PRD 16: N-Representability and the 2-RDM Method

Pure Thought AI Challenge 16

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 Quantum Many-Body Theory
Timeline: 6-9 months
Difficulty: High
Prerequisites: Quantum mechanics, linear algebra, convex optimization, functional analysis

---

0.1 1. Problem Statement

0.1.1 Scientific Context

The N-representability problem is central to quantum chemistry: Given a 2-electron reduced density matrix (2-RDM), when does it come from an N-electron wavefunction?

Traditional quantum chemistry methods scale poorly:

- **Full CI** (Configuration Interaction): Exponential in system size (intractable for N > 20 electrons)
- **DFT** (Density Functional Theory): Approximate exchange-correlation functional unknown
- **Coupled Cluster**: Polynomial scaling but still expensive for large systems

The **2-RDM method** offers a revolutionary alternative:

- The ground state energy depends ONLY on the 2-RDM (not the full N-electron wavefunction)
- 2-RDM has polynomial size: O(n) where n = number of orbitals
- **Key challenge**: Constrain 2-RDM to be N-representable (comes from actual N-electron state)

N-Representability Conditions:

- **P-conditions** (necessary, based on Pauli exclusion)
- **Q-conditions** (necessary, from 3-RDM positivity)
- **G-conditions** (necessary, from higher-order RDMs)
- **T1/T2 conditions** (stronger necessary conditions)

When combined with **semidefinite programming (SDP)**, this becomes a tractable variational method.

0.1.2 Core Question

Can we solve the electronic structure problem for molecules using ONLY 2-RDM constraints and SDP optimization—without computing the exponentially-large wavefunction?

Specifically:

- Formulate energy minimization as SDP over 2-RDM cone

- Implement P, Q, G, T conditions as SDP constraints
- Optimize 2-RDM to find ground state energy
- Extract properties: dipole moments, bond energies, excitation spectra
- Certify bounds: E*variational* *E*exact (variational principle)
- Compare accuracy to Full CI (benchmark) and DFT (practical baseline)

0.1.3 Why This Matters

Theoretical Impact:
- Bypasses exponential wall of quantum many-body problem
- Provides rigorous lower bounds on ground state energy
- Connects quantum chemistry to convex optimization

Practical Benefits:

- Polynomial scaling: can treat larger molecules than Full CI
- Systematic improvability: add tighter constraints → better energy
- Black-box: no functional approximations needed (unlike DFT)

Pure Thought Advantages:

- Hamiltonian specified exactly (Coulomb interactions + kinetic energy)
- N-representability is pure mathematics (convex geometry)
- SDP solvers provide certificates of optimality
- No empirical fitting or experimental data

0.2 2. Mathematical Formulation

0.2.1 Problem Definition

The **electronic Hamiltonian** for a molecule is:

```
1    =  _i  h_i + (1/2)  _ {ij} v_{ij}
```

where h*i is single-electron kinetic + nuclear attraction,* *v*ij = 1/|r*i* - *r*j| is Coulomb repulsion.
Ground state energy:

```
1 E_0 = min_{|     }      |  |
```

This can be rewritten using only 1-RDM (D) and 2-RDM ():

```
1 E[D,   ] = Tr(h D) + (1/2) Tr(v   )
```

2-RDM Optimization Problem:

```
minimize    E[D,   ]
subject to          N-representable cone
            Tr(D) = N  (particle number)
```

N-Representability Cone:
The set of all 2-RDMs that come from N-electron states. Characterized by:

- **P-conditions**: 0, D 0 (positivity)
- **Q-conditions**: Q 0 where Q is 3-RDM contracted from
- **G-conditions**: G-matrix 0 (particle-hole positivity)
- **T1, T2**: Higher-order positivity conditions

0.2.2 Certificate of Optimality

Given optimal 2-RDM *:

- **Energy Lower Bound**: E[*] $E_exact(variationalprinciple)$

- **Dual Certificate**: SDP dual provides rigorous bound
- **Precision**: |E[*] - E_exact| < $(certificateofaccuracy)$
- **Properties**: Extract ⟨O⟩ = Tr(O *) for any 2-body observable O

0.2.3 Input/Output Specification

Input:

```python
from sympy import *
import numpy as np
from typing import List, Tuple

class MolecularSystem:
    num_electrons: int  # N
    num_orbitals: int  # n (spatial orbitals)

    # Hamiltonian matrices
    h_matrix: np.ndarray  # h_ij (n  n): kinetic + nuclear
    v_tensor: np.ndarray  # v_ijkl (n  n  n  n): Coulomb integrals

    # Geometry (for reference, not used in optimization)
    atoms: List[Tuple[str, np.ndarray]]  # [(element, position), ...]
```

