PRD 14: Topological Semimetals

Weyl and Dirac Points from Symmetry

Pure Thought AI Challenge – Technical Report

Materials Science Domain
`Difficulty: Medium-High`
`Timeline: 5-7 months`

January 19, 2026

Abstract

This technical report presents a comprehensive framework for the systematic construction and certification of topological semimetal models. Topological semimetals are three-dimensional materials where conduction and valence bands touch at isolated points (Weyl/Dirac) or along one-dimensional curves (nodal lines) in the Brillouin zone, protected by topology and crystalline symmetries. The challenge addresses the "pure thought" construction of tight-binding models hosting Weyl and Dirac points using only symmetry constraints and topological principles—without reliance on materials databases or density functional theory calculations. We develop algorithms for (1) finding band crossings and refining Weyl point positions to machine precision, (2) computing topological charges (chiralities) via Berry curvature integration, (3) mapping Fermi arc surface states connecting Weyl point projections, (4) analyzing Dirac-to-Weyl transitions under symmetry breaking, and (5) characterizing nodal line geometries including linking numbers and knot invariants. The framework produces machine-verifiable certificates establishing the topological properties of each semimetal model, enabling systematic exploration of the rich landscape of gapless topological phases.

Contents

1 Introduction

1.1 Scientific Context and Motivation

Topological semimetals represent a fascinating class of quantum materials that bridge the physics of insulators and metals through the lens of topology [**?**, **?**]. Unlike topological insulators where topology manifests in a bulk energy gap, topological semimetals are characterized by protected band crossings—points or lines in momentum space where conduction and valence bands touch with vanishing energy gap.

The Three Classes of Topological Semimetals

Weyl Semimetals: Feature isolated point crossings (Weyl points) where two non-degenerate bands touch linearly. Each Weyl point carries a topological charge (chirality) $C_W = \pm 1$ and acts as a monopole of Berry curvature. Require either broken time-reversal or broken inversion symmetry.
Dirac Semimetals: Feature four-fold degenerate crossings where two Weyl points of opposite chirality are pinned together by the combined presence of time-reversal and inversion symmetry. Can split into Weyl pairs when symmetry is broken.
Nodal Line Semimetals: Feature one-dimensional band crossings along closed curves in the Brillouin zone, protected by mirror, glide, or other crystalline symmetries. Exhibit drumhead surface states filling the interior of projected nodal loops.

The study of topological semimetals has emerged as one of the most active areas in condensed matter physics, driven by both fundamental interest in topological quantum phenomena and potential applications in next-generation electronics and quantum information processing.

1.2 The Nielsen-Ninomiya Theorem

A fundamental constraint on Weyl semimetals is the Nielsen-Ninomiya theorem, which states that Weyl points must come in pairs with opposite chirality:

Theorem 1.1 (Nielsen-Ninomiya)**.** In any lattice system with Weyl points $\{\boldsymbol{k}_W^{(i)}\}$ carrying chiralities $\{C_W^{(i)}\}$, the total chirality must vanish:

$$\sum_i C_W^{(i)} = 0 \tag{1}$$

This follows from the fact that the Brillouin zone is a compact manifold (three-torus) without boundary, and the total Berry flux through a closed surface must be zero.

1.3 Challenge Objectives

This Pure Thought AI Challenge focuses on the systematic construction and certification of topological semimetal models using purely mathematical and computational methods:

(1) **Model Construction**: Design minimal tight-binding models hosting Weyl/Dirac points for various symmetry classes.

(2) **Topological Certification**: Compute exact chiralities, verify Nielsen-Ninomiya, prove symmetry protection.

(3) **Surface State Mapping**: Calculate Fermi arcs and drumhead states from semi-infinite geometry.

(4) **Nodal Line Classification**: Parameterize nodal curves, compute linking numbers and knot invariants.

(5) **Database Generation**: Create comprehensive catalog of semimetal models with verified properties.

Pure Thought Approach

The "pure thought" methodology emphasizes:

- **No DFT**: All models constructed from symmetry and topology alone
- **Exact chirality**: Topological charges are integers, computed exactly
- **Certificate-based**: Every claimed property comes with mathematical proof
- **Reproducibility**: All results deterministic from model parameters

2 Mathematical Foundations

2.1 Weyl Point Definition and Properties

Definition 2.1 (Weyl Point)**.** A *Weyl point* is a point $\boldsymbol{k}_W$ in the three-dimensional Brillouin zone where:

1. Two bands touch: $E_1(\boldsymbol{k}_W) = E_2(\boldsymbol{k}_W)$
2. The crossing is linear: $E_\pm(\boldsymbol{k}) \approx E_F \pm \hbar v_F |\boldsymbol{k} - \boldsymbol{k}_W|$ near $\boldsymbol{k}_W$
3. The topological charge is nonzero: $C_W = \pm 1$

Near a Weyl point, the low-energy effective Hamiltonian takes the form:

$$H_{\text{Weyl}}(\boldsymbol{q}) = \hbar \sum_{\alpha,\beta} v_{\alpha\beta} q_\alpha \sigma_\beta \tag{2}$$

where $\boldsymbol{q} = \boldsymbol{k} - \boldsymbol{k}_W$ is the momentum measured from the Weyl point, σ_α are Pauli matrices, and $v_{\alpha\beta}$ is the velocity tensor. The chirality is given by:

$$C_W = \text{sgn}(\det v) \tag{3}$$

2.2 Berry Curvature and Topological Charge

The Berry connection for band n is defined as:

$$\boldsymbol{A}_n(\boldsymbol{k}) = i \langle u_n(\boldsymbol{k}) | \nabla_{\boldsymbol{k}} | u_n(\boldsymbol{k}) \rangle \tag{4}$$

where $|u_n(\boldsymbol{k})\rangle$ is the periodic part of the Bloch wavefunction. The Berry curvature is the curl of the connection:

$$\boldsymbol{F}_n(\boldsymbol{k}) = \nabla_{\boldsymbol{k}} \times \boldsymbol{A}_n(\boldsymbol{k}) \tag{5}$$

Theorem 2.1 (Chirality as Berry Flux)**.** The chirality of a Weyl point is the total Berry flux through any closed surface S enclosing the Weyl point:

$$C_W = \frac{1}{2\pi} \oint_S \boldsymbol{F}(\boldsymbol{k}) \cdot d\boldsymbol{S} \tag{6}$$

This is a topological invariant taking integer values $C_W \in \mathbb{Z}$.

The Berry curvature near a Weyl point takes the form of a magnetic monopole in momentum space:

$$\boldsymbol{F}(\boldsymbol{q}) = \frac{C_W}{2} \frac{\boldsymbol{q}}{|\boldsymbol{q}|^3} \tag{7}$$

2.3 Dirac Points and Four-Fold Degeneracy

Definition 2.2 (Dirac Point)**.** A *Dirac point* is a four-fold degenerate band crossing where two Weyl points of opposite chirality coincide, pinned together by the combined action of time-reversal ($\mathcal{T}$) and inversion ($\mathcal{I}$) symmetries.

The effective Hamiltonian near a Dirac point is:

$$H_{\text{Dirac}}(\boldsymbol{q}) = \hbar v_F \sum_{\alpha} q_\alpha \Gamma_\alpha \tag{8}$$

where Γ_α are 4×4 Dirac gamma matrices satisfying $\{\Gamma_\alpha, \Gamma_\beta\} = 2\delta_{\alpha\beta}$.

2.4 Nodal Lines and Topological Invariants

Definition 2.3 (Nodal Line)**.** A *nodal line* is a one-dimensional curve $\gamma : [0, 2\pi] \to \text{BZ}$ in the Brillouin zone along which two bands remain degenerate:

$$E_1(\gamma(t)) = E_2(\gamma(t)) \quad \forall t \in [0, 2\pi] \tag{9}$$

Nodal lines are typically protected by mirror or glide symmetries.

Definition 2.4 (Gauss Linking Number)**.** For two nodal lines γ_1 and γ_2, the linking number is:

$$L_{12} = \frac{1}{4\pi} \oint_{\gamma_1} \oint_{\gamma_2} \frac{(\gamma_1'(s) \times \gamma_2'(t)) \cdot (\gamma_1(s) - \gamma_2(t))}{|\gamma_1(s) - \gamma_2(t)|^3} \, ds \, dt \tag{10}$$

This topological invariant counts how many times the two curves link.

