PRD 18: Optimal Transport for Molecular Systems

Pure Thought AI Challenge 18

Pure Thought AI Challenges Project

January 18, 2026

Abstract

This document presents a comprehensive Product Requirement Document (PRD) for implementing a pure-thought computational challenge. The problem can be tackled using only symbolic mathematics, exact arithmetic, and fresh code—no experimental data or materials databases required until final verification. All results must be accompanied by machine-checkable certificates.

Contents

Domain: Chemistry Applied Mathematics
Timeline: 6-9 months
Difficulty: High
Prerequisites: Optimal transport theory, measure theory, convex optimization, quantum chemistry, computational geometry

---

0.1 1. Problem Statement

0.1.1 Scientific Context

Optimal transport (OT) theory, pioneered by Monge (1781) and formalized by Kantorovich (1942), provides a mathematical framework for comparing probability distributions by computing the minimal "cost" of transforming one into another. In recent decades, OT has emerged as a powerful tool in quantum chemistry, offering rigorous reformulations of fundamental concepts like exchange energy, correlation, and reaction pathways. The **Wasserstein distance** $W_p(,)quantifiesthep-thmomentcostoftransportingmeasureto, definingnaturalmetriconthespaceofprobab$

Multi-marginal optimal transport (MMOT) extends classical OT to N 2 probability measures, with applications to N-electron quantum systems. Seidl et al. (1999) showed that the **exact exchange energy** of density functional theory (DFT) can be reformulated as a multi-marginal OT problem: minimizing the Coulomb repulsion 1/|r-r'| d(r,r') over all N-particle distributions with prescribed one-electron density . This **Seidl functional** is exact but computationally intractable for N > 2, making it a challenging pure-thought problem. Cotar, Friesecke, and Pass (2013) proved that for Coulomb cost, the optimal is supported on the graph of a measure-preserving map—a deep connection between OT and quantum mechanics.

Wasserstein distances between molecular electron densities provide a geometric measure of chemical similarity. Piccioni and Gori-Giorgi (2017) computed W distances for atoms and small molecules, finding that Wasserstein metrics capture chemical trends (atomic number, bonding) more faithfully than traditional overlap integrals. **Reaction pathways** can be formulated as geodesics in Wasserstein space, with McCann interpolation providing the minimum-cost continuous deformation connecting reactant and product densities. This offers an alternative to traditional intrinsic reaction coordinate (IRC) methods, grounded in rigorous mathematics rather than ad hoc potential energy surface navigation.

0.1.2 Core Question

Can we reformulate electronic structure problems using optimal transport theory, providing exact functionals, computable reaction coordinates, and machine-checkable certificates—without empirical approximations?

Specifically:

- Compute Wasserstein-1 and Wasserstein-2 distances between molecular electron densities (1D and 3D)
- Implement Sinkhorn algorithm for entropic regularized OT (computationally tractable approximation)
- Solve multi-marginal OT for 2-electron systems (exact Seidl exchange)
- Construct OT geodesics (McCann interpolation) as reaction pathways
- Generate certificates: OT dual potentials, transport maps, optimality conditions
- Validate against exact quantum chemistry (Hartree-Fock exchange, coupled cluster)

0.1.3 Why This Matters

- **Exact Functionals**: Seidl OT provides the **exact** exchange energy, bypassing GGA/hybrid approximations that dominate DFT
- **Chemical Similarity**: Wasserstein distances offer rigorous metrics for molecular comparison, applicable to drug discovery and materials design
- **Reaction Coordinates**: OT geodesics reveal minimum-energy pathways without requiring pre-specified collective variables
- **Mathematical Rigor**: OT is pure measure theory + convex optimization; all results have proofs and duality certificates
- **Computational Challenge**: Multi-marginal OT is NP-hard, making efficient algorithms a frontier research area

0.1.4 Pure Thought Advantages

- **Certificate-Based**: All OT solutions come with Kantorovich dual potentials , certifying optimality
- **Exact for N=2**: Two-electron Seidl exchange is computable via standard OT solvers (linear programming)
- **No Experimental Data**: Electron densities computed from wavefunctions (Schrödinger equation); no fitting parameters
- **Convex Optimization**: Kantorovich formulation is convex; global optimum guaranteed (though high-dimensional)
- **Geometric Interpretation**: Wasserstein space is a Riemannian manifold; geodesics have clear physical meaning

---

0.2 2. Mathematical Formulation

0.2.1 Optimal Transport Problem

Monge problem (1781): Given measures , on $^{d}andcostc(x,y), findtransportmapTminimizing$:
inf*T: T =* c(x, T(x)) d(x)
where T$_{=means(B)=(T\acute{z}(B))forallmeasurableB(push-forwardcondition)}$.
Kantorovich relaxation (1942): Instead of deterministic map T, optimize over couplings (,):
W*c(,) = inf* **(,) c(x,y) d(x,y)**
where (,) = joint distributions with marginals , . This is a **linear program** in infinite dimensions.
Kantorovich duality: For c(x,y) = h(x-y) convex, the dual is:
W*c(,) = sup*,: (x)+(y) c(x,y) d + d
The supremum is attained, and (, ,) *satisfy* ***complementary slackness:*** supported on (x,y): *(x)+*(y) = c(x,y).

0.2.2 Wasserstein Distances

Wasserstein-p distance for c(x,y) = |x-y|^p :
W*p(,)* = *(inf* (,) |x-y|$^p d(x,y))^{1/p}$
W is a metric on P(^{d})($probabilitymeasureswithfinitesecondmoment$), $inducingthe$Wasserstein space(
1D case (d=1): Wasserstein distances have closed form via quantile functions.
If F, G are CDFs:
W$_p(,)^p = \acute{z}|F\acute{z}(u) - G\acute{z}(u)|^p du$
Optimal map: T(x) = G^{1}(F(x)) (push-forward of F along G).

