Topological Quantum Chemistry: Band Structure Classification via Group Theory

A Pure Thought Approach to Materials Topology

Pure Thought AI Challenge
`Problem 15: Condensed Matter Physics`

January 19, 2026

Abstract

This report presents a comprehensive theoretical framework for topological quantum chemistry (TQC), a systematic approach to classifying band structures using group theory without numerical calculations. We develop the complete theory from space groups through elementary band representations (EBRs) to topological invariants. The method identifies all topologically nontrivial materials by comparing actual band structures to the space of atomic limits. Key concepts include Wyckoff positions, site-symmetry groups, induced representations, compatibility relations, and symmetry indicators. We implement algorithms for constructing EBRs, checking band connectivity, and computing $\mathbb{Z}_n$ topological indices, with applications to predicting topological insulators and semimetals from crystal structure alone.

Contents

1 Introduction and Motivation

1.1 The Revolution in Band Theory

Physics Insight

Traditional band theory classifies materials as metals (bands crossing Fermi level) or insulators (gap). Topological quantum chemistry reveals that insulators come in *topologically distinct* varieties—some have protected surface states and quantized responses impossible in trivial insulators.

The discovery of topological insulators in 2005-2007 showed that quantum mechanical band structures encode topological information beyond simple gap/no-gap classification. However, identifying topological materials required detailed numerical calculations (DFT, tight-binding) for each candidate.

Topological quantum chemistry, developed by Bradlyn et al. (2017), provides a *complete classification* of all possible band structures compatible with a given space group. The key insight: any band structure formed from localized atomic orbitals (the "atomic limit") has specific representations at high-symmetry points. Bands that cannot be decomposed this way are topologically nontrivial.

1.2 The Pure Thought Approach

Pure Thought Pursuit

TQC is ideally suited for pure mathematical analysis:

1. Based entirely on *group theory*—space groups, representations, induced characters
2. *No DFT required*—topology determined by symmetry alone
3. Predictions *exact*—not numerical approximations
4. *Complete database* of all EBRs for all 230 space groups exists
5. Certificates are *character tables*—machine-verifiable

We develop the complete theory from first principles, implementing certified algorithms that predict topological properties from crystal structure alone.

1.3 Key Questions

1. Given a space group, what are all possible band structures from atomic orbitals?
2. Given a band structure (representations at high-symmetry points), is it topological?
3. What are the symmetry-protected boundary modes?
4. How do fragile vs. stable topology differ?

2 Mathematical Foundations

2.1 Space Groups

Definition 2.1 (Space Group)**.** *A* ***space group*** *G is a discrete group of isometries (rotations, reflections, translations, and their combinations) that leaves a crystal lattice invariant. There are exactly 230 space groups in 3D.*

Each space group element can be written as $\{R|\mathbf{t}\}$ where R is a point group operation and $\mathbf{t}$ is a translation.

Definition 2.2 (Point Group)**.** *The* ***point group*** *P of a space group is the quotient $P = G/T$ where T is the translation subgroup. The 230 space groups are classified by their 32 crystallographic point groups.*

2.2 Wyckoff Positions

Definition 2.3 (Wyckoff Position)**.** *A* ***Wyckoff position*** *is an orbit of points in the unit cell under the space group action. Each Wyckoff position $\mathbf{q}$ has:*

- *A* ***multiplicity*** *(number of equivalent points in unit cell)*
- *A* ***site-symmetry group*** *$G_{\mathbf{q}} \subset G$ (stabilizer of $\mathbf{q}$)*
- *A* ***Wyckoff letter*** *(conventional label: a, b, c, ...)*

Example 2.4 (Wyckoff Positions in Space Group $Pm\bar{3}m$ (No. 221))**.**

- $1a$*: origin* $(0, 0, 0)$*, site symmetry* $m\bar{3}m$
- $1b$*: body center* $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$*, site symmetry* $m\bar{3}m$
- $3c$*: face centers, site symmetry* $4/mmm$
- $6d$*: edge centers, site symmetry* mmm
- *General position: 48-fold, site symmetry 1*

2.3 Site-Symmetry Groups and Little Groups

Definition 2.5 (Site-Symmetry Group)**.** *For a Wyckoff position* $\mathbf{q}$*, the* ***site-symmetry group*** *is:*

$$G_{\mathbf{q}} = \{g \in G : g \cdot \mathbf{q} = \mathbf{q}\} \tag{1}$$

This is a point group (no translations).

