PRD 20: Ab Initio Path Integral Molecular Dynamics

Pure Thought AI Challenges

January 19, 2026

Contents

Domain: Chemistry & Quantum Statistical Mechanics **Timeline**: 7-10 months **Difficulty**: High **Prerequisites**: Path integrals, statistical mechanics, quantum chemistry, Monte Carlo methods, numerical integration

—

0.1 1. Problem Statement

0.1.1 Scientific Context

Path Integral Molecular Dynamics (PIMD) revolutionizes the simulation of quantum many-body systems at finite temperature by mapping the quantum partition function onto a classical-like ensemble of "ring polymers." This approach, rooted in Feynman's path integral formulation (1948), treats nuclear quantum effects **exactly** within the Born-Oppenheimer approximation, without needing to solve the time-dependent Schrödinger equation.

The fundamental insight is **isomorphism**: a single quantum particle at temperature T is statistically equivalent to a classical ring polymer of P beads coupled by harmonic springs at temperature T. As P→, the discretized path integral converges to the exact quantum statistical mechanics. For finite P (typically 16-64), PIMD provides systematically improvable approximations with polynomial computational cost O(P·N^3) where N = number of atoms.

Nuclear Quantum Effects neglected by classical molecular dynamics include: - **Zero-Point Energy (ZPE)**: Even at T=0, quantum particles vibrate with energy $\frac{1}{2}$, affecting bond lengths, binding energies, and phase boundaries - **Quantum Tunneling**: Proton transfer reactions, H-atom diffusion in metals, and enzyme catalysis involve barrier penetration forbidden classically - **Quantum Delocalization**: Light nuclei (H, D, He) exhibit spatial spreading beyond thermal fluctuations, affecting structure factors and spectroscopy - **Isotope Effects**: H/D substitution changes vibrational frequencies by factor 2, leading to measurable kinetic isotope effects (KIE) in reaction rates

Ab Initio PIMD combines path integrals with quantum chemistry: the potential energy V(R) for each bead configuration is computed "on-the-fly" using density functional theory (DFT), Hartree-Fock (HF), or coupled-cluster methods. This avoids empirical force fields, enabling predictive simulations of chemical reactions, phase transitions, and spectroscopy where classical MD fails.

Applications: - **Hydrogen-bonded systems**: Water, ice, protonated clusters — ZPE weakens H-bonds by 10% - **Proton transfer**: Malonaldehyde tautomerization, enzymatic acid-base catalysis - **High-pressure phases**: Solid hydrogen metallization, He superfluidity in confined geometries - **Surface chemistry**: H dissociation on metals, quantum solvation effects in electrochemistry - **Isotope fractionation**: D/H ratios in geochemistry, paleoclimate proxies

Computational Challenge: Each PIMD step requires P independent electronic structure calculations (one per bead). For P=32 beads and DFT taking 1 second per geometry, a single MD step costs 32 seconds — making trajectory convergence expensive. Normal mode transformations, staging coordinates, and efficient integrators (PILE, PI-GLE) reduce this cost significantly.

0.1.2 Core Question

Can we implement ab initio PIMD achieving exact quantum statistical mechanics using ONLY path integral discretization and quantum chemistry — without solving the Schrödinger equation or using empirical potentials?

Specifically: 1. Discretize imaginary-time path integral into **P-bead ring polymers** with systematically improvable convergence 2. Compute electronic potential V(R) and forces for each bead using **ab initio methods** (HF, DFT, CCSD) 3. Sample the effective classical ensemble using **PIMD integrators** (normal modes, staging, PILE thermostat) 4. Extract **quantum thermodynamic properties** via virial, primitive, and centroid estimators 5. Validate against **exact solutions** (harmonic oscillator, H molecule) and benchmark **isotope effects** 6. Generate **certificates** with convergence proofs (P→ limits, energy conservation, ergodicity)

0.1.3 Why This Matters

- **Chemical Accuracy**: Classical MD fails for light nuclei — PIMD predicts correct bond lengths (H-bonds, proton wires) and reaction barriers (tunneling) - **Materials Discovery**: High-pressure phase diagrams of H, D, HO require nuclear quantum effects for accurate P-T boundaries - **Drug Design**: Proton transfer in enzyme active sites, isotope effects in metabolism (KIE), binding free energies with ZPE corrections - **Fundamental Physics**: Tests of quantum mechanics at mesoscale (100-1000 atoms), connections to quantum field theory (Euclidean QFT)

0.1.4 Pure Thought Advantages

1. **Exact quantum statistics**: PIMD converges to exact Boltzmann average as P→; no uncontrolled approximations (unlike TDDFT, semiclassical methods) 2. **Certificates**: Convergence proven by monitoring P-dependence; energy estimators have rigorous error bounds (O(1/P^2)) 3. **No empirical fitting**: Ab initio potentials eliminate force field parameters; purely first-principles physics (quantum mechanics + statistical mechanics) 4. **Polynomial scaling**: Unlike exact diagonalization (exponential in N), PIMD scales as O(P·N^3) via electronic structure, enabling 100-1000 atom simulations

—

0.2 2. Mathematical Formulation

0.2.1 Path Integral Representation

The quantum partition function for N nuclei at inverse temperature = 1/(k_B T) is: \$\$Z = Tr e$^{-\beta\hat{H}} = \int d\mathbf{R}\,\langle\mathbf{R}|e^{-\beta\hat{H}}|\mathbf{R}\rangle$\$\$

Trotter Decomposition: Split imaginary time into P slices: \$\$e$^{-\beta\hat{H}} = \left(e^{-\epsilon\hat{H}}\right)^P + O(\epsilon^2) \quad \text{where} \quad \epsilon = \beta/P$\$\$

Path Integral (Feynman 1948): \$\$Z $= \lim_{P\to\infty}\left(\frac{mP}{2\pi\beta\hbar^2}\right)^{3NP/2}\int\prod_{p=1}^{P} d\mathbf{R}^{(p)}\,\exp\left(-\beta\_PS[\{\mathbf{R}^{(p)}\}]\right)$ \$\$

where the **discretized action** is: \$\$S[{R$^{(p)}$}] $= \sum_{p=1}^{P}\left[\frac{mP}{2\beta^2\hbar^2}\left|\mathbf{R}^{(p+1)} - \mathbf{R}^{(p)}\right|^2 + \frac{1}{P}V(\mathbf{R}^{(p)})\right]$ \$\$

with periodic boundary conditions \$R$^{(P+1)} = \mathbf{R}^{(1)}$\$.

