Chapter 6: Answer Key

Hartree-Fock SCF from Integrals
2302638 Advanced Quantum Chemistry

Department of Chemistry, Chulalongkorn University

Contents

1 Checkpoint Question Answers

This section provides detailed answers to all 11 checkpoint questions from Chapter 6, organized by their location in the chapter.

1.1 Checkpoint 6.1: Big Picture (Week 6)

Section 6.1 – Learning Goals

Question: Where in the SCF loop does each integral type (**S**, **h**, ERIs) get used?
Answer:
Each integral type plays a specific role in the SCF iteration:

1. **Overlap matrix S:**
   - Used *once at the start* to build the orthogonalizer $\mathbf{X} = \mathbf{S}^{-1/2}$
   - Used in *every iteration* to solve the generalized eigenvalue problem $\mathbf{FC} = \mathbf{SC}\boldsymbol{\varepsilon}$ (via transformation $\mathbf{F}' = \mathbf{X}^\top \mathbf{F}\mathbf{X}$)
   - Used in *every iteration* to compute the SCF residual $\mathbf{R} = \mathbf{FPS} - \mathbf{SPF}$
   - Used to verify electron count: $N_e = \mathrm{tr}\{\mathbf{PS}\}$
2. **Core Hamiltonian $\mathbf{h} = \mathbf{T} + \mathbf{V}$:**
   - Used in *every iteration* to form the Fock matrix: $\mathbf{F} = \mathbf{h} + \mathbf{J} - \frac{1}{2}\mathbf{K}$
   - Used to compute the electronic energy: $E_{\text{elec}} = \frac{1}{2}\,\mathrm{tr}\{\mathbf{P}(\mathbf{h} + \mathbf{F})\}$
   - May be used for the *initial guess* (diagonalizing **h** in the **S**-metric)
3. **Two-electron integrals (ERIs):**
   - Used in *every iteration* to build **J** and **K** matrices:

$$J_{\mu\nu} = \sum_{\lambda\sigma} (\mu\nu|\lambda\sigma) P_{\lambda\sigma}, \quad K_{\mu\nu} = \sum_{\lambda\sigma} (\mu\lambda|\nu\sigma) P_{\lambda\sigma}$$

   - The most expensive part of each SCF iteration
   - In direct SCF, ERIs are recomputed each iteration rather than stored

The iteration structure is:

$$\underbrace{\mathbf{P}}_{\text{from prev.}} \xrightarrow{\text{ERIs}} \underbrace{\mathbf{J}, \mathbf{K}}_{\text{2e contrib.}} \xrightarrow{\mathbf{h}} \underbrace{\mathbf{F}}_{\text{Fock}} \xrightarrow{\mathbf{S}} \underbrace{(\boldsymbol{\varepsilon}, \mathbf{C})}_{\text{orbitals}} \to \underbrace{\mathbf{P}_{\text{new}}}_{\text{next iter.}}$$

1.2 Checkpoint 6.2: Chemist's vs Physicist's Notation

Section 6.2 – Born–Oppenheimer Hamiltonian

Question: Understand the difference between chemist's notation $(\mu\nu|\lambda\sigma)$ and physicist's notation $\langle\mu\lambda|\nu\sigma\rangle$.
Answer:
The key difference is which indices share the same electron coordinate:

Notation	Index Pattern	Same Electron
Chemist: $(\mu\nu\|\lambda\sigma)$	Adjacent pairs	(μ,ν) share $\mathbf{r}_1$; (λ,σ) share $\mathbf{r}_2$
Physicist: $\langle\mu\lambda\|\nu\sigma\rangle$	Alternating pairs	(μ,ν) share $\mathbf{r}_1$; (λ,σ) share $\mathbf{r}_2$

The conversion is:

$$\langle\mu\lambda|\nu\sigma\rangle_{\text{physicist}} = (\mu\nu|\lambda\sigma)_{\text{chemist}}$$

Why this matters: Mixing these conventions leads to incorrect index contractions when building **J** and **K**. For example, in chemist's notation:

$$J_{\mu\nu} = \sum_{\lambda\sigma} (\mu\nu|\lambda\sigma) P_{\lambda\sigma} \qquad \text{(density indices in ket)}$$

$$K_{\mu\nu} = \sum_{\lambda\sigma} (\mu\lambda|\nu\sigma) P_{\lambda\sigma} \qquad \text{(density indices "crossed")}$$

Converting to physicist's notation changes which indices appear where.

1.3 Checkpoint 6.3: Understanding Coulomb vs Exchange

Section 6.3 – HF Variational Principle

Questions:

1. Why is the Coulomb operator $\hat{J}_j$ called "local" and the exchange operator $\hat{K}_j$ called "nonlocal"?
2. If antisymmetry were not required, which operator would vanish?
3. In the RHF Fock operator, explain why there is a factor of 2 in front of $\hat{J}_j$ but not $\hat{K}_j$.

Answers:
1. Local vs Nonlocal Character:
Examine the operator definitions:

$$(\hat{J}_j\varphi)(\mathbf{r}_1) = \underbrace{\left(\int \frac{|\phi_j(\mathbf{r}_2)|^2}{r_{12}} \, d\mathbf{r}_2 \right)}_{V_j(\mathbf{r}_1) \text{ — a function of } \mathbf{r}_1 \text{ only}} \varphi(\mathbf{r}_1)$$

The Coulomb operator is **local** because:

- The integral produces a potential $V_j(\mathbf{r}_1)$ that depends only on position $\mathbf{r}_1$
- The result at point $\mathbf{r}_1$ depends only on the value of φ at that same point
- This is like multiplication by a function: $\hat{J}_j\varphi = V_j \cdot \varphi$

$$(\hat{K}_j\varphi)(\mathbf{r}_1) = \underbrace{\left(\int \frac{\phi_j(\mathbf{r}_2)\varphi(\mathbf{r}_2)}{r_{12}} \, d\mathbf{r}_2 \right)}_{\text{integral involving } \varphi(\mathbf{r}_2) \text{ at all } \mathbf{r}_2} \phi_j(\mathbf{r}_1)$$