0.2.3 Multi-Marginal Optimal Transport

For N measures , ..., $_N on^d, MMOT minimizes$:
W*c(,...,*N) = inf *(,...,N)* c(x,...,x*N) d(x,...,x*N)
where (,...,$_N) = N - point distributions with marginals.$**
Complexity: For discrete with M atoms each, MMOT is LP with M$^N variables^{-} exponential in N.$
Coulomb cost: c(r,...,r*N) = i<j 1/|r*i - r*j| (electron-electron repulsion).

0.2.4 Seidl Exchange Functional

For N-electron system with density (r) = N |(r,r,...,r*N)/*^{2} *dr...drN:*
E*x[] = -½ inf* $1/|r - r'|d(r,r')$
where $_{=2-marginals of N-particle distributions with 1-density, antisymmetric.}$
For N=2: This reduces to standard 2-marginal OT with Coulomb cost. The optimal is unique (Brenier theorem for smooth).
Duality: The dual involves electrostatic potentials , satisfying (r) + (r') -1/|r-r'|.

0.2.5 Certificate Specification

An optimal transport certificate must contain:

- **Primal solution: Coupling *(or transport map T)*, cost c d***
- **Dual solution: Potentials , with (x)+(y) c(x,y) and d + d**
- **Complementary slackness: supp*()* *(x,y): (x)+*(y) = c(x,y)***
- **Duality gap: |Primal - Dual| < (= 10 for numerical solutions)**
- **Marginal constraints: * has marginals , within tolerance = 10**

0.3 3. Implementation Approach

This is a 6-phase project spanning 6-9 months, progressing from 1D OT to multi-marginal chemistry applications.

0.3.1 Phase 1: 1D Wasserstein Distances (Months 1-2)

Objective: Implement exact W and W for 1D distributions, validate against scipy.

```python
import numpy as np
from scipy import integrate
from typing import Tuple, Dict
import matplotlib.pyplot as plt

def cdf_and_quantile(density: np.ndarray, grid: np.ndarray) ->
    Tuple[np.ndarray, callable]:
    """
    Compute CDF F(x) and quantile function  F  (u) from density.

    Args:
        density: Probability density  (x) on grid
        grid: 1D spatial grid

    Returns: (cdf_values, quantile_function)
    """
    # Normalize density
    dx = grid[1] - grid[0]
    rho = density / (np.sum(density) * dx)

    # CDF via cumulative sum
    cdf = np.cumsum(rho) * dx
    cdf = np.clip(cdf, 0, 1)  # Ensure [0,1]

    # Quantile function (inverse CDF)
    def quantile(u):
        """ F  (u): return x such that F(x) = u."""
        return np.interp(u, cdf, grid)

    return cdf, quantile

def wasserstein_1d(rho1: np.ndarray, rho2: np.ndarray, grid:
    np.ndarray, p: int = 2) -> float:
    """
    Exact Wasserstein-p distance for 1D densities.

    W_p(    ,    )^p =        |  F    (u) -  F    (u)|^p du

    Args:
        rho1, rho2: Probability densities on grid
        grid: 1D spatial grid
        p: Wasserstein exponent (1 or 2)

    Returns: W_p distance
    """
    # Compute quantile functions
    cdf1, q1 = cdf_and_quantile(rho1, grid)
    cdf2, q2 = cdf_and_quantile(rho2, grid)

    # Integrate |q1(u) - q2(u)|^p over u    [0,1]
    u_vals = np.linspace(0, 1, 1000)
    q1_vals = np.array([q1(u) for u in u_vals])
    q2_vals = np.array([q2(u) for u in u_vals])

    integrand = np.abs(q1_vals - q2_vals)**p
```

```
    Wp_p = np.trapz(integrand, u_vals)

    return Wp_p**(1/p)

def optimal_transport_map_1d(rho1: np.ndarray, rho2: np.ndarray, grid:
    np.ndarray) -> callable:
    """
    Compute optimal transport map T:           in 1D.

    T(x) =  F _     (F_  (x)) (monotone rearrangement)

    Returns: Transport map function T(x).
    """
    cdf1, q1 = cdf_and_quantile(rho1, grid)
    cdf2, q2 = cdf_and_quantile(rho2, grid)

    def T(x):
        """Optimal map T(x) = q2(cdf1(x))."""
        # Find u =  F  (x)
        u = np.interp(x, grid, cdf1)
        # Return    F     (u)
        return q2(u)

    return T

# Example: Wasserstein distance between Gaussians
if __name__ == "__main__":
    # Grid
    grid = np.linspace(-5, 5, 1000)
    dx = grid[1] - grid[0]

    # Two Gaussian densities
    mu1, sigma1 = 0.0, 1.0
    mu2, sigma2 = 2.0, 0.5

    rho1 = np.exp(-0.5*((grid - mu1)/sigma1)**2) / (sigma1 *
        np.sqrt(2*np.pi))
    rho2 = np.exp(-0.5*((grid - mu2)/sigma2)**2) / (sigma2 *
        np.sqrt(2*np.pi))

    # Wasserstein distances
    W1 = wasserstein_1d(rho1, rho2, grid, p=1)
    W2 = wasserstein_1d(rho1, rho2, grid, p=2)

    print(f"1D Wasserstein Distances:")
    print(f"   W   = {W1:.6f}")
    print(f"   W   = {W2:.6f}")

    # Analytical  W   for Gaussians: sqrt((     -     )   +
        (     -     )  )
    W2_exact = np.sqrt((mu1 - mu2)**2 + (sigma1 - sigma2)**2)
    print(f"   W   (exact) = {W2_exact:.6f}")
    print(f"  Error: {abs(W2 - W2_exact):.2e}")

    # Optimal map
    T_opt = optimal_transport_map_1d(rho1, rho2, grid)
```

```
    x_test = np.array([-1, 0, 1, 2])
    print(f"\nOptimal transport map T(x):")
    for x in x_test:
        print(f"  T({x:.1f}) = {T_opt(x):.3f}")
```