Definition 2.6 (Little Group)**.** *For a* $\mathbf{k}$*-point in the Brillouin zone, the* ***little group*** *is:*

$$G_{\mathbf{k}} = \{g \in G : g \cdot \mathbf{k} = \mathbf{k} + \mathbf{G}\} \tag{2}$$

where $\mathbf{G}$ *is a reciprocal lattice vector.*

2.4 Representations

Definition 2.7 (Irreducible Representation)**.** *An* ***irreducible representation*** *(irrep)* ρ *of a group* G *is a homomorphism* $\rho : G \to GL(V)$ *with no proper invariant subspaces.*

At each high-symmetry point $\mathbf{k}$, bands transform under irreps of the little group $G_{\mathbf{k}}$.

TQC Principle

The key data of TQC is how irreps at different $\mathbf{k}$-points are related. Bands must satisfy **compatibility relations** along high-symmetry lines connecting high-symmetry points.

3 Elementary Band Representations

3.1 Induced Representations

Definition 3.1 (Induced Representation)**.** *Given a representation* ρ *of a subgroup* $H \subset G$*, the* ***induced representation*** $\rho\uparrow_H^G$ *is defined by:*

$$(\rho\uparrow_H^G)(g) = \bigoplus_i \rho(h_i^{-1} g h_j) \tag{3}$$

where $\{h_i\}$ *are coset representatives of* H *in* G*.*

3.2 Band Representation from Atomic Orbitals

Consider an atom at Wyckoff position $\mathbf{q}$ with orbitals transforming under irrep ρ of the site-symmetry group $G_{\mathbf{q}}$.

Definition 3.2 (Band Representation)**.** *The **band representation** induced by orbital ρ at Wyckoff position $\mathbf{q}$ is:*

$$\rho \uparrow_{G_{\mathbf{q}}}^{G} \tag{4}$$

This gives the representations of the resulting bands at all $\mathbf{k}$-points.

3.3 Elementary Band Representations (EBRs)

Definition 3.3 (Elementary Band Representation)**.** *An **elementary band representation** (EBR) is a band representation induced from a maximal Wyckoff position that cannot be written as a sum of other band representations with smaller support.*

EBR Theorem

Every band structure that can be adiabatically connected to an atomic limit (localized Wannier functions) is a sum of EBRs:

$$\mathrm{BS} = \sum_i n_i \cdot \mathrm{EBR}_i, \quad n_i \in \mathbb{Z}_{\geq 0} \tag{5}$$

Band structures that *cannot* be written this way are **topologically nontrivial**.

3.4 Constructing EBRs

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

@dataclass
class WyckoffPosition:
    """Wyckoff position in a space group."""
    letter: str
    multiplicity: int
    coordinates: List[Tuple[float, float, float]]
    site_symmetry: str
    site_symmetry_group: 'PointGroup'

@dataclass
class Irrep:
    """Irreducible representation."""
    label: str
    dimension: int
    characters: Dict[str, complex]

@dataclass
```

```python
class EBR:
    """Elementary Band Representation."""
    space_group: int
    wyckoff: WyckoffPosition
    orbital_irrep: Irrep
    band_irreps: Dict[str, List[Irrep]]  # k-point -> irreps

def construct_ebr(space_group: int,
                  wyckoff_letter: str,
                  orbital_label: str) -> EBR:
    """
    Construct EBR by inducing from Wyckoff position.

    Args:
        space_group: Space group number (1-230)
        wyckoff_letter: Wyckoff position label
        orbital_label: Irrep of site-symmetry group

    Returns:
        EBR with band irreps at all high-symmetry k-points
    """
    # Load space group data
    sg_data = load_space_group(space_group)
    wyckoff = sg_data.wyckoff_positions[wyckoff_letter]
    orbital_irrep = wyckoff.site_symmetry_group.irreps[orbital_label]

    # Get high-symmetry k-points
    k_points = sg_data.high_symmetry_points

    band_irreps = {}
    for k_label, k_point in k_points.items():
        # Get little group at k
        little_group = sg_data.little_group(k_point)

