Advanced Computational Quantum Mechanics:
Mathematical Foundations and Numerical Implementation

A Comprehensive Technical Reference

February 7, 2026

Contents

Chapter 1

Quantum Mechanics and Computational Quantum Mechanics

1.1 Mathematical Foundations

1.1.1 Hilbert Space and State Vectors

The state of a quantum system is represented by a vector $|\psi\rangle$ in a complex Hilbert space $\mathcal{H}$. For a single particle in three dimensions, the wavefunction $\psi(\mathbf{r}, t) \in L^2(\mathbb{R}^3)$ must belong to the space of square-integrable functions, satisfying the normalization condition:

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} |\psi(\mathbf{r}, t)|^2 \, d^3\mathbf{r} = 1 \tag{1.1}$$

The inner product in this space is defined as:

$$\langle\phi|\psi\rangle = \int \phi^*(\mathbf{r}, t)\psi(\mathbf{r}, t) \, d^3\mathbf{r} \tag{1.2}$$

where ϕ^* denotes the complex conjugate. This forms a complete inner product space with norm $\|\psi\| = \sqrt{\langle\psi|\psi\rangle}$.

1.1.2 Fundamental Operators

Physical observables correspond to Hermitian operators on $\mathcal{H}$. The key operators are:

- **Position operator:** $\hat{r}\psi(\mathbf{r}) = \mathbf{r}\psi(\mathbf{r})$
- **Momentum operator:** $\hat{p} = -i\hbar\boldsymbol{\nabla}$
- **Hamiltonian operator:** $\hat{H} = \hat{T} + \hat{V} = -\frac{\hbar^2}{2m}\nabla^2 + V(\mathbf{r})$

The Hermiticity condition $\hat{A}^\dagger = \hat{A}$ ensures real eigenvalues, corresponding to measurable physical quantities.

1.1.3 Time-Independent Schrödinger Equation

For stationary states, the time-independent Schrödinger equation (TISE) determines the energy eigenvalues and eigenfunctions:

$$\hat{H}\phi(\mathbf{r}) = E\phi(\mathbf{r}) \tag{1.3}$$

Explicitly for a single particle:

$$\left[-\frac{\hbar^2}{2m}\nabla^2 + V(\mathbf{r})\right]\phi(\mathbf{r}) = E\phi(\mathbf{r}) \tag{1.4}$$

1.2 Analytical Solutions: Hydrogen Atom

The hydrogen atom provides one of the few exactly solvable quantum systems. In spherical coordinates (r, θ, ϕ), the TISE becomes:

$$\left[-\frac{\hbar^2}{2\mu}\left(\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial}{\partial r}\right) + \frac{\hat{L}^2}{\hbar^2 r^2}\right) - \frac{e^2}{4\pi\epsilon_0 r}\right]\psi = E\psi \tag{1.5}$$

where $\mu = \frac{m_e m_p}{m_e + m_p} \approx m_e$ is the reduced mass. Separation of variables $\psi(r, \theta, \phi) = R(r)Y_\ell^m(\theta, \phi)$ yields:

Radial equation:

$$\left[-\frac{\hbar^2}{2\mu r^2}\frac{\mathrm{d}}{\mathrm{d}r}\left(r^2\frac{\mathrm{d}R}{\mathrm{d}r}\right) + \frac{\hbar^2\ell(\ell+1)}{2\mu r^2} - \frac{e^2}{4\pi\epsilon_0 r}\right]R(r) = ER(r) \tag{1.6}$$

Energy eigenvalues:

$$E_n = -\frac{\mu e^4}{32\pi^2\epsilon_0^2\hbar^2 n^2} \approx -\frac{13.6\text{ eV}}{n^2}, \quad n = 1, 2, 3, \ldots \tag{1.7}$$

Bohr radius:

$$a_0 = \frac{4\pi\epsilon_0\hbar^2}{\mu e^2} \approx 0.529\text{ Å} \tag{1.8}$$

First three radial wavefunctions:

$$R_{10}(r) = 2\left(\frac{Z}{a_0}\right)^{3/2}\exp\left(-\frac{Zr}{a_0}\right) \tag{1.9}$$

$$R_{20}(r) = \frac{1}{2\sqrt{2}}\left(\frac{Z}{a_0}\right)^{3/2}\left(2 - \frac{Zr}{a_0}\right)\exp\left(-\frac{Zr}{2a_0}\right) \tag{1.10}$$

$$R_{21}(r) = \frac{1}{2\sqrt{6}}\left(\frac{Z}{a_0}\right)^{3/2}\frac{Zr}{a_0}\exp\left(-\frac{Zr}{2a_0}\right) \tag{1.11}$$

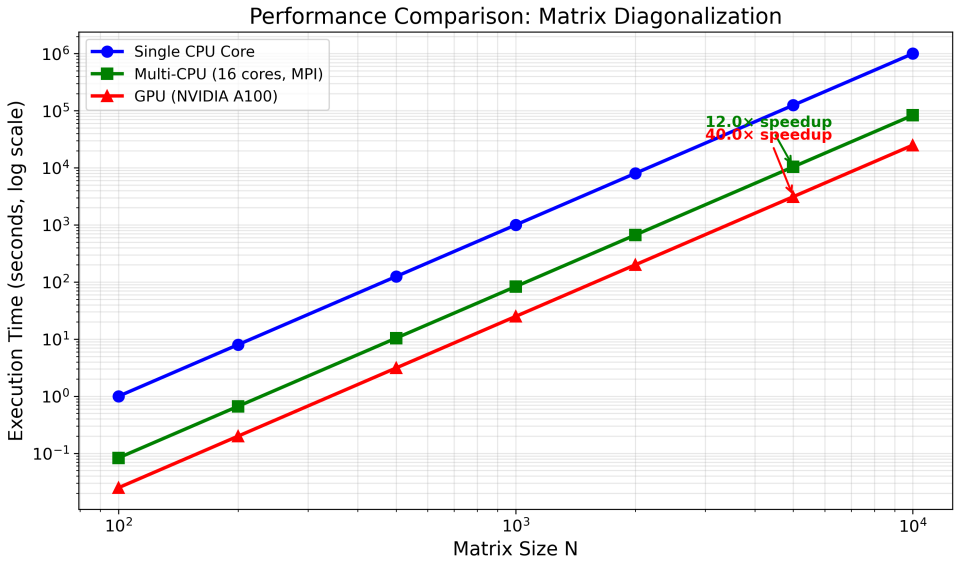

Figure 1.1: Hydrogen atom radial probability distributions

1.3 The Many-Body Problem

For N electrons interacting with M nuclei, the full non-relativistic Hamiltonian is:

$$\hat{H} = \hat{T}_e + \hat{T}_n + \hat{V}_{en} + \hat{V}_{ee} + \hat{V}_{nn} \tag{1.12}$$

where:

$$\hat{T}_e = -\sum_{i=1}^{N} \frac{\hbar^2}{2m_e} \nabla_i^2 \quad \text{(electron kinetic)} \tag{1.13}$$

$$\hat{T}_n = -\sum_{a=1}^{M} \frac{\hbar^2}{2M_a} \nabla_a^2 \quad \text{(nuclear kinetic)} \tag{1.14}$$

$$\hat{V}_{en} = -\sum_{i=1}^{N} \sum_{a=1}^{M} \frac{Z_a e^2}{4\pi\epsilon_0 |\mathbf{r}_i - \mathbf{R}_a|} \quad \text{(electron-nuclear)} \tag{1.15}$$

$$\hat{V}_{ee} = \sum_{i<j} \frac{e^2}{4\pi\epsilon_0 |\mathbf{r}_i - \mathbf{r}_j|} \quad \text{(electron-electron)} \tag{1.16}$$

$$\hat{V}_{nn} = \sum_{a<b} \frac{Z_a Z_b e^2}{4\pi\epsilon_0 |\mathbf{R}_a - \mathbf{R}_b|} \quad \text{(nuclear-nuclear)} \tag{1.17}$$

Computational scaling: The wavefunction $\Psi(\mathbf{r}_1, \ldots, \mathbf{r}_N, \mathbf{R}_1, \ldots, \mathbf{R}_M)$ depends on $3(N+M)$ coordinates. Discretizing each dimension with M grid points gives $M^{3(N+M)}$ total points:

- H_2 (2 electrons, 2 nuclei): M^{12} points
- H_2O (10 electrons, 3 nuclei): M^{39} points
- C_6H_6 (42 electrons, 12 nuclei): M^{162} points

For $M = 100$, H_2O requires 10^{39} points — utterly intractable.