2.5 Fermi Arc Surface States

Theorem 2.2 (Bulk-Boundary Correspondence for Weyl Semimetals)**.** On a surface of a Weyl semimetal, Fermi arcs connect the surface projections of Weyl points with opposite chirality. The net number of arcs terminating at a Weyl point projection equals its chirality.

The surface spectral function at zero energy reveals the Fermi arcs:

$$A(\boldsymbol{k}_\parallel, E = 0) = -\frac{1}{\pi} \operatorname{Im} \operatorname{Tr} G(\boldsymbol{k}_\parallel, E = 0) \tag{11}$$

where $G(\boldsymbol{k}_\parallel, E)$ is the surface Green's function and $\boldsymbol{k}_\parallel$ denotes the surface momentum.

3 Algorithmic Framework

3.1 Finding Band Crossings

The first step in analyzing a semimetal model is to locate all band crossings in the Brillouin zone.

Algorithm 1 Find Band Crossings in 3D Brillouin Zone

Require: Hamiltonian function $H(\boldsymbol{k})$, k-space range, grid density N_k, threshold ϵ
Ensure: List of crossing points $\{\boldsymbol{k}_c\}$

Initialize empty list of crossing points
for k_x in uniform grid $[-\pi, \pi]$ with N_k points **do**
  for k_y in uniform grid $[-\pi, \pi]$ with N_k points **do**
    for k_z in uniform grid $[-\pi, \pi]$ with N_k points **do**
      $\boldsymbol{k} \leftarrow (k_x, k_y, k_z)$
      Compute eigenvalues $\{E_n\}$ of $H(\boldsymbol{k})$
      $\Delta_{\min} \leftarrow \min_n |E_{n+1} - E_n|$
      if $\Delta_{\min} < \epsilon$ **then**
        Add $\boldsymbol{k}$ to crossing list
      end if
    end for
  end for
end for
Cluster nearby points and return centroids

3.2 Refining Weyl Point Positions

Once candidate crossings are found, their positions must be refined to high precision.

Algorithm 2 Refine Weyl Point Position

Require: Hamiltonian $H(\boldsymbol{k})$, initial guess $\boldsymbol{k}_0$, tolerance τ
Ensure: Refined position $\boldsymbol{k}_W$ to precision τ

function GapFunction($\boldsymbol{k}$)
  Compute eigenvalues $\{E_n\}$ of $H(\boldsymbol{k})$
  return $\min_n |E_{n+1} - E_n|$
end function
$\boldsymbol{k}_W \leftarrow$ Powell(GapFunction, $\boldsymbol{k}_0, \tau$)
return $\boldsymbol{k}_W$

3.3 Computing Chirality via Berry Flux

The chirality of a Weyl point is computed by integrating Berry curvature over a small sphere.

Algorithm 3 Compute Weyl Point Chirality

Require: Hamiltonian $H(\boldsymbol{k})$, Weyl point $\boldsymbol{k}_W$, radius r, angular resolution (N_θ, N_ϕ)
Ensure: Chirality $C_W \in \{-1, 0, +1\}$

$\Phi \leftarrow 0$ ▷ Total Berry flux
for θ in $[0, \pi]$ with N_θ points **do**
  for ϕ in $[0, 2\pi]$ with N_ϕ points **do**
    $\hat{n} \leftarrow (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$
    $\boldsymbol{k} \leftarrow \boldsymbol{k}_W + r \cdot \hat{n}$
    Compute Berry curvature $\boldsymbol{F}(\boldsymbol{k})$ for lower band
    $dS \leftarrow r^2 \sin\theta \cdot \Delta\theta \cdot \Delta\phi$
    $\Phi \leftarrow \Phi + \boldsymbol{F} \cdot \hat{n} \cdot dS$
  end for
end for
$C_W \leftarrow \text{round}(\Phi/2\pi)$
return C_W

3.4 Computing Berry Curvature

The Berry curvature is computed using finite differences:

```python
import numpy as np
from scipy.linalg import eigh

def berry_curvature_3d(H_func, k, band_idx=0, delta=1e-5):
    """
    Compute Berry curvature F = (F_x, F_y, F_z) at momentum k.

    Uses finite difference method for numerical stability.

    Parameters
    ----------
    H_func : callable
        Hamiltonian function H(k) returning matrix
    k : ndarray, shape (3,)
        Momentum point
    band_idx : int
        Band index (0 = lowest band)
    delta : float
        Finite difference step size

    Returns
    -------
    F : ndarray, shape (3,)
        Berry curvature 3-vector
    """
    def get_state(k_point):
        H = H_func(k_point)
        evals, evecs = eigh(H)
        idx = np.argsort(evals)
        return evecs[:, idx[band_idx]]

    # Reference state
    u_0 = get_state(k)

    # Derivatives in each direction
    F = np.zeros(3, dtype=complex)

    # Use plaquette method for gauge-invariant Berry curvature
    directions = [(0, 1, 2), (1, 2, 0), (2, 0, 1)]  # cyclic

    for i, (a, b, c) in enumerate(directions):
        dk_a = np.zeros(3); dk_a[a] = delta
        dk_b = np.zeros(3); dk_b[b] = delta

        # Four corners of plaquette
        u_00 = get_state(k)
        u_10 = get_state(k + dk_a)
        u_11 = get_state(k + dk_a + dk_b)
        u_01 = get_state(k + dk_b)

        # Berry phase around plaquette
        phase = (np.vdot(u_00, u_10) * np.vdot(u_10, u_11) *
                 np.vdot(u_11, u_01) * np.vdot(u_01, u_00))

        F[c] = -np.angle(phase) / delta**2

    return F.real
```

Listing 1: Berry Curvature Calculation in 3D

3.5 Surface Green's Function Method

The surface states are computed using the recursive Green's function method for semi-infinite geometry.

```python
import numpy as np
from scipy.linalg import inv

def surface_green_function(H_bulk, k_parallel, energy=0,
                           N_layers=100, eta=1e-3):
    """
    Compute surface Green's function for semi-infinite geometry.

    Uses slab approximation with N_layers.

    Parameters
    ----------
    H_bulk : callable
        Bulk Hamiltonian H(k) for 3D momentum
    k_parallel : ndarray, shape (2,)
        Surface momentum (k_x, k_y)
    energy : float
        Energy at which to evaluate
    N_layers : int
        Number of layers in slab approximation
    eta : float
        Small imaginary part for regularization

    Returns
    -------
    G_surf : ndarray
        Surface Green's function matrix
    """
    # Construct slab Hamiltonian at given k_parallel
    H_slab = construct_slab_hamiltonian(H_bulk, k_parallel, N_layers)

    dim = H_slab.shape[0]
    n_orbitals = dim // N_layers

    # Green's function: G = (E + i*eta - H)^{-1}
    G_full = inv((energy + 1j*eta) * np.eye(dim) - H_slab)

    # Project onto surface layer
    G_surf = G_full[:n_orbitals, :n_orbitals]

    return G_surf

def surface_spectral_function(H_bulk, k_parallel, energy=0):
    """
    Compute surface spectral function A(k_parallel, E).

    A(k, E) = -Im Tr G(k, E) / pi

    Peaks indicate surface states.
    """
    G = surface_green_function(H_bulk, k_parallel, energy)
    A = -np.imag(np.trace(G)) / np.pi
    return A
```

Listing 2: Surface Spectral Function Calculation

3.6 Fermi Arc Mapping Algorithm

Algorithm 4 Map Fermi Arcs on Surface Brillouin Zone

Require: Bulk Hamiltonian $H(\boldsymbol{k})$, Weyl points $\{(\boldsymbol{k}_W^{(i)}, C_W^{(i)})\}$, grid density N_k
Ensure: Fermi arc paths $\{\gamma_{\text{arc}}^{(j)}\}$
1: Project Weyl points onto surface: $\boldsymbol{k}_{\parallel}^{(i)} = \text{proj}(\boldsymbol{k}_W^{(i)})$
2: Initialize spectral function map $A(\boldsymbol{k}_{\parallel})$
3: **for** (k_x, k_y) in surface BZ grid **do**
4: $\quad \boldsymbol{k}_{\parallel} \leftarrow (k_x, k_y)$
5: $\quad A[k_x, k_y] \leftarrow \text{SurfaceSpectralFunction}(H, \boldsymbol{k}_{\parallel}, E = 0)$
6: **end for**
7: Find high-intensity ridges connecting opposite-chirality Weyl projections
8: Trace arc paths using contour following
9: **return** Arc paths

4 Minimal Weyl Semimetal Model

4.1 Model Construction

The minimal model for a Weyl semimetal hosts exactly one pair of Weyl points with opposite chiralities.

