Tutorial: Thermoelectric Properties of Monolayer MoS_2 using Quantum ESPRESSO and BoltzTraP2

Contents

1 Overview and Physical Assumptions

This document describes a detailed workflow to compute thermoelectric (TE) properties of a two-dimensional (2D) material, using monolayer MoS_2 as an example, with Quantum ESPRESSO (QE) for the electronic structure, and BoltzTraP2 for semiclassical transport.

BoltzTraP2 solves the linearized Boltzmann transport equation within the constant relaxation time approximation and a rigid-band picture. Given band energies $\varepsilon_n(\mathbf{k})$ (and optionally velocities or momentum matrix elements) on a regular Brillouin zone grid, it computes the Onsager coefficients $L^{(0)}$, $L^{(1)}$, $L^{(2)}$ as functions of temperature T and chemical potential μ, and then derives the standard thermoelectric quantities.

The relevant relations (tensor indices α, β) are:

$$L^{(0)}_{\alpha\beta}(\mu,T) = \sum_{n,\mathbf{k}} \tau\, v_{n\alpha}(\mathbf{k}) v_{n\beta}(\mathbf{k}) \left(-\frac{\partial f(\varepsilon_{n\mathbf{k}},\mu,T)}{\partial \varepsilon}\right), \tag{1}$$

$$L^{(1)}_{\alpha\beta}(\mu,T) = \sum_{n,\mathbf{k}} \tau\, (\varepsilon_{n\mathbf{k}}-\mu)\, v_{n\alpha}(\mathbf{k}) v_{n\beta}(\mathbf{k}) \left(-\frac{\partial f}{\partial \varepsilon}\right), \tag{2}$$

$$L^{(2)}_{\alpha\beta}(\mu,T) = \sum_{n,\mathbf{k}} \tau\, (\varepsilon_{n\mathbf{k}}-\mu)^2\, v_{n\alpha}(\mathbf{k}) v_{n\beta}(\mathbf{k}) \left(-\frac{\partial f}{\partial \varepsilon}\right), \tag{3}$$

where f is the Fermi–Dirac distribution and τ is the relaxation time (assumed constant for the given calculation).

From these, BoltzTraP2 computes:

$$\sigma_{\alpha\beta}(\mu,T) = e^2 L^{(0)}_{\alpha\beta}(\mu,T), \tag{4}$$

$$S_{\alpha\beta}(\mu,T) = \frac{1}{eT}\left[L^{(1)}(\mu,T)\, L^{(0)}(\mu,T)^{-1}\right]_{\alpha\beta}, \tag{5}$$

$$\kappa_{e,\alpha\beta}(\mu,T) = \frac{1}{T}\Big(L^{(2)}_{\alpha\beta}(\mu,T) - \sum_{\gamma\delta} L^{(1)}_{\alpha\gamma}(\mu,T)(L^{(0)}(\mu,T)^{-1})_{\gamma\delta} L^{(1)}_{\delta\beta}(\mu,T)\Big). \tag{6}$$

BoltzTraP2 outputs $\sigma_{\alpha\beta}/\tau$ and $\kappa_{e,\alpha\beta}/\tau$; you must supply τ (for example from experiment or separate scattering calculations) to obtain absolute conductivities.

For a 2D system such as monolayer MoS_2, the code treats the system as 3D with the supercell volume including vacuum. One typically uses the in-plane components σ_{xx}, σ_{yy}, S_{xx}, S_{yy}, and rescales from 3D to effective 2D units using the cell height and an effective thickness, as discussed later.

Limitations of the approach:

- Constant-τ approximation.
- Rigid-band approximation when scanning chemical potential μ.
- No explicit inclusion of electron–phonon scattering details in BoltzTraP2 itself.

2 Software and Environment

2.1 Quantum ESPRESSO

Quantum ESPRESSO is used to perform the DFT calculations. Requirements:

- A recent QE version (6.x or 7.x) that writes the XML file `data-file-schema.xml` in `prefix.save/`.
- Collinear spin calculations only for use with the BoltzTraP2 QE loader. Non-collinear spin–orbit calculations are not supported by the current QE parser in BoltzTraP2.
- Access to suitable pseudopotentials (e.g. PBE PAW or norm-conserving).

