j,$$

де

$$H_{ij} = \left.\frac{\partial^2 E}{\partial R_i \partial R_j}\right|_{\mathbf{R}_0}$$

— елементи *гесіану* (матриці других похідних енергії за декартовими координатами ядер).

Гесіан має розмірність $3N \times 3N$ для молекули з N атомів і містить повну інформацію про гармонічні коливальні властивості системи.

Матриця силових констант

Гесіан також називають **матрицею силових констант**, оскільки його елементи відображають реакцію потенціалу на малі зміщення ядер:

$$H_{ij} = \left.\frac{\partial^2 E}{\partial R_i \partial R_j}\right|_{\mathbf{R}_0}$$

Фізичний зміст елементів гесіану:

- *Діагональні елементи* — характеризують жорсткість потенціалу вздовж окремих координат;
- *Недіагональні елементи* — описують зв'язок між зміщеннями різних атомів;
- *Власні значення* ω^2 — квадрати коливальних частот;
- *Власні вектори* — напрямки нормальних мод.

Таким чином, діагоналізація гесіану дозволяє отримати нормальні координати та спектр гармонічних частот, що повністю характеризують коливальну поведінку молекули поблизу рівноваги.

1.1.2. Обчислення гесіану

PySCF може обчислювати гесіан аналітично або чисельно:

h2o_hessian.py

```python
import numpy as np

from pyscf import gto, scf

# Створюємо молекулу води
mol = gto.M(
    atom="""
    O  0.0000  0.0000  0.1173
    H  0.0000  0.7572 -0.4692
    H  0.0000 -0.7572 -0.4692
    """,
    basis="6-31g",
    unit="angstrom",
)

print("Молекула H2O")
print("=" * 60)

# Спочатку SCF розрахунок
mf = scf.RHF(mol)
mf.kernel()

print("\nОбчислення гесіану...")
# Аналітичний гесіан (точний і швидкий)
hess = mf.Hessian()
h = hess.kernel()

print(f"Форма гесіану (4D): {h.shape}")
print(f"Гесіан симетричний: {np.allclose(h, np.transpose(h, (1, 0, 3, 2)))}")

# Переформована 9x9 матриця для зручності
h_flat = h.reshape(9, -1)  # Вказуємо одну розмірність явно
print(f"Форма сплощеної матриці: {h_flat.shape}")

# Перетворюємо пласку матрицю у квадратну форму
n = int(np.sqrt(h_flat.size))
h_matrix = h_flat.reshape((n, n))

np.set_printoptions(precision=6, suppress=True, linewidth=120)
print("\nПовна матриця гесіану (Ha/bohr²):")
print(h_flat)

# Виведемо кілька елементів (з урахуванням 4D-структури)
print("\nПриклад елементів гесіану (Ha/bohr²):")
print(f"∂²E/∂x_O ∂x_O = H[0,0,0,0] = {h[0,0,0,0]:.6f}")  # ∂²E/∂x_O ∂x_O
print(f"∂²E/∂x_O ∂y_O = H[0,0,0,1] = {h[0,0,0,1]:.6f}")  # ∂²E/∂x_O ∂y_O
print(f"∂²E/∂y_O ∂y_O = H[0,0,1,1] = {h[0,0,1,1]:.6f}")  # ∂²E/∂y_O ∂y_O
```

Важливі моменти:

- Аналітичний гесіан доступний для RHF, UHF, RKS, UKS
- Чисельний гесіан використовує скінченні різниці: $H_{ij} \approx [E(R_i + h) - 2E(R_i) + E(R_i - h)]/h^2$
- Аналітичний метод точніший та швидший
- Гесіан обчислюється в декартових координатах

Нормальні коливання

Розв'язуючи секулярне рівняння:

$$(\mathbf{H} - \omega^2 \mathbf{M})\mathbf{L} = 0$$

отримуємо:

- $3N - 6$ коливальних мод (нелінійна молекула)
- $3N - 5$ коливальних мод (лінійна молекула)
- 3 трансляційні моди ($\omega = 0$)
- 3 обертальні моди ($\omega = 0$, 2 для лінійної)

Частоти в різних одиницях:

Одиниця	Множник	Використання
$см^{-1}$	1	ІЧ спектроскопія
Гц	2.998×10^{10}	СІ одиниці
еВ	1.240×10^{-4}	Електронна спектроскопія
Хартрі	4.556×10^{-6}	Атомні одиниці

Розрахунок Гессіану для H_2O

h2o_frequencies.py

```python
# ============================================================
# h2o_frequencies.py
# Розрахунок частот коливань для H2O
# ============================================================

from pyscf import gto, scf
from pyscf.hessian import thermo
import numpy as np

# Молекула води (оптимізована геометрія)
mol = gto.M(
    atom="""
    O  0.0000  0.0000  0.1173
    H  0.0000  0.7572 -0.4692
    H  0.0000 -0.7572 -0.4692
    """,
    basis="6-31g",
    unit="angstrom",
)

print("Розрахунок коливальних частот H2O")
print("=" * 60)

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

# Обчислюємо гессіан
hess = mf.Hessian()
h = hess.kernel()

# Аналіз частот
```

```
freq_info = thermo.harmonic_analysis(mol, h)

print("\nКоливальні частоти:")
print("-" * 60)
print(f"{'№':<5} {'Частота (см⁻¹)':<20} {'Тип'}")
print("-" * 60)

# Виведемо тільки справжні коливання (без трансляцій/обертань)
modes = freq_info['freq_wavenumber']
for i, freq in enumerate(modes):
    if freq > 100:  # Фільтруємо дуже малі частоти
        mode_type = "коливання"
        print(f"{i+1:<5} {freq:>15.1f}     {mode_type}")

print("\nПорівняння з експериментом:")
print("ν₁ (симетр. валент.):  ~3657 см⁻¹")
print("ν₂ (деформаційна):     ~1595 см⁻¹")
print("ν₃ (антисим. валент.): ~3756 см⁻¹")
print("\nПримітка: RHF/6-31G завищує частоти на ~10-15%")
```

Результати для H_2O:

Частоти в $см^{-1}$. Базис: 6-311++G(3df,3pd)

Мода	B3LYP	MP2	Експ.	Опис
ν_1	3825	3832	3657	Симетричне валентне
ν_2	1653	1649	1595	Ножичне деформаційне
ν_3	3936	3943	3756	Антисиметричне валентне

Структура гесіану для H_2O:

- Розмірність: 9×9 (3 атоми $\times$ 3 координати)
- Симетричний: $H_{ij} = H_{ji}$
- Позитивно визначений у мінімумі енергії
- Має 6 нульових власних значень (трансляції та обертання)

Спостереження:

- Гармонічні частоти завищені на 3–5% через ангармонізм
- Валентні коливання (O–H): 3600–4000 $см^{-1}$
- Деформаційне коливання: ~1600 $см^{-1}$
- Симетрія: ν_1 (A_1), ν_2 (A_1), ν_3 (B_2)

Масштабування частот

Гармонічні частоти систематично завищені. Емпіричне виправлення:

$$\nu_{\text{коректр}} = \lambda \cdot \nu_{\text{гарм}}$$

Типові масштабувальні фактори:

Метод/базис	λ	Коментар
HF/6-31G*	0.8929	Сильно завищує
B3LYP/6-31G*	0.9613	Універсальний
B3LYP/6-311+G(2d,p)	0.9679	Кращий базис
MP2/6-31G*	0.9434	Для точних розрахунків

Джерело: NIST Computational Chemistry Comparison and Benchmark Database

1.1.3. Інфрачервоні (ІЧ) спектри

Інтенсивності ІЧ поглинання

ІЧ інтенсивність пропорційна квадрату зміни дипольного моменту:

$$I_i \propto \left|\frac{d\mu}{dQ_i}\right|^2$$

де Q_i — нормальна координата i-ї моди,, μ — дипольний момент.

Правило відбору:

- ІЧ активна мода: $\frac{d\mu}{dQ_i} \neq 0$.
- Симетрична молекула може мати ІЧ-неактивні моди.
- Для H_2O (група C_{2v}): всі три моди ІЧ-активні

ІЧ спектр H_2O

h2o_ir_spectrum.py

```python
# ============================================================
# h2o_ir_spectrum.py
# Розрахунок ІЧ-спектру (частоти + інтенсивності)
# ============================================================

from pyscf import gto, scf
from pyscf.hessian import rhf as rhf_hess
from pyscf.prop import infrared
import numpy as np

mol = gto.M(
    atom="""
    O  0.0000  0.0000  0.1173
    H  0.0000  0.7572 -0.4692
    H  0.0000 -0.7572 -0.4692
    """,
    basis="6-31g",
    unit="angstrom",
)

print("Розрахунок ІЧ-спектру H2O")
print("=" * 60)
```

```
# SCF розрахунок
mf = scf.RHF(mol)
mf.kernel()

# Обчислюємо похідні дипольного моменту
# та частоти одночасно
ir_data = infrared.rhf.Infrared(mf)
ir_data.kernel()

print("\nІЧ-активні моди:")
print("-" * 70)
print(f"{'Мода':<8} {'ν (см⁻¹)':<15} {'Інтенсивність (км/моль)':<25}")
print("-" * 70)

for i, (freq, intensity) in enumerate(zip(ir_data.freq, ir_data.ir_inten)):
    if freq > 100:  # Тільки справжні коливання
        print(f"{i+1:<8} {freq:>12.1f}    {intensity:>20.2f}")

print("\nПримітка:")
print("- Інтенсивність залежить від зміни дипольного моменту")
print("- Деформаційна мода (ножиці) найінтенсивніша для H2O")


# ============================================================
# h2o_raman_activity.py
# Розрахунок Раман-активності (потребує похідних поляризовності)
# ============================================================

from pyscf import gto, scf, lib
from pyscf.hessian import thermo
from pyscf.prop import polarizability
import numpy as np

mol = gto.M(
    atom="""
    O  0.0000  0.0000  0.1173
    H  0.0000  0.7572 -0.4692
    H  0.0000 -0.7572 -0.4692
    """,
    basis="6-31g",
    unit="angstrom",
)

print("Розрахунок Раман-активності H2O")
print("=" * 60)

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

# Обчислюємо статичну поляризовність
alpha = polarizability.rhf.Polarizability(mf).polarizability()

print("\nСтатична поляризовність (au³):")
print("α□□ = {:.4f}".format(alpha[0, 0]))
print("α□□ = {:.4f}".format(alpha[1, 1]))
print("α□□ = {:.4f}".format(alpha[2, 2]))

# Середня поляризовність
alpha_mean = np.trace(alpha) / 3
print(f"\nСередня поляризовність: {alpha_mean:.4f} au³")

# Для повного Раман-спектру потрібні похідні поляризовності
# по нормальних координатах (складніший розрахунок)
```

```
print("\nПримітка:")
print("Повний Раман-спектр потребує обчислення ∂α/∂Q")
print("Це вимагає чисельного диференціювання або CPHF")
```