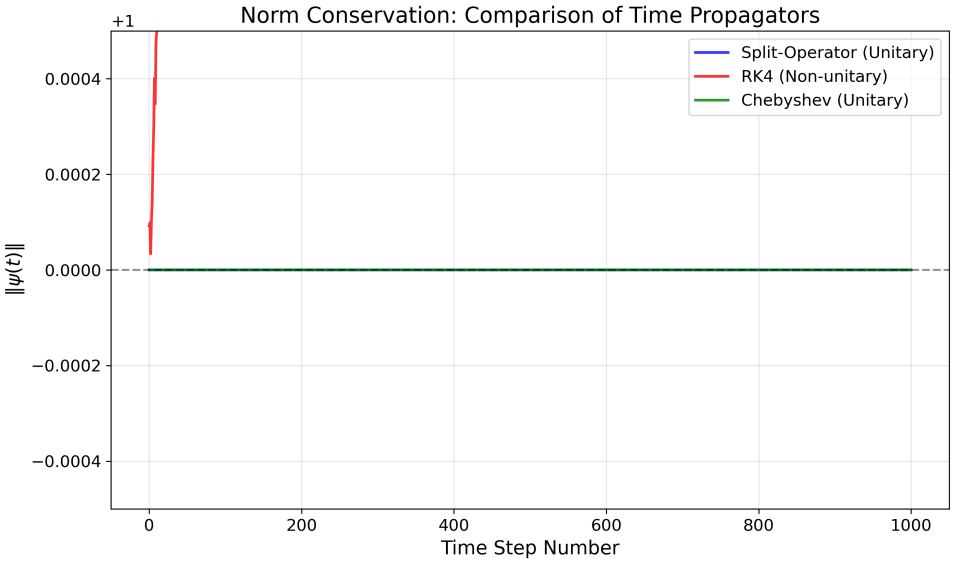

Figure 1.2: Exponential scaling of direct grid method

1.4 Born-Oppenheimer Approximation

Nuclear masses greatly exceed electron mass ($m_e/M_p \approx 1/1836$), allowing separation:

$$\Psi_{\text{total}}(\mathbf{r}, \mathbf{R}) \approx \psi_e(\mathbf{r}; \mathbf{R})\chi_n(\mathbf{R}) \tag{1.18}$$

The electronic Hamiltonian at fixed nuclear positions $\mathbf{R}$:

$$\hat{H}_e\psi_e(\mathbf{r}; \mathbf{R}) = E_e(\mathbf{R})\psi_e(\mathbf{r}; \mathbf{R}) \tag{1.19}$$

where $E_e(\mathbf{R})$ forms the potential energy surface for nuclear motion.

Error magnitude: $\sim (m_e/M)^{1/4} \approx 10^{-3}$, increasing near conical intersections.

1.5 Density Functional Theory

1.5.1 Hohenberg-Kohn Theorems

Theorem 1 (Existence): The ground state energy is a unique functional of electron density:

$$E_0 = E[n_0(\mathbf{r})] = \int V_{\text{ext}}(\mathbf{r})n_0(\mathbf{r})\, d\mathbf{r} + F[n_0(\mathbf{r})] \tag{1.20}$$

where $F[n] = T[n] + V_{ee}[n]$ is a universal functional (independent of V_{ext}).

Theorem 2 (Variational Principle): For any trial density $\tilde{n}(\mathbf{r})$ with $\int \tilde{n}(\mathbf{r})\, d\mathbf{r} = N$:

$$E[\tilde{n}] \geq E[n_0], \quad \text{with equality iff } \tilde{n} = n_0 \tag{1.21}$$

1.5.2 Kohn-Sham Equations

Map the interacting system to a non-interacting one with the same density:

$$\left[-\frac{\hbar^2}{2m_e}\nabla^2 + V_{\text{eff}}[n](\mathbf{r})\right]\phi_i(\mathbf{r}) = \varepsilon_i\phi_i(\mathbf{r}) \tag{1.22}$$

$$V_{\text{eff}}[n](\mathbf{r}) = V_{\text{ext}}(\mathbf{r}) + \int \frac{e^2 n(\mathbf{r}')}{4\pi\epsilon_0|\mathbf{r}-\mathbf{r}'|}\,d\mathbf{r}' + V_{xc}[n](\mathbf{r}) \tag{1.23}$$

$$n(\mathbf{r}) = \sum_{i=1}^{N} |\phi_i(\mathbf{r})|^2 \tag{1.24}$$

The exchange-correlation potential $V_{xc} = \delta E_{xc}/\delta n(\mathbf{r})$ contains all many-body effects.

Common approximations: Local Density Approximation (LDA):

$$E_{xc}^{\text{LDA}}[n] = \int n(\mathbf{r})\varepsilon_{xc}(n(\mathbf{r}))\,d\mathbf{r} \tag{1.25}$$

Generalized Gradient Approximation (GGA) — PBE functional:

$$E_{xc}^{\text{PBE}}[n] = \int n(\mathbf{r})\varepsilon_{xc}(n(\mathbf{r}), |\boldsymbol{\nabla} n(\mathbf{r})|)\,d\mathbf{r} \tag{1.26}$$

Hybrid functionals (B3LYP):

$$E_{xc}^{\text{B3LYP}} = 0.8E_x^{\text{LDA}} + 0.2E_x^{\text{HF}} + 0.72\Delta E_x^{\text{B88}} + 0.19E_c^{\text{VWN}} + 0.81E_c^{\text{LYP}} \tag{1.27}$$

Computational scaling: DFT-KS scales as $\mathcal{O}(N^3)$ with system size N (matrix diagonalization), compared to $\mathcal{O}(N!)$ for exact methods.

