

ElecTra User Guide

Patrizio Graziosi^{1,2} and Neophytos Neophytou²

¹ CNR – ISMN, via Gobetti 101, 40129, Bologna

² School of Engineering, University of Warwick, CV4 7AL, Coventry

1. Introduction and Terms of Use	2
2. Physical concepts	4
3. Installation and execution	18
4. Input data preparation – format and structure	21
5. Input file preparation	24
6. Scattering parameters file preparation	41
7. Computed quantities	48
a. Plotting the results with the plot GUI	52
8. Run the code	56
a. on a supercomputing facility	56
b. via a dedicated GUI	57
9. Validation	63
10. Example of using bandstructure from a DFT code	67

1. Introduction

The *ElecTra* simulator solves the Boltzmann transport equation (BTE) for charge transport using the wave-vector and energy dependent momentum relaxation time approximation. The simulator computes the charge transport coefficients. It considers charge carrier scattering with phonons, ionized dopants, and alloy scattering [1, 2].

ElecTra takes as input the electronic bandstructure and certain scattering parameters as detailed below and returns the charge transport coefficients electrical conductivity (σ), Seebeck coefficient (S), thermoelectric power factor ($PF = \sigma S^2$), and electronic part of the thermal conductivity (κ_e) as function of Fermi level (E_F) position and temperature (T). These are computed using the transport distribution function (TDF) within the Linearized Boltzmann Transport equation. The overall energy dependent relaxation time and mean-free-path (MFP) are computed as well, also for each scattering mechanism separately. All these quantities are computed as rank-2 tensors.

The purpose of this manual is to introduce the physics behind the functionality of the code, the several different input options that the user has, and provide a few operational examples to help the user become more familiar with the code and use it effectively.

NOTE

The code is written in MATLAB® and makes use of the Parallel Computing Toolbox™. A version ported in C is under test.

Terms of Use

ElecTra is an open-source code, released under the GNU General Public License, <https://www.gnu.org/licenses/gpl-3.0.en.html>, and developed under the Marie Curie project *GENESIS* - Generic semiclassical transport simulator for new generation thermoelectric materials, project ID 788465, funded by the European Commission under the MSCA actions.

We shall appreciate if any work done using the *ElecTra* code will cite the code source and reference [1] and, if either bipolar calculation is performed, reference [2] as well.

2. Physical concepts

The *ElecTra* code is based on the semiclassical description of charge transport where the charge carrier dynamics are described by the linearized BTE. The carrier interaction with the scatterers (phonons, ionized impurities, alloy defects) is quantum mechanically described by Fermi's Golden Rule. [1 - 4]. Here we present the theory that simulator is built upon [5].

The charge transport coefficients, electrical conductivity, σ , and the Seebeck coefficient, S , are rank-2 tensors in the Cartesian components ij and have the form:

$$\sigma_{ij(E_F, T)} = q_0^2 \int_E \Xi_{ij}(E) \left(-\frac{\partial f_0}{\partial E} \right) dE, \quad (1a)$$

$$S_{ij(E_F, T)} = \frac{q_0 k_B}{\sigma_{ij}} \int_E \Xi_{ij}(E) \left(-\frac{\partial f_0}{\partial E} \right) \frac{E - E_F}{k_B T} dE, \quad (1b)$$

$$\kappa_{eij} = \frac{1}{T} \int_E \Xi_{ij}(E) \left(-\frac{\partial f_0}{\partial E} \right) (E - E_F)^2 dE - \sigma S^2 T \quad (1c)$$

where $\Xi_{ij}(E)$ is the Transport Distribution Function (TDF) defined below in equation (2), E_F , T , q_0 , k_B , and f_0 , are the Fermi level, the absolute temperature, the electronic charge, the Boltzmann constant, and the equilibrium Fermi distribution, respectively. The simulator performs a trapezoidal integration over the energy; step of 2 to 3 meV is generally sufficient, but the user can use any value. The most critical aspect is the step in the k -space mesh, where a spacing of at most ~ 0.02 in units of π/a is recommended for the material's bandstructure under consideration (see validation section for more details). A numerical band interpolation is implemented, if desired, at the stage of the extraction of the bandstructure from the '.bxsf' format file. Such interpolation is barely numerical and based on the natural neighbour interpolation method; each band is interpolated separately according to the band index, and the required time is of the order of minutes, or tens of minutes, on a laptop, depending on the number of bands and of k -points. [8] In addition, a linear interpolation is performed inside each mesh element to construct the constant energy surfaces.

The TDF is central to the computation, and is computed as a surface integral over the constant energy surfaces, \mathfrak{Q}_E^n , for each band, and then summed over the bands, as [1, 2, 5]

$$\Xi_{ij(E,E_F,T)} = \frac{s}{(2\pi)^3} \sum_{\mathbf{k},n}^{\mathfrak{Q}_E^n} v_{i(\mathbf{k},n,E)} v_{j(\mathbf{k},n,E)} \tau_{i(\mathbf{k},n,E,E_F,T)} \frac{dA_{\mathbf{k}_{\mathfrak{Q}_E^n}}}{|\vec{v}_{(\mathbf{k},n,E)}|} \quad (2)$$

where $\mathbf{k}_{\mathfrak{Q}_E^n}$ represents a state on the surface \mathfrak{Q}_E^n and $dA_{\mathbf{k}_{\mathfrak{Q}_E^n}}$ is its corresponding surface area element, computed as explained below in the “Extraction of the constant energy surfaces” section. [1] $v_{i(\mathbf{k},n,E)}$ is the i -component of the band velocity of the transport state, computed as the gradient of the bandstructure, calculated using the Lehmann-Taut method. [7] $\tau_{i(\mathbf{k},n,E)}$ is its momentum relaxation time (combining the relaxation times of each scattering mechanism using Matthiessen’s rule), $\frac{dA_{(\mathbf{k},n,E)}}{|\vec{v}_{(\mathbf{k},n,E)}|}$ its density-of-states (DOS), [1] and s the spin degeneracy, for which $s = 2$ is used when the two spin sub-bands are degenerate, i.e. when the material is not ferromagnetic. To perform the surface integrals in Eq. (2), the bandstructure, generally computed on a defined mesh in the reciprocal space as an $E(\mathbf{k})$, is transformed in a $\mathbf{k}(E)$, describing all the k -points at a given energy in an underlying uniform energy grid. This describes the constant energy surfaces, (for example the well-known ellipsoids in common semiconductors), and every summation becomes a surface integral with \mathfrak{Q}_E^n being the integration domain. The points on these surface are the transport states, uniquely defined by the (\mathbf{k},n,E) triad.

In the case of 2D band structures, constant energy contours are composed, instead of surfaces, and the TDF and charge transport coefficients are composed by multiplying the above expression by the inverse of the 2D layer thickness, as explained in reference [8],

Extraction of the constant energy surfaces

The code offers two possibilities to construct the constant energy surfaces. The first is based on the Delaunay Triangulation (DT) performed locally on the k -space mesh elements which

are crossed by the surface of the constant energy of interest. The second is based on a scan of the nearest neighbour points on the k -mesh. Although the latter is an approximation because it detects only the points along the edges of the k -mesh 3D elements, it is around 15 to 30 times faster, but without noticeable penalty in the results compared to the DT method. For isotropic bandstructures the discrepancies are at the level of the numerical noise, while for anisotropic bands, in terms of the density of states (DOS), the differences increase from zero at the band edge to $\sim 1\%$ around 0.2 eV from the band edge and to $\sim 4\%$ around 0.5 eV from the band edge.

1. Local Delaunay Triangulation (DT)

In the case of DT, the reciprocal unit cell is scanned to detect the mesh elements which are crossed by a certain constant energy surface, for each energy (and each band). The code checks the energy of all the vertex pairs (12 pairs in 3D) for the condition if at least one of these pairs consists of a vertex with energy above the value under consideration, and one below. All the relevant elements are then triangulated. The $E(\mathbf{k})$ is approximated to be linear between these points, and thus a \mathbf{k}_i -point of energy E_i between the vertexes $v1$ and $v2$ is selected on each edge of the element of the triangulation as [9]:

$$\mathbf{k}_i = \mathbf{k}_{v1} + (\mathbf{k}_{v2} - \mathbf{k}_{v1}) \frac{E_i - E_{v1}}{E_{v2} - E_{v1}} \quad (3).$$

Each tetrahedron will have three or four points (depending on whether the vertexes above/below the energy under consideration are one or two as shown in **Figure 1** below). These define a surface computed with Heron's formula: $A = \sqrt{s_p(s_p - l_1)(s_p - l_2)(s_p - l_3)}$, where A is the triangle area, s_p is the semi-perimeter and l_i are the triangle sides. When the tetrahedron is crossed in four points, as in **Figure 1c**, the surface element is twice the average of the areas of the four defined triangles. Such surface is the dA_k used in Eq. (2) and is employed in the integration when computing the density of states. When we evaluate the anisotropic scattering

rates, the code needs the coordinates of the points associated to a given surface element, because the scattering rate depends on the wave-vector difference between the final and initial states. The code finds the barycentre of the surface element, and this is the k -point associated to the given surface element as shown by the yellow cross in **Figure 1**.