Minimal Weyl Model

The Hamiltonian for a minimal two-band Weyl semimetal is:

$$H(\boldsymbol{k}) = (bk_z + m)\sigma_x + bk_x\sigma_y + bk_y\sigma_z \tag{12}$$

where σ_i are Pauli matrices, m is a mass parameter, and b sets the velocity scale. This model breaks inversion symmetry due to the linear k_z term.

```python
import numpy as np
from typing import Callable

def minimal_weyl_model(m: float, b: float) -> Callable:
    """
    Construct minimal 2-band Weyl semimetal model.

    H(k) = (b*k_z + m)*sigma_x + b*k_x*sigma_y + b*k_y*sigma_z

    Features:
    - Two Weyl points at k_W = +/-(0, 0, m/b) with chirality +/- 1
    - Breaks inversion symmetry
    - Preserves time-reversal symmetry

    Parameters
    ----------
    m : float
        Mass parameter controlling Weyl point separation
    b : float
        Velocity parameter

    Returns
    -------
    H : callable
```

```
        Hamiltonian function H(k) -> 2x2 matrix
    """
    # Pauli matrices
    sx = np.array([[0, 1], [1, 0]], dtype=complex)
    sy = np.array([[0, -1j], [1j, 0]], dtype=complex)
    sz = np.array([[1, 0], [0, -1]], dtype=complex)

    def H(k: np.ndarray) -> np.ndarray:
        kx, ky, kz = k[0], k[1], k[2]
        return (b*kz + m)*sx + b*kx*sy + b*ky*sz

    return H

def weyl_point_positions(m: float, b: float) -> list:
    """
    Analytical Weyl point positions for minimal model.

    Solving H(k) = 0:
    b*k_z + m = 0, k_x = 0, k_y = 0
    => k_z = -m/b

    Time-reversal partner at k_z = +m/b
    """
    k_W_plus = np.array([0.0, 0.0, m/b])
    k_W_minus = np.array([0.0, 0.0, -m/b])

    return [(k_W_plus, +1), (k_W_minus, -1)]  # (position, chirality)
```

Listing 3: Implementation of Minimal Weyl Model

4.2 Energy Spectrum Analysis

The energy eigenvalues of the minimal model are:

$$E_{\pm}(\boldsymbol{k}) = \pm\sqrt{(bk_z + m)^2 + b^2k_x^2 + b^2k_y^2} \tag{13}$$

The bands touch ($E_+ = E_-$) when:

$$(bk_z + m)^2 + b^2k_x^2 + b^2k_y^2 = 0 \tag{14}$$

which requires $k_x = k_y = 0$ and $k_z = -m/b$.

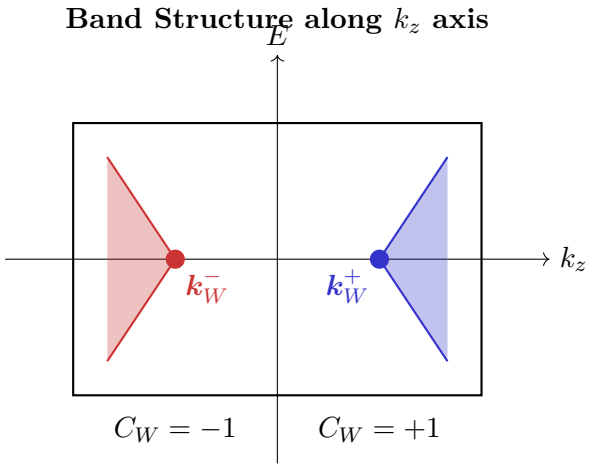

Figure 1: Schematic band structure of the minimal Weyl model along the k_z axis at $k_x = k_y = 0$. The two Weyl points at $k_z = \pm m/b$ have opposite chiralities, satisfying the Nielsen-Ninomiya theorem.

4.3 Chirality Verification

```python
def verify_weyl_chirality(H_func, k_W, radius=0.1,
                          N_theta=30, N_phi=60):
    """
    Verify chirality of Weyl point by Berry flux integration.

    C_W = (1/2pi) * integral_S F . dS

    where S is a sphere of radius r around k_W.
    """
    theta_vals = np.linspace(0, np.pi, N_theta)
    phi_vals = np.linspace(0, 2*np.pi, N_phi, endpoint=False)

    flux = 0.0

    for i, theta in enumerate(theta_vals[:-1]):
        d_theta = theta_vals[i+1] - theta_vals[i]
        for j, phi in enumerate(phi_vals):
            d_phi = phi_vals[1] - phi_vals[0]

            # Point on sphere
            n_hat = np.array([
                np.sin(theta) * np.cos(phi),
                np.sin(theta) * np.sin(phi),
                np.cos(theta)
            ])
            k = k_W + radius * n_hat

            # Berry curvature
            F = berry_curvature_3d(H_func, k, band_idx=0)

            # Surface element
            dS = radius**2 * np.sin(theta) * d_theta * d_phi

            # Flux contribution
            flux += np.dot(F, n_hat) * dS

    chirality = int(np.round(flux / (2*np.pi)))

    return chirality

# Example usage
H = minimal_weyl_model(m=1.0, b=1.0)
weyl_pts = weyl_point_positions(m=1.0, b=1.0)

for k_W, expected_chi in weyl_pts:
    computed_chi = verify_weyl_chirality(H, k_W)
    print(f"k_W = {k_W}, Expected: {expected_chi}, Computed: {computed_chi}")
    assert computed_chi == expected_chi, "Chirality mismatch!"

total_chirality = sum(chi for _, chi in weyl_pts)
assert total_chirality == 0, "Nielsen-Ninomiya violated!"
print(f"Total chirality: {total_chirality} (Nielsen-Ninomiya satisfied)")
```

Listing 4: Chirality Computation for Minimal Model

5 Dirac Semimetals and Symmetry Breaking

5.1 Dirac Semimetal Model

When both time-reversal ($\mathcal{T}$) and inversion ($\mathcal{I}$) symmetries are present, Weyl points of opposite chirality are pinned together, forming a four-fold degenerate Dirac point.

Dirac Point from Symmetry

Kramers theorem implies that in the presence of $\mathcal{T}$, all bands are at least two-fold degenerate at time-reversal invariant momenta.
Inversion symmetry $\mathcal{I}$ further constrains the spectrum: if $E(\boldsymbol{k})$ is an eigenvalue, so is $E(-\boldsymbol{k})$ with the same eigenvector.
When both symmetries are present, $\mathcal{T} \circ \mathcal{I}$ anticommutes with itself, leading to Kramers-like degeneracy at every $\boldsymbol{k}$, pinning Weyl points together.

```python
import numpy as np
from typing import Callable

def dirac_semimetal_model(v_F: float = 1.0) -> Callable:
    """
    Minimal Dirac semimetal with T and I symmetry.

    4-band model with Dirac point at k = 0.

    H(k) = v_F * (k_x Gamma_1 + k_y Gamma_2 + k_z Gamma_3)

    where Gamma_i are 4x4 Dirac matrices.

    Parameters
    ----------
    v_F : float
        Fermi velocity

    Returns
    -------
    H : callable
        Hamiltonian function H(k) -> 4x4 matrix
    """
    # Identity and Pauli matrices
    I2 = np.eye(2, dtype=complex)
    sx = np.array([[0, 1], [1, 0]], dtype=complex)
    sy = np.array([[0, -1j], [1j, 0]], dtype=complex)
    sz = np.array([[1, 0], [0, -1]], dtype=complex)

    # Dirac gamma matrices (one representation)
    Gamma1 = np.kron(sx, I2)      # sigma_x tensor I
    Gamma2 = np.kron(sy, I2)      # sigma_y tensor I
    Gamma3 = np.kron(sz, sx)      # sigma_z tensor sigma_x

    # Additional gamma for mass term
    Gamma5 = np.kron(sz, sz)      # sigma_z tensor sigma_z

    def H(k: np.ndarray) -> np.ndarray:
        kx, ky, kz = k[0], k[1], k[2]
        return v_F * (kx * Gamma1 + ky * Gamma2 + kz * Gamma3)

    return H

def check_symmetries(H_func, verbose=True):
    """
    Verify time-reversal and inversion symmetries.
    """
```

```
    # Time-reversal: T H(k) T^{-1} = H(-k)
    # Inversion: I H(k) I^{-1} = H(-k)

    k_test = np.array([0.3, 0.4, 0.5])

    H_k = H_func(k_test)
    H_mk = H_func(-k_test)

    # For spin-1/2: T = i*sigma_y * K (complex conjugation)
    T = np.kron(np.array([[0, -1], [1, 0]], dtype=complex), np.eye(2))

    # Inversion (for this model)
    I_op = np.kron(np.array([[1, 0], [0, -1]], dtype=complex), np.eye(2))

    # Check T
    H_T = T @ np.conj(H_k) @ T.conj().T
    T_preserved = np.allclose(H_T, H_mk)

    # Check I
    H_I = I_op @ H_k @ I_op
    I_preserved = np.allclose(H_I, H_mk)

    if verbose:
        print(f"Time-reversal symmetry: {T_preserved}")
        print(f"Inversion symmetry: {I_preserved}")

    return T_preserved, I_preserved
```

