```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.linalg as la
from scipy.special import sph_harm

ligands = np.array([
    [-0.83533, -1.37452,  1.52230],
    [-0.83533, -1.37452, -1.52230],
    [-1.59046,  1.37893, -1.52230],
    [-1.59046,  1.37893,  1.52230],
    [ 1.01344,  0.62558,  2.22720],
    [ 2.23727, -0.50498,  0.00000],
    [ 1.01344,  0.62558, -2.22720],
    [ 0.54474,  2.24847,  0.00000]
])

def cartesian_to_spherical(x, y, z):
    r = np.sqrt(x**2 + y**2 + z**2)
    theta = np.arccos(z / r)
    phi = np.arctan2(y, x)
    return r, theta, phi


spherical_coords = np.array([cartesian_to_spherical(*lig) for lig in
ligands])
print (spherical_coords)
```

```
[[ 2.21460577  0.8129057  -2.11687653]
 [ 2.21460577  2.32868696 -2.11687653]
 [ 2.59776986  2.19691319  2.42731121]
 [ 2.59776986  0.94467947  2.42731121]
 [ 2.52563473  0.49105103  0.55303127]
 [ 2.29355223  1.57079633 -0.22199259]
 [ 2.52563473  2.65054162  0.55303127]
 [ 2.31351659  1.57079633  1.33310475]]
```

```python
def compute_Bkq(k, q, spherical_coords, q_eff):
    Bkq = 0
    for i, (r, theta, phi) in enumerate(spherical_coords):
        Ckq = sph_harm(q, k, phi, theta)  # SciPy uses (phi, theta)
        Bkq += q_eff[i] * Ckq / r**(k + 1)
    return Bkq.real  # Real part only (usually Bkq is real for
physical systems)

k_values = [2, 4, 6]
q_values = {
    2: [0,3,-3],
    4: [0, 3,-3, 6,-6],
    6: [0, 3,-3, 6,-6],
}
```

```
Bkq_dict = {}
q_eff = [-0.24] * 8  ########### Tunable (Put either 0.2 or 0.4 or any
value in between) #############

for k in k_values:
    for q in q_values[k]:
        Bkq_dict[(k, q)] = compute_Bkq(k, q, spherical_coords, q_eff)
print(Bkq_dict)
#print(len(Bkq_dict))
```

```
{(2, 0): -0.006218567832834601, (2, 3): nan, (2, -3): nan, (4, 0):
0.0018039043150083142, (4, 3): 7.284483346386983e-19, (4, -3): -
5.72594272343907e-19, (4, 6): nan, (4, -6): nan, (6, 0):
0.0007182277059456796, (6, 3): -1.2305813234646548e-20, (6, -3):
6.820998775343037e-20, (6, 6): -0.00012627611205802612, (6, -6): -
0.00012627611205802617}
```

```
from sympy.physics.wigner import wigner_3j
from sympy import S

l = 3
m_vals = np.arange(-l, l+1)       # m = -3 to +3
dim = 2*l + 1

print("Any NaNs in H_cf:", np.isnan(H_cf).any())
print("Any Infs in H_cf:", np.isinf(H_cf).any())
for kq, val in Bkq_dict.items():
    if not np.isfinite(val):
        print(f"{kq}: {val} is not finite!")
```

```
Any NaNs in H_cf: False
Any Infs in H_cf: False
(2, 3): nan is not finite!
(2, -3): nan is not finite!
(4, 6): nan is not finite!
(4, -6): nan is not finite!
```

```
import numpy as np
from collections import defaultdict
from scipy.linalg import eigh
from sympy.physics.wigner import wigner_3j

def crystal_field_matrix(Bkq_dict, l=3):
    dim = 2 * l + 1
    H_cf = np.zeros((dim, dim), dtype=np.complex128)
    m_vals = np.arange(-l, l + 1)

    # 1. Ensure all missing Bkq values default to 0.0
    Bkq_dict = defaultdict(lambda: 0.0, Bkq_dict)

    # 2. Loop over Stevens parameters
```

```
    for (k, q), Bkq in Bkq_dict.items():
        for i, m_prime in enumerate(m_vals):
            for j, m in enumerate(m_vals):

                # Only valid if m' - m == q
                if m_prime - m != q:
                    continue

                # Compute matrix element using Wigner 3j
                try:
                    w3j = float(wigner_3j(l, k, l, -m_prime, q, m))
                    if w3j != 0:
                        prefactor = ((-1)**(m_prime)) * Bkq
                        coeff = prefactor * np.sqrt((2*l + 1)) * w3j
                        H_cf[i, j] += coeff
                except Exception as e:
                    print(f"Error computing Wigner 3j for (k={k},
q={q}, m'={m_prime}, m={m}): {e}")
                    continue

    # 3. Debug check for NaNs/Infs
    if not np.all(np.isfinite(H_cf)):
        print("Hamiltonian contains invalid (NaN or Inf) entries.")
        print("Inspecting problematic entries:")
        for i in range(dim):
            for j in range(dim):
                if not np.isfinite(H_cf[i, j]):
                    print(f"H[{i},{j}] = {H_cf[i,j]}")
        raise ValueError("Cannot diagonalize: Crystal Field
Hamiltonian contains NaNs or Infs.")