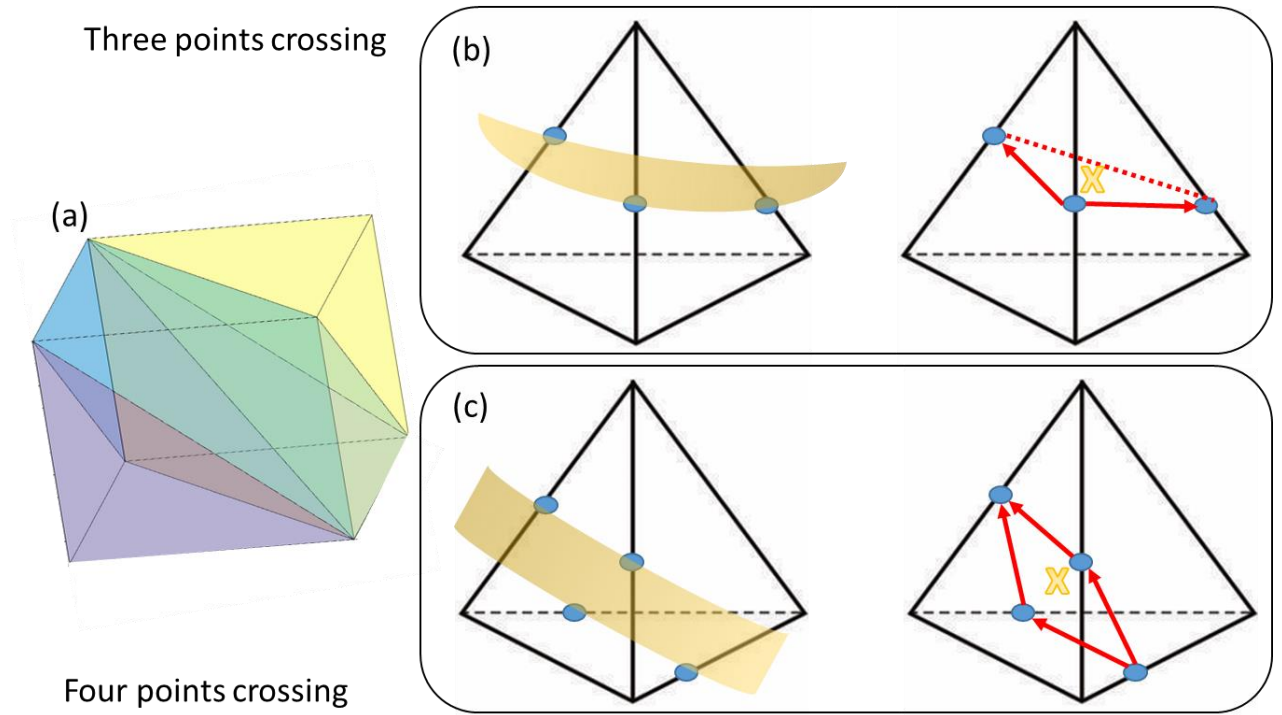


Figure 1: (a) Delaunay Triangulation performed using MATLAB® for a 3D mesh element. (b) and (c) show the cases where the constant energy surface of interest crosses a tetrahedron in three points (b) or four points (c). A part of the energy surface is depicted in yellow and the crossing points in blue. The surface element will be the area of the triangle in (b) or the parallelogram in (c) defined by the red arrows. Its associated element coordinates are represented by the coordinates of the barycentre shown by the yellow cross.

2. Fast nearest neighbour scan method (NN Scan)

In this method, the code scans all the \mathbf{k} -points and checks for their nearest neighbours along the edges of the \mathbf{k} -mesh. This is depicted in **Figure 2(a)**, where the \mathbf{k} -point of interest is shown in orange and its nearest neighbours in green. If the \mathbf{k} -point of interest and a neighbour have energies one above and one below the energy value of interest, the code selects a new \mathbf{k} -point along the edge of the mesh assuming a linear $E(\mathbf{k})$ between the two points, as above in Eq. (3). This new point becomes the \mathbf{k} -point at the energy of interest for the scattering rate evaluation – the purple cross in **Figure 2(b)**. In this way the code does not resolve a constant energy surface element, and the constant energy surface is never constructed, but acquires a collection of points on the energy surface of interest. Then, the code assigns an effective dA_k surface element area value to each point to allow us to extract the density of states associated with that \mathbf{k} -point. For this, after grouping the dispersion points on the constant energy surfaces, the surface in the neighbourhood of each \mathbf{k} -point is explored to detect its neighbours on the surface. This is done in a radius of $1.25 \times dk$ where dk is the average distance between adjacent points in the initial mesh, the one used in the DFT bands to calculate the bands. Then, the code calculates the average distance between the given point and its detected neighbours, $\langle \Delta k \rangle$. The surface element associated to the \mathbf{k} -point is approximated by a circle of radius half the average distance to the neighbouring points, i.e., $dA_k = \pi \left(\frac{\langle \Delta k \rangle}{2} \right)^2$. These are represented in purple in **Figure 2(b)**, where the neighbouring points on the surfaces are represented by black crosses. Excellent agreement is achieved between this method and the DT method, as well as excellent match to analytical parabolic and non-parabolic results (see validation section and reference [1]).

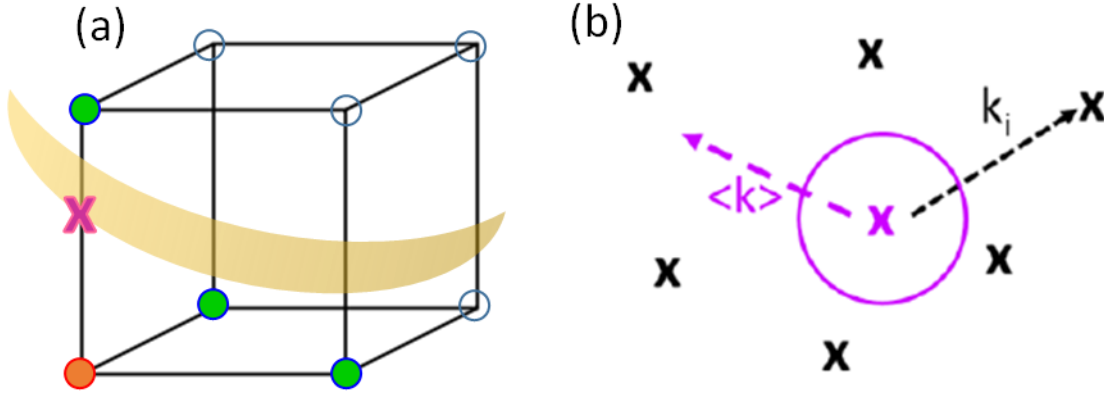


Figure 2: (a) A k -point in a 3D k -mesh element (orange) with some of its nearest neighbours (in green). When a constant energy surface (yellow) crosses the edge between the point under consideration and its neighbours, the crossing point (purple) is selected. (b) The conceptual schematic for the formation of the dA_k , where the specific k -point under consideration is in purple, its neighbours in back, k_i is the distance between the point in purple and the i^{th} -neighbour and $\langle k \rangle$ is the average distance. The surface element is represented by the purple circle. In both the schemes, the velocity, and hence the DOS, is computed using the approach by Lehmann and Taut. [7]