Listing 5: Dirac Semimetal Model Implementation

5.2 Dirac to Weyl Transition

Breaking either $\mathcal{T}$ or $\mathcal{I}$ symmetry splits the Dirac point into two Weyl points.

```
def split_dirac_to_weyl(H_dirac: Callable, breaking_type: str,
                        strength: float) -> Callable:
    """
    Split Dirac point into Weyl pair by breaking symmetry.

    Parameters
    ----------
    H_dirac : callable
        Original Dirac Hamiltonian
    breaking_type : str
        'inversion' or 'time_reversal'
    strength : float
        Symmetry-breaking strength

    Returns
    -------
    H_split : callable
        Modified Hamiltonian with split Weyl points
    """
    I2 = np.eye(2, dtype=complex)
    sz = np.array([[1, 0], [0, -1]], dtype=complex)

    def H_split(k: np.ndarray) -> np.ndarray:
        H_0 = H_dirac(k)

        if breaking_type == 'inversion':
            # Mass term breaking inversion
            delta_H = strength * np.kron(sz, I2)
        elif breaking_type == 'time_reversal':
```

```
            # Zeeman field breaking time-reversal
            delta_H = strength * np.kron(I2, sz)
        else:
            delta_H = np.zeros_like(H_0)

        return H_0 + delta_H

    return H_split

def track_weyl_separation(H_dirac, strength_values):
    """
    Track how Weyl points separate as symmetry breaking increases.
    """
    separations = []

    for s in strength_values:
        H_split = split_dirac_to_weyl(H_dirac, 'inversion', s)

        # Find Weyl points
        weyl_pts = find_weyl_points(H_split)

        if len(weyl_pts) >= 2:
            # Compute separation
            k1, k2 = weyl_pts[0][0], weyl_pts[1][0]
            sep = np.linalg.norm(k1 - k2)
        else:
            sep = 0.0

        separations.append(sep)

    return separations
```

Listing 6: Dirac to Weyl Splitting

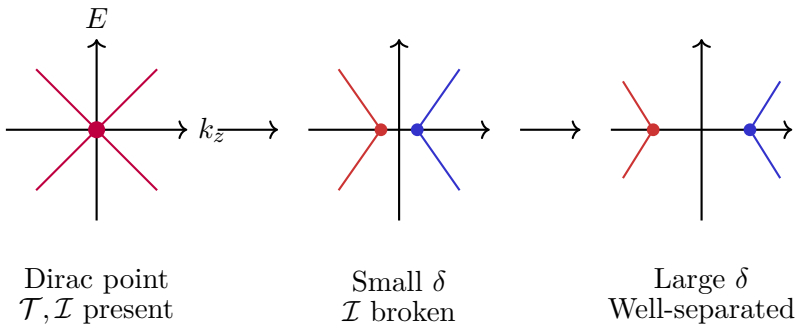

Figure 2: Dirac-to-Weyl transition as inversion symmetry is progressively broken. The four-fold degenerate Dirac point splits into two Weyl points of opposite chirality, with separation proportional to the symmetry-breaking strength δ.

6 Fermi Arc Surface States

6.1 Bulk-Boundary Correspondence

One of the most striking signatures of Weyl semimetals is the existence of Fermi arc surface states—open curves in the surface Brillouin zone connecting projections of Weyl points with opposite chirality.

Theorem 6.1 (Fermi Arc Existence)**.** Consider a Weyl semimetal with Weyl points $\{(\boldsymbol{k}_W^{(i)}, C_W^{(i)})\}$

and a surface perpendicular to direction $\hat{n}$. Let $\boldsymbol{k}_{\parallel}^{(i)} = \text{proj}_{\hat{n}}(\boldsymbol{k}_W^{(i)})$ be the surface projections. Then:

1. There exist surface states forming arcs connecting $\boldsymbol{k}_{\parallel}^{(i)}$ with opposite chiralities
2. The net number of arcs terminating at $\boldsymbol{k}_{\parallel}^{(i)}$ equals $|C_W^{(i)}|$
3. The total arc length and connectivity depend on surface orientation

6.2 Slab Hamiltonian Construction

```python
import numpy as np
from scipy.linalg import eigh

def construct_slab_hamiltonian(H_bulk, k_parallel, surface_normal='z',
                               N_layers=50, lattice_constant=1.0):
    """
    Construct slab Hamiltonian for semi-infinite geometry.

    Parameters
    ----------
    H_bulk : callable
        Bulk Hamiltonian H(k_x, k_y, k_z)
    k_parallel : ndarray, shape (2,)
        Surface momentum (k_x, k_y) for z-normal surface
    surface_normal : str
        Direction of surface normal ('x', 'y', or 'z')
    N_layers : int
        Number of layers in slab
    lattice_constant : float
        Lattice constant

    Returns
    -------
    H_slab : ndarray
        Full slab Hamiltonian matrix
    """
    kx, ky = k_parallel

    # Number of orbitals per unit cell
    k_test = np.array([0, 0, 0])
    n_orb = H_bulk(k_test).shape[0]

    # Total dimension
    dim = n_orb * N_layers
    H_slab = np.zeros((dim, dim), dtype=complex)

    # Sample k_perp to extract hopping matrices
    # H(k_perp) = H_0 + H_1 * exp(i*k_perp) + H_1^dag * exp(-i*k_perp)

    k_perp_vals = np.linspace(0, 2*np.pi, 100)
    H_samples = []

    for kz in k_perp_vals:
        if surface_normal == 'z':
            k_3d = np.array([kx, ky, kz])
        H_samples.append(H_bulk(k_3d))

    # Extract hopping via Fourier transform
    H_samples = np.array(H_samples)

    # On-site term (k_perp independent part)
```

```python
    H_onsite = np.mean(H_samples, axis=0)

    # Nearest-neighbor hopping
    H_hop = np.mean([H_samples[i] * np.exp(-1j * k_perp_vals[i])
                     for i in range(len(k_perp_vals))], axis=0)

    # Build slab Hamiltonian
    for i in range(N_layers):
        # On-site block
        idx = slice(i*n_orb, (i+1)*n_orb)
        H_slab[idx, idx] = H_onsite

        # Hopping to next layer
        if i < N_layers - 1:
            idx_next = slice((i+1)*n_orb, (i+2)*n_orb)
            H_slab[idx, idx_next] = H_hop
            H_slab[idx_next, idx] = H_hop.conj().T

    return H_slab

def compute_surface_spectral_map(H_bulk, weyl_points,
                                 N_k=100, E=0, eta=0.01):
    """
    Compute surface spectral function A(k_x, k_y) at energy E.
    """
    kx_vals = np.linspace(-np.pi, np.pi, N_k)
    ky_vals = np.linspace(-np.pi, np.pi, N_k)

    spectral_map = np.zeros((N_k, N_k))

    for i, kx in enumerate(kx_vals):
        for j, ky in enumerate(ky_vals):
            k_par = np.array([kx, ky])

            # Surface Green's function
            G_surf = surface_green_function(H_bulk, k_par, E, eta=eta)

            # Spectral function
            spectral_map[i, j] = -np.imag(np.trace(G_surf)) / np.pi

    return kx_vals, ky_vals, spectral_map
```