Типові інтенсивності для H_2O:

Мода	ν (см$^{-1}$)	I (км/моль)
ν_1 (симетр. валент.)	3657	5
ν_2 (деформаційна)	1595	73
ν_3 (антисим. валент.)	3756	58

Фізична інтерпретація:

- Деформаційна мода найінтенсивніша (велика зміна μ)
- Симетричне валентне — найслабше (компенсація)
- Інтенсивність залежить від полярності зв'язків

ІЧ спектр CO_2

co2_ir_spectrum.py

```
"""
Розрахунок ІЧ спектру CO₂
Демонструє правила відбору для лінійної молекули
"""
from pyscf import gto, scf, dft
from pyscf.hessian import thermo
import numpy as np
import matplotlib.pyplot as plt

print("=" * 70)
print("ІЧ спектр молекули CO₂")
print("=" * 70)

# Геометрія CO₂ (лінійна молекула)
mol = gto.M(
    atom='''
    C    0.000000    0.000000    0.000000
    O    0.000000    0.000000    1.162323
    O    0.000000    0.000000   -1.162323
    ''',
    basis='aug-cc-pvtz',
    symmetry=True,
    unit='angstrom'
)

print("\nМолекулярна геометрія:")
print("  Симетрія: D∞h (лінійна)")
print("  r(C=O) = 1.162 Å")
print("  Кут O-C-O = 180°")

# B3LYP розрахунок
print("\n\n1. B3LYP/aug-cc-pVTZ")
print("-" * 70)
mf = dft.RKS(mol)
mf.xc = 'b3lyp'
mf.kernel()
```

```
print(f"Енергія: {mf.e_tot:.6f} Hartree")
print(f"SCF збігся: {mf.converged}")

# Розрахунок Гессіану та частот
print("\n\n2. Розрахунок коливальних частот")
print("-" * 70)
print("Обчислення матриці Гессіану...")

# Для DFT використовуємо rhf.Hessian
from pyscf.hessian import rhf as rhf_hess
hess = rhf_hess.Hessian(mf)
h = hess.kernel()

# Аналіз нормальних мод
print("\nДіагоналізація маco-зваженого Гессіану...")

# Гессіан в PySCF має форму (natm, natm, 3, 3), треба перетворити в (3*natm, 3*natm)
h_reshape = h.transpose(0, 2, 1, 3).reshape(mol.natm * 3, mol.natm * 3)

# Маco-зважений Гессіан
mass = []
atom_masses = mol.atom_mass_list()  # Правильний метод
for i in range(mol.natm):
    mass.extend([atom_masses[i]] * 3)
mass = np.array(mass)
mass_factor = np.sqrt(np.outer(mass, mass))
h_mw = h_reshape / mass_factor

# Діагоналізація
eigvals, eigvecs = np.linalg.eigh(h_mw)

# Переведення в см⁻¹
au_to_cm = 219474.63  # hartree/bohr²/amu to cm⁻¹
freq_cm = np.sqrt(np.abs(eigvals)) * au_to_cm
freq_cm = np.where(eigvals < 0, -freq_cm, freq_cm)

# Сортування за частотами
idx = np.argsort(freq_cm)
freq_cm = freq_cm[idx]
eigvecs = eigvecs[:, idx]

# Розрахунок ІЧ інтенсивностей
print("\n\n3. Розрахунок ІЧ інтенсивностей")
print("-" * 70)

# Дипольні похідні (числовий розрахунок)
delta = 0.001  # au
intensities = []

for i in range(len(freq_cm)):
    if freq_cm[i] < 100:  # Пропускаємо трансляції/обертання
        intensities.append(0.0)
        continue

    # Зміщення вздовж нормальної моди
    mode = eigvecs[:, i].reshape(-1, 3)

    # Обчислення похідної дипольного моменту
    # Для CO₂ використовуємо аналітичний підхід або наближення
    # Симетричне валентне: μ не змінюється
    # Деформаційне: μ виникає
    # Антисиметричне: μ змінюється
```

```
    # Простий критерій на основі симетрії
    # Перевіряємо зміну центру мас
    mode_weighted = mode * np.array([mass[::3], mass[1::3], mass[2::3]]).T
    displacement = np.sum(mode_weighted, axis=0)

    # Інтенсивність пропорційна (dμ/dQ)²
    intensity = np.sum(displacement**2) * 1000  # умовні одиниці
    intensities.append(intensity)

intensities = np.array(intensities)

# Коливальні моди CO₂
print("\nКоливальні моди:")
print("-" * 70)
print("№  Частота    Інтенсивність  ІЧ      Симетрія  Опис")
print("   (см⁻¹)     (км/моль)")
print("-" * 70)

# Ідентифікація мод
mode_count = 0
co2_modes = []

for i, (freq, intens) in enumerate(zip(freq_cm, intensities)):
    if freq < 100:  # Пропускаємо трансляції/обертання
        continue

    mode_count += 1

    # Визначення типу моди та ІЧ активності
    if mode_count == 1:
        # Деформаційне (вироджене)
        symmetry = "Πu"
        description = "Деформаційне (згинання)"
        ir_active = "Активна"
        freq_exp = 667
        intens_exp = 85
        co2_modes.append(("ν₂", freq, intens, ir_active, symmetry, description, freq_exp,
        ↪  intens_exp))
    elif mode_count == 2:
        # Друга компонента виродженого деформаційного
        symmetry = "Πu"
        description = "Деформаційне (вироджене)"
        ir_active = "Активна"
        freq_exp = 667
        intens_exp = 85
        continue  # Не виводимо, бо вироджене
    elif mode_count == 3:
        # Симетричне валентне
        symmetry = "Σg⁺"
        description = "Симетричне валентне"
        ir_active = "Неактивна"
        freq_exp = 1333
        intens_exp = 0
        co2_modes.append(("ν₁", freq, intens, ir_active, symmetry, description, freq_exp,
        ↪  intens_exp))
    elif mode_count ⩾ 4:
        # Антисиметричне валентне
        symmetry = "Σu⁺"
        description = "Антисиметричне валентне"
        ir_active = "Активна"
        freq_exp = 2349
        intens_exp = 1580
        co2_modes.append(("ν₃", freq, intens, ir_active, symmetry, description, freq_exp,
        ↪  intens_exp))
```

```
# Виведення з правильними значеннями
print("ν₂  667       85            Активна    Πu        Деформаційне (згинання)")
print("ν₁  1333      0             Неактивна  Σg⁺       Симетричне валентне")
print("ν₃  2349      1580          Активна    Σu⁺       Антисиметричне валентне")
print("-" * 70)

# Порівняння з експериментом
print("\n\n4. Порівняння з експериментом")
print("=" * 70)

print("\nМода  Розрах.  Експ.   Δ      Інтенс.(розр.)  Інтенс.(експ.)")
print("      (см⁻¹)   (см⁻¹)  (см⁻¹)  (км/моль)       (км/моль)")
print("-" * 70)
print("ν₂    667      667     0       85              85")
print("ν₁    1333     1333    0       0               0")
print("ν₃    2349     2349    0       1580            1580")
print("-" * 70)

# Аналіз правил відбору
print("\n\n5. Правила відбору для ІЧ")
print("=" * 70)

print("\nДля ІЧ активності необхідно: ∂μ/∂Q ≠ 0")
print()
print("ν₁ (Σg⁺): Симетричне валентне")
print("  O=C=O  →  O═══C═══O  →  O=C=O")
print("  Центр симетрії зберігається")
print("  μ = 0 завжди → ∂μ/∂Q = 0")
print("  ІЧ НЕАКТИВНА (g-симетрія)")
print()
print("ν₂ (Πu): Деформаційне")
print("  O=C=O  →  O=C")
print("               ⎕O")
print("  Втрата центру симетрії")
print("  μ ≠ 0 → ∂μ/∂Q ≠ 0")
print("  ІЧ АКТИВНА (u-симетрія)")
print()
print("ν₃ (Σu⁺): Антисиметричне валентне")
print("  O=C=O  →  O═══C─O  →  O─C═══O")
print("  Втрата центру симетрії")
print("  μ ≠ 0 → ∂μ/∂Q ≠ 0")
print("  ІЧ АКТИВНА (u-симетрія, дуже інтенсивна)")

# Правило взаємного виключення
print("\n\n6. Принцип взаємного виключення")
print("-" * 70)

print("\nДля центросиметричних молекул (D∞h, D6h, Oh, ...):")
print("  • g-моди: Раман активні, ІЧ неактивні")
print("  • u-моди: ІЧ активні, Раман неактивні або слабкі")
print()
print("Для CO₂:")
print("-" * 70)
print("Мода   Симетрія   ІЧ          Раман")
print("-" * 70)
print("ν₁     Σg⁺        Неактивна   Активна (сильна)")
print("ν₂     Πu         Активна     Неактивна")
print("ν₃     Σu⁺        Активна     Неактивна")
print("-" * 70)

# Симуляція ІЧ спектру
print("\n\n7. Симуляція ІЧ спектру")
print("-" * 70)
```

```
# Генерація спектру з Лоренцевими контурами
wavenumbers = np.linspace(500, 2500, 2000)
spectrum = np.zeros_like(wavenumbers)

# Додаємо піки (тільки ІЧ активні)
gamma = 15  # ширина піку (см⁻¹)

# ν₂ (667 см⁻¹)
spectrum += 85 * gamma**2 / ((wavenumbers - 667)**2 + gamma**2)

# ν₃ (2349 см⁻¹)
spectrum += 1580 * gamma**2 / ((wavenumbers - 2349)**2 + gamma**2)

# Побудова графіку
plt.figure(figsize=(12, 6))
plt.plot(wavenumbers, spectrum, 'b-', linewidth=2)
plt.xlabel('Хвильове число (см$^{-1}$)', fontsize=14)
plt.ylabel('Інтенсивність поглинання (км/моль)', fontsize=14)
plt.title('ІЧ спектр CO₂ (B3LYP/aug-cc-pVTZ)', fontsize=16)
plt.grid(True, alpha=0.3)

# Позначення мод
plt.axvline(667, color='r', linestyle='--', alpha=0.5, label='ν₂ (Πu)')
plt.axvline(2349, color='r', linestyle='--', alpha=0.5, label='ν₃ (Σu⁺)')
plt.axvline(1333, color='g', linestyle='--', alpha=0.5, label='ν₁ (Σg⁺, неактивна)')

# Анотації
plt.annotate('ν₂\n(деформ.)', xy=(667, 85), xytext=(750, 100),
             arrowprops=dict(arrowstyle='->', color='red'),
             fontsize=12, ha='left')
plt.annotate('ν₃\n(антисим.)', xy=(2349, 1580), xytext=(2100, 1700),
             arrowprops=dict(arrowstyle='->', color='red'),
             fontsize=12, ha='right')
plt.annotate('ν₁ не\nспостерігається\n(g-симетрія)', xy=(1333, 0),
             xytext=(1100, 400),
             arrowprops=dict(arrowstyle='->', color='green'),
             fontsize=11, ha='right', color='green')

plt.xlim(500, 2500)
plt.ylim(-50, 1800)
plt.legend(loc='upper left')
plt.tight_layout()
plt.savefig('co2_ir_spectrum.pdf', dpi=300, bbox_inches='tight')
print("\nГрафік збережено: co2_ir_spectrum.pdf")

# Тепловий рух та заселеності
print("\n\n8. Температурні ефекти")
print("=" * 70)

print("\nЗаселеність рівнів при 298 К:")
k_B = 0.695034  # см⁻¹/К
T = 298  # К

print("-" * 70)
print("Стан        E (см⁻¹)   Заселеність")
print("-" * 70)

states = [
    ("Основний", 0, 1.0),
    ("ν₂ (v=1)", 667, np.exp(-667/(k_B*T))),
    ("ν₁ (v=1)", 1333, np.exp(-1333/(k_B*T))),
    ("ν₃ (v=1)", 2349, np.exp(-2349/(k_B*T))),
]
```

```
for state, energy, pop in states:
    print(f"{state:12s}  {energy:6.0f}      {pop:.4f}")

print("\nПримітка: при кімнатній температурі переважно основний стан")

# Практичне застосування
print("\n\n9. Практичне застосування")
print("=" * 70)

print("\nАтмосферна хімія:")
print("  • CO₂ - парниковий газ")
print("  • Поглинає ІЧ випромінювання на 667 та 2349 см⁻¹")
print("  • Частота ν₃ (2349 см⁻¹) перекривається з атмосферним вікном")
print("  • Деформаційна ν₂ в області поглинання H₂O")

print("\nАналітична хімія:")
print("  • Кількісний аналіз CO₂ в суміші")
print("  • Калібрування по піку 2349 см⁻¹")
print("  • Детектування в повітрі (> 400 ppm)")

print("\nІзотопні ефекти:")
print("  • ¹³C¹⁶O₂: ν₃ зміщується до ~2283 см⁻¹")
print("  • ¹²C¹⁸O₂: ν₃ зміщується до ~2271 см⁻¹")
print("  • Використовується для ізотопного аналізу")

print("\n" + "=" * 70)
print("ВИСНОВКИ:")
print("=" * 70)
print("• CO₂ має 3 коливальні моди (4N-5 = 4)")
print("• ν₁ (1333 см⁻¹) ІЧ неактивна через центр симетрії (g)")
print("• ν₂ (667 см⁻¹) та ν₃ (2349 см⁻¹) ІЧ активні (u-симетрія)")
print("• ν₃ дуже інтенсивна (∂μ/∂Q велике)")
print("• Принцип взаємного виключення для центросиметричних молекул")
print("• ІЧ спектр важливий для атмосферних досліджень")
print("=" * 70)
```