Scattering rate calculation

For each transport state and each scattering mechanism m_s , the corresponding relaxation time $\tau_{i(k,n,E)}^{(m_s)}$ is defined as:

$$\frac{1}{\tau_{i(k,n,E)}^{(m_s)}} = \frac{1}{(2\pi)^3} \sum_{\mathbf{k}'} |S_{\mathbf{k},\mathbf{k}'}^{(m_s)}| \left(1 - \frac{v_{i(\mathbf{k}')}}{v_{i(\mathbf{k},n,E)}} \right) \quad (4)$$

where the sum runs over all the allowed final states \mathbf{k}' . This formalism assumes that the scattering does not change the spin of the carrier, [3, 4] however, if it does, then the rate needs to be doubled (not included in this version of the code). $|S_{\mathbf{k},\mathbf{k}'}|$ is the transition rate between the initial \mathbf{k} and final \mathbf{k}' states, computed as detailed by Eq. 5 below. The $\left(1 - \frac{v_{ij(\mathbf{k}')}}{v_{ij(\mathbf{k},n,E)}} \right)$ term is an

approximation for the momentum relaxation time, [9, 12] which is the type of relaxation time that matters when computing the transport coefficients. [4] The simulator computes the scattering rates using Fermi's Golden Rule. [3,4] Equations 5 below report the actual equations used in *ElecTra* for the 3D case (5a-5f) and 2D case (5g-n):

$$|S_{\mathbf{k},\mathbf{k}'}^{(ADP)}| = 2 \frac{\pi}{\hbar} D_{ADP}^2 \frac{k_B T}{\rho v_s^2} g_{\mathbf{k}'} \quad (5a)$$

$$|S_{\mathbf{k},\mathbf{k}'}^{(ODP)}| = \frac{\pi D_{ODP}^2}{\rho \omega} \left(N_\omega + \frac{1}{2} \mp \frac{1}{2} \right) g_{\mathbf{k}'} \quad (5b)$$

$$|S_{\mathbf{k},\mathbf{k}'}^{(IVS)}| = \frac{\pi D_{IVS}^2}{\rho \omega} \left(N_\omega + \frac{1}{2} \mp \frac{1}{2} \right) g_{\mathbf{k}'} \quad (5c)$$

$$|S_{\mathbf{k},\mathbf{k}'}^{(POP)}| = \frac{\pi e^2 \omega}{|\mathbf{k}-\mathbf{k}'|^2 \varepsilon_0} \left(\frac{1}{k_\infty} - \frac{1}{k_s} \right) \left(N_{\omega, BE} + \frac{1}{2} \mp \frac{1}{2} \right) g_{\mathbf{k}'} \quad (5d)$$

$$|S_{\mathbf{k},\mathbf{k}'}^{(IIS)}| = \frac{2\pi}{\hbar} \frac{Z^2 e^4}{k_s^2 \varepsilon_0^2} \frac{N_{imp}}{\left(|\mathbf{k}-\mathbf{k}'|^2 + \frac{1}{L_D^2} \right)^2} g_{\mathbf{k}'} \quad (5e)$$

$$|S_{\mathbf{k},\mathbf{k}'}^{(Alloy)}| = \frac{2\pi}{\hbar} \Omega_c x (1-x) G_{\mathbf{k}-\mathbf{k}'} \Delta E_G^2 g_{\mathbf{k}'} \quad (5f)$$

$$|S_{\mathbf{k},\mathbf{k}'}^{(ADP)}| = \frac{1}{t} 2 \frac{\pi}{\hbar} D_{ADP}^2 \frac{k_B T}{\rho v_s^2} g_{\mathbf{k}'} \quad (5g)$$

$$|S_{\mathbf{k},\mathbf{k}'}^{(ODP)}| = \frac{1}{t} \frac{\pi D_{ODP}^2}{\rho \omega} \left(N_\omega + \frac{1}{2} \mp \frac{1}{2} \right) g_{\mathbf{k}'} \quad (5h)$$

$$|S_{\mathbf{k},\mathbf{k}'}^{(IVS)}| = \frac{1}{t} \frac{\pi D_{IVS}^2}{\rho \omega} \left(N_\omega + \frac{1}{2} \mp \frac{1}{2} \right) g_{\mathbf{k}'} \quad (5i)$$

$$|S_{\mathbf{k},\mathbf{k}'}^{(POP)}| = \frac{\pi q_0^2 \omega}{2 |\mathbf{k}-\mathbf{k}'| \varepsilon_0} \left(\frac{1}{k_\infty} - \frac{1}{k_s} \right) \left(N_{\omega, BE} + \frac{1}{2} \mp \frac{1}{2} \right) g_{\mathbf{k}'} \quad (5l)$$

$$|S_{\mathbf{k},\mathbf{k}'}^{(IIS)}| = \frac{\pi}{2\hbar} \frac{Z^2 q_0^4}{k_s^2 \varepsilon_0^2} \frac{N_{imp}}{|\mathbf{k}-\mathbf{k}'|^2 + \frac{1}{L_D^2}} g_{\mathbf{k}'} \quad (5m)$$

$$|S_{\mathbf{k},\mathbf{k}'}^{(Alloy)}| = \frac{1}{t} \frac{2\pi}{\hbar} \Omega_c x (1-x) G_{\mathbf{k}-\mathbf{k}'} \Delta E_G^2 g_{\mathbf{k}'} \quad (5n)$$

Above, ADP stands for 'Acoustic Deformation Potential' and represents the scattering between charge carriers and acoustic phonons, which combines emission and absorption [9] in

the approximation of neglecting the energy of the acoustic phonon. ODP stands for ‘Optical Deformation Potential’ and describes the charge carrier inelastic scattering with non-polar optical phonons. IVS stands for Inter-Valley Scattering and is specific for the inelastic intervalley scattering. POP stands for ‘Polar Optical Phonon’ and describes the inelastic scattering of charge carriers with polar phonons. IIS stands for ‘Ionized Impurity Scattering’ and describes the elastic scattering rate due to the ionized dopants. “Alloy” represents the alloy scattering due to intrinsic disorder in alloys or solid solutions. \mathbf{k} and \mathbf{k}' are the wave vectors of the initial and final states. The variables that appear in Eqs. 5 are as follows: D_{ADP} , D_{ODP} , D_{IVS} are the deformation potentials for the ADP, ODP, and IVS mechanisms. ρ is the mass density, v_s the sound velocity, ω the dominant frequency of optical phonons, considered as constant over the whole reciprocal unit cell, which has been validated to be a satisfactory approximation, [13] and N_ω is the phonon Bose-Einstein statistical distribution. e is the electronic charge, ϵ_0 the vacuum dielectric constant, k_s and k_∞ the static and high frequency relative permittivities, Z the electric charge of the ionized impurity, and N_{imp} is the density of the ionized impurities.

$g_{\mathbf{k}'} = \frac{dA_{(\mathbf{k}',n,E)}}{|\vec{v}_{(\mathbf{k}',n,E)}|}$ or $g_{\mathbf{k}'} = \frac{d\ell_{(\mathbf{k}',n,E)}}{|\vec{v}_{(\mathbf{k},n,E)}|}$ are the single-spin DOS of the final state for the 3D (eq.s 5a-5f) and 2D (eq.s 5g-5n) cases, respectively. The final state’s energy is the same as that of the initial state under elastic scattering, while for inelastic processes it is increased/decreased by $\hbar\omega$ for absorption/emission processes, respectively, represented by “−” and “+” in Eqs. 5b-5d.

$L_D = \sqrt{\frac{k_s \epsilon_0}{e} \left(\frac{\partial n}{\partial E_F} \right)^{-1}}$ and $L_D = 2 \frac{k_s \epsilon_0}{e} \left(\frac{\partial n}{\partial E_F} \right)^{-1}$ are the screening length with E_F being the Fermi level and n the carrier density, for 3D and 2D bandstructures, respectively. [1,8] Ω_c is the volume of the primitive cell, entered in units of m^3 also for the 2D case, x the fraction of one of the alloy elements, and ΔE_G the difference between the energy gap of the two constituent materials that form the alloy. [3, 9] The G function is the form factor of a hard sphere or disk according to the 3D or 2D case, respectively. [10,11] Rigorous studies show that the ΔE_G can

be substituted by an effective scattering potential to better fit experimental values. [9] However, the code uses the ΔE_G term for general use for unknown, not yet synthesized or characterized, materials.