Compilation is standard (via `configure` and `make`). You must have the executable `pw.x` available.

2.2 BoltzTraP2

BoltzTraP2 is a Python/C++ package. Installation via `pip`:

```
pip install BoltzTraP2
```

or via conda:

```
conda install -c conda-forge boltztrap2
```

After installation, the command-line entry point `btp2` should be in your `PATH`. A reasonably recent version (for example $\geq$ 22.x) is recommended, where the QE XML loader is available and stable.

BoltzTraP2 reads band structure data either from its own binary files (after interpolation) or directly from code-specific formats, including QE's `data-file-schema.xml`.

3 Quantum ESPRESSO Workflow for Monolayer MoS_2

3.1 Geometry and General Setup

We consider monolayer 1H-MoS_2 in a hexagonal geometry:

- In-plane lattice constant $a \approx 3.16$–3.20 Å.
- Supercell height $c \approx 20$ Å to provide vacuum along the out-of-plane direction.
- Mo in the middle layer; S atoms forming top and bottom layers (S–Mo–S trilayer).

In Cartesian form, a convenient hexagonal primitive cell with $a = 3.18$ Å, $c = 20$ Å is:

$$\mathbf{a}_1 = (3.18, 0.00, 0.0)\,\text{Å},$$
$$\mathbf{a}_2 = (-1.59, 2.754, 0.0)\,\text{Å},$$
$$\mathbf{a}_3 = (0.00, 0.00, 20.0)\,\text{Å}.$$

A plausible starting internal structure uses fractional coordinates:

- Mo at $(0, 0, 0.5)$,
- S at $(\frac{1}{3}, \frac{2}{3}, z_{\text{top}})$,
- S at $(\frac{2}{3}, \frac{1}{3}, z_{\text{bottom}})$,

with $z_{\text{top}} \approx 0.5782$, $z_{\text{bottom}} \approx 0.4218$. This corresponds to a S–S separation of about 3.13 Å.

You may perform structural relaxation to refine these parameters. Below, the same geometry is used in all input examples for simplicity.

3.2 Structural Relaxation Input

A typical input for structural relaxation (`calculation = 'relax'`) is:

```
&CONTROL
  calculation   = 'relax',
  prefix        = 'mos2',
  outdir        = './tmp',
  pseudo_dir    = './pseudo',
  wf_collect    = .true.,
/
&SYSTEM
  ibrav         = 0,
  nat           = 3,
  ntyp          = 2,
  ecutwfc       = 60.0,
  ecutrho       = 480.0,
  occupations   = 'fixed',
  input_dft     = 'PBE',
  assume_isolated = '2D',
/
&ELECTRONS
  conv_thr      = 1.0d-8,
  mixing_beta   = 0.3,
/
&IONS
  ion_dynamics  = 'bfgs',
/
ATOMIC_SPECIES
  Mo  95.95   Mo.pbe-spn-kjpaw_psl.1.0.0.UPF
  S   32.065  S.pbe-n-kjpaw_psl.1.0.0.UPF

CELL_PARAMETERS angstrom
  3.18   0.00   0.0
 -1.59   2.754  0.0
  0.00   0.00  20.0

ATOMIC_POSITIONS crystal
  Mo  0.0000  0.0000  0.5000
```

```
  S   0.3333  0.6667  0.5782
  S   0.6667  0.3333  0.4218

K_POINTS automatic
  12 12 1  0 0 0
```

Run:

```
pw.x < mos2_relax.in > mos2_relax.out
```

After convergence, copy the relaxed lattice vectors and atomic positions to the SCF and NSCF inputs.