The exchange operator is **nonlocal** because:

- The value at point $\mathbf{r}_1$ depends on $\varphi(\mathbf{r}_2)$ at *all* points $\mathbf{r}_2$
- Cannot be written as multiplication by a potential
- Mixes information from different spatial locations

2. Antisymmetry and Exchange:
The **exchange operator $\hat{K}_j$ would vanish** if antisymmetry were not required.

The exchange term $\langle ij|ji\rangle$ in the energy arises from the Slater determinant structure. For distinguishable particles (bosons), the wavefunction would be a permanent rather than a determinant, and the cross-terms that give rise to exchange would have the same sign as the Coulomb terms, doubling the classical repulsion. The subtraction of exchange is uniquely a consequence of fermionic antisymmetry.

3. Factor of 2 in RHF:

In RHF, each spatial orbital is doubly occupied (one α, one β electron).

- **Coulomb:** Both electrons in orbital j contribute to the electrostatic potential seen by other electrons. Since there are 2 electrons in each orbital, we get $2\hat{J}_j$ per occupied orbital.
- **Exchange:** Exchange only occurs between electrons of the *same spin*. An α electron in orbital i only exchanges with α electrons in other orbitals. Thus, only one electron per doubly-occupied orbital contributes exchange to any given electron—no factor of 2.

The net result is $\hat{F} = \hat{h} + \sum_j^{\text{occ}}(2\hat{J}_j - \hat{K}_j)$.

1.4 Checkpoint 6.4: Factor of 2 in RHF Density

Section 6.4 – RHF in AO Basis

Question: The factor of 2 in $P_{\mu\nu} = 2\sum_i C_{\mu i}C_{\nu i}$ accounts for double occupancy. Where does this factor reappear or get compensated in the Fock matrix and energy expressions?
Answer:
The factor of 2 in the density matrix propagates through the formalism in a coupled way:
1. In the Fock matrix:

$$\mathbf{F} = \mathbf{h} + \mathbf{J} - \frac{1}{2}\mathbf{K}$$

The factor of 2 from $\mathbf{P}$ appears in building $\mathbf{J}$ and $\mathbf{K}$:

$$J_{\mu\nu} = \sum_{\lambda\sigma}(\mu\nu|\lambda\sigma)P_{\lambda\sigma} \qquad \text{(factor of 2 from } \mathbf{P} \text{ is absorbed here)}$$

This is why the Fock matrix has $-\frac{1}{2}\mathbf{K}$ rather than $-\mathbf{K}$: the "missing" factor of 2 in the exchange term compensates for the double-counting in $\mathbf{P}$.
2. In the energy expression:

$$E_{\text{elec}} = \frac{1}{2}\,\text{tr}\{\mathbf{P}(\mathbf{h} + \mathbf{F})\}$$

The prefactor $\frac{1}{2}$ compensates for the factor of 2 in $\mathbf{P}$:

- $\text{tr}\{\mathbf{Ph}\}$ gives $2\sum_i\langle i|\hat{h}|i\rangle$ (the 2 counts both electrons)
- The $\frac{1}{2}$ in front of the two-electron part prevents double-counting pairs

The coupling is essential: If you remove the 2 from $\mathbf{P}$ (making $P_{\mu\nu} = \sum_i C_{\mu i}C_{\nu i}$), you must simultaneously:

- Change $\mathbf{F} = \mathbf{h} + 2\mathbf{J} - \mathbf{K}$ (move the 2 to the Coulomb term)
- Change the energy formula to match

Both conventions appear in the literature; consistency is what matters.

1.5 Checkpoint 6.5: Coulomb vs Exchange Index Patterns

Section 6.4 – J and K Derivation

Question: Compare the index contractions in $J_{\mu\nu}$ and $K_{\mu\nu}$. Why are the density indices "crossed" in the exchange term?
Answer:
Index patterns:

$$J_{\mu\nu} = \sum_{\lambda\sigma}(\mu\nu|\lambda\sigma)P_{\lambda\sigma} \qquad \text{density indices } (\lambda,\sigma) \text{ in the } \textbf{ket}$$

$$K_{\mu\nu} = \sum_{\lambda\sigma}(\mu\lambda|\nu\sigma)P_{\lambda\sigma} \qquad \text{density indices } (\lambda,\sigma) \text{ are } \textbf{crossed}$$

In the Coulomb case, both density indices (λ,σ) describe the same electron (electron 2 in the ERI). This represents the classical picture: the density $|\phi(\mathbf{r}_2)|^2 \sim P_{\lambda\sigma}\chi_\lambda(\mathbf{r}_2)\chi_\sigma(\mathbf{r}_2)$ generates a potential that acts on electron 1.
In the exchange case, the crossed pattern arises from the nonlocal structure of $\hat{K}$:

$$\langle\chi_\mu|\hat{K}_j|\chi_\nu\rangle = \iint \chi_\mu(\mathbf{r}_1)\ \underbrace{\phi_j(\mathbf{r}_2)\chi_\nu(\mathbf{r}_2)}_{\text{mixing } \chi_\nu \text{ with } \phi_j}\ \frac{1}{r_{12}}\ \underbrace{\phi_j(\mathbf{r}_1)}_{\text{returns } \phi_j \text{ at } \mathbf{r}_1}\ d\mathbf{r}_1 d\mathbf{r}_2$$

The test function χ_ν appears at position $\mathbf{r}_2$ alongside the occupied orbital, while the result involves $\phi_j(\mathbf{r}_1)$. This "swapping" of electron coordinates is the hallmark of exchange.
Mnemonic: In chemist's notation $(\mu\nu|\lambda\sigma)$:

- **Coulomb:** Contract density with the "other" electron (indices 3,4)
- **Exchange:** One density index goes to each electron (crossed: 2,4)

1.6 Checkpoint 6.6: Avoiding Double-Counting

Section 6.4 – Energy Expressions

Question: Explain why $E_{\text{elec}} = \frac{1}{2}\,\text{tr}\{\mathbf{P}(\mathbf{h}+\mathbf{F})\}$ avoids double-counting electron–electron interaction even though $\mathbf{F} = \mathbf{h} + \mathbf{G}$.
Answer:
Expand the expression:

$$\begin{aligned}\frac{1}{2}\,\text{tr}\{\mathbf{P}(\mathbf{h}+\mathbf{F})\} &= \frac{1}{2}\,\text{tr}\{\mathbf{P}(\mathbf{h}+\mathbf{h}+\mathbf{G})\}\\ &= \frac{1}{2}\,\text{tr}\{\mathbf{P}\cdot 2\mathbf{h}\} + \frac{1}{2}\,\text{tr}\{\mathbf{PG}\}\\ &= \text{tr}\{\mathbf{Ph}\} + \frac{1}{2}\,\text{tr}\{\mathbf{PG}\}\end{aligned}$$

This is exactly the standard energy expression where:

- $\text{tr}\{\mathbf{Ph}\}$ is the one-electron energy (kinetic + nuclear attraction)
- $\frac{1}{2}\,\text{tr}\{\mathbf{PG}\}$ is the two-electron energy with the $\frac{1}{2}$ preventing double-counting

Why the factor of $\frac{1}{2}$ is needed for two-electron terms:
The electron–electron repulsion involves *pairs* of electrons. When we sum over all electron pairs using $\mathbf{P}$ twice (once for each electron in the pair), we count each pair twice:

$$\sum_{i<j}\langle ij||ij\rangle \quad \text{(each pair once)} \quad \neq \quad \frac{1}{2}\sum_{i,j}\langle ij||ij\rangle \quad \text{(each pair twice, then halved)}$$

The trace formula $\text{tr}\{\mathbf{PG}\}$ corresponds to the unrestricted double sum, so the $\frac{1}{2}$ corrects for this.
The "trick" in the half-trace formula is that $\mathbf{h}$ appears once in $(\mathbf{h}+\mathbf{F})$ *and* once inside $\mathbf{F}$, so summing and dividing by 2 gives exactly one contribution from $\mathbf{h}$ and the correct half-contribution from $\mathbf{G}$.

1.7 Checkpoint 6.7: Linear Dependence from Diffuse Functions

Section 6.5 – Orthonormalization

Question: Why do diffuse basis functions tend to increase linear dependence problems? Relate your explanation to overlap between nearby Gaussians.
Answer:
Physical picture:
Diffuse basis functions have small Gaussian exponents α, meaning they extend far from their nuclear center. When two atoms are separated by a typical bond distance (1–2 Å),

their diffuse functions can have significant spatial extent (e.g., 3–5 Å half-width).
Overlap analysis:
The overlap between two Gaussians centered at **A** and **B** is (for s-type primitives):

$$S_{AB} \propto \exp\left(-\frac{\alpha\beta}{\alpha+\beta}|\mathbf{A}-\mathbf{B}|^2\right)$$

When both α and β are small (diffuse functions):

- The reduced exponent $\mu = \frac{\alpha\beta}{\alpha+\beta}$ is small
- The exponential decay with distance $|\mathbf{A}-\mathbf{B}|$ is slow
- Even at moderate separations, S_{AB} remains close to 1

Consequence for linear dependence:
When $S_{AB} \approx 1$, the two basis functions are nearly identical—they span almost the same subspace. The overlap matrix **S** develops small eigenvalues, and the basis becomes ill-conditioned:

- **S** has eigenvalues approaching zero
- The orthogonalizer $\mathbf{X} = \mathbf{S}^{-1/2}$ has eigenvalues approaching infinity
- Small errors in the overlap or Fock matrices get amplified

Practical implications:

- Augmented basis sets (aug-cc-pVXZ) with diffuse functions are more prone to linear dependence
- Large molecules with overlapping diffuse shells require eigenvalue thresholds (dropping eigenvectors with $s_i < 10^{-6}$)
- Anions and Rydberg states, which require diffuse functions, are particularly challenging

1.8 Checkpoint 6.8: Why the Initial Guess Matters

Section 6.6 – SCF as Fixed-Point Iteration

Questions:

1. Why might the core Hamiltonian guess cause convergence problems for many-electron systems or stretched bonds?
2. How does the HOMO–LUMO gap in the initial guess affect SCF stability?
3. What does the different convergence behavior for H_2O vs stretched H_2 suggest about easy vs hard SCF cases?

Answers:
1. Problems with the core Hamiltonian guess:
The core Hamiltonian $\mathbf{h} = \mathbf{T} + \mathbf{V}$ ignores all electron–electron repulsion. This creates problems because:

- **Many electrons:** The true Fock matrix includes substantial $\mathbf{J} - \frac{1}{2}\mathbf{K}$ contributions. Starting from $\mathbf{h}$ alone means the initial orbitals are far from the self-consistent ones.
- **Stretched bonds:** At large internuclear distances, electrons localize on separate atoms. The core guess tends to produce delocalized orbitals that poorly represent this near-dissociation regime.
- The initial density $\mathbf{P}^{(0)}$ may place electrons in the "wrong" orbitals, requiring large changes in subsequent iterations.

2. HOMO–LUMO gap and stability:
The HOMO–LUMO gap affects SCF stability because:

- **Large gap (core guess):** The occupied and virtual orbitals are well-separated. Small perturbations do not cause electrons to "swap" between occupied and virtual spaces. This is *stabilizing* for convergence.
- **Small gap (near convergence or stretched bonds):** The occupied-virtual energy difference is small. The SCF iteration can oscillate as electrons move between nearly-degenerate orbitals.
- The core guess typically has a *larger* gap than the converged solution because it ignores electron-electron repulsion that raises orbital energies.