The *ElecTra* simulator is targeted towards the predictive modelling of thermoelectric properties of materials and it is validated to room and higher temperatures. It considers that all dopants are ionized and does not capture the carrier freeze out region. Note that the equipartition theorem at the basis of Eq. 5a does not hold at low temperatures. Screening in the electron-phonon scattering processes is implemented for the POP mechanism only, where the

screening factor $\frac{|k-k'|^2}{\left(|k-k'|^2 + \frac{1}{L_D^2}\right)^2}$ for 3D or $\frac{1}{1 + \frac{1}{|k-k'|^2} \frac{1}{L_D^2}}$ for 2D, can be added if the user desires. The

screening in the carrier-phonon scattering event has a variable effect on the transport properties according to the bandstructures features, however it increases the computational complexity (as the scattering rates become Fermi level dependent), so the user can choose whether considering this part or not. The development of the POP, IIS, and related screening, in 2D, is based on the different Fourier transforms of the Coulomb-like potential in 3D and 2D, as explained in reference [8] and references therein.

Calculation of the momentum exchange vector: The POP and IIS mechanisms are anisotropic as they depend on the distance in the k -space between the initial and final states $|k - k'|^2$. To compute this momentum exchange vector, the simulator takes into consideration every final state's k -point position on all neighbouring reciprocal unit cells (as shown in **Figure 3**), and then the minimum distance from the initial point is used. For this, as shown in the 2D schematic of **Figure 3**, the final point (red bullet), is shifted by the reciprocal lattice vectors in all possible directions, creating the green bullets. Then the code considers the closest to the initial point (blue bullet). This is necessary, because for example two k -points that are located at opposite edges of the Brillouin zone (BZ) are actually very near if the equivalent points in

the nearest neighbour BZs are considered. **Figure 3** shows a 2D schematic of this, where the cell with bold edges represents the cell used in the simulations and the other cells are the equivalent ones. The blue point is the initial point, the red point is the nominal final point, the green points are all the equivalent final points in other reciprocal unit cells. The physical scattering vector is the green one, and not the red one.

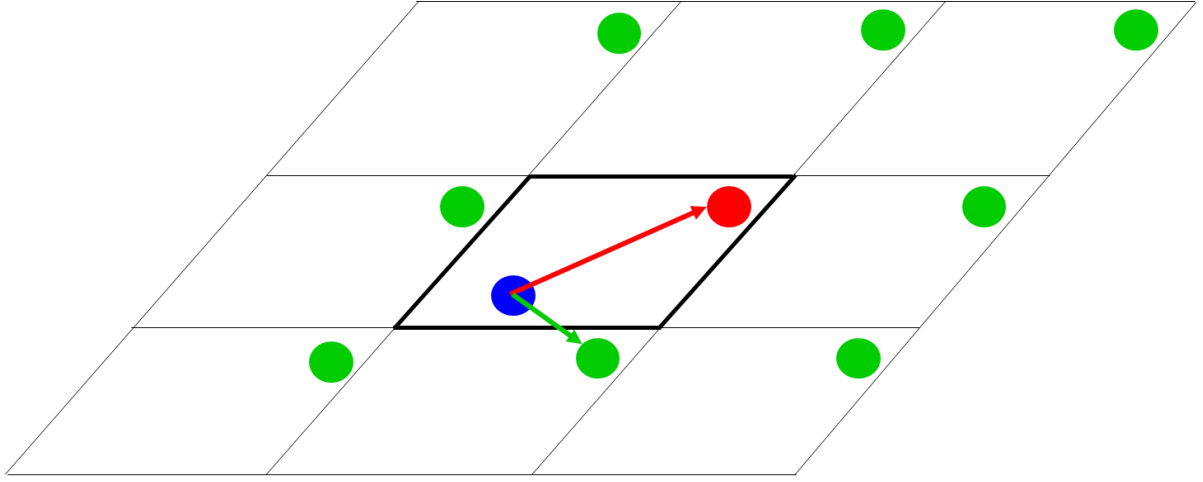


Figure 3: 2D schematic of the reciprocal unit cell used in the calculation, with bold edges, and its equivalent cells around it. The initial \mathbf{k} -point is in blue and the final \mathbf{k} -point in red. The equivalent final \mathbf{k} -points are shown in green, obtained by translating the red point by one reciprocal lattice vector in all possible directions. In the anisotropic POP and IIS scattering mechanisms, all the equivalent \mathbf{k} -points are explored, and the final \mathbf{k} -point considered is the one closest to the initial.

Bipolar transport: When bipolar transport is considered, the TDF is constructed as in Eq. (2) for the whole energy range of interest, accounting for both the conduction and valence bands. Then the transport coefficients are computed by the integrals in Eq. (1) over the whole energy range. With regards to screening in the case of bipolar transport, since the screening arising from majority and minority carriers is additive, [3, 14] the simulator computes the

screening lengths for both the majority and minority carriers ($L_{D,maj}$ and $L_{D,min}$) which are then summed up as: $L_D^2 = L_{D,maj}^2 + L_{D,min}^2$. Note that consideration of unipolar transport for each band and then a merge of the results from the two bands by utilizing the usual relations is not accurate, and we recommend the combined full energy and band type calculation. [6,15]

The bipolar transport calculation can be done in two ways. (i) by defining one type of majority carriers; this is like a unipolar calculation but considering that there are also the minority carriers. (ii) by considering both type of doping, i.e. electrons as majority carriers and then holes as majority carriers, in the same calculation.

Constant relaxation time approximation (CRTA): The simulator can consider the constant relaxation time approximation (CRTA) and the constant mean-free-path approximation (CMFP). The user can choose the CRTA as an input option and set the value of the relaxation time, in fs units. When the CRTA option is selected, the CMFP calculations are also automatically performed with a constant mean free path value of 5 nm, commonly employed in the CMFP literature. [16]

Additional computed quantities

The code also computes an overall energy dependent mean-free-path as follows, where i is a cartesian coordinate,

$$\lambda_{i(E,E_F,T)} = \frac{\sum_{k,n}^{\sigma_E^n} |v_{i(k,n,E)} \tau_{i(k,n,E,E_F,T)}| DOS(k,n,E)}{\sum_{k,n}^{\sigma_E^n} DOS(k,n,E)} \quad (6)$$

and an overall energy-dependent relaxation time as

$$\tau_{i(E,E_F,T)} = \frac{\sum_{k,n}^{\sigma_E^n} \tau_{i(k,n,E,E_F,T)} DOS(k,n,E)}{\sum_{k,n}^{\sigma_E^n} DOS(k,n,E)} \quad (7).$$

The TDF, eq. (2), overall mean-free-path, eq. (6), and relaxation time, eq. (7), are computed in addition for each individual band and each separate phonon/alloy scattering mechanism, at all the entered temperatures and, for the case of POP when the charge screening is selected, also each E_F .

Implemented scattering types:

As general indications, the scattering mechanisms operate as follows with **Figure 4** indicating intra- and inter-band transitions.

- ADP, Acoustic Deformation Potential, elastic scattering with acoustic phonons. It is treated as *intra*-band (process (1) in **Figure 4**).
- ADP_IVS, ADP as above, but both *intra*- and *inter*- band scattering transitions are considered (processes (1) and (3) in **Figure 4**). ADP and ADP_IVS are mutually exclusive, thus, if both are selected, the code will adopt ADP.
- the `scattering vector restriction` option restricts the ADP and ADP_IVS scattering to only the points which are inside the portion of the BZ enclosed by the cutoff threshold specified by the user as '`scattering vector cutoff`' in the input file. For instance, if the user enters '`0.2`' for this value, only the points that are closer than a fraction of 0.2 of the reciprocal unit cell size will be considered for ADP – the equivalent points in neighbouring cells are considered, as in the case for IIS and POP. This option considers the fact that acoustic phonons have generally low momentum and cannot easily allow electronic transitions that require large momentum in k -space.
- ODP, non-polar Optical Deformation Potential, describes the inelastic electron scattering with non-polar optical phonons. It is considered as *both intra*- and *inter*-band, when the `multivalley` flag is '`no`' (processes (2) and (4) in **Figure 4**), and

- intra*- band only, when the `multivalley` flag is set to 'yes' (process (2) in **Figure 4**). In the “Basic” GUI operation the `multivalley` flag is automatically set to ‘no’.
- POP, Polar Optical Phonon: It describes the long range, short wave vector, interaction with longitudinal polar optical phonons using the Fröhlich approach. It is considered as *intra*- and *inter*- band (processes (2) and (4) in **Figure 4**). See reference [17] for more details.
 - IIS, Ionized Impurity Scattering: Describes the scattering with the ionized impurities mediated by the Coulomb potential. Although usually considered as *intra*- band and in the ‘Basic’ mode it is treated as *intra*-band, process (1) in **Figure 4**, in the ‘Advanced’ mode the user can choose to have it as *inter*-band as well, process (3) in **Figure 4**.
 - Alloy: This term considers the carrier scattering related to the intrinsic disorder that appears in alloys or solid solutions, [3] and is set to allow both *intra*- and *inter*- band transitions, processes (1) and (3) in **Figure 4**.
 - IVS: This stands for InterValley Scattering, which describes the *inter*-band inelastic scattering, process (4) in **Figure 4**. It allows several different processes with different deformation potentials when the simulator is operated in the ‘multivalley’ mode. In the other case, the IVS operates as the initial and final valleys were the same, with a certain multiplicity; for example in the case of silicon CB, where the six valleys belong to the same band, a multiplicity of 1/6 or 4/6 is used for *g*- and *f*- processes.
 - `overlap_integrals_analytical`: Computes the overlap integrals between wavefunctions using the analytical equations of reference [18]. When selected, the code will compute the overlap integral analytically and will add this term to the scattering rate. [3, 18] Otherwise, the overlap integrals are considered to be one.