0.2.2 Ring Polymer Interpretation

The path integral defines an **effective classical Hamiltonian**: \$\$H$_{\text{RP}} = \sum_{p=1}^{P}\left[\frac{|\mathbf{p}^{(p)}|^2}{2m} + V(\mathbf{R}^{(p)}) + \frac{m\omega\_P^2}{2}\left|\mathbf{R}^{(p\right.$

where: - \$R$^{(p)} \in \mathbb{R}^{3N}$\$$isthepositionofbead$\$p\$$(cyclicwithperiodP)-$\$$\omega_P = P/(\beta\hbar)$\$$isthespringfrequency$
\$$V(\mathbf{R})$\$$istheBorn-Oppenheimerpotentialenergysurface(fromelectronicstructure)$

Isomorphism Theorem: Sampling the classical canonical ensemble of \$H$_{\text{RP}}$\$$attemperatureTyieldsexact$
$dependentobservables$: \$\$$\langle\hat{A}\rangle_{\text{quantum}} = \lim_{P\to\infty}\left\langle\frac{1}{P}\sum_{p=1}^{P}A(\mathbf{R}^{(p)})\right\rangle_{\text{classical RP}}$ \$\$

0.2.3 Normal Mode Transformation

The harmonic springs couple all beads, making direct Cartesian sampling inefficient. **Normal modes** diagonalize the kinetic + spring terms: \$\$R$^{(p)} = \frac{1}{\sqrt{P}}\sum_{k=0}^{P-1}\mathbf{u}\_k\,e^{2\pi ipk/P}$\$\$

In normal mode space: - **Centroid mode** (k=0): \$u_0 = $1\frac{}{\sqrt{P}\sum_p\mathbf{R}^{(p)}\$(classical-likecenterofmass)-**Fluctuation}$

The effective Hamiltonian becomes: $$\mathrm{H}_{\mathrm{RP}} = \sum_{k=0}^{P-1} \left[\frac{|\mathbf{q}\_k|^2}{2m} + \frac{m\omega\_k^2}{2} |\mathbf{u}\_k|^2 + \frac{1}{P} V\_k \right]$$ where \$V_k\$ requires inverse Fourier transform (coupling between modes).

0.2.4 Thermodynamic Estimators

Primitive Energy Estimator: $$\mathrm{E}_{\mathrm{prim}} = \left\langle \sum_{p=1}^{P} \left[\frac{3N}{2\beta P} + \frac{1}{P} V(\mathbf{R}^{(p)}) \right] \right\rangle$$ $ConvergesasO(1/P)buthaslargevar$

Virial Energy Estimator: $$\mathrm{E}_{\mathrm{vir}} = \left\langle \frac{3N}{2\beta} - \frac{1}{2P} \sum_{p=1}^{P} \mathbf{R}^{(p)} \cdot \nabla V(\mathbf{R}^{(p)}) + \frac{m\omega\_P^2}{2P} \sum_p (\mathbf{R}^{(p+1)} - \mathbf{R}^{(p)})^2 \right\rangle$$ C

Centroid Estimator (for observables): $$\langle A \rangle = \left\langle A \left(\frac{1}{P} \sum_{p=1}^{P} \mathbf{R}^{(p)} \right) \right\rangle$$

Heat Capacity: $$\mathrm{C\_V} = \beta^2 \left(\langle E^2 \rangle - \langle E \rangle^2 \right)$$

Isotope Fractionation: $$\alpha = \frac{Z_{\mathrm{heavy}}}{Z_{\mathrm{light}}} = \exp\left(-\beta \Delta F\right)$$ $computedviathermodynamicintegration.$

0.2.5 Certificate Specification

A **valid PIMD certificate** must include:

1. **Convergence Certificate** (P-dependence): - Energy vs P: data for P = 4, 8, 16, 32, 64 - Fit to \$E(P) = $\mathrm{E}_\infty + c/P^2$\$$(extrapolatetoP ß) - Residualfromfit < 0.01 kcal/mol$

2. **Equilibration Certificate**: - Autocorrelation function C(t) for energy, centroid position - Integrated autocorrelation time _int - Production run length ¿ 100·_int

3. **Electronic Structure Certificate**: - Basis set convergence: energy vs basis (STO-3G, 6-31G, cc-pVDZ, cc-pVTZ) - SCF convergence: all geometries converged to $10^{-8} Hartree - Forcesvalidated : finite - differencevsanalyticgradients < 10^{-6} Hartree/Bohr$

4. **Exact Benchmark** (for simple systems): - Harmonic oscillator: PIMD energy vs $(\frac{1}{2} + 1/(\mathrm{e}-1)) < 0.1\% - H molecule : vibrationallevelsvsexactSchrödinger < 1\%$

5. **Isotope Effect Certificate**: - H vs D: fractionation factor vs experimental/exact ¡ 5%

Export Format: JSON with: - Trajectory snapshots (HDF5): $\mathrm{R}^(p)\_i(t) forallbeads, atoms, timesteps - Energies : E\_prim(t), E\_vir(t) withuncertainties(blockaveraging) - Certificates : convergenceplots, autocorr$

