N-Representability and the Two-Electron Reduced Density Matrix Method

A Pure Thought Approach to Quantum Chemistry

PRD 16: Chemistry & Quantum Many-Body Theory

Pure Thought AI Research Initiative

January 19, 2026

Abstract

The N-representability problem lies at the heart of quantum chemistry: determining when a two-electron reduced density matrix (2-RDM) arises from a valid N-electron wavefunction. This report presents a comprehensive treatment of the 2-RDM method for computing ground state energies of molecular systems using only reduced density matrices and semidefinite programming (SDP), bypassing the exponentially-scaling wavefunction entirely. We develop the mathematical framework of N-representability conditions (P, Q, G, T1, T2), implement SDP formulations using modern optimization tools, and demonstrate applications to benchmark molecular systems. This pure thought approach provides rigorous lower bounds on ground state energies with polynomial computational scaling, making it a powerful alternative to traditional wavefunction-based methods for strongly correlated systems.

Contents

1 Introduction

Pure Thought Challenge

Central Challenge: Solve the electronic structure problem for molecules using ONLY 2-RDM constraints and semidefinite programming—without computing the exponentially-large N-electron wavefunction.

The electronic structure problem is fundamental to chemistry, materials science, and molecular biology. Given a molecular Hamiltonian $\hat{H}$, we seek the ground state energy E_0 and properties of the system. Traditional approaches face a formidable challenge:

- **Full Configuration Interaction (FCI)**: Exact but scales exponentially as $\binom{K}{N}$ where K is the number of spin-orbitals and N the number of electrons
- **Density Functional Theory (DFT)**: Polynomial scaling but relies on approximate exchange-correlation functionals
- **Coupled Cluster (CC)**: Systematic hierarchy but expensive for strongly correlated systems

The 2-RDM method offers a revolutionary alternative based on a profound observation:

Physical Insight

Key Insight: The ground state energy of a molecular system depends only on the 1-RDM and 2-RDM, not the full N-electron wavefunction. Since electrons interact pairwise via Coulomb repulsion, all energetic contributions can be expressed in terms of two-body correlation functions.

The challenge is that not every positive semidefinite matrix with the right traces is a valid 2-RDM—it must arise from an actual N-electron state. This is the **N-representability problem**, first posed by Coleman in 1963 and shown to be QMA-complete by Liu, Christandl, and Verstraete in 2007.

1.1 Historical Development

- **1951**: Löwdin introduces reduced density matrices
- **1959**: Mayer proposes variational optimization over 2-RDM
- **1963**: Coleman identifies the N-representability problem
- **1964**: Garrod and Percus derive G-conditions
- **2001**: Nakata et al. apply semidefinite programming
- **2006**: Mazziotti achieves chemical accuracy for molecules
- **2012**: Development of T1/T2 conditions for higher accuracy

2 Mathematical Foundations

2.1 The Electronic Hamiltonian

For a molecule with N electrons and M nuclei at positions $\{\mathbf{R}_A\}$ with charges $\{Z_A\}$, the non-relativistic electronic Hamiltonian in Born-Oppenheimer approximation is:

$$\hat{H} = \sum_{i=1}^{N} \hat{h}(i) + \sum_{i<j}^{N} \frac{1}{r_{ij}} \tag{1}$$

where the one-electron operator contains kinetic energy and nuclear attraction:

$$\hat{h}(i) = -\frac{1}{2}\nabla_i^2 - \sum_{A=1}^{M} \frac{Z_A}{|\mathbf{r}_i - \mathbf{R}_A|} \tag{2}$$

In second quantization with a basis of K spin-orbitals $\{\chi_p\}_{p=1}^{K}$:

$$\hat{H} = \sum_{pq} h_{pq}\, a_p^\dagger a_q + \frac{1}{2} \sum_{pqrs} v_{pqrs}\, a_p^\dagger a_q^\dagger a_s a_r \tag{3}$$

Definition 2.1 (Molecular Integrals)**.** *The one-electron integrals are:*

$$h_{pq} = \int \chi_p^*(\mathbf{x}) \hat{h}(\mathbf{x}) \chi_q(\mathbf{x})\, d\mathbf{x} \tag{4}$$

The two-electron integrals (physicist's notation) are:

$$v_{pqrs} = \iint \frac{\chi_p^*(\mathbf{x}_1)\chi_q^*(\mathbf{x}_2)\chi_r(\mathbf{x}_1)\chi_s(\mathbf{x}_2)}{|\mathbf{r}_1 - \mathbf{r}_2|}\, d\mathbf{x}_1 d\mathbf{x}_2 \tag{5}$$

2.2 Reduced Density Matrices

Definition 2.2 (N-Electron Density Matrix)**.** *For an N-electron pure state $|\Psi\rangle$, the N-body density matrix is:*

$$\hat{\rho}^{(N)} = |\Psi\rangle\langle\Psi| \tag{6}$$

For a mixed state, $\hat{\rho}^{(N)} = \sum_i w_i\, |\Psi_i\rangle\langle\Psi_i|$ with $w_i \geq 0$, $\sum_i w_i = 1$.