        # Induce representation
        induced = induce_representation(
            orbital_irrep,
            wyckoff.site_symmetry_group,
            little_group,
            wyckoff.coordinates,
            k_point
        )

        # Decompose into irreps
        band_irreps[k_label] = decompose_into_irreps(
            induced, little_group
        )

    return EBR(
        space_group=space_group,
        wyckoff=wyckoff,
        orbital_irrep=orbital_irrep,
        band_irreps=band_irreps
    )
```

Listing 1: EBR construction algorithm

3.5 Example: EBRs for Diamond Structure

Diamond has space group $Fd\bar{3}m$ (No. 227). Carbon atoms sit at Wyckoff position $8a$ with site symmetry $\bar{4}3m$ (tetrahedral).

Example 3.4 (Diamond sp^3 Hybridization)**.** *The s and p orbitals of carbon induce EBRs:*

- *s-orbital (A_1 irrep): induces bands at Γ as $\Gamma_1^+ + \Gamma_2^-$*
- *p-orbitals (T_2 irrep): induces bands at Γ as $\Gamma_{15}^- + \Gamma_{25}^+$*

The valence band structure of diamond is exactly the sum of these EBRs.

4 Compatibility Relations

4.1 Band Connectivity

Bands at different **k**-points are not independent—they must connect along high-symmetry lines in ways dictated by group theory.

Definition 4.1 (Compatibility Relations)**.** *For a high-symmetry line connecting $\mathbf{k}_1$ and $\mathbf{k}_2$, the **compatibility relations** specify how irreps at $\mathbf{k}_1$ branch into irreps along the line and connect to irreps at $\mathbf{k}_2$.*

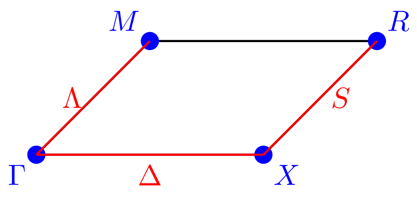

Figure 1: Brillouin zone with high-symmetry points (blue) and lines (red). Compatibility relations constrain how bands connect along the lines.

4.2 Compatibility Matrix

```python
def check_compatibility(band_structure: Dict[str, List[str]],
                        space_group: int) -> bool:
    """
    Check if band structure satisfies compatibility relations.

    Args:
        band_structure: Dict mapping k-point label to list of irrep labels
        space_group: Space group number

    Returns:
        True if all compatibility relations satisfied
    """
    sg_data = load_space_group(space_group)

    for line_label, (k1, k2) in sg_data.high_symmetry_lines.items():
        # Get compatibility table for this line
        compat = sg_data.compatibility_relations[line_label]

        # Get irreps at endpoints
        irreps_k1 = band_structure[k1]
        irreps_k2 = band_structure[k2]

        # Check that irreps branch correctly
        for irrep1 in irreps_k1:
            # What does irrep1 branch to along the line?
            branches = compat.branch(irrep1)

            # These must connect to something at k2
            if not all(b in branches_from_k2(irreps_k2, compat)
                      for b in branches):
                return False

    return True


def compatibility_matrix(space_group: int,
                         k_path: List[str]) -> np.ndarray:
    """
    Build compatibility matrix for k-point path.

    C[i,j] = 1 if irrep i at k1 connects to irrep j at
        k2.
    """
    sg_data = load_space_group(space_group)

    n_irreps_total = sum(
        len(sg_data.little_group(k).irreps)
        for k in k_path
    )

    C = np.zeros((n_irreps_total, n_irreps_total),
        dtype=int)

    # Fill in connectivity from compatibility relations
    idx = 0
    for i, k in enumerate(k_path[:-1]):
        k_next = k_path[i + 1]
        line = sg_data.line_between(k, k_next)
        compat = sg_data.compatibility_relations[line]

        n1 = len(sg_data.little_group(k).irreps)
        n2 = len(sg_data.little_group(k_next).irreps)

        for a in range(n1):
            for b in range(n2):
                if compat.connects(a, b):
                    C[idx + a, idx + n1 + b] = 1