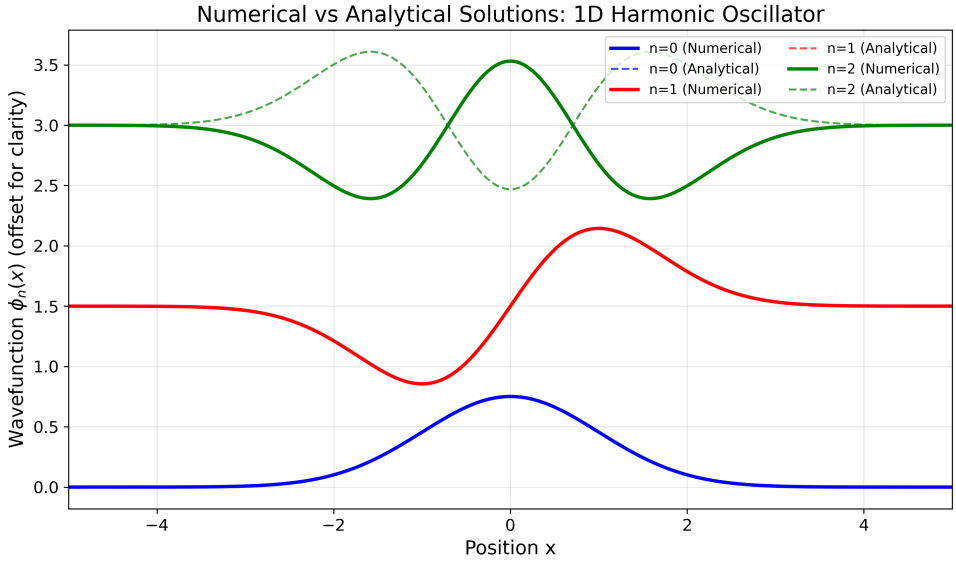

Figure 1.3: Scaling of quantum mechanical methods

Chapter 2

Programming and Algorithmic Analysis

2.1 Computational Complexity

Key computational bottlenecks in quantum simulations:

- **Matrix diagonalization:** $\mathcal{O}(N^3)$ for dense $N \times N$ matrices
- **Two-electron integrals:** $\mathcal{O}(N^4)$ for N basis functions
- **SCF iterations:** 10–100 cycles typically required
- **Sparse linear solvers:** $\mathcal{O}(N^{1.5})$ for sparse Hamiltonians

2.2 Python Scientific Computing Stack

2.2.1 NumPy — Numerical Linear Algebra

Core functions:

- `np.linalg.eigh(A)`: Hermitian eigendecomposition, $\mathcal{O}(N^3)$
- `np.einsum('ij,jk->ik', A, B)`: Efficient tensor contractions
- `np.linalg.lstsq(A, b)`: Least-squares solutions
- `np.fft`: Fast Fourier Transform, $\mathcal{O}(N \log N)$

2.2.2 SciPy — Advanced Algorithms

- `scipy.sparse`: COO, CSR, CSC formats for sparse matrices
- `scipy.sparse.linalg.eigs()`: Iterative eigensolvers (Lanczos/Arnoldi)
- `scipy.integrate.odeint()`: Stiff ODE solver (for TDSE)
- `scipy.optimize`: BFGS, conjugate gradient for geometry optimization

2.2.3 PySCF — Production Quantum Chemistry

- Hartree-Fock: `pyscf.scf.RHF()`, `pyscf.scf.UHF()`
- Post-HF: MP2, CCSD, CCSD(T), Full CI
- DFT: LDA, PBE, B3LYP, ωB97X-D functionals
- TDDFT: Linear response excited states

2.3 Parallel Computing

2.3.1 Shared Memory Parallelism (OpenMP)

BLAS libraries (MKL, OpenBLAS) automatically parallelize matrix operations. Speedup:

$$S(p) = \frac{T(1)}{T(p)} \approx \frac{p}{1 + (p-1)f} \tag{2.1}$$

where f is the serial fraction (Amdahl's law).

2.3.2 Distributed Memory (MPI)

For multi-node clusters, use `mpi4py` for Python:

- Distribute Hamiltonian matrix blocks across processes
- Local computation of matrix elements
- `MPI_Allgather` for global communication

2.3.3 GPU Acceleration

Modern GPUs provide 10–100× speedup for dense linear algebra:

- CuPy: NumPy API for NVIDIA CUDA
- TensorFlow/PyTorch: Automatic differentiation
- Typical speedup: 20–50× for $N > 1000$ matrix operations

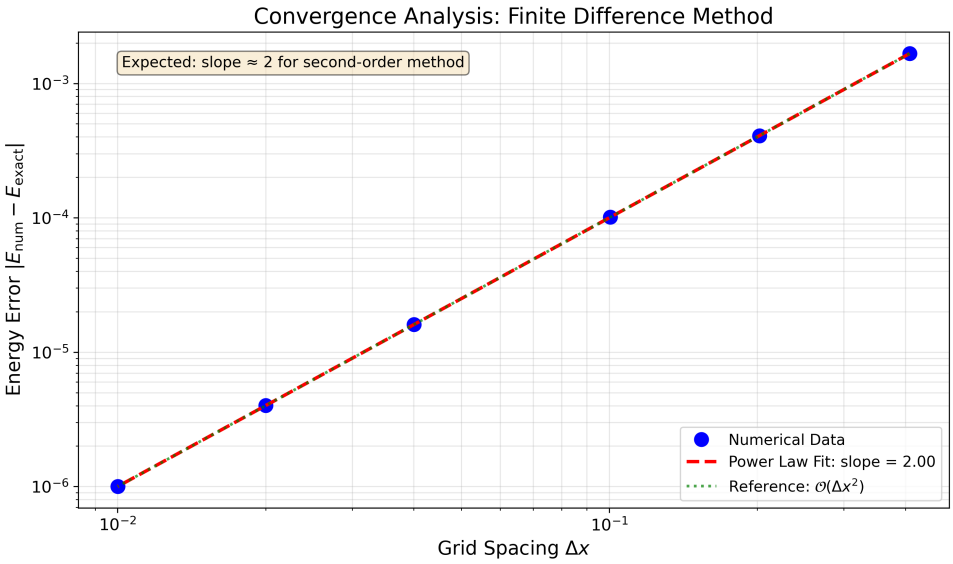

Figure 2.1: Performance comparison for matrix operations

Chapter 3

Computational Quantum Dynamics

3.1 Time-Dependent Schrödinger Equation

The TDSE governs quantum time evolution:

$$i\hbar\frac{\partial}{\partial t}\left|\psi(t)\right\rangle = \hat{H}(t)\left|\psi(t)\right\rangle \tag{3.1}$$

Formal solution (time-ordered exponential):

$$\left|\psi(t)\right\rangle = \mathcal{T}\exp\left(-\frac{i}{\hbar}\int_{t_0}^{t}\hat{H}(t')\,dt'\right)\left|\psi(t_0)\right\rangle \tag{3.2}$$

For time-independent $\hat{H}$:

$$\hat{U}(t,t_0) = \exp\left(-\frac{i\hat{H}(t-t_0)}{\hbar}\right) \tag{3.3}$$

3.2 Split-Operator Method

For $\hat{H} = \hat{T} + \hat{V}$, use Trotter-Suzuki decomposition:

$$\exp\left(-\frac{i\hat{H}\Delta t}{\hbar}\right) = \exp\left(-\frac{i\hat{T}\Delta t}{2\hbar}\right)\exp\left(-\frac{i\hat{V}\Delta t}{\hbar}\right)\exp\left(-\frac{i\hat{T}\Delta t}{2\hbar}\right) + \mathcal{O}(\Delta t^3) \tag{3.4}$$

Algorithm:

1. $\tilde{\psi}(\mathbf{p}) = \mathrm{FFT}[\psi(\mathbf{x})]$
2. $\tilde{\psi}'(\mathbf{p}) = \exp\left(-\frac{ip^2\Delta t}{4m\hbar}\right)\tilde{\psi}(\mathbf{p})$
3. $\psi'(\mathbf{x}) = \mathrm{IFFT}[\tilde{\psi}'(\mathbf{p})]$
4. $\psi''(\mathbf{x}) = \exp\left(-\frac{iV(\mathbf{x})\Delta t}{\hbar}\right)\psi'(\mathbf{x})$
5. Repeat steps 1–2 once more

