Computational Materials Science 200 (2021) 110828

Contents lists available at ScienceDirect

Computational Materials Science

journal homepage: www.elsevier.com/locate/commatsci

Advanced modeling of materials with PAOFLOW 2.0: New features and software design

Frank T. Cerasoli [a], Andrew R. Supka [b,c], Anooja Jayaraj [a], Marcio Costa [d], Ilaria Siloi [e], Jagoda Sławińska [f], Stefano Curtarolo [g], Marco Fornari [b,c,g], Davide Ceresoli [h], Marco Buongiorno Nardelli [a,g,*]

[a] Department of Physics and Department of Chemistry, University of North Texas, Denton TX, USA
[b] Department of Physics, Central Michigan University, Mount Pleasant MI, USA
[c] Science of Advanced Materials Program, Central Michigan University, Mount Pleasant MI, USA
[d] Department of Physics, Fluminense Federal University, Niterói24210-346 Rio de Janeiro, Brazil
[e] Department of Physics and Astronomy, University of Southern California, Los Angeles, CA 90089, United States
[f] Zernike Institute for Advanced Materials, University of Groningen, Nijenborgh 4, NL-9747AG, Groningen, NL, The Netherlands
[g] Center for Materials Genomics, Duke University, Durham, NC 27708, USA
[h] Consiglio Nazionale Delle Ricerche, Istituto di Scienze e Tecnologie Chimiche "G. Natta" (CNR-SCITEC), 20133 Milan, Italy

ARTICLE INFO

Dataset link: https://github.com/marcobn/PAOFLOW

Keywords:
DFT
Electronic structure
Ab initio tight-binding
High-throughput calculations

ABSTRACT

Recent research in materials science opens exciting perspectives to design novel quantum materials and devices, but it calls for quantitative predictions of properties which are not accessible in standard first principles packages. PAOFLOW, is a software tool that constructs tight-binding Hamiltonians from self-consistent electronic wavefunctions by projecting onto a set of atomic orbitals. The electronic structure provides numerous materials properties that otherwise would have to be calculated via phenomenological models. In this paper, we describe recent re-design of the code as well as the new features and improvements in performance. In particular, we have implemented symmetry operations for unfolding equivalent **k**-points, which drastically reduces the runtime requirements of first principles calculations, and we have provided internal routines of projections onto atomic orbitals enabling generation of real space atomic orbitals. Moreover, we have included models for non-constant relaxation time in electronic transport calculations, doubling the real space dimensions of the Hamiltonian as well as the construction of Hamiltonians directly from analytical models. Importantly, PAOFLOW has been now converted into a Python package, and is streamlined for use directly within other Python codes. The new object oriented design treats PAOFLOW's computational routines as class methods, providing an API for explicit control of each calculation.

1. Introduction

Exploring phenomena and properties of novel materials requires accurate and efficient computational tools that can be easily customized and manipulated. In this context, *ab initio* tight-binding (TB) Hamiltonians constructed from self-consistent quantum-mechanical wavefunctions projected onto a set of atomic orbitals have been very successful, since they allow calculations for materials that cannot be properly addressed using only density functional theory (DFT) such as large moiré superstructures or properties of exotic quantum systems where spin and topology play an important role. PAOFLOW [1] is a new software tool that employs an efficient procedure of projecting the full plane-wave solution on a reduced space of pseudoatomic orbitals [2,3], and provides an interpolated electronic structure to promptly compute a plethora of relevant quantities, including optical and magnetic properties, charge and spin transport as well as topological invariants. Importantly, in contrast with other common approaches the projection does not require any additional inputs and can be successively integrated in high-throughput calculations of arbitrary complex materials. The code has been employed in multiple areas of materials science since its initial release in 2016. In particular, several groups used it to compute the (spin) Berry curvature as well as spin and/or anomalous Hall conductivity (SHC and AHC) in a variety of materials, ranging from β-W to magnetic antiperovskites [4–8]. Transport quantities, such as the electrical and thermal conductivity, were also computed in order to analyze carrier mobility in thermoelectrics [9,10].

* Corresponding author.
E-mail address: mbn@unt.edu (M. Buongiorno Nardelli).

https://doi.org/10.1016/j.commatsci.2021.110828
Received 9 July 2021; Received in revised form 20 August 2021; Accepted 22 August 2021
Available online 2 September 2021

The software package has recently undergone a major refactor, resulting in a large variety of properties that can be calculated as well as highly improved performance. Many improvements were made to simplify the user experience, to make the package more modular, and to create an API for manipulating TB Hamiltonians. PAOFLOW, now installed as a Python package, features an object oriented design and contains an importable PAOFLOW class, allowing multiple Hamiltonians to be constructed and manipulated simultaneously. This framework enables high-throughput materials analysis within a single python file. While explicit descriptions of the code's methodology are available in the original PAOFLOW paper [1], in this paper we outline PAOFLOW's modified features, detail new functionalities, and provide a user manual for operating the various methods available within the package. Currently, PAOFLOW is publicly available under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or any later version. It is also integrated in the AFLOWπ high-throughput framework [11] and distributed at http://www.aflow.org/src/aflowpi and http://www.aflow.org/src/paoflow [12,13].

2. Installation and software design

PAOFLOW is written in Python 3.8 (using the Python standard libraries, `NumPy` and `SciPy`). Parallelization on CPUs uses the openMPI protocol through the `mpi4py` module. The PAOFLOW package can be easily installed on any hardware. Installation directly from the Python package index (PyPi) is possible with the single command: `pip install paoflow`. Otherwise, one may clone the PAOFLOW repository from GitHub and install it from the root directory with the command: `python setup.py install`. The package requires no specific setup, provided that the prerequisite Python 3.8 modules are installed on the system. Example codes illustrating PAOFLOW's capabilities are included on GitHub, in the `examples/` directory of the PAOFLOW repository. Once installed, PAOFLOW can be imported into any Python code and used in conjunction with other software packages. It should be noted that, in PAOFLOW 1.0 two control files were required: `main.py` to begin the execution, and `inputfile.xml` to provide details of the calculation. In PAOFLOW 2.0 the desired routines are called directly from the Python code, eliminating the need for `inputfile.xml`. To facilitate the transition between the versions, in the `examples/` directory we have provided an updated `main.py` file which allows use of the version 1.0 XML inputfile structure in PAOFLOW 2.0.

Generally, PAOFLOW requires a few basic calculations performed with the Quantum ESPRESSO (QE) package [14]. The first (self-consistent) run generates a converged electronic density and Kohn–Sham (KS) potential on an appropriate Monkhorst and Pack (MP) **k**-point mesh (`pw.x`). The second (non self-consistent) one (`pw.x`) evaluates eigenvalues and eigenfunctions on a larger MP mesh and often for an increased number of bands. In PAOFLOW 1.0 the latter required full **k**-point mesh without using symmetries, i.e. setting flags `nosym=.true.` and `noinv=.true.` in QE inputfile, while version 2.0 can reconstruct equivalent **k**-points from symmetry operations and has no such requirement, which greatly reduces the computation time at the level of DFT. As a next step, the KS wavefunctions from QE must be projected onto PAO basis functions. Also here we have introduced an important change — a new capability is the generation of real space atomic orbitals, constructed from the product of radial components from pseudopotential files and spherical harmonics specifying angular momentum dependence. The user can thus choose between two different options: project the KS wavefunctions onto this internally constructed PAO basis set (Section 3.2), or post-processing the QE output with `projwfc.x` in order to perform the projection, as it was done in version 1.0. If the projections are computed using QE, the eigenfunctions must be read by PAOFLOW before the construction of a PAO Hamiltonian (Section 3.3). Performing the projections in PAOFLOW enables construction of real space PAO wavefunctions for computing spatially resolved quantities like charge density. Finally, PAOFLOW 2.0 introduces a possibility to operate with no preprocessing requirements from QE, where built-in or user-defined models serve as the recipe for building Hamiltonians. Implementing these models requires specific information about the atomic system *a priori*, and their usage is described in Section 4.

Central to PAOFLOW 2.0 is an internal object called the `DataController`, which has the sole responsibility of collecting and maintaining important information about the atomic system and its corresponding Hamiltonian. The `DataController` is initially populated with data from the QE calculation, providing many of PAOFLOW's routines with required information and simplifying the way of calling functions by the user. Moreover, the `DataController` saves quantities needed for subsequent calculations, such as the Hamiltonian's gradient or the adaptive smearing parameters. In fact, many routines require calculation of some quantities as a prerequisite. For example, the `spin_Hall` routine requires the Hamiltonian's gradient, which means that the `gradient_and_momenta` should be called first to populate the `DataController` with the gradient and momenta. The `DataController` stores the system information in two dictionaries: one for strings and scalar attributes (`data_attributes`) and another for vector and tensor quantities (`data_arrays`). Such a structure allows computed quantities to be easily accessed from PAOFLOW and utilized in customized calculations defined by the user. Note that the dictionary keys are consistent with the naming conventions of PAOFLOW 1.0 to facilitate backwards compatibility with the XML inputfiles and minimize differences in the user experience when transitioning from the previous version.