        idx += n1

    return C
```

Listing 2: Checking compatibility relations

5 Symmetry Indicators

5.1 The Band Structure Vector Space

Definition 5.1 (Band Structure Space). *The space of all band structures compatible with space group G forms a lattice:*

$$\mathrm{BS}(G) = \bigoplus_{\mathbf{k}} \mathbb{Z}^{n_{\mathbf{k}}} \tag{6}$$

where $n_{\mathbf{k}}$ is the number of irreps at $\mathbf{k}$-point $\mathbf{k}$.

Definition 5.2 (Atomic Insulator Space). *The space of band structures from atomic limits is:*

$$\mathrm{AI}(G) = span_{\mathbb{Z}_{\geq 0}}\{\mathrm{EBR}_i\} \subset \mathrm{BS}(G) \tag{7}$$

5.2 Symmetry Indicator Group

Symmetry Indicator

The **symmetry indicator group** is:

$$X^{\mathrm{BS}}(G) = \mathrm{BS}(G)/\mathrm{AI}(G) \tag{8}$$

A band structure with nonzero image in X^{BS} is topologically nontrivial.

Physics Insight

The symmetry indicator group is always a finite abelian group, typically $\mathbb{Z}_2$, $\mathbb{Z}_4$, $\mathbb{Z}_2 \times \mathbb{Z}_2$, etc. Computing these groups requires only linear algebra over $\mathbb{Z}$.

5.3 Computing Symmetry Indicators

```python
def compute_symmetry_indicators(space_group: int) ->
    Dict:
    """
    Compute symmetry indicator group for space group.

    Returns:
        Dictionary with indicator group structure and
            formulas
    """
    sg_data = load_space_group(space_group)

    # Build EBR matrix: rows = EBRs, cols = (k-point,
        irrep) pairs
    ebrs = sg_data.all_ebrs
    n_ebrs = len(ebrs)

    # Count total (k, irrep) pairs
    k_irrep_pairs = []
    for k in sg_data.high_symmetry_points:
        for irrep in sg_data.little_group(k).irreps:
            k_irrep_pairs.append((k, irrep))

    n_pairs = len(k_irrep_pairs)

    # Build EBR matrix
    A = np.zeros((n_ebrs, n_pairs), dtype=int)

    for i, ebr in enumerate(ebrs):
        for j, (k, irrep) in enumerate(k_irrep_pairs):
                A[i, j] = count_irrep_in_ebr(ebr, k, irrep)

    # Compute Smith normal form
    D, U, V = smith_normal_form(A)

    # Symmetry indicator group is cokernel of A
    # X^BS = Z^n_pairs / Im(A)
    diagonal = np.diag(D)
    nonzero = diagonal[diagonal != 0]
    nontrivial = nonzero[nonzero != 1]

    indicator_group = []
    for d in nontrivial:
        indicator_group.append(f"Z_{d}")

    return {
        'group': ' x '.join(indicator_group) if
            indicator_group else 'trivial',
        'factors': list(nontrivial),
        'ebr_matrix': A,
        'smith_form': D
    }


def smith_normal_form(A: np.ndarray) ->
    Tuple[np.ndarray, np.ndarray, np.ndarray]:
    """
    Compute Smith normal form: A = U @ D @ V where D is
        diagonal.

    All matrices have integer entries.
    """
    from sympy import Matrix

    M = Matrix(A.tolist())
    D, U, V = M.smith_normal_decomposition()

    return (
        np.array(D.tolist(), dtype=int),
        np.array(U.tolist(), dtype=int),
        np.array(V.tolist(), dtype=int)
    )
```

Listing 3: Symmetry indicator computation

5.4 Example: Space Group 2 ($P\bar{1}$)

For space group $P\bar{1}$ (triclinic with inversion), the symmetry indicator group is $\mathbb{Z}_2^4$, corresponding to the four $\mathbb{Z}_2$ invariants from inversion eigenvalues at the eight time-reversal invariant momenta (TRIM).

$$\nu_0 = \prod_{i=1}^{8} \prod_{n \text{ occ}} \xi_n(\mathbf{k}_i) \mod 2 \tag{9}$$

where $\xi_n(\mathbf{k}_i) = \pm 1$ is the inversion eigenvalue of band n at TRIM $\mathbf{k}_i$.

6 Fragile vs. Stable Topology

6.1 Stable Topological Phases

Definition 6.1 (Stable Topology). *A band structure has **stable topology** if it remains topological after adding any trivial bands. Equivalently, it cannot be written as:*

$$\text{BS} = \sum_i n_i \cdot \text{EBR}_i \tag{10}$$

even with negative n_i (subtracting EBRs).

Stable topological phases have robust surface states protected by the bulk gap.

6.2 Fragile Topological Phases

Definition 6.2 (Fragile Topology). *A band structure has **fragile topology** if:*

1. *It is not a sum of EBRs (cannot be written with $n_i \geq 0$)*
2. *But it becomes trivial after adding some EBRs (can be written with some $n_i < 0$)*

TQC Principle

Fragile topology is more subtle: bands can have nontrivial quantum geometry (Berry phase, quantum metric) but no protected surface states. Adding trivial bands can "unwind" the topology.