In **Figure 4** we pictorially show the different scattering transitions as:
process (1): ADP, IIS, Alloy

process (2): ODP

process (3): ADP, IIS, Alloy – the first two only if selected by the user

process (4): ODP (only if 'multivalley' is not selected), IVS

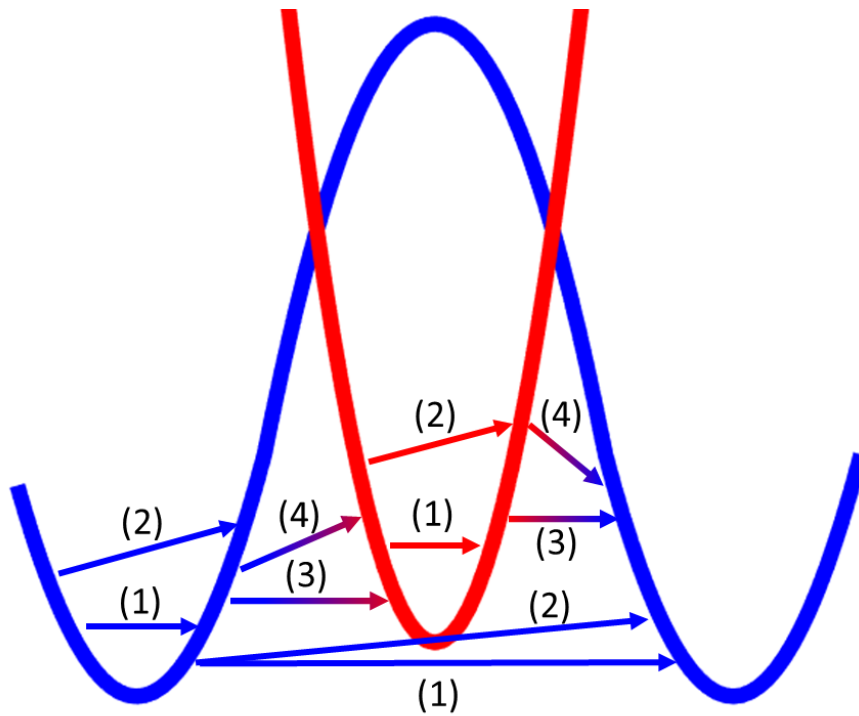


Figure 4: Schematic of two bands (one with many valleys, in blue, and one single valley, in red) with the four types of allowed transitions: (1) elastic intra-band; (2) inelastic intra-band, (3) elastic inter-band, (4) inelastic inter-band.

3. Installation and execution

The *ElecTra* simulator is written in MATLAB® and version R2019a or higher is recommended. An amount of memory around few Gb per processor is generally required during the code execution. *ElecTra* code can be executed in parallel on multiple processors on a one or more nodes and scales almost linearly up to 45 processors at least. [8] The code diagram is shown in the schematic of **Figure 5a** and examples of how to launch the code on HPC facilities are given in Section 8. To execute the code on an HPC infrastructure, the users prepare on their own machine the input data and code instructions, then moves them in an HPC cluster where the code can be executed using a dedicated script (an example of a bash script is given in Section 8). After the simulation is finished, the saved data are imported in the user's machine for data analysis and post-processing. Different clusters likely have different specifics to launch MATLAB® programs. A plotting App is provided to plot the data even if MATLAB® is not installed. So, the user prepare the input and scattering files as text files, with '.m' extension, and passes them to the HPC, where MATLAB® is required, and can plot the results, even without having MATLAB® installed in own's PC.

The user needs to compose the 'input' file and launch the 'run' file. These are located in the working directory, where the simulation results will be saved. The working directory has two sub-directories. One is named 'code', and contains all the simulator files, except the `run` and `input` files. The other directory is named 'Data' and contains the bandstructure and scattering parameters. A screen shot of the working directory is shown in **Figure 5b**. The Input script contains the input information, alternatively, an Input file which contains the same information can be composed via the GUI. The `run` file launches the simulation. When the directory is on a supercomputing cluster, it will also contain the script that launches the `run` file in the HPC node. Depending on the system where *ElecTra* is run, the user will simply launch the 'ELECTRA_run_Linux' or the

'ELECTRA_run_Windows' file. For Mac, the Linux option must be used. It is possible to simulate more materials, one after the other, launching the code only once, by using the 'ELECTRA_multiple_run_Linux' / 'ELECTRA_multiple_run_Windows' files.

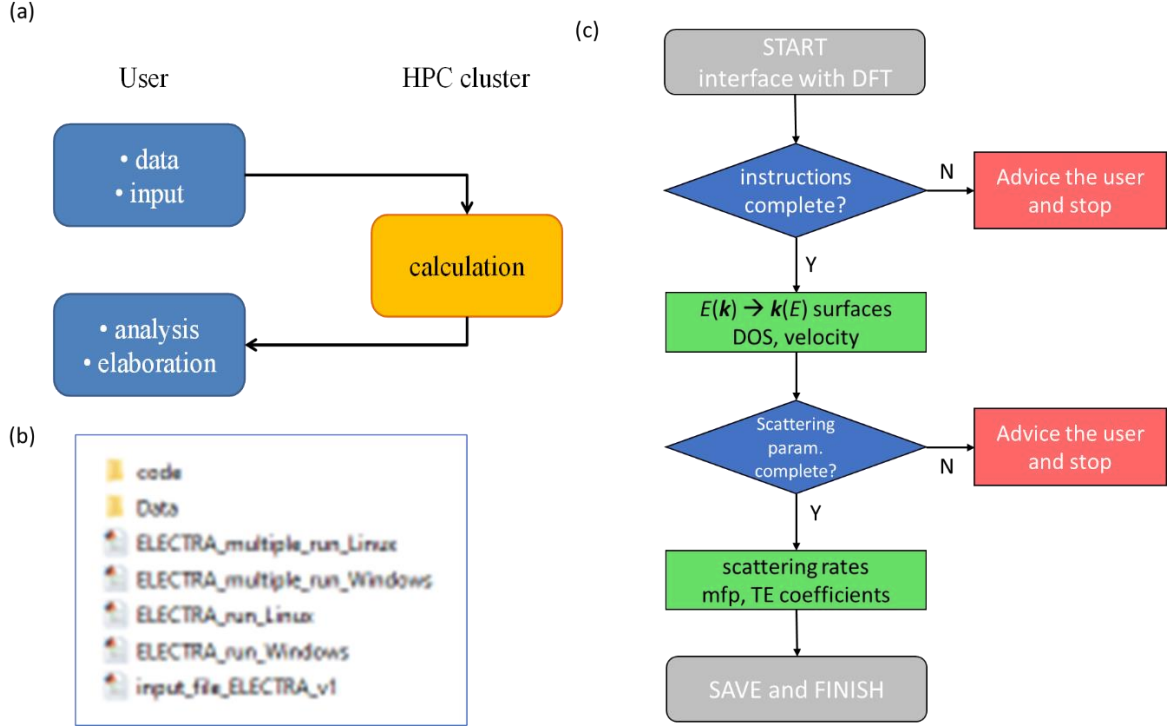


Figure 5: (a) Schematic of the code execution on an HPC cluster; (b) Screen shot of the code working folder; (c) Flow chart of the code.

