Chapter 2: Answer Key

Gaussian Basis Sets and Orthonormalization

2302638 Advanced Quantum Chemistry

Department of Chemistry, Chulalongkorn University

Contents

1 Overview

This answer key covers:

- **11 Checkpoint Questions** (2.1–2.11) embedded throughout Chapter 2, testing conceptual understanding
- **3 Python Labs** (2A, 2B, 2C) with expected numerical results and validation criteria

Validation standard: All numerical results should match PySCF reference calculations to within the specified tolerances.

2 Checkpoint Question Answers

This section provides detailed answers to all 11 checkpoint questions from Chapter 2, organized by their location in the chapter.

2.1 Checkpoint 2.1: Why This Chapter Matters

Section 2.1 – Overlap Metric Motivation

Question: In an AO basis, $\mathbf{S} \neq \mathbf{I}$. This single fact explains:
- why the HF equations are a *generalized* eigenproblem $\mathbf{FC} = \mathbf{SC}\boldsymbol{\varepsilon}$,
- why SCF can fail to converge with diffuse basis sets,
- why the electron count is $\mathrm{tr}\{\mathbf{PS}\}$ rather than $\mathrm{tr}\{\mathbf{P}\}$.

Master the overlap metric now, and these mysteries disappear.

Answer: This is a motivational checkpoint rather than a direct question. However, students should understand:

Why $\mathbf{S} \neq \mathbf{I}$ creates a generalized eigenproblem: The MO orthonormality condition in function space is $\langle\phi_p|\phi_q\rangle = \delta_{pq}$. When MOs are expanded as $\phi_p = \sum_\mu C_{\mu p}\chi_\mu$, this becomes:

$$\langle\phi_p|\phi_q\rangle = \sum_{\mu\nu} C^*_{\mu p}\langle\chi_\mu|\chi_\nu\rangle C_{\nu q} = \mathbf{C}^\top\mathbf{SC} = \mathbf{I}$$

If $\mathbf{S} = \mathbf{I}$ (orthonormal basis), this reduces to $\mathbf{C}^\top\mathbf{C} = \mathbf{I}$, and the Fock eigenvalue problem becomes ordinary. Since $\mathbf{S} \neq \mathbf{I}$ for GTOs, we get the generalized form.

Why diffuse basis sets cause SCF convergence issues: Diffuse functions on neighboring atoms have large overlaps ($S_{\mu\nu} \approx 1$), making eigenvalues of $\mathbf{S}$ very small. The orthogonalizer $\mathbf{X} = \mathbf{U}\mathbf{s}^{-1/2}$ involves dividing by $\sqrt{s_i}$, which amplifies numerical noise when $s_i \to 0$. This creates an ill-conditioned Fock matrix transformation that can destabilize SCF.

Why electron count is $\mathrm{tr}\{\mathbf{PS}\}$: The electron density is $\rho(\mathbf{r}) = \sum_{\mu\nu} P_{\mu\nu}\chi_\mu(\mathbf{r})\chi_\nu(\mathbf{r})$. Integrating:

$$N_e = \int \rho(\mathbf{r})\, d\mathbf{r} = \sum_{\mu\nu} P_{\mu\nu} \int \chi_\mu\chi_\nu\, d\mathbf{r} = \sum_{\mu\nu} P_{\mu\nu}S_{\mu\nu} = \mathrm{tr}\{\mathbf{PS}\}$$

The overlap matrix converts the density matrix (which counts orbital populations) into actual electron counts by accounting for the spatial overlap of basis functions.

2.2 Checkpoint 2.2: Primitives vs Contractions

Section 2.2 – Gaussian Basis Functions

Question: What is the difference between a primitive and a contracted Gaussian? Why do basis sets use contractions instead of just primitives? (Hint: consider both computational cost and the need to approximate STO-like behavior.)
Answer: Primitive Gaussian: A single Gaussian function of the form:

$$g_{lmn}(\alpha, \mathbf{r} - \mathbf{A}) = N_{lmn}(\alpha)(x - A_x)^l (y - A_y)^m (z - A_z)^n e^{-\alpha|\mathbf{r}-\mathbf{A}|^2}$$

Each primitive has one exponent α and is centered at one point $\mathbf{A}$.
Contracted Gaussian: A fixed linear combination of primitives sharing the same center and angular momentum:

$$\chi_{lmn}(\mathbf{r}) = \sum_{p=1}^{n_c} d_p \, g_{lmn}(\alpha_p, \mathbf{r} - \mathbf{A})$$

The contraction coefficients $\{d_p\}$ and exponents $\{\alpha_p\}$ are predetermined and fixed during the calculation.
Why use contractions?

1. **Better approximation to STOs:** Single Gaussians have the wrong cusp (zero derivative at nucleus for s-type) and decay too fast ($e^{-\alpha r^2}$ vs $e^{-\zeta r}$). Combining tight (large α) and diffuse (small α) primitives can mimic the correct STO shape.

2. **Reduced computational cost:** Integral cost scales with the number of *basis functions* for building matrices ($\sim N^2$ to N^4), but primitive combinations can be precomputed. If we used K primitives as K separate basis functions, we would have K^4 ERIs to compute. Using contractions, the number of basis functions $N < K$ while still capturing the physics encoded in the primitives.

3. **Transferability:** Contraction schemes are optimized for atoms and transfer well to molecules. The contracted functions represent chemically meaningful orbitals (core, valence) rather than arbitrary mathematical expansions.

Numerical example: STO-3G uses 3 primitives per contracted function. For water with 7 AOs (STO-3G), we have 7 basis functions but $7 \times 3 = 21$ primitives. The integral engine works at the primitive level, but the Fock matrix is only 7×7.

Warning: Common Error

A common error is confusing the *number of basis functions* with the *number of primitives*. When counting basis set size (e.g., for scaling analysis), count contracted functions, not primitives. However, integral evaluation cost does depend on the number of primitives.