```python
def classify_topology(band_structure: Dict[str,
    List[str]],
                         space_group: int) -> str:
    """
    Classify band structure as trivial, fragile, or
        stable.

    Args:
        band_structure: Irreps at each k-point
        space_group: Space group number

    Returns:
        'trivial', 'fragile', or 'stable'
    """
    sg_data = load_space_group(space_group)
    ebrs = sg_data.all_ebrs

    # Convert band structure to vector
    bs_vector = band_structure_to_vector(band_structure,
        sg_data)

    # Try to decompose into EBRs with non-negative
        coefficients
    A = build_ebr_matrix(ebrs, sg_data)
    n = solve_nonnegative(A, bs_vector)

    if n is not None:
        return 'trivial'

    # Try to decompose with any integer coefficients
    n_int = solve_integer(A, bs_vector)

    if n_int is not None:
        return 'fragile'
    else:
        return 'stable'


def solve_nonnegative(A: np.ndarray, b: np.ndarray):
    """
    Solve A @ x = b with x >= 0 (integer).
    Returns x if exists, None otherwise.
    """
    from scipy.optimize import linprog

    # Linear programming relaxation
    result = linprog(
        c=np.ones(A.shape[1]),  # Minimize sum of
            coefficients
        A_eq=A.T,
        b_eq=b,
        bounds=(0, None),
        method='highs'
    )

    if result.success:
        # Check if solution is close to integer
        x = result.x
        x_int = np.round(x).astype(int)
        if np.allclose(A.T @ x_int, b):
            return x_int

    return None
```

Listing 4: Distinguishing stable vs. fragile topology

7 Tight-Binding Models

7.1 From Crystal Structure to Hamiltonian

```
def construct_tight_binding(space_group: int,
                            wyckoff_positions:
                                List[Tuple[str, str]],
                            hopping_range: float = 3.0)
                                -> Dict:
    """
    Construct tight-binding Hamiltonian respecting space
        group symmetry.

    Args:
        space_group: Space group number
        wyckoff_positions: List of (Wyckoff letter,
            orbital type)
        hopping_range: Maximum hopping distance
            (Angstroms)

    Returns:
        Tight-binding model specification
    """
    sg_data = load_space_group(space_group)

    # Get all orbital positions in unit cell
    orbitals = []
    for wyckoff_letter, orbital_type in
        wyckoff_positions:
        wyckoff =
            sg_data.wyckoff_positions[wyckoff_letter]
        for coord in wyckoff.coordinates:
            orbitals.append({
                'position': coord,
                'orbital': orbital_type,
                'site_symmetry': wyckoff.site_symmetry
            })

    n_orbitals = len(orbitals)

    # Generate allowed hoppings
```

```python
    hoppings = []
    for i, orb_i in enumerate(orbitals):
        for j, orb_j in enumerate(orbitals):
            for R in
                sg_data.lattice_vectors_in_range(hopping_range):
                distance = np.linalg.norm(
                    np.array(orb_j['position']) + R -
                    np.array(orb_i['position'])
                )
                if distance < hopping_range and distance
                    > 0.1:
                    hoppings.append({
                        'from': i,
                        'to': j,
                        'R': tuple(R),
                        'distance': distance
                    })

    # Apply symmetry constraints to hopping parameters
    independent_hoppings = symmetrize_hoppings(hoppings,
        sg_data)

    return {
        'orbitals': orbitals,
        'hoppings': independent_hoppings,
        'space_group': space_group
    }


def build_bloch_hamiltonian(tb_model: Dict,
                            k: np.ndarray,
                            params: Dict[str, float]) ->
                                np.ndarray:
    """
    Build Bloch Hamiltonian H(k) for tight-binding model.