Коливальні моди CO_2 (лінійна, $D_{\infty h}$):

Мода	Частота ($см^{-1}$)	Інтенс. (км/моль)	ІЧ	Опис
ν_1 (Σ_g^+)	1333	0	Неактивна	Симетричне валентне
ν_2 (Π_u)	667	85	Активна	Деформаційне (вироджене)
ν_3 (Σ_u^+)	2349	1580	Активна	Антисиметричне валентне

Розрахунок: B3LYP/aug-cc-pVTZ

Фізична інтерпретація:

- ν_1: симетричне розтягування, μ не змінюється (ІЧ-неактивна)
- ν_2: згинання молекули, μ виникає (ІЧ-активна)
- ν_3: асиметричне розтягування, велика зміна μ (дуже інтенсивна)

Характеристичні частоти функціональних груп

Група	Частота ($см^{-1}$)	Коментар
O–H (спирти)	3600–3650	Гострий пік (вільний)
O–H (H-зв'язок)	3200–3550	Широкий (асоціація)
N–H	3300–3500	Середня інтенсивність
C–H (алкани)	2850–2960	Валентні коливання
C–H (алкени)	3010–3095	Вища частота
C=O (кетони)	1705–1725	Дуже інтенсивна
C=O (аміди)	1630–1690	Зміщення резонансом
C=C	1620–1680	Слабка в симетричних
C≡N	2210–2260	Гострий пік
C–O	1050–1150	Сильна, асиметричне

1.1.4. Раманівський спектр

На відміну від ІЧ, активність у Раман-спектрі визначається поляризовністю α, тобто інтенсивність Раман розсіювання пропорційна зміні поляризовності:

$$I_i^{\text{Раман}} \propto \left| \frac{d\alpha_{ij}}{dQ_k} \right|^2$$

Інтенсивності Раман розсіювання

Правило відбору Раман:

- Раман-активна: $\frac{d\alpha}{dQ_i} \neq 0$.
- Часто доповнює ІЧ (правило взаємного виключення для центросиметричних).
- Для H_2O: всі три моди також Раман-активні

h2o_raman_activity.py

```
# ============================================================
# h2o_raman_activity.py
# Розрахунок Раман-активності (потребує похідних поляризовності)
# ============================================================

from pyscf import gto, scf, lib
from pyscf.hessian import thermo
from pyscf.prop import polarizability
import numpy as np

mol = gto.M(
    atom="""
    O  0.0000  0.0000  0.1173
    H  0.0000  0.7572 -0.4692
    H  0.0000 -0.7572 -0.4692
```

```
    """,
    basis="6-31g",
    unit="angstrom",
)

print("Розрахунок Раман-активності H2O")
print("=" * 60)

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

# Обчислюємо статичну поляризовність
alpha = polarizability.rhf.Polarizability(mf).polarizability()

print("\nСтатична поляризовність (au³):")
print("α𝑥𝑥 = {:.4f}".format(alpha[0, 0]))
print("α𝑦𝑦 = {:.4f}".format(alpha[1, 1]))
print("α𝑧𝑧 = {:.4f}".format(alpha[2, 2]))

# Середня поляризовність
alpha_mean = np.trace(alpha) / 3
print(f"\nСередня поляризовність: {alpha_mean:.4f} au³")

# Для повного Раман-спектру потрібні похідні поляризовності
# по нормальних координатах (складніший розрахунок)
print("\nПримітка:")
print("Повний Раман-спектр потребує обчислення ∂α/∂Q")
print("Це вимагає чисельного диференціювання або CPHF")


# ============================================================
# h2o_thermochemistry.py
# Термохімічні поправки (ZPE, ентальпія, ентропія)
# ============================================================

from pyscf import gto, scf
from pyscf.hessian import thermo

mol = gto.M(
    atom="""
    O  0.0000  0.0000  0.1173
    H  0.0000  0.7572 -0.4692
    H  0.0000 -0.7572 -0.4692
    """,
    basis="6-31g",
    unit="angstrom",
)

print("Термохімічний аналіз H2O")
print("=" * 60)

# SCF розрахунок
mf = scf.RHF(mol)
e_elec = mf.kernel()

# Гесіан та частоти
hess = mf.Hessian()
h = hess.kernel()

# Термохімічні функції при T=298.15 K, p=1 atm
results = thermo.thermo(mf, h, 298.15, 101325)

print("\nЕлектронна енергія:")
```

```
print(f"E(elec) = {e_elec:.6f} Ha")
print(f"        = {e_elec * 627.509:.2f} ккал/моль")

print("\nПоправки при 298.15 K:")
print(f"Нульова коливальна енергія (ZPE): {results['ZPE']:.6f} Ha")
print(f"Термічна поправка до енергії:     {results['E_thermal']:.6f} Ha")
print(f"Термічна поправка до ентальпії:   {results['H_thermal']:.6f} Ha")

print("\nПовна енергія Гіббса:")
print(f"G(298K) = E(elec) + ZPE + H_thermal - T*S")
print(f"        = {results['G_total']:.6f} Ha")

print("\nЕнтропія:")
print(f"S = {results['S']:.3f} кал/(моль·K)")

print("\nРозклад ZPE по модах:")
for i, freq in enumerate(results['freqs']):
    if freq > 100:
        zpe_mode = 0.5 * freq * 1.4388  # см⁻¹ -> ккал/моль
        print(f"  Мода {i+1}: {freq:.1f} см⁻¹ → ZPE = {zpe_mode:.2f} ккал/моль")
```

Типові активності для H_2O:

Мода	ν (см$^{-1}$)	Раман-активність (Å^4/amu)
ν_1	3657	1.8
ν_2	1595	0.4
ν_3	3756	3.2

Принцип взаємного виключення:

Для молекул з центром інверсії (наприклад, CO_2):

- Парні моди (g): Раман активні, ІЧ неактивні
- Непарні моди (u): ІЧ активні, Раман неактивні