—

0.3 3. Implementation Approach

0.3.1 Phase 1: Ring Polymer Initialization (Months 1-2)

Goal: Construct P-bead ring polymer for N atoms with harmonic spring coupling.

```
import numpy as np
from dataclasses import dataclass
from typing import List, Tuple

@dataclass
class RingPolymer:
    """
    P-bead ring polymer for N atoms.

    Attributes:
        num_beads: P (Trotter discretization)
        num_atoms: N
        masses: (N,) atomic masses in amu
        positions: (P, N, 3) bead positions in Angstrom
        velocities: (P, N, 3) bead velocities in Angstrom/fs
    """
    num_beads: int
    num_atoms: int
    masses: np.ndarray  # (N,)
    positions: np.ndarray  # (P, N, 3)
    velocities: np.ndarray  # (P, N, 3)

    def __post_init__(self):
        assert self.positions.shape == (self.num_beads, self.num_atoms,
            3)
        assert self.velocities.shape == (self.num_beads, self.num_atoms,
             3)
        assert self.masses.shape == (self.num_atoms,)


def initialize_ring_polymer_harmonic(num_beads: int, num_atoms: int,
                                     masses: np.ndarray, T: float,
                                     equilibrium_positions: np.ndarray)
                                        -> RingPolymer:
    """
    Initialize ring polymer for harmonic potential.

    Sample from Gaussian distribution:
    P(R^(p))     exp(- _P  S[R])

    For harmonic springs, this is a multivariate Gaussian.
    """
    P = num_beads
    N = num_atoms
    beta = 1.0 / (k_B * T)  # k_B in kcal/(mol K)
    omega_P = P / (beta * hbar)

    # Initialize positions: start with classical equilibrium + small
       perturbation
    positions = np.zeros((P, N, 3))

    for p in range(P):
        # Each bead starts near equilibrium with thermal fluctuations
        sigma = np.sqrt(k_B * T / (masses[:, None] * omega_P**2))
        positions[p] = equilibrium_positions + np.random.normal(0, sigma
            , (N, 3))

    # Initialize velocities from Maxwell-Boltzmann
    velocities = np.zeros((P, N, 3))
    for p in range(P):
        sigma_v = np.sqrt(k_B * T / masses[:, None])
        velocities[p] = np.random.normal(0, sigma_v, (N, 3))

    return RingPolymer(P, N, masses, positions, velocities)


def spring_potential(polymer: RingPolymer, T: float) -> float:
    """
    Harmonic spring potential coupling adjacent beads.

    V_spring =  _p  (m    _P   / 2) |R^(p+1) - R^(p)|
    """
    P = polymer.num_beads
    beta = 1.0 / (k_B * T)
    omega_P = P / (beta * hbar)

    V_spring = 0.0
    for p in range(P):
        p_next = (p + 1) % P  # Periodic boundary
        dr = polymer.positions[p_next] - polymer.positions[p]

        # Sum over atoms
        for i in range(polymer.num_atoms):
            V_spring += 0.5 * polymer.masses[i] * omega_P**2 * np.sum(dr
               [i]**2)

    return V_spring


def spring_forces(polymer: RingPolymer, T: float) -> np.ndarray:
    """
    Forces from harmonic springs: F^(p) = - _ {R^(p)} V_spring.

    F^(p) = m   _P   [R^(p-1) + R^(p+1) - 2 R^(p)]
    """
    P = polymer.num_beads
    beta = 1.0 / (k_B * T)
    omega_P = P / (beta * hbar)

    forces = np.zeros_like(polymer.positions)

    for p in range(P):
        p_prev = (p - 1) % P
        p_next = (p + 1) % P

        # Spring force (restoring force)
        dr_prev = polymer.positions[p_prev] - polymer.positions[p]
        dr_next = polymer.positions[p_next] - polymer.positions[p]

        for i in range(polymer.num_atoms):
            forces[p, i] = polymer.masses[i] * omega_P**2 * (dr_prev[i]
               + dr_next[i])

    return forces


# Test: harmonic oscillator
def test_harmonic_oscillator_1D():
    """
    Verify PIMD for 1D quantum harmonic oscillator.

    Exact: E =      (   + 1/(exp(       )-1))
    """
    # Parameters
    omega = 1.0  # frequency in atomic units
    m = 1.0
    T = 1.0
    beta = 1.0 / T

    # Exact quantum result
    E_exact = hbar * omega * (0.5 + 1.0 / (np.exp(beta * hbar * omega) -
        1.0))

    # PIMD with P beads
    for P in [4, 8, 16, 32, 64]:
        polymer = RingPolymer(
            num_beads=P,
            num_atoms=1,
            masses=np.array([m]),
            positions=np.random.normal(0, 1, (P, 1, 1)),
            velocities=np.random.normal(0, 1, (P, 1, 1))
        )

        # Run PIMD sampling (simplified)
        E_pimd = run_pimd_harmonic(polymer, T, omega, steps=10000)

        error = abs(E_pimd - E_exact) / E_exact * 100
        print(f"P={P:2d}: E_PIMD={E_pimd:.6f}, E_exact={E_exact:.6f},
           error={error:.2f}%")
```

Output: Convergence plot showing E(P) → E_exact as P increases.

—

0.3.2 Phase 2: Electronic Structure Interface (Months 2-3)

Goal: Compute Born-Oppenheimer potential V(R) and forces for each bead using quantum chemistry.

```
from pyscf import gto, scf, grad
import mpmath as mp

def compute_electronic_energy_pyscf(positions: np.ndarray,
                                    atoms: List[str],
                                    basis: str = '6-31g') -> float:
    """
    Compute electronic energy V(R) using PySCF (Hartree-Fock or DFT).

    Args:
        positions: (N, 3) atomic positions in Angstrom
        atoms: ['H', 'H'] list of elements
        basis: Gaussian basis set

    Returns:
        E_elec in Hartree
    """
    # Build molecule
    atom_spec = [(atoms[i], positions[i]) for i in range(len(atoms))]
    mol = gto.M(atom=atom_spec, basis=basis, unit='Angstrom')

    # Run restricted Hartree-Fock
    mf = scf.RHF(mol)
    mf.verbose = 0
    mf.conv_tol = 1e-10
    E_elec = mf.kernel()

    return E_elec


def compute_forces_pyscf(positions: np.ndarray,
                         atoms: List[str],
                         basis: str = '6-31g') -> np.ndarray:
    """
    Compute forces F = - V  using analytic gradients.
    Returns:
        forces: (N, 3) in Hartree/Bohr
    """
    atom_spec = [(atoms[i], positions[i]) for i in range(len(atoms))]
    mol = gto.M(atom=atom_spec, basis=basis, unit='Angstrom')

    mf = scf.RHF(mol).run(verbose=0, conv_tol=1e-10)

    # Analytic gradient
    g = grad.RHF(mf)
    forces_raw = g.kernel()  # (N, 3) in Hartree/Bohr

    # Negative gradient is force
    forces = -forces_raw

    return forces


def compute_polymer_energies_and_forces(polymer: RingPolymer,
                                        atoms: List[str],
                                        basis: str = '6-31g') -> Tuple[
                                            np.ndarray, np.ndarray]:
    """
    Compute V and F for all P beads.

    Returns:
        energies: (P,) array of electronic energies
        forces: (P, N, 3) array of forces
    """
    P = polymer.num_beads
    N = polymer.num_atoms

    energies = np.zeros(P)
    forces = np.zeros((P, N, 3))

    for p in range(P):
        energies[p] = compute_electronic_energy_pyscf(polymer.positions[
            p], atoms, basis)
        forces[p] = compute_forces_pyscf(polymer.positions[p], atoms,
            basis)

    return energies, forces


# Validation: compare analytic forces to finite differences
def validate_forces_finite_difference(positions: np.ndarray, atoms: List
    [str]):
    """
    Check that analytic gradients match finite differences.
    """
    delta = 1e-5  # Angstrom

    # Analytic forces
    F_analytic = compute_forces_pyscf(positions, atoms)

    # Finite difference forces
    N = len(atoms)
    F_fd = np.zeros((N, 3))
    for i in range(N):
        for j in range(3):  # x, y, z
            pos_plus = positions.copy()
            pos_minus = positions.copy()

            pos_plus[i, j] += delta
            pos_minus[i, j] -= delta

            E_plus = compute_electronic_energy_pyscf(pos_plus, atoms)
            E_minus = compute_electronic_energy_pyscf(pos_minus, atoms)

            F_fd[i, j] = -(E_plus - E_minus) / (2 * delta)

    # Compare
    max_error = np.max(np.abs(F_analytic - F_fd))
    print(f"Max force error (analytic vs FD): {max_error:.2e} Hartree/
        Angstrom")

    assert max_error < 1e-6, "Forces do not match finite differences!"
```