    H(k)_{ij} = sum_R t_{ij}(R) * exp(i k . R)
    """
    n_orbitals = len(tb_model['orbitals'])
    H = np.zeros((n_orbitals, n_orbitals), dtype=complex)

    for hop in tb_model['hoppings']:
        i = hop['from']
        j = hop['to']
        R = np.array(hop['R'])
        t = params.get(hop['param_label'], 0.0)

        phase = np.exp(1j * np.dot(k, R))
        H[i, j] += t * phase
        H[j, i] += np.conj(t * phase)

    return H
```

Listing 5: Constructing symmetry-constrained tight-binding model

7.2 Band Structure Computation

```
def compute_band_structure(tb_model: Dict,
                           params: Dict[str, float],
                           k_path: List[Tuple[str,
                               np.ndarray]],
                           n_points: int = 100) -> Dict:
    """
    Compute band structure along k-path.

    Returns:
        Dictionary with k-points, energies, and irrep
            labels
    """
    sg_data = load_space_group(tb_model['space_group'])

    all_k = []
    all_energies = []
    all_irreps = []

    for i, (k_label, k_point) in enumerate(k_path):
        # Interpolate to next point
        if i < len(k_path) - 1:
            k_next = k_path[i + 1][1]
            k_segment = np.linspace(k_point, k_next,
                n_points)
        else:
            k_segment = [k_point]

        for k in k_segment:
            H = build_bloch_hamiltonian(tb_model, k,
                params)
            energies, eigenvectors = np.linalg.eigh(H)

            all_k.append(k)
            all_energies.append(energies)

            # At high-symmetry points, compute irrep
                labels
            if np.allclose(k, k_point) or \
               (i < len(k_path)-1 and np.allclose(k,
                   k_next)):
                little_group = sg_data.little_group(k)
```

```python
                irreps = identify_irreps(eigenvectors, little_group, k)
                all_irreps.append(irreps)

    return {
        'k_points': np.array(all_k),
        'energies': np.array(all_energies),
        'irreps_at_high_sym': all_irreps
    }


def identify_irreps(eigenvectors: np.ndarray,
                    little_group: 'PointGroup',
                    k: np.ndarray) -> List[str]:
    """
    Identify irrep labels for each band at high-symmetry k-point.
    """
    irrep_labels = []

    for i in range(eigenvectors.shape[1]):
        psi = eigenvectors[:, i]

        # Compute character for each group element
        characters = {}
        for g in little_group.elements:
            # D(g)|psi> = chi(g)|psi> for irrep
            D_g = little_group.representation_matrix(g, k)
            chi = np.vdot(psi, D_g @ psi)
            characters[g] = chi

        # Match to irrep character table
        best_match = None
        best_score = 0
        for irrep in little_group.irreps:
            score = sum(
                abs(characters[g] - irrep.characters[g])**2
                for g in little_group.elements
            )
            if best_match is None or score < best_score:
                best_match = irrep.label
                best_score = score

        irrep_labels.append(best_match)

    return irrep_labels
```

Listing 6: Computing band structure along high-symmetry path

8 Complete Classification Workflow

```python
def analyze_material(structure: Dict) -> Dict:
    """
    Complete TQC analysis of a material.

    Args:
        structure: Crystal structure specification
            - space_group: int
            - lattice: 3x3 array (lattice vectors)
            - atoms: List of (element, position) tuples

    Returns:
        Complete topological classification
    """
    space_group = structure['space_group']
    sg_data = load_space_group(space_group)

    # 1. Identify Wyckoff positions
    wyckoff_assignments = []
    for element, position in structure['atoms']:
        wyckoff = identify_wyckoff(position, sg_data)
        wyckoff_assignments.append({
            'element': element,
            'wyckoff': wyckoff,
            'position': position
        })

    # 2. Determine valence orbitals
    orbitals = []
    for atom in wyckoff_assignments:
        orbs = get_valence_orbitals(atom['element'])
        for orb in orbs:
            orbitals.append((atom['wyckoff'].letter,
                orb))