3. Code description and package usage

PAOFLOW's most fundamental procedure is the construction of accurate PAO Hamiltonians, and the code's object-oriented design allows users to manipulate multiple Hamiltonians easily. A PAOFLOW object is responsible for a single Hamiltonian, which is constructed and operated on with PAOFLOW's class methods. If instead of DFT electronic wavefunctions a TB model is used to construct the Hamiltonian, the new PAOFLOW object will create the Hamiltonian immediately. Otherwise, the KS wavefunctions are read from the output of QE. The atomic orbitals are constructed and the KS wavefunctions are projected onto them with the `projections` routine, creating the PAO basis. If the projections are performed by QE's module `projwfc.x`, they must be read explicitly with `read_atomic_proj_QE`. Then, the Hamiltonian is constructed with `build_pao_hamiltonian`, which allows PAOFLOW's other class methods to become functional. Listing 1 provides an example source code for building the PAOFLOW object, reading projections performed by QE, and constructing the PAO Hamiltonian. Listing 2 performs the same initialization procedure, but uses the internal atomic orbital projection scheme. An ellipsis appearing in any listing indicates that other PAOFLOW routines may follow.

The following subsections outline PAOFLOW's individual routines and the arguments that they accept for control. These routines belong to the file PAOFLOW.py, located in the package's `src/` directory, and should be called directly by the user.

3.1. *The constructor:* `PAOFLOW`

The PAOFLOW constructor acquires information about the python execution, the names of input/output/working directories, and about the atomic system. It builds and populates the `DataController`, which will maintain the important quantities involved in calculations, handle communication in multi-core runs, and write files to disc when necessary.

For PAOFLOW to build a Hamiltonian, the constructor must be passed data with either the location of Quantum ESPRESSO's `.save` directory or required specifications for an analytical TB model. The QE `.save` directory contain up to two files from the execution of

QE: `data-file-schema.xml` (`data-file.xml` in previous versions of QE) generated by the main run with `pw.x`, and `atomic_proj.xml` generated by the post-processing tool `projwfc.x`, if the projections are computed with QE. To implement a TB model from scratch, rather than starting from the KS wave function solutions of DFT, a dictionary containing the model's label and other required parameters should be passed into the **model** argument (see Section 4).

Arguments for the constructor, `PAOFLOW`:

- **workpath** (string) – *Default*: `'./'` – Path to the working directory. Defaults to the current working directory.
- **outputdir** (string) – *Default*: `'output'` – Name of the directory to store output data files. The directory is created automatically in the **workpath**, if it does not already exist.
- **inputfile** (string) – *Default*: `None` - This argument is primarily for backwards compatibility with PAOFLOW 1.0. It names the XML inputfile with control parameters described in Ref. [1]. The XML inputfile provides also a consistent descriptor format for highly automated calculations, utilized by AFLOWπ.
- **savedir** (string) – *Default*: `None` – Name of the Quantum ESPRESSO `.save` directory, relative to the working directory.
- **model** (dict) – *Default*: `None` – Dictionary specifying parameters required to implement a TB model. See Section 4.
- **npool** (integer) – *Default*: `1` – Number of batches to process when communicating between processors. This value will be automatically increased if the Hamiltonian size exceeds `mpi4py`'s limit for a single cross core message.
- **smearing** (string) – *Default*: `'gauss'` – Selects the broadening technique used to smooth computed quantities. Options include `'gauss'`, `'m-p'` (Methfessel–Paxton), and `None`.
- **acbn0** (bool) – *Default*: `False` – Read overlaps to construct a non-orthogonal PAO Hamiltonian. Necessary to perform ACBN0 calculations [15,16].
- **verbose** (bool) – *Default*: `False` – Flag for high verbosity. Set `True` to include additional information in the PAOFLOW output.
- **restart** (bool) – *Default*: `False` – Indicates the continuation of a previous run from the saved state. Once the PAOFLOW object is instantiated, the `restart_load` routine should be called with the save file's prefix as an argument. An example is provided in listings 4 and 5.

3.2. `projections`

Perform projections of the KS eigenfunctions onto the pseudoatomic orbital basis which is constructed by PAOFLOW based on the information in the atomic pseudopotential files. This operation requires that PAOFLOW is instantiated by passing a QE `.save` directory with required XML file from the self-consistent and non self-consistent calculations. Listing 2 provides example usage of the `projections` routine, and a complete description of the projection methodology can be found in the Appendix of Ref. [2].

3.3. `read_atomic_proj_QE`

Read the projections of KS wavefunctions onto the atomic orbital basis of the pseudopotential, performed by the QE routine `projwfc.x` and written to `atomic_proj.xml`. Any time that the Hamiltonian is built directly from the projections of QE, this routine should be called immediately after the constructor.

read_atomic_proj_QE does not accept any arguments.

3.4. `projectability`

The `projectability` routine determines which bands do not meet projectability requirements, flagging them for shift into the null space. The projectability $p_{\mathbf{k}}$ is a quantity measuring how well a KS Bloch state is represented by orbitals of the PAO basis, as described in Section 3 of Ref. [1]. If $p_{\mathbf{k}} \approx 1$, the PAO basis set accurately represents that particular Bloch state for $\mathbf{k}$. $p_{\mathbf{k}} \ll 1$ indicates that the state is poorly represented and should be excluded. The projection threshold **pthr** selects the minimum allowed projectability for accepting a band. All bands which do not meet the criteria are projected to the null space during the Hamiltonian's construction, in one of two ways. Either as

$$\hat{H}(\mathbf{k}) = AEA^{\dagger} + \kappa\left(I - AA^{\dagger}\right) \tag{1}$$

following Ref. [17] or

$$\hat{H}(\mathbf{k}) = AEA^{\dagger} + \kappa\left(I - A\left(A^{\dagger}A\right)^{-1}A^{\dagger}\right) \tag{2}$$

according to Ref. [2]. Unless the **shift** argument is explicitly set to a floating point value, the shifting parameter κ is determined automatically by this routine. Which method is used to remove low-projectability bands during the Hamiltonian construction is selected by argument **shift_type**, in the `pao_hamiltonian` routine (Section 3.5).

Arguments for `projectability`:

- **pthr** (float) – *Default*: `0.95` – The projectability threshold [2]. All bands with a minimum projectability of the **pthr** value or higher are included in the Hamiltonian. Decreasing this threshold will, in general, increase the number of included bands. However, including states with lower projectability risks nonphysical contributions entering into the Hamiltonian.
- **shift** (string or float) – *Default*: `'auto'` – Float to indicate the value (in eV) of the null space cutoff (κ in Eqs. (1) and (2)) [2]. Bands beneath the projectability threshold will be shifted to this value. Providing the default argument `'auto'` automatically sets **shift**'s value to the minimum energy of the first band that fails the projectability threshold.

3.5. `pao_hamiltonian`

This routine constructs the Hamiltonian in both real space and momentum space. After this routine is completed, the data controller will contain arrays `HRs` and `Hks`, for the respective real space and **k**-space Hamiltonians.

Listing 1: `main.py` **- Build Hamiltonian**

```
from PAOFLOW import PAOFLOW

pao = PAOFLOW.PAOFLOW(savedir='system.save')
pao.read_atomic_proj_QE()
pao.projectability(pthr=0.95)
pao.pao_hamiltonian()
...
```

Arguments for `pao_hamiltonian`:

- **shift_type** (integer) – *Default*: `1` – Determines which method (Eq. (2) by default) is used to remove bands into the null space. 0 — Eq. (1), 1 — Eq. (2), or 2 — No shift.
- **insulator** (bool) — *Default*: `False` — Setting this flag as `True` asserts that the system is insulating, setting the top of the highest occupied band to 0 eV. The Fermi energy is calculated for metallic systems, which corrects numerical discrepancies from the projection routine and irreducible wedge unfolding. This flag is set `True` automatically if the QE output does not contain smearing parameters.

- **write_binary** (bool) – *Default*: `False` – Flag to write the files necessary for the ACBN0 routine [18]. Overlaps from `projwfc.x` are required from prerequisite QE calculations, and this flag is required for further use with ACBN0.
- **expand_wedge** (bool) – *Default*: `True` – Applies symmetry operations to the **k**-mesh unfolding the irreducible wedge into the complete Brillouin zone, and evaluates the Hamiltonian matrix elements for every **k**-point. PAOFLOW routines act on the full grid of **k**-points (`True`), while ACBN0 only requires the irreducible wedge (`False`).
- **symmetrize** (bool) – *Default*: `False` – The Hamiltonian incurs numerical errors during the process of unfolding the wedge. Certain routines, such as `find_weyl_points`, are sensitive to the Hamiltonian's symmetric components. Setting this flag to `True` symmetrizes the Hamiltonian with an iterative procedure to reduce numerical errors in the Hamiltonian's symmetry.
- **thresh** (float) – *Default*: `1e-6` – The tolerance of symmetrization, if the procedure is performed.
- **max_iter** (integer) – *Default*: `16` – The maximum number of iterations that the symmetrization procedure will perform.

3.6. bands

Computes the band structure along the AFLOW standard path for the specified Bravais lattice. Alternatively, a custom path can be created by defining the high symmetry points in a dictionary and the band path as a string (see Listing 2). The path is Fourier interpolated to an arbitrary resolution, controlled by argument **nk**.

Listing 2: `main.py` **- Bands**

```
from PAOFLOW import PAOFLOW

pao = PAOFLOW.PAOFLOW(savedir='system.save')
pao.projections()
pao.projectability()
pao.pao_hamiltonian()

path = 'G-X-S-Y-G'
sym_points = {'G':[0.0, 0.0, 0.0],
              'S':[0.5, 0.5, 0.0],
              'X':[0.5, 0.0, 0.0],
              'Y':[0.0, 0.5, 0.0]}
pao.bands(ibrav=8,
          nk=1000,
          band_path=path,
          high_sym_points=sym_points)
...
```

Arguments for `bands`:

- **ibrav** (integer) – *Default*: `None` – The Bravais lattice identifier, as specified by Quantum ESPRESSO.
- **band_path** (string) – *Default*: `None` – String of high symmetry point labels separated by '-' for a line connecting two points or '|' to place the points directly adjacent on the path (see Listing 2). If **band_path** is `None` the standard AFLOW path will be used [19].
- **high_sym_points** (dictionary) – *Default*: `None` – A dictionary mapping the string label of a high symmetry point to its three-dimensional crystal coordinate. (Listing 2)
- **fname** (string) – *Default*: `'bands'` – File name prefix for the bands. One file is written for each spin component (see listing 2).
- **nk** (int) – *Default*: `500` – Number of points to compute along the band path.