Test Case: H molecule — verify forces vanish at equilibrium bond length.

—

0.3.3 Phase 3: Normal Mode Transformation (Months 3-4)

Goal: Transform to normal modes for efficient sampling (diagonalizes kinetic + spring terms).

```
def positions_to_normal_modes(positions: np.ndarray) -> np.ndarray:
    """
    Forward Fourier transform: R^(p)     u_k.

    u_k = (1/ P )  _p  R^(p) exp(-2 i  pk/P)

    Args:
        positions: (P, N, 3)

    Returns:
        normal_modes: (P, N, 3) complex array
    """
    P = positions.shape[0]

    # Use FFT along bead axis
    normal_modes = np.fft.fft(positions, axis=0) / np.sqrt(P)

    return normal_modes


def normal_modes_to_positions(normal_modes: np.ndarray) -> np.ndarray:
    """
    Inverse Fourier transform: u_k     R^(p).

    R^(p) = (1/ P )  _k  u_k exp(+2 i  pk/P)
    """
    P = normal_modes.shape[0]

    positions = np.fft.ifft(normal_modes, axis=0) * np.sqrt(P)
    # Should be real (imaginary part is numerical noise)
    return np.real(positions)


def normal_mode_frequencies(num_beads: int, T: float) -> np.ndarray:
    """
    Frequencies of normal modes.

     _k  = 2  _P  sin( k /P) for k = 0, 1, ..., P-1
     _0  = 0 (centroid mode, free translation)
    """
    P = num_beads
    beta = 1.0 / (k_B * T)
    omega_P = P / (beta * hbar)

    k_values = np.arange(P)
    omega_k = 2 * omega_P * np.abs(np.sin(np.pi * k_values / P))

    return omega_k


def kinetic_energy_normal_modes(velocities: np.ndarray, masses: np.
   ndarray) -> float:
    """
    Kinetic energy in normal mode representation.

    K =  _k  (1/2m) |q_k|
    """
    # Transform velocities to normal modes
    q_k = positions_to_normal_modes(velocities)

    # Kinetic energy (sum over modes, atoms, dimensions)
    K = 0.5 * np.sum(masses[None, :, None] * np.abs(q_k)**2)

    return K
```