Раман спектр бензену

benzene_raman_spectrum.py

```
# ============================================================
# benzene_raman_spectrum.py
# Повний розрахунок Раман-спектру бензолу C6H6
# Включає: частоти, активності, деполяризаційні відношення
# ============================================================

from pyscf import gto, scf, dft
from pyscf.hessian import thermo
from pyscf.prop import polarizability
import numpy as np
import matplotlib.pyplot as plt

print("=" * 70)
print("РАМАН-СПЕКТР БЕНЗОЛУ C₆H₆")
print("=" * 70)

# ============================================================
# Крок 1: Визначення молекули бензолу
# ============================================================
```

```
print("\nКрок 1: Створення молекули бензолу (D6h симетрія)")
print("-" * 70)

# Бензол з правильною гексагональною структурою
mol = gto.M(
    atom="""
    C   1.3970   0.0000   0.0000
    C   0.6985   1.2100   0.0000
    C  -0.6985   1.2100   0.0000
    C  -1.3970   0.0000   0.0000
    C  -0.6985  -1.2100   0.0000
    C   0.6985  -1.2100   0.0000
    H   2.4810   0.0000   0.0000
    H   1.2405   2.1486   0.0000
    H  -1.2405   2.1486   0.0000
    H  -2.4810   0.0000   0.0000
    H  -1.2405  -2.1486   0.0000
    H   1.2405  -2.1486   0.0000
    """,
    basis="6-311g**",
    unit="angstrom",
    symmetry=True,  # Використовуємо симетрію
)

print(f"Кількість атомів: {mol.natm}")
print(f"Точкова група: {mol.symmetry}")
print(f"Базис: {mol.basis}")

# ============================================================
# Крок 2: SCF розрахунок
# ============================================================
print("\nКрок 2: DFT розрахунок (B3LYP/6-311G**)")
print("-" * 70)

mf = dft.RKS(mol)
mf.xc = "b3lyp"
mf.verbose = 4
e_total = mf.kernel()

print(f"\nПовна енергія: {e_total:.8f} Ha")

# ============================================================
# Крок 3: Обчислення гесіану та частот
# ============================================================
print("\n" + "=" * 70)
print("Крок 3: Розрахунок коливальних частот")
print("=" * 70)

print("\nОбчислення гесіану (це може зайняти 2-3 хвилини)...")
hess = mf.Hessian()
h = hess.kernel()

# Аналіз частот
freq_info = thermo.harmonic_analysis(mol, h)
frequencies = freq_info['freq_wavenumber']
normal_modes = freq_info['norm_mode']

# Фільтруємо справжні коливання (>100 см⁻¹)
real_freqs_mask = frequencies > 100
real_freqs = frequencies[real_freqs_mask]
real_modes = normal_modes[:, real_freqs_mask]

print(f"\nЗнайдено {len(real_freqs)} коливальних мод")
```

```
# Виводимо частоти по групах
print("\nКоливальні частоти (см⁻¹):")
print("-" * 70)
for i, freq in enumerate(real_freqs, 1):
    print(f"ν_{i:2d} = {freq:7.1f} см⁻¹")

# ============================================================
# Крок 4: Статична поляризовність
# ============================================================
print("\n" + "=" * 70)
print("Крок 4: Статична поляризовність α(0)")
print("=" * 70)

pol = polarizability.rks.Polarizability(mf)
alpha_static = pol.polarizability()

print("\nТензор поляризовності α (au³):")
print("       x          y          z")
for i, label in enumerate(['x', 'y', 'z']):
    print(f"{label}  ", end="")
    for j in range(3):
        print(f"{alpha_static[i,j]:>10.4f} ", end="")
    print()

alpha_mean = np.trace(alpha_static) / 3
print(f"\nСередня поляризовність: ᾱ = {alpha_mean:.4f} au³")

# ============================================================
# Крок 5: Похідні поляризовності (Раман-активність)
# ============================================================
print("\n" + "=" * 70)
print("Крок 5: Обчислення Раман-активностей")
print("=" * 70)

print("\nОбчислення похідних поляризовності ∂α/∂Q...")
print("(Використовуємо чисельне диференціювання)")

# Параметри для чисельного диференціювання
delta = 0.01  # Зміщення в атомних одиницях (bohr)

raman_activities = []
depolarization_ratios = []

for mode_idx in range(len(real_freqs)):
    print(f"\rОбробка моди {mode_idx+1}/{len(real_freqs)}...", end="", flush=True)

    # Нормальна мода у декартових координатах
    mode = real_modes[:, mode_idx]

    # Зміщення вздовж нормальної моди
    coords_orig = mol.atom_coords()

    # +delta
    coords_plus = coords_orig + delta * mode.reshape(-1, 3)
    mol_plus = gto.M(
        atom=[(mol.atom_symbol(i), coords_plus[i]) for i in range(mol.natm)],
        basis=mol.basis,
        unit="bohr",
        symmetry=False,
        verbose=0,
    )
    mf_plus = dft.RKS(mol_plus)
    mf_plus.xc = mf.xc
    mf_plus.verbose = 0
```

```
    mf_plus.kernel()
    pol_plus = polarizability.rks.Polarizability(mf_plus)
    alpha_plus = pol_plus.polarizability()

    # -delta
    coords_minus = coords_orig - delta * mode.reshape(-1, 3)
    mol_minus = gto.M(
        atom=[(mol.atom_symbol(i), coords_minus[i]) for i in range(mol.natm)],
        basis=mol.basis,
        unit="bohr",
        symmetry=False,
        verbose=0,
    )
    mf_minus = dft.RKS(mol_minus)
    mf_minus.xc = mf.xc
    mf_minus.verbose = 0
    mf_minus.kernel()
    pol_minus = polarizability.rks.Polarizability(mf_minus)
    alpha_minus = pol_minus.polarizability()

    # Похідна: dα/dQ ≈ (α(+δ) - α(-δ)) / (2δ)
    dalpha_dQ = (alpha_plus - alpha_minus) / (2 * delta)

    # Раман-активність (середнє поле)
    # S = 45 * (dā/dQ)² + 7 * (dγ/dQ)²
    # де ā - середня поляризовність, γ - анізотропія

    dalpha_mean = np.trace(dalpha_dQ) / 3

    # Анізотропія похідної
    dalpha_aniso_sq = (
        (dalpha_dQ[0,0] - dalpha_dQ[1,1])**2 +
        (dalpha_dQ[1,1] - dalpha_dQ[2,2])**2 +
        (dalpha_dQ[2,2] - dalpha_dQ[0,0])**2 +
        6 * (dalpha_dQ[0,1]**2 + dalpha_dQ[0,2]**2 + dalpha_dQ[1,2]**2)
    ) / 2

    # Раман-активність (Å⁴/amu)
    S = 45 * dalpha_mean**2 + 7 * dalpha_aniso_sq
    raman_activities.append(S)

    # Деполяризаційне відношення (для неполяризованого світла)
    # ρ = 3γ² / (45ā² + 4γ²)
    if 45 * dalpha_mean**2 + 4 * dalpha_aniso_sq > 1e-10:
        rho = 3 * dalpha_aniso_sq / (45 * dalpha_mean**2 + 4 * dalpha_aniso_sq)
    else:
        rho = 0.0
    depolarization_ratios.append(rho)

print("\n")

raman_activities = np.array(raman_activities)
depolarization_ratios = np.array(depolarization_ratios)

# ============================================================
# Крок 6: Аналіз результатів
# ============================================================
print("\n" + "=" * 70)
print("Крок 6: Аналіз Раман-спектру")
print("=" * 70)

print("\nРаман-активні моди бензолу:")
print("-" * 70)
print(f"{'№':<5} {'ν (см⁻¹)':<12} {'Активність':<15} {'ρ':<10} {'Тип'}")
```

```
print("-" * 70)

# Класифікація мод за деполяризаційним відношенням
for i, (freq, activity, rho) in enumerate(
    zip(real_freqs, raman_activities, depolarization_ratios), 1
):
    # Класифікація
    if activity > 10:  # Значуща активність
        if rho < 0.1:
            mode_type = "Повністю симетрична"
        elif 0.1 ⩽ rho < 0.5:
            mode_type = "Симетрична"
        else:
            mode_type = "Деполяризована"

        print(f"{i:<5} {freq:>10.1f}  {activity:>12.2f}  {rho:>8.4f}  {mode_type}")

# Найінтенсивніші моди
print("\nНайінтенсивніші Раман-лінії:")
print("-" * 70)
top_indices = np.argsort(raman_activities)[-5:][::-1]
for idx in top_indices:
    freq = real_freqs[idx]
    activity = raman_activities[idx]
    rho = depolarization_ratios[idx]
    print(f"ν = {freq:7.1f} см⁻¹, S = {activity:8.2f}, ρ = {rho:.4f}")

# ============================================================
# Крок 7: Візуалізація спектру
# ============================================================
print("\n" + "=" * 70)
print("Крок 7: Побудова Раман-спектру")
print("=" * 70)

# Сітка частот
freq_grid = np.linspace(0, 3500, 3500)

# Параметр уширення (FWHM)
gamma = 10  # см⁻¹

def lorentzian(x, x0, gamma, intensity):
    """Лоренцева функція"""
    return intensity * (gamma/2)**2 / ((x - x0)**2 + (gamma/2)**2)

# Будуємо спектр
spectrum = np.zeros_like(freq_grid)
for freq, activity in zip(real_freqs, raman_activities):
    if activity > 1:  # Тільки значущі
        spectrum += lorentzian(freq_grid, freq, gamma, activity)

# Візуалізація
fig, axes = plt.subplots(3, 1, figsize=(12, 10))

# 1. Штрих-спектр
ax1 = axes[0]
for freq, activity in zip(real_freqs, raman_activities):
    if activity > 1:
        ax1.stem([freq], [activity], linefmt='b-', markerfmt='bo', basefmt=' ')
ax1.set_xlim(0, 3500)
ax1.set_ylabel('Раман-активність (Ų/amu)', fontsize=11)
ax1.set_title('Раман-спектр бензолу C₆H₆ (штрих-спектр)', fontsize=13, fontweight='bold')
ax1.grid(alpha=0.3)

# 2. Уширений спектр
```

```
ax2 = axes[1]
ax2.plot(freq_grid, spectrum, 'b-', linewidth=2)
ax2.set_xlim(0, 3500)
ax2.set_ylabel('Інтенсивність (відн. од.)', fontsize=11)
ax2.set_title('Уширений спектр (γ=10 см⁻¹)', fontsize=13)
ax2.grid(alpha=0.3)

# Позначки характерних смуг
characteristic_peaks = [
    (992, "Дихання кільця (A₁g)"),
    (1606, "C=C розтяг (E₂g)"),
    (3064, "C-H розтяг (E₂g)"),
]
for freq, label in characteristic_peaks:
    if freq < 3500:
        ax2.axvline(freq, color='r', linestyle='--', alpha=0.5)
        ax2.text(freq, ax2.get_ylim()[1]*0.9, label,
                rotation=90, verticalalignment='top', fontsize=9)

# 3. Деполяризаційні відношення
ax3 = axes[2]
colors = ['green' if rho < 0.1 else 'orange' if rho < 0.5 else 'red'
          for rho in depolarization_ratios]
for freq, rho, color, activity in zip(real_freqs, depolarization_ratios,
                                      colors, raman_activities):
    if activity > 1:
        ax3.scatter([freq], [rho], c=color, s=activity*2, alpha=0.7)

ax3.axhline(0.75, color='k', linestyle=':', alpha=0.5, label='ρ_max (деполяризована)')
ax3.axhline(0.0, color='k', linestyle=':', alpha=0.5, label='ρ=0 (повна симетрія)')
ax3.set_xlim(0, 3500)
ax3.set_ylim(-0.1, 1.0)
ax3.set_xlabel('Частота (см⁻¹)', fontsize=11)
ax3.set_ylabel('Деполяризаційне відношення ρ', fontsize=11)
ax3.set_title('Класифікація мод за симетрією', fontsize=13)
ax3.grid(alpha=0.3)
ax3.legend(fontsize=9)

plt.tight_layout()
plt.savefig('benzene_raman_spectrum.pdf', dpi=300, bbox_inches='tight')
print("\nСпектр збережено у файл: benzene_raman_spectrum.pdf")

# ============================================================
# Крок 8: Порівняння з експериментом
# ============================================================
print("\n" + "=" * 70)
print("Крок 8: Порівняння з експериментом")
print("=" * 70)

experimental_data = [
    (606, "A₂g (out-of-plane)"),
    (992, "A₁g (ring breathing)", "⁻"),
    (1178, "E₂g (C-H bend)"),
    (1596, "E₂g (C=C stretch)", "⁻"),
    (3047, "A₁g (C-H stretch)"),
    (3064, "E₂g (C-H stretch)", "⁻"),
]

print("\nХарактерні експериментальні частоти бензолу:")
print("-" * 70)
print(f"{'ν_exp (см⁻¹)':<15} {'Присвоєння':<30} {'Інтенсивність'}")
print("-" * 70)

for data in experimental_data:
```

```
    freq_exp = data[0]
    assignment = data[1]
    intensity = data[2] if len(data) > 2 else ""
    print(f"{freq_exp:<15} {assignment:<30} {intensity}")

print("\nЛегенда: ⎚ = сильна лінія")

print("\n" + "=" * 70)
print("ВИСНОВКИ")
print("=" * 70)

print("""
1. Раман-спектр бензолу:
   - Характеризується високою симетрією (D₆h)
   - Найінтенсивніша лінія: ~992 см⎚¹ (дихання кільця, A₁g)
   - Режим E₂g при ~1606 см⎚¹ (розтяг C=C)
   - C-H розтяги у області 3000-3100 см⎚¹

2. Симетрія та правила відбору:
   - Повністю симетричні моди (ρ≈0): A₁g
   - Деполяризовані моди (ρ≈0.75): E₁g, E₂g
   - Неактивні в Раман: A₂u, E₁u (ІЧ-активні)

3. Порівняння з експериментом:
   - B3LYP/6-311G** завищує частоти на ~5-10%
   - Для точності потрібно масштабувати на ~0.96
   - Відносні інтенсивності добре відтворюються

4. Рекомендації:
   - Для високої точності: використовуйте cc-pVTZ або більший базис
   - Враховуйте ангармонічні ефекти для C-H розтягів
   - Для твердого стану: додайте кристалічні ефекти
""")

print("=" * 70)
print("Розрахунок завершено!")
print("=" * 70)
```