3.7. interpolated_hamiltonian

Fourier interpolation of the PAO Hamiltonian can increase the **k**-grid to an arbitrary density, as described and illustrated in the manuscript for PAOFLOW 1.0. The new desired dimension should be specified for `nk1`, `nk2`, and `nk3`. The default behavior is to double the original **nk** dimension of any unspecified nfft argument. This routine populates the `DataController` with a new array 'Hksp', the interpolated Hamiltonian.

Arguments for `interpolated_hamiltonian`:

- **nfft1** (integer) – *Default*: `None` – The desired new dimension for the Hamiltonian's previous dimension nk1. The nfft dimension should be greater than or equal to the previous nk dimension. If no argument is provided the original **k**-grid dimension is doubled.
- **nfft2** (integer) – *Default*: `None` – New interpolated dimension for nk2, following the same scheme as **nfft1**.
- **nfft3** (integer) – *Default*: `None` – New interpolated dimension for nk3, following the same scheme as **nfft1** and **nfft2**.
- **reshift_Ef** (bool) – *Default*: `False` – Shift the Hamiltonian's diagonal elements such that zero lies at the recomputed Fermi energy or at the top of the valence band.

3.8. spin_operator

The spin operator plays important role in several calculations performed by the PAOFLOW code. Generally, when the spin operator $\mathbf{S}_j$ is required, PAOFLOW automatically constructs it. However, $\mathbf{S}_j$ can be explicitly computed by calling this routine. The shell levels and their occupations are automatically read from pseudopotentials in the `.save` directory.

Arguments for `spin_operator`:

- **spin_orbit** (bool) – *Default*: `False` – Set this flag to `True` if spin orbit coupling is added at the PAO level (with the `adhoc_spin_orbit` routine).

3.9. add_external_fields

PAOFLOW supports the addition of electric fields, onsite Zeeman fields, or Hubbard corrections directly to the PAO Hamiltonian [15,20]. Fields must be added after the Hamiltonian's construction. Listing 3 provides an example where an electric field and Hubbard correction are simultaneously added to a Hamiltonian.

Listing 3: `main.py` **- External fields**

```
from PAOFLOW import PAOFLOW

pao = PAOFLOW.PAOFLOW(savedir='system.save')
pao.projections()
pao.projectability()
pao.pao_hamiltonian()

hubbardU = np.zeros(32, dtype=float)
hubbardU[1:4] = .1
hubbardU[17:20] = 2.31
pao.add_external_fields(Efield=[.1,0.,0.],
                        HubbardU=hubbardU)
...
```

Arguments for `add_external_fields`:

- **Efield** (ndarray or list) – *Default*: `[0.]` – An array of the form $[E_x, E_y, E_z]$, added to the diagonal elements of the Hamiltonian.

Listing 3 provides an example of adding an electric field with one non-zero component.

- **Bfield** (ndarray or list) – *Default*: `[0.]` – An array of the form $[B_x, B_y, B_z]$, specifying the strength and direction of an on-site magnetic field.
- **HubbardU** (ndarray or list) – *Default*: `[0.]` – An array with one U entry for each orbital, e.g. $[U_1, U_2, \ldots, U_n]$ where n is the number of orbitals. An example is provided in Listing 3.

3.10. `adhoc_spin_orbit`

This routine allows the addition of spin–orbit coupling (SOC) at the PAO level. SOC is implemented for the following shell configurations, provided as the **orb_pseudo** argument: s, sp, spd, ps, ssp, sspd, and ssppd.

Arguments for `adhoc_spin_orbit`:

- **naw** (ndarray or list) – *Default*: `[1]` – List containing the number of wave functions for each pseudopotential.
- **phi** (flaot) – *Default*: `0.` – Spin orbit azimuthal angle.
- **theta** (float) – *Default*: `0.` – Spin orbit polar angle.
- **lambda_p** (ndarray or list) – *Default*: `[0.]` – Array of `p`-orbital coupling strengths.
- **lambda_d** (ndarray or list) – *Default*: `[0.]` – Array of `d`-orbital coupling strengths.
- **orb_pseudo** (list) – *Default*: `['s']` – List of strings, containing the orbital configuration for each pseudopotential.

3.11. `doubling_Hamiltonian`

Doubles the real space dimensions of the Hamiltonian, creating a supercell in any desired direction. Naturally, the number of wavefunctions in the Hamiltonian increases by a factor $2^{\mathbf{nx}} \times 2^{\mathbf{ny}} \times 2^{\mathbf{nz}}$. Doubling is performed one time in each dimension by default, and doubling can be suppressed by setting the argument for a dimension to 0.

Arguments for `doubling_Hamiltonian`:

- **nx** (int) – *Default*: `1` – Number of times to double the `x` dimension. If **nx** is set to 2 the resulting cell is 4 times larger in the `x` direction.
- **ny** (int) – *Default*: `1` – Number of times to double the `y` dimension.
- **nz** (int) – *Default*: `1` – Number of times to double the `z` dimension.

3.12. `topology`

The `topology` routine calculates various quantities along the AFLOW standard **k**-path. If the user generates a custom path with `bands`, prior to this routine, this path will be used when calculating the Z2 invariance and topological properties.

Arguments for `topology`:

- **eff_mass** (bool) – *Default*: `False` – Setting this flag `True` computes the Hamiltonian's second derivative along the **k**-path. The effective mass is calculated and saved to file with naming convention `effmass_IJ_S.dat`. The indices `I` and `J` are specified by the arguments **ipol** and **jpol**. Index `S` is specified by spin polarization at the DFT level.
- **Berry** (bool) – *Default*: `False` – Set `True` to calculate the Berry curvature along the **k**-path and writes results to file as `Omega_IJ.dat`. Here, `I` and `J` are indices of the Berry curvature (Ω_{ij}) and are specified by **ipol** and **jpol** respectively.
- **spin_Hall** (bool) – *Default*: `False` – Setting `True` calculates the spin Berry curvature (Ω^s_{ij}) along the **k**-path and writes the results to files `Omegaj_S_IJ.dat`. Here, the indices `I`, `J`, and `S` are specified by the arguments **ipol**, **jpol**, and **spol**. This routine automatically computes the Berry curvature, but no files for **Berry** are written unless its flag is explicitly set `True`.

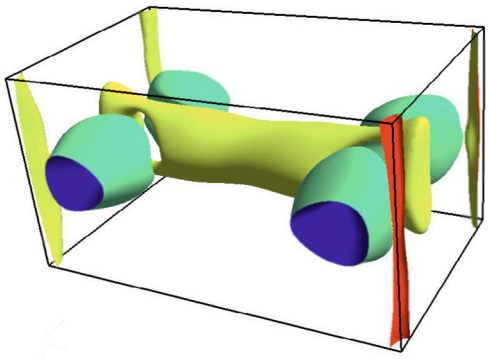

Fig. 1. Fermi surface of FeP calculated on a ultra-dense **k**-grid in PAOFLOW and visualized in FermiSurfer [21].
Source: For description of DFT calculations see Ref. [22].

- **spol** (integer) – *Default*: `None` – Spin polarization index of the spin Berry curvature calculation. This selects which component of the spin operator is used.
- **ipol** (integer) – *Default*: `None` – The index i in effective mass and (spin) Berry calculations.
- **jpol** (integer) – *Default*: `None` – The index j in effective mass and (spin) Berry calculations.

3.13. `pao_eigh`

The routine `pao_eigh` computes the eigenspectrum for the entire **k**-grid, saving the eigen-values and -vectors as new arrays in the `DataController` under keys 'E_k' and 'v_k' respectively. Some of the previously described functions compute the eigenvalues and eigenvectors along a path, such as `bands` and `topology`. This routine replaces values computed by such routines with a new set of eigenfunctions, running across the entire Brillouin zone.

No arguments are accepted when calling `pao_eigh`.

3.14. `trim_non_projectable_bands`

Removes eigenvalues and momenta which do not meet the projectability requirement set by `projectability` from respective data arrays. This routine should be called after `pao_eigh`, if such trimming is desired.

No arguments are accepted by `trim_non_projectable_bands`.

3.15. `fermi_surface`

Computes the bands with energies between **fermi_up** and **fermi_dw**. The results are saved in the NumPy npz format with naming convention `Fermi_surf_band_N_M.npz`, where `N` is the band index and `M` is the spin index. The Fermi surface is saved with resolution of the existing **k**-grid (see Fig. 1).

Arguments for `fermi_surface`:

- **fermi_up** (float) – *Default*: `1` – The upper energy bound for selecting bands. Bands within the range [**fermi_dw**, **fermi_up**] are included.
- **fermi_dw** (float) – *Default*: `1` – The lower energy bound for selecting bands.