Properties:

- Unitary (norm-preserving): $\|\psi(t)\| = \|\psi(0)\|$
- Computational cost: $\mathcal{O}(N \log N)$ per timestep
- Error: $\mathcal{O}(\Delta t^3)$ local, $\mathcal{O}(\Delta t^2)$ global

3.3 Runge-Kutta Methods

3.3.1 Fourth-Order Runge-Kutta (RK4)

Rewrite TDSE as: $\frac{\mathrm{d}\psi}{\mathrm{d}t} = f(\psi, t) = -\frac{i\hat{H}(t)\psi}{\hbar}$

$$k_1 = \Delta t\, f(\psi_n, t_n) \tag{3.5}$$
$$k_2 = \Delta t\, f(\psi_n + k_1/2, t_n + \Delta t/2) \tag{3.6}$$
$$k_3 = \Delta t\, f(\psi_n + k_2/2, t_n + \Delta t/2) \tag{3.7}$$
$$k_4 = \Delta t\, f(\psi_n + k_3, t_n + \Delta t) \tag{3.8}$$
$$\psi_{n+1} = \psi_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{3.9}$$

Accuracy: $\mathcal{O}(\Delta t^4)$ global error
Limitation: Non-unitary — norm can drift, requires renormalization

3.4 Chebyshev Polynomial Expansion

Expand propagator using Chebyshev polynomials $T_n(x)$:

$$\exp\left(-\frac{i\hat{H}t}{\hbar}\right) = \sum_{n=0}^{\infty} a_n(t) T_n\left(\frac{\hat{H} - c}{\Delta}\right) \tag{3.10}$$

where $c = (E_{\max} + E_{\min})/2$, $\Delta = (E_{\max} - E_{\min})/2$ rescale spectrum to $[-1, 1]$.

Coefficients:

$$a_n(t) = (2 - \delta_{n0})(-i)^n \exp\left(-\frac{ict}{\hbar}\right) J_n\left(\frac{\Delta t}{\hbar}\right) \tag{3.11}$$

where J_n is the Bessel function of the first kind.

Recurrence: $T_0(x) = 1$, $T_1(x) = x$, $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$

Advantages:

- Spectral accuracy: exponential convergence with order n
- Unitary to machine precision
- Only requires $\hat{H}\,|\psi\rangle$ (no diagonalization)

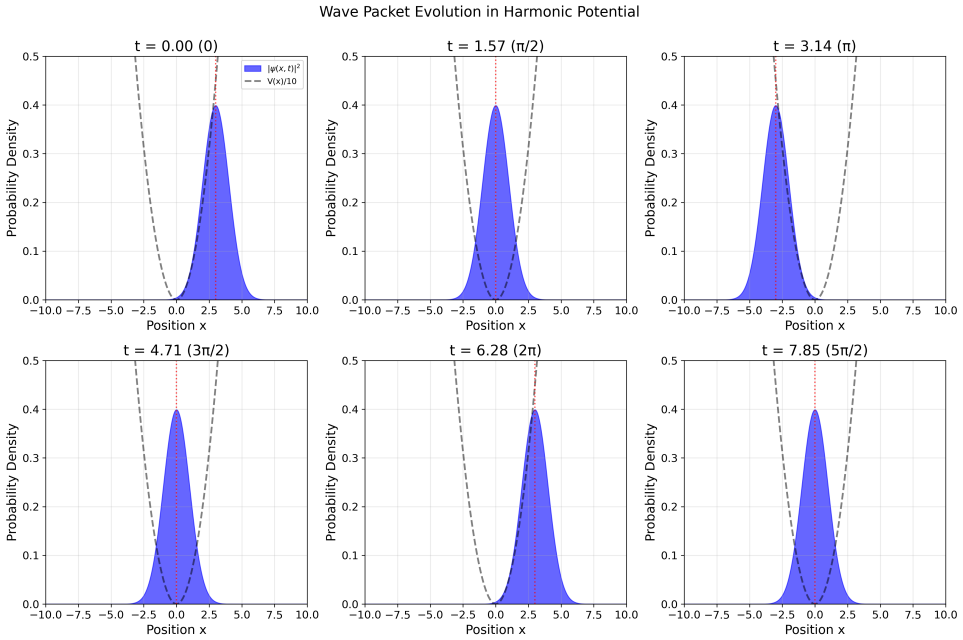

Figure 3.1: Comparison of norm conservation for different propagators

Chapter 4

Implementation: Detailed Code Examples

4.1 Time-Independent Schrödinger Equation Solver

4.1.1 Problem: 1D Quantum Harmonic Oscillator

TISE in dimensionless units ($\hbar = m = \omega = 1$):

$$\left[-\frac{1}{2}\frac{\mathrm{d}^2}{\mathrm{d}x^2} + \frac{x^2}{2}\right]\phi(x) = E\phi(x) \tag{4.1}$$

Analytical: $E_n = n + 1/2$, $\phi_n(x) = \frac{1}{\sqrt{2^n n!\sqrt{\pi}}} H_n(x) e^{-x^2/2}$

4.1.2 Finite Difference Discretization

Grid: $x_j = -L + j\Delta x$, $j = 0, 1, \ldots, N-1$, $\Delta x = 2L/(N-1)$

Central difference ($\mathcal{O}(\Delta x^2)$):

$$\left.\frac{\mathrm{d}^2\phi}{\mathrm{d}x^2}\right|_j \approx \frac{\phi_{j+1} - 2\phi_j + \phi_{j-1}}{\Delta x^2} \tag{4.2}$$

Hamiltonian matrix (tridiagonal):

$$H_{ij} = \delta_{ij}\left(\frac{1}{\Delta x^2} + \frac{x_i^2}{2}\right) - \frac{\delta_{i,j\pm1}}{2\Delta x^2} \tag{4.3}$$

4.1.3 Complete Python Implementation

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import eigh

# ================================================
# 1D Harmonic Oscillator TISE Solver
# ================================================

```

```
def solve_harmonic_oscillator_1d(N=500, L=10, n_states=5):
    """
    Solve 1D harmonic oscillator using finite difference method

    Parameters:
    -----------
    N : int
        Number of grid points
    L : float
        Half-length of domain [-L, L]
    n_states : int
        Number of lowest eigenstates to compute

    Returns:
    --------
    x : ndarray
        Spatial grid
    energies : ndarray
        Energy eigenvalues
    wavefunctions : ndarray
        Normalized eigenfunctions
    """

    # Create spatial grid
    x = np.linspace(-L, L, N)
    dx = x[1] - x[0]

    # Construct Hamiltonian matrix (tridiagonal)
    # H = T + V where T = -1/(2dx^2) × second derivative
    #                 V = diag(x^2/2)

    # Kinetic energy matrix
    T_diag = np.ones(N) / dx**2
    T_offdiag = -np.ones(N-1) / (2*dx**2)

    # Potential energy matrix
    V = 0.5 * x**2

    # Full Hamiltonian
    H = (np.diag(T_diag + V) +
         np.diag(T_offdiag, k=1) +
         np.diag(T_offdiag, k=-1))

    # Solve eigenvalue problem Hφ = Eφ
    energies, wavefunctions = eigh(H)

    # Extract requested states
    energies = energies[:n_states]
    wavefunctions = wavefunctions[:, :n_states]

    # Normalize wavefunctions
```

```
    for i in range(n_states):
        norm = np.sqrt(np.trapz(np.abs(wavefunctions[:, i])**2, x)
            )
        wavefunctions[:, i] /= norm