3.3 Self-Consistent Field (SCF) Calculation

The SCF calculation produces a converged charge density:

```
&CONTROL
  calculation   = 'scf',
  prefix        = 'mos2',
  outdir        = './tmp',
  pseudo_dir    = './pseudo',
  wf_collect    = .true.,
/
&SYSTEM
  ibrav         = 0,
  nat           = 3,
  ntyp          = 2,
  ecutwfc       = 60.0,
  ecutrho       = 480.0,
  occupations   = 'fixed',
  input_dft     = 'PBE',
  assume_isolated = '2D',
/
&ELECTRONS
  conv_thr      = 1.0d-8,
  mixing_beta   = 0.3,
/
ATOMIC_SPECIES
  Mo  95.95   Mo.pbe-spn-kjpaw_psl.1.0.0.UPF
  S   32.065  S.pbe-n-kjpaw_psl.1.0.0.UPF

CELL_PARAMETERS angstrom
  3.18   0.00   0.0
 -1.59   2.754  0.0
  0.00   0.00  20.0

ATOMIC_POSITIONS crystal
  Mo  0.0000  0.0000  0.5000
```

```
  S   0.3333  0.6667  0.5782
  S   0.6667  0.3333  0.4218

K_POINTS automatic
  12 12 1  0 0 0
```

Run:

```
pw.x < mos2_scf.in > mos2_scf.out
```

Notes:

- The `wf_collect = .true.` flag is convenient when running in parallel, as it stores all wavefunctions in a single directory.
- The SCF k-point grid ($12\times12\times1$) is adequate for generating a well-converged charge density for MoS_2; later we will use a denser grid in the NSCF step.

3.4 Non-SCF (NSCF) Calculation on a Dense Grid

The NSCF calculation on a denser regular Monkhorst–Pack grid is the key input for BoltzTraP2. Example:

```
&CONTROL
  calculation   = 'nscf',
  prefix        = 'mos2',
  outdir        = './tmp',
  pseudo_dir    = './pseudo',
  wf_collect    = .true.,
/
&SYSTEM
  ibrav         = 0,
  nat           = 3,
  ntyp          = 2,
  ecutwfc       = 60.0,
  ecutrho       = 480.0,
  occupations   = 'fixed',
  input_dft     = 'PBE',
  assume_isolated = '2D',
  nspin         = 1,
  nbnd          = 60
/
&ELECTRONS
  conv_thr      = 1.0d-10,
  mixing_beta   = 0.3,
/
ATOMIC_SPECIES
  Mo  95.95   Mo.pbe-spn-kjpaw_psl.1.0.0.UPF
  S   32.065  S.pbe-n-kjpaw_psl.1.0.0.UPF
```

```
CELL_PARAMETERS angstrom
  3.18   0.00   0.0
 -1.59   2.754  0.0
  0.00   0.00  20.0

ATOMIC_POSITIONS crystal
  Mo  0.0000  0.0000  0.5000
  S   0.3333  0.6667  0.5782
  S   0.6667  0.3333  0.4218

K_POINTS automatic
  36 36 1  0 0 0
```

Run:

```
pw.x < mos2_nscf.in > mos2_nscf.out
```

Important points:

- The dense grid (36×36×1, or denser) is necessary because thermoelectric coefficients are sensitive to the band-structure details near the Fermi level.
- The number of bands `nbnd` must be large enough to cover all relevant conduction and valence states in the energy window you plan to explore with BoltzTraP2.
- After completion, QE writes `./tmp/mos2.save/data-file-schema.xml` and the full set of eigenvalues and related data, which will be used by BoltzTraP2.

4 BoltzTraP2 Workflow

4.1 Preparing the Working Directory

A simple arrangement is:

```
mkdir btp2_mos2
cd btp2_mos2
cp -r ../tmp/mos2.save .
```

Now `btp2_mos2/` contains the `mos2.save/` directory with `data-file-schema.xml` and all the necessary QE output files.

4.2 Interpolation

The first BoltzTraP2 step is to interpolate the band structure on a very dense k-grid using Fourier interpolation. For example:

```
btp2 -n 8 -vv interpolate -m 20 mos2.save
```

Options:

- `-n 8`: use 8 CPU cores (optional).
- `-vv`: verbose output.
- `interpolate`: selects the interpolation subcommand.
- `-m 20`: Fourier interpolation factor; typical values are 10–30.
- `mos2.save`: path to the QE `.save` directory containing `data-file-schema.xml`.

You can restrict the energy window around the Fermi level with, for example:

```
btp2 -vv interpolate -o mos2.bt2 -e -0.5 -E 0.5 -m 20 mos2.save
```

where

- `-o mos2.bt2`: specify output interpolation file.
- `-e -0.5`, `-E 0.5`: define lower and upper energy bounds (in eV) relative to the Fermi level.