Вибрані моди бензену C_6H_6 (D_{6h}):

Мода	Симетрія	Частота ($см^{-1}$)	ІЧ	Раман
ν_1	A_{1g}	993	Неактивна	Активна
ν_2	A_{1g}	3062	Неактивна	Активна (сильна)
ν_6	E_{2g}	608	Неактивна	Активна
ν_{18}	E_{1u}	1038	Активна	Неактивна
ν_{19}	E_{1u}	3080	Активна	Неактивна

Розрахунок: B3LYP/6-311+G(2d,p)

Характерні Раман смуги:

- 993 $см^{-1}$: дихальна мода кільця (ring breathing)
- 3062 $см^{-1}$: симетричне валентне C–H
- 608 $см^{-1}$: деформація кільця

Рис. 1.1. Раман спектр бензену демонструє принцип взаємного виключення. Симетричні моди (g) активні в Раман, але неактивні в ІЧ.

Порівняння ІЧ та Раман

Властивість	**ІЧ спектроскопія**	**Раман спектроскопія**
Правило відбору	$\frac{d\mu}{dQ} \neq 0$	$\frac{d\alpha}{dQ} \neq 0$
Джерело	ІЧ лампа	Лазер (видиме/УФ)
Зразок	Розчин, плівка, KBr	Розчин, кристал
Вода	Сильно поглинає	Слабкий сигнал
Скло	Непрозоре	Прозоре
Полярні групи	Сильний сигнал	Слабкий
Неполярні	Слабкий	Сильний сигнал
C=C, C≡C	Слабкі/неактивні	Сильні
C=O, O–H	Дуже сильні	Слабкі

1.1.5. Термохімія та нульова енергія

Використовуючи частоти коливань, можна обчислити термодинамічні функції в наближенні гармонічного осцилятора та жорсткого ротатора.

Нульова коливальна енергія (ZPE)

Навіть при $T = 0$ K молекула має енергію коливань:

$$E_{\text{ZPE}} = \sum_{i=1}^{3N-6} \frac{1}{2}\hbar\omega_i$$

Важливість ZPE:

- Для точних термохімічних розрахунків
- Ізотопні ефекти (HD vs H_2)
- Тунелювання через бар'єри
- Типово 5–15 ккал/моль для органічних молекул

Для H_2O:

$$E_{\text{ZPE}} \approx \frac{1}{2}(3657 + 1595 + 3756)\ \text{см}^{-1} \times 1.439\ \text{ккал/(моль·см}^{-1}) \approx 13.3\ \text{ккал/моль}$$

Термохімічні поправки

При температурі T коливальний внесок:

$$E_{\text{vib}}(T) = \sum_k \frac{\hbar\omega_k}{\exp(\hbar\omega_k/k_B T) - 1}.$$

Повна енергія при температурі T:

$$E_{\text{total}}(T) = E_{\text{elec}} + E_{\text{ZPE}} + E_{\text{vib}}(T) + E_{\text{rot}}(T) + E_{\text{trans}}(T)$$

Ентальпія:

$$H(T) = E_{\text{total}}(T) + RT$$

Ентропія:

$$S(T) = S_{\text{trans}} + S_{\text{rot}} + S_{\text{vib}} + S_{\text{elec}}$$

Вільна енергія Гіббса:

$$G(T) = H(T) - TS(T)$$

Термохімічний розрахунок для H_2O

h2o_thermochemistry.py

```python
# ============================================================
# h2o_thermochemistry.py
# Термохімічні поправки (ZPE, ентальпія, ентропія)
# ============================================================

from pyscf import gto, scf
from pyscf.hessian import thermo

mol = gto.M(
    atom="""
    O  0.0000  0.0000  0.1173
    H  0.0000  0.7572 -0.4692
    H  0.0000 -0.7572 -0.4692
    """,
    basis="6-31g",
    unit="angstrom",
)

print("Термохімічний аналіз H2O")
print("=" * 60)

# SCF розрахунок
mf = scf.RHF(mol)
e_elec = mf.kernel()

# Гесіан та частоти
hess = mf.Hessian()
h = hess.kernel()

# Термохімічні функції при T=298.15 K, p=1 atm
results = thermo.thermo(mf, h, 298.15, 101325)

print("\nЕлектронна енергія:")
print(f"E(elec) = {e_elec:.6f} Ha")
print(f"        = {e_elec * 627.509:.2f} ккал/моль")

print("\nПоправки при 298.15 K:")
print(f"Нульова коливальна енергія (ZPE): {results['ZPE']:.6f} Ha")
print(f"Термічна поправка до енергії:     {results['E_thermal']:.6f} Ha")
print(f"Термічна поправка до ентальпії:   {results['H_thermal']:.6f} Ha")

print("\nПовна енергія Гіббса:")
print(f"G(298K) = E(elec) + ZPE + H_thermal - T*S")
```

```
print(f"        = {results['G_total']:.6f} Ha")

print("\nЕнтропія:")
print(f"S = {results['S']:.3f} кал/(моль·K)")

print("\nРозклад ZPE по модах:")
for i, freq in enumerate(results['freqs']):
    if freq > 100:
        zpe_mode = 0.5 * freq * 1.4388  # см⁻¹ -> ккал/моль
        print(f"  Мода {i+1}: {freq:.1f} см⁻¹ → ZPE = {zpe_mode:.2f} ккал/моль")


# ============================================================
# h2co_tddft.py
# Розрахунок УФ-спектру формальдегіду методом TDDFT
# ============================================================

from pyscf import gto, scf, dft, tddft

# Молекула формальдегіду H2CO
mol = gto.M(
    atom="""
    C  0.0000  0.0000  0.0000
    O  0.0000  0.0000  1.2050
    H  0.0000  0.9428 -0.5876
    H  0.0000 -0.9428 -0.5876
    """,
    basis="aug-cc-pvdz",
    unit="angstrom",
)

print("TDDFT розрахунок для H2CO (формальдегід)")
print("=" * 60)

# Основний стан: DFT з функціоналом CAM-B3LYP
mf = dft.RKS(mol)
mf.xc = "cam-b3lyp"  # Range-separated функціонал
mf.kernel()

print(f"\nЕнергія основного стану: {mf.e_tot:.6f} Ha")

# TDDFT для збуджених станів
# Розраховуємо перші 5 синглетних збуджень
td = tddft.TDDFT(mf)
td.nstates = 5
td.kernel()

print("\nЗбуджені стани (синглети):")
print("-" * 80)
print(f"{'Стан':<8} {'ΔE (eV)':<12} {'λ (нм)':<12} {'f':<12} {'Характер'}")
print("-" * 80)

# Конвертуємо Hartree -> eV -> nm
au2ev = 27.2114
for i, e in enumerate(td.e):
    energy_ev = e * au2ev
    wavelength = 1240 / energy_ev  # eV -> nm
    osc_str = td.oscillator_strength()[i]

    # Простий аналіз характеру
    if i == 0:
        char = "n→π*"
    elif i == 1:
        char = "π→π*"
```

```
    else:
        char = "Рідберг/змішаний"

    print(f"S_{i+1:<6} {energy_ev:>10.3f}  {wavelength:>10.1f}  "
          f"{osc_str:>10.4f}  {char}")

print("\nПримітка:")
print("- Сила осцилятора f показує інтенсивність переходу")
print("- n→π* перехід слабкий (заборонений за симетрією)")
print("- π→π* перехід сильний (дозволений)")
```

Типові поправки для H_2O при 298.15 K:

- Електронна енергія: $E_{\text{elec}} = -76.067$ Ha
- ZPE: $+0.021$ Ha (13.3 ккал/моль)
- Термічна поправка: $+0.003$ Ha (1.9 ккал/моль)
- Ентропія: $S = 45.1$ кал/(моль·K)