3.16. gradient_and_momenta

The Hamiltonian's gradient is initially calculated in real space, as it takes a simpler form. Afterward, it is Fourier transformed back into reciprocal space. Thus, the Hamiltonian's **k**-space gradient is given by

$$\vec{\nabla}_{\mathbf{k}} \hat{H}(\mathbf{k}) = \sum_{\alpha} i\mathbf{R} \exp(i\mathbf{k}\cdot\mathbf{R})\, \hat{H}(\mathbf{R}) \tag{3}$$

where $\hat{H}(\mathbf{R})$ is the real space PAO matrix and $|\psi_n(\mathbf{k})\rangle = \exp(-i\mathbf{k}\cdot\mathbf{r})|u_n(\mathbf{k})\rangle$ are Bloch's functions [23]. The Hamiltonian's derivative is saved as a new array key 'dHksp' in the DataController.

Next, the momenta are computed from the Hamiltonian's gradient as

$$\begin{aligned}\mathbf{p}_{nm}(\mathbf{k}) &= \langle\psi_n(\mathbf{k})|\, \hat{p}\, |\psi_m(\mathbf{k})\rangle = \\ &= \langle u_n(\mathbf{k})|\, \frac{m_0}{\hbar} \vec{\nabla}_{\mathbf{k}} \hat{H}(\mathbf{k})\, |u_m(\mathbf{k})\rangle\end{aligned} \tag{4}$$

Additionally, the Hamiltonian's second derivative can be computed by setting the **band_curvature** argument to True.

Arguments for gradient_and_momenta:

- **band_curvature** (bool) – *Default*: False – Compute the Hamiltonian's second derivative, stored as an array in the DataController under the key 'd2Ed2k'.

3.17. adaptive_smearing

Generates the adaptive smearing parameters, stored in the DataController as 'deltakp', used to compute quantities on energy intervals, such as the density of states or spin Hall conductivity. Allowed adaptive smearing types are gaussian ('gauss'), Methfessel–Paxton ('m-p'), or None. Implementation of the Gaussian broadening parameters are described in Section 3 of Ref. [1].

Arguments for adaptive_smearing:

- **smearing** (string) – *Default*: 'gauss' – Method of broadening used to smooth the discrete sampling of quantities computed on energy intervals.

3.18. dos

Compute the density of states (DOS) and/or projected density of states (PDOS) within a user defined energy range. If this routine is called after adaptive_smearing, the 'deltakp' smearing parameter is used to smooth the DOS calculations.

Arguments for dos:

- **do_dos** (bool) – *Default*: True – Flag to control whether the DOS is computed.
- **do_pdos** (bool) – *Default*: True – Flag to control whether the PDOS is computed.
- **delta** (float) – *Default*: 0.01 – Width of the gaussian at each energy, used to smooth the dos curves. If it has been computed with the adaptive_smearing routine, 'deltakp' replaces this quantity.
- **emin** (float) – *Default*: -10 – Lower limit for the energy range considered.
- **emax** (float) – *Default*: 2 – Upper limit for the energy range considered.
- **ne** (int) – *Default*: 1000 – The number of points to evaluate within the energy range [emin,emax].

3.19. z2_pack

Writes the real space Hamiltonian to a dat file, for use with the Z2Pack code [24] (see Fig. 2).

Arguments for z2_pack:

- **fname** (string) — *Default*: 'z2pack_hamiltonian.dat' — Name for the dat file, written to PAOFLOW's output directory.

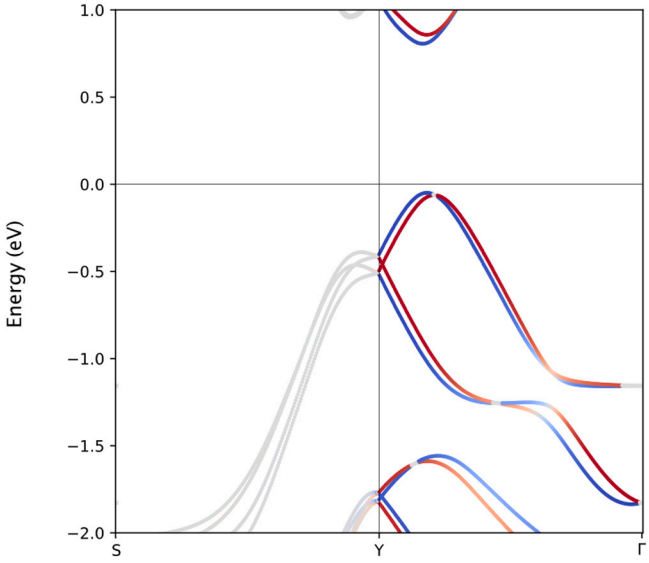

Fig. 2. Spin texture (S_z) of two-dimensional ferroelectric SnTe along high-symmetry lines (example08).
Source: See Ref. [25]. for details.

3.20. spin_texture

Compute spin texture as the spin operator's expectation value for each band and for each **k**-point:

$$\boldsymbol{\Omega}_n(\mathbf{k}) = \langle\psi_n(\mathbf{k})|\mathbf{S}_j|\psi_n(\mathbf{k})\rangle \tag{5}$$

$\mathbf{S}_j$ is the spin operator, and $|\psi_n(\mathbf{k})\rangle$ are the momentum space PAO wavefunctions for band index n. The spin texture is computed for bands which have values within energy range specified by **fermi_up** and **fermi_dw**. Calling this routine after bands results in the spin texture along the same **k**-path as bands, while calling it after pao_eigh computes spin texture across the entire BZ. Results are written to a file named spin-texture-bands.dat for bands along a path and to NumPy .npz format with naming convention spin_text_band_N.npz for bands across the BZ. Here, N is the band index, and each file contains spin texture computed on PAOFLOW's **k**-grid.

Arguments for spin_texture:

- **fermi_up** (float) – *Default*: 1 – The spin texture is computed only for bands which contain energies beneath this upper bound.
- **fermi_dw** (float) – *Default*: 1 – The spin texture is computed only for bands which contain energies above this lower bound.

3.21. anomalous_Hall

Calculating anomalous Hall conductivity (AHC, σ_{ij}) relies on accurate evaluation of the Berry curvature and requires a preliminary run of gradient_and_momenta. PAOFLOW implements a standard Kubo formula for evaluating the **k**-resolved Berry curvature [26]. Details are given in the previous paper and in the comprehensive reference by Gradhand et al. [27]. The AHC is computed with adaptive smearing, provided that the broadening parameters are calculated beforehand by adaptive_smearing.

Arguments for anomalous_Hall:

- **do_ac** (bool) – *Default*: False – Compute the magnetic circular dichroism (MCD) on the energy range [**emin**, **emax**].
- **emin** (float) – *Default*: -1 – The minimum energy in the range on which the AHC is computed.
- **emax** (float) – *Default*: 1 – The maximum energy in the range. 500 points are evaluated within the interval [**emin**, **emax**].

- **fermi_up** (float) – *Default*: 1 – Selects the upper energy bound for evaluating the Berry curvature.
- **fermi_dw** (float) – *Default*: -1 – Selects the lower energy bound for evaluating the Berry curvature.
- **a_tensor** (list) – *Default*: None – List of tensor elements to evaluate. For example, setting this argument to [[0,0], [1,2]] calculates the two components σ_{xx} and σ_{yz}. All 9 components are computed, if the argument is left as None.

3.22. spin_Hall

The spin Hall conductivity (SHC, σ_{ij}^{k}) for spin polarization along **k** and charge (spin) current along i (j), is computed in a similar manner to AHC [28]. Here, evaluation of the *spin* Berry curvature is performed with the spin operator and Hamiltonian gradient as ingredients. Again, the gradient_and_momenta routine is a prerequisite, and running adaptive_smearing beforehand controls the inclusion of broadening parameters in the calculation.

Arguments for spin_Hall:

- **twoD** (bool) – *Default*: False – Setting this flag True outputs the spin_Hall quantities in 2-dimensional units Ω^{-1}, removing any dependence on the sample height. It is assumed that the dimensions of interest are oriented in the x-y plane, and the slab is oriented along z.
- **do_ac** (bool) – *Default*: False – Compute the spin circular dichroism (SCD) on the energy range [**emin**, **emax**].
- **emin** (float) – *Default*: -1 – The minimum energy in the range on which the SHC and SCD are computed.
- **emax** (float) – *Default*: 1 – The maximum energy in the range. Again, 500 points are evaluated in the interval [**emin**, **emax**].
- **fermi_up** (float) – *Default*: 1 – Selects the upper energy bound for evaluating the spin Berry curvature.
- **fermi_dw** (float) – *Default*: -1 – Selects the lower energy bound for evaluating the spin Berry curvature.
- **s_tensor** (list) – *Default*: None – List of tensor elements to evaluate. To calculate σ_{xx}^{x} and σ_{xy}^{z} components use [[0,0,0],[0,1,2]]. If the argument is left as None, all 27 components are computed.

3.23. doping

Determine the chemical potential corresponding to a specified doping concentration and temperature range.

Arguments for doping:

- **tmin** (float) – *Default*: 300 – Minimum temperature for which to evaluate the chemical potential.
- **tmax** (float) – *Default*: 300 – Maximum temperature to compute chemical potential.
- **nt** (int) – *Default*: 1 – The number of temperatures to evaluate in the range [**tmin**, **tmax**].
- **delta** (float) – *Default*: 0.01 – Gaussian broadening width, used to smooth the density of states along the energy range. Doping calculation involves an integration over density of states and therefore includes a call to the dos module.
- **emin** (float) – *Default*: -1 – Lowest value of energy of the occupied bands.
- **emax** (float) – *Default*: 1 – At least the energy of the minimum of the conduction bands to obtain accurate results.
- **ne** (int) – *Default*: 1000 – Number of points in the energy grid.
- **doping_conc** (float) – *Default*: 0 – The doping concentration in carriers/cm^3 for which to compute the chemical potential. Specify negative value for n-type doping and positive value for p-type doping.
- **core_electrons** (int) – *Default*: 0 – If the total number of electrons in the lower energy bands is known, this value can be introduced here. In this case, **emin** does not have to be the lowest energy value of occupied bands but can instead be set above energies of the core bands, to speed up integration.
- **fname** (string) – *Default*: 'doping_' – Prefix for the output file containing doping versus temperature.