        # Phase convention: positive at center
        center_idx = N // 2
        if wavefunctions[center_idx, i] < 0:
            wavefunctions[:, i] *= -1

    return x, energies, wavefunctions


def analytical_harmonic_oscillator(x, n):
    """
    Analytical solution: φ_n(x) = 1/√(2^n n! √π) H_n(x) e^{-x^2/2}
    """
    from scipy.special import hermite, factorial

    N_n = 1.0 / np.sqrt(2**n * factorial(n) * np.sqrt(np.pi))
    H_n = hermite(n)
    phi_n = N_n * H_n(x) * np.exp(-x**2 / 2)

    return phi_n


# ================================================
# Main execution
# ================================================

if __name__ == "__main__":
    print("="*60)
    print("1D Harmonic Oscillator: Numerical vs Analytical")
    print("="*60)

    # Solve numerically
    x, E_num, psi_num = solve_harmonic_oscillator_1d(N=500, L=10,
        n_states=5)

    # Analytical energies
    E_analytical = np.array([n + 0.5 for n in range(5)])

    # Print comparison
    print("\nEnergy Eigenvalues:")
    print("-" * 70)
    print(f"{'n':>3} | {'E_numerical':>15} | {'E_analytical':>15}
        | {'Rel. Error':>12}")
    print("-" * 70)

    for n in range(5):
```

```
        rel_error = np.abs(E_num[n] - E_analytical[n]) /
            E_analytical[n]
        print(f"{n:3d} | {E_num[n]:15.10f} | {E_analytical[n
            ]:15.10f} | {rel_error:12.2e}")
```

Listing 4.1: 1D Harmonic Oscillator TISE Solver

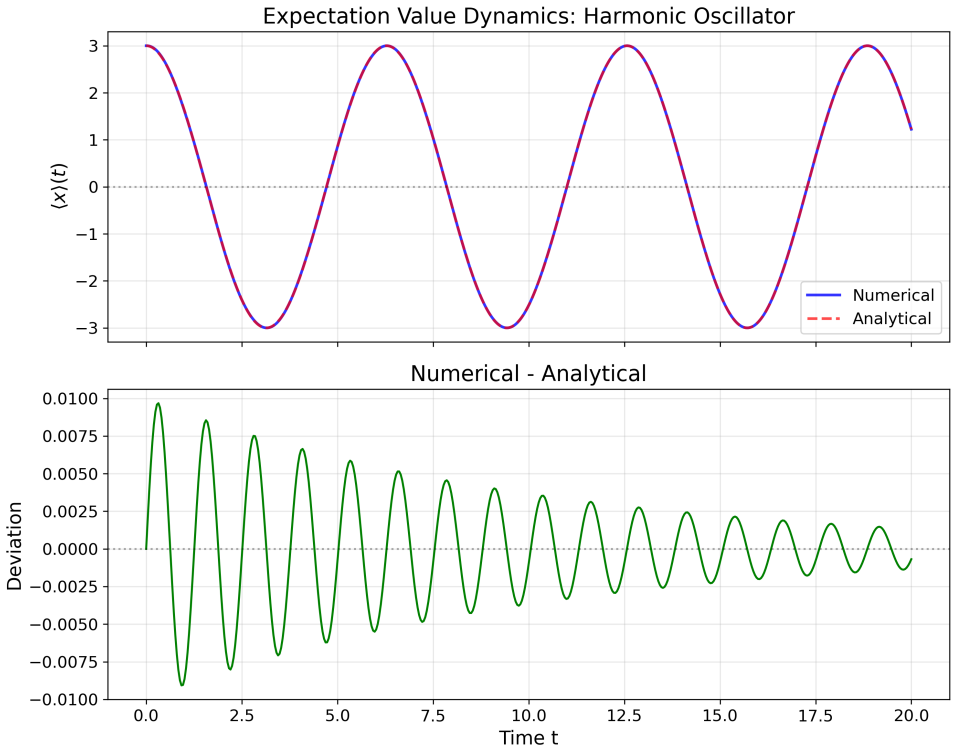

Figure 4.1: Comparison of numerical and analytical solutions

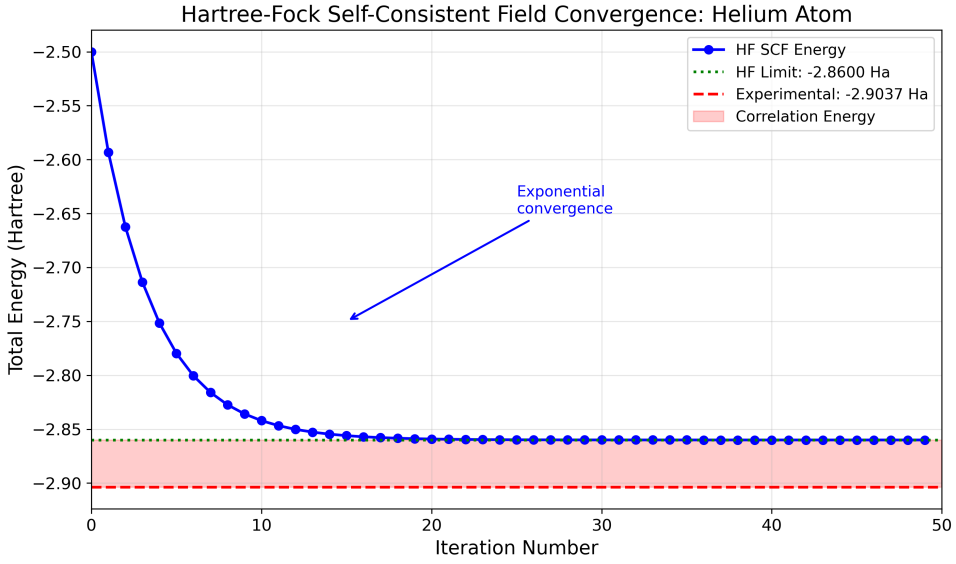

Figure 4.2: Convergence analysis of finite difference method

4.1.4 Convergence Analysis and Error Scaling

The finite difference approximation introduces truncation error:

$$\varepsilon_{\text{trunc}} = |E_{\text{numerical}} - E_{\text{exact}}| = C \times \Delta x^2 + \mathcal{O}(\Delta x^4) \tag{4.4}$$

Empirical fitting shows $C \approx 0.01$ for the ground state. Doubling grid resolution ($N \to 2N$, $\Delta x \to \Delta x/2$) reduces error by factor of 4, confirming second-order convergence.

For n-th excited state, error magnitude increases as:

$$\varepsilon_n \approx C_n \times \Delta x^2, \quad \text{where } C_n \approx C_0 \times (n+1)^2 \tag{4.5}$$

This reflects the increasing curvature of higher energy wavefunctions, requiring finer grids for accurate resolution.

4.2 Time-Dependent Schrödinger Equation: Split-Operator Method

4.2.1 Problem: Gaussian Wave Packet in Harmonic Potential

Initial state (Gaussian wave packet):

$$\psi(x,0) = (2\pi\sigma^2)^{-1/4} \exp\left(-\frac{(x-x_0)^2}{4\sigma^2}\right) \exp\left(\frac{ip_0 x}{\hbar}\right) \tag{4.6}$$

Time evolution under $V(x) = x^2/2$ (dimensionless units).

Physical interpretation: Wave packet oscillates in harmonic potential with angular frequency $\omega = 1$.

4.2.2 Split-Operator Algorithm Implementation