0.3.2 Phase 2: Sinkhorn Algorithm for Entropic OT (Months 2-3)

Objective: Implement Sinkhorn iterations for fast approximate OT in higher dimensions.

```
def sinkhorn_distance(mu: np.ndarray, nu: np.ndarray, C: np.ndarray,
                      epsilon: float = 0.1, max_iter: int = 1000) ->
                         Dict:
    """
    Compute entropic regularized OT via Sinkhorn algorithm.

    Minimizes:        c(x,y) d   +   KL  (   |        )

    Args:
        mu, nu: Discrete probability distributions (1D arrays summing
           to 1)
        C: Cost matrix C[i,j] = c(x_i, y_j)
        epsilon: Regularization parameter (smaller = closer to true OT)
        max_iter: Maximum iterations

    Returns: Dictionary with cost, coupling, dual variables.
    """
    M, N = C.shape
    assert len(mu) == M and len(nu) == N

    # Kernel K = exp(-C/  )
    K = np.exp(-C / epsilon)

    # Initialize dual variables (log-domain for stability)
    u = np.ones(M)
    v = np.ones(N)

    for iteration in range(max_iter):
        u_prev = u.copy()

        # Sinkhorn iterations
        u = mu / (K @ v)
        v = nu / (K.T @ u)

        # Check convergence
        if np.max(np.abs(u - u_prev)) < 1e-9:
            break

    # Optimal coupling     = diag(u) K diag(v)
    pi = u[:, None] * K * v[None, :]

    # Cost
    cost = np.sum(pi * C)

    return {
```

```
        'cost': cost,
        'coupling': pi,
        'dual_u': u,
        'dual_v': v,
        'iterations': iteration + 1
    }

def pairwise_distances(X: np.ndarray, Y: np.ndarray = None, metric: str = 'euclidean') -> np.ndarray:
    """
    Compute pairwise distances between points.

    Args:
        X: Array of shape (M, d)
        Y: Array of shape (N, d) (if None, use Y=X)
        metric: 'euclidean' or 'sqeuclidean'

    Returns: Cost matrix C of shape (M, N)
    """
    if Y is None:
        Y = X

    M, N = X.shape[0], Y.shape[0]
    C = np.zeros((M, N))

    for i in range(M):
        for j in range(N):
            diff = X[i] - Y[j]
            if metric == 'euclidean':
                C[i, j] = np.sqrt(np.sum(diff**2))
            elif metric == 'sqeuclidean':
                C[i, j] = np.sum(diff**2)

    return C

# Example: 2D point clouds
if __name__ == "__main__":
    # Two point clouds in 2D
    np.random.seed(42)

    # Cloud 1: Gaussian around (0,0)
    n1 = 50
    X1 = np.random.randn(n1, 2) * 0.5

    # Cloud 2: Gaussian around (2,1)
    n2 = 50
    X2 = np.random.randn(n2, 2) * 0.5 + np.array([2.0, 1.0])

    # Uniform weights
    mu = np.ones(n1) / n1
    nu = np.ones(n2) / n2

    # Cost matrix (squared Euclidean distance)
    C = pairwise_distances(X1, X2, metric='sqeuclidean')

    # Sinkhorn OT
```

```
    result = sinkhorn_distance(mu, nu, C, epsilon=0.05)

    print(f"Sinkhorn OT:")
    print(f"  Cost (  W   ): {result['cost']:.6f}")
    print(f"   W  : {np.sqrt(result['cost']):.6f}")
    print(f"  Iterations: {result['iterations']}")

    # Verify marginals
    pi = result['coupling']
    marginal_mu = np.sum(pi, axis=1)
    marginal_nu = np.sum(pi, axis=0)
    print(f"  Marginal error (  ): {np.max(np.abs(marginal_mu -
        mu)):.2e}")
    print(f"  Marginal error (  ): {np.max(np.abs(marginal_nu -
        nu)):.2e}")
```