3.24. density

Calculate the electronic density on a real space grid, performed for silicon in Listing 4 and displayed in Fig. 3. Wavefunctions in **k**-space (produced by pao_eigh) are required as a prerequisite, and the PAO projections *must* be performed by PAOFLOW's projections method. This algorithm serves as a recipe for constructing the real space PAO wavefunctions. Although this is currently the only routine to utilize such construction, future versions of PAOFLOW will include other methods for computing spatially resolved quantities. The real space grid dimension defaults to 48x48x48 but can be specified with the optional arguments **nr1**, **nr2**, and **nr3**.

Arguments for density:

- **nr1** (int) – *Default*: 48 – Number of points in the first dimension of the real space grid, over which to compute the charge density.
- **nr2** (int) – *Default*: 48 – Number of points in the second dimension of the real space grid.
- **nr3** (int) – *Default*: 48 – Number of points in the third dimension of the real space grid.

Listing 4: main.py **- Density**

```
from PAOFLOW import PAOFLOW

pao = PAOFLOW.PAOFLOW(savedir='system.save')
pao.projections()
pao.projectability()
pao.pao_hamiltonian()
pao.pao_eigh()

pao.density(nr1=48, nr2=48, nr3=48)

pao.finish_execution()
```

3.25. transport

Calculate the transport properties, such as electrical conductivity, Seebeck coefficients, and thermal conductivity. The transport properties are computed in the constant relaxation time approximation, unless built-in or user defined τ models are provided. See example09 for detailed information on specifying models for the relaxation time τ.

Arguments for transport:

- **tmin** (float) – *Default*: 300 – Minimum temperature for which to evaluate transport properties.
- **tmax** (float) – *Default*: 300 – Maximum temperature to evaluate transport properties.
- **nt** (float) – *Default*: 1 – The number of temperatures to evaluate in the range [**tmin**,**tmax**].
- **emin** (float) – *Default*: 1 – Minimum value in the energy grid [**emin**,**emax**].
- **emax** (float) – *Default*: 10 – Maximum value of energy in the grid.
- **ne** (int) – *Default*: 1000 – Number of points in the energy range [**emin**,**emax**].
- **scattering_channels** (list) – *Default*: None – List of strings and/or TauModel objects containing the scattering models to be included in the calculation of τ.
- **scattering_weights** (list) – *Default*: None – Initial guess for the parameters a_{imp}, a_{ac}, a_{op} etc to be used for the fitting procedure if fit is set to True. The default behavior with this argument set to None is to use unity as every scattering weight.

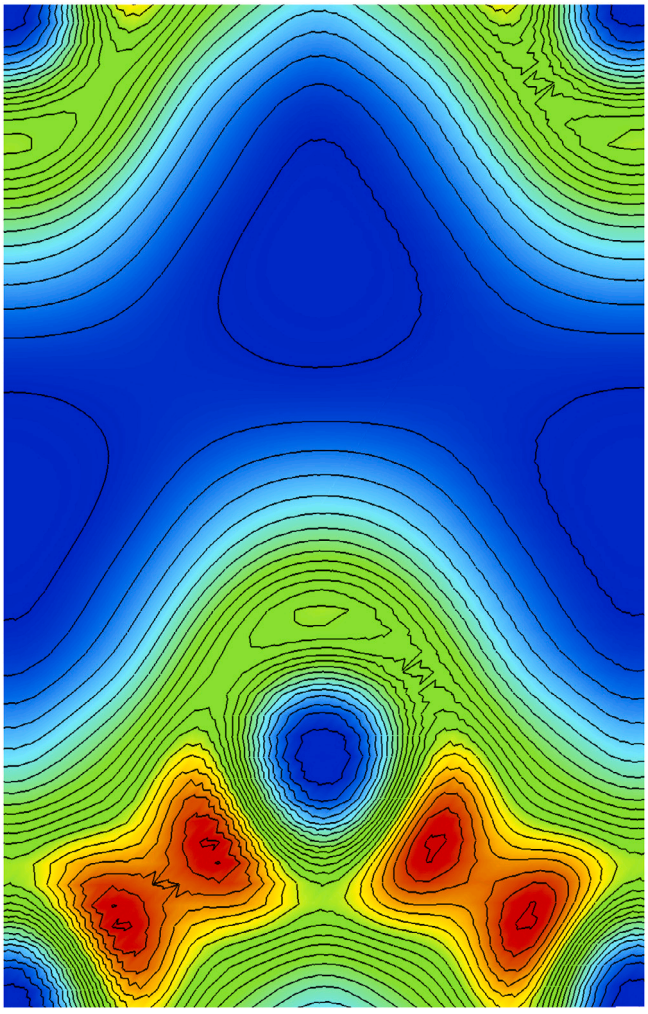

Fig. 3. Electronic density for diamond structure of silicon on the $\langle 1, 0, -1 \rangle$ plane cut, calculated from the real space PAO wavefunctions (`example01`).

- **tau_dict** (dict) – *Default*: `{}` – Dictionary of parameters required for the calculation of scattering models.
- **write_to_file** (bool) – *Default*: `True` – Controls the output of the several data fields produced by this routine. No files are written if the flag is set to `False`.
- **save_tensors** (bool) – *Default*: `False` – Setting this flag `True` stores the resulting electrical conductivity, Seebeck coefficient, and thermal conductivity to the data_arrays dictionary with respective keys 'sigma', 'S', and 'kappa'.

3.26. `find_weyl_points`

Perform a search for Weyl points within the first Brillouin zone. The search identifies Weyl point candidates by utilizing `Scipy`'s `minimize` function with the 'L-BFGS-B' algorithm.

Arguments for `find_weyl_points`:

- **symmetrize** (bool) – *Default*: `False` – Use QE symmetry operations to unfold equivalent **k**-points. If equivalent **k**-points are Weyl points, all such points are reported.
- **search_grid** (list) – *Default*: `[8, 8, 8]` – Dimensions of the grid on which the minimization routine is performed. Bands are Fourier interpolated on this grid to improve resolution.

3.27. `restart_dump`

PAOFLOW's computational state can be saved at any time with the `restart_dump` routine. Data is stored in the `json` format, and the naming convention for such files can be chosen with the **fname_prefix** argument. Each processor saves a file in the **workpath** directory with name `fname_prefix_N.json`, where `N` is the core's rank. For this reason, restarted calculations must be executed with the same number of cores. See Listing 5 for an example where the gradient and momenta are computed and dumped for reuse in another calculation.

Arguments for `restart_dump`:

- **fname_prefix** (string) – *Default*: `'paoflow_dump'` – The name prefix for the dump files, written to the working directory.

Listing 5: `main.py` **- Restart (dump)**

```
from PAOFLOW import PAOFLOW

pao = PAOFLOW.PAOFLOW(savedir='system.save')
pao.projections()
pao.projectability()
pao.pao_hamiltonian()
pao.pao_eigh()
pao.gradient_and_momenta()
pao.restart_dump(fname_prefix='pao')
```

3.28. `restart_load`

Recover PAOFLOW's calculation state from a previous run, saved to the json format with `restart_dump`. A restarted run must be executed with the same number of cores as the run which produced the dump files. Listing 6 provides example usage for `restart_load`.

Arguments for `restart_load`:

- **fname_prefix** (string) – *Default*: `'paoflow_dump'` – The name prefix for the dump files, written to the working directory.

Listing 6: `main.py` **- Restart (load)**

```
from PAOFLOW import PAOFLOW

pao = PAOFLOW.PAOFLOW(restart=True)
pao.restart_load(fname_prefix='pao')
pao.adaptive_smearing()
...
```

3.29. `finish_execution`

Conclude the PAOFLOW run and remove references to memory intensive quantities. Details about the execution are provided, such as run duration and total memory requirements. This routine should be called once all desired calculations are performed for a given PAOFLOW object, especially if the code continues to create other PAOFLOW Hamiltonians.

`finish_execution` accepts no arguments.

4. Tight-binding models

PAOFLOW is capable of generating a Hamiltonian from analytical TB models, such as the Kane–Mele or Slater–Koster models [29,30]. For each type of model, one needs to specify a few system properties, such as hopping parameters, lattice constant, etc. These parameters, including the label selecting the model to implement, should be initially stored in a dictionary, which is subsequently passed into PAOFLOW's constructor as the **model** argument. Once the Hamiltonian is constructed, PAOFLOW's class methods can be applied in the standard

way to compute any desired quantities. Two examples are provided in Listing 7 and 8, and more are provided in the `examples/` directory.

4.1. Cubium

Creates a Hamiltonian for a single atom in the simple-cubic geometry, containing one orbital per site. The hopping parameter is defined by including an entry in the parameters dictionary with key 't', and should have units of eV.

Required dictionary entries:

- *Key*: 'label' — Keyword identifier for the model: 'cubium', in this case. The labels are not case sensitive.
- *Key*: 't' — The hopping parameter for nearest neighbor interactions, in units of eV.

4.2. Cubium II

Creates a Hamiltonian for a single atom in the simple-cubic geometry, implementing the double band model with two orbitals per site. The hopping parameter and band gap energy are given by 't' and 'Eg' respectively.