3. Easy vs hard SCF cases:

Property	**Easy (H_2O eq.)**	**Hard (stretched H_2)**
HOMO–LUMO gap	Large (~ 0.5 Hartree)	Small ($\to 0$ at dissociation)
Electron localization	Delocalized on molecule	Localizing on separate atoms
Initial guess quality	Reasonable	Poor (wrong character)
Convergence	Fast (15–20 iterations)	Slow/oscillatory (30+ or fails)

General principle: SCF is "easy" when:

- The HOMO–LUMO gap is large
- The electronic structure is well-described by the initial guess
- No near-degeneracies or symmetry breaking

SCF is "hard" when any of these conditions fails—especially near bond breaking, for open-shell systems, or with small gaps.

1.9 Checkpoint 6.9: DIIS Subspace Size Tradeoffs

Section 6.7 – SCF Convergence Aids

Questions:

1. What are the tradeoffs in choosing the DIIS subspace size $m_{\max}$?
2. What happens to the condition number of $\mathbf{B}$ as you approach convergence?
3. Why might DIIS need a "restart"?

Answers:
1. Subspace size tradeoffs:

$m_{\max}$	**Advantages**	**Disadvantages**
Small (2–4)	• Low memory • Stable **B** matrix	• Limited extrapolation power • Cannot capture complex error structure
Medium (6–10)	• Good balance • Effective for most systems	• May need monitoring near convergence
Large (12–20)	• Maximum extrapolation • Can help difficult cases	• **B** becomes ill-conditioned • High memory usage • Old vectors become irrelevant

Typical choice: $m_{\max} = 6$–8 works well for most systems.
2. Condition number near convergence:
As SCF approaches convergence:

- Residual vectors $\mathbf{r}_i$ become small: $\|\mathbf{r}_i\| \to 0$
- Successive residuals become nearly parallel (all pointing toward zero)
- The error inner-product matrix $B_{ij} = \mathbf{r}_i^\top \mathbf{r}_j$ develops rows that are nearly proportional
- $\det(\mathbf{B}) \to 0$, and $\kappa(\mathbf{B}) \to \infty$

This is a *good* problem to have—it means SCF is converging! But it creates numerical difficulties for solving the DIIS linear system.
3. Why restart DIIS:
Restarting (clearing the DIIS history) is needed when:

- **B** becomes singular or nearly singular, causing numerical instability
- DIIS coefficients become unreasonable (e.g., $|c_i| > 10$)
- The extrapolated Fock matrix produces an energy *increase*
- Old vectors in the history are no longer representative of the current solution (e.g., after a major orbital reordering)

After restart, DIIS rebuilds its history with fresh iterations near the current (hopefully better) solution.

1.10 Checkpoint 6.10: DIIS Constraint Interpretation

Section 6.7 – SCF Convergence Aids

Question: Why must DIIS include the constraint $\sum_i c_i = 1$? Interpret this geometrically: the extrapolated Fock matrix lies in the *affine hull* of the previous Fock matrices. What would happen if $\sum_i c_i \neq 1$?

Answer:

Geometric interpretation:

Consider the space of Fock matrices as vectors. The previous Fock matrices $\{\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_m\}$ define a set of points in this space.

- **Linear span:** $\sum_i c_i \mathbf{F}_i$ with no constraint on c_i. This includes the origin (all $c_i = 0$) and scaled versions.
- **Affine hull:** $\sum_i c_i \mathbf{F}_i$ with $\sum_i c_i = 1$. This is the "flat" subspace passing through the points—like a line through two points or a plane through three non-collinear points.

The constraint $\sum_i c_i = 1$ ensures:

1. The extrapolated $\mathbf{F}_{\text{DIIS}}$ has the same "scale" as the input Fock matrices
2. If all input matrices are close to convergence, the output stays in that neighborhood
3. The extrapolation is an *interpolation/extrapolation* rather than an arbitrary linear combination

What happens without the constraint:

If $\sum_i c_i \neq 1$:

- $\sum_i c_i < 1$: The extrapolated Fock matrix is "scaled down." This would artificially reduce orbital energies and electron-electron repulsion.
- $\sum_i c_i > 1$: The extrapolated Fock matrix is "scaled up." Orbital energies and repulsion terms would be artificially increased.
- $\sum_i c_i = 0$: You get the zero matrix, which is clearly wrong!

The constraint ensures physical consistency: the extrapolated Fock matrix represents a plausible HF solution, not an arbitrary scaled version.

Mathematical formulation:

The DIIS problem with Lagrange multiplier:

$$\min_{\mathbf{c}} \left\| \sum_i c_i \mathbf{r}_i \right\|^2 \quad \text{subject to} \quad \sum_i c_i = 1$$

leads to the augmented linear system in Algorithm 6.2.

1.11 Checkpoint 6.11: Connecting the Chapters

Section 6.8 – Integral-Driven Viewpoint

Question: Trace the complete computational path from molecular geometry to SCF energy.
Answer:
The full computational pipeline, connecting all chapters in Part I:

1. **Geometry → Basis set (Chapter 2):**
   - Nuclear coordinates $\{\mathbf{R}_A\}$ and charges $\{Z_A\}$
   - Assign contracted Gaussian basis functions to each atom
   - Establish shell structure (s, p, d, ... on each center)
2. **Basis → One-electron integrals (Chapter 3):**
   - Use Gaussian Product Theorem for overlap: $S_{\mu\nu}$
   - Compute kinetic integrals: $T_{\mu\nu}$
   - Compute nuclear attraction using Boys function: $V_{\mu\nu}$
   - Form core Hamiltonian: $\mathbf{h} = \mathbf{T} + \mathbf{V}$
3. **Basis → Two-electron integrals (Chapters 4–5):**
   - For each shell quartet (A, B, C, D):
     (a) Apply Schwarz screening
     (b) If significant: compute Boys function $F_n(T)$
     (c) Use Rys quadrature for efficient ERI evaluation
     (d) Contract primitives into shell-block integrals
   - Store (in-core) or use immediately (direct)
4. **Integrals + SCF iteration (Chapter 6):**
   - Build orthogonalizer $\mathbf{X} = \mathbf{S}^{-1/2}$
   - Initial guess: diagonalize $\mathbf{h}$ or use SAD
   - SCF loop:
     (a) Build $\mathbf{J}, \mathbf{K}$ from ERIs and $\mathbf{P}$
     (b) Form $\mathbf{F} = \mathbf{h} + \mathbf{J} - \frac{1}{2}\mathbf{K}$
     (c) Apply DIIS acceleration
     (d) Solve $\mathbf{F}'\mathbf{C}' = \mathbf{C}'\boldsymbol{\varepsilon}$
     (e) Back-transform: $\mathbf{C} = \mathbf{X}\mathbf{C}'$
     (f) Update density: $\mathbf{P} = 2\mathbf{C}_{\text{occ}}\mathbf{C}_{\text{occ}}^{\top}$
     (g) Check convergence
   - Compute final energy: $E_{\text{HF}} = \frac{1}{2}\,\text{tr}\{\mathbf{P}(\mathbf{h} + \mathbf{F})\} + E_{\text{nuc}}$

Summary diagram:

$$\boxed{\text{Geometry}} \xrightarrow{\text{Ch 2}} \boxed{\text{Basis}} \xrightarrow{\text{Ch 3,4,5}} \boxed{S, h, \text{ERI}} \xrightarrow{\text{Ch 6}} \boxed{E_{\text{HF}}, \mathbf{C}, \mathbf{P}}$$

1.12 Checkpoint 6.12: Debugging Energy Differences

Section 6.10 – Lab 6A

Question: If your RHF energy differs from PySCF by more than $\sim 10^{-8}$ Hartree, what should you check?

Answer:

Debugging checklist in order of likelihood:

1. **Integral ordering/symmetry format mismatch:**
   - PySCF's `int2e` with `aosym="s1"` returns a 4D tensor with chemist's notation: `eri[p,q,r,s]` $= (pq|rs)$
   - Ensure your `einsum` contractions match:

```
J = np.einsum("pqrs,rs->pq", eri, P)  # Correct for J
K = np.einsum("prqs,rs->pq", eri, P)  # Correct for K
```

   - A common error: using `pqrs,qs->pr` for K (wrong index pattern)
2. **Normalization or basis convention differences:**
   - PySCF uses normalized contracted Gaussians by default
   - Spherical vs Cartesian: PySCF defaults to spherical harmonics for $\ell \geq 2$
   - Check `mol.cart = True/False` if using d or higher functions
3. **Convergence criteria differences:**
   - PySCF default: `conv_tol = 1e-9` (energy), `conv_tol_grad = 3e-4` (orbital gradient)
   - Your code may use different thresholds
   - A "converged" energy at loose tolerance may differ by 10^{-6}–10^{-7}
4. **Initial guess and DIIS settings:**
   - Different initial guesses converge to the same minimum but at different rates
   - If using DIIS, implementation details (subspace size, restart criteria) can affect final convergence
5. **Linear dependence threshold:**
   - If your code drops different eigenvectors than PySCF, the basis dimension differs
   - This usually causes larger ($> 10^{-4}$) energy differences

Quick diagnostic:

```
# Compare intermediate quantities
print("Tr[P*S] =", np.trace(P @ S))  # Should equal nelec
print("||S - S_pyscf|| =", np.linalg.norm(S -
    mol.intor("int1e_ovlp")))
print("||h - h_pyscf|| =", np.linalg.norm(h -
    (mol.intor("int1e_kin")
                                          +
                                             mol.intor("int1e_nuc"))))
```

1.13 Checkpoint 6.13: DIIS Behavior for Difficult Cases

Section 6.10 – Lab 6B

Question: For stretched H_2, does DIIS always help? If you observe oscillations or divergence, try adding a few cycles of damping before DIIS starts. Explain why this combination can be more robust than either approach alone.

Answer:

DIIS behavior for stretched H_2:

DIIS does *not* always help for difficult cases like stretched H_2. The problem arises because:

- At large bond distances, the HOMO–LUMO gap becomes very small
- The core Hamiltonian guess is far from the converged solution
- Early SCF iterations may oscillate wildly, producing residuals that point in inconsistent directions
- DIIS extrapolation from poor early iterations can *amplify* errors rather than reduce them

Why damping + DIIS works:

Method	Early iterations	Near convergence
Damping only	Stable but slow	Very slow (linear)
DIIS only	May diverge if far from solution	Fast (superlinear)
Damping then DIIS	Stable approach	Fast finish

The strategy:

1. **First 3–5 iterations:** Use damping ($\alpha = 0.3$–0.5) to stabilize the density toward a reasonable region
2. **Once residual is small enough:** Switch to DIIS for rapid final convergence
3. Alternatively: Always damp but start collecting DIIS vectors from iteration 3–4

Implementation hint:

```
for it in range(1, max_cycle+1):
    # ... build F, R ...

    if it <= 5:
        # Damping phase: don't use DIIS yet
        P = 0.5 * P_new + 0.5 * P  # 50% damping
    else:
        # DIIS phase
        if diis is not None:
            F = diis.update(F, R)
```

Physical interpretation:

Damping keeps the density from "jumping" too far in early iterations when the Fock matrix is far from self-consistent. This builds a stable foundation of reasonable iterations that DIIS can then extrapolate from effectively.