The user needs to enter three files:

- i. The transport calculation specifics, contained in the `Inputs.mat` file composed with the input GUI, or contained in the `input_file_ELECTRA_v1.m` file which is written by the user as described in Section 5. Important: when the `input_file_ELECTRA_v1.m` is present, the code prioritizes it so, when the user uses the GUI, this file must be removed or renamed.

- ii. The electronic structure, usually computed by DFT, contained in the `Ek_'material_name'.mat` file. A routine for the extraction of this from the `bsxf` file are available from one of the authors, PG, and can be found on <https://github.com/PatrizioGraziosi/Extraction-from-bsxf-files>. This enables compatibility with several DFT codes, as exemplified in section 10.
- iii. The scattering parameters, contained in the `scattering_parameters_'material_name'.mat` file composed with the scattering GUI, or in the `scattering_parameter_'material_name'.m` file described in Section 6.

The code flow chart is conceptually shown in **Figure 5c**. After the start of the simulation, the code checks that all instructions are consistent, and then extracts the constant energy surfaces – $k(E)$. Then, it checks if all the necessary scattering parameters are present. The scattering rates are then calculated and afterwards the TE coefficients are computed.

4. Input data preparation – format and structure

We detail here the input data that are required by the simulator. These are contained in a ‘.mat’ file named `Ek_‘material_name’`, e.g. `Ek_TiNiSn`, which contains the bandstructure file and the coordinates of the \mathbf{k} -points that map the reciprocal unit cell. The code requires the information from *the whole reciprocal unit cell*, or Brillouin Zone, and not the Irreducible wedge of the Brillouin Zone (IBZ). Future versions will also implement the exploitation of the symmetry to reduce the computational cost. Note that the code considers all the inputs as in the International System (SI) units except the energy, in eV.

According to the dimensionality of the bandstructure, the simulator requires as input a bandstructure named `Ek`, which is a matrix of dimensionality $D+1$, for instance in the case of a 3D material, `Ek` is 4D matrix. The variable `Ek` contains the energy values for each \mathbf{k} -point and for each band in the format $n_x \times n_y \times n_z \times n_b$ where $n_{x/y/z}$ are the number of point of the \mathbf{k} -grid along the three dimensions, and n_b is the number of bands / eigenvalues for each \mathbf{k} -point. Each \mathbf{k} -point is identified by numerical indexes (i_x , i_y , i_z), where i_x ranges from 1 to n_x , i_y ranges from 1 to n_y and i_z ranges from 1 to n_z . There is a one-to-one correspondence between the numerical indexes and the coordinates in the reciprocal space, i.e. the \mathbf{k} -points which are nearest neighbour in the matrix are also nearest neighbour in the \mathbf{k} -space. **Figure 6** sketches this concept. It depicts the points of a $3 \times 3 \times 3$ Monkhorst-Pack grid for a Zincblende material, which lie in the $i_z = 1$ plane ($k_z = 0 \text{ nm}^{-1}$). The numerical indexes of the represented points are indicated, to show that the numerical indexes reflect the physical positions in the reciprocal space, e.g. the point (i_x , i_y , i_z) and the point (i_x+1 , i_y , i_z), nearest neighbour in the matrix, are also nearest neighbour in the \mathbf{k} -coordinates.

The cartesian coordinates of the points must be contained in three $n_x \times n_y \times n_z$ matrixes `kx_matrix`, `ky_matrix` and `kz_matrix`, for the k_x , k_y , k_z coordinates, respectively.

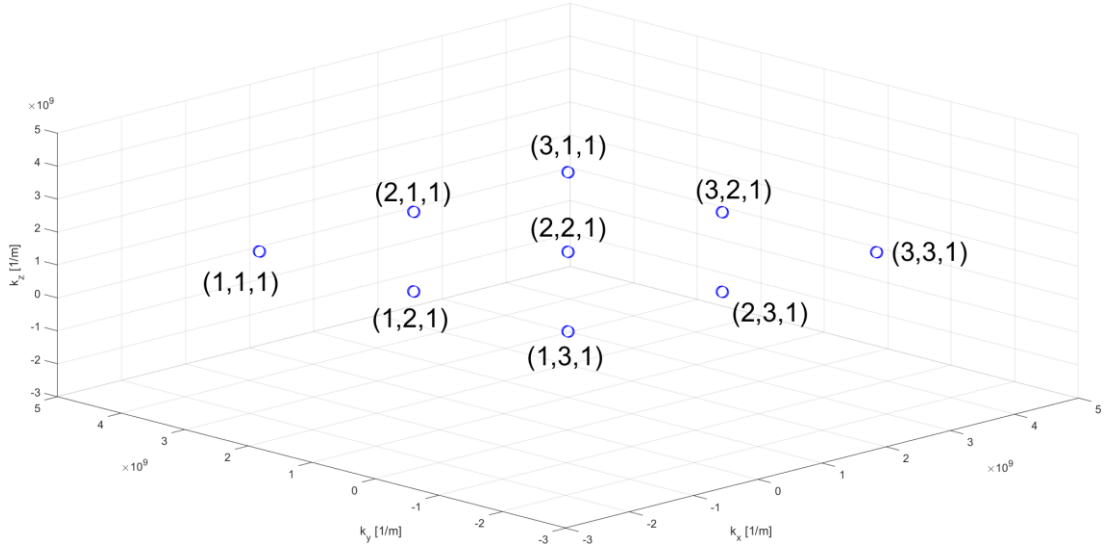


Figure 6: Representation in Cartesian axes of a $3 \times 3 \times 3$ Monkhorst-Pack grid that samples the reciprocal unit cell of a zincblende lattice, projection on the $z = 0$ plane.

ElecTra allows the user to use the electronic structure computed with *any DFT code capable of saving the data in 'bxsf' format*. *ElecTra* accepts either '.mat' and '.bxsf' files as input, as detailed in section 5. Anyway, specific interfaces can be developed upon reasonable request, by contacting the authors.

If the user wants to convert a '.bxsf' file in a '.mat' file before launching *ElecTra* two dedicated functions are provided in the directory 'Data', `bxsf_to_ELECTRA` for 3D bands and `bxsf_to_ELECTRA_2D` for 2D band structures. The use requires five input information, as follows: `bxsf_to_ELECTRA('bxsf-file-name', 'material-name', lattice-param-in-nanometres, 'SOC-not-magnetic-flag', interpolation-factor)`. The first is the full name of the `bxsf` file, including the extension, the second is the material-name that will be used to save the `Ek` file, the third is the numeric value of the lattice parameter, the one labelled 'alat' in quantum espresso, for instance, in nanometres. Then a text flag must be entered in the case the SOC is used in a non-magnetic system, because in this case every band is doubled, but is identical; in this case the identical bands are removed; type 'y' in this case, everything else in the other

cases. The last value is the interpolation factor, that is the number of times that the number of k -points are increased along each direction; if this number is lower or equal to one, the interpolation is not done. As example:

```
bxsf_to_ELECTRA( 'TiNiSn.bxsf', 'TiNiSn ', 0.417, 'n', 2.5)
```

where the file 'TiNiSn.bxsf' is converted in a 'Ek_TiNiSn.mat' file, the lattice parameter used in the DFT calculation is 0.417 nm, the SOC is not used, an interpolation of $2.5\times$ is requested, for example a mesh of $51\times 51\times 51$ becomes $129\times 129\times 129$ because the function rounds at the next odd integer.

However, in section 5 it is shown how to skip this step by using directly the bsxf file as input bandstructure.

5. input file preparation

The `'input'` file contains a set of options that are flagged as 'yes' or 'no', plus words or numerical instructions: the material name, the range of Fermi levels and Temperatures, and the width of the energy step for the integration. The `input` file can be prepared both as a script or as a `.mat` file using the Graphical User Interface (GUI). The GUI allows simplicity and flexibility, and has two options, the "Basic" and the "Advanced". Both are described below. The GUI prepares automatically an `'input.mat'` file which will be moved to the HPC facility. The `'input'` `.mat` file is not human readable, but it is straightforwardly composed based on the user's selection in the GUI.