Required dictionary entries:

- *Key*: 'label' — Keyword identifier for the model: 'cubium2'
- *Key*: 't' — The hopping parameter for nearest neighbor interactions.
- *Key*: 'Eg' — Band gap energy, in eV.

4.3. Graphene

A simple TB model for graphene, considering only nearest neighbor interactions. The hopping parameter is specified with parameter 't'. The lattice constant is taken as $a = 2.46$ Å, and lattice vectors are given in a standard form: $\mathbf{a_1} = a\langle 1, 0, 0\rangle$, $\mathbf{a_2} = a\langle \frac{1}{2}, \frac{\sqrt{3}}{2}, 0\rangle$, $\mathbf{a_3} = a\langle 0, 0, 10\rangle$.

Required dictionary entries:

- *Key*: 'label' — Keyword identifier for the model: 'graphene'
- *Key*: 't' — The hopping parameter for nearest neighbor interactions.

4.4. Kane–Mele model

Constructs a Kane–Mele Hamiltonian for graphene. The first nearest neighbors are handled in the standard manner, with hopping parameter 't'. Second nearest neighbors are treated with spin dependent amplitude, characterized by the parameter 'soc_par'. See an example in Listing 7.

Required dictionary entries:

- *Key*: 'label' — Keyword identifier for the model: 'kane_mele'.
- *Key*: 'alat' — The lattice parameter, a. The lattice vectors are the same as in Section 4.3.
- *Key*: 't' — The hopping parameter for nearest neighbor interactions.
- *Key*: 'soc_par' — The spin–orbit coupling parameter for second nearest neighbor interactions.

Listing 7: `main.py` **- Kane–Mele model**

```
from PAOFLOW import PAOFLOW

model = {'label':'Kane_Mele', 't':1.0,
         'soc_par':0.1, 'alat':1.0}

paoflow = PAOFLOW.PAOFLOW(model=model,
                  outputdir='./kane_mele')
...
```

4.5. Slater–Koster model

A generalized Slater–Koster TB model in the two-center approximation, that includes only `s` and `p` orbitals of first nearest neighbors. The user must specify the lattice vectors, the atomic positions, the included orbitals for each atom, and the hopping parameters. See Listing 8 for further details.

Required dictionary entries:

- *Key*: 'label' — Keyword identifier for the model: 'slater_koster'.
- *Key*: 'a_vectors' — A numpy array containing the three primitive lattice vectors.
- *Key*: 'atoms' — A dictionary with entries specifying atomic information for each atom. Dictionary keys label the atoms numerically with strings (e.g. the first atom has key '0'), and the corresponding values are dictionaries with information about the atom. The species, position (in crystal coordinates), and string identifier for each represented orbital should be saved in the atomic dictionary with respective keys: 'name', 'tau', and 'orbitals'. The name is simply a string, the atomic position is a 3-vector, and orbitals are a list of strings denoting the orbitals belonging to each atom.
- *Key*: 'hoppings' — A dictionary defining different hopping parameters, in eV. The Slater–Koster hopping parameters should be labeled: sss, sps, pps, and ppp.

Listing 8: `main.py` **- Slater-Koster model**

```
from PAOFLOW import PAOFLOW
import numpy as np

model = {'label':'slater_koster'}

avecs = np.array([[.5,.5,0],
                  [.5,0,.5],
                  [0,.5,.5]])

atoms = {'0' :
         {'name':'Si',
          'tau':[0,0,0],
          'orbitals':['s','px','py','pz']},
         '1' :
         {'name':'Si',
          'tau':[.25,.25,.25],
          'orbitals':['s','px','py','pz']}}

hops = {'sss':-2.36233, 'sps': 1.86401,
        'pps': 2.85882, 'ppp':-0.94687}

model['a_vectors'] = avecs
model['atoms'] = atoms
model['hoppings'] = hops

paoflow = PAOFLOW.PAOFLOW(model=model,
                  outputdir='./kane_mele')
...
```

5. Scattering models

PAOFLOW supports a diverse set of scattering effects by allowing users to implement temperature and energy dependent models for the relaxation time parameter τ. Functional models are defined with the `TauModel` class. There are many built-in models, which only require the

specification of empirical constants, and users can define new models to pass into PAOFLOW directly. A Python dictionary containing required parameters for any selected built-in models must be passed to the *transport* routine as the **tau_dict** argument (see Listing 9). Table 1 details the various constant parameters and their key strings for dictionary entries. Models for τ models are necessarily dependent on two quantities, the temperature and the Hamiltonian's energy eigenvalues. Other varying parameters can be supplied to `TauModels` through the params dictionary. As such, a python function accepting three arguments (the temperature, the energy, and the parameters dictionary) is the required format when constructing a custom `TauModel` object. The τ for each included model are computed, by evaluating the `TauModel` functions. Then, the τs are harmonically summed to obtain the effective τ for all scattering channels. The functional form for a `TauModel` is presented in Listing 9 and a usage case in `example10`.

Listing 9: Define TauModel

```
from PAOFLOW.defs.TauModel import TauModel
from PAOFLOW import PAOFLOW

# Compute quantites required for transport
pao = PAOFLOW.PAOLFOW(savedir='system.save')
pao.projections()
pao.projectability()
pao.pao_hamiltonian()
pao.pao_eigh()
pao.gradient_and_momenta()
pao.adaptive_smearing()

# Define the functional model
#  for acoustic scattering.
rho = 5.3e3; v = 5.2e3; D = 7*1.6e-19
m = .7*9.11e-13; h_bar = 6.58e-16;
ac_const = 2*np.pi*h_bar**4*rho*v**2
ac_const /= ((2*m)**(3/2)*D**2)
def acoustic_scat ( temp, ene, params ):
  return ac_const/(temp*np.sqrt(ene))

# Define TauModel object
ac_model = TauModel(function=acoustic_scat)

channels = [ac_model, 'optical']

# Define parameters for built-in models
tau_params = {'ms':0.7', 'hwlo':[0.03536]
              'eps_inf':11.6, 'eps_0':13.5}

pao.transport(scattering_channels=channels,
              tau_dict=tau_params)

pao.finish_execution()
```

5.1. Charged impurity scattering

In order to include the effect of electron scattering from impurities, include 'impurity' in the list **scattering_channels** [31,32]. This calculates the relaxation time as

$$\tau_{im}(E,T) = \frac{E^{\frac{3}{2}}\sqrt{2m^*}4\pi\varepsilon^2}{(log(1+\frac{1}{x}) - \frac{1}{1+x})\pi n_I Z_I^2 e^4} \tag{6}$$

$$x = \frac{E}{k_B T} \tag{7}$$

Required parameters are $m^*, \varepsilon_0, \varepsilon_{\inf}, n_I, Z_I$.

5.2. Acoustic scattering

In order to include the effect of electron scattering from acoustic phonons, include 'acoustic' in the list **scattering_channels**. This calculates the relaxation time according to [31,32]

$$\tau_{ac}(E,T) = \frac{2\pi\hbar^4\rho v^2}{(2m^*)^{\frac{3}{2}} k_B T D_{ac}^2 \sqrt{E}} \tag{8}$$

Required parameters are m^*, ρ, v, D_{ac}.

5.3. Optical scattering

In order to include the effect of electron scattering from optical phonons, include 'optical' in the list **scattering_channels**. This calculates the relaxation time following [31,32]

$$\tau_{op}(E,T) = \frac{\sqrt{2k_B T}\pi x_o \hbar^2 \rho}{(m^*)^{\frac{3}{2}} D_{op}^2 N_{op}\sqrt{x+x_o} + (N_{op}+1)\sqrt{x-x_o}} \tag{9}$$

$$N_{op} = \frac{1}{\exp\frac{\hbar\omega_l}{k_B T} - 1}, \qquad x = \frac{E}{k_B T}, \qquad x_o = \frac{\hbar\omega_l}{k_B T}, \tag{10}$$

Required parameters are $m^*, \rho, \omega_l, D_{op}, N_{op}$.

5.4. Polar acoustic scattering

In order to include the effect of electron scattering from acoustic phonons in polar materials, include 'polar_acoustic' in the list **scattering_channels**. This calculates the relaxation time following [31,32]

$$\tau_{pac}(E,T) = \frac{\sqrt{2E}2\pi\varepsilon^2\hbar^2\rho v^2}{p^2 e^2\sqrt{m^* k_B T}} \times\left[1 - \frac{\epsilon_o}{2E}\log(1+4\frac{E}{\epsilon_o}) + \frac{1}{1+4\frac{E}{\epsilon_o}}\right] \tag{11}$$

Required parameters are $m^*, \varepsilon_0, \varepsilon_{\inf}, \rho, v, p$.