Output:

```python
class TwoRDMCertificate:
    system: MolecularSystem

    # Optimized RDMs
    one_rdm: np.ndarray  # D_ij (n  n)
    two_rdm: np.ndarray  #  _ijkl  (n  n  n  n)

    # Energy
```

```
    ground_state_energy: float  # E_0
    energy_components: dict  # {'kinetic': ..., 'nuclear': ...,
        'coulomb': ...}

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

    # Comparison
    exact_energy: Optional[float]  # If known (e.g., H2, small
        molecules)
    error_vs_exact: Optional[float]  # |E_2RDM - E_exact|

    # Properties
    dipole_moment: np.ndarray
    bond_lengths: dict  # Optimized geometry
    natural_orbitals: np.ndarray  # Eigenvectors of D
    occupation_numbers: np.ndarray  # Eigenvalues of D

    # Verification artifacts
    constraint_violations: dict  # Check P, Q, G conditions
    dual_certificate: np.ndarray  # SDP dual solution
```

0.3 3. Implementation Approach

0.3.1 Phase 1: Hamiltonian Construction (Month 1)

Build molecular Hamiltonian from atomic positions:

```
import numpy as np
from scipy.special import erf
from pyscf import gto, scf

def build_molecular_hamiltonian(atoms: List[Tuple[str, np.ndarray]],
                                basis: str = 'sto-3g') ->
                                    MolecularSystem:
    """
    Construct h_matrix and v_tensor for molecule.

    Uses Gaussian basis sets and PySCF for integral evaluation.
    """
    # Build molecule in PySCF
    mol = gto.M(
        atom=[(elem, pos) for elem, pos in atoms],
        basis=basis,
        unit='Angstrom'
    )

    # Number of spatial orbitals
    n_orb = mol.nao

    # One-electron integrals: h_ij =   i  | -     /2 + V_nuc | j
```

```
    h_matrix = mol.intor('int1e_kin') + mol.intor('int1e_nuc')

    # Two-electron integrals: v_ijkl =  ij |1/r| kl
    v_tensor = mol.intor('int2e')  # (n n n n)

    # Reshape to physicist's notation
    v_tensor = v_tensor.reshape(n_orb, n_orb, n_orb, n_orb)

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

def compute_exact_energy_small_molecules(system: MolecularSystem) ->
    float:
    """
    For benchmarking: compute exact Full CI energy for small systems.

    Only works for N    ~12 electrons (combinatorial explosion beyond).
    """
    from pyscf import fci

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

    # Full CI calculation
    mf = scf.RHF(mol).run()
    cisolver = fci.FCI(mol, mf.mo_coeff)
    E_fci = cisolver.kernel()[0]

    return E_fci
```

Validation: Reproduce H binding curve using exact integrals.

0.3.2 Phase 2: SDP Formulation with P-Conditions (Months 1-3)

Implement basic 2-RDM optimization with P-conditions only:

```
import cvxpy as cp

def optimize_2rdm_p_conditions(system: MolecularSystem) ->
    TwoRDMCertificate:
    """
    Minimize energy subject to P-conditions (positivity + traces).

    This is a semidefinite program (SDP).
    """
    n = system.num_orbitals
    N = system.num_electrons

    # Variables
```

```
    # 1-RDM: D[i,j] =    a_i     a _ j
    D = cp.Variable((n, n), symmetric=True)

    # 2-RDM:   [i,j,k,l] =    a_i     a _j  a_l  a _ k
    # Represent as matrix via reshaping
    Gamma = cp.Variable((n*n, n*n), symmetric=True)

    # Objective: E = Tr(h D) + (1/2) Tr(v   )
    h_flat = system.h_matrix.flatten()
    v_flat = system.v_tensor.reshape(n*n, n*n)

    energy = cp.trace(system.h_matrix @ D) + 0.5 * cp.trace(v_flat @
        Gamma)

    # Constraints
    constraints = []

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

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

    # Contraction: D_ij = (1/(N-1))  _k    _ijkk
    for i in range(n):
        for j in range(n):
            contraction_sum = 0
            for k in range(n):
                idx = (i*n + j, k*n + k)
                contraction_sum += Gamma[idx]

            constraints.append(D[i, j] == contraction_sum / (N - 1))

    # P3: Normalization of 2-RDM
    # Tr(  ) = N(N-1)/2
    constraints.append(cp.trace(Gamma) == N*(N-1)/2)

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

    # Extract solution
    D_opt = D.value
    Gamma_opt = Gamma.value.reshape(n, n, n, n)

    cert = TwoRDMCertificate(
        system=system,
        one_rdm=D_opt,
        two_rdm=Gamma_opt,
        ground_state_energy=prob.value,
        sdp_solver_status=prob.status,
        primal_objective=prob.value,
        dual_objective=prob.value,  # Should be equal at optimum
        duality_gap=abs(prob.value - prob.value)
    )
```

```
68
69      return cert
```

Test: H molecule—compare to exact energy, check error.