Термохімічний розрахунок для CH_4

ch4_thermochemistry.py

```
"""
Термохімічний розрахунок для метану CH₄
Демонструє обчислення ZPE, ентальпії, ентропії та вільної енергії Гіббса
"""
from pyscf import gto, scf, dft
from pyscf.hessian import rhf as rhf_hess
import numpy as np

print("=" * 70)
print("Термохімічний аналіз метану CH₄")
print("=" * 70)

# Геометрія метану (тетраедрична)
mol = gto.M(
    atom='''
    C    0.000000    0.000000    0.000000
    H    0.629118    0.629118    0.629118
    H   -0.629118   -0.629118    0.629118
    H   -0.629118    0.629118   -0.629118
    H    0.629118   -0.629118   -0.629118
    ''',
    basis='6-311+g(2d,p)',
    symmetry=True,
    unit='angstrom'
)

print("\nМолекулярна геометрія:")
print("  Симетрія: Td (тетраедрична)")
print("  r(C-H) = 1.089 Å")
print("  ∠(H-C-H) = 109.47°")

# B3LYP розрахунок
print("\n\n1. B3LYP/6-311+G(2d,p)")
print("-" * 70)
mf = dft.RKS(mol)
mf.xc = 'b3lyp'
mf.kernel()
```

```
E_elec = mf.e_tot
print(f"Електронна енергія: {E_elec:.8f} Hartree")
print(f"                    {E_elec * 627.509:.2f} kcal/mol")

# Розрахунок Гессіану
print("\n\n2. Розрахунок коливальних частот")
print("-" * 70)
print("Обчислення матриці Гессіану...")

hess = rhf_hess.Hessian(mf)
h = hess.kernel()

# Перетворення Гессіану
h_reshape = h.transpose(0, 2, 1, 3).reshape(mol.natm * 3, mol.natm * 3)

# Масо-зважений Гессіан
mass = []
atom_masses = mol.atom_mass_list()
for i in range(mol.natm):
    mass.extend([atom_masses[i]] * 3)
mass = np.array(mass)
mass_factor = np.sqrt(np.outer(mass, mass))
h_mw = h_reshape / mass_factor

# Діагоналізація
eigvals, eigvecs = np.linalg.eigh(h_mw)

# Переведення в см⁻¹
au_to_cm = 219474.63
freq_cm = np.sqrt(np.abs(eigvals)) * au_to_cm
freq_cm = np.where(eigvals < 0, -freq_cm, freq_cm)

# Відбір коливальних мод (частоти > 100 см⁻¹)
vib_freq = freq_cm[freq_cm > 100]
vib_freq = np.sort(vib_freq)

print(f"\nКількість коливальних мод: {len(vib_freq)}")
print("Очікується 3N-6 = 15-6 = 9 мод для нелінійної молекули")

print("\nКоливальні частоти (см⁻¹):")
print("-" * 70)
for i, freq in enumerate(vib_freq):
    print(f"  ν{i+1:2d}: {freq:7.1f} см⁻¹")

# Симетрійна класифікація для Td
print("\nСиметрійна класифікація (Td):")
print("-" * 70)
print("A₁:   ν₁ = 2917 см⁻¹  (симетричне валентне)")
print("E:    ν₂ = 1534 см⁻¹  (деформаційне, вироджене)")
print("F₂:   ν₃ = 3019 см⁻¹  (антисим. валентне, триразово вироджене)")
print("F₂:   ν₄ = 1306 см⁻¹  (деформаційне, триразово вироджене)")

# Розрахунок ZPE
print("\n\n3. Нульова коливальна енергія (ZPE)")
print("=" * 70)

# ZPE = (1/2) Σ ℏω
h_planck = 6.62607015e-34  # J·s
c_light = 299792458  # m/s
hartree_to_J = 4.3597447222071e-18  # J

ZPE_cm = 0.5 * np.sum(vib_freq)  # см⁻¹
ZPE_hartree = ZPE_cm / au_to_cm
ZPE_kcal = ZPE_hartree * 627.509
```

```
print(f"ZPE = {ZPE_cm:.2f} см⁻¹")
print(f"    = {ZPE_hartree:.6f} Hartree")
print(f"    = {ZPE_kcal:.2f} kcal/mol")
print(f"\nЕкспериментальне ZPE: 27.8 kcal/mol")
print(f"Похибка: {ZPE_kcal - 27.8:.2f} kcal/mol ({(ZPE_kcal - 27.8)/27.8 * 100:.1f}%)")

# Термохімічні поправки при 298.15 K
print("\n\n4. Термохімічні поправки при T = 298.15 K")
print("=" * 70)

T = 298.15  # K
k_B = 1.380649e-23  # J/K (Boltzmann constant)
R = 8.314462618  # J/(mol·K) (gas constant)
N_A = 6.02214076e23  # 1/mol (Avogadro)

# Переведення частот в Дж
freq_J = vib_freq * 100 * c_light * h_planck  # см⁻¹ → Дж

# Коливальна енергія
E_vib = 0
S_vib = 0
for freq_j in freq_J:
    x = freq_j / (k_B * T)
    E_vib += N_A * freq_j / (np.exp(x) - 1)
    S_vib += R * (x / (np.exp(x) - 1) - np.log(1 - np.exp(-x)))

E_vib_kcal = E_vib / 4184  # J/mol → kcal/mol
S_vib_cal = S_vib / 4.184  # J/(mol·K) → cal/(mol·K)

print(f"\nКоливальна енергія E_vib(T): {E_vib_kcal:.3f} kcal/mol")
print(f"Коливальна ентропія S_vib(T): {S_vib_cal:.2f} cal/(mol·K)")

# Обертальна енергія (нелінійна молекула)
E_rot = 1.5 * R * T / 1000  # kJ/mol
E_rot_kcal = E_rot / 4.184
print(f"\nОбертальна енергія E_rot(T): {E_rot_kcal:.3f} kcal/mol")

# Обертальна ентропія (Td симетрія, σ = 12)
# Для обертальної ентропії потрібні моменти інерції
# Спрощене обчислення для CH4
I_CH4 = 5.31e-47  # кг·м² (момент інерції CH4)
sigma = 12  # число симетрії для Td
S_rot = R * (1.5 + np.log((2 * np.pi * I_CH4 * k_B * T / h_planck**2)**(1.5) / sigma))
S_rot_cal = S_rot / 4.184

print(f"Обертальна ентропія S_rot(T): {S_rot_cal:.2f} cal/(mol·K)")

# Трансляційна енергія
E_trans = 1.5 * R * T / 1000  # kJ/mol
E_trans_kcal = E_trans / 4.184
print(f"\nТрансляційна енергія E_trans(T): {E_trans_kcal:.3f} kcal/mol")

# Трансляційна ентропія (Sackur-Tetrode)
M = 16.043  # g/mol (молярна маса CH4)
M_kg = M / 1000 / N_A  # кг
P = 101325  # Pa (1 atm)
S_trans = R * (2.5 + 1.5 * np.log(2 * np.pi * M_kg * k_B * T / h_planck**2) +
               np.log(k_B * T / P))
S_trans_cal = S_trans / 4.184

print(f"Трансляційна ентропія S_trans(T): {S_trans_cal:.2f} cal/(mol·K)")

# Електронна ентропія (основний стан синглет, S_elec = 0)
```

```
S_elec = 0
print(f"Електронна ентропія S_elec: {S_elec:.2f} cal/(mol·K)")

# Сумарні величини
print("\n\n5. Термодинамічні функції при 298.15 K")
print("=" * 70)

# Внутрішня енергія
U = E_elec * 627.509 + ZPE_kcal + E_vib_kcal + E_rot_kcal + E_trans_kcal
print(f"\nВнутрішня енергія U(T):")
print(f"  U = E_elec + ZPE + E_vib + E_rot + E_trans")
print(f"  U = {E_elec * 627.509:.2f} + {ZPE_kcal:.2f} + {E_vib_kcal:.3f} + {E_rot_kcal:.3f} +
↪  {E_trans_kcal:.3f}")
print(f"  U = {U:.2f} kcal/mol")

# Ентальпія
H = U + R * T / 4184  # додаємо PV = RT
print(f"\nЕнтальпія H(T):")
print(f"  H = U + RT")
print(f"  H = {H:.2f} kcal/mol")

# Поправка до ентальпії від 0 K
H_corr = ZPE_kcal + E_vib_kcal + E_rot_kcal + E_trans_kcal + R * T / 4184
print(f"\nТермічна поправка H(298) - H(0):")
print(f"  ΔH = {H_corr:.2f} kcal/mol")

# Ентропія
S_total = S_trans_cal + S_rot_cal + S_vib_cal + S_elec
print(f"\nЕнтропія S(T):")
print(f"  S = S_trans + S_rot + S_vib + S_elec")
print(f"  S = {S_trans_cal:.2f} + {S_rot_cal:.2f} + {S_vib_cal:.2f} + {S_elec:.2f}")
print(f"  S = {S_total:.2f} cal/(mol·K)")

# Вільна енергія Гіббса
G = H - T * S_total / 1000  # kcal/mol
print(f"\nВільна енергія Гіббса G(T):")
print(f"  G = H - TS")
print(f"  G = {H:.2f} - {T:.2f} × {S_total/1000:.5f}")
print(f"  G = {G:.2f} kcal/mol")

# Теплоємність
C_v = 3 * R / 4.184  # 3R для нелінійної молекули (наближення)
# Точніше: C_v = C_trans + C_rot + C_vib
C_v_vib = 0
for freq_j in freq_J:
    x = freq_j / (k_B * T)
    C_v_vib += R * x**2 * np.exp(x) / (np.exp(x) - 1)**2

C_v_total = (1.5 * R + 1.5 * R + C_v_vib) / 4.184
print(f"\nТеплоємність C_V(T):")
print(f"  C_V = C_trans + C_rot + C_vib")
print(f"  C_V = {C_v_total:.2f} cal/(mol·K)")

# Порівняння з експериментом
print("\n\n6. Порівняння з експериментом")
print("=" * 70)

exp_data = {
    'ZPE': 27.8,
    'H_corr': 2.48,
    'S': 44.5,
    'C_v': 6.0
}
```

```
print(f"\nВеличина                  Розрах.    Експ.     Δ")
print("-" * 70)
print(f"ZPE (kcal/mol)           {ZPE_kcal:6.2f}     {exp_data['ZPE']:6.2f}    {ZPE_kcal -
↪  exp_data['ZPE']:+.2f}")
print(f"H(298)-H(0) (kcal/mol)   {H_corr:6.2f}     {exp_data['H_corr']:6.2f}    {H_corr -
↪  exp_data['H_corr']:+.2f}")
print(f"S(298) (cal/mol·K)       {S_total:6.1f}     {exp_data['S']:6.1f}    {S_total -
↪  exp_data['S']:+.1f}")
print(f"C_V(298) (cal/mol·K)     {C_v_total:6.1f}     {exp_data['C_v']:6.1f}    {C_v_total -
↪  exp_data['C_v']:+.1f}")

# Застосування
print("\n\n7. Застосування термохімії")
print("=" * 70)

print("\n1. Розрахунок енергії реакції:")
print("   ΔG_rxn = Σ G_products - Σ G_reactants")
print("   Важливо включати ZPE та термічні поправки!")

print("\n2. Константа рівноваги:")
print("   K_eq = exp(-ΔG°/RT)")
print("   ΔG° визначає положення рівноваги")

print("\n3. Швидкість реакції (рівняння Ейрінга):")
print("   k = (k_B T / h) exp(-ΔG‡/RT)")
print("   ΔG‡ - вільна енергія активації")

print("\n4. Ізотопні ефекти:")
print("   Заміна H → D змінює ZPE")
print("   Впливає на швидкості реакцій")

# Методологічні поради
print("\n\n8. Методологічні рекомендації")
print("=" * 70)

print("\n✓ Для термохімії:")
print("  • Базис: 6-311+G(2d,p) або aug-cc-pVTZ")
print("  • Метод: B3LYP (швидко), ωB97X-D (точніше)")
print("  • Обов'язково: оптимізація + перевірка на мінімум")
print("  • Масштабуйте частоти (λ ≈ 0.968 для B3LYP)")

print("\n✓ Точність:")
print("  • ZPE: ±0.5 kcal/mol (3-5% похибка)")
print("  • ΔH: ±1 kcal/mol для малих молекул")
print("  • ΔG: ±2 kcal/mol (ентропія менш точна)")

print("\n✓ Пастки:")
print("  • Не забувайте ZPE (може бути 10+ kcal/mol)")
print("  • Перевіряйте на уявні частоти")
print("  • Для конформерів: Больцманівське усереднення")

print("\n" + "=" * 70)
print("ВИСНОВКИ:")
print("=" * 70)
print("• ZPE становить ~28 kcal/mol для CH₄ (не можна ігнорувати!)")
print("• Термічна поправка H(298)-H(0) ≈ 2.5 kcal/mol")
print("• Ентропія важлива для ΔG (T·S може бути значним)")
print("• B3LYP дає хорошу згоду з експериментом")
print("• Термохімія критична для розрахунків реакцій")
print("=" * 70)
```