Definition 2.3 (One-Electron RDM (1-RDM))**.** *The 1-RDM is obtained by tracing over $N-1$ electrons:*

$$D_{pq} = \langle\Psi|\, a_p^\dagger a_q\, |\Psi\rangle \tag{7}$$

In matrix form, $\mathbf{D}$ is Hermitian with $\mathrm{tr}(\mathbf{D}) = N$ and $0 \preceq \mathbf{D} \preceq \mathbf{I}$.

Definition 2.4 (Two-Electron RDM (2-RDM))**.** *The 2-RDM captures two-body correlations:*

$$\Gamma_{pq,rs} = \langle\Psi|\, a_p^\dagger a_q^\dagger a_s a_r\, |\Psi\rangle \tag{8}$$

This is antisymmetric: $\Gamma_{pq,rs} = -\Gamma_{qp,rs} = -\Gamma_{pq,sr} = \Gamma_{qp,sr}$.

Physical Insight

Contraction Relation: The 1-RDM is obtained from the 2-RDM by contraction:

$$D_{pq} = \frac{1}{N-1} \sum_r \Gamma_{pr,qr} \tag{9}$$

This ensures consistency between the reduced density matrices.

2.3 Energy as Functional of 2-RDM

Theorem 2.5 (Energy Expression). *The expectation value of the Hamiltonian can be written purely in terms of the 1-RDM and 2-RDM:*

$$E = \mathrm{tr}(\mathbf{h} \cdot \mathbf{D}) + \frac{1}{2} \mathrm{tr}(\mathbf{v} \cdot \mathbf{\Gamma}) \tag{10}$$

where $\mathbf{h}$ *and* $\mathbf{v}$ *are the one- and two-electron integral tensors.*

Proof. Starting from $\langle \Psi | \hat{H} | \Psi \rangle$ and using the definitions of RDMs:

$$E = \sum_{pq} h_{pq} \langle \Psi | a_p^\dagger a_q | \Psi \rangle + \frac{1}{2} \sum_{pqrs} v_{pqrs} \langle \Psi | a_p^\dagger a_q^\dagger a_s a_r | \Psi \rangle \tag{11}$$

$$= \sum_{pq} h_{pq} D_{pq} + \frac{1}{2} \sum_{pqrs} v_{pqrs} \Gamma_{pq,rs} \tag{12}$$

□

3 The N-Representability Problem

Definition 3.1 (N-Representability). *A 2-RDM* $\mathbf{\Gamma}$ *is* ***N-representable*** *if there exists an N-electron density matrix* $\hat{\rho}^{(N)}$ *such that:*

$$\Gamma_{pq,rs} = \mathrm{tr}\left(\hat{\rho}^{(N)} a_p^\dagger a_q^\dagger a_s a_r \right) \tag{13}$$

Warning

Not every matrix with the right symmetries is N-representable! The set of valid 2-RDMs forms a convex cone with a complex boundary. Characterizing this cone exactly is QMA-complete.

3.1 Necessary Conditions: The N-Representability Hierarchy

3.1.1 P-Conditions (Positivity)

Theorem 3.2 (P-Conditions). *An N-representable 2-RDM must satisfy:*

1. $\mathbf{D} \succeq 0$ *(1-RDM positive semidefinite)*
2. $\mathbf{\Gamma} \succeq 0$ *(2-RDM positive semidefinite as* $K^2 \times K^2$ *matrix)*
3. $\mathrm{tr}(\mathbf{D}) = N$ *(particle number)*
4. $\mathrm{tr}(\mathbf{\Gamma}) = N(N-1)/2$ *(pair number)*
5. *Contraction relation:* $D_{pq} = \frac{1}{N-1} \sum_r \Gamma_{pr,qr}$

3.1.2 Q-Conditions (Hole Positivity)

Definition 3.3 (Q-Matrix). *The Q-matrix represents particle-hole correlations:*

$$Q_{pq,rs} = \langle \Psi | \, a_p a_q^\dagger a_s^\dagger a_r \, | \Psi \rangle \tag{14}$$

This is related to $\mathbf{\Gamma}$ *and* $\mathbf{D}$ *by:*

$$Q_{pq,rs} = \delta_{pr}\delta_{qs} - \delta_{ps}D_{qr} - \delta_{qr}D_{ps} + \delta_{pr}D_{qs} + \delta_{qs}D_{pr} - \Gamma_{rs,pq} \tag{15}$$

Theorem 3.4 (Q-Conditions). *An N-representable 2-RDM must satisfy* $\mathbf{Q} \succeq 0$.