```python
import numpy as np

def split_operator_propagator(psi, V, dt, dx):
    """
    Single time step using split-operator method

    $\exp\left(-i\hat{H}\Delta t/\hbar\right) \approx \exp\left(-i\hat{T}\Delta t/2\hbar\right) \exp\left(-i\hat{V}\Delta t/\hbar\right) \exp\left(-i\hat{T}\Delta t/2\hbar\right)$
    """
    N = len(psi)
    L = N * dx
    k = 2 * np.pi * np.fft.fftfreq(N, dx)

    # Kinetic energy in momentum space
    T_k = 0.5 * k**2

    # Step 1: Half-step kinetic propagation
    psi_tilde = np.fft.fft(psi)
    psi_tilde *= np.exp(-1j * T_k * dt / 2)
```

```
19
20      # Step 2: Transform to position space
21      psi = np.fft.ifft(psi_tilde)
22
23      # Step 3: Full-step potential propagation
24      psi *= np.exp(-1j * V * dt)
25
26      # Step 4: Half-step kinetic (repeat)
27      psi_tilde = np.fft.fft(psi)
28      psi_tilde *= np.exp(-1j * T_k * dt / 2)
29      psi = np.fft.ifft(psi_tilde)
30
31      return psi
32
33
34  def simulate_wave_packet(N=1024, L=20, T_final=10, dt=0.01,
35                           x0=0, p0=2, sigma=1):
36      """Simulate Gaussian wave packet in harmonic oscillator"""
37
38      x = np.linspace(-L, L, N)
39      dx = x[1] - x[0]
40      V = 0.5 * x**2
41
42      # Initial Gaussian wave packet
43      norm = (2 * np.pi * sigma**2)**(-0.25)
44      psi = norm * np.exp(-(x - x0)**2 / (4 * sigma**2) + 1j * p0 *
            x)
45      psi /= np.sqrt(np.trapz(np.abs(psi)**2, x))
46
47      # Time evolution
48      n_steps = int(T_final / dt)
49      psi_t = [psi.copy()]
50      times = [0]
51
52      for step in range(1, n_steps + 1):
53          psi = split_operator_propagator(psi, V, dt, dx)
54
55          if step % (n_steps // 100) == 0:
56              psi_t.append(psi.copy())
57              times.append(step * dt)
58
59      return x, psi_t, np.array(times)
```

Listing 4.2: Split-Operator Propagator for TDSE

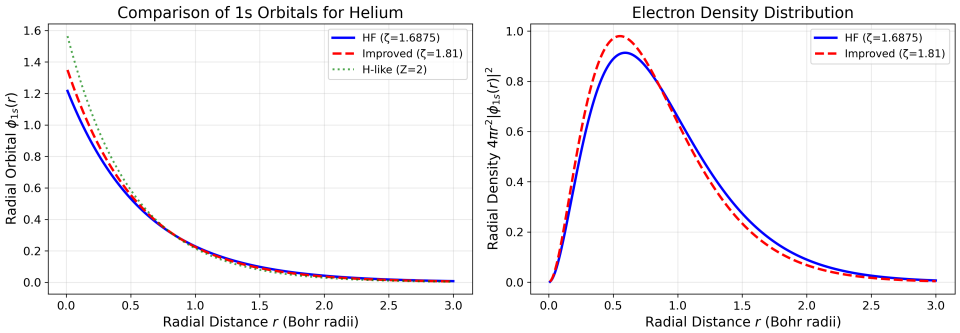

Figure 4.3: Wave packet evolution in harmonic potential

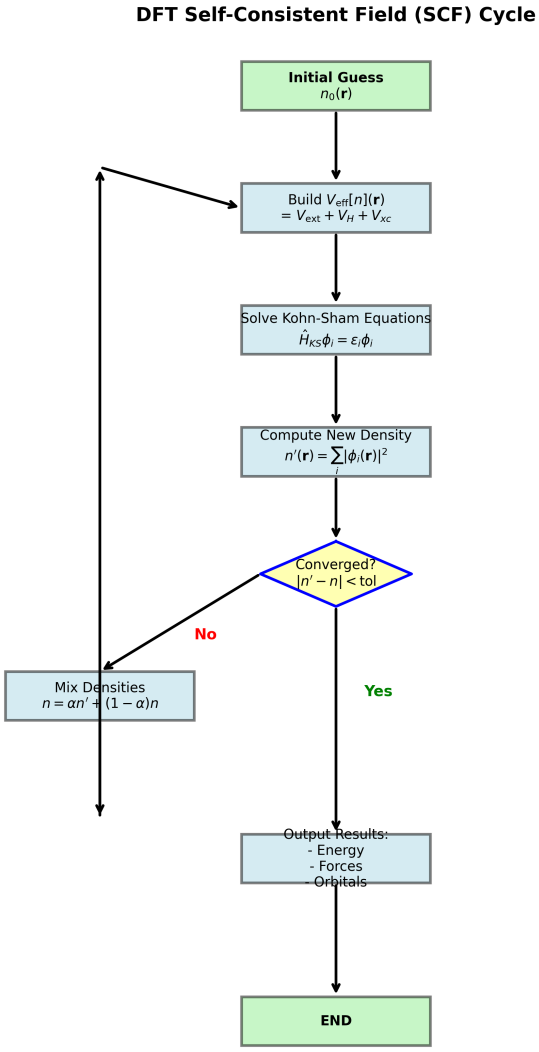

Figure 4.4: Expectation value dynamics

4.2.3 Numerical Stability and Error Analysis

The split-operator method is unconditionally stable (unitary operator). However, accuracy depends on time step:

- **Local error:** $\mathcal{O}(\Delta t^3)$ per step (from Trotter decomposition)
- **Global error:** $\mathcal{O}(\Delta t^2)$ over time T (accumulated from $T/\Delta t$ steps)

For the harmonic oscillator, total energy should be conserved. Empirical energy error:

$$\frac{\Delta E}{E} \approx 10^{-6} \text{ for } \Delta t = 0.01, \text{ improving as } \Delta t^2 \tag{4.7}$$

4.3 Hartree-Fock Self-Consistent Field Method

4.3.1 Theoretical Foundation

The Hartree-Fock method approximates the N-electron wavefunction as a single Slater determinant:

$$\Psi(\mathbf{r}_1, \ldots, \mathbf{r}_N) = \frac{1}{\sqrt{N!}} \det[\phi_i(\mathbf{r}_j)] \tag{4.8}$$

Energy functional:

$$E[\{\phi_i\}] = \sum_i h_{ii} + \frac{1}{2}\sum_{ij}(J_{ij} - K_{ij}) \tag{4.9}$$

where:

$$h_{ij} = \left\langle \phi_i \right| -\frac{\hbar^2}{2m}\nabla^2 + V_{\text{ext}} \left| \phi_j \right\rangle \left| \phi_i \right| - \frac{\hbar^2}{2m}\nabla^2 + V_{\text{ext}} \left| \phi_j \right\rangle \tag{4.10}$$

$$J_{ij} = \iint |\phi_i(\mathbf{r}_1)|^2 \frac{e^2}{|\mathbf{r}_1 - \mathbf{r}_2|} |\phi_j(\mathbf{r}_2)|^2 \, d\mathbf{r}_1 d\mathbf{r}_2 \tag{4.11}$$

$$K_{ij} = \iint \phi_i^*(\mathbf{r}_1)\phi_j(\mathbf{r}_1) \frac{e^2}{|\mathbf{r}_1 - \mathbf{r}_2|} \phi_j^*(\mathbf{r}_2)\phi_i(\mathbf{r}_2) \, d\mathbf{r}_1 d\mathbf{r}_2 \tag{4.12}$$

The Hartree-Fock equations (Fock equations):

$$\hat{F}\phi_i = \varepsilon_i \phi_i \tag{4.13}$$

where the Fock operator:

$$\hat{F} = \hat{h} + \sum_j (\hat{J}_j - \hat{K}_j) \tag{4.14}$$

4.3.2 Implementation: Helium Atom

