





tightbinder: A Python package for semi-empirical tight-binding models of crystalline and disordered solids

Alejandro José Uría-Álvarez ¹ and Juan José Palacios ^{1,2}

¹ Departamento de Física de la Materia Condensada, Universidad Autónoma de Madrid, 28049 Madrid, Spain ² Instituto Nicolás Cabrera, Condensed Matter Physics Centre (IFIMAC), 28049 Madrid, Spain  Corresponding author

DOI: [10.21105/joss.05810](https://doi.org/10.21105/joss.05810)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Antonia Mey](#)  

Reviewers:

- [@yw-fang](#)
- [@mbkumar](#)

Submitted: 28 July 2023

Published: 01 February 2024

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)).

Summary

tightbinder is a Python package for Slater-Koster, semi-empirical tight-binding calculations of the electronic structure of solids. Tight-binding models are ubiquitous in condensed matter physics, since they provide an inexpensive description of electrons in materials. Although the package can be used in principle for any kind of material (metals and insulators), it originates in the context of topological phases of matter. Since the prediction of topological insulators (Kane & Mele, 2005), there has been a huge effort understanding and characterizing topological materials, resulting in a complete classification of any crystalline system (Vergniory et al., 2019). However, not so much is known in the context of disordered solids. This is the aim of the library: to enable numerical studies of crystalline and disordered materials to identify possible topological systems where the usual techniques are not useful. In any case, it also serves as a general purpose tight-binding framework, due to the modular approach taken in its construction.

Statement of need

The determination of the band structure of a solid is the starting point for any calculation in condensed matter physics. This amounts to determining the matrix elements $t_{ij}^{\alpha\beta}$ of the electronic Hamiltonian:

$$H = \sum_{ij,\alpha\beta} t_{ij}^{\alpha\beta} c_{i\alpha}^\dagger c_{j\beta}$$

where the indices i, j run over atomic positions, and the indices α, β run over orbitals. $c_{i\alpha}^\dagger$ ($c_{i\alpha}$) are creation (annihilation) operators of electrons at atom i and orbital α . There exist several techniques to address this problem, varying in degrees of sophistication and scope. The most established method is density-functional theory (DFT) (Jones, 2015), which provides an accurate description of the electronic structure, usually at the cost of slower computations. Tight-binding models are as equally popular since they constitute a quick and inexpensive way to model systems, although by construction, they are restricted to simpler, effective description of the bands. Slater-Koster tight-binding models (Slater & Koster, 1954) provide a middle ground since they allow to give a more accurate description of the material based on empirical considerations, while still being simple to compute. Then, supposed that the Slater-Koster model captures the key features of the material, it can be used instead of DFT to describe the solid, as long as the desired properties depend on those relevant features. In general, this approach allows for a qualitative exploration of the properties of materials, while one should look for first principles calculations when seeking quantitative results.

Currently, there are several tight-binding packages available, such as PyBinding (Moldovan et al., 2020), Pyqula, PythTB, Kwant (Groth et al., 2014) and PySKTB (Radha, 2020) as well as \mathbb{Z}_2 Pack (Gresch et al., 2017) for the computation of topological invariants. Of those libraries, only PySKTB was designed to build Slater-Koster models, while the rest require specifying directly the hopping amplitudes. On top of that, they are usually oriented towards crystalline structures, lacking tools for the description of disorder. `tightbinder` provides all the standard functionality expected from a tight-binding code, focusing specifically on Slater-Koster models, which can be used to describe realistic amorphous materials. It also provides tools for the topological characterization of solids, similar to those of \mathbb{Z}_2 Pack but integrated within the API. These features give the library its own identity within the landscape of tight-binding frameworks, not necessarily competing but providing alternative tools and a different perspective.