—

0.3.4 Phase 4: PIMD Integrators (Months 4-6)

Goal: Implement velocity Verlet integrator with PILE thermostat for efficient sampling.

```
def pimd_step_staging(polymer: RingPolymer, T: float, dt: float,
                      atoms: List[str], forces_external: np.ndarray):
    """
    PIMD integrator using staging coordinates (Tuckerman 2010).

    Staging coordinates: s^(p) = linear combination of R^(p) that
       decouples
    the kinetic energy.

    Args:
        polymer: current ring polymer state
        T: temperature
        dt: timestep
        atoms: element symbols
        forces_external: (P, N, 3) forces from V(R)
    """
    P = polymer.num_beads
    # Step 1: Update velocities (half step) with forces
    F_spring = spring_forces(polymer, T)
    F_total = forces_external + F_spring

    polymer.velocities += 0.5 * dt * F_total / polymer.masses[None, :,
        None]

    # Step 2: Update positions (full step)
    polymer.positions += dt * polymer.velocities

    # Step 3: Recompute forces at new positions
    energies, forces_new = compute_polymer_energies_and_forces(polymer,
        atoms)
    F_spring_new = spring_forces(polymer, T)
    F_total_new = forces_new + F_spring_new

    # Step 4: Update velocities (half step)
    polymer.velocities += 0.5 * dt * F_total_new / polymer.masses[None,
        :, None]


def pimd_with_pile_thermostat(polymer: RingPolymer, T: float, dt: float,
                              atoms: List[str], gamma: float):
    """
    PIMD with Langevin (PILE) thermostat applied to normal modes.

    PILE = Path Integral Langevin Equation thermostat
    Each normal mode thermalized independently with friction   .

    Args:
        gamma: friction coefficient (1/fs)
    """
    P = polymer.num_beads

    # Transform to normal modes
    u_k = positions_to_normal_modes(polymer.positions)
    v_k = positions_to_normal_modes(polymer.velocities)

    # Get normal mode frequencies
    omega_k = normal_mode_frequencies(P, T)

    # Evolve each mode with Langevin dynamics
    for k in range(P):
        # Ornstein-Uhlenbeck process for mode k
        c1 = np.exp(-gamma * dt)
        c2 = np.sqrt((1 - c1**2) * k_B * T / polymer.masses[:, None])

        # Update velocity with friction and random force
        v_k[k] = c1 * v_k[k] + c2 * np.random.normal(size=(polymer.
            num_atoms, 3))

        # Update position
        if omega_k[k] > 0:
            # Harmonic mode
            cos_wt = np.cos(omega_k[k] * dt)
            sin_wt = np.sin(omega_k[k] * dt)
            u_k_new = cos_wt * u_k[k] + (sin_wt / omega_k[k]) * v_k[k]
            v_k_new = -omega_k[k] * sin_wt * u_k[k] + cos_wt * v_k[k]

            u_k[k] = u_k_new
            v_k[k] = v_k_new
        else:
            # Centroid mode (free particle)
            u_k[k] += dt * v_k[k]

    # Transform back to Cartesian coordinates
    polymer.positions = normal_modes_to_positions(u_k)
    polymer.velocities = normal_modes_to_positions(v_k)


def run_pimd_trajectory(polymer: RingPolymer, T: float, dt: float,
                        atoms: List[str], num_steps: int) -> Dict:
    """
    Run full PIMD trajectory with thermalization.

    Returns:
        trajectory: dict with {energies, positions, times}
    """
    energies_vir = []
    energies_prim = []
    positions_centroid = []

    for step in range(num_steps):
        # Compute electronic forces
        V_beads, F_beads = compute_polymer_energies_and_forces(polymer,
            atoms)

        # PIMD step
        pimd_step_staging(polymer, T, dt, atoms, F_beads)

        # Apply PILE thermostat every 10 steps
        if step % 10 == 0:
            pimd_with_pile_thermostat(polymer, T, dt, atoms, gamma=0.1)

        # Compute energy estimators
        E_vir = virial_energy_estimator(polymer, T, F_beads)
        E_prim = primitive_energy_estimator(polymer, T, V_beads)

        energies_vir.append(E_vir)
        energies_prim.append(E_prim)

        # Centroid position
        centroid = np.mean(polymer.positions, axis=0)
        positions_centroid.append(centroid)

        if step % 1000 == 0:
            print(f"Step {step}/{num_steps}: E_vir={E_vir:.6f} Hartree")

    return {
        'energies_virial': np.array(energies_vir),
        'energies_primitive': np.array(energies_prim),
        'positions_centroid': np.array(positions_centroid)
    }
```

—

0.3.5 Phase 5: Thermodynamic Estimators (Months 6-7)

Goal: Implement virial and primitive estimators for energy, pressure, heat capacity.

```
def primitive_energy_estimator(polymer: RingPolymer, T: float,
                               V_beads: np.ndarray) -> float:
    """
    Primitive estimator for total energy.

    E_prim = (3N / 2  ) + (1/P)  _p  V(R^(p))

    Converges as O(1/ P  ) but high variance.
    """
    P = polymer.num_beads
    N = polymer.num_atoms
    beta = 1.0 / (k_B * T)

    # Quantum kinetic energy (primitive)
    K_prim = (3 * N) / (2 * beta)

    # Average potential
    V_avg = np.mean(V_beads)

    E_prim = K_prim + V_avg

    return E_prim


def virial_energy_estimator(polymer: RingPolymer, T: float,
                            F_beads: np.ndarray) -> float:
    """
    Virial estimator for total energy (preferred, lower variance).

    E_vir = (3N / 2  ) - (1/2P)  _p  R^(p)    F^(p)

    Converges as O(1/P).
    """
    P = polymer.num_beads
    N = polymer.num_atoms
    beta = 1.0 / (k_B * T)

    # Classical kinetic energy
    K_classical = (3 * N) / (2 * beta)

    # Virial correction
    virial = 0.0
    for p in range(P):
        virial += np.sum(polymer.positions[p] * F_beads[p])
    virial /= P

    E_vir = K_classical - 0.5 * virial

    return E_vir


def heat_capacity(energies: np.ndarray, T: float) -> float:
    """
    Heat capacity from energy fluctuations.

    C_V =      (<E  > - <E>  )
    """
    beta = 1.0 / (k_B * T)

    E_mean = np.mean(energies)
    E2_mean = np.mean(energies**2)

    C_V = beta**2 * (E2_mean - E_mean**2)

    return C_V


def compute_isotope_effect(polymer_H: RingPolymer, polymer_D:
   RingPolymer,
                          T: float, atoms: List[str]) -> float:
    """
    Isotope fractionation:    = Z_D / Z_H.

    Use thermodynamic integration:  F  = F_D - F_H.
    """
    # Run PIMD for both isotopes
    traj_H = run_pimd_trajectory(polymer_H, T, dt=0.5, atoms=atoms,
       num_steps=10000)
    traj_D = run_pimd_trajectory(polymer_D, T, dt=0.5, atoms=atoms,
       num_steps=10000)

    # Free energy difference (simplified: use energy difference)
    E_H = np.mean(traj_H['energies_virial'])
    E_D = np.mean(traj_D['energies_virial'])

    Delta_F = E_D - E_H

    # Fractionation factor
    beta = 1.0 / (k_B * T)
    alpha = np.exp(-beta * Delta_F)

    return alpha
```

Benchmark: Compare H vs D zero-point energy difference to exact quantum result.
—