The interpolation step produces a binary file (by default `interpolation.bt2` or the name given by `-o`) containing the interpolated band structure on a dense k-mesh.

4.3 Transport Integration

The second step is to integrate the transport coefficients over the Brillouin zone as a function of μ and T. For example:

```
btp2 -n 8 -vv integrate -p mos2 interpolation.bt2 300:800:100
```

or, if you produced `mos2.bt2`:

```
btp2 -n 8 -vv integrate -p mos2 mos2.bt2 300:800:100
```

Options:

- `integrate`: selects the transport integration subcommand.
- `-p mos2`: base prefix for outputs (e.g. `mos2.trace`, `mos2.condtens`, etc.).
- Last argument `300:800:100`: temperature range, here $T = 300, 400, 500, 600, 700, 800$ K.

Typical output files:

- `mos2.trace`: compact set of thermoelectric quantities (including Seebeck, σ/τ, κ_e/τ, carrier concentration, etc.) as a function of μ and T.
- `mos2.condtens`: conductivity tensor and related quantities for each μ and T, usually as flattened 3×3 tensors.
- `mos2.halltens`: Hall tensor data.
- `mos2.btj` and related files: information for advanced analyses.

5 Interpreting the Output for a 2D Material

5.1 Typical Content of `.trace` and `.condtens`

The `.trace` file typically contains columns such as:

- μ (eV) relative to some reference (usually the QE Fermi level).
- T (K).
- Carrier concentration or density (for example n_{e} in $1/\mathrm{cm}^3$).
- Seebeck coefficients S_{xx}, S_{yy}, S_{zz} in V/K.
- Conductivity per relaxation time $\sigma_{xx}/\tau, \sigma_{yy}/\tau, \ldots$.
- Electronic thermal conductivity per relaxation time $\kappa_{e,xx}/\tau, \ldots$.

The exact column names and order are indicated in the header line (starting with `#`). It is important to read this header carefully before postprocessing.

The `.condtens` file contains tensor-resolved quantities (e.g. $\sigma_{\alpha\beta}/\tau$) for each μ and T, typically stored as a sequence of 9 elements per tensor (flattened 3×3).

5.2 Seebeck Coefficient and Conductivity

From the BoltzTraP2 output:

- S_{xx} is usually given directly in V/K, which is often converted to μV/K by multiplying by 10^6.
- σ_{xx}/τ is given in appropriate conductivity units per time (e.g. $\mathrm{S\,m^{-1}s^{-1}}$).
- To obtain σ_{xx}, you must supply a relaxation time τ:

$$\sigma_{xx}(\mu, T) = \left[\frac{\sigma_{xx}}{\tau}\right](\mu, T)\,\tau(\mu, T). \tag{7}$$

The relaxation time can be obtained from experiment (fitting to measured conductivity at a known doping level and temperature) or from separate calculations (for example electron–phonon coupling).

5.3 2D vs 3D Units

BoltzTraP2 uses the volume from the DFT supercell. For a monolayer with vacuum, this implies that the computed "3D" conductivity and thermal conductivity are reduced by the large cell height c_{cell}. This is a generic issue in 2D calculations using 3D codes.

Two common approaches:

1. **Effective thickness approach.** Choose an effective monolayer thickness t_{eff}, for example the interlayer spacing in bulk MoS_2 (about 6.15 Å) or the S–S separation plus a small buffer. Then:

$$\sigma_{2D}^{\text{sheet}} = \sigma_{3D} \times c_{\text{cell}}, \tag{8}$$

$$\sigma_{3D}^{\text{eff}} = \frac{\sigma_{2D}^{\text{sheet}}}{t_{\text{eff}}}. \tag{9}$$

   Similarly, the electronic thermal conductivity κ_e can be rescaled:

$$\kappa_{e,2D}^{\text{sheet}} = \kappa_{e,3D} \times c_{\text{cell}}. \tag{10}$$

2. **Reporting sheet conductance.** For device-level modelling, it can be more natural to report sheet conductance (S per square) directly, using $\sigma_{2D}^{\text{sheet}}$, and treat the thickness explicitly in the device geometry.