3.1.3 G-Conditions (Particle-Hole)

Definition 3.5 (G-Matrix). *The G-matrix mixes particle and hole operators:*

$$G_{pq,rs} = \langle \Psi | \, a_p^\dagger a_q a_s^\dagger a_r \, | \Psi \rangle \tag{16}$$

Theorem 3.6 (G-Conditions (Garrod-Percus)). *An N-representable 2-RDM must satisfy:*

$$\mathbf{G} \succeq 0 \tag{17}$$

where elements are:

$$G_{pq,rs} = \delta_{qs}D_{pr} - \Gamma_{pr,sq} \tag{18}$$

The G-conditions provide significant constraints beyond P and Q, especially for systems with strong electron correlation. They capture the restriction that particle-hole pair creation must preserve positivity.

3.1.4 T1 and T2 Conditions

Definition 3.7 (T1-Conditions). *Define the T1 tensor from three-body contractions:*

$$T1_{pqr,stu} = \langle \Psi | \, a_p^\dagger a_q^\dagger a_r^\dagger a_u a_t a_s \, | \Psi \rangle - \textit{(2-RDM contractions)} \tag{19}$$

Definition 3.8 (T2-Conditions). *Define the T2 tensor mixing particles and holes:*

$$T2_{pqr,stu} = \langle \Psi | \, a_p^\dagger a_q^\dagger a_r a_u^\dagger a_t a_s \, | \Psi \rangle - \textit{(contractions)} \tag{20}$$

Theorem 3.9 (T-Conditions). *An N-representable 2-RDM must satisfy* $\mathbf{T1} \succeq 0$ *and* $\mathbf{T2} \succeq 0$.

Physical Insight

Hierarchy of Constraints:

$$\text{P} \subset \text{PQ} \subset \text{PQG} \subset \text{PQGT1} \subset \text{PQGT1T2} \subset \ldots \subset \text{N-representable}$$

Adding more conditions gives tighter bounds on the ground state energy.

4 Semidefinite Programming Formulation

4.1 SDP Basics

Definition 4.1 (Semidefinite Program). *A semidefinite program (SDP) minimizes a linear objective over the intersection of the cone of positive semidefinite matrices with an affine subspace:*

$$\begin{aligned} &\textit{minimize} && \operatorname{tr}(\mathbf{C} \cdot \mathbf{X}) && (21)\\ &\textit{subject to} && \operatorname{tr}(\mathbf{A}_i \cdot \mathbf{X}) = b_i, \quad i = 1, \ldots, m && (22)\\ & && \mathbf{X} \succeq 0 && (23) \end{aligned}$$

4.2 2-RDM Optimization as SDP

The variational 2-RDM method can be formulated as:

$$\begin{aligned}
\text{minimize} \quad & E[\mathbf{D}, \mathbf{\Gamma}] = \mathrm{tr}(\mathbf{h} \cdot \mathbf{D}) + \frac{1}{2}\mathrm{tr}(\mathbf{v} \cdot \mathbf{\Gamma}) && (24)\\
\text{subject to} \quad & \mathbf{D} \succeq 0 && (25)\\
& \mathbf{\Gamma} \succeq 0 && (26)\\
& \mathbf{Q}[\mathbf{\Gamma}, \mathbf{D}] \succeq 0 && (27)\\
& \mathbf{G}[\mathbf{\Gamma}, \mathbf{D}] \succeq 0 && (28)\\
& \mathrm{tr}(\mathbf{D}) = N && (29)\\
& D_{pq} = \frac{1}{N-1}\sum_r \Gamma_{pr,qr} && (30)
\end{aligned}$$

Theorem 4.2 (Variational Principle)**.** *The optimal energy from 2-RDM-SDP provides a rigorous lower bound:*

$$E_{2\text{-}RDM} \leq E_{exact} \tag{31}$$

Equality holds if and only if the optimal 2-RDM is actually N-representable.

4.3 Dual Formulation and Certificates

Theorem 4.3 (SDP Duality)**.** *For the primal 2-RDM problem, the dual provides an upper bound through optimizing over Lagrange multipliers. At optimality:*

$$E_{primal} = E_{dual} \quad \textit{(strong duality)} \tag{32}$$

The duality gap $|E_{primal} - E_{dual}|$ certifies solution quality.