0.3.3 Phase 3: Molecular Electron Densities (Months 3-4)

Objective: Compute electron densities from molecular orbitals, calculate Wasserstein distances.

```
from scipy.special import sph_harm

def hydrogen_1s_density(r: np.ndarray, R: np.ndarray =
    np.array([0,0,0])) -> np.ndarray:
    """
    Electron density for hydrogen 1s orbital:   (r) = |  (r)|   =
        (1/  ) exp(-2|r-R|).

    Args:
        r: Grid points (Nx3 array)
        R: Nuclear position

    Returns: Density at each grid point
    """
    r_shifted = r - R[None, :]
    r_norm = np.sqrt(np.sum(r_shifted**2, axis=1))
    rho = (1 / np.pi) * np.exp(-2 * r_norm)
    return rho

def molecular_density_H2(r: np.ndarray, R1: np.ndarray, R2: np.ndarray)
    -> np.ndarray:
    """
    Approximate  H   density as sum of two H 1s orbitals (LCAO
        approximation).

      (r)     |     (r) +      (r)|

    Args:
        r: Grid points
        R1, R2: Positions of two H atoms

    Returns: Electron density
    """
```

```
    psi1 = np.sqrt(hydrogen_1s_density(r, R1) * np.pi)  # Unnormalized
        wavefunction
    psi2 = np.sqrt(hydrogen_1s_density(r, R2) * np.pi)

    # Bonding orbital (normalized)
    psi_bond = (psi1 + psi2) / np.sqrt(2 * (1 +
        np.exp(-np.linalg.norm(R1 - R2))))

    rho = psi_bond**2

    return rho

def compute_wasserstein_molecular(rho1: np.ndarray, rho2: np.ndarray,
                                  grid: np.ndarray, epsilon: float =
                                      0.05) -> float:
    """
    Wasserstein-2 distance between two 3D molecular densities via
        Sinkhorn.

    Args:
        rho1, rho2: Densities on 3D grid (N   arrays)
        grid: 1D grid for each dimension (N points)
        epsilon: Entropic regularization

    Returns:  W   distance
    """
    # Flatten densities
    rho1_flat = rho1.flatten()
    rho2_flat = rho2.flatten()

    # Normalize to probability distributions
    rho1_flat /= np.sum(rho1_flat)
    rho2_flat /= np.sum(rho2_flat)

    # Grid coordinates (Cartesian product)
    N = len(grid)
    xx, yy, zz = np.meshgrid(grid, grid, grid, indexing='ij')
    coords = np.stack([xx.flatten(), yy.flatten(), zz.flatten()],
        axis=1)

    # Cost matrix (squared Euclidean)
    print(f"Computing cost matrix ({N**3}    {N**3})...")
    # For large grids, subsample or use GPU
    # Here, assume N     32 (32    = 32,768 points, manageable)

    C = pairwise_distances(coords, metric='sqeuclidean')

    # Sinkhorn
    print(f"Running Sinkhorn...")
    result = sinkhorn_distance(rho1_flat, rho2_flat, C, epsilon=epsilon)

    W2 = np.sqrt(result['cost'])

    return W2

# Example: H vs  H   densities
```

```python
if __name__ == "__main__":
    # 3D grid (coarse for speed)
    N = 16  # 16   = 4096 points
    grid_1d = np.linspace(-3, 3, N)
    dx = grid_1d[1] - grid_1d[0]

    xx, yy, zz = np.meshgrid(grid_1d, grid_1d, grid_1d, indexing='ij')
    r_grid = np.stack([xx.flatten(), yy.flatten(), zz.flatten()],
        axis=1)

    # H atom density (centered at origin)
    rho_H = hydrogen_1s_density(r_grid,
        R=np.array([0,0,0])).reshape((N, N, N))

    #  H   molecule density (bond length 1.4 Bohr)
    R1 = np.array([-0.7, 0, 0])
    R2 = np.array([0.7, 0, 0])
    rho_H2 = molecular_density_H2(r_grid, R1, R2).reshape((N, N, N))

    # Wasserstein distance
    W2 = compute_wasserstein_molecular(rho_H, rho_H2, grid_1d,
        epsilon=0.1)

    print(f"\nWasserstein-2 distance:")
    print(f"   W  (H,  H  ) = {W2:.6f} Bohr")
```

0.3.4 Phase 4: Seidl Exchange for 2-Electron Systems (Months 4-6)

Objective: Compute exact exchange energy via 2-marginal OT with Coulomb cost.

```python
def coulomb_cost_matrix(r1_grid: np.ndarray, r2_grid: np.ndarray) ->
    np.ndarray:
    """
    Coulomb cost matrix C[i,j] = 1/|r1[i] - r2[j]|.

    Args:
        r1_grid, r2_grid: Grid points (Nx3 arrays)

    Returns: Cost matrix (N    N)
    """
    N1, N2 = r1_grid.shape[0], r2_grid.shape[0]
    C = np.zeros((N1, N2))

    for i in range(N1):
        for j in range(N2):
            r_ij = np.linalg.norm(r1_grid[i] - r2_grid[j])
            C[i, j] = 1.0 / (r_ij + 1e-10)  # Regularize for r_ij     0

    return C

def seidl_exchange_2electron(rho: np.ndarray, grid: np.ndarray) ->
    float:
    """
    Compute Seidl exchange energy for 2-electron system via OT.

    E_x = -   inf_{         (   ,   )}         1/|r-r'|  d  (r,r')
```

```
25
26      Args:
27          rho: Electron density on 3D grid (N   array)
28          grid: 1D grid for each dimension
29
30      Returns: Exchange energy E_x (in Hartree)
31      """
32      # Flatten density
33      rho_flat = rho.flatten()
34      rho_flat /= np.sum(rho_flat)
35
36      # Grid coordinates
37      N = len(grid)
38      xx, yy, zz = np.meshgrid(grid, grid, grid, indexing='ij')
39      r_grid = np.stack([xx.flatten(), yy.flatten(), zz.flatten()],
            axis=1)
40
41      # Coulomb cost
42      print(f"Computing Coulomb cost matrix...")
43      C_coulomb = coulomb_cost_matrix(r_grid, r_grid)
44
45      # Solve OT (using entropic regularization for tractability)
46      epsilon = 0.01  # Small    to approximate true OT
47      result = sinkhorn_distance(rho_flat, rho_flat, C_coulomb,
            epsilon=epsilon)
48
49      # Exchange energy: E_x = -   * OT_cost
50      E_x = -0.5 * result['cost']
51
52      return E_x
53
54  # Example: He atom exchange energy
55  if __name__ == "__main__":
56      # He density (approximate as spherically symmetric)
57      N = 12  # Coarse grid for demonstration
58      grid_1d = np.linspace(-2, 2, N)
59
60      xx, yy, zz = np.meshgrid(grid_1d, grid_1d, grid_1d, indexing='ij')
61      r_grid_3d = np.stack([xx, yy, zz], axis=-1)
62      r_norm = np.sqrt(np.sum(r_grid_3d**2, axis=-1))
63
64      # He 1s   density:   (r)     2 * (Z  /  ) exp(-2Zr) with Z=2 (He
            nuclear charge)
65      Z = 2
66      rho_He = 2 * (Z**3 / np.pi) * np.exp(-2 * Z * r_norm)
67
68      # Seidl exchange
69      E_x = seidl_exchange_2electron(rho_He, grid_1d)
70
71      print(f"\nSeidl Exchange Energy:")
72      print(f"  E_x(He) = {E_x:.6f} Hartree")
73
74      # Compare to exact Hartree-Fock exchange for He: E_x     -1.026
            Hartree
75      E_x_exact = -1.026
76      print(f"  E_x(exact HF) = {E_x_exact:.6f} Hartree")
```

```
    print(f"  Error: {abs(E_x - E_x_exact):.4f} Hartree")
```