Alternatively, the user can write an `'input'` text file. It is preferable that the user modifies the default example provided in the Appendix for the case of the TiNiSn half-Heusler compound. It is essential that the file name is `'input_file_ELECTRA_v1.m'`. The code searches for the input instructions in the working directory. In the case where both an `'input'` script file and an `'input.mat'` file exist, the code will prioritize the `.m` script file. Thus, when the user composes the `'input.mat'` file using the GUI, the script must not be present in the working directory. Importantly, the `'m'` file is just a text file with the extension `'m'`, and can be edited with *any text editor*.

Thus, the user can choose if using the GUI or not. If the user uses the GUI, then the instructions below are followed. If the user prefers to use the script, the available options are the same as the advanced GUI, the numerical entries must be entered as vectors, the options will be defined as `'yes'` or `'no'`.

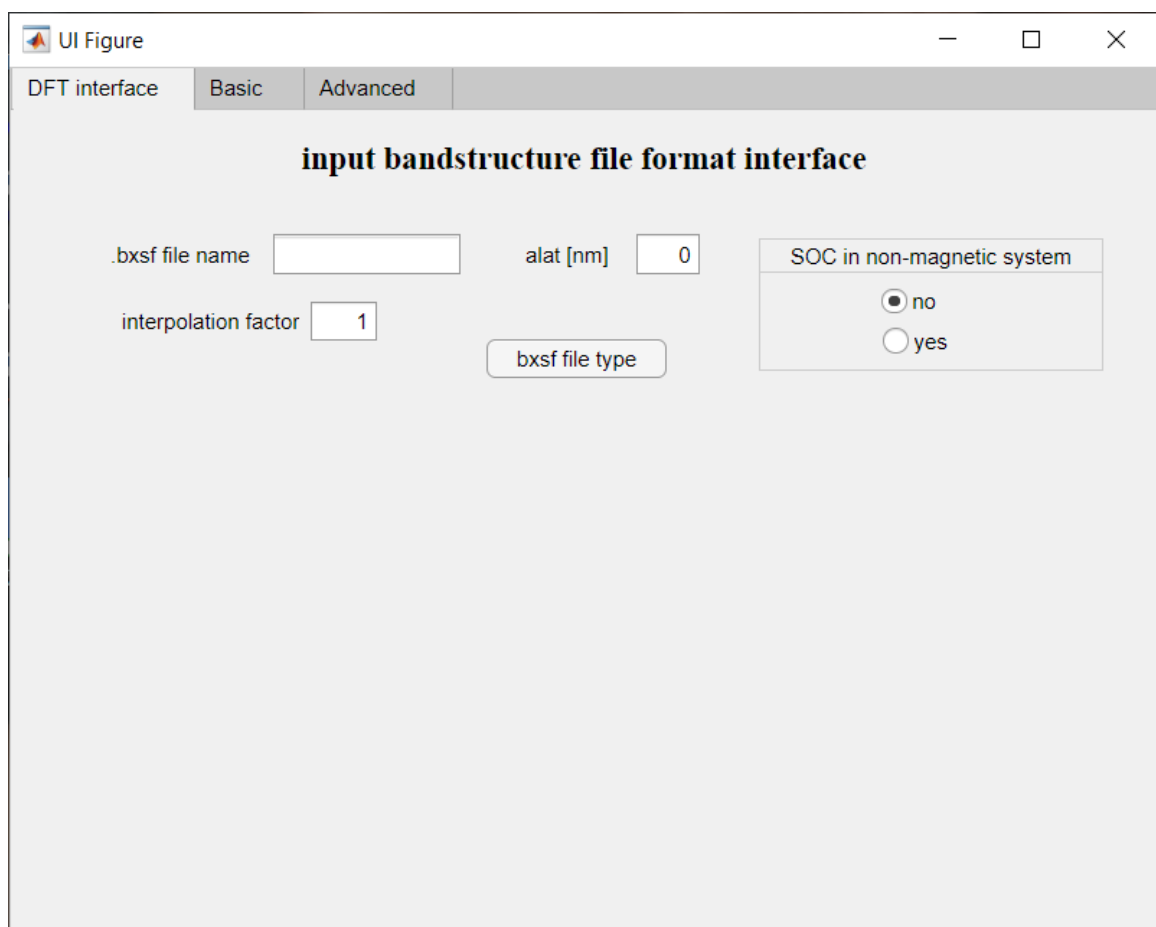
NOTE: the user can also run the code using directly the run GUI application, named `'Run_ELECTRA_App_W'` or `'Run_ELECTRA_App_L'` for Windows or Linux systems as

shown in **Section 8b**. This can be used to run *ElecTra* on own's pc or in an HPC infrastructure allowing the use of GUIs.

The GUI

- Interfacing with DFT codes

ElecTra allows to use as bandstructure the output of any DFT code, if saved in the bxsf format. This interfacing possibility is performed in the first tab of the input GUI, **Figure 7**.



The image shows a software window titled 'UI Figure' with three tabs: 'DFT interface', 'Basic', and 'Advanced'. The 'DFT interface' tab is selected. The main area is titled 'input bandstructure file format interface'. It contains several input fields and a button: a text box for '.bxsf file name', a numeric box for 'alat [nm]' with the value '0', a numeric box for 'interpolation factor' with the value '1', a button labeled 'bxsf file type', and a group box titled 'SOC in non-magnetic system' containing two radio buttons, 'no' (which is selected) and 'yes'.

Figure 7: GUI tab to enter the details to interface *ElecTra* with the output of DFT codes saved in ‘bxsf’ format.

If the user chooses to use the ‘.bxsf’ file directly as input, he/she must enter the instructions and click the ‘bxsf file type’ button. In this way, when in the following tabs the GUI will

save the input instructions, it will teach *ElecTra* to use a bxsf file as bandstructure, which will be saved in '.mat' format when *ElecTra* will run. At this stage, the bxsf file name is the full name of the file *without* the extension, alat is the lattice parameter in nanometres, and the interpolation factor is the times the mesh wants to be increased, as detailed in **Section 4**. If this number is lower than or equal to one, the interpolation will not be done. Finally, if the SOC has been used in a non-magnetic system (so that spin-up and spin-down bands are identical) this must be told to remove the duplicate bands. Different,y from what explained in **Section 4**, here the material-name is not required because it is entered in the other tabs.

- **The Basic Mode**

The screen shot for the Basic GUI mode is shown below, **Figure 8**. The code considers all quantities in International System units excepted the energy which is in eV.

Input file creation app - Basic

1) General

Material Name Dimensionality

2) Scattering mechanisms

constant time [fs]

scattering vector cutoff

3) Transport specifics

Temperatures Fermi levels Energy step

new band gap

number of parallel cpu number of nodes cluster

account queue wall time

Figure 8: GUI for the Basic input mode.

It has four levels of information, detailed below.

1) General

These include the material name, which is used to load the $E(k)$ file and the scattering parameters file, and the dimensionality of the system, that is specified as a number: 2, or 3. Then the user chooses the way the reciprocal space is scanned to build the constant energy surface. `Tetrahedra Scan` adopts the tetrahedron methods, using Delaunay Triangulation, while '`NN Scan`' performs the “fast k -scan” described above in the paragraph “**Extraction of the constant energy surfaces**”. The '`NN Scan`' scheme is developed to speed up the whole calculation. It is around 15× to 30× faster in 3D bandstructures while in 2D bands the speed is comparable.

2) Scattering mechanisms details

The user can choose the constant relaxation time (CRT) approximation and input a constant relaxation time in femtoseconds. In this case, the code will also perform a constant mean-free-path (CMFP) approximation calculation using 5 nm as CMFP value (although the user needs to have in mind that this in general is not consistent with the CRT value). Otherwise, the user can select the scattering mechanisms as explained above via selecting the corresponding button.

Then the user can choose whether to save the relaxation times for all k -states or not, as these quantities are memory intensive. If this button is selected, a structure type data containing all the relaxation times (RT) for each k -state and each mechanism will be saved. In this case the code will save three files: `RelaxTimes_separate`, `RelaxTimes_Matthiessen`, `RelaxTimes_IIS`, with the additional specification of the material name, reciprocal space scan type and carrier. (These data, especially the third one, can take several tens of Gb).