5.5. Polar optical scattering

In order to include the effect of electron scattering from optical phonons in polar materials, include 'polar_optical' in the list **scattering_channels**. This calculates the relaxation time based on [31–33].

$$\tau_{pop}(E,T) = \sum_i \frac{Z(E,T,\omega_i^l)E^{\frac{3}{2}}}{C(E,T,\omega_i^l) - A(E,T,\omega_i^l) - B(E,T,\omega_i^l)} \tag{12}$$

$$A(E,T,\omega_l) = n(\omega_l+1)\frac{f_0(E+\hbar\omega_l)}{f_0(E)}[(2E+\hbar\omega_l) sinh^{-1}(\frac{E}{\hbar\omega_l})^{\frac{1}{2}} - [E(E+\hbar\omega_l)]^{\frac{1}{2}}] \tag{13}$$

$$B(E,T,\omega_l) = \theta(E-\hbar\omega_l)n(\omega_l)\frac{f_0(E-\hbar\omega_l)}{f_0(E)}[(2E-\hbar\omega_l) cosh^{-1}(\frac{E}{\hbar\omega_l})^{\frac{1}{2}} - [E(E-\hbar\omega_l)]^{\frac{1}{2}}] \tag{14}$$

$$C(E,T,\omega_l) = 2E[n(\omega_l+1)\frac{f_0(E+\hbar\omega_l)}{f_0(E)} sinh^{-1}(\frac{E}{\hbar\omega_l})^{\frac{1}{2}} + \theta(E-\hbar\omega_l)n(\omega_l)\frac{f_0(E-\hbar\omega_l)}{f_0(E)} cosh^{-1}(\frac{E}{\hbar\omega_l})^{\frac{1}{2}}] \tag{15}$$

Table 1
Symbols, units, and corresponding key in **tau_dict** for the parameters required in various scattering models.

Parameter	Symbol	Units	Key
Mass density	ρ	kg/m^3	rho
Low freq. dielectric constant	ϵ_0	–	eps_0
High freq. dielectric constant	ϵ_∞	–	eps_inf
Acoustic velocity	v	m/s	v
Effective mass ratio	m^*	–	ms
Acoustic deformation potential	D_{ac}	eV	D_ac
Optical deformation potential	D_{op}	eV	D_op
Optical phonon energy	$\hbar\omega_l$	eV	hwlo
Number of impurities	n_I	cm^{-3}	nI
Charge on impurity	Z_I	–	Zi
Piezoelectric constant	p	C/m^2	piezo

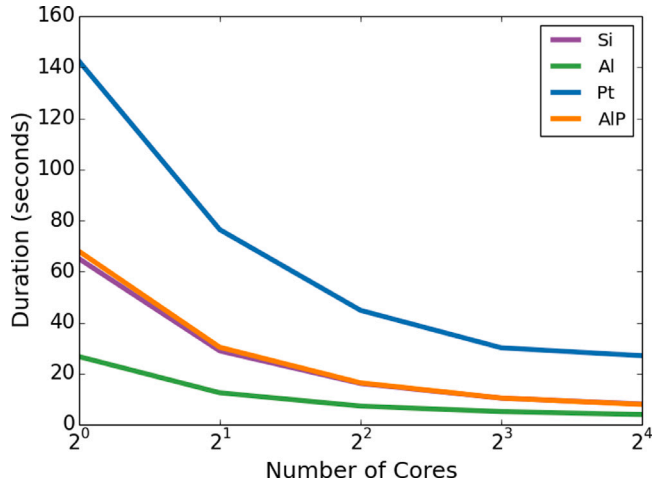

Fig. 4. Selected examples (see `examples/` on GitHub) performed on an increasing number of processors. Parallelized routines provide run time scaling nearly proportional to the number of cores used in a calculation, closely approaching the speed increase limit of Amdahl's Law.

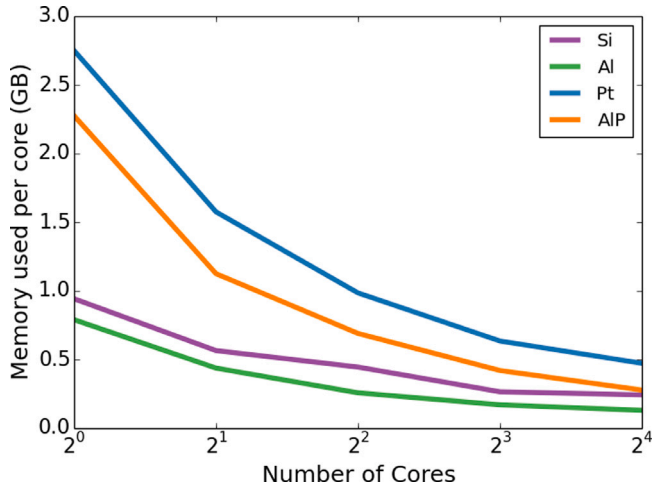

Fig. 5. Memory scaling per core (in GB), for selected examples. An increasing core count reduces the memory requirements per processor.

$$Z(\omega_l) = \frac{2}{W_0(\hbar\omega_l)^{\frac{1}{2}}},\ W_0(\omega_l) = \frac{e^2\sqrt{2m^*\omega_l}\varepsilon^{-1}}{4\pi\hbar^{\frac{3}{2}}} \tag{16}$$

Required parameters are $m^*, \varepsilon_0, \varepsilon_{\inf}, \omega_{LO}$.

5.6. Effective scattering time

$$\frac{1}{\tau_{total}(E,T)} = \frac{1}{\tau_{im}(E,T)} + \frac{1}{\tau_{ac}(E,T)} + \cdots \tag{17}$$

The total scattering time is calculated as a harmonic sum of the specified scattering mechanisms, evaluated for each energy and temperature, Eq. (17). The effective scattering time τ_{total} is computed for each energy and temperature during the execution of the `transport` routine.

6. Performance

Benchmark performance tests reveal excellent scaling for massively parallelized calculations. PAOFLOW exploits parallelization over bands whenever possible, primarily in the calculation of gradients. However, most routines are parallelized across the **k**-point mesh or path. PAOFLOW also possesses excellent scaling of memory requirements, in parallel runs. Increasing the number of processors can reduce the memory load on each processor, as many of the large arrays are distributed evenly among the cores. Because quantities must be stored on every processor, the *total* memory requirements are slightly higher for increasing number of cores. Performance is analyzed on a Dell PowerEdge R730 server with two 2.4 GHz Intel Xeon E5-2680 v4 fourteen-core processors, and results for several examples are presented in Figs. 4 and 5. PAOFLOW demonstrates excellent scaling properties on manycore systems and possesses massively parallel capabilities.

7. Conclusions

PAOFLOW provides a lightweight, robust tool for efficient materials and Hamiltonian analysis. Continuous development of the package has streamlined its functionality and enabled many new tools for effectively characterizing the electronic properties of solids. The updated framework offers an ideal tool for high throughput condensed matter simulations and generation for materials genomics [34].

CRediT authorship contribution statement

Frank T. Cerasoli: Writing, editing, Software development. **Andrew R. Supka:** Software development. **Anooja Jayaraj:** Software development, Writing. **Marcio Costa:** Software development. **Jagoda Sławińska:** Writing, Editing, Software development. **Stefano Curtarolo:** Software development, Reviewal. **Marco Fornari:** Software development, Reviewal. **Davide Ceresoli:** Software development. **Marco Buongiorno Nardelli:** Project Management, Software development, Reviewal.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data generated for this manuscript's production is available on `GitHub` (https://github.com/marcobn/PAOFLOW), provided within the various examples directories outlined in the text above.

Acknowledgments

We are grateful for computational resources provided by the High Performance Computing Center at the University of North Texas and the Texas Advanced Computing Center at the University of Texas, Austin. The members of the AFLOW Consortium (http://www.aflow.org) acknowledge support by DOD-ONR (N00014-13-1-0635, N00014-11-1-0136, N00014-15-1-2863). The authors also acknowledge Duke University Center for Materials Genomics, United States.

References

[1] M. Buongiorno Nardelli, F. Cerasoli, M. Costa, S. Curtarolo, R. Gennaro, M. Fornari, L. Liyanage, A. Supka, H. Wang, PAOFLOW: A utility to construct and operate on ab initio Hamiltonians from the projections of electronic wavefunctions on atomic orbital bases, including characterization of topological materials, Comput. Mater. Sci. 143 (2018) 462–472, http://dx.doi.org/10.1016/j.commatsci.2017.11.034.

[2] L.A. Agapito, S. Ismail-Beigi, S. Curtarolo, M. Fornari, M. Buongiorno Nardelli, Accurate tight-binding Hamiltonian matrices from ab initio calculations: Minimal basis sets, Phys. Rev. B 93 (3) (2016) 035104–035109.

[3] L.A. Agapito, M. Fornari, D. Ceresoli, A. Ferretti, S. Curtarolo, M. Buongiorno Nardelli, Accurate tight-binding Hamiltonians for two-dimensional and layered materials, Phys. Rev. B 93 (12) (2016) 125137–125138.

[4] O.L. McHugh, W.F. Goh, M. Gradhand, D.A. Stewart, Impact of impurities on the spin Hall conductivity in β-W, Phys. Rev. Mater 4 (2020) 094404.

[5] Vu Thi Ngoc Huyen, Yuki Yanagi, Michi-To Suzuki, Spin and anomalous hall effects emerging from topological degeneracy in Dirac fermion system CuMnAs, 2021, arXiv:2104.13704.

[6] D.-F. Shao, G. Gurung, S.-H. Zhang, E.Y. Tsymbal, Dirac nodal line metal for topological antiferromagnetic spintronics, Phys. Rev. Lett. 122 (2019) 077203, http://dx.doi.org/10.1103/PhysRevLett.122.077203.

[7] T. Nan, C. Quintela, J. Irwin, G. Gurung, D. Shao, J. Gibbons, N. Campbell, K. Song, S.-Y. Choi, L. Guo, R. Johnson, P. Manuel, R. Chopdekar, I. Hallsteinsen, T. Tybell, P. Ryan, J.-W. Kim, Y. Choi, P. Radaelli, D. Ralph, E. Tsymbal, M. Rzchowski, C. Eom, Controlling spin current polarization through non-collinear antiferromagnetism, Nature Commun. 11 (1) (2020) 4671.