Features

The band structure is the spectrum of the Hamiltonian of the system, which is obtained by computing and diagonalizing its matrix representation. Therefore, the library is mainly built using linear algebra operations from the common Python libraries for scientific computing, i.e. NumPy (Harris et al., 2020), SciPy (Virtanen et al., 2020) and matplotlib (Hunter, 2007) for visualization of the results. `tightbinder` focuses on providing the necessary routines and classes to build, modify and compute properties of empirical tight-binding models. The main features of the library are:

- Determination of Slater-Koster tight-binding models with matrix elements involving up to d orbitals, with intraatomic spin-orbit coupling. One can specify hoppings up to the n th-nearest neighbours, and use atoms from different chemical species with different numbers of electrons.
- Configuration file based description of the model of the solid: Using a standardized format for configuration files, one can specify all the relevant parameters of the model, from the lattice vectors to the Slater-Koster hopping amplitudes.
- Two main classes defined, one to describe crystalline Slater-Koster models (`SlaterKoster`), and another one for amorphous solids (`AmorphousSlaterKoster`). There are also predefined models, and the possibility of defining custom models which inherit from a base `System` class.
- Methods and routines to modify systems: once they are built, there are methods to modify the size or the boundaries of the solid, as well as routines to introduce different forms of disorder and external fields.
- Topology identification: Computation of the \mathbb{Z}_2 invariant of time-reversal topological insulators and of the entanglement spectrum from a specified cut of the system.
- Computation of observables from the eigenstates of the system, e.g. bands, expected value of the spin, density of states (also available using the kernel polynomial method), localization. There are plotting routines available for the different quantities.
- Fitting of the Slater-Koster parameters (or any user-defined model parameter) to reproduce some given energy bands, usually from DFT calculations.
- Transport calculations in two-terminal devices based on the non-equilibrium Green's function formalism and the Landauer formula.

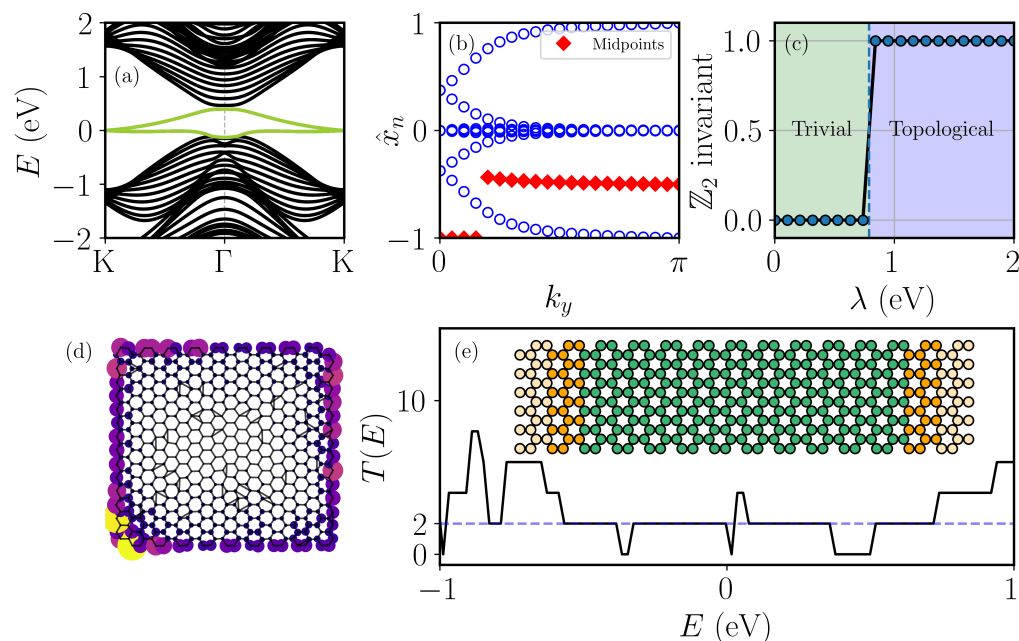


Figure 1: Characterization of Bi(111) with the library: (a) Band structure of a zigzag nanoribbon, with the edge bands highlighted in green. (b) Evolution of the Wannier charge centers (WCC). (c) The topological invariant can be obtained algorithmically from the WCC, allowing to compute the topological phase diagram as a function of the spin-orbit coupling. (d) Probability density of an edge state. (e) Transmission as a function of energy on an armchair nanoribbon.