The Seebeck coefficient S is independent of this scaling (it is a ratio of transport integrals) and can be used directly for the 2D system, provided that the in-plane tensor components are considered.

5.4 Rigid-Band Approximation and Doping

BoltzTraP2 simulates doping by shifting the chemical potential μ in a rigid band structure. This is valid when the band structure does not change significantly with doping and when doping concentrations are moderate. At very high carrier concentrations, band renormalization or many-body effects may become significant, and the rigid-band approximation may break down.

For heavily doped cases, it is more accurate to perform explicit calculations with dopants, charged cells, or external fields, and then re-run BoltzTraP2 for each doped configuration.

5.5 Lattice Thermal Conductivity and Figure of Merit

BoltzTraP2 provides only the electronic thermal conductivity κ_e. The total thermal conductivity κ is:

$$\kappa = \kappa_e + \kappa_{\text{ph}}, \tag{11}$$

where κ_{ph} is the lattice (phonon) contribution. To evaluate the thermoelectric figure of merit:

$$ZT = \frac{S^2 \sigma T}{\kappa_e + \kappa_{\text{ph}}}, \tag{12}$$

one must provide κ_{ph}, typically from phonon transport calculations (e.g. via Boltzmann transport for phonons) or from experimental data.

6 Python Postprocessing: S_{xx} vs Temperature and Carrier Concentration

This section provides a self-contained Python script to read a BoltzTraP2 `.trace` file (for example `mos2.trace`) and plot:

- S_{xx} as a function of temperature T at an approximately intrinsic chemical potential (for example $\mu \approx 0\,\text{eV}$).
- S_{xx} as a function of carrier concentration at fixed temperature (e.g. 300 K).

6.1 Assumptions About `.trace` Format

The script assumes:

- The first non-empty line starting with `#` is a header listing column names, e.g.

```
# mu[eV] T[K] ne[1/cm^3] Sxx[V/K] Syy[V/K] ...
```

- The data lines follow, with one row per pair (μ, T).

Because the exact column names may differ between BoltzTraP2 versions or setups, the script:

1. Reads the header line and obtains a list of column names.
2. Builds a dictionary mapping name to column index.
3. Requires you to provide the exact header tokens corresponding to μ, T, S_{xx}, and the carrier concentration column.