2.3 Checkpoint 2.3: Cartesian vs Spherical d-Functions

Section 2.3 – Angular Momentum Components

Question: Why do d-functions have 6 Cartesian components but only 5 spherical components? Where does the "extra" Cartesian degree of freedom go? (Hint: Consider what happens to $x^2 + y^2 + z^2$.)
Answer: Counting formulas:

- Cartesian: $N_{\text{cart}}(L) = \frac{(L+1)(L+2)}{2}$. For $L = 2$: $N_{\text{cart}}(2) = \frac{3\times4}{2} = 6$
- Spherical: $N_{\text{sph}}(L) = 2L + 1$. For $L = 2$: $N_{\text{sph}}(2) = 5$

The 6 Cartesian d-functions are:

$$x^2, \quad y^2, \quad z^2, \quad xy, \quad xz, \quad yz$$

The 5 spherical d-functions are:

$$d_{z^2} \propto (3z^2 - r^2), \quad d_{xz} \propto xz, \quad d_{yz} \propto yz, \quad d_{xy} \propto xy, \quad d_{x^2-y^2} \propto (x^2 - y^2)$$

Where does the extra component go?
The combination $x^2 + y^2 + z^2 = r^2$ is a *purely radial* function with no angular dependence. It transforms as an **s-function** ($L = 0$), not a d-function ($L = 2$). This is the "spurious" component.
Mathematically, the 6-dimensional space of Cartesian d-functions decomposes as:

$$6 \text{ Cartesian } d\text{-functions} = \underbrace{5 \text{ true } d\text{-functions}}_{L=2} \oplus \underbrace{1 \text{ } s\text{-function}}_{L=0}$$

General pattern: For angular momentum L, Cartesian Gaussians contain spurious components with $L - 2$, $L - 4$, etc. The number of spurious functions is:

$$N_{\text{cart}}(L) - N_{\text{sph}}(L) = \frac{L(L-1)}{2}$$

L	Spurious	Formula	Character
0 (s)	0	0	—
1 (p)	0	0	—
2 (d)	1	$\frac{2\times1}{2} = 1$	s-type (r^2)
3 (f)	3	$\frac{3\times2}{2} = 3$	p-type (xr^2, yr^2, zr^2)
4 (g)	6	$\frac{4\times3}{2} = 6$	d-type + s-type

Why spherical functions are preferred: Using Cartesian d-functions includes an extra s-type component that (1) increases basis set size unnecessarily and (2) can worsen linear dependence if an explicit s-function is also present.

2.4 Checkpoint 2.4: Basis Set Tradeoffs

Section 2.4 – Basis Set Selection

Question: Why not simply use the largest possible basis set with many polarization and diffuse functions? Beyond computational cost ($\mathcal{O}(N^4)$ for ERIs), what numerical issue limits basis set size?

Answer: The numerical issue: Near-linear dependence

Large basis sets with many diffuse functions create **near-linear dependence** in the overlap matrix $\mathbf{S}$. This manifests as:

1. Very small eigenvalues of $\mathbf{S}$ (approaching machine precision)
2. Large condition number $\kappa(\mathbf{S}) = s_{\max}/s_{\min}$
3. Numerical instability in orthogonalization and SCF

Physical origin: Diffuse functions on neighboring atoms have significant spatial overlap. When two basis functions χ_μ and χ_ν satisfy $S_{\mu\nu} \approx 1$, their linear combination $\chi_\mu - \chi_\nu$ has near-zero norm ($\|\chi_\mu - \chi_\nu\|^2 \approx 0$). This means the combination is nearly the zero function—the basis is "almost linearly dependent."

Numerical consequences:

- Orthogonalization requires computing $\mathbf{s}^{-1/2}$, which amplifies errors when eigenvalues are small
- The generalized eigenproblem becomes ill-conditioned
- SCF may fail to converge or converge to wrong solutions
- Small changes in geometry can cause large changes in MO coefficients

The remedy: Eigenvalue thresholding removes near-dependent directions (typically $\tau = 10^{-6}$ to 10^{-8}). This is a *numerical* fix, not a physical approximation—we are removing basis combinations that contribute nothing beyond numerical noise.

Quantitative example (from text):

Basis	N_{AO}	$\min(s_i)$
STO-3G	7	~ 0.35
cc-pVDZ	24	~ 0.05
aug-cc-pVDZ	41	~ 0.003

Adding diffuse functions (aug-) decreases $\min(s_i)$ by an order of magnitude.

2.5 Checkpoint 2.5: Near-Linear Dependence

Section 2.5 – Near-Linear Dependence

Question: If two basis functions become nearly linearly dependent, what happens to one eigenvalue of $\mathbf{S}$? What happens to $\kappa(\mathbf{S})$?

Answer: What happens to eigenvalues:

When two basis functions χ_μ and χ_ν become nearly identical (i.e., $S_{\mu\nu} \to 1$), one eigenvalue of $\mathbf{S}$ approaches zero.

Mathematical explanation: Consider the simple 2×2 case where both functions are normalized ($S_{\mu\mu} = S_{\nu\nu} = 1$):

$$\mathbf{S} = \begin{pmatrix} 1 & S_{\mu\nu} \\ S_{\mu\nu} & 1 \end{pmatrix}$$

The eigenvalues are:

$$s_\pm = 1 \pm S_{\mu\nu}$$

As $S_{\mu\nu} \to 1$: $s_+ \to 2$ and $s_- \to 0$.
The eigenvector for $s_- \to 0$ is proportional to $(1, -1)^\top$, representing the combination $\chi_\mu - \chi_\nu$ which has vanishing norm.
What happens to $\kappa(\mathbf{S})$:
The condition number $\kappa(\mathbf{S}) = s_{\max}/s_{\min}$ grows without bound as $s_{\min} \to 0$:

$$\kappa(\mathbf{S}) = \frac{s_+}{s_-} = \frac{1 + S_{\mu\nu}}{1 - S_{\mu\nu}} \xrightarrow{S_{\mu\nu} \to 1} \infty$$

Physical interpretation:

- A large condition number means the overlap matrix is "almost singular"
- The direction corresponding to $s_- \approx 0$ represents a linear combination of basis functions that contributes essentially nothing to the function space
- Trying to orthogonalize this direction amplifies numerical noise because we divide by $\sqrt{s_-}$

General case: In a large basis, near-linear dependence can involve multiple functions. The overlap matrix may have several small eigenvalues, each corresponding to a nearly redundant linear combination. The smallest eigenvalue determines the stability of the orthogonalization procedure.

Key Formula

For a 2×2 overlap matrix with normalized functions:

$$\mathbf{S} = \begin{pmatrix} 1 & S_{12} \\ S_{12} & 1 \end{pmatrix} \quad \Rightarrow \quad s_\pm = 1 \pm S_{12}, \quad \kappa = \frac{1 + S_{12}}{1 - S_{12}}$$

2.6 Checkpoint 2.6: Gram–Schmidt Stability

Section 2.6 – Gram–Schmidt Stability

Question: How does the numerical instability of Gram–Schmidt relate to the condition number $\kappa(\mathbf{S})$? When $\kappa(\mathbf{S})$ is large, what happens to the intermediate vectors in the orthogonalization process, and why does MGS help (but not fully solve) this problem?
Answer: Relationship between instability and $\kappa(\mathbf{S})$:
The Gram–Schmidt process iteratively subtracts projections:

$$\tilde{\phi}_k = \chi_k - \sum_{j=1}^{k-1} \phi_j \langle \phi_j | \chi_k \rangle$$

then normalizes: $\phi_k = \tilde{\phi}_k / \|\tilde{\phi}_k\|$.

When $\kappa(\mathbf{S})$ is large, some intermediate vectors $\tilde{\phi}_k$ become nearly zero in norm. The normalization step divides by this small norm, amplifying any rounding errors.