0.3.3 Phase 3: Q-Conditions (Months 3-5)

Add Q-conditions (3-RDM positivity):

```
1  def optimize_2rdm_pq_conditions(system: MolecularSystem) ->
       TwoRDMCertificate:
2      """
3      Add Q-conditions: 3-RDM contracted from 2-RDM must be positive.
4
5      Q-matrix elements: Q_ijk,lmn =     a_i      a _j    a _k   a_n a_m
           a _ l
6
7      These can be expressed as linear combinations of    elements.
8      """
9      n = system.num_orbitals
10     N = system.num_electrons
11
12     D = cp.Variable((n, n), symmetric=True)
13     Gamma = cp.Variable((n*n, n*n), symmetric=True)
14
15     energy = cp.trace(system.h_matrix @ D) + 0.5 *
           cp.trace(system.v_tensor.reshape(n*n, n*n) @ Gamma)
16
17     constraints = [
18         D >> 0,
19         Gamma >> 0,
20         cp.trace(D) == N,
21         # ... (P-conditions as before)
22     ]
23
24     # Q-conditions: Build Q-matrix from
25     # Q has size ( n       n  )  can   be very large!
26     # Use sparse representation or sampling
27
28     # Simplified Q-condition (for small systems)
29     Q_dim = n**3
30
31     Q_matrix = cp.Variable((Q_dim, Q_dim), symmetric=True)
32
33     # Relate Q to     via contraction formulas
34     # Q_ijk,lmn = (expressed as linear function of   _abcd  )
35
36     for block in generate_q_blocks(n):
37         # Each block is a smaller SDP constraint
38         constraints.append(Q_matrix[block] >> 0)
39
40     # ... (contraction relations)
41
42     prob = cp.Problem(cp.Minimize(energy), constraints)
43     prob.solve(solver=cp.MOSEK)
44
```

```
    return TwoRDMCertificate(...)
```

Challenge: Q-conditions increase problem size dramatically—need smart implementation.

0.3.4 Phase 4: G and T Conditions (Months 5-7)

Implement advanced N-representability conditions:

```
def optimize_2rdm_full_conditions(system: MolecularSystem,
                                  conditions: List[str] = ['P', 'Q',
                                     'G', 'T1', 'T2']) ->
                                     TwoRDMCertificate:
    """
    Full 2-RDM optimization with all known N-representability
       conditions.

    - G-conditions: Particle-hole duality (   and its complement both
          0)
    - T1, T2: Higher-order positivity (from 4-RDM, 5-RDM contractions)
    """
    n = system.num_orbitals
    N = system.num_electrons

    D = cp.Variable((n, n), symmetric=True)
    Gamma = cp.Variable((n**2, n**2), symmetric=True)

    energy = ...  # As before

    constraints = []

    if 'P' in conditions:
        constraints.extend(p_conditions(D, Gamma, N))

    if 'Q' in conditions:
        Q = construct_q_matrix(Gamma, n)
        constraints.append(Q >> 0)

    if 'G' in conditions:
        # G-matrix: measures particle-hole symmetry
        G = construct_g_matrix(Gamma, n, N)
        constraints.append(G >> 0)

    if 'T1' in conditions:
        # T1 conditions (from 3-RDM     4-RDM contractions)
        T1_matrices = construct_t1_conditions(Gamma, n, N)
        for T in T1_matrices:
            constraints.append(T >> 0)

    if 'T2' in conditions:
        # T2 conditions (strongest known necessary conditions)
        T2_matrices = construct_t2_conditions(Gamma, n, N)
        for T in T2_matrices:
            constraints.append(T >> 0)

    prob = cp.Problem(cp.Minimize(energy), constraints)