0.3.5 Phase 5: OT Geodesics as Reaction Pathways (Months 6-7)

Objective: Compute McCann interpolation connecting reactant and product densities.

```
def mccann_interpolation(rho_0: np.ndarray, rho_1: np.ndarray, grid: np.ndarray,
                         num_steps: int = 20) -> list:
    """
    Compute Wasserstein geodesic connecting                via McCann interpolation.

     _t  = [(1-t) id  + t  T]_#

    where T is optimal transport map.

    Args:
        rho_0: Initial density (N  array)
        rho_1: Final density (N  array)
        grid: 1D grid
        num_steps: Number of interpolation steps

    Returns: List of densities [ _t  for t in [0,1]]
    """
    # For simplicity, use entropic OT to get coupling
    rho0_flat = rho_0.flatten()
    rho1_flat = rho_1.flatten()

    rho0_flat /= np.sum(rho0_flat)
    rho1_flat /= np.sum(rho1_flat)

    # Grid coordinates
    N = len(grid)
    xx, yy, zz = np.meshgrid(grid, grid, grid, indexing='ij')
    r_coords = np.stack([xx.flatten(), yy.flatten(), zz.flatten()], axis=1)

    # Cost and OT
    C = pairwise_distances(r_coords, metric='sqeuclidean')
    ot_result = sinkhorn_distance(rho0_flat, rho1_flat, C, epsilon=0.05)
    pi = ot_result['coupling']

    # Approximate transport map T via barycentric projection
    # T(x)     E[y | x] =  _y  y   (x,y) /  _y    (x,y)
    T_map = np.zeros_like(r_coords)
    for i in range(len(r_coords)):
        weights = pi[i, :]
        if np.sum(weights) > 1e-10:
            T_map[i] = np.sum(r_coords * weights[:, None], axis=0) / np.sum(weights)
        else:
            T_map[i] = r_coords[i]

```

```
    # McCann interpolation
    pathway = []
    t_vals = np.linspace(0, 1, num_steps)

    for t in t_vals:
        # Interpolated map: T_t = (1-t) id  + t  T
        T_t = (1 - t) * r_coords + t * T_map

        # Push forward       via T_t (approximate by resampling)
        # For simplicity, just linearly interpolate densities (not
          exact McCann)
        rho_t_flat = (1 - t) * rho0_flat + t * rho1_flat
        rho_t = rho_t_flat.reshape((N, N, N))

        pathway.append(rho_t)

    return pathway

# Example:  H   dissociation pathway
if __name__ == "__main__":
    N = 16
    grid_1d = np.linspace(-3, 3, N)

    xx, yy, zz = np.meshgrid(grid_1d, grid_1d, grid_1d, indexing='ij')
    r_grid = np.stack([xx.flatten(), yy.flatten(), zz.flatten()],
       axis=1)

    # Initial:  H   molecule (bond length 1.4 Bohr)
    R1_init = np.array([-0.7, 0, 0])
    R2_init = np.array([0.7, 0, 0])
    rho_0 = molecular_density_H2(r_grid, R1_init, R2_init).reshape((N,
       N, N))

    # Final: Two separated H atoms (distance 6 Bohr)
    R1_final = np.array([-3, 0, 0])
    R2_final = np.array([3, 0, 0])
    rho_1_H1 = hydrogen_1s_density(r_grid, R1_final).reshape((N, N, N))
    rho_1_H2 = hydrogen_1s_density(r_grid, R2_final).reshape((N, N, N))
    rho_1 = rho_1_H1 + rho_1_H2

    # Compute pathway
    print("Computing OT geodesic pathway...")
    pathway = mccann_interpolation(rho_0, rho_1, grid_1d, num_steps=10)

    print(f"Generated {len(pathway)} intermediate densities along
       pathway")

    # Compute energies along pathway (placeholder: would require full
       quantum calc)
    print("\nPathway densities:")
    for i, rho_t in enumerate(pathway):
        total_density = np.sum(rho_t) * (grid_1d[1] - grid_1d[0])**3
        print(f"  Step {i}: Total density = {total_density:.6f}")
```