What happens to intermediate vectors:

Near-linear dependence means some χ_k is nearly expressible as a linear combination of $\{\chi_1, \ldots, \chi_{k-1}\}$. After subtracting projections, the residual $\tilde{\phi}_k$ should be small. However, rounding errors in computing the projections mean $\tilde{\phi}_k$ is not exactly what it should be—it contains a component parallel to the already-orthonormalized vectors.

When we normalize this "small but wrong" vector, the error is amplified by a factor $\sim 1/\|\tilde{\phi}_k\|$. If $\|\tilde{\phi}_k\| \sim 10^{-8}$, normalization amplifies errors by $\sim 10^8$.

Why MGS helps (but doesn't fully solve the problem):

Classical GS computes all projections using the *original* χ_k:

$$\tilde{\phi}_k = \chi_k - \sum_{j=1}^{k-1} \phi_j \langle \phi_j | \chi_k \rangle \quad \text{(CGS)}$$

Modified GS updates incrementally:

$$\mathbf{v}^{(0)} = \chi_k, \quad \mathbf{v}^{(j)} = \mathbf{v}^{(j-1)} - \phi_j \langle \phi_j | \mathbf{v}^{(j-1)} \rangle \quad \text{(MGS)}$$

In MGS, each projection is computed against a vector that is already closer to orthogonal with respect to $\phi_1, \ldots, \phi_{j-1}$. This reduces error accumulation because the intermediate vector $\mathbf{v}^{(j-1)}$ has had its components along earlier vectors removed.

Why MGS doesn't fully solve the problem:

Even with MGS, the fundamental issue remains: when $\|\tilde{\phi}_k\|$ is very small (comparable to machine precision), no amount of careful projection can rescue the situation. The signal is lost in the noise. MGS improves *relative* error but cannot overcome *absolute* precision limits.

The robust solution:

Eigendecomposition-based orthogonalization (Algorithm 2.2) handles this by explicitly identifying small eigenvalues and dropping them. This is mathematically equivalent to recognizing that the corresponding directions contribute nothing meaningful and removing them from the basis.

2.7 Checkpoint 2.7: Eigenvalue Thresholding

Section 2.7 – Eigenvalue Thresholding

Question: If you drop eigenvalues below τ, what changes physically? Are you changing the molecule, or are you changing the numerical representation of the same molecule?

Answer: You are changing the numerical representation, not the molecule.

Mathematical perspective:

The eigendecomposition $\mathbf{S} = \mathbf{U}\mathbf{s}\mathbf{U}^\top$ reveals the principal directions of the overlap structure. The eigenvector $\mathbf{u}_i$ with eigenvalue $s_i \approx 0$ corresponds to a linear combination of AOs:

$$\tilde{\chi}_i = \sum_\mu U_{\mu i} \chi_\mu$$

that has $\langle \tilde{\chi}_i | \tilde{\chi}_i \rangle = s_i \approx 0$. This combination is nearly the zero function—it contributes essentially nothing to the span of the basis.

Physical perspective:

Dropping this eigenvalue means excluding a direction that:

1. Has negligible overlap with any physical observable
2. Cannot meaningfully distinguish any electronic configurations
3. Contributes only numerical noise to the calculation

The *molecule* (nuclei positions, electron count) is unchanged. The *Hilbert space* we are approximating is unchanged. We are merely removing a redundant, numerically problematic parameterization of that space.
Analogy: Consider describing 3D vectors using 4 coordinates where the 4th is always 0.999× (sum of first 3). The 4th coordinate adds no information but creates numerical problems. Dropping it doesn't change what vectors we can represent.
Caveat—when thresholding can matter:
For certain properties (e.g., polarizabilities, anion binding), the diffuse tails of wavefunctions are physically important. Aggressive thresholding might remove basis directions needed to describe:

- Weakly bound electrons in anions
- Long-range response to electric fields
- Rydberg state character

In these cases, the "near-zero" eigenvalue direction may be chemically relevant, and thresholding is a physical approximation, not just numerical cleanup. This is rare but should be considered when very diffuse functions are essential to the physics.
Practical guideline: Use conservative thresholds ($\tau = 10^{-6}$ to 10^{-8}) to avoid removing physically relevant directions while still maintaining numerical stability.

2.8 Checkpoint 2.8: Linear Dependence vs BSSE

Section 2.7 – Linear Dependence vs BSSE

Question: Students sometimes confuse *near-linear dependence* (discussed here) with *Basis Set Superposition Error* (BSSE). These are distinct phenomena.
The paradox: Larger basis sets *reduce* BSSE (approaching the complete basis set limit) but *increase* linear dependence. Both involve diffuse functions, but for entirely different reasons.
Answer: Near-Linear Dependence (Numerical Problem):

- **Cause:** Diffuse functions on neighboring atoms have large overlap ($S_{\mu\nu} \approx 1$)
- **Symptom:** Small eigenvalues of $\mathbf{S}$, ill-conditioned generalized eigenproblem
- **Effect:** Numerical instability in SCF, potentially wrong convergence
- **Cure:** Eigenvalue thresholding (Algorithm 2.2)
- **Nature:** Numerical artifact, not a physical approximation

Basis Set Superposition Error (Physical Artifact):

- **Cause:** In supermolecular calculations (A + B), each fragment "borrows" basis functions from the other to improve its own description
- **Symptom:** Interaction energies are artificially too attractive (over-binding)

- **Effect:** Systematic error in binding energies, barriers, etc.
- **Cure:** Counterpoise correction (calculating each fragment in the full dimer basis)
- **Nature:** Physical artifact of finite basis incompleteness

Why the paradox occurs:
Large basis → Less BSSE: With a complete basis, each monomer is already perfectly described—there's nothing to "borrow." BSSE decreases monotonically toward zero as basis set quality improves.
Large basis → More linear dependence: Adding functions (especially diffuse ones) increases the chance that some linear combination is nearly redundant. The overlap matrix grows, and more eigenvalues can become small.
The shared culprit—diffuse functions: Both phenomena involve diffuse functions, but for opposite reasons:

- BSSE involves diffuse functions because they extend far enough to be "borrowed" by the partner fragment
- Linear dependence involves diffuse functions because they extend far enough to overlap strongly with functions on neighboring centers

Summary table:

	Linear Dependence	BSSE
Nature	Numerical	Physical
Cause	$S_{\mu\nu} \approx 1$	Incomplete monomer basis
Symptom	$s_{\min} \to 0$	E_{int} too negative
Cure	Thresholding	Counterpoise
Larger basis	Worse	Better

2.9 Checkpoint 2.9: Cholesky vs Eigendecomposition

Section 2.8 – Cholesky vs Eigendecomposition

Question: Why does Cholesky factorization fail when $\mathbf{S}$ has very small eigenvalues, whereas eigenvalue thresholding (Algorithm 2.2) handles this case gracefully? When would you prefer Cholesky over eigendecomposition-based methods?
Answer: Why Cholesky fails with small eigenvalues:
Cholesky factorization computes $\mathbf{S} = \mathbf{L}\mathbf{L}^\top$ where $\mathbf{L}$ is lower triangular. The algorithm proceeds column by column:

$$L_{jj} = \sqrt{S_{jj} - \sum_{k=1}^{j-1} L_{jk}^2}$$

When $\mathbf{S}$ has a very small eigenvalue, at some step the quantity under the square root becomes very small or negative (due to rounding). The algorithm either:

1. Produces a very small L_{jj}, leading to large entries in $\mathbf{L}^{-1}$ and numerical instability
2. Encounters a negative argument, causing failure

There is no natural way in standard Cholesky to "skip" a problematic direction—the algorithm processes the matrix in index order, not eigenvalue order.