```

```
    # May need specialized SDP solver for large problems
    prob.solve(solver=cp.MOSEK,
        mosek_params={'MSK_DPAR_INTPNT_CO_TOL_DFEAS': 1e-8})

    cert = TwoRDMCertificate(
        system=system,
        one_rdm=D.value,
        two_rdm=Gamma.value.reshape(n, n, n, n),
        ground_state_energy=prob.value,
        sdp_solver_status=prob.status,
        primal_objective=prob.value,
        dual_objective=prob.dual_value,
        duality_gap=abs(prob.value - prob.dual_value)
    )

    # Extract properties
    cert.natural_orbitals, cert.occupation_numbers =
        np.linalg.eigh(D.value)
    cert.dipole_moment = compute_dipole(D.value, system)

    return cert

def construct_g_matrix(Gamma: cp.Variable, n: int, N: int) ->
    cp.Variable:
    """
    Construct G-matrix from 2-RDM.

    G represents particle-hole transformed 2-RDM.
    """
    # G_ijkl =  _ik   _jl  N(N-1)/2 - (N-1)  _ijkl  +  _ikjl

    G = cp.Variable((n**2, n**2), symmetric=True)

    # ... (implement G-matrix formula)

    return G
```

0.3.5 Phase 5: Benchmarking and Validation (Months 7-8)

Test on standard molecules:

```
def benchmark_suite() -> dict:
    """
    Run 2-RDM method on standard test molecules.

    Compare to Full CI (exact) and DFT (practical baseline).
    """
    test_molecules = {
        'H2': (['H', 'H'], [[0, 0, 0], [0, 0, 0.74]]),
        'LiH': (['Li', 'H'], [[0, 0, 0], [0, 0, 1.60]]),
        'BeH2': (['Be', 'H', 'H'], [[0, 0, 0], [0, 0, 1.33], [0, 0,
            -1.33]]),
        'H2O': (['O', 'H', 'H'], [[0, 0, 0], [0.76, 0.59, 0], [-0.76,
            0.59, 0]]),
        'N2': (['N', 'N'], [[0, 0, 0], [0, 0, 1.10]]),
    }
```

```
14
15      results = {}
16
17      for name, (atoms_list, positions) in test_molecules.items():
18          atoms = list(zip(atoms_list, positions))
19
20          print(f"Testing {name}...")
21
22          # Build Hamiltonian
23          system = build_molecular_hamiltonian(atoms, basis='6-31g')
24
25          # 2-RDM method
26          cert = optimize_2rdm_full_conditions(system, conditions=['P',
                'Q', 'G', 'T1'])
27
28          # Full CI (if small enough)
29          E_exact = None
30          if system.num_electrons <= 10:
31              E_exact = compute_exact_energy_small_molecules(system)
32              cert.exact_energy = E_exact
33              cert.error_vs_exact = abs(cert.ground_state_energy -
                    E_exact)
34
35          # DFT (B3LYP)
36          E_dft = compute_dft_energy(atoms, basis='6-31g')
37
38          results[name] = {
39              '2RDM_energy': cert.ground_state_energy,
40              'exact_energy': E_exact,
41              'dft_energy': E_dft,
42              '2RDM_error': cert.error_vs_exact,
43              'certificate': cert
44          }
45
46      return results
47
48  def plot_convergence_vs_conditions():
49      """
50      Show how adding more conditions improves accuracy.
51      """
52      system = build_molecular_hamiltonian([('H', [0,0,0]), ('H',
            [0,0,0.74])])
53
54      condition_sets = [
55          ['P'],
56          ['P', 'Q'],
57          ['P', 'Q', 'G'],
58          ['P', 'Q', 'G', 'T1'],
59          ['P', 'Q', 'G', 'T1', 'T2']
60      ]
61
62      energies = []
63      for conds in condition_sets:
64          cert = optimize_2rdm_full_conditions(system, conditions=conds)
65          energies.append(cert.ground_state_energy)
66
```

```
    E_exact = compute_exact_energy_small_molecules(system)

    # Plot: energy vs number of conditions
    # Should converge toward E_exact as more conditions added
```