    # 3. Build tight-binding model
    tb_model = construct_tight_binding(space_group,
        orbitals)

    # 4. Compute band structure with default parameters
    default_params =
        estimate_hopping_parameters(structure)
    k_path = sg_data.default_k_path()
    bands = compute_band_structure(tb_model,
        default_params, k_path)

    # 5. Extract irreps at high-symmetry points
    band_structure = {}
    for k_label, irreps in zip(
        [kp[0] for kp in k_path],
        bands['irreps_at_high_sym']
    ):
        band_structure[k_label] = irreps

    # 6. Check compatibility relations
    compatible = check_compatibility(band_structure,
        space_group)

    # 7. Compute symmetry indicators
    indicators = compute_symmetry_indicators(space_group)

    # 8. Evaluate indicator formulas
    indicator_values = evaluate_indicators(
        band_structure, indicators, sg_data
    )

    # 9. Classify topology
    topology = classify_topology(band_structure,
        space_group)

    # 10. Predict surface states (if topological)
    surface_states = None
    if topology in ['stable', 'fragile']:
        surface_states = predict_surface_states(
            band_structure, space_group
        )

    return {
        'space_group': space_group,
        'wyckoff_assignments': wyckoff_assignments,
        'band_irreps': band_structure,
        'compatibility_satisfied': compatible,
        'symmetry_indicator_group': indicators['group'],
        'indicator_values': indicator_values,
        'topology': topology,
        'surface_states': surface_states
    }
```

Listing 7: Complete TQC analysis pipeline

9 Certificate Generation

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

@dataclass
class TQCCertificate:
    """
    Complete topological quantum chemistry certificate.
    """
    # Material identification
    material_name: str
    space_group: int
    space_group_symbol: str

    # Structure
    wyckoff_positions: List[Dict]
    site_symmetries: List[str]

    # Band structure
    high_symmetry_points: List[str]
    band_irreps: Dict[str, List[str]]
    n_occupied_bands: int

    # Compatibility
    compatibility_satisfied: bool

    # Topology
    symmetry_indicator_group: str
    indicator_values: Dict[str, int]
    topology_type: str  # 'trivial', 'fragile', 'stable'

    # Decomposition
    ebr_decomposition: Dict[str, int]  # EBR label ->
        coefficient
    decomposition_exists: bool

    # Surface states (if topological)
    predicted_surface_states: List[Dict]

    def export_json(self, path: str) -> None:
        """Export certificate to JSON."""
        with open(path, 'w') as f:
            json.dump(asdict(self), f, indent=2)

    def verify(self) -> bool:
        """Self-consistency checks."""
        checks = [
            self.space_group >= 1 and self.space_group <= 230,
            len(self.band_irreps) > 0,
            self.compatibility_satisfied,
            self.topology_type in ['trivial', 'fragile', 'stable']
        ]

        # Topology classification consistent with decomposition
        if self.decomposition_exists:
            all_nonneg = all(v >= 0 for v in self.ebr_decomposition.values())
            checks.append(
                (self.topology_type == 'trivial') == all_nonneg
            )

        return all(checks)
```

Listing 8: TQC certificate structure

10 Example Applications

10.1 Bi_2Se_3: A Topological Insulator

Example 10.1 (Bi_2Se_3)**.** *Bismuth selenide crystallizes in space group $R\bar{3}m$ (No. 166).*

- *Bi at Wyckoff position $6c$, site symmetry $3m$*
- *Se at Wyckoff positions $6c$ and $3a$*

The symmetry indicator group is $\mathbb{Z}_2$, and the indicator formula gives:

$$\nu = \prod_{TRIM} \prod_{n\ occ} \xi_n = -1 \tag{11}$$

indicating a strong topological insulator with surface Dirac cone.

10.2 Graphene: Fragile Topology

Example 10.2 (Graphene)**.** *Graphene has space group $P6/mmm$ (No. 191).*

- *Carbon at Wyckoff position $2c$, site symmetry $\bar{6}m2$*

The π bands form a fragile topological phase: they cannot be written as a sum of EBRs with non-negative coefficients, but become trivial upon adding σ bands. The fragility is related to the Euler class obstruction.