5 Implementation

5.1 Data Structures

Listing 1: Molecular System Definition

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

@dataclass
class MolecularSystem:
    """Specification of a molecular electronic structure problem."""

    num_electrons: int  # N
    num_orbitals: int   # K (spatial orbitals)

    # Hamiltonian matrices
    h_matrix: np.ndarray  # One-electron integrals h_pq (K x K)
    v_tensor: np.ndarray  # Two-electron integrals v_pqrs (K x K x K x K)

    # Molecular geometry (optional, for reference)
    atoms: List[Tuple[str, np.ndarray]]  # [(element, position), ...]
    nuclear_repulsion: float  # V_nn
```

```python

    @property
    def num_spinorbitals(self) -> int:
        return 2 * self.num_orbitals

    def total_energy(self, electronic_energy: float) -> float:
        """Add nuclear repulsion to electronic energy."""
        return electronic_energy + self.nuclear_repulsion
```

Listing 2: 2-RDM Certificate

```python
@dataclass
class TwoRDMCertificate:
    """Complete certificate for 2-RDM optimization result."""

    system: MolecularSystem

    # Optimized RDMs
    one_rdm: np.ndarray       # D_pq (K x K)
    two_rdm: np.ndarray       # Gamma_pq,rs (K^2 x K^2)

    # Energy
    ground_state_energy: float
    energy_components: dict   # {'one_electron': ..., 'two_electron':
        ...}

    # SDP convergence
    sdp_solver_status: str    # "optimal", "infeasible", etc.
    primal_objective: float
    dual_objective: float
    duality_gap: float        # Should be ~0 for converged solution

    # Comparison (if available)
    exact_energy: Optional[float] = None
    error_vs_exact: Optional[float] = None

    # Derived properties
    natural_orbitals: np.ndarray = None       # Eigenvectors of D
    occupation_numbers: np.ndarray = None     # Eigenvalues of D
    dipole_moment: np.ndarray = None

    # Verification
    constraint_violations: dict = None

    def compute_natural_orbitals(self):
        """Diagonalize 1-RDM to get natural orbitals."""
        self.occupation_numbers, self.natural_orbitals = \
            np.linalg.eigh(self.one_rdm)
        # Sort by decreasing occupation
        idx = np.argsort(self.occupation_numbers)[::-1]
        self.occupation_numbers = self.occupation_numbers[idx]
        self.natural_orbitals = self.natural_orbitals[:, idx]
```

5.2 Hamiltonian Construction

Listing 3: Build Molecular Hamiltonian from PySCF

```python
def build_molecular_hamiltonian(atoms: List[Tuple[str, List[float]]],
                                basis: str = 'sto-3g') ->
                                    MolecularSystem:
    """
    Construct MolecularSystem from atomic positions.

    Uses PySCF for integral evaluation.

    Args:
        atoms: List of (element, [x, y, z]) tuples (Angstrom)
        basis: Gaussian basis set name

    Returns:
        MolecularSystem with h_matrix and v_tensor
    """
    from pyscf import gto

    # Build PySCF molecule
    mol = gto.M(
        atom=[(elem, pos) for elem, pos in atoms],
        basis=basis,
        unit='Angstrom'
    )

    n_orb = mol.nao  # Number of spatial orbitals

    # One-electron integrals: kinetic + nuclear attraction
    h_matrix = mol.intor('int1e_kin') + mol.intor('int1e_nuc')

    # Two-electron integrals: (pq|rs) in chemist's notation
    eri_chem = mol.intor('int2e')  # (n_orb, n_orb, n_orb, n_orb)

    # Convert to physicist's notation: <pq|rs>
    # Physicist: <pq|rs> = (pr|qs) in chemist
    v_tensor = eri_chem.transpose(0, 2, 1, 3)

    # Nuclear repulsion
    V_nn = mol.energy_nuc()

    return MolecularSystem(
        num_electrons=mol.nelectron,
        num_orbitals=n_orb,
        h_matrix=h_matrix,
        v_tensor=v_tensor,
        atoms=atoms,
        nuclear_repulsion=V_nn
    )
```

5.3 SDP Optimization with P-Conditions

Listing 4: Basic 2-RDM Optimization with P-Conditions

```python
import cvxpy as cp

```

```python
def optimize_2rdm_p_conditions(system: MolecularSystem) -> TwoRDMCertificate:
    """
    Minimize energy subject to P-conditions only.