Результати для CH_4 при 298.15 K:

Величина	Значення	Одиниці
ZPE	28.03	ккал/моль
$E_{\text{vib}}(T)$	0.10	ккал/моль
$E_{\text{rot}}(T)$	0.89	ккал/моль
$E_{\text{trans}}(T)$	0.89	ккал/моль
$H(T) - H(0)$	2.48	ккал/моль
$S(T)$	44.5	кал/(моль·K)
C_V	6.0	кал/(моль·K)

Розрахунок: B3LYP/6-311+G(2d,p)

Порівняння з експериментом:

- ZPE (експ.): 27.8 ккал/моль — відмінна згода
- S_{298} (експ.): 44.5 кал/(моль·K) — ідеальна згода
- Термохімічні поправки надійні для DFT

1.1.6. Ізотопні ефекти

Заміна ізотопу змінює частоти через зміну маси:

$$\frac{\omega_{\text{D}}}{\omega_{\text{H}}} \approx \sqrt{\frac{m_{\text{H}}}{m_{\text{D}}}} = \sqrt{\frac{1}{2}} \approx 0.707$$

Ізотопний ефект у H_2O/D_2O

Порівняння H_2O та D_2O:

Мода	H_2O	D_2O	Співвідношення
	($см^{-1}$)	($см^{-1}$)	
Симетричне валентне	3657	2671	0.730
Деформаційне	1595	1178	0.738
Антисим. валентне	3756	2788	0.742

Експериментальні дані

ZPE ефект:

- ZPE(H_2O): 13.26 ккал/моль
- ZPE(D_2O): 9.66 ккал/моль
- ΔZPE: 3.60 ккал/моль

Наслідки:

- Дейтеровані сполуки стабільніші (нижча ZPE)
- Кінетичний ізотопний ефект: $k_{\text{H}}/k_{\text{D}} \approx 7$ для розриву C–H
- Використання в механістичних дослідженнях

1.1.7. Перехідні стани та уявні частоти

Характеристика стаціонарних точок

Гессіан дозволяє класифікувати стаціонарні точки:

Тип	Уявних частот	Характеристика
Мінімум	0	Локальний мінімум
Перехідний стан	1	Сідлова точка 1-го порядку
Сідло 2-го порядку	2	Нестабільна структура

Уявна частота: $\omega^2 < 0 \Rightarrow \omega = i|\omega|$
У виводі PySCF позначається як негативна частота.

Приклад: інверсія аміаку

Профіль енергії інверсії NH_3:

Структура	E_{rel} (ккал/моль)	Уявних частот	Тип
Пірамідальна (C_{3v})	0.0	0	Мінімум
Планарна (D_{3h})	5.8	1	ПС
Пірамідальна (інша)	0.0	0	Мінімум

Рис. 1.2. Профіль енергії інверсії NH_3. Планарна структура є перехідним станом з однією уявною частотою (зонтичний рух).

Уявна мода в ПС:
- Частота: $\omega = i1044$ см$^{-1}$ (або $-1044i$ в PySCF)
- Напрямок: зонтичний рух (umbrella mode)
- Зв'язує два еквівалентні мінімуми
- Квантове тунелювання → розщеплення ЯМР сигналу

Інтрінсична координата реакції (IRC)

Від перехідного стану можна простежити шлях реакції:
- Рух вздовж уявної моди
- З'єднує реагенти та продукти
- Доводить, що ПС належить до шуканої реакції

1.1.8. Ангармонічні поправки

Гармонічне наближення добре для малих амплітуд, але:
- Завищує частоти на 3–5%
- Не описує обертонів та комбінаційних смуг
- Не враховує асиметрію потенціалу

Ангармонічний потенціал

Розклад потенціалу до кубічних та квартичних термів:

$$V = V_0 + \frac{1}{2}\sum_{ij} k_{ij} Q_i Q_j + \frac{1}{6}\sum_{ijk} k_{ijk} Q_i Q_j Q_k + \frac{1}{24}\sum_{ijkl} k_{ijkl} Q_i Q_j Q_k Q_l$$

Методи розрахунку:
- VPT2 (Vibrational Perturbation Theory 2-го порядку)
- VSCF (Vibrational Self-Consistent Field)
- VCI (Vibrational Configuration Interaction)

Ангармонічні частоти H_2O

Мода	Гарм.	VPT2	Масштаб.	Експ.
	($см^{-1}$)	($см^{-1}$)	($см^{-1}$)	($см^{-1}$)
ν_1	3825	3707	3702	3657
ν_2	1653	1618	1600	1595
ν_3	3936	3814	3808	3756

B3LYP/aug-cc-pVTZ, масштабувальний фактор 0.968

Висновки:
- VPT2 суттєво покращує згоду з експериментом
- Масштабування простіше, але менш фізичне
- Для валентних коливань ангармонізм $\sim 100 - 150$ $см^{-1}$

1.1.9. Практичні рекомендації

Вибір методу та базису

1.2. Електронні переходи та УФ-видимі спектри

1.2.1. Теорія збуджених станів

Для розрахунку електронних переходів використовуємо методи:
- **CIS** (Configuration Interaction Singles) — базовий
- **TDHF/TDDFT** (Time-Dependent HF/DFT) — точніший

- **EOM-CCSD** (Equation-of-Motion CCSD) — високоточний

  Енергія збудження n-го стану:

$$\omega_n = E_n - E_0$$

  Сила осцилятора (інтенсивність):

$$f_n = \frac{2}{3}\omega_n |\langle \Psi_0 | \hat{\mu} | \Psi_n \rangle|^2$$

1.2.2. TDDFT розрахунок для формальдегіду

Розглянемо молекулу формальдегіду H_2CO як приклад:

h2co_tddft.py

```python
# ============================================================
# h2co_tddft.py
# Розрахунок УФ-спектру формальдегіду методом TDDFT
# ============================================================

from pyscf import gto, scf, dft, tddft

# Молекула формальдегіду H2CO
mol = gto.M(
    atom="""
    C  0.0000  0.0000  0.0000
    O  0.0000  0.0000  1.2050
    H  0.0000  0.9428 -0.5876
    H  0.0000 -0.9428 -0.5876
    """,
    basis="aug-cc-pvdz",
    unit="angstrom",
)

print("TDDFT розрахунок для H2CO (формальдегід)")
print("=" * 60)

# Основний стан: DFT з функціоналом CAM-B3LYP
mf = dft.RKS(mol)
mf.xc = "cam-b3lyp"  # Range-separated функціонал
mf.kernel()

print(f"\nЕнергія основного стану: {mf.e_tot:.6f} Ha")

# TDDFT для збуджених станів
# Розраховуємо перші 5 синглетних збуджень
td = tddft.TDDFT(mf)
td.nstates = 5
td.kernel()

print("\nЗбуджені стани (синглети):")
print("-" * 80)
print(f"{'Стан':<8} {'ΔE (eV)':<12} {'λ (нм)':<12} {'f':<12} {'Характер'}")
print("-" * 80)

# Конвертуємо Hartree -> eV -> nm
au2ev = 27.2114
for i, e in enumerate(td.e):
    energy_ev = e * au2ev
    wavelength = 1240 / energy_ev  # eV -> nm
```

```
    osc_str = td.oscillator_strength()[i]

    # Простий аналіз характеру
    if i == 0:
        char = "n→π*"
    elif i == 1:
        char = "π→π*"
    else:
        char = "Рідберг/змішаний"

    print(f"S_{i+1:<6} {energy_ev:>10.3f}  {wavelength:>10.1f}  "
          f"{osc_str:>10.4f}  {char}")

print("\nПримітка:")
print("- Сила осцилятора f показує інтенсивність переходу")
print("- n→π* перехід слабкий (заборонений за симетрією)")
print("- π→π* перехід сильний (дозволений)")
```

Аналіз збуджень для H_2CO:

Перехід	Тип	λ (нм)	f	Характер
S_1	$n \to \pi^*$	355	0.001	Заборонений
S_2	$\pi \to \pi^*$	185	0.152	Дозволений
S_3	$n \to 3s$	172	0.021	Рідбергівський

Фізична інтерпретація:

- $n \to \pi^*$: неподілена пара O → антизв'язуюча МО C=O
- Низька сила осцилятора через симетрію
- $\pi \to \pi^*$: основна смуга поглинання в УФ
- Енергії залежать від функціоналу DFT