0.3.6 Phase 6: Certificate Generation (Months 7-10)

Goal: Generate convergence certificates and validate against exact benchmarks.

```
import json
import h5py

def generate_pimd_certificate(polymer: RingPolymer, trajectory: Dict,
                              T: float, atoms: List[str],
                              exact_energy: float = None) -> Dict:
    """
    Generate complete PIMD certificate with convergence proofs.
    """
    P = polymer.num_beads

    # 1. Energy convergence (P-dependence)
    E_mean_vir = np.mean(trajectory['energies_virial'])
    E_std_vir = np.std(trajectory['energies_virial']) / np.sqrt(len(
        trajectory['energies_virial']))

    # 2. Autocorrelation analysis
    autocorr = compute_autocorrelation(trajectory['energies_virial'])
    tau_int = integrated_autocorrelation_time(autocorr)

    # 3. Exact comparison (if available)
    exact_error = None
    if exact_energy is not None:
        exact_error = abs(E_mean_vir - exact_energy)

    # Build certificate
    certificate = {
        'system': {
            'atoms': atoms,
            'temperature_K': T,
            'num_beads': P,
            'num_atoms': len(atoms)
        },
        'energy': {
            'virial_estimator_mean': float(E_mean_vir),
            'virial_estimator_std': float(E_std_vir),
            'primitive_estimator_mean': float(np.mean(trajectory['
                energies_primitive'])),
        },
        'convergence': {
            'autocorrelation_time': float(tau_int),
            'num_samples': len(trajectory['energies_virial']),
            'effective_samples': len(trajectory['energies_virial']) /
                tau_int
        },
        'validation': {
            'exact_energy': float(exact_energy) if exact_energy else
                None,
            'exact_error': float(exact_error) if exact_error else None
        },
        'certificate_version': '1.0'
    }

    return certificate


def compute_autocorrelation(data: np.ndarray, max_lag: int = 1000) -> np
    .ndarray:
    """
    Compute autocorrelation function C(t) = <x(0) x(t)> / <x  >.
    """
    mean = np.mean(data)
    var = np.var(data)

    autocorr = np.zeros(max_lag)

    for lag in range(max_lag):
        autocorr[lag] = np.mean((data[:-lag] - mean) * (data[lag:] -
            mean)) / var if lag > 0 else 1.0
    return autocorr


def integrated_autocorrelation_time(autocorr: np.ndarray) -> float:
    """
     _int  =  _t  C(t) (sum until C(t) < 0).
    """
    tau_int = 0.5  # Initial value

    for t in range(len(autocorr)):
        if autocorr[t] > 0:
            tau_int += autocorr[t]
        else:
            break

    return tau_int


def export_pimd_certificate(certificate: Dict, trajectory: Dict,
   filename: str):
    """
    Export certificate as JSON + trajectory as HDF5.
    """
    # JSON certificate
    with open(filename + '.json', 'w') as f:
        json.dump(certificate, f, indent=2)

    # HDF5 trajectory
    with h5py.File(filename + '.h5', 'w') as f:
        f.create_dataset('energies_virial', data=trajectory['
           energies_virial'])
        f.create_dataset('energies_primitive', data=trajectory['
           energies_primitive'])
        f.create_dataset('positions_centroid', data=trajectory['
           positions_centroid'])

    print(f"Certificate exported to {filename}.json and {filename}.h5")


def verify_pimd_certificate(cert_file: str) -> bool:
    """
    Verify PIMD certificate claims.
    """
    with open(cert_file, 'r') as f:
        cert = json.load(f)

    print("=== PIMD Certificate Verification ===")

    # Check autocorrelation time
    tau_int = cert['convergence']['autocorrelation_time']
    num_samples = cert['convergence']['num_samples']

    assert num_samples > 100 * tau_int, "Insufficient sampling (need >
       100  _int  )"
    print(f"    Sampling adequate: {num_samples} > 100  {tau_int:.1f}")

    # Check exact error (if available)
    if cert['validation']['exact_error'] is not None:
        error_pct = 100 * cert['validation']['exact_error'] / abs(cert['
           validation']['exact_energy'])
        assert error_pct < 5, f"Error too large: {error_pct:.2f}%"
        print(f"    Exact error: {error_pct:.2f}% < 5%")

    print("\ n   ALL CHECKS PASSED")
    return True
```

—

0.4 4. Example Starting Prompt

```
You are a computational chemist implementing Ab Initio Path Integral
   Molecular Dynamics (PIMD).
Your task is to treat nuclear quantum effects EXACTLY using Feynman path
    integrals combined with
quantum chemistry for the potential energy surface     NO empirical
   force fields, NO Born-Oppenheimer dynamics.

OBJECTIVE: Compute quantum thermodynamic properties of  H   molecule at
   300K using PIMD with HF/6-31G
electronic structure, achieving < 1% error vs exact Schr dinger
   solution.

PHASE 1 (Months 1-2): Ring Polymer Initialization
- Implement RingPolymer class with P=16 beads for 2 H atoms
- Initialize positions from harmonic ground state (Gaussian sampling)
- Compute harmonic spring potential V_spring =  _p  (m   _P   / 2) |R^(p
   +1) - R^(p)|
- Verify: P=1 (classical limit) reproduces classical harmonic oscillator
- Test: 1D quantum harmonic oscillator with P=4,8,16,32,64
   convergence to exact E =      (   + n  )

PHASE 2 (Months 2-3): Electronic Structure Interface
- Interface with PySCF: compute V(R) using RHF/6-31G for each bead
   configuration
- Implement analytic force computation via PySCF gradient module
- Validate forces: compare analytic   V   to finite differences (error <
   10^{-6} Hartree/Bohr)
- Benchmark:  H   potential energy curve V(R) vs bond length, compare to
    literature

PHASE 3 (Months 3-4): Normal Mode Transformation
- Implement FFT-based transform: R^(p)     u_k (normal modes)
- Compute mode frequencies  _k  = 2 _P  sin( k /P)
- Verify: centroid mode (k=0) has  _0  = 0, highest mode (k=P/2) has  _
   {P/2} = 2 _P
- Test: transform round-trip error < 10^{-12}

PHASE 4 (Months 4-6): PIMD Integrators
- Implement velocity Verlet integrator in staging coordinates
- Add PILE thermostat: Langevin dynamics on each normal mode with
   friction
- Run equilibration: 10^4 steps, monitor energy convergence
- Production run: 10^5 steps, save trajectory every 10 steps

PHASE 5 (Months 6-7): Thermodynamic Estimators
```

```
- Implement virial estimator: E_vir = 3N/(2  ) - (1/2P)  _p  R^(p) F ^(p
   )
- Implement primitive estimator: E_prim = 3N/(2  ) + (1/P)  _p  V(R^(p))
- Compute heat capacity: C_V =     (<E  > - <E>  )
- Autocorrelation analysis: compute  _int , block averaging for error
   bars
- Compare E_vir vs E_prim: virial should have lower variance

PHASE 6 (Months 7-8): Convergence and Benchmarking
- P-convergence study: run PIMD for P = 4, 8, 16, 32, 64
- Fit E(P) =  E _  + c/P  and extrapolate to  P
- Compare to exact quantum result from Schr dinger equation (DVR or
   matrix diagonalization)
- Target: |E_PIMD - E_exact| < 0.001 Hartree (< 1% of ZPE)

PHASE 7 (Months 8-10): Isotope Effects and Advanced Systems
- Compute  H   vs  D   zero-point energy difference (expect  E     0.01
    Hartree)
- Validate:  E_ZPE  = ( 2  - 1)        /2 for harmonic (   scales as
   1/  m  )
- Apply to water dimer: compute quantum H-bond strength (ZPE weakens by
   ~1 kcal/mol)
- Apply to malonaldehyde: proton transfer barrier with tunneling
   corrections

SUCCESS CRITERIA:
- **MVR (Months 3-4)**:  H   quantum kinetic energy within 5% of exact,
   P-convergence demonstrated
- **Strong (Months 6-7)**: Virial estimator implemented, H/D isotope
   effect accurate to 10%,
  autocorrelation analysis with error bars
- **Publication (Months 9-10)**: Water dimer with quantum H-bonds (
   compare to experiment/CCSD(T)),
  proton transfer rates in malonaldehyde, comprehensive database with 5+
      molecules

VERIFICATION PROTOCOL:
1. Harmonic oscillator: PIMD vs exact E =      (   + 1/(e^{      }-1))
   for T=100,300,500K
2.  H   molecule: PIMD energy vs exact Schr dinger (DVR with 100 grid
   points) < 1%
3. Forces: analytic gradients vs finite differences < 10^{-6} Hartree/
   Bohr
4. Classical limit: P=1 matches classical MD energy within 0.1%
5. Isotope effect:  E ( D  ) -  E ( H  ) matches analytical  2  scaling
    within 5%

EXPORT:
- `h2_pimd_certificate.json`: Convergence data, energies,
   autocorrelation times
- `h2_trajectory.h5`: Full trajectory (P beads    N atoms    3 coords
    timesteps)
- `pimd_module.py`: Reusable code for ring polymers, integrators,
   estimators

This is a PURE THOUGHT challenge: use ONLY path integrals + ab initio
   quantum chemistry.
```

```
NO empirical potentials, NO wavefunction propagation, NO semiclassical
    approximations.
```