    return H_cf


# Build the Hamiltonian and diagonalize
H_cf = crystal_field_matrix(Bkq_dict, l=3)
eigenvals, eigenvecs = eigh(H_cf)

# Output results
print("Eigenvalues (Crystal Field Levels in arbitrary units):")
print(np.round(eigenvals.real, 6))  # Real part only for inspection
```

```
Error computing Wigner 3j for (k=2, q=0, m'=-3, m=-3): Integers to
negative integer powers are not allowed.
Error computing Wigner 3j for (k=2, q=0, m'=-1, m=-1): Integers to
negative integer powers are not allowed.
Error computing Wigner 3j for (k=4, q=0, m'=-3, m=-3): Integers to
negative integer powers are not allowed.
Error computing Wigner 3j for (k=4, q=0, m'=-2, m=-2): Integers to
negative integer powers are not allowed.
```

```
Error computing Wigner 3j for (k=4, q=0, m'=-1, m=-1): Integers to
negative integer powers are not allowed.
Error computing Wigner 3j for (k=4, q=-3, m'=-3, m=0): Integers to
negative integer powers are not allowed.
Error computing Wigner 3j for (k=4, q=-3, m'=-2, m=1): Integers to
negative integer powers are not allowed.
Error computing Wigner 3j for (k=4, q=-3, m'=-1, m=2): Integers to
negative integer powers are not allowed.
Error computing Wigner 3j for (k=6, q=0, m'=-3, m=-3): Integers to
negative integer powers are not allowed.
Error computing Wigner 3j for (k=6, q=0, m'=-2, m=-2): Integers to
negative integer powers are not allowed.
Error computing Wigner 3j for (k=6, q=0, m'=-1, m=-1): Integers to
negative integer powers are not allowed.
Error computing Wigner 3j for (k=6, q=-3, m'=-3, m=0): Integers to
negative integer powers are not allowed.
Error computing Wigner 3j for (k=6, q=-3, m'=-2, m=1): Integers to
negative integer powers are not allowed.
Error computing Wigner 3j for (k=6, q=-3, m'=-1, m=2): Integers to
negative integer powers are not allowed.
Error computing Wigner 3j for (k=6, q=-6, m'=-3, m=3): Integers to
negative integer powers are not allowed.
Eigenvalues (Crystal Field Levels in arbitrary units):
[-3.634e-03 -2.797e-03 -2.000e-06 -0.000e+00  0.000e+00  1.001e-03
  3.615e-03]

scaling_eV = 0.012  # 1 unit = 0.01 eV

e_sorted = np.sort(eigenvals)
e_shifted = e_sorted - np.min(e_sorted)

e_shifted = [0.0, 1.3845e-2, 1.3846e-2, 1.4852e-2, ..., 2.3858e-2]

ΔE_model = 0.023858 # np.max(e_shifted)   (arbitrary units)

ΔE_exp = 8.69

#scaling_eV = (ΔE_exp / 1000) / ΔE_model

scaling_eV = 0.6700  ############## empirical tuning, no need to
justify this number #############

def plot_crystal_field_levels_meV(eigenvals, scaling_eV):
    eigenvals = np.sort(eigenvals)
    eigenvals_shifted = eigenvals - np.min(eigenvals)
    eigenvals_meV = eigenvals_shifted * scaling_eV * 1000

    plt.figure(figsize=(4, 6))
    for e in eigenvals_meV:
        plt.hlines(e, 0.4, 0.6, color='darkred', linewidth=2)
```

```
        plt.text(0.65, e, f"{e:.2f} meV", va='center', fontsize=9)
    plt.title("Crystal Field Splitting (Tm³⁺ in TmCrO₃)", fontsize=14)
    plt.ylabel("Energy (meV)", fontsize=12)
    plt.xticks([])
    plt.ylim(-0.5, np.max(eigenvals_meV) + 1)
    plt.grid(axis='y', linestyle='--', alpha=0.3)
    plt.tight_layout()
    plt.show()

plot_crystal_field_levels_meV(eigenvals, scaling_eV)
```