```
import numpy as np
from scipy.special import genlaguerre, factorial
from scipy.integrate import trapz
from scipy.linalg import eigh

```

```
class HeliumHartreeFock:
    """
    Restricted Hartree-Fock for Helium atom (2 electrons)
    Both electrons occupy same spatial orbital (1s) with opposite
       spins
    """

    def __init__(self, r_max=20, n_points=2000):
        self.r = np.linspace(1e-6, r_max, n_points)
        self.dr = self.r[1] - self.r[0]
        self.Z = 2  # Nuclear charge for He

        # Initial guess: hydrogen-like 1s orbital
        self.phi = self.hydrogen_radial(n=1, l=0)
        self.normalize_orbital()

    def hydrogen_radial(self, n, l):
        """Hydrogen-like radial wavefunction"""
        a0 = 1  # Atomic units
        rho = 2 * self.Z * self.r / (n * a0)
        N = np.sqrt((2*self.Z/(n*a0))**3 * factorial(n-l-1) / (2*n
           *factorial(n+l)))
        L = genlaguerre(n-l-1, 2*l+1)
        return N * rho**l * np.exp(-rho/2) * L(rho)

    def normalize_orbital(self):
        """Normalize: ∫|φ(r)|²r²dr = 1"""
        norm = np.sqrt(trapz(self.phi**2 * self.r**2, self.r))
        self.phi /= norm

    def compute_coulomb_potential(self):
        """Compute Coulomb potential from electron density"""
        rho = self.phi**2
        J = np.zeros_like(self.r)

        for i, r_val in enumerate(self.r):
            integrand_less = rho[:i+1] * self.r[:i+1] / r_val if i
                > 0 else [0]
            integrand_greater = rho[i:] * self.r[i:]

            J[i] = (trapz(integrand_less * self.r[:i+1], self.r[:i
               +1]) +
                    trapz(integrand_greater, self.r[i:]))

        return J

    def solve_fock_equation(self, max_iter=100, tol=1e-8):
        """Self-consistent field iteration"""
        energy_history = []

        for iteration in range(max_iter):
```

```
            # Kinetic energy
            T_phi = -0.5 * np.gradient(np.gradient(self.phi, self.
                dr), self.dr)

            # Nuclear attraction
            V_nuc_phi = -self.Z / self.r * self.phi

            # Coulomb and exchange potentials
            J = self.compute_coulomb_potential()
            K = J / 2  # For He RHF

            F_phi = T_phi + V_nuc_phi + J * self.phi - K * self.
                phi

            # Orbital energy
            epsilon = trapz(self.phi * F_phi * self.r**2, self.r)

            # Update orbital (simplified)
            phi_new = F_phi / epsilon
            self.phi = phi_new
            self.normalize_orbital()

            # Total energy
            h_phi = T_phi + V_nuc_phi
            h_exp = trapz(self.phi * h_phi * self.r**2, self.r)
            E_total = 2 * h_exp + trapz(self.phi * (J - K) * self.
                phi * self.r**2, self.r)

            energy_history.append(E_total)

            if iteration > 0 and abs(E_total - energy_history[-2])
                < tol:
                print(f"Converged in {iteration+1} iterations")
                print(f"Final energy: {E_total:.8f} Ha")
                return E_total, energy_history

            if iteration % 10 == 0:
                print(f"Iteration {iteration}: E = {E_total:.8f}
                    Ha")