0.3.6 Phase 6: Property Calculation and Export (Months 8-9)

Compute molecular properties from optimized 2-RDM:

```
def compute_properties_from_2rdm(cert: TwoRDMCertificate) -> dict:
    """
    Extract physical observables from 2-RDM.
    """
    D = cert.one_rdm
    Gamma = cert.two_rdm

    properties = {}

    # Dipole moment:    =  _ij  D_ij  i |r| j
    r_integrals = compute_position_integrals(cert.system)
    properties['dipole'] = np.einsum('ij,ij -> ', D, r_integrals)

    # Bond order: measure of chemical bonding strength
    properties['bond_orders'] = compute_bond_orders(D,
        cert.system.atoms)

    # Natural orbitals and occupations
    occs, nos = np.linalg.eigh(D)
    properties['natural_occupations'] = occs[::-1]  # Decreasing order
    properties['natural_orbitals'] = nos[:, ::-1]

    # Electron correlation: deviation from Hartree-Fock
    HF_energy = compute_hartree_fock(cert.system)
    properties['correlation_energy'] = cert.ground_state_energy -
        HF_energy

    # Export certificate
    export_2rdm_certificate(cert, properties)

    return properties

def export_2rdm_certificate(cert: TwoRDMCertificate, props: dict,
    output_path: Path):
    """
    Export certificate as HDF5 with all data.
    """
    import h5py

    with h5py.File(output_path, 'w') as f:
        f.create_dataset('one_rdm', data=cert.one_rdm)
        f.create_dataset('two_rdm', data=cert.two_rdm)

        f.attrs['energy'] = cert.ground_state_energy
        f.attrs['duality_gap'] = cert.duality_gap
        f.attrs['num_electrons'] = cert.system.num_electrons

```

```
        # Properties
        grp = f.create_group('properties')
        for key, val in props.items():
            grp.create_dataset(key, data=val)
```

0.4 4. Example Starting Prompt

```
You are a quantum chemist implementing the 2-RDM method for electronic
    structure. Solve
molecular Hamiltonians using ONLY RDM constraints and  S D P no
    wavefunction computation.

OBJECTIVE: Find ground state energy of  H O  using 2-RDM method,
    compare to Full CI.

PHASE 1 (Month 1): Hamiltonian construction
- Build h_matrix and v_tensor for  H O  (10 electrons, 6-31g basis)
- Validate integrals against PySCF
- Compute Hartree-Fock energy (baseline)

PHASE 2 (Months 1-3): P-conditions SDP
- Formulate energy minimization with D,    variables
- Implement positivity, trace, and contraction constraints
- Solve using MOSEK SDP solver
- Test on  H  : verify P-only gives ~80% correlation

PHASE 3 (Months 3-5): Add Q-conditions
- Construct 3-RDM positivity constraints
- Use sparse/block structure for efficiency
- Re-solve  H  : should improve to ~95% correlation

PHASE 4 (Months 5-7): G and T conditions
- Implement G-matrix (particle-hole duality)
- Add T1 conditions (if computationally feasible)
- Benchmark on LiH,  B e H ,  H O

PHASE 5 (Months 7-8): Full benchmarking
- Compare 2RDM vs Full CI vs DFT for 5 molecules
- Plot: energy error vs N-representability conditions
- Verify variational principle: E_2RDM     E_exact

PHASE 6 (Months 8-9): Properties
- Compute dipole moments, bond orders
- Extract natural orbitals and occupations
- Export certificates with dual bounds

SUCCESS CRITERIA:
- MVR:  H   energy within 1 mHartree of exact using P+Q
- Strong:  H O  energy within 5 mHartree using P+Q+G
- Publication: Systematic study of 10 molecules, scalability analysis

VERIFICATION:
- Energy satisfies E_2RDM     E_FCI (variational principle)
```

```
- Duality gap < 10       (SDP converged)
- Natural occupations     [0, 1] (N-representability)
- Dipole moments match experiment within 10%

Pure quantum mechanics + convex optimization. No empirical functionals.
All results certificate-based with SDP dual bounds.
```

0.5 5. Success Criteria

MVR (3 months): P+Q method working for H, within 1 mHartree
Strong (6-7 months): P+Q+G for 5 molecules, systematic comparison
Publication (8-9 months): Benchmark suite, scalability analysis, property calculations

---

0.6 6. Verification Protocol

- Cross-check against Full CI (Molpro, PySCF FCI module)
- Verify variational principle holds
- Check SDP dual certificates
- Compare properties to experimental values

0.7 7. Resources Milestones

References:

- Mazziotti (2012): "Two-Electron Reduced Density Matrix as the Basic Variable"
- Nakata et al. (2001): "Variational Calculations using Reduced Density Matrices"
- Gidofalvi Mazziotti (2008): "Active-Space Two-Electron Reduced-Density-Matrix Method"

Milestones:

- Month 3: P+Q working for diatomics
- Month 6: G-conditions implemented
- Month 8: Benchmark suite complete

0.8 8. Extensions

- **Excited States**: Constrained 2-RDM for electronically excited states
- **Time-Dependent**: 2-RDM approach to dynamics
- **Periodic Systems**: 2-RDM for solids (momentum-space formulation)

End of PRD 16