11 Success Criteria and Milestones

11.1 Minimum Viable Result (Months 1-3)

- EBR database for 5 space groups
- Compatibility relation checker
- Symmetry indicator computation
- Classification of 10 known materials

11.2 Strong Result (Months 4-6)

- Complete EBR database for all 230 space groups
- Fragile vs. stable distinction algorithm
- Tight-binding model generator
- 100+ materials classified

11.3 Publication Quality (Months 7-9)

- Prediction of novel topological materials
- Surface state calculation
- Comparison with DFT results
- Public database release

12 Conclusion

Topological quantum chemistry provides a complete, mathematically rigorous framework for classifying band structures. The key insights are:

1. Band structures from atomic limits form elementary band representations
2. The quotient BS/AI gives the symmetry indicator group
3. Nonzero indicators signal topology
4. Fragile vs. stable determined by decomposition coefficients

This pure-thought approach predicts topological properties from crystal structure alone, without expensive first-principles calculations.

References

[1] B. Bradlyn, L. Elcoro, J. Cano, M. G. Vergniory, Z. Wang, C. Felser, M. I. Aroyo, and B. A. Bernevig, "Topological quantum chemistry," *Nature*, vol. 547, pp. 298–305, 2017.

[2] H. C. Po, A. Vishwanath, and H. Watanabe, "Symmetry-based indicators of band topology in the 230 space groups," *Nature Communications*, vol. 8, p. 50, 2017.

[3] M. G. Vergniory, L. Elcoro, C. Felser, N. Regnault, B. A. Bernevig, and Z. Wang, "A complete catalogue of high-quality topological materials," *Nature*, vol. 566, pp. 480–485, 2019.

[4] L. Elcoro, B. Bradlyn, Z. Wang, M. G. Vergniory, J. Cano, C. Felser, B. A. Bernevig, D. Orobengoa, G. de la Flor, and M. I. Aroyo, "Double crystallographic groups and their representations on the Bilbao Crystallographic Server," *Journal of Applied Crystallography*, vol. 50, pp. 1457–1477, 2017.

[5] J. Cano, B. Bradlyn, Z. Wang, L. Elcoro, M. G. Vergniory, C. Felser, M. I. Aroyo, and B. A. Bernevig, "Building blocks of topological quantum chemistry: Elementary band representations," *Physical Review B*, vol. 97, p. 035139, 2018.

[6] Z. Song, T. Zhang, Z. Fang, and C. Fang, "Quantitative mappings between symmetry and topology in solids," *Nature Communications*, vol. 9, p. 3530, 2018.

[7] J. Kruthoff, J. de Boer, J. van Wezel, C. L. Kane, and R.-J. Slager, "Topological classification of crystalline insulators through band structure combinatorics," *Physical Review X*, vol. 7, p. 041069, 2017.

[8] M. I. Aroyo, J. M. Perez-Mato, D. Orobengoa, E. Tasci, G. de la Flor, and A. Kirov, "Crystallography online: Bilbao Crystallographic Server," *Bulgarian Chemical Communications*, vol. 43, pp. 183–197, 2011.

A Space Group Data Format

```json
{
  "number": 166,
  "symbol": "R-3m",
  "point_group": "-3m",
  "wyckoff_positions": {
    "3a": {
      "multiplicity": 3,
      "coordinates": [[0, 0, 0]],
      "site_symmetry": "-3m"
    },
    "6c": {
      "multiplicity": 6,
      "coordinates": [[0, 0, "z"], [0, 0, "-z"]],
      "site_symmetry": "3m"
    }
  },
  "high_symmetry_points": {
    "Gamma": [0, 0, 0],
    "Z": [0.5, 0.5, 0.5],
    "F": [0.5, 0.5, 0],
    "L": [0.5, 0, 0]
  }
}
```

B Symmetry Indicator Formulas

For centrosymmetric space groups with time-reversal, the $\mathbb{Z}_2$ indicators are:

$$\nu_0 = \sum_{\text{TRIM}} n_-(\mathbf{k}) \mod 2 \tag{12}$$

$$\nu_i = \sum_{\mathbf{k}:k_i=\pi} n_-(\mathbf{k}) \mod 2 \tag{13}$$

where $n_-(\mathbf{k})$ counts bands with negative parity at TRIM $\mathbf{k}$.