—

0.5 5. Success Criteria

0.5.1 Minimum Viable Result (MVR) — Months 3-4

Deliverable: Working PIMD code for simple systems with P-convergence.

Specific Metrics: 1. **Harmonic Oscillator**: - Quantum energy E_PIMD within 5% of exact for P=16 beads - P-convergence demonstrated: E(P=4,8,16,32,64) extrapolates to E_exact

2. **H Molecule**: - Quantum kinetic energy computed using virial estimator - Comparison to exact Schrödinger solution ¡ 10% error

3. **Code Validation**: - Force validation: analytic vs finite difference ¡ $10^{-6} - Classical limit : P = 1 matches classical MD within 1\%$

Certificate: JSON with P-convergence plot, energy uncertainties (block averaging).

—

0.5.2 Strong Result — Months 6-7

Deliverable: Production-quality PIMD with isotope effects and low-variance estimators.

Specific Metrics: 1. **Accurate Energies**: - H quantum energy within 1% of exact (after P→ extrapolation) - Virial estimator variance ¡ 50% of primitive estimator

2. **Isotope Effects**: - H vs D zero-point energy difference accurate to 5% - Validation: E scales as 1/m for harmonic systems

3. **Sampling Quality**: - Autocorrelation time _int ¡ 100 steps - Effective sample size ¿ 1000 (total steps / _int) - Heat capacity C_V with error bars ¡ 10%

4. **Realistic Systems**: - Water dimer with quantum H-bonds (expect ZPE -1 kcal/mol correction)

Certificate: Autocorrelation functions, block averaging error analysis, isotope fractionation factors.

—

0.5.3 Publication-Quality Result — Months 9-10

Deliverable: Novel scientific results, comprehensive benchmarks, formal validation.

Specific Metrics: 1. **Novel Contribution**: - Proton transfer rates in malonaldehyde with tunneling (compare to experiment) - Quantum effects in hydrogen bonding: systematic study of 5+ H-bonded systems - Isotope effects in enzymatic reactions (KIE from PIMD)

2. **Database**: - 5+ molecules with complete PIMD analysis (H, D, HO, (HO), NH, CH, malonaldehyde) - Each system: energies, heat capacities, structural properties (RDFs), isotope effects

3. **Validation**: - Comparison to experiment: vibrational frequencies, ZPE corrections - Comparison to high-level theory: CCSD(T) benchmarks - Formal convergence proofs: error scaling as $O(1/P^2)$ or O(1/P)

4. **Publication Targets**: - *Journal of Chemical Physics* (PIMD methodology) - *Journal of Physical Chemistry A* (isotope effects) - *Physical Review B* (condensed matter applications: solid H phase diagram)