0.3.6 Phase 6: Certificate Generation and Export (Months 7-9)

Objective: Generate machine-checkable certificates for all OT computations.

```python
from dataclasses import dataclass, asdict
import json
from datetime import datetime

@dataclass
class OptimalTransportCertificate:
    """Certificate for optimal transport computation."""

    # Problem specification
    problem_type: str  # "Wasserstein", "Seidl_exchange", "Geodesic"
    dimension: int
    grid_size: int

    # Primal solution
    transport_cost: float
    marginal_error: float  # max deviation from exact marginals

    # Dual solution
    dual_gap: float  # |Primal - Dual|
    complementary_slackness_error: float

    # Molecular details (if applicable)
    molecule_1: str  # "H", "He", "H2", etc.
    molecule_2: str
    bond_length: float  # Bohr radii

    # Numerical parameters
    epsilon_regularization: float
    sinkhorn_iterations: int

    # Verification
    certificate_valid: bool
    timestamp: str
    computation_time: float

def generate_ot_certificate(ot_result: Dict, problem_info: Dict) ->
    OptimalTransportCertificate:
    """Generate certificate from OT computation result."""

    # Verify marginals
    pi = ot_result['coupling']
    mu_reconstructed = np.sum(pi, axis=1)
    nu_reconstructed = np.sum(pi, axis=0)

    mu_target = problem_info['mu']
    nu_target = problem_info['nu']

    marginal_error = max(
        np.max(np.abs(mu_reconstructed - mu_target)),
        np.max(np.abs(nu_reconstructed - nu_target))
    )

    # Dual gap (for Sinkhorn, should be ~0 at convergence)
```

```
    # Placeholder: true dual requires solving dual problem
    dual_gap = 1e-6  # Approximate

    # Complementary slackness
    # Check:   [i,j] > 0       [i] +   [j]     C[i,j]
    cs_error = 0.0  # Placeholder

    cert = OptimalTransportCertificate(
        problem_type=problem_info['type'],
        dimension=problem_info['dimension'],
        grid_size=problem_info['grid_size'],
        transport_cost=ot_result['cost'],
        marginal_error=marginal_error,
        dual_gap=dual_gap,
        complementary_slackness_error=cs_error,
        molecule_1=problem_info.get('mol1', 'N/A'),
        molecule_2=problem_info.get('mol2', 'N/A'),
        bond_length=problem_info.get('bond_length', 0.0),
        epsilon_regularization=problem_info['epsilon'],
        sinkhorn_iterations=ot_result['iterations'],
        certificate_valid=(marginal_error < 1e-4 and dual_gap < 1e-4),
        timestamp=datetime.now().isoformat(),
        computation_time=problem_info.get('time', 0.0)
    )

    return cert

def export_certificate_json(cert: OptimalTransportCertificate,
    filepath: str):
    """Export certificate to JSON."""
    with open(filepath, 'w') as f:
        json.dump(asdict(cert), f, indent=2)

    print(f"Certificate exported to {filepath}")

# Example: Full pipeline
if __name__ == "__main__":
    # Simplified example: 1D Gaussians
    grid = np.linspace(-5, 5, 100)
    mu1, sigma1 = 0.0, 1.0
    mu2, sigma2 = 2.0, 0.5

    rho1 = np.exp(-0.5*((grid - mu1)/sigma1)**2) / (sigma1 *
       np.sqrt(2*np.pi))
    rho2 = np.exp(-0.5*((grid - mu2)/sigma2)**2) / (sigma2 *
       np.sqrt(2*np.pi))

    # Discretize
    rho1 /= np.sum(rho1)
    rho2 /= np.sum(rho2)

    # Cost matrix
    C = pairwise_distances(grid[:, None], metric='sqeuclidean')

    # Sinkhorn
    import time
```

```
    start = time.time()
    ot_result = sinkhorn_distance(rho1, rho2, C, epsilon=0.05)
    comp_time = time.time() - start

    # Problem info
    problem_info = {
        'type': 'Wasserstein',
        'dimension': 1,
        'grid_size': len(grid),
        'mu': rho1,
        'nu': rho2,
        'epsilon': 0.05,
        'mol1': 'Gaussian_1',
        'mol2': 'Gaussian_2',
        'time': comp_time
    }

    # Generate certificate
    cert = generate_ot_certificate(ot_result, problem_info)

    # Export
    export_certificate_json(cert, "ot_certificate.json")

    print("\nCertificate Summary:")
    print(f"  Problem: {cert.problem_type}")
    print(f"  Transport cost: {cert.transport_cost:.8f}")
    print(f"  Marginal error: {cert.marginal_error:.2e}")
    print(f"  Valid: {cert.certificate_valid}")
```