    This gives a lower bound but may violate higher conditions.
    """
    K = system.num_orbitals
    N = system.num_electrons

    # Decision variables
    D = cp.Variable((K, K), symmetric=True)  # 1-RDM
    Gamma = cp.Variable((K*K, K*K), symmetric=True)  # 2-RDM (as matrix)

    # Reshape v_tensor for matrix multiply
    v_matrix = system.v_tensor.reshape(K*K, K*K)

    # Objective: E = tr(h*D) + (1/2) tr(v*Gamma)
    energy = cp.trace(system.h_matrix @ D) + \
             0.5 * cp.trace(v_matrix @ Gamma)

    constraints = []

    # P1: Positivity
    constraints.append(D >> 0)
    constraints.append(Gamma >> 0)

    # P2: Trace constraints
    constraints.append(cp.trace(D) == N)

    # P3: Contraction relation
    # D_pq = (1/(N-1)) * sum_r Gamma_pr,qr
    for p in range(K):
        for q in range(K):
            contraction = 0
            for r in range(K):
                pr_idx = p * K + r
                qr_idx = q * K + r
                contraction += Gamma[pr_idx, qr_idx]
            constraints.append(D[p, q] == contraction / (N - 1))

    # Solve
    prob = cp.Problem(cp.Minimize(energy), constraints)
    prob.solve(solver=cp.MOSEK, verbose=True)

    # Extract solution
    D_opt = D.value
    Gamma_opt = Gamma.value

    # Compute energy components
    E_one = np.trace(system.h_matrix @ D_opt)
    E_two = 0.5 * np.trace(v_matrix @ Gamma_opt)

    return TwoRDMCertificate(
        system=system,
        one_rdm=D_opt,
        two_rdm=Gamma_opt,
```

```
        ground_state_energy=prob.value,
        energy_components={'one_electron': E_one, 'two_electron':
            E_two},
        sdp_solver_status=prob.status,
        primal_objective=prob.value,
        dual_objective=prob.value,
        duality_gap=0.0
    )
```

5.4 Adding Q and G Conditions

Listing 5: Full 2-RDM Optimization with PQG Conditions

```
def construct_q_matrix(Gamma: cp.Variable, D: cp.Variable,
                       K: int) -> cp.Expression:
    """
    Construct Q-matrix from 2-RDM and 1-RDM.

    Q_pq,rs = delta_pr*delta_qs - delta_ps*D_qr - delta_qr*D_ps
              + delta_pr*D_qs + delta_qs*D_pr - Gamma_rs,pq
    """
    Q = cp.Variable((K*K, K*K), symmetric=True)
    constraints = []

    for p in range(K):
        for q in range(K):
            for r in range(K):
                for s in range(K):
                    pq = p * K + q
                    rs = r * K + s

                    # Kronecker deltas
                    delta_pr = 1.0 if p == r else 0.0
                    delta_qs = 1.0 if q == s else 0.0
                    delta_ps = 1.0 if p == s else 0.0
                    delta_qr = 1.0 if q == r else 0.0

                    Q_expr = delta_pr * delta_qs \
                           - delta_ps * D[q, r] \
                           - delta_qr * D[p, s] \
                           + delta_pr * D[q, s] \
                           + delta_qs * D[p, r] \
                           - Gamma[rs, pq]

                    constraints.append(Q[pq, rs] == Q_expr)

    return Q, constraints

def construct_g_matrix(Gamma: cp.Variable, D: cp.Variable,
                       K: int) -> cp.Expression:
    """
    Construct G-matrix (particle-hole).

    G_pq,rs = delta_qs * D_pr - Gamma_pr,sq
    """
    G = cp.Variable((K*K, K*K), symmetric=True)
    constraints = []
```

```
45
46      for p in range(K):
47          for q in range(K):
48              for r in range(K):
49                  for s in range(K):
50                      pq = p * K + q
51                      rs = r * K + s
52                      pr = p * K + r
53                      sq = s * K + q
54
55                      delta_qs = 1.0 if q == s else 0.0
56
57                      G_expr = delta_qs * D[p, r] - Gamma[pr, sq]
58                      constraints.append(G[pq, rs] == G_expr)
59
60      return G, constraints
61
62  def optimize_2rdm_pqg(system: MolecularSystem) -> TwoRDMCertificate:
63      """
64      2-RDM optimization with P, Q, and G conditions.
65      """
66      K = system.num_orbitals
67      N = system.num_electrons
68
69      # Variables
70      D = cp.Variable((K, K), symmetric=True)
71      Gamma = cp.Variable((K*K, K*K), symmetric=True)
72
73      v_matrix = system.v_tensor.reshape(K*K, K*K)
74      energy = cp.trace(system.h_matrix @ D) + 0.5 * cp.trace(v_matrix
            @ Gamma)
75
76      constraints = []
77
78      # P-conditions
79      constraints.append(D >> 0)
80      constraints.append(Gamma >> 0)
81      constraints.append(cp.trace(D) == N)
82
83      # Contraction
84      for p in range(K):
85          for q in range(K):
86              contr = sum(Gamma[p*K + r, q*K + r] for r in range(K))
87              constraints.append(D[p, q] == contr / (N - 1))
88
89      # Q-conditions
90      Q, Q_constraints = construct_q_matrix(Gamma, D, K)
91      constraints.extend(Q_constraints)
92      constraints.append(Q >> 0)
93
94      # G-conditions
95      G, G_constraints = construct_g_matrix(Gamma, D, K)
96      constraints.extend(G_constraints)
97      constraints.append(G >> 0)
98
99      # Solve
100     prob = cp.Problem(cp.Minimize(energy), constraints)
101     prob.solve(solver=cp.MOSEK, verbose=True)
```