—

0.6 6. Verification Protocol

0.6.1 Automated Checks (Run After Each Phase)

```
def verify_pimd_implementation():
    """
    Comprehensive verification suite for PIMD.
    """
    print("=== PIMD Verification Suite ===\n")

    # 1. Harmonic oscillator (exact analytical solution)
    print("1. Testing Harmonic Oscillator")
    test_harmonic_oscillator_convergence()

    # 2. Classical limit (P=1)
    print("\n2. Testing Classical Limit (P=1)")
    test_classical_limit()

    # 3. Force validation
    print("\n3. Validating Forces (analytic vs FD)")
    test_force_validation()

    # 4. Normal mode transformation
    print("\n4. Testing Normal Mode Transform")
    test_normal_mode_roundtrip()

    # 5. Energy conservation (NVE ensemble)
    print("\n5. Testing Energy Conservation")
    test_energy_conservation()

    # 6. Isotope effect scaling
    print("\n6. Testing Isotope Effect Scaling")
    test_isotope_scaling()

    print("\n=== ALL TESTS PASSED ===")


def test_harmonic_oscillator_convergence():
    """Test P-convergence for 1D quantum harmonic oscillator."""
    omega = 1.0
    T = 1.0
    beta = 1.0 / T

    E_exact = hbar * omega * (0.5 + 1.0 / (np.exp(beta * hbar * omega) -
        1.0))

    P_values = [4, 8, 16, 32, 64]
    errors = []

    for P in P_values:
        E_pimd = run_pimd_1d_harmonic(P, omega, T, steps=10000)
        error = abs(E_pimd - E_exact) / E_exact
        errors.append(error)
        print(f"  P={P:2d}: error = {error*100:.2f}%")

    # Check O(1/P  ) convergence
    assert errors[-1] < 0.01, "P=64 error should be < 1%"
    print("      Convergence to exact solution")
```

```
def test_classical_limit():
    """Verify P=1 reproduces classical result."""
    # Classical harmonic oscillator: E = k_B T
    T = 300.0
    E_classical = k_B * T

    E_pimd_p1 = run_pimd_1d_harmonic(P=1, omega=1.0, T=T, steps=5000)

    error = abs(E_pimd_p1 - E_classical) / E_classical
    assert error < 0.02, "P=1 should match classical within 2%"
    print(f"      Classical limit: error = {error*100:.2f}%")


def test_force_validation():
    """Compare analytic forces to finite differences."""
    positions = np.array([[0.0, 0.0, 0.0], [0.74, 0.0, 0.0]])  #  H
    atoms = ['H', 'H']

    F_analytic = compute_forces_pyscf(positions, atoms)

    # Finite difference
    delta = 1e-5
    F_fd = finite_difference_forces(positions, atoms, delta)

    max_error = np.max(np.abs(F_analytic - F_fd))
    assert max_error < 1e-5, f"Force error {max_error} too large"
    print(f"      Force accuracy: {max_error:.2e} Hartree/Bohr")


def test_normal_mode_roundtrip():
    """Test FFT roundtrip error."""
    P, N = 16, 2
    positions = np.random.normal(0, 1, (P, N, 3))

    # Forward + inverse transform
    u_k = positions_to_normal_modes(positions)
    positions_reconstructed = normal_modes_to_positions(u_k)

    error = np.max(np.abs(positions - positions_reconstructed))
    assert error < 1e-12, "FFT roundtrip error too large"
    print(f"      FFT roundtrip error: {error:.2e}")


def test_isotope_scaling():
    """Check isotope effect scales as 1/ m  for harmonic."""
    # For harmonic:  E_ZPE  =      ( 2  - 1) / 2 for  H D
    omega = 1.0
    m_H = 1.0
    m_D = 2.0
    T = 300.0

    E_H = run_pimd_1d_harmonic(P=32, omega=omega, T=T, mass=m_H, steps
       =10000)
    E_D = run_pimd_1d_harmonic(P=32, omega=omega/np.sqrt(2), T=T, mass=
       m_D, steps=10000)
    Delta_E_pimd = E_D - E_H
    Delta_E_exact = hbar * omega * (1/np.sqrt(2) - 1) * 0.5

    error = abs(Delta_E_pimd - Delta_E_exact) / abs(Delta_E_exact)
    assert error < 0.1, "Isotope effect scaling incorrect"
    print(f"      Isotope scaling: error = {error*100:.2f}%")
```

Manual Checks: 1. Visualize ring polymer configurations — beads should form closed loops 2. Plot energy autocorrelation — should decay exponentially with time constant _int 3. Compare to literature: H vibrational frequency (4401 cm^1), dissociation energy (4.75 eV)

—

0.7 7. Resources and Milestones

0.7.1 Essential References

Path Integral Formalism: - Feynman, R. P., & Hibbs, A. R. (1965). *Quantum Mechanics and Path Integrals*. McGraw-Hill. - Chandler, D., & Wolynes, P. G. (1981). "Exploiting the isomorphism between quantum theory and classical statistical mechanics of polyatomic fluids." *Journal of Chemical Physics* 74(7): 4078-4095.

PIMD Methodology: - Tuckerman, M. E. (2010). *Statistical Mechanics: Theory and Molecular Simulation*. Oxford University Press. (Chapter 12: Path Integrals) - Markland, T. E., & Ceriotti, M. (2018). "Nuclear quantum effects enter the mainstream." *Nature Reviews Chemistry* 2: 0109.

Ab Initio PIMD: - Marx, D., & Parrinello, M. (1996). "Ab initio path integral molecular dynamics: Basic ideas." *Journal of Chemical Physics* 104(11): 4077-4082. - Habershon, S., et al. (2013). "Ring-polymer molecular dynamics: Quantum effects in chemical dynamics from classical trajectories in an extended phase space." *Annual Review of Physical Chemistry* 64: 387-413.

Software: - PySCF (quantum chemistry), i-PI (PIMD driver), LAMMPS (with PIMD plugin)

0.7.2 Milestone Checklist

Month 1-2: - [ ] RingPolymer class with spring potentials implemented - [ ] 1D harmonic oscillator: P-convergence to exact within 5% - [ ] Classical limit (P=1) verified

Month 3-4: - [ ] PySCF interface: electronic energies and forces computed - [ ] Force validation: analytic vs FD ¡ $10^{-6} - []Normalmodetransformationtested(FFTroundtrip < 10^{-12})$

Month 5-6: - [ ] PIMD integrator (staging/PILE) producing stable trajectories - [ ] Equilibration protocol: energy converges within $10^4 steps-[]Virialandprimitiveestimatorsimplemented$

Month 7-8: - [ ] H benchmark: energy within 1% of exact Schrödinger - [ ] Autocorrelation analysis: _int computed, effective sample size ¿ 1000 - [ ] H/D isotope effect validated

Month 9-10: - [ ] Water dimer with quantum H-bonds - [ ] Database of 5+ molecules with full analysis - [ ] Certificate generation and validation scripts - [ ] Draft manuscript on quantum nuclear effects

0.7.3 Common Pitfalls

1. **Insufficient P**: For light nuclei (H) at low T, need P 32 for 1% accuracy; monitor P-convergence carefully

2. **Poor Thermalization**: PILE thermostat critical for ergodic sampling; check energy distribution is Boltzmann

3. **Electronic Structure Convergence**: SCF failures at distorted geometries; use tight conv_tol $(10^{-10}) and stable guess (previous step)$

4. **Autocorrelation Neglect**: Energy autocorrelation times can be 100-1000 steps; use block averaging for error bars

5. **Normal Mode Coupling**: Centroid mode evolves slowly; decouple with mode-specific thermostats

—

End of PRD 20