Why eigenvalue thresholding works:
Eigendecomposition explicitly computes $\mathbf{S} = \mathbf{U}\mathbf{s}\mathbf{U}^\top$, revealing which directions are problematic (small s_i). We can:

1. Identify eigenvalues below threshold τ
2. Exclude those directions from $\mathbf{U}$
3. Form $\mathbf{X} = \mathbf{U}_k\mathbf{s}_k^{-1/2}$ using only "healthy" eigenvalues

The result is a rectangular $N \times M$ matrix that spans the numerically stable subspace. The problematic directions are simply absent.
When to prefer Cholesky:
Cholesky is faster: $\mathcal{O}(N^3/3)$ vs $\mathcal{O}(N^3)$ for eigendecomposition (with a larger prefactor for eigen). Use Cholesky when:

1. $\mathbf{S}$ is well-conditioned (minimal/small basis, no diffuse functions)
2. Speed is critical (large-scale calculations where orthogonalization is a bottleneck)
3. The code can fall back to eigendecomposition if Cholesky fails

Practical strategy in production codes:

1. Attempt Cholesky first (fast)
2. If Cholesky fails or produces poor condition number, fall back to eigendecomposition with thresholding
3. For known problematic cases (aug-cc-pVTZ on anions), go directly to eigendecomposition

Additional note—pivoted Cholesky:
A pivoted Cholesky factorization reorders columns to process large diagonals first, deferring small/problematic pivots to the end. This allows truncation similar to eigenvalue thresholding but at Cholesky's lower cost. Some production codes use this approach.

2.10 Checkpoint 2.10: Role of the Overlap Matrix

Section 2.9 – Role of the Overlap Matrix

Question: In Algorithm 2.3, why do we orthogonalize $\mathbf{S}$ and not $\mathbf{F}$? What role does $\mathbf{S}$ play in defining orthonormal MOs?
Answer: Why we orthogonalize S, not F:
The overlap matrix $\mathbf{S}$ defines the **metric** of our function space—it determines what "orthogonal" and "normalized" mean for functions expanded in the AO basis. This metric is:

- Fixed by the basis set choice
- Independent of the Hamiltonian or electronic state
- The same for any operator matrix ($\mathbf{F}$, $\mathbf{h}$, etc.)

The Fock matrix $\mathbf{F}$ is just one operator expressed in this basis. Different operators ($\mathbf{h}$, $\mathbf{F}$ at different SCF iterations, property integrals) all share the same metric $\mathbf{S}$.

Mathematical role of S:
The orthonormality condition for MOs in function space is:

$$\langle \phi_p | \phi_q \rangle = \delta_{pq}$$

Expanding MOs in the AO basis: $\phi_p = \sum_\mu C_{\mu p} \chi_\mu$, this becomes:

$$\sum_{\mu\nu} C^*_{\mu p} \langle \chi_\mu | \chi_\nu \rangle C_{\nu q} = \mathbf{C}^\top \mathbf{S} \mathbf{C} = \mathbf{I}$$

The key insight: $\mathbf{S}$ appears because basis functions are not orthonormal. We construct $\mathbf{X}$ satisfying $\mathbf{X}^\top \mathbf{S} \mathbf{X} = \mathbf{I}$ so that coefficients in the transformed basis automatically satisfy the orthonormality constraint.
What if we "orthogonalized" F instead?
If we tried $\mathbf{F} = \mathbf{U}_F \boldsymbol{\varepsilon} \mathbf{U}_F^\top$ (ordinary diagonalization), the eigenvectors $\mathbf{U}_F$ would satisfy $\mathbf{U}_F^\top \mathbf{U}_F = \mathbf{I}$, but **not** $\mathbf{U}_F^\top \mathbf{S} \mathbf{U}_F = \mathbf{I}$.
The resulting MOs would:

1. Not be orthonormal in the function space sense
2. Give wrong electron counts ($\mathrm{tr}\{\mathbf{PS}\} \neq N_e$)
3. Lead to incorrect energies and properties

The separation of concerns:

- $\mathbf{S}$ defines the geometry of the basis (metric, orthogonality)
- $\mathbf{F}$ defines the physics (orbital energies, which MOs are occupied)
- Algorithm 2.3 first handles the geometry ($\mathbf{X}$ from $\mathbf{S}$), then the physics ($\mathbf{F}' = \mathbf{X}^\top \mathbf{F} \mathbf{X}$)

This separation is efficient: compute $\mathbf{X}$ once from $\mathbf{S}$, reuse it for every $\mathbf{F}$ (at each SCF iteration, for different properties, etc.).

2.11 Checkpoint 2.11: Near-Linear Dependence in Practice

Section 2.10 – Near-Linear Dependence in Practice

Question: In practice, why might a code prefer dropping near-dependent directions rather than trying to keep all basis functions at all costs?
Answer: Reasons to drop near-dependent directions:

1. **Numerical stability is paramount:** A calculation that converges to a wrong answer (or doesn't converge at all) is useless, regardless of how "complete" the basis is. Dropping problematic directions ensures reliable results.
2. **Near-zero eigenvalue directions contribute nothing:** A direction with $s_i \sim 10^{-10}$ represents a linear combination of AOs that has essentially zero norm. This combination cannot distinguish any physical states—it's noise, not signal.
3. **Cost without benefit:** Keeping an extra basis direction adds $\mathcal{O}(N^3)$ to diagonalization cost and $\mathcal{O}(N^4)$ to ERI contractions. If that direction contributes only 10^{-10} to any observable, the cost far exceeds any possible benefit.

4. **Condition number affects all downstream calculations:** A large $\kappa(\mathbf{S})$ doesn't just affect orthogonalization—it propagates to the Fock matrix eigenvalue problem, DIIS convergence, response property calculations, etc. One problematic direction can destabilize an entire workflow.

5. **The "lost" information is usually irrelevant:** For typical chemistry (ground-state energies, structures, common properties), the diffuse regions described by near-dependent functions are beyond the precision that matters. Dropping them changes results by $\sim 10^{-8}$ Hartree or less.