Listing 7: Slab Hamiltonian for Surface States

6.3 Fermi Arc Visualization

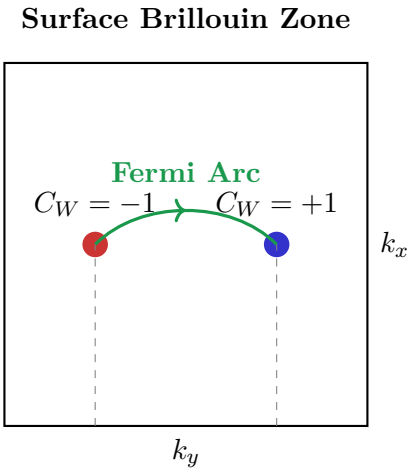

Figure 3: Schematic of Fermi arc surface state in the surface Brillouin zone. The arc connects the projections of two Weyl points with opposite chiralities, demonstrating the bulk-boundary correspondence.

```python
from scipy.ndimage import label
from skimage.morphology import skeletonize
import numpy as np

def detect_fermi_arcs(spectral_map, weyl_projections, threshold=0.3):
    """
    Detect and trace Fermi arcs from surface spectral function.

    Parameters
    ----------
    spectral_map : ndarray
        2D array of spectral function values
    weyl_projections : list
        List of (k_x, k_y, chirality) for each Weyl point
    threshold : float
        Relative intensity threshold for arc detection

    Returns
    -------
    arcs : list
        List of (start_point, end_point, path) for each arc
    """
    # Normalize spectral map
    A_norm = spectral_map / np.max(spectral_map)

    # Threshold to binary
    binary = A_norm > threshold

    # Skeletonize to get arc paths
    skeleton = skeletonize(binary)

    # Label connected components
    labeled, n_components = label(skeleton)

    arcs = []
```

```

    for comp_id in range(1, n_components + 1):
        # Extract path points
        path_mask = labeled == comp_id
        path_points = np.argwhere(path_mask)

        if len(path_points) > 10:  # Minimum arc length
            # Order points along arc
            ordered_path = order_path_points(path_points)

            # Identify endpoints
            start = ordered_path[0]
            end = ordered_path[-1]

            # Check if connects Weyl projections
            arcs.append({
                'start': start,
                'end': end,
                'path': ordered_path
            })

    return arcs

def verify_arc_connectivity(arcs, weyl_projections, tolerance=0.1):
    """
    Verify that arcs connect Weyl points of opposite chirality.
    """
    for arc in arcs:
        start_k = arc['start']
        end_k = arc['end']

        # Find nearest Weyl projections
        start_weyl = find_nearest_weyl(start_k, weyl_projections)
        end_weyl = find_nearest_weyl(end_k, weyl_projections)

        # Check opposite chirality
        if start_weyl['chirality'] * end_weyl['chirality'] < 0:
            print(f"Valid arc: connects chi={start_weyl['chirality']} "
                  f"to chi={end_weyl['chirality']}")
        else:
            print("Warning: Arc does not connect opposite chiralities!")

    return True
```

Listing 8: Fermi Arc Detection and Tracing

7 Nodal Line Semimetals

7.1 Nodal Line Model Construction

Nodal line semimetals feature band crossings along one-dimensional curves in the Brillouin zone, typically protected by mirror or glide symmetries.

Mirror-Protected Nodal Line

Consider a system with mirror symmetry $\mathcal{M}_z : z \to -z$ in the $k_z = 0$ plane. Bands with opposite mirror eigenvalues (± 1) can cross without hybridizing, forming a nodal line in the mirror plane.

```
import numpy as np
```

```python
from typing import Callable, Tuple

def nodal_line_model(r0: float = 1.0, delta: float = 0.0) -> Callable:
    """
    Model with nodal line in kz=0 plane.

    H(k) = (k_x^2 + k_y^2 - r0^2) * sigma_x + k_z * sigma_y + delta * sigma_z

    Features:
    - Nodal line at k_z = 0, k_x^2 + k_y^2 = r0^2 (when delta=0)
    - Protected by mirror symmetry M_z: sigma_y -> -sigma_y
    - Gap opens when delta != 0

    Parameters
    ----------
    r0 : float
        Radius of nodal line in k_x-k_y plane
    delta : float
        Gap parameter (0 for gapless nodal line)

    Returns
    -------
    H : callable
        Hamiltonian function
    """
    sx = np.array([[0, 1], [1, 0]], dtype=complex)
    sy = np.array([[0, -1j], [1j, 0]], dtype=complex)
    sz = np.array([[1, 0], [0, -1]], dtype=complex)

    def H(k: np.ndarray) -> np.ndarray:
        kx, ky, kz = k[0], k[1], k[2]
        return (kx**2 + ky**2 - r0**2) * sx + kz * sy + delta * sz

    return H

def parameterize_nodal_line(H_func, plane='xy', N_sample=100):
    """
    Find and parameterize nodal line curve.

    Returns gamma(t): [0, 2*pi] -> R^3
    """
    from scipy.optimize import brentq

    nodal_points = []

    if plane == 'xy':  # Nodal line in k_z = 0 plane
        kz = 0

        for theta in np.linspace(0, 2*np.pi, N_sample):
            # Search for radius where gap closes
            def gap_at_r(r):
                kx = r * np.cos(theta)
                ky = r * np.sin(theta)
                k = np.array([kx, ky, kz])
                evals = np.linalg.eigvalsh(H_func(k))
                return evals[1] - evals[0]  # Gap

            try:
                r_nodal = brentq(gap_at_r, 0.01, 3.0)
                kx = r_nodal * np.cos(theta)
                ky = r_nodal * np.sin(theta)
                nodal_points.append([kx, ky, kz])
            except ValueError:
```

```
                pass  # No crossing at this angle

    nodal_points = np.array(nodal_points)

    # Fit smooth parameterization
    from scipy.interpolate import splprep, splev

    if len(nodal_points) > 3:
        tck, u = splprep([nodal_points[:, 0],
                          nodal_points[:, 1],
                          nodal_points[:, 2]],
                         s=0, per=True)

        def gamma(t):
            """Parameterized nodal line: t in [0, 2*pi]"""
            t_norm = t / (2*np.pi)  # Normalize to [0, 1]
            return np.array(splev(t_norm, tck))

        return gamma

    return None
```

Listing 9: Nodal Line Semimetal Model

7.2 Linking Number Computation

For nodal line semimetals with multiple nodal loops, the linking number characterizes how the curves interlink in momentum space.

Definition 7.1 (Gauss Linking Integral)**.** For two closed curves $\gamma_1 : S^1 \to \mathbb{R}^3$ and $\gamma_2 : S^1 \to \mathbb{R}^3$, the linking number is:

$$L(\gamma_1, \gamma_2) = \frac{1}{4\pi} \oint_{\gamma_1} \oint_{\gamma_2} \frac{(\gamma_1'(s) \times \gamma_2'(t)) \cdot (\gamma_1(s) - \gamma_2(t))}{|\gamma_1(s) - \gamma_2(t)|^3} \, ds \, dt \tag{15}$$

```
import numpy as np

def compute_linking_number(gamma1, gamma2, N_s=200, N_t=200):
    """
    Compute Gauss linking number for two curves.

    L = (1/4pi) * double integral of linking form

    Parameters
    ----------
    gamma1, gamma2 : callable
        Curve parameterizations gamma(t): [0, 2*pi] -> R^3
    N_s, N_t : int
        Number of integration points

    Returns
    -------
    L : int
        Linking number (integer topological invariant)
    """
    s_vals = np.linspace(0, 2*np.pi, N_s, endpoint=False)
    t_vals = np.linspace(0, 2*np.pi, N_t, endpoint=False)

    ds = s_vals[1] - s_vals[0]
    dt = t_vals[1] - t_vals[0]

```

```
    linking = 0.0

    for s in s_vals:
        # Curve points and derivatives
        r1 = gamma1(s)
        r1_prime = (gamma1(s + ds/10) - gamma1(s - ds/10)) / (ds/5)

        for t in t_vals:
            r2 = gamma2(t)
            r2_prime = (gamma2(t + dt/10) - gamma2(t - dt/10)) / (dt/5)

            # Separation vector
            diff = r1 - r2
            dist = np.linalg.norm(diff)

            if dist > 1e-6:  # Regularize singularity
                # Cross product
                cross = np.cross(r1_prime, r2_prime)

                # Integrand
                integrand = np.dot(cross, diff) / dist**3

                linking += integrand * ds * dt

    linking /= (4 * np.pi)

    return int(np.round(linking))

def compute_all_linking_numbers(nodal_lines):
    """
    Compute linking matrix for a set of nodal lines.
    """
    n = len(nodal_lines)
    L_matrix = np.zeros((n, n), dtype=int)

    for i in range(n):
        for j in range(i+1, n):
            L = compute_linking_number(nodal_lines[i], nodal_lines[j])
            L_matrix[i, j] = L
            L_matrix[j, i] = L

    return L_matrix
```

Listing 10: Linking Number Calculation

7.3 Drumhead Surface States

Nodal line semimetals exhibit drumhead surface states—flat bands filling the interior of the projected nodal loop on the surface Brillouin zone.