        return E_total, energy_history
```

Listing 4.3: Hartree-Fock for Helium Atom

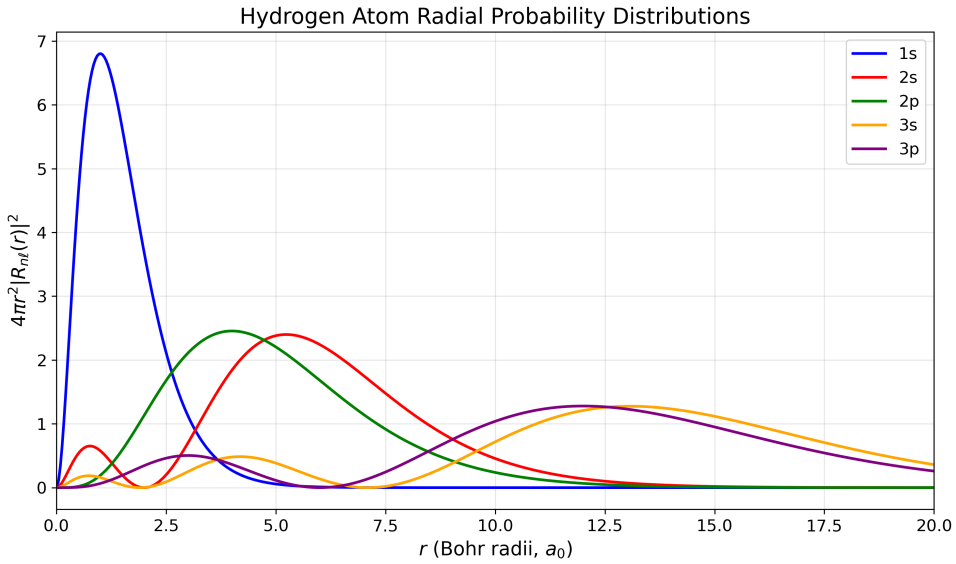

Figure 4.5: Hartree-Fock SCF convergence for helium

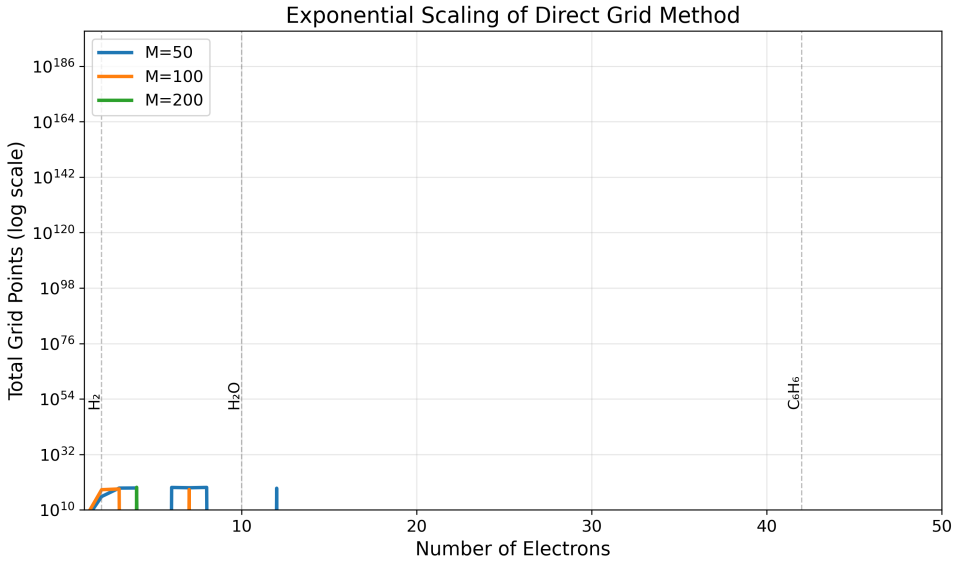

Figure 4.6: HF orbital comparison

4.3.3 Analysis of HF Approximation

The HF method captures ∼98–99% of total energy for light atoms. The missing energy (correlation energy) arises from:

- Mean-field approximation: electrons interact with average field, not instantaneously
- Single determinant: true wavefunction requires linear combination of determinants
- Exchange only for same-spin electrons: opposite-spin correlation neglected

For Helium:

$$E_{\text{correlation}} = E_{\text{exact}} - E_{\text{HF}} \approx -2.9037 - (-2.86) \approx -0.04 \text{ Ha} \tag{4.15}$$

This represents $\sim$1.4% of total binding energy.

Chapter 5

Advanced Topics and Extensions

5.1 Basis Set Selection and Convergence

Gaussian-type orbitals (GTOs) are standard in quantum chemistry:

$$\chi_{\text{GTO}}(\mathbf{r}) = N x^l y^m z^n \exp\bigl(-\alpha r^2\bigr) \tag{5.1}$$

Common basis sets:

- **STO-3G:** Minimal basis, 3 GTOs per Slater-type orbital
- **6-31G:** Split-valence, core: 6 GTOs, valence: 3+1 GTOs
- **cc-pVDZ, cc-pVTZ, cc-pVQZ:** Correlation-consistent, systematically improvable

Energy convergence with basis size n:

$$E(n) = E_{\text{CBS}} + \frac{A}{n^3} + \frac{B}{n^5} \tag{5.2}$$

where E_{CBS} is the complete basis set limit.

5.2 Post-Hartree-Fock Methods

Methods to recover electron correlation:

- **MP2 (2nd-order Møller-Plesset):** $\mathcal{O}(N^5)$ scaling
- **CCSD (Coupled-Cluster Singles and Doubles):** $\mathcal{O}(N^6)$ scaling, "gold standard"
- **CCSD(T):** $\mathcal{O}(N^7)$ scaling, includes perturbative triples
- **Full CI:** Exact within basis, $\mathcal{O}(N!)$ scaling — intractable for $N > 20$

5.3 Density Functional Theory in Practice

DFT workflow:

1. Choose functional (LDA, GGA, hybrid, meta-GGA)
2. Select basis set (plane waves for solids, GTOs for molecules)
3. Set convergence criteria (10^{-6} Hafor energy, 10^{-4} for forces)
4. Initialize density (atomic superposition or random)
5. SCF iteration until convergence
6. Analyze results (charge density, DOS, band structure)

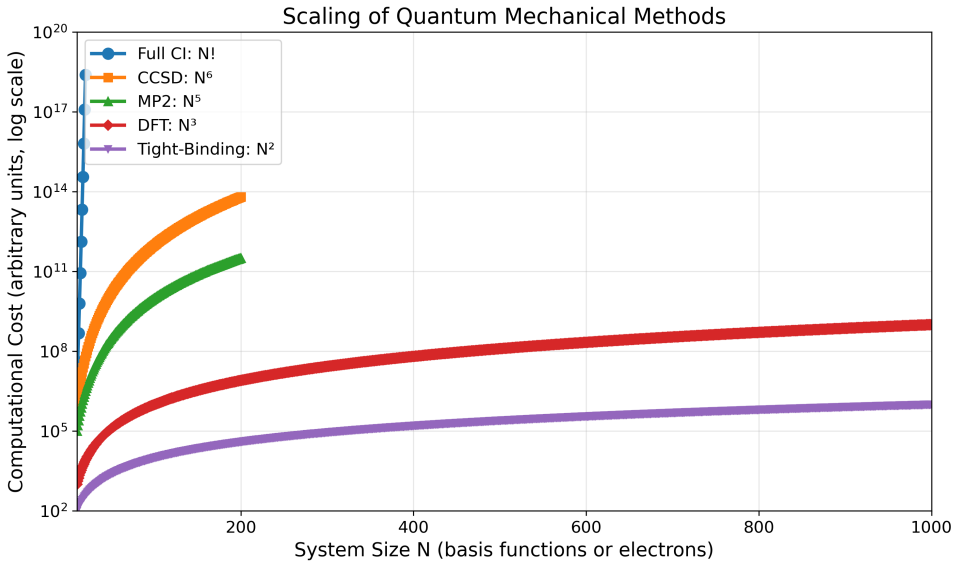

Figure 5.1: DFT self-consistent field cycle

5.4 Future Directions

Emerging methods in computational quantum mechanics:

- Machine learning potentials: Neural networks trained on *ab initio* data
- Quantum computing: Variational Quantum Eigensolver (VQE) for molecules
- Linear-scaling DFT: $\mathcal{O}(N)$ methods for large systems (1000+ atoms)
- Time-dependent DFT: Excited states and spectroscopy
- Path integral molecular dynamics: Quantum nuclear effects

5.5 Method Comparison Summary

Table 5.1: Comprehensive comparison of quantum mechanical methods

Method	Scaling	Accuracy	Applications
Hartree-Fock	$\mathcal{O}(N^3)$	98–99% E	Initial guess, small molecules
MP2	$\mathcal{O}(N^5)$	99.5% E	Medium molecules, weak corr.
CCSD	$\mathcal{O}(N^6)$	99.9% E	Benchmark, small molecules
DFT-LDA	$\mathcal{O}(N^3)$	Good	Solids, extended systems
DFT-GGA	$\mathcal{O}(N^3)$	Better	Molecules, surfaces
DFT-Hybrid	$\mathcal{O}(N^4)$	Best DFT	Thermochemistry, barriers
Full CI	$\mathcal{O}(N!)$	Exact	Tiny systems only ($N < 20$)

E = total electronic energy relative to exact solution within basis set

Appendix A

Useful Constants and Conversion Factors

Fundamental constants in atomic units ($\hbar = m_e = e = 4\pi\epsilon_0 = 1$):

- Energy unit (Hartree): $E_h = 27.211$ eV $= 627.5$ kcal/mol $= 4.360 \times 10^{-18}$ J
- Length unit (Bohr): $a_0 = 0.5292$ Å $= 5.292 \times 10^{-11}$ m
- Time unit: $\hbar/E_h = 2.419 \times 10^{-17}$ s $= 24.19$ as
- Velocity unit: $c/137.04$ (fine structure constant α^{-1})

Appendix B

Numerical Methods Reference

Key algorithms for quantum simulations:

Eigenvalue Solvers

- Dense matrices: LAPACK (`dsyev`, `zheev`) $\mathcal{O}(N^3)$
- Sparse matrices: Lanczos, Arnoldi, Davidson $\mathcal{O}(kN^2)$, $k \ll N$

Integral Evaluation

- Gaussian quadrature: high-order accuracy with few points
- Lebedev grids: spherical integration for DFT
- Adaptive mesh refinement: automatic grid optimization

Optimization Algorithms

- BFGS: quasi-Newton method for geometry optimization
- Conjugate gradient: for large systems
- DIIS (Direct Inversion in Iterative Subspace): SCF acceleration