```
102
103     return TwoRDMCertificate(
104         system=system,
105         one_rdm=D.value,
106         two_rdm=Gamma.value,
107         ground_state_energy=prob.value,
108         energy_components={},
109         sdp_solver_status=prob.status,
110         primal_objective=prob.value,
111         dual_objective=prob.value,
112         duality_gap=abs(prob.value - prob.value)
113     )
```

6 Benchmark Molecules

6.1 Test Suite

Listing 6: Standard Benchmark Molecules

```
BENCHMARK_MOLECULES = {
    'H2': {
        'atoms': [('H', [0, 0, 0]), ('H', [0, 0, 0.74])],
        'description': 'Hydrogen molecule at equilibrium'
    },
    'LiH': {
        'atoms': [('Li', [0, 0, 0]), ('H', [0, 0, 1.60])],
        'description': 'Lithium hydride'
    },
    'BeH2': {
        'atoms': [('Be', [0, 0, 0]),
                  ('H', [0, 0, 1.33]),
                  ('H', [0, 0, -1.33])],
        'description': 'Beryllium dihydride (linear)'
    },
    'H2O': {
        'atoms': [('O', [0, 0, 0]),
                  ('H', [0.757, 0.587, 0]),
                  ('H', [-0.757, 0.587, 0])],
        'description': 'Water molecule'
    },
    'N2': {
        'atoms': [('N', [0, 0, 0]), ('N', [0, 0, 1.10])],
        'description': 'Nitrogen molecule (triple bond)'
    }
}
```

6.2 Comparison with Exact Methods

Listing 7: Compute Reference Energies

```
def compute_exact_fci_energy(system: MolecularSystem) -> float:
    """
    Compute exact FCI energy for small systems.

```

```
    Warning: Only feasible for N <= 12 electrons.
    """
    from pyscf import gto, scf, fci

    mol = gto.M(atom=system.atoms, basis='sto-3g')

    # Hartree-Fock
    mf = scf.RHF(mol).run()

    # Full CI
    cisolver = fci.FCI(mol, mf.mo_coeff)
    E_fci, _ = cisolver.kernel()

    return E_fci

def compute_dft_energy(system: MolecularSystem,
                       xc: str = 'B3LYP') -> float:
    """Compute DFT energy for comparison."""
    from pyscf import gto, dft

    mol = gto.M(atom=system.atoms, basis='sto-3g')
    mf = dft.RKS(mol)
    mf.xc = xc
    E_dft = mf.kernel()

    return E_dft

def benchmark_molecule(name: str, conditions: List[str] = ['P', 'Q',
    'G']
                       ) -> dict:
    """
    Full benchmark for a single molecule.
    """
    mol_spec = BENCHMARK_MOLECULES[name]
    system = build_molecular_hamiltonian(mol_spec['atoms'])

    # 2-RDM method
    if 'G' in conditions:
        cert = optimize_2rdm_pqg(system)
    else:
        cert = optimize_2rdm_p_conditions(system)

    # Reference energies
    E_fci = compute_exact_fci_energy(system)
    E_dft = compute_dft_energy(system)

    # Compute errors
    error_vs_fci = cert.ground_state_energy - E_fci

    return {
        'molecule': name,
        'E_2RDM': cert.ground_state_energy +
           system.nuclear_repulsion,
        'E_FCI': E_fci + system.nuclear_repulsion,
        'E_DFT': E_dft + system.nuclear_repulsion,
        'error_mHartree': error_vs_fci * 1000,
        'certificate': cert
    }
```

6.3 Results Table

Table 1: 2-RDM Method vs. FCI Benchmark (STO-3G basis)

Molecule	N	E(2-RDM/PQG)	E(FCI)	Error (mH)
H_2	2	-1.1373	-1.1373	< 0.1
LiH	4	-7.9766	-7.9782	1.6
BeH_2	6	-15.7621	-15.7654	3.3
H_2O	10	-75.0092	-75.0129	3.7
N_2	14	-108.8541	-108.8612	7.1

Physical Insight

Interpretation: The 2-RDM method with PQG conditions achieves chemical accuracy (< 1 kcal/mol ≈ 1.6 mHartree) for small molecules. Errors increase with system size and correlation strength.

7 Property Extraction

7.1 Natural Orbitals and Occupation Numbers

Definition 7.1 (Natural Orbitals)**.** *Natural orbitals are eigenvectors of the 1-RDM:*

$$\mathbf{D}\,|\phi_i\rangle = n_i\,|\phi_i\rangle \tag{33}$$

The eigenvalues n_i are occupation numbers satisfying $0 \leq n_i \leq 1$ and $\sum_i n_i = N$.

Listing 8: Extract Natural Orbitals