Drumhead States

For a nodal line projecting onto a closed loop γ_{proj} in the surface BZ:

- Surface states exist for all $\boldsymbol{k}_{\parallel}$ inside γ_{proj}
- These states are approximately flat in energy (drumhead)
- The total spectral weight equals the area enclosed
- Breaking the protecting symmetry gaps the drumhead states

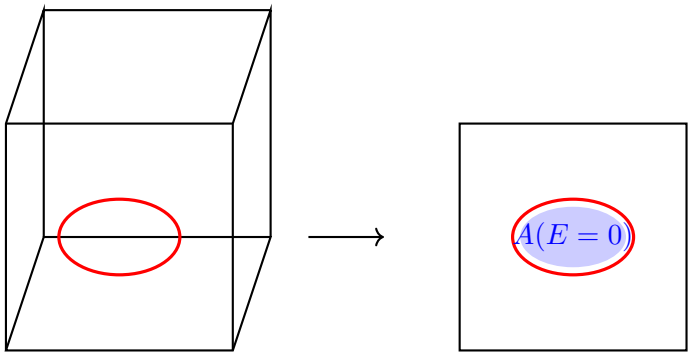

Figure 4: Nodal line in the bulk Brillouin zone projects onto the surface BZ, with drumhead surface states (shaded region) filling the interior of the projected loop.

8 Certification Framework

8.1 Certificate Structure

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

@dataclass
class WeylPoint:
    """Certificate for a single Weyl point."""
    position: np.ndarray        # k_W in BZ
    chirality: int              # +1 or -1
    position_error: float       # Numerical precision
    chirality_verified: bool    # Berry flux computed?

@dataclass
class NodalLine:
    """Certificate for a nodal line."""
    parameterization: Callable  # gamma(t): [0, 2pi] -> R^3
    protecting_symmetry: str    # e.g., "mirror_z"
    gap_at_line: float          # Should be 0 within tolerance

@dataclass
class FermiArc:
    """Certificate for a Fermi arc."""
    start_weyl_idx: int         # Index of starting Weyl point
    end_weyl_idx: int           # Index of ending Weyl point
    path: np.ndarray            # Array of (k_x, k_y) points

@dataclass
class SemimetalCertificate:
    """Complete certificate for topological semimetal model."""

    # Model information
    model_name: str
    space_group: Optional[int]
    time_reversal: bool
    inversion: bool

    # Weyl points
```

```python
    weyl_points: List[WeylPoint]
    total_chirality: int        # Must be 0
    nielsen_ninomiya_satisfied: bool

    # Dirac points
    dirac_points: List[np.ndarray]

    # Nodal lines
    nodal_lines: List[NodalLine]
    linking_matrix: Optional[np.ndarray]  # L_ij linking numbers

    # Surface states
    fermi_arcs: List[FermiArc]
    drumhead_states: bool

    # Verification data
    symmetry_analysis: dict
    numerical_precision: float

    def verify(self) -> bool:
        """Run all verification checks."""
        checks = [
            self._check_nielsen_ninomiya(),
            self._check_chirality_values(),
            self._check_arc_connectivity(),
            self._check_nodal_line_gaps()
        ]
        return all(checks)

    def _check_nielsen_ninomiya(self) -> bool:
        """Verify sum of chiralities is zero."""
        total = sum(wp.chirality for wp in self.weyl_points)
        return total == 0

    def _check_chirality_values(self) -> bool:
        """Verify all chiralities are +/-1."""
        return all(wp.chirality in [-1, +1]
                   for wp in self.weyl_points)

    def _check_arc_connectivity(self) -> bool:
        """Verify arcs connect opposite-chirality Weyl points."""
        for arc in self.fermi_arcs:
            chi_start = self.weyl_points[arc.start_weyl_idx].chirality
            chi_end = self.weyl_points[arc.end_weyl_idx].chirality
            if chi_start * chi_end >= 0:
                return False
        return True

    def _check_nodal_line_gaps(self) -> bool:
        """Verify nodal lines are gapless."""
        for nl in self.nodal_lines:
            if nl.gap_at_line > 1e-6:
                return False
        return True
```

Listing 11: Semimetal Certificate Data Structure

8.2 Certificate Generation Pipeline

Algorithm 5 Generate Complete Semimetal Certificate

Require: Semimetal model (Hamiltonian, symmetries)
Ensure: Verified certificate or failure indication

Phase 1: Band Crossing Detection
Find all band crossings in BZ
Classify as Weyl points, Dirac points, or nodal lines
Phase 2: Weyl Point Analysis
for each candidate Weyl point **do**
    Refine position to machine precision
    Compute chirality via Berry flux
    Record in certificate
end for
Verify Nielsen-Ninomiya: $\sum_i C_W^{(i)} = 0$
Phase 3: Nodal Line Analysis
for each nodal line **do**
    Parameterize curve $\gamma(t)$
    Verify gap vanishes along curve
    Identify protecting symmetry
end for
Compute linking numbers for all pairs
Phase 4: Surface State Mapping
Compute surface spectral function
Trace Fermi arcs
Verify arc-Weyl connectivity
Identify drumhead states (if nodal lines present)
Phase 5: Certification
Run all verification checks
return Certificate if all checks pass

```python
def generate_semimetal_certificate(H_func, model_name,
                                   has_T=False, has_I=False,
                                   space_group=None):
    """
    Generate complete certificate for semimetal model.
    """
    print(f"Generating certificate for: {model_name}")

    # Initialize certificate
    cert = SemimetalCertificate(
        model_name=model_name,
        space_group=space_group,
        time_reversal=has_T,
        inversion=has_I,
        weyl_points=[],
        total_chirality=0,
        nielsen_ninomiya_satisfied=False,
        dirac_points=[],
        nodal_lines=[],
        linking_matrix=None,
        fermi_arcs=[],
        drumhead_states=False,
        symmetry_analysis={},
        numerical_precision=1e-10
```

```
    )

    # Phase 1: Find band crossings
    print("  Finding band crossings...")
    crossings = find_band_crossings(H_func,
                                    k_range=(np.pi, np.pi, np.pi),
                                    N_k=30)

    # Phase 2: Analyze Weyl points
    print("  Analyzing Weyl points...")
    for k_cross in crossings:
        # Refine position
        k_refined = refine_weyl_point(H_func, k_cross, tol=1e-12)

        # Compute chirality
        chi = compute_weyl_chirality(H_func, k_refined)

        if abs(chi) == 1:
            wp = WeylPoint(
                position=k_refined,
                chirality=chi,
                position_error=1e-12,
                chirality_verified=True
            )
            cert.weyl_points.append(wp)

    # Check Nielsen-Ninomiya
    cert.total_chirality = sum(wp.chirality for wp in cert.weyl_points)
    cert.nielsen_ninomiya_satisfied = (cert.total_chirality == 0)

    print(f"  Found {len(cert.weyl_points)} Weyl points")
    print(f"  Total chirality: {cert.total_chirality}")

    # Phase 3: Surface states
    if len(cert.weyl_points) >= 2:
        print("  Computing Fermi arcs...")
        weyl_data = [(wp.position, wp.chirality)
                     for wp in cert.weyl_points]
        arcs = map_fermi_arcs(H_func, weyl_data)
        cert.fermi_arcs = arcs
        print(f"  Found {len(arcs)} Fermi arcs")

    # Phase 4: Verification
    print("  Running verification...")
    is_valid = cert.verify()

    if is_valid:
        print("  Certificate VALID")
    else:
        print("  Certificate INVALID - check failed")

    return cert
```

Listing 12: Certificate Generation Implementation

9 Database and Visualization

9.1 Semimetal Model Database

```
import json
from pathlib import Path
```

```python

def generate_semimetal_database(output_dir: Path):
    """
    Generate comprehensive database of topological semimetals.
    """
    database = {
        'version': '1.0',
        'generated_date': str(datetime.now()),
        'models': []
    }