1.14 Checkpoint 6.14: Why Direct Can Be Faster

Section 6.10 – Lab 6C

Question: Why might `mf.get_jk` (direct J/K building) be faster than explicit in-core `einsum` even for small systems?
Answer:
Several factors contribute to direct J/K being faster:

1. **Cache locality and memory hierarchy:**
   - In-core: Must load the entire ERI tensor (N^4 elements) from memory during contraction
   - Direct: Computes and uses small ERI blocks that fit in CPU cache
   - Modern CPUs are heavily memory-bound; cache-friendly algorithms are much faster
2. **Schwarz screening:**
   - In-core `einsum`: Contracts *all* ERIs with density, including zeros
   - Direct: Skips shell quartets with $|(\mu\nu|\lambda\sigma)| < \tau$ (typically 30–50% of quartets for organic molecules)
   - Even for small systems, screening saves significant work
3. **Symmetry exploitation:**
   - In-core with `aosym="s1"`: Stores and contracts all N^4 elements
   - Direct: Uses 8-fold symmetry to compute each unique integral only once
   - Effective speedup factor of ~ 8 in integral computation
4. **Optimized integral batching:**
   - libcint processes shell quartets in optimized batches
   - SIMD vectorization over primitive contractions
   - Low-level C code vs Python-level `einsum`
5. **Simultaneous J and K:**
   - `get_jk` computes both matrices in one pass over shell quartets
   - In-core: Requires two separate `einsum` calls with different index patterns

Quantitative comparison (typical results):

System	In-core einsum	Direct get_jk
H_2O/STO-3G (7 AOs)	2 ms	0.5 ms
N_2/6-31G (18 AOs)	50 ms	5 ms
C_6H_6/cc-pVDZ (114 AOs)	30 s	0.5 s

The advantage of direct methods increases rapidly with system size because the in-core approach scales as $O(N^4)$ in both memory and time, while direct methods can exploit sparsity and screening.

2 Lab Answer Keys

This section provides detailed answers and expected outputs for Labs 6A–6C.

2.1 Lab 6A: Minimal RHF SCF from AO Integrals

Lab 6A – Complete Implementation and Validation

Objective: Implement Algorithm 6.1 (minimal RHF SCF) and validate against PySCF.
Expected Output for H_2O/STO-3G:

```
it=  1  E_tot=-73.285612594242  |R|=1.380e+00  |dP|=2.122e+00
it=  2  E_tot=-74.680096614127  dE=-1.394e+00  |R|=4.980e-01
    |dP|=9.051e-01
it=  3  E_tot=-74.918538453108  dE=-2.384e-01  |R|=1.391e-01
    |dP|=2.604e-01
it=  4  E_tot=-74.944057174894  dE=-2.552e-02  |R|=4.016e-02
    |dP|=6.905e-02
it=  5  E_tot=-74.945785987613  dE=-1.729e-03  |R|=1.181e-02
    |dP|=1.879e-02
...
it= 15  E_tot=-74.945893803889  dE=-1.118e-11  |R|=4.447e-07
    |dP|=7.234e-07
Educational RHF energy: -74.94589380388879
PySCF RHF energy      : -74.94589380388876
difference            : 2.842170943040401e-14
```

Key Implementation Details:

1. **Symmetric orthogonalizer:**

```python
def symm_orth(S, thresh=1e-10):
    e, U = np.linalg.eigh(S)
    keep = e > thresh
    X = U[:, keep] @ np.diag(e[keep]**-0.5) @ U[:, keep].T
    return X
```

The threshold drops eigenvectors with small eigenvalues to prevent numerical instability.

2. **J and K construction:**

```python
J = np.einsum("pqrs,rs->pq", eri, P, optimize=True)
K = np.einsum("prqs,rs->pq", eri, P, optimize=True)
```

Note the different index patterns: $(pq|rs) \cdot P_{rs}$ for J vs $(pr|qs) \cdot P_{rs}$ for K.

3. **Energy calculation:**

```python
E_elec = np.einsum("pq,pq->", P_new, h) +
    0.5*np.einsum("pq,pq->", P_new, G)
E_tot = E_elec + Enuc
```

Using the trace formula with $\mathbf{G} = \mathbf{J} - 0.5\mathbf{K}$.

Convergence Analysis:

Without DIIS, convergence is approximately linear. The energy decreases monotonically (for well-behaved cases), and the residual norm $\|\mathbf{R}\|_F$ provides a more sensitive convergence measure than the energy change alone.

Validation Criteria:

- Agreement with PySCF: $|E_{\text{our}} - E_{\text{PySCF}}| < 10^{-10}$ Hartree
- Electron count: $\text{tr}\{\mathbf{PS}\} = N_e$ (within 10^{-12})
- Residual: $\|\mathbf{FPS} - \mathbf{SPF}\|_F < 10^{-8}$

Common Issues and Solutions:

Problem	Solution
Energy differs by ~ 0.01	Check J/K einsum index patterns
Energy differs by ~ 0.1	Check factor of 2 in $\mathbf{P}$ or $\frac{1}{2}$ in energy
SCF does not converge	Lower `conv_tol`, add damping, or use DIIS
`LinAlgError` in `eigh`	Check for NaN in integrals or $\mathbf{P}$

2.2 Lab 6B: Pulay DIIS and Difficult SCF Cases

Lab 6B – DIIS Implementation and Convergence Study

Objective: Implement DIIS acceleration and compare convergence for stretched H_2.