When to be cautious:
There are cases where aggressive thresholding could affect results:

- Anion calculations (weakly bound electrons in diffuse orbitals)
- Polarizabilities and other response properties
- Excited states with Rydberg character
- Weak intermolecular interactions at long range

In these cases, use a more conservative threshold ($\tau = 10^{-10}$ instead of 10^{-6}) and verify that results are converged with respect to τ.
The practical philosophy:
Electronic structure theory is always an approximation (finite basis, finite correlation treatment, etc.). Adding one more source of approximation (dropping near-dependent directions) is acceptable if:

1. The error is smaller than other approximations in the calculation
2. The gain in stability/reliability is significant
3. The user can verify convergence by varying the threshold

3 Lab Solutions

3.1 Lab 2A: Eigenvalues of S and Conditioning vs Basis Set

Lab 2A Objectives

- Compute overlap eigenvalue spectra for different basis sets
- Observe how diffuse functions affect conditioning
- Calculate effective condition numbers

Expected numerical results:
For H_2O with geometry `O 0 0 0; H 0.7586 0 0.5043; H -0.7586 0 0.5043` (Angstrom):

Basis	N_{AO}	s_{min}	s_{max}	$\kappa(\mathbf{S})$
STO-3G	7	3.54×10^{-1}	2.00	5.7
cc-pVDZ	24	2.47×10^{-2}	2.71	1.1×10^{2}
aug-cc-pVDZ	41	2.67×10^{-3}	3.20	1.2×10^{3}
aug-cc-pVTZ	92	7.12×10^{-4}	4.04	5.7×10^{3}

Key observations:

1. **Basis size increases:** More functions provide more flexibility but also more potential for redundancy.
2. **Minimum eigenvalue decreases:** Adding diffuse functions (aug-) dramatically decreases $s_{\min}$. The aug-cc-pVDZ minimum is ∼100x smaller than cc-pVDZ.
3. **Condition number grows:** $\kappa(\mathbf{S})$ increases roughly by an order of magnitude with each step up in basis quality.
4. **Maximum eigenvalue is stable:** $s_{\max}$ grows slowly (from 2 to 4) because it reflects the most localized, tightly bound core functions which don't change much.

Complete solution code:

```python
import numpy as np
from pyscf import gto

def overlap_spectrum(atom, basis, unit="Angstrom"):
    """Compute overlap eigenvalues and condition number."""
    mol = gto.M(atom=atom, basis=basis, unit=unit, verbose=0)
    S = mol.intor("int1e_ovlp")
    eigs = np.linalg.eigvalsh(S)
    eigs_sorted = np.sort(eigs)[::-1]  # descending

    # Effective condition number (ignore numerically zero eigenvalues)
    tiny = 1e-14
    eigs_keep = eigs_sorted[eigs_sorted > tiny]
    cond = eigs_keep[0] / eigs_keep[-1]

    return mol.nao_nr(), eigs_sorted, cond

# Water geometry
atom = "O 0 0 0; H 0.7586 0 0.5043; H -0.7586 0 0.5043"

print("Basis Set Conditioning Analysis for H2O")
print("=" * 60)
print(f"{'Basis':<15} {'N_AO':>5} {'min(s)':>12} {'max(s)':>8}
    {'kappa':>12}")
print("-" * 60)

for basis in ["sto-3g", "cc-pVDZ", "aug-cc-pVDZ", "aug-cc-pVTZ"]:
    nao, eigs, cond = overlap_spectrum(atom, basis)
    print(f"{basis:<15} {nao:>5d} {eigs.min():>12.3e}
        {eigs.max():>8.2f} {cond:>12.1e}")

print("\nSmallest 5 eigenvalues for aug-cc-pVDZ:")
_, eigs, _ = overlap_spectrum(atom, "aug-cc-pVDZ")
for i, e in enumerate(np.sort(eigs)[:5]):
    print(f"  s_{i+1} = {e:.6e}")
```

Sample output:

```
Basis Set Conditioning Analysis for H2O
============================================================
Basis             N_AO       min(s)   max(s)        kappa
------------------------------------------------------------
```

```
sto-3g                7     3.542e-01      2.00       5.7e+00
cc-pVDZ              24     2.473e-02      2.71       1.1e+02
aug-cc-pVDZ          41     2.672e-03      3.20       1.2e+03
aug-cc-pVTZ          92     7.124e-04      4.04       5.7e+03

Smallest 5 eigenvalues for aug-cc-pVDZ:
  s_1 = 2.672349e-03
  s_2 = 7.441825e-03
  s_3 = 1.086731e-02
  s_4 = 1.524639e-02
  s_5 = 3.217584e-02
```

Warning: Common Error

Students often confuse $\kappa(\mathbf{S})$ with $\kappa(\mathbf{F})$. The overlap matrix condition number is *independent* of the electronic state—it depends only on the basis set and geometry. The Fock matrix condition number varies during SCF.

3.2 Lab 2B: Build X such that X^T S X = I

Lab 2B Objectives

- Implement the canonical orthogonalizer with eigenvalue thresholding
- Implement Cholesky and Gram–Schmidt orthogonalizers for comparison
- Verify orthogonality of the resulting transformation matrices

Expected numerical results:
For H_2O with aug-cc-pVDZ basis:

Method	Kept dims (M)	$\|\mathbf{X}^\top\mathbf{S}\mathbf{X} - \mathbf{I}\|_F$	Notes
Canonical ($\tau = 10^{-10}$)	41	$< 10^{-14}$	Stable
Canonical ($\tau = 10^{-6}$)	41	$< 10^{-14}$	Stable
Symmetric ($\tau = 10^{-10}$)	41	$< 10^{-14}$	Stable
Cholesky	41	$< 10^{-14}$	Stable
Gram–Schmidt	41	$\sim 10^{-10}$	Less accurate