[8] G. Gurung, D.-F. Shao, T.R. Paudel, E.Y. Tsymbal, Anomalous hall conductivity of noncollinear magnetic antiperovskites, Phys. Rev. Mater 3 (2019) 044409, http://dx.doi.org/10.1103/PhysRevMaterials.3.044409.

[9] T. Hagiwara, K. Suekuni, P. Lemoine, A.R. Supka, R. Chetty, E. Guilmeau, B. Raveau, M. Fornari, M. Ohta, R. Al Rahal Al Orabi, H. Saito, K. Hashikuni, M. Ohtaki, Key role of d0 and d10 cations for the design of semiconducting colusites: Large thermoelectric ZT in Cu26Ti2Sb6S32 compounds, Chem. Mater. 33 (9) (2021) 3449–3456, http://dx.doi.org/10.1021/acs.chemmater.1c00872.

[10] V. Pavan Kumar, P. Lemoine, V. Carnevali, G. Guélou, O.I. Lebedev, P. Boullay, B. Raveau, R. Al Rahal Al Orabi, M. Fornari, C. Prestipino, D. Menut, C. Candolfi, B. Malaman, J. Juraszek, E. Guilmeau, Ordered sphalerite derivative Cu5Sn2S7: a degenerate semiconductor with high carrier mobility in the Cu–Sn–S diagram, J. Mater. Chem. A 9 (2021) 10812–10826, http://dx.doi.org/10.1039/D1TA01615F.

[11] A.R. Supka, T.E. Lyons, L. Liyanage, P. D'Amico, R. Al Rahal Al Orabi, S. Mahatara, P. Gopal, C. Toher, D. Ceresoli, A. Calzolari, S. Curtarolo, M. Buongiorno Nardelli, M. Fornari, AFLOWpi: A minimalist approach to high-throughput ab initio calculations including the generation of tight-binding hamiltonians, Comput. Mater. Sci. 136 (2017) 76–84.

[12] S. Curtarolo, W. Setyawan, G.L.W. Hart, M. Jahnatek, R.V. Chepulskii, R.H. Taylor, S. Wang, J. Xue, K. Yang, O. Levy, M.J. Mehl, H.T. Stokes, D.O. Demchenko, D. Morgan, AFLOW: An automatic framework for high-throughput materials discovery, Comput. Mater. Sci. 58 (C) (2012) 218–226.

[13] S. Curtarolo, W. Setyawan, S. Wang, J. Xue, K. Yang, R.H. Taylor, L.J. Nelson, G.L.W. Hart, S. Sanvito, M. Buongiorno Nardelli, N. Mingo, O. Levy, AFLOWLIB.ORG: A distributed materials properties repository from high-throughput ab initio calculations, Comput. Mater. Sci. 58 (C) (2012) 227–235.

[14] P. Giannozzi, O. Andreussi, T. Brumme, O. Bunau, M.B. Nardelli, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, M. Cococcioni, N. Colonna, I. Carnimeo, A. Dal Corso, S. De Gironcoli, P. Delugas, R. DiStasio Jr, A. Ferretti, A. Floris, G. Fratesi, G. Fugallo, R. Gebauer, U. Gerstmann, F. Giustino, T. Gorni, J. Jia, M. Kawamura, H.-Y. Ko, A. Kokalj, E. Kucukbenli, M. Lazzeri, M. Marsili, N. Marzari, F. Mauri, N. Nguyen, H.-V. Nguyen, A.O. de-la Roza, L. Paulatto, S. Poncé, D. Rocca, R. Sabatini, B. Santra, M. Schlipf, A. Seitsonen, A. Smogunov, I. Timrov, T. Thonhauser, P. Umari, N. Vast, X. Wu, S. Baroni, Advanced capabilities for materials modelling with quantum ESPRESSO, J. Phys.: Condensed Matter 29 (2017) 465901.

[15] L. Agapito, S. Curtarolo, M. Buongiorno Nardelli, Reformulation of DFT+U as a pseudo-hybrid hubbard density functional for accelerated materials discovery, Phys. Rev. X 5 (1) (2015) 011006.

[16] P. Gopal, M. Fornari, S. Curtarolo, L.A. Agapito, L.S.I. Liyanage, M. Buongiorno Nardelli, Improved predictions of the physical properties of Zn- and cd-based wide band-gap semiconductors: A validation of the ACBN0 functional, Phys. Rev. B 91 (24) (2015) 245202.

[17] L.A. Agapito, A. Ferretti, A. Calzolari, S. Curtarolo, M. Buongiorno Nardelli, Effective and accurate representation of extended Bloch states on finite Hilbert spaces, Phys. Rev. B 88 (16) (2013) 165127.

[18] L.A. Agapito, S. Curtarolo, M. Buongiorno Nardelli, Reformulation of DFT+U as a pseudohybrid hubbard density functional for accelerated materials discovery, Phys. Rev. X 5 (2015) 011006, http://dx.doi.org/10.1103/PhysRevX.5.011006.

[19] C. Toher, C. Oses, D. Hicks, E. Gossett, F. Rose, P. Nath, D. Usanmaz, D.C. Ford, E. Perim, C.E. Calderon, J.J. Plata, Y. Lederer, M. Jahnátek, W. Setyawan, S. Wang, J. Xue, K. Rasch, R.V. Chepulskii, R.H. Taylor, G. Gomez, H. Shi, A.R. Supka, R.A.R.A. Orabi, P. Gopal, F.T. Cerasoli, L. Liyanage, H. Wang, I. Siloi, L.A. Agapito, C. Nyshadham, G.L.W. Hart, J. Carrete, F. Legrain, N. Mingo, E. Zurek, O. Isayev, A. Tropsha, S. Sanvito, R.M. Hanson, I. Takeuchi, M.J. Mehl, A.N. Kolmogorov, K. Yang, P. D'Amico, A. Calzolari, M. Costa, R.D. Gennaro, M.B. Nardelli, M. Fornari, O. Levy, S. Curtarolo, The AFLOW fleet for materials discovery, 2017, arXiv:1712.00422.

[20] M. Graf, P. Vogl, Electromagnetic fields and dielectric response in empirical tight-binding theory, Phys. Rev. B 51 (8) (1995) 4940.

[21] M. Kawamura, FermiSurfer: Fermi-surface viewer providing multiple representation schemes, Comput. Phys. Comm. 239 (2019) 197–203.

[22] D. Campbell, J. Collini, J. Sławińska, C. Autieri, L. Wang, K. Wang, B. Wilfong, Y. Eo, P. Neves, D. Graf, E. Rodriguez, N. Butch, M. Buongiorno Nardelli, J. Paglione, Topologically driven linear magnetoresistance in helimagnetic FeP, Npj Quantum Mater 6 (1) (2021) 38.

[23] P. D'Amico, L. Agapito, A. Catellani, A. Ruini, S. Curtarolo, M. Fornari, M.B. Nardelli, A. Calzolari, Accurate *ab initio* tight-binding Hamiltonians: effective tools for electronic transport and optical spectroscopy from first principles, Phys. Rev. B 94 (2016) 165166.

[24] D. Gresch, G. Autès, O.V. Yazyev, M. Troyer, D. Vanderbilt, B.A. Bernevig, A.A. Soluyanov, Z2Pack: Numerical implementation of hybrid wannier centers for identifying topological materials, Phys. Rev. B 95 (2017) 075146.

[25] J. Sławińska, F.T. Cerasoli, P. Gopal, M. Costa, S. Curtarolo, M.B. Nardelli, Ultrathin SnTe films as a route towards all-in-one spintronics devices, 2D Materials 7 (2) (2020) 025026.

[26] Y. Yao, L. Kleinman, A.H. MacDonald, J. Sinova, T. Jungwirth, D.-s. Wang, E. Wang, Q. Niu, First principles calculation of anomalous Hall conductivity in ferromagnetic bcc Fe, Phys. Rev. Lett. 92 (3) (2004).

[27] M. Gradhand, D.V. Fedorov, F. Pientka, P. Zahn, I. Mertig, B.L. Györffy, First-principle calculations of the berry curvature of Bloch states for charge and spin transport of electrons, J. Phys.: Condens. Matter 24 (21) (2012) 213202–213224.

[28] G. Guo, S. Murakami, T. Chen, N. Nagaosa, Intrinsic spin Hall effect in platinum: First-principles calculations, Phys. Rev. Lett. 100 (9) (2008) 096401–096404.

[29] J. Slater, G. Koster, Simplified LCAO method for the periodic potential problem, Phys. Rev. 94 (1954) 1498–1524, http://dx.doi.org/10.1103/PhysRev.94.1498.

[30] C. Kane, E. Mele, Quantum spin Hall effect in graphene, Phys. Rev. Lett. 95 (22) (2005) http://dx.doi.org/10.1103/physrevlett.95.226801.

[31] C. Jacoboni, Theory of Electron Transport in Semiconductors: A Pathway from Elementary Physics To Nonequilibrium Green Functions, vol. 165, Springer Science & Business Media, 2010.

[32] R. Farris, M.B. Maccioni, A. Filippetti, V. Fiorentini, Theory of thermoelectricity in Mg3Sb2 with an energy- and temperature-dependent relaxation time, J. Phys.: Condens. Matter 31 (6) (2018) 065702, http://dx.doi.org/10.1088/1361-648x/aaf364.

[33] B. Ridley, Polar-optical-phonon and electron-electron scattering in large-bandgap semiconductors, J. Phys.: Condens. Matter 10 (30) (1998) 6717.

[34] W. Setyawan, S. Curtarolo, High-throughput electronic band structure calculations: challenges and tools, Comput. Mater. Sci. 49 (2010) 299–312.