6.2 Python Script

Save the script below as, for example, `plot_btp2_mos2.py` in the directory containing `mos2.trace`.

```
#!/usr/bin/env python3
"""
Post-processing script for BoltzTraP2 .trace output.

It will:
1) Plot Sxx vs T at approximately intrinsic conditions (mu ~ 0).
2) Plot Sxx vs carrier concentration at a chosen temperature.

You must adjust the COLUMN_NAME_* variables to match the header
of your mos2.trace file.
"""

import numpy as np
```

```
import matplotlib.pyplot as plt

TRACE_FILE = "mos2.trace"

# ---------------------------------------------------------------------
# 1. Column names from the header
#
# Open mos2.trace and inspect the first line starting with '#'.
# Example (this is just an illustration, not a guarantee):
#   # mu[eV] T[K] ne[1/cm^3] Sxx[V/K] Syy[V/K] Szz[V/K] ...
#
# Put the exact tokens that appear in that header here:
COLUMN_NAME_MU   = "mu[eV]"       # chemical potential
COLUMN_NAME_T    = "T[K]"         # temperature
COLUMN_NAME_SXX  = "Sxx[V/K]"     # Seebeck Sxx
COLUMN_NAME_NE   = "ne[1/cm^3]"   # electron (or net) carrier concentration

# If your file uses different labels, e.g. "mu" or "S[V/K]" or "neff",
# replace the strings above accordingly.

# ---------------------------------------------------------------------
# 2. Utility functions
def read_trace_with_header(filename):
    """
    Reads a BoltzTraP2 .trace file that has a one-line header starting with '#'
    containing the column names.

    Returns
    -------
    names : list of str
        Column names as parsed from the header line.
    data : ndarray (nrows, ncols)
        Numerical data.
    """
    header_line = None
    with open(filename, "r") as f:
        for line in f:
            if not line.strip():
                continue
            if line.lstrip().startswith("#"):
                header_line = line.lstrip()[1:].strip()
                break
    if header_line is None:
        raise RuntimeError("Could not find a header line beginning with '#' in {}".
   format(filename))

    names = header_line.split()
```

```
    data = np.loadtxt(filename, comments="#")
    if data.ndim == 1:
        data = data[None, :]
    if data.shape[1] != len(names):
        raise RuntimeError(
            "Number of columns in data ({}) does not match number of names ({})"
            .format(data.shape[1], len(names))
        )
    return names, data


def build_name_to_col(names):
    """
    Build a dictionary mapping column name to column index.
    """
    return {name: i for i, name in enumerate(names)}


def extract_Sxx_vs_T_at_mu(data, col_mu, col_T, col_Sxx, mu_target=0.0, atol=0.05):
    """
    For each temperature, find the Sxx at a chemical potential closest
    to mu_target (default 0 eV). This approximates intrinsic conditions.

    Parameters
    ----------
    data : ndarray
        Full data array.
    col_mu, col_T, col_Sxx : int
        Column indices for mu, T, and Sxx.
    mu_target : float
        Target chemical potential in eV.
    atol : float
        Maximum distance in eV; used only for sanity checks.

    Returns
    -------
    T_vals : ndarray
        Sorted unique temperatures.
    Sxx_microV_per_K : ndarray
        Corresponding Sxx in microvolt per kelvin.
    """
    T_vals = np.unique(data[:, col_T])
    T_vals = np.sort(T_vals)

    S_vals = []
    for T0 in T_vals:
        mask_T = np.isclose(data[:, col_T], T0)
```

```
        subset = data[mask_T]
        if subset.size == 0:
            continue

        # find row where |mu - mu_target| is minimal
        idx = np.argmin(np.abs(subset[:, col_mu] - mu_target))
        mu_sel = subset[idx, col_mu]
        S_sel = subset[idx, col_Sxx]

        if np.abs(mu_sel - mu_target) > atol:
            print(
                "Warning: at T = {:.1f} K, closest mu to target is {:.3f} eV".
    format(
                    T0, mu_sel
                )
            )

        # Convert from V/K to microV/K if needed.
        S_microV = S_sel * 1.0e6
        S_vals.append(S_microV)

    return T_vals, np.array(S_vals)


def extract_Sxx_vs_n_at_T(data, col_T, col_Sxx, col_n, T_target=300.0, atol=1.0):
    """
    For a given temperature T_target, return Sxx vs carrier concentration.

    Parameters
    ----------
    data : ndarray
    col_T, col_Sxx, col_n : int
        Column indices for T, Sxx, and carrier concentration.
    T_target : float
        Target temperature in K.
    atol : float
        Tolerance for matching T (K).

    Returns
    -------
    n_vals : ndarray
        Carrier concentrations (as stored in the file), e.g. 1/cm^3.
    Sxx_microV_per_K : ndarray
        Seebeck coefficients in microvolt per kelvin.
    """
    mask_T = np.isclose(data[:, col_T], T_target, atol=atol)
    subset = data[mask_T]
```

```
    if subset.size == 0:
        raise RuntimeError("No data found at T = {} K".format(T_target))

    n_vals = subset[:, col_n]
    S_vals_microV = subset[:, col_Sxx] * 1.0e6
    return n_vals, S_vals_microV


# ---------------------------------------------------------------------
# 3. Main analysis and plotting
if __name__ == "__main__":
    names, data = read_trace_with_header(TRACE_FILE)
    name_to_col = build_name_to_col(names)

    print("Columns found in {}:".format(TRACE_FILE))
    for i, name in enumerate(names):
        print("  {:2d}: {}".format(i, name))

    # Resolve the column indices from the chosen names
    try:
        col_mu  = name_to_col[COLUMN_NAME_MU]
        col_T   = name_to_col[COLUMN_NAME_T]
        col_Sxx = name_to_col[COLUMN_NAME_SXX]
        col_ne  = name_to_col[COLUMN_NAME_NE]
    except KeyError as err:
        raise SystemExit(
            "Column name {} not found. Adjust COLUMN_NAME_* variables at top of
   script."
            .format(err)
        )

    # 3.1 Plot Sxx vs T at approximately intrinsic conditions (mu_target = 0 eV)
    mu_target = 0.0  # adjust if you want another reference
    T_vals, S_T = extract_Sxx_vs_T_at_mu(
        data, col_mu, col_T, col_Sxx, mu_target=mu_target
    )

    plt.figure()
    plt.plot(T_vals, S_T, marker="o")
    plt.xlabel("Temperature T (K)")
    plt.ylabel(r"$S_{xx}$ ($\mu$V/K)")
    plt.title(r"$S_{xx}(T)$ at $\mu \approx {:.2f}$ eV".format(mu_target))
    plt.grid(True)
    plt.tight_layout()
    plt.savefig("Sxx_vs_T.png", dpi=300)

    # 3.2 Plot Sxx vs carrier concentration at a chosen T
```

```
T_target = 300.0  # K
n_vals, S_n = extract_Sxx_vs_n_at_T(
    data, col_T, col_Sxx, col_ne, T_target=T_target
)

plt.figure()
plt.plot(n_vals, S_n, marker="o")
plt.xlabel(r"Carrier concentration $n$ (1/cm$^3$)")
plt.ylabel(r"$S_{xx}$ ($\mu$V/K)")
plt.title(r"$S_{xx}(n)$ at T = {:.1f} K".format(T_target))
plt.xscale("symlog")  # often helpful, you can use 'log' if n > 0
plt.grid(True)
plt.tight_layout()
plt.savefig("Sxx_vs_n_T{:g}K.png".format(T_target), dpi=300)

plt.show()
```