1.2.3. Вибір функціоналу для TDDFT

Різні функціонали дають різну точність для збуджень:

formaldehyde_functional_comparison.py

```
# ============================================================
# formaldehyde_functional_comparison.py
# Порівняння різних функціоналів для TDDFT
# ============================================================

from pyscf import gto, dft, tddft

mol = gto.M(
    atom="""
    C  0.0000  0.0000  0.0000
    O  0.0000  0.0000  1.2050
    H  0.0000  0.9428 -0.5876
    H  0.0000 -0.9428 -0.5876
    """,
    basis="aug-cc-pvdz",
    unit="angstrom",
)
```

```python
print("Порівняння функціоналів DFT для збуджень H2CO")
print("=" * 60)

# Список функціоналів для тестування
functionals = ["b3lyp", "pbe0", "cam-b3lyp", "wb97x-d"]

results = {}

for xc in functionals:
    print(f"\n{xc.upper()}:")
    print("-" * 40)

    # DFT розрахунок
    mf = dft.RKS(mol)
    mf.xc = xc
    mf.verbose = 0
    mf.kernel()

    # TDDFT
    td = tddft.TDDFT(mf)
    td.nstates = 3
    td.verbose = 0
    td.kernel()

    # Перший перехід (n→π*)
    energy_ev = td.e[0] * 27.2114
    wavelength = 1240 / energy_ev
    osc_str = td.oscillator_strength()[0]

    results[xc] = (energy_ev, wavelength, osc_str)

    print(f"  S₁: {energy_ev:.3f} eV ({wavelength:.1f} nm)")
    print(f"  f = {osc_str:.4f}")

print("\n" + "=" * 60)
print("Порівняння для n→π* переходу:")
print("-" * 60)
print(f"{'Функціонал':<15} {'λ (нм)':<12} {'Відхилення'}")
print("-" * 60)

exp_wavelength = 330  # Експериментальне значення (нм)

for xc, (e, wl, f) in results.items():
    diff = wl - exp_wavelength
    print(f"{xc.upper():<15} {wl:>10.1f}  {diff:+7.1f} нм")

print("-" * 60)
print(f"{'ЕКСПЕРИМЕНТ':<15} {exp_wavelength:>10.1f}  {'---'}")

print("\nВисновки:")
print("- Range-separated функціонали (CAM-B3LYP, ωB97X-D) найточніші")
print("- B3LYP завищує довжини хвиль")
print("- Для переносу заряду обов'язково використовувати range-separated")
```

Порівняння функціоналів (перехід $n \to \pi^*$):

Функціонал	λ (нм)	Похибка (нм)
B3LYP	355	+25
PBE0	342	+12
CAM-B3LYP	335	+5
ωB97X-D	332	+2
Експеримент	330	—

Рекомендації:

- Гібридні функціонали краще за чисті GGA
- Range-separated (CAM-B3LYP, ωB97X-D) найточніші
- Для переносу заряду обов'язково range-separated
- B3LYP часто завищує довжини хвиль

1.2.4. Візуалізація спектру

Для побудови спектру використовуємо гаусові або лоренцеві контури:

$$\varepsilon(\lambda) = \sum_n f_n \cdot \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(\lambda - \lambda_n)^2}{2\sigma^2}\right]$$

plot_uv_spectrum.py

```python
# ============================================================
# plot_uv_spectrum.py
# Побудова УФ-спектру з уширенням
# ============================================================

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

mol = gto.M(
    atom="""
    C  0.0000  0.0000  0.0000
    O  0.0000  0.0000  1.2050
    H  0.0000  0.9428 -0.5876
    H  0.0000 -0.9428 -0.5876
    """,
    basis="aug-cc-pvdz",
    unit="angstrom",
)

# TDDFT розрахунок
mf = dft.RKS(mol)
mf.xc = "cam-b3lyp"
mf.verbose = 0
mf.kernel()

td = tddft.TDDFT(mf)
td.nstates = 10
td.verbose = 0
td.kernel()

# Отримуємо енергії та сили осциляторів
energies = td.e * 27.2114  # Ha -> eV
```

```python
wavelengths = 1240 / energies  # eV -> nm
osc_strengths = td.oscillator_strength()

print("Побудова УФ-спектру")
print("=" * 60)
print("\nЗбудження:")
for i, (wl, f) in enumerate(zip(wavelengths, osc_strengths)):
    if f > 0.001:  # Тільки значущі переходи
        print(f"S_{i+1}: λ={wl:.1f} nm, f={f:.4f}")

# Будуємо спектр з гаусовим уширенням
def gaussian(x, center, sigma):
    return np.exp(-(x - center)**2 / (2 * sigma**2)) / (sigma * np.sqrt(2*np.pi))

# Сітка довжин хвиль
wl_grid = np.linspace(150, 400, 1000)
spectrum = np.zeros_like(wl_grid)

# Параметр уширення
sigma = 10  # nm (для газової фази)

# Додаємо кожний перехід
for wl, f in zip(wavelengths, osc_strengths):
    if 150 < wl < 400:  # Тільки в УФ області
        spectrum += f * gaussian(wl_grid, wl, sigma)

# Малюємо
plt.figure(figsize=(10, 6))

# Штрих-спектр (окремі переходи)
plt.subplot(2, 1, 1)
for wl, f in zip(wavelengths, osc_strengths):
    if 150 < wl < 400 and f > 0.001:
        plt.stem([wl], [f], linefmt='b-', markerfmt='bo', basefmt=' ')
plt.xlim(150, 400)
plt.ylabel('Сила осцилятора')
plt.title('H₂CO: Штрих-спектр (TDDFT/CAM-B3LYP)')
plt.grid(alpha=0.3)

# Уширений спектр
plt.subplot(2, 1, 2)
plt.plot(wl_grid, spectrum, 'b-', linewidth=2)
plt.xlim(150, 400)
plt.xlabel('Довжина хвилі (нм)')
plt.ylabel('Інтенсивність (відн. од.)')
plt.title('Уширений спектр (σ=10 нм)')
plt.grid(alpha=0.3)

plt.tight_layout()
plt.savefig('h2co_uv_spectrum.pdf', dpi=300, bbox_inches='tight')
print("\nСпектр збережено у файл h2co_uv_spectrum.pdf")
```

Рис. 1.3. УФ-спектр формальдегіду, розрахований методом TDDFT/CAM-B3LYP/aug-cc-pVDZ.

Параметри уширення:

- Типове $\sigma = 0.3$–0.5 eV для конденсованої фази
- $\sigma = 0.1$–0.2 eV для газової фази

- Експериментальне уширення враховує розподіл за T

1.2.5. Аналіз характеру переходів

TDDFT надає інформацію про орбіталі, задіяні у переході:

analyze_transitions.py

```python
# ============================================================
# analyze_transitions.py
# Детальний аналіз характеру електронних переходів
# ============================================================

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

mol = gto.M(
    atom="""
    C  0.0000  0.0000  0.0000
    O  0.0000  0.0000  1.2050
    H  0.0000  0.9428 -0.5876
    H  0.0000 -0.9428 -0.5876
    """,
    basis="aug-cc-pvdz",
    unit="angstrom",
)

print("Детальний аналіз переходів H2CO")
print("=" * 60)

# DFT + TDDFT
mf = dft.RKS(mol)
mf.xc = "cam-b3lyp"
mf.verbose = 0
mf.kernel()

td = tddft.TDDFT(mf)
td.nstates = 5
td.verbose = 0
td.kernel()

# Аналізуємо кожен збуджений стан
for i in range(min(3, td.nstates)):  # Перші 3 стани
    energy_ev = td.e[i] * 27.2114
    wavelength = 1240 / energy_ev
    f = td.oscillator_strength()[i]

    print(f"\n{'='*60}")
    print(f"Збуджений стан {i+1}: {energy_ev:.3f} eV ({wavelength:.1f} nm)")
    print(f"Сила осцилятора f = {f:.4f}")
    print(f"{'='*60}")

    # Отримуємо амплітуди переходів X → Y
    # td.xy[i] містить (X, Y) амплітуди
    x_amp = td.xy[i][0]  # Амплітуди збудження

    # Індекси орбіталей
    nocc = mol.nelectron // 2  # Кількість зайнятих орбіталей

    print("\nОсновні внески (|амплітуда| > 0.1):")
    print(f"{'Перехід':<20} {'Амплітуда':<12} {'Внесок (%)'}")
    print("-" * 50)

    # Знаходимо значущі внески
```

```
    nvir = len(x_amp[0])  # Кількість віртуальних

    contributions = []
    for occ_i in range(nocc):
        for vir_i in range(nvir):
            amp = x_amp[occ_i, vir_i]
            contrib = amp**2 * 100  # У відсотках

            if abs(amp) > 0.1:
                homo_label = occ_i - nocc  # -1 для HOMO, -2 для HOMO-1, ...
                lumo_label = vir_i  # 0 для LUMO, 1 для LUMO+1, ...

                if homo_label == -1:
                    orb_from = "HOMO"
                elif homo_label < -1:
                    orb_from = f"HOMO{homo_label+1}"
                else:
                    orb_from = f"Occ{occ_i}"

                if lumo_label == 0:
                    orb_to = "LUMO"
                else:
                    orb_to = f"LUMO+{lumo_label}"

                transition = f"{orb_from} → {orb_to}"
                contributions.append((transition, amp, contrib))

    # Сортуємо за внеском
    contributions.sort(key=lambda x: x[2], reverse=True)

    for trans, amp, contrib in contributions[:5]:  # Топ-5
        print(f"{trans:<20} {amp:>10.4f}  {contrib:>10.1f}")

    # Інтерпретація
    if contributions:
        main_trans = contributions[0][0]
        print(f"\nДомінуючий перехід: {main_trans}")

        if "HOMO" in main_trans and "LUMO" in main_trans:
            print("Тип: одноелектронне збудження")
            if i == 0:
                print("Характер: n→π* (неподілена пара O → π* C=O)")
            elif i == 1:
                print("Характер: π→π* (зв'язуюча → антизв'язуюча)")

print("\n" + "="*60)
print("Примітка: Внески показують ймовірність кожного переходу")
print("Сума квадратів амплітуд дорівнює 1")
```

Приклад виводу для S_1 стану H_2CO:

```
Excited State 1: 3.492 eV (355 nm)  f=0.0012
    HOMO-1 -> LUMO     0.11 (1.2%)
    HOMO   -> LUMO     0.69 (47.6%)
    HOMO   -> LUMO+1   0.08 (0.6%)
```

Інтерпретація:

- Основний внесок: HOMO → LUMO (48%)
- HOMO — неподілена пара n(O)

- LUMO — антизв'язуюча π^*(C=O)
- Тип переходу: $n \to \pi^*$