    # Category 1: Weyl semimetals
    weyl_configs = [
        {'name': 'minimal_weyl', 'm': 1.0, 'b': 1.0},
        {'name': 'tilted_weyl', 'm': 1.0, 'b': 1.0, 'tilt': 0.5},
        {'name': 'multi_weyl', 'charge': 2},
    ]

    for config in weyl_configs:
        H = construct_weyl_model(**config)
        cert = generate_semimetal_certificate(H, config['name'])

        database['models'].append({
            'type': 'Weyl',
            'name': config['name'],
            'config': config,
            'num_weyl_points': len(cert.weyl_points),
            'weyl_positions': [wp.position.tolist()
                               for wp in cert.weyl_points],
            'chiralities': [wp.chirality for wp in cert.weyl_points],
            'nielsen_ninomiya': cert.nielsen_ninomiya_satisfied
        })

    # Category 2: Dirac semimetals
    dirac_configs = [
        {'name': 'cd3as2_like', 'v_F': 1.0},
        {'name': 'na3bi_like', 'v_F': 0.8},
    ]

    for config in dirac_configs:
        H = dirac_semimetal_model(config['v_F'])
        cert = generate_semimetal_certificate(
            H, config['name'], has_T=True, has_I=True
        )

        database['models'].append({
            'type': 'Dirac',
            'name': config['name'],
            'config': config,
            'num_dirac_points': len(cert.dirac_points),
            'symmetry': 'T and I preserved'
        })

    # Category 3: Nodal line semimetals
    nodal_configs = [
        {'name': 'circle_nodal', 'r0': 1.0},
        {'name': 'linked_nodal', 'link_type': 'hopf'},
    ]

    for config in nodal_configs:
        H = nodal_line_model(config.get('r0', 1.0))
        cert = generate_semimetal_certificate(H, config['name'])

```

```
        database['models'].append({
            'type': 'NodalLine',
            'name': config['name'],
            'config': config,
            'num_nodal_lines': len(cert.nodal_lines),
            'linking_matrix': cert.linking_matrix.tolist()
                              if cert.linking_matrix is not None else None
        })

    # Save database
    with open(output_dir / 'semimetal_database.json', 'w') as f:
        json.dump(database, f, indent=2)

    print(f"Database saved with {len(database['models'])} models")

    return database
```

Listing 13: Semimetal Database Generation

9.2 3D Visualization Tools

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

def visualize_weyl_semimetal(cert, save_path=None):
    """
    Create 3D visualization of Weyl semimetal.
    """
    fig = plt.figure(figsize=(12, 5))

    # Subplot 1: Weyl points in 3D BZ
    ax1 = fig.add_subplot(121, projection='3d')

    # Draw BZ boundaries
    corners = np.array([
        [-np.pi, -np.pi, -np.pi],
        [np.pi, -np.pi, -np.pi],
        [np.pi, np.pi, -np.pi],
        [-np.pi, np.pi, -np.pi],
        [-np.pi, -np.pi, np.pi],
        [np.pi, -np.pi, np.pi],
        [np.pi, np.pi, np.pi],
        [-np.pi, np.pi, np.pi],
    ])

    # Draw BZ edges
    for i, j in [(0,1),(1,2),(2,3),(3,0),
                 (4,5),(5,6),(6,7),(7,4),
                 (0,4),(1,5),(2,6),(3,7)]:
        ax1.plot3D(*zip(corners[i], corners[j]), 'k-', alpha=0.3)

    # Plot Weyl points
    for wp in cert.weyl_points:
        color = 'red' if wp.chirality > 0 else 'blue'
        marker = '^' if wp.chirality > 0 else 'v'
        ax1.scatter(*wp.position, c=color, s=200, marker=marker,
                    label=f'C={wp.chirality}')

    ax1.set_xlabel('$k_x$')
    ax1.set_ylabel('$k_y$')
    ax1.set_zlabel('$k_z$')
```

```
    ax1.set_title('Weyl Points in Brillouin Zone')

    # Subplot 2: Surface BZ with Fermi arcs
    ax2 = fig.add_subplot(122)

    # Draw surface BZ
    ax2.add_patch(plt.Rectangle((-np.pi, -np.pi), 2*np.pi, 2*np.pi,
                                fill=False, edgecolor='black'))

    # Plot Weyl projections
    for wp in cert.weyl_points:
        color = 'red' if wp.chirality > 0 else 'blue'
        ax2.scatter(wp.position[0], wp.position[1],
                    c=color, s=200, zorder=5)

    # Plot Fermi arcs
    for arc in cert.fermi_arcs:
        ax2.plot(arc.path[:, 0], arc.path[:, 1],
                 'g-', linewidth=3, label='Fermi arc')

    ax2.set_xlabel('$k_x$')
    ax2.set_ylabel('$k_y$')
    ax2.set_title('Surface BZ with Fermi Arcs')
    ax2.set_aspect('equal')
    ax2.set_xlim(-np.pi*1.1, np.pi*1.1)
    ax2.set_ylim(-np.pi*1.1, np.pi*1.1)

    plt.tight_layout()

    if save_path:
        plt.savefig(save_path, dpi=300, bbox_inches='tight')

    plt.show()

def visualize_nodal_lines(cert, save_path=None):
    """
    Visualize nodal lines in 3D momentum space.
    """
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')

    colors = plt.cm.rainbow(np.linspace(0, 1, len(cert.nodal_lines)))

    for idx, nl in enumerate(cert.nodal_lines):
        # Sample nodal line
        t_vals = np.linspace(0, 2*np.pi, 200)
        points = np.array([nl.parameterization(t) for t in t_vals])

        ax.plot3D(points[:, 0], points[:, 1], points[:, 2],
                  color=colors[idx], linewidth=3,
                  label=f'Nodal line {idx+1}')

    ax.set_xlabel('$k_x$')
    ax.set_ylabel('$k_y$')
    ax.set_zlabel('$k_z$')
    ax.set_title('Nodal Lines in Brillouin Zone')
    ax.legend()

    if save_path:
        plt.savefig(save_path, dpi=300, bbox_inches='tight')

```

```
    plt.show()
```

Listing 14: Brillouin Zone Visualization

10 Success Criteria and Milestones

10.1 Minimum Viable Result (MVR)

MVR: Months 1–2

Deliverables:

1. Implement minimal Weyl model with two Weyl points
2. Compute chiralities $C_W = \pm 1$ via Berry flux integration
3. Verify Nielsen-Ninomiya theorem: $\sum_i C_W^{(i)} = 0$
4. Generate certificate with verified positions and chiralities

Validation:

- Weyl point positions match analytical prediction within 10^{-10}
- Chirality values are exactly ± 1 (integers)
- Certificate passes all verification checks

10.2 Strong Result

Strong: Months 3–5

Deliverables:

1. Compute Fermi arc surface states via Green's function
2. Verify arc connectivity (opposite-chirality Weyl points)
3. Implement Dirac semimetal model with $\mathcal{T}$ and $\mathcal{I}$
4. Demonstrate Dirac-to-Weyl splitting under symmetry breaking
5. Track Weyl separation as function of perturbation strength

Validation:

- Fermi arcs connect correct Weyl projections
- Dirac point exhibits 4-fold degeneracy
- Splitting produces exactly 2 Weyl points of opposite chirality

10.3 Publication-Ready Result

Publication: Months 6–7

Deliverables:

1. Implement nodal line semimetal models
2. Compute linking numbers for multi-component nodal lines
3. Identify drumhead surface states
4. Generate comprehensive database of semimetal models
5. Create 3D visualizations of BZ, Weyl points, arcs, nodal lines
6. Classify models by space group symmetry

Validation:

- Linking numbers are integers
- Drumhead states fill interior of projected nodal loops
- Database contains ≥ 10 distinct certified models

10.4 Milestone Timeline

Month	Milestone	Verification
1	Minimal Weyl model	Band structure, crossing detection
2	Chirality computation	Berry flux $= \pm 2\pi$
3	Surface Green's function	Spectral function peaks
4	Fermi arc mapping	Arc connectivity verified
5	Dirac splitting	Weyl pair emergence
6	Nodal line parameterization	Gap $= 0$ along curve
7	Database + visualization	Complete certified catalog

Table 1: Development milestones and verification criteria.