Complete solution code:

```python
import numpy as np
from pyscf import gto

def canonical_orthogonalizer(S, thresh=1e-10):
    """Canonical orthogonalizer: X = U @ diag(s^{-1/2})"""
    eigs, U = np.linalg.eigh(S)
    keep = eigs > thresh
    n_dropped = np.sum(~keep)
    Uk = U[:, keep]
    ek = eigs[keep]
    X = Uk @ np.diag(ek ** -0.5)
    return X, ek, n_dropped

def symmetric_orthogonalizer(S, thresh=1e-10):
```

```
    """Symmetric (Lowdin) orthogonalizer: X = S^{-1/2} = U @
       diag(s^{-1/2}) @ U^T"""
    eigs, U = np.linalg.eigh(S)
    keep = eigs > thresh
    n_dropped = np.sum(~keep)
    Uk = U[:, keep]
    ek = eigs[keep]
    X = Uk @ np.diag(ek ** -0.5) @ Uk.T
    return X, ek, n_dropped

def cholesky_orthogonalizer(S):
    """Cholesky orthogonalizer: X = L^{-T}"""
    L = np.linalg.cholesky(S)
    X = np.linalg.inv(L.T)
    return X

def gram_schmidt_orthogonalizer(S, thresh=1e-12):
    """Modified Gram-Schmidt orthogonalizer under S-metric."""
    N = S.shape[0]
    X = np.zeros((N, N))
    m = 0
    for k in range(N):
        v = np.zeros(N)
        v[k] = 1.0
        # Modified Gram-Schmidt under S
        for j in range(m):
            r = X[:, j].T @ S @ v
            v = v - r * X[:, j]
        n2 = v.T @ S @ v
        if n2 < thresh:
            continue  # Skip near-dependent direction
        X[:, m] = v / np.sqrt(n2)
        m += 1
    return X[:, :m], m

def check_orthogonality(S, X):
    """Compute ||X^T S X - I||_F"""
    I_approx = X.T @ S @ X
    return np.linalg.norm(I_approx - np.eye(I_approx.shape[0]))

# Setup molecule
mol = gto.M(
    atom="O 0 0 0; H 0.7586 0 0.5043; H -0.7586 0 0.5043",
    basis="aug-cc-pVDZ",
    unit="Angstrom",
    verbose=0
)
S = mol.intor("int1e_ovlp")
N = S.shape[0]

print(f"Testing orthogonalizers for H2O/aug-cc-pVDZ (N={N})")
print("=" * 70)

# Canonical with different thresholds
for tau in [1e-6, 1e-8, 1e-10, 1e-12]:
    X, eigs, n_drop = canonical_orthogonalizer(S, thresh=tau)
    err = check_orthogonality(S, X)
    print(f"Canonical (tau={tau:.0e}): M={X.shape[1]:3d},
```

```
        dropped={n_drop}, err={err:.2e}")

# Symmetric
X_sym, eigs, n_drop = symmetric_orthogonalizer(S, thresh=1e-10)
err_sym = check_orthogonality(S, X_sym)
print(f"Symmetric (tau=1e-10):   M={X_sym.shape[1]:3d},
    dropped={n_drop}, err={err_sym:.2e}")

# Cholesky
try:
    X_chol = cholesky_orthogonalizer(S)
    err_chol = check_orthogonality(S, X_chol)
    print(f"Cholesky:                M={X_chol.shape[1]:3d},
        err={err_chol:.2e}")
except np.linalg.LinAlgError:
    print("Cholesky: FAILED (matrix not positive definite)")

# Gram-Schmidt
X_gs, m_gs = gram_schmidt_orthogonalizer(S, thresh=1e-12)
err_gs = check_orthogonality(S, X_gs)
print(f"Gram-Schmidt:            M={m_gs:3d}, err={err_gs:.2e}")

# Compare canonical vs symmetric matrices
X_can, _, _ = canonical_orthogonalizer(S, thresh=1e-10)
diff = np.linalg.norm(X_can - X_sym[:, :X_can.shape[1]])
print(f"\n||X_can - X_sym||_F = {diff:.2e} (should be nonzero)")
print("(Different orthogonalizers, same orthogonality property)")
```

Sample output:

```
Testing orthogonalizers for H2O/aug-cc-pVDZ (N=41)
======================================================================
Canonical (tau=1e-06): M= 41, dropped=0, err=1.71e-15
Canonical (tau=1e-08): M= 41, dropped=0, err=1.71e-15
Canonical (tau=1e-10): M= 41, dropped=0, err=1.71e-15
Canonical (tau=1e-12): M= 41, dropped=0, err=1.71e-15
Symmetric (tau=1e-10):   M= 41, dropped=0, err=4.05e-15
Cholesky:                M= 41, err=2.84e-15
Gram-Schmidt:            M= 41, err=8.21e-11

||X_can - X_sym||_F = 6.14e+01 (should be nonzero)
(Different orthogonalizers, same orthogonality property)
```

Key observations:

1. All methods achieve $\mathbf{X}^\top \mathbf{S}\mathbf{X} \approx \mathbf{I}$ to high precision for well-conditioned cases.
2. Gram–Schmidt shows slightly larger error (10^{-10} vs 10^{-15}) due to accumulated rounding.
3. Canonical and symmetric orthogonalizers are different matrices ($\|\mathbf{X}_{\text{can}} - \mathbf{X}_{\text{sym}}\| \gg 0$) but both valid.
4. For aug-cc-pVDZ on water, no eigenvalues are dropped even at $\tau = 10^{-6}$ because the smallest eigenvalue (~ 0.003) is well above threshold.

3.3 Lab 2C: Solve FC = SCe via Orthogonalization

Lab 2C Objectives

- Build the core Hamiltonian from one-electron integrals
- Transform to an orthonormal basis and solve the ordinary eigenproblem
- Verify that the resulting MO coefficients satisfy $\mathbf{C}^\top\mathbf{SC} = \mathbf{I}$

Expected numerical results:
For H_2 at $R = 0.74$ Å with cc-pVDZ basis:

Quantity	Expected Value
Number of AOs	10
Number of kept MOs (after thresholding)	10
$\|\mathbf{C}^\top\mathbf{SC} - \mathbf{I}\|_F$	$< 10^{-14}$
Lowest 5 eigenvalues	See below

Lowest 5 core Hamiltonian eigenvalues (H_2/cc-pVDZ):

```
eps[0] = -1.1287  (sigma bonding)
eps[1] = -0.4927  (sigma antibonding)
eps[2] =  0.1489
eps[3] =  0.2117
eps[4] =  0.5404
```

Complete solution code:

```python
import numpy as np
from pyscf import gto
from scipy.linalg import eigh as scipy_eigh

def canonical_orthogonalizer(S, thresh=1e-10):
    """Canonical orthogonalizer with eigenvalue thresholding."""
    eigs, U = np.linalg.eigh(S)
    keep = eigs > thresh
    Uk = U[:, keep]
    ek = eigs[keep]
    X = Uk @ np.diag(ek ** -0.5)
    return X, ek

def solve_gen_eig_orthog(F, S, thresh=1e-10):
    """
    Solve FC = SC*eps via orthogonalization (Algorithm 2.3).

    Returns:
        eps: eigenvalues (orbital energies)
        C: eigenvectors (MO coefficients in AO basis)
        X: orthogonalizer used
    """
    # Step 1: Build orthogonalizer
    X, _ = canonical_orthogonalizer(S, thresh=thresh)

    # Step 2: Transform F to orthonormal basis
    F_prime = X.T @ F @ X