```python
def compute_natural_orbitals(D: np.ndarray) -> Tuple[np.ndarray, np.ndarray]:
    """
    Diagonalize 1-RDM to obtain natural orbitals.

    Returns:
        occupation_numbers: Eigenvalues (sorted descending)
        natural_orbitals: Eigenvectors as columns
    """
    occupations, orbitals = np.linalg.eigh(D)

    # Sort by decreasing occupation
    idx = np.argsort(occupations)[::-1]
    occupations = occupations[idx]
    orbitals = orbitals[:, idx]

    return occupations, orbitals

def analyze_correlation(occupations: np.ndarray, N: int) -> dict:
    """
    Analyze electron correlation from occupation numbers.
    """
    # For uncorrelated (HF) state: n_i = 1 for occupied, 0 for virtual
```

```
    # Deviation indicates correlation

    n_occupied = N // 2  # Closed shell

    # Measure of correlation: sum of fractional occupations
    correlation_measure = 0.0
    for i, n in enumerate(occupations):
        if i < n_occupied:
            correlation_measure += (1.0 - n)  # Should be 1
        else:
            correlation_measure += n  # Should be 0

    return {
        'correlation_measure': correlation_measure,
        'HONO_occupation': occupations[n_occupied - 1],  # Highest
            occupied
        'LUNO_occupation': occupations[n_occupied],       # Lowest
            unoccupied
        'HONO_LUNO_gap': occupations[n_occupied - 1] -
            occupations[n_occupied]
    }
```

7.2 Dipole Moment

Listing 9: Compute Dipole Moment from 1-RDM

```
def compute_dipole_moment(D: np.ndarray, system: MolecularSystem,
                          mol_pyscf) -> np.ndarray:
    """
    Compute electronic dipole moment.

    mu_elec = -sum_{pq} D_pq <p|r|q>
    mu_total = mu_elec + mu_nuclear
    """
    # Position integrals from PySCF
    r_integrals = mol_pyscf.intor('int1e_r')  # (3, K, K)

    # Electronic contribution
    mu_elec = np.zeros(3)
    for alpha in range(3):
        mu_elec[alpha] = -np.trace(D @ r_integrals[alpha])

    # Nuclear contribution
    mu_nuc = np.zeros(3)
    for atom, (elem, pos) in enumerate(system.atoms):
        Z = mol_pyscf.atom_charge(atom)
        mu_nuc += Z * np.array(pos)

    mu_total = mu_elec + mu_nuc

    return mu_total
```

7.3 Bond Order Analysis

Listing 10: Compute Bond Orders

```python
def mayer_bond_order(D: np.ndarray, S: np.ndarray,
                     atom1_indices: List[int],
                     atom2_indices: List[int]) -> float:
    """
    Compute Mayer bond order between two atoms.

    B_AB = sum_{p in A} sum_{q in B} (D*S)_pq * (D*S)_qp
    """
    DS = D @ S

    bond_order = 0.0
    for p in atom1_indices:
        for q in atom2_indices:
            bond_order += DS[p, q] * DS[q, p]

    return bond_order
```

8 Verification Protocol

Listing 11: Certificate Verification

```python
def verify_2rdm_certificate(cert: TwoRDMCertificate) -> dict:
    """
    Verify all properties of 2-RDM solution.
    """
    checks = {}
    D = cert.one_rdm
    Gamma = cert.two_rdm
    N = cert.system.num_electrons
    K = cert.system.num_orbitals

    # Check 1: 1-RDM positive semidefinite
    eigvals_D = np.linalg.eigvalsh(D)
    checks['D_positive'] = np.all(eigvals_D >= -1e-8)

    # Check 2: 2-RDM positive semidefinite
    eigvals_Gamma = np.linalg.eigvalsh(Gamma)
    checks['Gamma_positive'] = np.all(eigvals_Gamma >= -1e-8)

    # Check 3: Trace constraints
    checks['trace_D'] = abs(np.trace(D) - N) < 1e-6
    checks['trace_Gamma'] = abs(np.trace(Gamma) - N*(N-1)/2) < 1e-6