6.3 Usage Notes

1. Inspect the header of `mos2.trace` using, for example:

   ```
   head mos2.trace
   ```

   and identify the exact column labels for μ, T, S_{xx}, and carrier concentration.

2. Put these exact tokens into:

   ```
   COLUMN_NAME_MU   = "mu[eV]"
   COLUMN_NAME_T    = "T[K]"
   COLUMN_NAME_SXX  = "Sxx[V/K]"
   COLUMN_NAME_NE   = "ne[1/cm^3]"
   ```

3. Run the script:

   ```
   python3 plot_btp2_mos2.py
   ```

4. The script prints the column names with indices, warns if the closest μ to the target is not very close, and generates:

   - `Sxx_vs_T.png`: $S_{xx}(T)$ at $\mu \approx \mu_{\text{target}}$.
   - `Sxx_vs_n_T300K.png`: $S_{xx}(n)$ at $T = 300$ K.

The functions `extract_Sxx_vs_T_at_mu` and `extract_Sxx_vs_n_at_T` can be adapted to:

- Use a different reference μ_{target}.

- Fix carrier concentration and obtain $S_{xx}(T)$.
- Work with in-plane isotropic averages (for example average of S_{xx} and S_{yy}).

7 Minimal Recipe Summary

For convenience, the full workflow for monolayer MoS_2 is:

1. Build a monolayer S–Mo–S structure with large vacuum (e.g. $c = 20$ Å).
2. Relax the geometry (`calculation = 'relax'`) if needed.
3. Run an SCF calculation with a moderate 12×12×1 k-mesh.
4. Run an NSCF calculation with a dense 36×36×1 (or denser) k-mesh, sufficient `nbnd`, and `wf_collect = .true.`.
5. Copy `mos2.save` into a clean directory and run:

```
btp2 -n 8 -vv interpolate -m 20 mos2.save
btp2 -n 8 -vv integrate -p mos2 interpolation.bt2 300:800:100
```

6. Parse `mos2.trace` (and/or `mos2.condtens`) with the Python script above to obtain S_{xx}, σ_{xx}/τ, and $\kappa_{e,xx}/\tau$ as functions of T, μ or carrier concentration.
7. Rescale conductivities for the 2D system using the cell height c_{cell} and an effective thickness t_{eff} if you need 3D-like values.
8. Provide or estimate a relaxation time τ and a lattice thermal conductivity κ_{ph}, and compute the figure of merit ZT.