```

```python
    # Step 3: Solve ordinary eigenproblem
    eps, C_prime = np.linalg.eigh(F_prime)

    # Step 4: Back-transform to AO basis
    C = X @ C_prime

    return eps, C, X

# Setup molecule
mol = gto.M(
    atom="H 0 0 0; H 0 0 0.74",
    basis="cc-pVDZ",
    unit="Angstrom",
    verbose=0
)

# Build integrals
S = mol.intor("int1e_ovlp")
T = mol.intor("int1e_kin")
V = mol.intor("int1e_nuc")
H = T + V  # Core Hamiltonian

print(f"H2/cc-pVDZ: N_AO = {mol.nao_nr()}")
print("=" * 50)

# Solve using our implementation
eps, C, X = solve_gen_eig_orthog(H, S, thresh=1e-10)

# Verify orthonormality
I_check = C.T @ S @ C
orth_err = np.linalg.norm(I_check - np.eye(C.shape[1]))
print(f"||C^T S C - I||_F = {orth_err:.2e}")

# Verify eigenvalue equation: H C[:, i] = S C[:, i] * eps[i]
residual = H @ C - S @ C @ np.diag(eps)
res_norm = np.linalg.norm(residual)
print(f"||H C - S C eps||_F = {res_norm:.2e}")

print(f"\nLowest 5 eigenvalues:")
for i in range(min(5, len(eps))):
    print(f"  eps[{i}] = {eps[i]:+.6f}")

# Compare with scipy's generalized eigenvalue solver
print("\n--- Validation against scipy.linalg.eigh ---")
eps_scipy, C_scipy = scipy_eigh(H, S)
eps_diff = np.max(np.abs(eps - eps_scipy))
print(f"Max |eps_ours - eps_scipy| = {eps_diff:.2e}")

# Also verify scipy's C satisfies orthonormality
orth_err_scipy = np.linalg.norm(C_scipy.T @ S @ C_scipy -
   np.eye(C_scipy.shape[1]))
print(f"||C_scipy^T S C_scipy - I||_F = {orth_err_scipy:.2e}")
```

Sample output:

```
H2/cc-pVDZ: N_AO = 10
==================================================
||C^T S C - I||_F = 1.33e-15
```

```
||H C - S C eps||_F = 1.22e-14

Lowest 5 eigenvalues:
  eps[0] = -1.128704
  eps[1] = -0.492678
  eps[2] = +0.148875
  eps[3] = +0.211698
  eps[4] = +0.540418

--- Validation against scipy.linalg.eigh ---
Max |eps_ours - eps_scipy| = 2.22e-15
||C_scipy^T S C_scipy - I||_F = 8.88e-16
```

Validation criteria:

Check	Tolerance	Purpose
$\|\mathbf{C}^\top\mathbf{SC} - \mathbf{I}\|_F$	$< 10^{-12}$	MO orthonormality
$\|\mathbf{hC} - \mathbf{SC}\boldsymbol{\varepsilon}\|_F$	$< 10^{-12}$	Eigenvalue equation
$\|\varepsilon_{\text{ours}} - \varepsilon_{\text{scipy}}\|$	$< 10^{-12}$	Cross-validation

Warning: Common Student Errors

1. **Forgetting to back-transform:** Using $\mathbf{C}'$ instead of $\mathbf{C} = \mathbf{XC}'$ gives MOs that are not orthonormal under $\mathbf{S}$.
2. **Wrong matrix order:** $\mathbf{F}' = \mathbf{XFX}^\top$ is incorrect; must be $\mathbf{F}' = \mathbf{X}^\top\mathbf{FX}$.
3. **Checking wrong orthonormality:** Testing $\mathbf{C}^\top\mathbf{C} = \mathbf{I}$ instead of $\mathbf{C}^\top\mathbf{SC} = \mathbf{I}$.

4 Additional Notes for Instructors

4.1 Common Misconceptions

1. **"More basis functions is always better":** Near-linear dependence can make calculations worse, not just slower.
2. **"Eigenvalue thresholding loses accuracy":** It removes numerically meaningless directions; the physical content is preserved.
3. **"Gram–Schmidt is production-ready":** It builds intuition but should not be used in real codes due to stability issues.
4. **"BSSE and linear dependence are the same":** They are distinct phenomena with opposite trends vs basis size.

4.2 Suggested Discussion Questions

1. Why does PySCF default to spherical Gaussians rather than Cartesian?
2. If thresholding removes basis directions, how do we know we haven't removed something important?
3. What happens to the MO energies if we use an ill-conditioned basis without thresholding?
4. Why is the overlap matrix computed once and reused, while the Fock matrix is recomputed at each SCF iteration?

4.3 Extensions for Advanced Students

1. Implement pivoted Cholesky and compare with eigenvalue thresholding for ill-conditioned cases.
2. Study how the eigenvalue spectrum of $\mathbf{S}$ changes with molecular geometry (bond stretching).
3. Investigate the relationship between basis set condition number and SCF convergence rate.
4. Compare computational cost of Cholesky vs eigendecomposition for various system sizes.

5 References

1. Boys, S.F. (1950). "Electronic wave functions. I. A general method of calculation for the stationary states of any molecular system." *Proc. R. Soc. A* **200**, 542–554.
2. Löwdin, P.O. (1950). "On the Non-Orthogonality Problem Connected with the Use of Atomic Wave Functions in the Theory of Molecules and Crystals." *J. Chem. Phys.* **18**, 365–375.
3. Roothaan, C.C.J. (1951). "New Developments in Molecular Orbital Theory." *Rev. Mod. Phys.* **23**, 69–89.
4. Szabo, A. & Ostlund, N.S. (1989). *Modern Quantum Chemistry*. Dover Publications. Chapter 3.

6 Exercise Answer Keys

Brief answers for the end-of-chapter exercises (Section 2.12).

6.1 Exercise 2.1: Basis-Set Dependence of Conditioning [Core]

Trend Analysis

For H_2O with fixed geometry, the overlap eigenvalue spectrum shows:

Basis	N_{AO}	$\min(s_i)$	$\kappa(\mathbf{S})$
STO-3G	7	~ 0.35	~ 6
cc-pVDZ	24	~ 0.025	~ 100
aug-cc-pVDZ	41	~ 0.003	~ 1000

Key trend: Adding diffuse functions (aug-) decreases $\min(s_i)$ dramatically because diffuse functions on neighboring atoms have large mutual overlaps ($S_{\mu\nu} \approx 1$), creating near-linear dependence. The condition number grows roughly by an order of magnitude with each step up in basis quality.

6.2 Exercise 2.2: Gram–Schmidt Ordering Dependence [Core]

Key Observation

With permuted basis ordering ($\mathbf{S}' = \mathbf{P}^\top \mathbf{S}\mathbf{P}$), the Gram–Schmidt orthogonalizers $\mathbf{X}$ and $\mathbf{X}'$ differ by more than just the permutation:

$$\|\mathbf{X} - \mathbf{P}\mathbf{X}'\| \neq 0$$

Why: Gram–Schmidt produces an orthonormal basis that depends on the processing order. The first vector is taken as-is (normalized), the second is orthogonalized against the first, etc. Different orderings give different triangular structures in the resulting transformation.
Lesson: Gram–Schmidt is not unique; the result depends on input ordering. Symmetric orthogonalization ($\mathbf{S}^{-1/2}$) produces a unique result independent of basis ordering.