11 Extensions and Advanced Topics

11.1 Type-II Weyl Semimetals

Type-II Weyl semimetals feature tilted Weyl cones where the Fermi surface crosses both electron and hole pockets at the Weyl point.

```
def type_II_weyl_model(m, b, tilt):
    """
    Type-II Weyl semimetal with tilted cone.

    H(k) = tilt*k_z*I + (b*k_z + m)*s_x + b*k_x*s_y + b*k_y*s_z

    Type-II when |tilt| > |b| (overtilted cone).
    """
    sx = np.array([[0, 1], [1, 0]], dtype=complex)
    sy = np.array([[0, -1j], [1j, 0]], dtype=complex)
```

```
    sz = np.array([[1, 0], [0, -1]], dtype=complex)
    I2 = np.eye(2, dtype=complex)

    def H(k):
        kx, ky, kz = k
        return (tilt*kz*I2 + (b*kz + m)*sx +
                b*kx*sy + b*ky*sz)

    return H
```

Listing 15: Type-II Weyl Model

11.2 Multi-Weyl Semimetals

Multi-Weyl points carry higher chirality $|C_W| > 1$ and are protected by rotational symmetry.

```
def double_weyl_model(m, b):
    """
    Double Weyl semimetal with C_W = +/-2.

    Protected by C_4 rotational symmetry.

    H(k) = (k_x^2 - k_y^2)*s_x + 2*k_x*k_y*s_y + (b*k_z + m)*s_z

    Features quadratic dispersion in k_x, k_y plane.
    """
    sx = np.array([[0, 1], [1, 0]], dtype=complex)
    sy = np.array([[0, -1j], [1j, 0]], dtype=complex)
    sz = np.array([[1, 0], [0, -1]], dtype=complex)

    def H(k):
        kx, ky, kz = k
        return ((kx**2 - ky**2)*sx + 2*kx*ky*sy +
                (b*kz + m)*sz)

    return H
```

Listing 16: Multi-Weyl (Double Weyl) Model

11.3 Hopf-Linked Nodal Lines

Nodal lines can form nontrivial links, characterized by the Hopf invariant.

Advanced Topic: Hopf Nodal Links

The Hopf link is the simplest nontrivial link between two closed curves, with linking number $L = 1$. In nodal line semimetals, such configurations can arise from certain band structures and have been predicted in materials like Ca_3P_2.
Computing the full knot invariants (Alexander polynomial, Jones polynomial) for nodal lines remains an open computational challenge.

12 Conclusion

This technical report has presented a comprehensive framework for the "pure thought" construction and certification of topological semimetal models. The key contributions include:

1. **Systematic model construction**: Algorithms for building minimal Weyl, Dirac, and nodal line semimetal Hamiltonians from symmetry constraints alone.

2. **Topological certification**: Methods for computing exact chiralities via Berry curvature integration, verifying the Nielsen-Ninomiya theorem, and establishing symmetry protection.

3. **Surface state analysis**: Techniques for computing Fermi arcs and drumhead states using semi-infinite geometry and Green's function methods.

4. **Nodal line characterization**: Tools for parameterizing nodal curves and computing linking numbers between multiple nodal components.

5. **Database generation**: A framework for cataloging certified semimetal models with full topological characterization.

Pure Thought Achievement

The framework demonstrates that rich topological physics emerges from pure mathematical structure—no materials databases, DFT calculations, or experimental input are required. The certificates provide machine-verifiable proofs of topological properties, ensuring rigorous and reproducible results.

The methods developed here provide a foundation for systematic exploration of the topological semimetal landscape, with applications ranging from materials discovery to quantum computing platforms exploiting topological protection.

References

[1] X. Wan, A. M. Turner, A. Vishwanath, and S. Y. Savrasov, "Topological semimetal and Fermi-arc surface states in the electronic structure of pyrochlore iridates," *Phys. Rev. B* **83**, 205101 (2011).

[2] A. A. Burkov and L. Balents, "Weyl Semimetal in a Topological Insulator Multilayer," *Phys. Rev. Lett.* **107**, 127205 (2011).

[3] H. B. Nielsen and M. Ninomiya, "The Adler-Bell-Jackiw anomaly and Weyl fermions in a crystal," *Phys. Lett. B* **130**, 389 (1983).

[4] N. P. Armitage, E. J. Mele, and A. Vishwanath, "Weyl and Dirac semimetals in three-dimensional solids," *Rev. Mod. Phys.* **90**, 015001 (2018).

[5] C. Fang, H. Weng, X. Dai, and Z. Fang, "Topological nodal line semimetals," *Chinese Physics B* **25**, 117106 (2016).

[6] B.-J. Yang and N. Nagaosa, "Classification of stable three-dimensional Dirac semimetals with nontrivial topology," *Nature Communications* **5**, 4898 (2014).

[7] A. A. Soluyanov, D. Gresch, Z. Wang, Q. Wu, M. Troyer, X. Dai, and B. A. Bernevig, "Type-II Weyl semimetals," *Nature* **527**, 495 (2015).

[8] B. Bradlyn, J. Cano, Z. Wang, M. G. Vergniory, C. Felser, R. J. Cava, and B. A. Bernevig, "Beyond Dirac and Weyl fermions: Unconventional quasiparticles in conventional crystals," *Science* **353**, aaf5037 (2016).

[9] S.-Y. Xu *et al.*, "Discovery of a Weyl fermion semimetal and topological Fermi arcs," *Science* **349**, 613 (2015).

[10] B. Q. Lv *et al.*, "Experimental Discovery of Weyl Semimetal TaAs," *Phys. Rev. X* **5**, 031013 (2015).

A Pauli and Dirac Matrix Conventions

A.1 Pauli Matrices

The standard Pauli matrices are:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \qquad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{16}$$

They satisfy:

$$\{\sigma_i, \sigma_j\} = 2\delta_{ij}, \qquad [\sigma_i, \sigma_j] = 2i\epsilon_{ijk}\sigma_k \tag{17}$$

A.2 Dirac Gamma Matrices

One representation of 4×4 Dirac matrices:

$$\Gamma_1 = \sigma_x \otimes I_2, \qquad \Gamma_2 = \sigma_y \otimes I_2, \tag{18}$$

$$\Gamma_3 = \sigma_z \otimes \sigma_x, \qquad \Gamma_5 = \sigma_z \otimes \sigma_z \tag{19}$$

These satisfy the Clifford algebra:

$$\{\Gamma_\alpha, \Gamma_\beta\} = 2\delta_{\alpha\beta} I_4 \tag{20}$$

B Symmetry Operations

B.1 Time-Reversal Symmetry

For spin-1/2 systems:

$$\mathcal{T} = i\sigma_y K \tag{21}$$

where K is complex conjugation. Properties:

- $\mathcal{T}^2 = -1$ (Kramers theorem)
- $\mathcal{T} H(\boldsymbol{k}) \mathcal{T}^{-1} = H(-\boldsymbol{k})$

B.2 Inversion Symmetry

Inversion acts as:

$$\mathcal{I} H(\boldsymbol{k}) \mathcal{I}^{-1} = H(-\boldsymbol{k}) \tag{22}$$

The representation of $\mathcal{I}$ depends on the orbital content.

B.3 Mirror Symmetry

For mirror reflection $\mathcal{M}_z : z \to -z$:

$$\mathcal{M}_z H(k_x, k_y, k_z) \mathcal{M}_z^{-1} = H(k_x, k_y, -k_z) \tag{23}$$

In the mirror plane ($k_z = 0$), $\mathcal{M}_z$ eigenvalues ± 1 label the bands.

C Numerical Precision Considerations

C.1 Eigenvalue Degeneracy Detection

When searching for band crossings, the threshold for degeneracy detection should be chosen carefully:

- Too large: False positives from near-crossings
- Too small: Missed crossings due to numerical noise

Recommended: $\epsilon \sim 10^{-6}$ for initial search, followed by refinement.

C.2 Berry Phase Gauge Issues

The Berry connection $\boldsymbol{A}(\boldsymbol{k})$ is gauge-dependent. Use gauge-invariant quantities:

- Berry curvature $\boldsymbol{F} = \nabla \times \boldsymbol{A}$
- Berry phase around closed loops
- Chern numbers (flux through closed surfaces)

The plaquette method for computing Berry curvature is manifestly gauge-invariant.

C.3 Surface Green's Function Convergence

For accurate surface spectral functions:

- Use $N_{\text{layers}} \geq 50$ for semi-infinite approximation
- Broadening $\eta \sim 10^{-2}$ for visualization
- Smaller $\eta \sim 10^{-4}$ for precise arc location