0.4 4. Example Starting Prompt

Use this prompt to initialize a long-running AI system for optimal transport in chemistry:

```
You are a mathematical chemist implementing optimal transport theory
   for molecular systems.
Your task is to compute Wasserstein distances between electron
   densities, solve multi-marginal
OT for exact exchange energy, and construct reaction pathways via OT
   geodesics.

CONTEXT:
Optimal transport provides a rigorous framework for comparing
   probability distributions
by computing the minimal cost of transforming one into another. The
   Wasserstein distance
W_p(  ,  ) quantifies this cost, defining a metric on probability
   spaces. In quantum chemistry,
electron densities   (r) = |  (r)|   are probability distributions, and
   Wasserstein distances
offer a geometric measure of molecular similarity.

The Seidl functional reformulates DFT exact exchange as a
   multi-marginal OT problem:
```

```
E_x[  ] = -   inf_{  }        1/|r-r'| d  (r,r'), where    is a
   2-marginal with prescribed density   .
For N=2 electrons, this reduces to standard OT with Coulomb cost,
   solvable via Sinkhorn
algorithm (entropic regularization).

OBJECTIVE:
Phase 1 (Months 1-2): Implement exact Wasserstein-1 and Wasserstein-2
   distances for 1D
  densities. Test on Gaussian distributions, verify against analytical
      W   = sqrt((     -     )   + (     -     ) ).
  Compute optimal transport map T(x) =  F _     (F_  (x)).

Phase 2 (Months 2-3): Implement Sinkhorn algorithm for entropic
   regularized OT. Test on
  2D point clouds, verify marginal constraints within tolerance    =
     10      . Analyze convergence
  vs regularization parameter   .

Phase 3 (Months 3-4): Compute 3D molecular electron densities for H,
   He,  H   from wavefunctions
  (hydrogen 1s, LCAO approximation for  H  ). Discretize on grid (16
     or 32   points). Calculate
  Wasserstein-2 distances  W  ( _H ,  _He ),  W  ( _H ,   _H   ) via
     Sinkhorn.

Phase 4 (Months 4-6): Implement Seidl exchange functional for
   2-electron systems. Construct
  Coulomb cost matrix C[i,j] = 1/|r_i - r_j|. Solve 2-marginal OT via
     Sinkhorn. Compute
  E_x(He), compare to exact Hartree-Fock exchange E_x     -1.026
     Hartree.

Phase 5 (Months 6-7): Generate OT geodesics via McCann interpolation.
   Compute pathway
  connecting  H   (bonded)     2H (dissociated). Visualize intermediate
     densities  _t . Compare
  to traditional IRC (intrinsic reaction coordinate).

Phase 6 (Months 7-9): Generate machine-checkable certificates:
  - Primal cost        c d  , dual potentials   ,
  - Marginal errors ||      -   ||, ||      -   ||
  - Duality gap |Primal - Dual|
  - Complementary slackness verification
  - Export as JSON with full numerical precision

PURE THOUGHT CONSTRAINTS:
- Use ONLY exact arithmetic for 1D problems (sympy for symbolic CDFs)
- All 3D computations via Sinkhorn (entropic OT) with        0.05
- Electron densities from wavefunctions (Schr dinger equation), no
   fitting
- Certificates must verify marginal constraints within    = 10
- Compare to exact quantum chemistry (Hartree-Fock, CCSD) for validation

SUCCESS CRITERIA:
- Minimum Viable Result (2-3 months): Wasserstein distances for 1D and
```

```
   3D densities,
  Sinkhorn algorithm operational
- Strong Result (5-6 months): Seidl exchange for He within 10% of
   exact, OT geodesics
  for  H   dissociation, certificates with marginal errors <10
- Publication-Quality (9 months): Multi-marginal OT for N=3,4 electrons
   (challenging),
  novel reaction pathways, comparison with experimental/computational
     benchmarks

START:
Begin with 1D Wasserstein distances (Phase 1). Implement CDF and
   quantile function
computation from density   (x) on grid. Compute  W   and  W   for two
   Gaussians with
     =0,      =1 and      =2,      =0.5. Verify   W    =
   (     -     )   + (     -     )   = 4 + 0.25 = 4.25,
so  W   = 2.062. Export optimal transport map T(x) and certificate.
```

0.5 5. Success Criteria

0.5.1 Minimum Viable Result (MVR) - 2-3 Months

Core Functionality:

- **Wasserstein-1 and Wasserstein-2 for 1D densities: exact via quantile functions**
- **Sinkhorn algorithm: entropic OT for 2D/3D point clouds**
- **Molecular densities: H atom (1s orbital), H molecule (LCAO)**
- **Basic certificates: marginal errors, transport costs**

Deliverables:

- `wasserstein`$_1 d.py$: $Exact W, W with optimal map T(x)$
- `sinkhorn.py`: **Entropic OT solver, marginal verification**
- `molecular`$_d ensity.py$: $Hydrogen densities on 3D grid$
- `certificates.json`: **Transport costs, errors**

Quality Metrics:

- **1D Gaussians: |W*computed* - *W*exact| $<$ 10**
- **Sinkhorn marginals: || - || $<$ 10, || - || $<$ 10**
- **Molecular W: 0.5 $<$ W(H, H) $<$ 2.0 Bohr (reasonable range)**

0.5.2 Strong Result - 5-6 Months

Extended Capabilities:

- **Seidl exchange for He: $\mathbf{E}_x within 10$**

- **OT geodesics: H → 2H dissociation pathway (20 steps)**
- **3D Wasserstein: W for multiple molecules (H, He, H, Li)**
- **Certificates: dual potentials, duality gap < 10**

Deliverables:

- $\texttt{seidl}_e xchange.py : 2 - marginal OT with Coulomb cost$
- $\texttt{ot}_g eodesic.py : McCann interpolation, pathway visualization$
- **`molecularotdatabase.json`: W distances for 10+ molecular pairs**
- **Research report: "Optimal Transport in Quantum Chemistry"**

Quality Metrics:

- **Seidl He exchange: -1.1 < $\mathbf{E}_x$ < $-0.95 Hartree (within 10$**

- **OT pathway continuity: $||t+t$ - t|| t (no jumps)**
- **Wasserstein triangle inequality: W(,) W(,) + W(,) verified**
- **Duality gap: |Cost - (d + d)| < 10**

0.5.3 Publication-Quality Result - 9 Months

Novel Contributions:

- **Multi-marginal OT for N=3,4 electrons (lithium, beryllium atoms)**
- **Novel reaction coordinates: OT geodesics for SN2 reactions, proton transfer**
- **Wasserstein-based molecular similarity: clustering of organic molecules**
- **Comparison with IRC: show OT pathways differ from steepest-descent IRC**

Deliverables:

- $\texttt{mmot}_N 3.py : 3 - marginal OT(Li atom), tensor decomposition methods$
- **Research paper: "Wasserstein Geometry of Molecular Reaction Pathways"**
- **Interactive database: Molecular W distances, geodesic animations**
- **Experimental validation: Compare OT pathways to kinetics data**

Quality Metrics:

- **Li exchange (N=3): $\mathbf{E}_x within 20$**

- **Novel pathways: At least 2 reaction systems with OT geodesics**
- **Molecular clustering: Wasserstein-based dendrogram matches chemical intuition (>80**
- **IRC comparison: OT and IRC differ by >10**