6.3 Exercise 2.3: Symmetric vs Canonical Orthogonalization [Core]

Two Valid Orthogonalizers

Both satisfy $\mathbf{X}^\top \mathbf{S}\mathbf{X} = \mathbf{I}$, but they are different matrices:
Canonical: $\mathbf{X}_{\text{can}} = \mathbf{U}\mathbf{s}^{-1/2}$

- Rectangular ($N \times M$ if thresholding applied)
- Columns are not related to original AOs in an obvious way
- Simpler to implement with thresholding

Symmetric: $\mathbf{X}_{\text{sym}} = \mathbf{U}\mathbf{s}^{-1/2}\mathbf{U}^\top = \mathbf{S}^{-1/2}$

- Square ($N \times N$) if no thresholding
- Orthonormalized AOs are "closest" to original AOs (Löwdin)
- Preserves symmetry properties of original basis

Numerically: $\|\mathbf{X}_{\text{can}} - \mathbf{X}_{\text{sym}}\| \gg 0$, but both are valid.

6.4 Exercise 2.4: Generalized Eigenproblem Consistency [Core]

Verification Checklist

For $\mathbf{F}\mathbf{C} = \mathbf{S}\mathbf{C}\boldsymbol{\varepsilon}$ solved via orthogonalization:

1. **MO orthonormality:** $\|\mathbf{C}^\top \mathbf{S}\mathbf{C} - \mathbf{I}\|_F < 10^{-12}$
2. **Eigenvalue equation:** $\|\mathbf{F}\mathbf{C} - \mathbf{S}\mathbf{C}\boldsymbol{\varepsilon}\|_F < 10^{-12}$
3. **Cross-validation:** Eigenvalues match `scipy.linalg.eigh(F, S)` to $< 10^{-12}$

Common error: Using $\mathbf{C}' = \mathbf{X}^\top \mathbf{C}_{\text{ordinary}}$ instead of $\mathbf{C} = \mathbf{X}\mathbf{C}'$ for back-transformation. The correct sequence is:

$$\mathbf{F}' = \mathbf{X}^\top \mathbf{F}\mathbf{X}, \quad \mathbf{F}'\mathbf{C}' = \mathbf{C}'\boldsymbol{\varepsilon}, \quad \mathbf{C} = \mathbf{X}\mathbf{C}'$$

6.5 Exercise 2.5: Why Does Thresholding Not "Change Chemistry"? [Advanced]

Conceptual Argument

A small eigenvalue $s_i \ll 1$ means the corresponding eigenvector $\tilde{\chi}_i = \sum_\mu U_{\mu i}\chi_\mu$ has $\langle\tilde{\chi}_i|\tilde{\chi}_i\rangle = s_i \approx 0$. This combination is *nearly the zero function*—it cannot distinguish any physical states and contributes only numerical noise.

Thresholding is numerical cleanup, not physical approximation:

- The molecule (nuclei, electrons) is unchanged
- The Hilbert space being approximated is unchanged
- We remove a redundant parameterization that creates numerical problems

Exception—when thresholding can matter: For properties requiring diffuse tails (polarizabilities, anion binding, Rydberg states), aggressive thresholding ($\tau > 10^{-6}$) might remove physically relevant directions. Use conservative thresholds ($\tau = 10^{-8}$ to 10^{-10}) for such calculations.

6.6 Exercise 2.6: Cartesian vs Spherical Gaussian Counting [Core]

Counting Formulas

(a) Oxygen with $[3s2p1d]$:

- Cartesian: $3 \times 1 + 2 \times 3 + 1 \times 6 = 3 + 6 + 6 = 15$
- Spherical: $3 \times 1 + 2 \times 3 + 1 \times 5 = 3 + 6 + 5 = 14$

(b) 10 heavy atoms $[4s3p2d1f]$ + 20 H atoms $[2s1p]$:

- Heavy (Cart): $10 \times (4 + 9 + 12 + 10) = 10 \times 35 = 350$
- Heavy (Sph): $10 \times (4 + 9 + 10 + 7) = 10 \times 30 = 300$
- H (both): $20 \times (2 + 3) = 100$
- Total Cartesian: $350 + 100 = 450$
- Total Spherical: $300 + 100 = 400$

(d) Why difference grows with L: Spurious components $= \frac{L(L-1)}{2}$: 0 for s/p, 1 for d, 3 for f, 6 for g. The r^2 contaminant in Cartesian d-functions is s-type; in f-functions, the xr^2, yr^2, zr^2 are p-type.

6.7 Exercise 2.7: Thresholding Sensitivity Study [Advanced]

Results for F^-/aug-cc-pVTZ

Typical findings:

τ	M (kept)	$\varepsilon_{\min}$ (Hartree)	$\kappa(\mathbf{X}^\top\mathbf{S}\mathbf{X})$
10^{-6}	~ 42	stable	~ 1
10^{-8}	~ 45	stable	~ 1
10^{-10}	~ 46	stable	~ 1
10^{-12}	46	may show noise	> 1

Practical choice: $\tau = 10^{-7}$ to 10^{-8} balances stability and completeness. For anions, err on the side of keeping more functions ($\tau = 10^{-8}$).
Instability signature: When τ is too small, $\kappa(\mathbf{X}^\top\mathbf{S}\mathbf{X}) > 10$ and eigenvalues show iteration-to-iteration fluctuations.

6.8 Exercise 2.8: GTO Normalization Verification [Core]

Normalization Constants

(a) s-type ($l = m = n = 0$) with $\alpha = 1.0$:

$$N_{000}(\alpha) = \left(\frac{2\alpha}{\pi}\right)^{3/4} = \left(\frac{2}{\pi}\right)^{3/4} \approx 0.7127$$

(b) Numerical verification: Using `scipy.integrate.tplquad` over $[-L, L]^3$ with $L \approx 5$ (Gaussian decays rapidly), the integral $\int |g_{000}|^2 d\mathbf{r} \approx 1.0$ to 6+ digits.
(c) p-type ($l = 1, m = n = 0$):

$$N_{100}(\alpha) = \left(\frac{2\alpha}{\pi}\right)^{3/4} \cdot \sqrt{\frac{4\alpha}{1}} = 2^{5/4}\alpha^{5/4}\pi^{-3/4}$$

For $\alpha = 1.0$: $N_{100} \approx 1.425$
(d) Contracted normalization: The contracted function $\chi = \sum_p d_p g_p$ has cross-terms:

$$\langle\chi|\chi\rangle = \sum_{p,q} d_p d_q \langle g_p|g_q\rangle = \sum_{p,q} d_p d_q S_{pq}$$

The primitive overlap $S_{pq} \neq \delta_{pq}$ when $\alpha_p \neq \alpha_q$, so the contracted normalization is *not* $\sum_p d_p N_p$ but involves all pairwise overlaps. Modern basis sets provide pre-normalized contraction coefficients.