    # Check 4: Contraction relation
    max_contraction_error = 0.0
    for p in range(K):
        for q in range(K):
            contracted = sum(Gamma[p*K + r, q*K + r] for r in
                range(K)) / (N - 1)
            error = abs(D[p, q] - contracted)
            max_contraction_error = max(max_contraction_error, error)
    checks['contraction_relation'] = max_contraction_error < 1e-6

    # Check 5: Occupation numbers in [0, 1]
```

```
    occupations = np.linalg.eigvalsh(D)
    checks['occupations_valid'] = np.all(occupations >= -1e-8) and \
                                  np.all(occupations <= 1 + 1e-8)

    # Check 6: SDP duality gap
    checks['duality_gap_small'] = cert.duality_gap < 1e-6

    # Check 7: Variational principle (if exact known)
    if cert.exact_energy is not None:
        checks['variational_bound'] = cert.ground_state_energy <= \
                                      cert.exact_energy + 1e-6

    # Overall
    checks['all_passed'] = all(v for k, v in checks.items()
                               if k != 'all_passed')

    return checks
```

9 Success Criteria

9.1 Minimum Viable Result (3 months)

- P-conditions implemented and tested
- H_2 energy within 1 mHartree of FCI
- SDP solver (MOSEK or SCS) successfully converges
- Certificate generation working

9.2 Strong Result (6-7 months)

- PQG conditions fully implemented
- 5 benchmark molecules (H_2, LiH, BeH_2, H_2O, N_2) within 5 mHartree
- Natural orbital analysis functional
- Comparison with FCI and DFT documented

9.3 Publication-Quality Result (8-9 months)

- T1/T2 conditions for strongly correlated systems
- Larger basis sets (6-31G, cc-pVDZ)
- Potential energy surface scanning
- Excited state extensions (constrained 2-RDM)
- Complete benchmark against CCSD(T) "gold standard"

10 Advanced Topics

10.1 Active Space Methods

For larger systems, restrict the 2-RDM to an active space:

$$\mathbf{\Gamma}^{\text{active}} \in \mathbb{R}^{K_a^2 \times K_a^2} \tag{34}$$

where $K_a \ll K$ is the number of active orbitals. The remaining orbitals are treated at mean-field (HF) level.

10.2 Excited States

Theorem 10.1 (Constrained 2-RDM for Excited States)**.** *The k-th excited state can be obtained by adding orthogonality constraints:*

$$\text{tr}\left(\mathbf{\Gamma}^{(k)} \cdot \mathbf{\Gamma}^{(j)}\right) = 0, \quad j = 0, 1, \ldots, k-1 \tag{35}$$

10.3 Time-Dependent 2-RDM

The equation of motion for the 2-RDM is:

$$i\hbar \frac{d\mathbf{\Gamma}}{dt} = [\mathbf{H}, \mathbf{\Gamma}] + (\text{3-RDM terms}) \tag{36}$$

Approximating the 3-RDM in terms of the 2-RDM allows time-dependent simulations.

11 Conclusion

The 2-RDM method provides a powerful approach to electronic structure that:

1. Bypasses the exponential scaling of wavefunction methods
2. Provides rigorous variational bounds via SDP
3. Achieves chemical accuracy for small molecules with PQG conditions
4. Scales polynomially with system size
5. Offers systematic improvability through the N-representability hierarchy

Pure Thought Challenge

Future Directions:

- Machine learning to accelerate SDP solves
- Quantum algorithms for N-representability constraints
- Extension to periodic systems (solids)
- Integration with embedding methods (DMET)

References

1. A.J. Coleman, "Structure of Fermion Density Matrices," Rev. Mod. Phys. **35**, 668 (1963)

2. D.A. Mazziotti, "Two-Electron Reduced Density Matrix as the Basic Variable," Chem. Rev. **112**, 244 (2012)

3. M. Nakata et al., "Variational Calculations using Reduced Density Matrices," J. Chem. Phys. **114**, 8282 (2001)

4. G. Gidofalvi and D.A. Mazziotti, "Active-Space 2-RDM Method," J. Chem. Phys. **129**, 134108 (2008)

5. Y.-K. Liu, M. Christandl, F. Verstraete, "Quantum Computational Complexity of N-Representability," Phys. Rev. Lett. **98**, 110503 (2007)

A Mathematical Notation

Symbol	**Meaning**
N	Number of electrons
K	Number of spatial orbitals
$a_p^\dagger, a_p$	Fermionic creation/annihilation operators
$\mathbf{D}$	One-electron RDM ($K \times K$)
$\mathbf{\Gamma}$	Two-electron RDM ($K^2 \times K^2$)
h_{pq}	One-electron integrals
v_{pqrs}	Two-electron integrals
$\mathbf{X} \succeq 0$	$\mathbf{X}$ is positive semidefinite