0.6 6. Verification Protocol

0.6.1 Automated Checks (Run After Every Phase)

```python
def verify_ot_certificate(cert: OptimalTransportCertificate) -> Dict[str, bool]:
    """
    Verify optimal transport certificate.

    Returns: Dictionary of Boolean checks.
    """
    checks = {}

    # 1. Marginal constraints
    checks['marginals_valid'] = cert.marginal_error < 1e-4

    # 2. Duality gap
    checks['dual_gap_small'] = cert.dual_gap < 1e-4

    # 3. Positive cost
    checks['cost_positive'] = cert.transport_cost >= 0

    # 4. Sinkhorn convergence
    checks['sinkhorn_converged'] = cert.sinkhorn_iterations < 5000

    # 5. Overall validity
    checks['certificate_valid'] = cert.certificate_valid

    return checks

# Example usage
cert_example = OptimalTransportCertificate(
    problem_type="Wasserstein",
    dimension=3,
    grid_size=16,
    transport_cost=1.234,
    marginal_error=3.2e-5,
    dual_gap=8.7e-5,
    complementary_slackness_error=1.2e-4,
    molecule_1="H",
    molecule_2="H2",
    bond_length=1.4,
    epsilon_regularization=0.05,
    sinkhorn_iterations=423,
    certificate_valid=True,
    timestamp=datetime.now().isoformat(),
    computation_time=12.3
)

verification = verify_ot_certificate(cert_example)
print("Certificate Verification:")
for check, passed in verification.items():
    status = "    PASS" if passed else "    FAIL"
    print(f"  {status}: {check}")
```

0.6.2 Cross-Validation Against Known Results

```python
KNOWN_EXCHANGE_ENERGIES = {
    'He': -1.026,  # Hartree-Fock exact exchange (Hartree)
    'H2': -1.145,  # For equilibrium  H   (Hartree)
}

def cross_validate_exchange(molecule: str, computed_Ex: float):
    """Compare computed Seidl exchange to exact HF values."""
    if molecule in KNOWN_EXCHANGE_ENERGIES:
        expected = KNOWN_EXCHANGE_ENERGIES[molecule]
        error = abs(computed_Ex - expected)
        rel_error = error / abs(expected)
        print(f"{molecule}: computed={computed_Ex:.4f},
            expected={expected:.4f}, rel_error={rel_error:.2%}")
        assert rel_error < 0.15, f"Exchange energy error >{15}%
            for {molecule}"
```

0.7 7. Resources and Milestones

0.7.1 Essential References

Foundational OT:

- **C. Villani, "Optimal Transport: Old and New" (Springer, 2009) [Comprehensive textbook]**
- **L. Kantorovich, "On the Translocation of Masses", C.R. Acad. Sci. USSR 37, 199 (1942)**

OT in Chemistry:

- **M. Seidl et al., "Strictly Correlated Electrons in Density-Functional Theory", PRL 84, 5070 (2000)**
- **C. Cotar, G. Friesecke, C. Pass, "Infinite-Body Optimal Transport with Coulomb Cost", Calc. Var. 54, 717 (2015)**
- **L. Piccioni, P. Gori-Giorgi, "Electronic Correlations from Optimal Transport Theory", J. Chem. Phys. 146, 064116 (2017)**

Computational OT:

- **M. Cuturi, "Sinkhorn Distances: Lightspeed Computation of Optimal Transport", NIPS (2013)**
- **G. Peyré, M. Cuturi, "Computational Optimal Transport", Found. Trends Mach. Learn. 11, 355 (2019)**

0.7.2 Software Tools

- **POT (Python Optimal Transport) (v0.9+): State-of-the-art OT library (Sinkhorn, regularization)**
- **NumPy (v1.24+): Numerical arrays, linear algebra**
- **SciPy (scipy.optimize): Linear programming for small-scale OT**
- **PySCF (optional): Quantum chemistry for exact densities and exchange**

0.7.3 Common Pitfalls

- **Curse of Dimensionality: 3D grids with N³ points; N=64 gives 64³ 260k points—intractable for naive OT**
- **Entropic Regularization: Sinkhorn is approximate; → 0 recovers true OT but requires more iterations**
- **Coulomb Singularity: C[i,j] = 1/|r*i* - *r*j| → as r*i* → *r*j; must regularize**
- **Multi-Marginal Complexity: N-marginal OT with M atoms has M^N***variables*; *exponentialin*N**
- **McCann Interpolation: Requires optimal map T, not just coupling ; extraction from is non-trivial**

0.7.4 Milestone Checklist

Month 2:
x Wasserstein-1 and Wasserstein-2: exact for 1D Gaussians

x Optimal map T(x): computed via quantile functions

x Sinkhorn algorithm: marginal errors <10

Month 4:

Molecular densities: H, He, H on 16³ grids

Wasserstein-2 distances: W(H, H) computed

Seidl exchange (2-electron): implemented, tested on He

Month 6:

Seidl He exchange: within 10

OT geodesic: H → 2H pathway (20 steps)

Certificates: marginal errors, duality gaps exported

Month 9:

Multi-marginal OT: N=3 (Li atom) attempted

Novel pathways: SN2 or proton transfer via OT geodesics

Database: W distances for 20+ molecular pairs

Research paper draft: "Wasserstein Geometry in Quantum Chemistry"

End of PRD 18: Optimal Transport for Molecular Systems

Pure thought application of measure theory and convex optimization to quantum chemistry. Exact exchange functionals, rigorous molecular similarity metrics, and geometric reaction pathways—all verifiable via OT duality certificates.