DIIS Implementation Key Points:

1. **Error vector:** Use the SCF commutator residual

$$\mathbf{r}_k = \text{vec}(\mathbf{F}_k\mathbf{P}_k\mathbf{S} - \mathbf{S}\mathbf{P}_k\mathbf{F}_k)$$

2. **B matrix construction:**

```
B = np.empty((m+1, m+1))
B[-1, :] = -1.0
B[:, -1] = -1.0
B[-1, -1] = 0.0
for i in range(m):
    for j in range(m):
        B[i, j] = np.dot(self.R_list[i], self.R_list[j])
```

3. **Linear system:**

$$\begin{pmatrix} \mathbf{B} & -\mathbf{1} \\ -\mathbf{1}^\top & 0 \end{pmatrix} \begin{pmatrix} \mathbf{c} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ -1 \end{pmatrix}$$

Expected Output for Stretched H_2 ($R = 2.5$ Å):

Without DIIS:

```
--- SCF without DIIS ---
it=  1  E_tot=-0.939573477892  |R|=2.476e-01  |dP|=6.173e-01
it=  2  E_tot=-0.983422461291  dE=-4.385e-02  |R|=1.257e-01
   |dP|=3.220e-01
it=  3  E_tot=-0.995118274623  dE=-1.170e-02  |R|=6.426e-02
   |dP|=1.668e-01
```

```
...
it= 25  E_tot=-0.999999827841  dE=-1.082e-08  |R|=1.238e-05
    |dP|=3.247e-05
it= 26  E_tot=-0.999999879514  dE=-5.167e-09  |R|=5.911e-06
    |dP|=1.551e-05
Converged in 26 iterations
```

With DIIS:

```
--- SCF with DIIS ---
it=  1  E_tot=-0.939573477892  |R|=2.476e-01  |dP|=6.173e-01
it=  2  E_tot=-0.983422461291  dE=-4.385e-02  |R|=1.257e-01
    |dP|=3.220e-01
it=  3  E_tot=-0.998841209433  dE=-1.542e-02  |R|=3.901e-02
    |dP|=1.014e-01
...
it=  9  E_tot=-0.999999999978  dE=-2.183e-10  |R|=1.524e-06
    |dP|=4.001e-06
it= 10  E_tot=-1.000000000000  dE=-2.239e-11  |R|=4.872e-08
    |dP|=1.280e-07
Converged in 10 iterations
```

Convergence Comparison:

System	Without DIIS	With DIIS	Speedup
H_2O/STO-3G (eq.)	15 iterations	8 iterations	1.9×
H_2/STO-3G (2.5 Å)	26 iterations	10 iterations	2.6×
H_2/STO-3G (3.0 Å)	35+ or fails	12 iterations	3×+

Residual Norm Behavior:

- **Without DIIS:** Residual decreases roughly linearly on a log scale, with possible oscillations for difficult cases
- **With DIIS:** Residual initially decreases slowly while building the subspace, then drops rapidly (often by several orders of magnitude in 2–3 iterations) as DIIS extrapolation becomes effective

When DIIS Struggles:
For very stretched bonds (R > 3.5 Å) or near-degenerate systems:

- DIIS may produce negative coefficients or $|c_i| > 5$
- The B matrix may become ill-conditioned (singular matrix warnings)
- Energy may increase instead of decrease

Solution: Use damping for the first 3–5 iterations before enabling DIIS:

```
diis_start = 4   # Start DIIS after iteration 4
damping = 0.5    # Mix 50% old density in early iterations

for it in range(1, max_cycle+1):
    # ... build F, R ...

    if it < diis_start:
```

```
        P_new = (1 - damping) * P_new + damping * P
    else:
        F = diis.update(F, R)
    # ...
```

2.3 Lab 6C: In-Core vs Direct J/K Building

Lab 6C – Performance Comparison

Objective: Compare in-core ERI + einsum against PySCF's direct `get_jk`.
Expected Output for N_2/6-31G:

```
direct get_jk time (s): 0.0023
incore einsum time (s): 0.0412
||J_inc - J_dir||_F = 2.84e-14
||K_inc - K_dir||_F = 3.12e-14
```

Timing Analysis by System Size:

System	NAO	In-core (s)	Direct (s)	Ratio
H_2O/STO-3G	7	0.002	0.0005	4×
N_2/6-31G	18	0.041	0.002	20×
C_2H_4/6-31G*	38	2.1	0.04	52×
C_6H_6/6-31G*	102	180	0.8	225×

Key Observations:

1. **Agreement:** J and K matrices agree to machine precision ($\sim 10^{-14}$), confirming equivalent physics
2. **Scaling:** Direct method advantage grows with system size
   - In-core: $O(N^4)$ storage + $O(N^4)$ contraction
   - Direct: $O(N^2)$ storage + screening-dependent compute
3. **Memory:** For C_6H_6/6-31G* (102 AOs):
   - Full ERI tensor: $102^4 \times 8$ bytes ≈ 850 MB
   - Direct: Only current shell block in memory (~ 1 MB)

Why Direct Wins:

1. **Schwarz screening:** Skip negligible shell quartets

$$|(\mu\nu|\lambda\sigma)| \leq \sqrt{(\mu\nu|\mu\nu)}\sqrt{(\lambda\sigma|\lambda\sigma)}$$

   For organic molecules, 30–50% of quartets are screened out.

2. **8-fold symmetry:** Each unique ERI computed once:

$$\begin{aligned}(\mu\nu|\lambda\sigma) &= (\nu\mu|\lambda\sigma) = (\mu\nu|\sigma\lambda) = (\nu\mu|\sigma\lambda) \\ &= (\lambda\sigma|\mu\nu) = (\sigma\lambda|\mu\nu) = (\lambda\sigma|\nu\mu) = (\sigma\lambda|\nu\mu)\end{aligned}$$

3. **Cache efficiency:** Shell blocks fit in L1/L2 cache
4. **SIMD vectorization:** libcint uses AVX/SSE instructions for primitive loops

Implementation Notes:
To see the direct algorithm more clearly, you can examine PySCF's source:

```
# In pyscf/scf/_vhf.py
def direct(mol, dm, with_j=True, with_k=True):
    # Loop over shell quartets with screening
    for ish in range(mol.nbas):
        for jsh in range(ish+1):
            # Schwarz screening check
            if diag[ish]*diag[jsh] < thresh:
                continue
            for ksh in range(ish+1):
                for lsh in range(ksh+1):
                    # More screening
                    # Compute ERI block
                    # Accumulate into J and K
```

3 Exercise Answer Keys

Brief answers for the end-of-chapter exercises (Section 6.12).