The basic workflow starts with the preparation of a configuration file, where we set all the parameters relative to the material we want to describe. This is, the crystallographic information and then the details of the Slater-Koster model, which imply specifying which orbitals participate for each chemical species, and the corresponding Slater-Koster amplitudes. With the configuration prepared, the model is initialized simply passing the parsed configuration to the class constructor. From here, one can perform transformations of the base model, or directly obtain its spectrum and then perform some postprocessing. In fig. 1 we illustrate the results of this process, where we declared a configuration file for Bi(111) and then used it to explore the topological nature of the material from different quantities. `tightbinder` has already been valuable for one previous work (Uría-Álvarez et al., 2022), and continues to be used in the research of topological amorphous materials. We hope that more researchers will benefit from the package in their study of topological disordered solids.

The library provides a stable API, but is still under development to incorporate new functionality. Future plans include additional routines to extract information from the models such as the pair distribution function $g(r)$, and rewriting core parts of the library in C++ to improve performance. For an up-to-date list of features, we recommend visiting the [documentation website](#), where we will also provide a changelist for each new version.

Acknowledgements

The authors acknowledge financial support from Spanish MICINN (Grant Nos. PID2019-109539GB-C43, TED2021-131323B-I00 & PID2022-141712NB-C21), María de Maeztu Program for Units of Excellence in R&D (GrantNo.CEX2018-000805-M), Comunidad Autónoma de Madrid through the Nanomag COST-CM Program (Grant No. S2018/NMT-4321), Generalitat Valenciana through Programa Prometeo (2021/017), Centro de Computación Científica of the Universidad Autónoma de Madrid, and the Red Española de Supercomputación.

References

- Gresch, D., Autès, G., Yazyev, O. V., Troyer, M., Vanderbilt, D., Bernevig, B. A., & Soluyanov, A. A. (2017). Z2Pack: Numerical implementation of hybrid wannier centers for identifying topological materials. *Phys. Rev. B*, *95*, 075146. <https://doi.org/10.1103/PhysRevB.95.075146>
- Groth, C. W., Wimmer, M., Akhmerov, A. R., & Waintal, X. (2014). Kwant: A software package for quantum transport. *New Journal of Physics*, *16*(6), 063065. <https://doi.org/10.1088/1367-2630/16/6/063065>
- Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., ... Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. <https://doi.org/10.1038/s41586-020-2649-2>
- Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. <https://doi.org/10.1109/MCSE.2007.55>
- Jones, R. O. (2015). Density functional theory: Its origins, rise to prominence, and future. *Rev. Mod. Phys.*, *87*, 897–923. <https://doi.org/10.1103/RevModPhys.87.897>
- Kane, C. L., & Mele, E. J. (2005). Z_2 topological order and the quantum spin hall effect. *Phys. Rev. Lett.*, *95*, 146802. <https://doi.org/10.1103/PhysRevLett.95.146802>
- Moldovan, D., Anđelković, M., & Peeters, F. (2020). *pybinding v0.9.5: a Python package for tight-binding calculations* (Version v0.9.5). Zenodo. <https://doi.org/10.5281/zenodo.4010216>
- Radha, S. K. (2020). *Santoshkumarradha/pysktb: Tightbinding electronic structure codes*. Zenodo. <https://doi.org/10.5281/ZENODO.4311595>
- Slater, J. C., & Koster, G. F. (1954). Simplified LCAO method for the periodic potential problem. *Phys. Rev.*, *94*, 1498–1524. <https://doi.org/10.1103/PhysRev.94.1498>
- Uría-Álvarez, A. J., Molpeceres-Mingo, D., & Palacios, J. J. (2022). Deep learning for disordered topological insulators through their entanglement spectrum. *Phys. Rev. B*, *105*, 155128. <https://doi.org/10.1103/PhysRevB.105.155128>
- Vergniory, M. G., Elcoro, L., Felser, C., Regnault, N., Bernevig, B. A., & Wang, Z. (2019). A complete catalogue of high-quality topological materials. *Nature*, *566*(7745), 480–485. <https://doi.org/10.1038/s41586-019-0954-4>
- Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., ... SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. <https://doi.org/10.1038/s41592-019-0686-2>