3.1 Exercise 6.1: Derive the RHF Energy in AO Form [Core]

Factor Tracking

Starting from the spin-orbital energy:

$$E_{\text{HF}} = \sum_{i}^{\text{occ}} \langle i|\hat{h}|i\rangle + \frac{1}{2}\sum_{i,j}^{\text{occ}} (\langle ij|ij\rangle - \langle ij|ji\rangle) + E_{\text{nuc}}$$

For RHF with $N_e/2$ doubly-occupied spatial orbitals:

$$\sum_{i}^{\text{spin-occ}} \langle i|\hat{h}|i\rangle = 2 \sum_{i}^{\text{spatial-occ}} \langle \phi_i|\hat{h}|\phi_i\rangle = \sum_{\mu\nu} P_{\mu\nu} h_{\mu\nu} = \text{tr}\{\mathbf{Ph}\}$$

For two-electron terms, summing over all spin orbitals and converting to spatial orbitals gives factors of 4 for Coulomb and 2 for exchange. Combined with the $\frac{1}{2}$ prefactor:

$$E_{2e} = \frac{1}{2}(4J - 2K) = 2J - K = \frac{1}{2}\,\text{tr}\{\mathbf{P}(2\mathbf{J} - \mathbf{K})\} = \frac{1}{2}\,\text{tr}\{\mathbf{PG}\}$$

where we use $\mathbf{G} = \mathbf{J} - \frac{1}{2}\mathbf{K}$ with an extra factor redistributed.

3.2 Exercise 6.2: Stationarity and the Commutator Residual [Core]

Proof Sketch

At convergence, Roothaan-Hall gives:

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

For occupied orbitals: $\mathbf{FC}_{\text{occ}} = \mathbf{SC}_{\text{occ}}\boldsymbol{\varepsilon}_{\text{occ}}$
Then:

$$\begin{aligned}\mathbf{FPS} &= \mathbf{F} \cdot 2\mathbf{C}_{\text{occ}}\mathbf{C}_{\text{occ}}^{\top} \cdot \mathbf{S} \\ &= 2\mathbf{SC}_{\text{occ}}\boldsymbol{\varepsilon}_{\text{occ}}\mathbf{C}_{\text{occ}}^{\top}\mathbf{S}\end{aligned}$$

Similarly for $\mathbf{SPF}$. Since $\boldsymbol{\varepsilon}_{\text{occ}}$ is diagonal and $\mathbf{C}_{\text{occ}}^{\top}\mathbf{SC}_{\text{occ}} = \mathbf{I}$, both terms are equal, so $\mathbf{R} = \mathbf{FPS} - \mathbf{SPF} = \mathbf{0}$.

3.3 Exercise 6.3: Direct SCF Conceptual Design [Core]

Key points to include:

1. Four nested loops over shells: (A, B, C, D) where $A \geq B$, $C \geq D$, and $(AB) \geq (CD)$ in combined index
2. Pre-compute diagonal ERIs $(\mu\nu|\mu\nu)$ for all shell pairs for Schwarz bounds
3. For each quartet, check: if $\sqrt{(AB|AB)}\sqrt{(CD|CD)} < \tau$, skip

4. For surviving quartets: call integral engine (Rys quadrature for higher ℓ)
5. Immediately contract: $J_{AB} += \sum_{CD}(AB|CD)P_{CD}$ and similar for K
6. Discard ERI block after use—no storage

3.4 Exercise 6.4: DIIS Behavior Study [Core]

Typical results:

System	No DIIS	With DIIS	Residual Pattern
H_2O/STO-3G	15 iter	8 iter	Smooth decrease with DIIS
H_2 (3.0 Å)	40+ iter	12 iter	Oscillations without DIIS

The residual norm provides earlier warning of convergence issues than energy changes.

3.5 Exercise 6.5: Reproduce PySCF Energies [Core]

For H_2O/STO-3G with geometry:

```
O  0.0000  0.0000  0.0000
H  0.7586  0.0000  0.5043
H -0.7586  0.0000  0.5043
```

Expected: $E_{\text{HF}} = -74.9458938$ Hartree (agreement within 10^{-10}).
Typical adjustments: matching convergence thresholds, using `aosym="s1"`.

3.6 Exercise 6.6: MO-Basis Gradient Check [Advanced]

The occupied-virtual gradient:

$$g_{ai} = 2(\mathbf{C}_{\text{vir}})^\top \mathbf{F}\mathbf{C}_{\text{occ}} = 2F_{ai}^{\text{MO}}$$

At convergence, $F_{ai}^{\text{MO}} = 0$ because the Fock matrix is diagonal in the MO basis. Both $\|g\|$ and $\|\mathbf{R}\|$ should be proportional (up to basis transformation factors).

3.7 Exercise 6.7: Level Shifting Implementation [Advanced]

Implementation: After solving $\mathbf{F}'\mathbf{C}' = \mathbf{C}'\boldsymbol{\varepsilon}$, add shift to virtual orbital energies:

```
eps[nocc:] += sigma  # Add shift to virtual orbital energies
```

Effect: Increases HOMO-LUMO gap, stabilizing early iterations for difficult cases.

3.8 Exercise 6.8: SCF Metadynamics [Research]

Multiple SCF solutions arise when:

- Symmetry breaking occurs (e.g., σ_u vs σ_g at stretched H_2)
- Different orbital orderings are possible (e.g., near-degenerate d-orbitals)

Experiments: Different initial guesses (core Hamiltonian vs SAD vs perturbed density) may converge to different minima with different energies. The lowest-energy solution is the HF ground state; higher solutions correspond to excited configurations or saddle points.