However, because to calculate the TE coefficients in Eq. 1, only the TDF is required and the storing of the k -state dependent relaxation times is memory intensive, the user can choose to not save them by *avoiding* selecting the button ‘save relaxation times’. In this case, the relaxation times are never saved. The code is parallelized for the energy values, thus when the relaxation times are not saved each processor of the parallel pool returns to the master cpu only the energy dependent quantities and the relaxation times of each k -state are not stored. This means lighter communication and less memory required by the master processor, nothing more than the other processors. Otherwise, each processor returns to the master processor all the state-specific times, which will require additional tens of Gb to be allocated for the master processor. In turns, this has the advantage to store each specific relaxation time, for each state

on the constant energy surfaces and each scattering mechanism. This is useful to look at very specific details or to disentangle odd behaviours or debug.

3) Transport specifics

Here the user inputs the temperatures (in K) and the Fermi level positions (in eV) for which the simulation will be performed. In semiconductors, the Fermi level zero reference is automatically set at the majority carriers band edge, so negative Fermi level values place the Fermi level in the gap and positive values place it into the band. This is always the case for both band types, because when the user wants to simulate hole transport, the valence band is automatically flipped in positive energies. Thus, positive values of Fermi levels are always into the bands, regardless whether the conduction or valence bands are simulated. For the case of full bipolar transport, the same entered Fermi level values are used for both, as detailed below.

Fermi levels and temperatures are entered as vectors, for instance, in the case of the Fermi levels, the user writes `'-0.2, -0.1, -0.05:0.01:0.05, 0.1, 0.2'`. This returns the following set of Fermi levels: `-0.2, -0.1, -0.05, -0.04, -0.03, -0.02, -0.01, 0.0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2`.

As each Fermi level corresponds to a specific doping level, it is preferable to input the Fermi level rather than the doping, because this gives to the user a direct control over energy references. Indeed, in new materials it is not possible to know what doping levels correspond to degenerate or non-degenerate conditions.

In practice, the doping level in the material does not change with the temperature (in the extrinsic regime) due to charge neutrality. Thus, by increasing the temperature the Fermi distribution broadens, which forces the Fermi level to shift down away from the bands to keep the carrier density constant and equal to the doping density, as depicted in **Figure 9**. Here a parabolic band is sketched with two different impurity densities $N_i^{(1)}$ for non-degenerate doping

in (a) and $N_i^{(2)}$ for degenerate doping in (b). The dashed lines represent the Fermi level positions at 300 K and 900 K. The Fermi distributions for these temperatures are depicted in (c) – blue line for 300 K and red line for 900 K. The broadening in the Fermi distribution at higher temperatures results in more states being occupied around the Fermi level E_F . Thus, to keep the carrier density constant, the Fermi level shifts down. This shift is more pronounced for the non-degenerate conditions.

The entered Fermi levels refer to the 300 K temperature. A unique carrier density and doping impurity density are found for each Fermi level. The code assumes the semiconductor to be in the extrinsic region at that temperature, i.e. all the dopants are ionized and all the carriers come from the dopants. At the rest of the temperatures entered, the code shifts those Fermi levels for each temperature such that the impurity density (and charge density) remains constant to their value at the lowest entered temperature. When bipolar simulations are performed, thermally excited minority carriers are considered, so that at higher temperatures (in respect of the bandgap) the carrier density can be higher than the impurity density, which is the quantity kept constant.

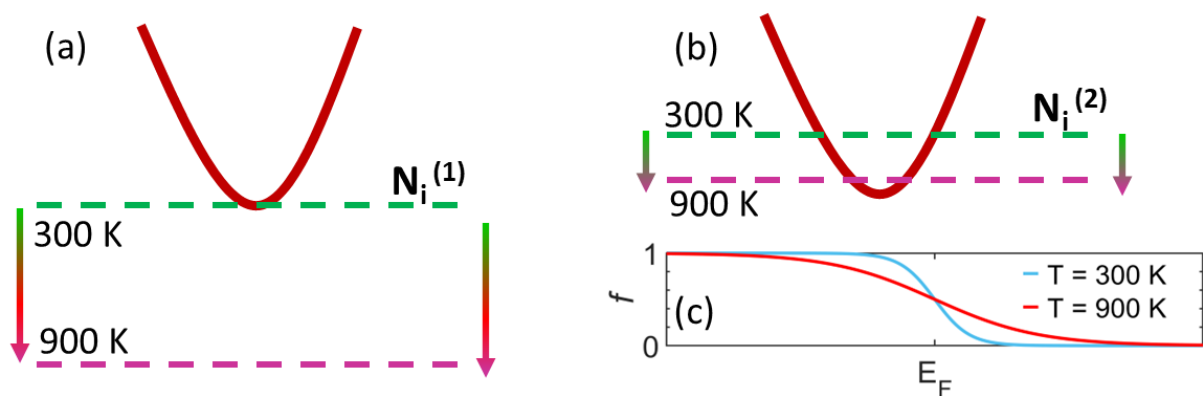


Figure 9: The shift in the Fermi level as the temperature increases from 300 K to 900 K, for: (a) non-degenerate conditions, (b) degenerate conditions. In (c), the Fermi distribution is shown for the two temperatures.

However, a user might want to fix the Fermi level positions for all temperatures, being aware that these will correspond to different doping/carrier densities. In this case the user can “lock” the Fermi levels by selecting the ‘Lock Fermi levels’ button, so that the entered Fermi levels will be used at all temperatures without considering E_F shifts.

NOTE: In the case of scattering with phonons, the Fermi levels do not enter in the evaluation of the scattering rates, so many E_F positions can be used for fine resolution of the transport results without computational expense. In the case of the IIS or of the screened POP scattering mechanisms, a different scattering rate is evaluated for each Fermi level value, so increasing the number of Fermi levels linearly increases the computation time. Also note that in the case of including IIS, the code also returns the values for the phonon-limited transport. For this, a finer array of 200 Fermi levels starting from the minimum and ending at the maximum entered ‘Fermi levels’ is used.

Then, the user inputs the ‘Energy step’ in eV, at which the integration energy step will be performed. As the code performs the integrals following the trapezoidal rule, an energy step of 0.002 or 0.003 eV is generally sufficient. At this version of the code the energy grid is uniform.

Then, the user chooses the majority carrier type: ‘electrons’ or ‘holes’. In the former case the conduction band will be considered, whereas in the latter case the valence band.

NOTE: In the case of hole transport in the VB, the bandstructure will be flipped in energy so that the VB will be treated as a CB, with the zero set at the VB edge and the VB energy flipped to positive. A positive Fermi level will correspond to degenerate conditions for both VB and CB. This must be done before running the *ElecTra* code.

If unipolar transport is considered, i.e. only electrons or only holes, the user should NOT select (leave unselected) the 'bipolar transport' button. If it is selected, both the conduction and valence bands will be considered, one as the majority carrier band and one as the minority band.

When the 'bipolar single carrier' is selected, the code will perform the bipolar calculation only for the selected majority carriers. This reproduces an experimental situation where a specimen is n -doped or p -doped. When this button is NOT selected, the code performs a full bipolar calculation, i.e. the code will compute the transport properties for the n -type case, electron majority and hole minority, as well as p -type case, hole majority and electron minority.

In the unipolar case, the Fermi level is entered with respect to the band edge, which is set to 0 eV. In the case where the bipolar transport is chosen, still the 0 eV reference is set to the band edge of the majority band. When the user performs a full bipolar calculation of both n -type and p -type transport, in the input instruction only the Fermi levels referred to the conduction band must be entered, and the code will replicate them for the valence band. At this level, the zero is the band edge so that, for example, a maximum Fermi level of 0.2 means that the highest considered doping will be the one with the Fermi level 0.2 eV into the conduction band or the valence band for n - and p -type respectively. The intrinsic Fermi level will also be computed and added by default. When the results will be saved, the zero of the Fermi levels will be the intrinsic Fermi level and not the band edge.

In the case the user enters Fermi level values below the intrinsic level, *ElecTra* will shift the inserted Fermi level values that reside in the bandgap, in the case where any of these values are below the intrinsic level. For example, let us consider a semiconductor with a bandgap of 0.2 eV and an intrinsic Fermi level (the Fermi level for which the number of electrons is identical to the number of holes) at -0.12 eV. If the user enters a value of -0.3 eV for the Fermi

level, the inserted Fermi level is shifted to -0.12 eV. In the case of many entered Fermi level values residing below the intrinsic Fermi level, the code will act as follows:

1. the lowest entered Fermi level is set at the intrinsic Fermi level,
2. the energy range between the intrinsic Fermi level and the band edge is divided in number of intervals so that the number of the Fermi levels in the bandgap is the same as entered.
3. the Fermi levels into the band are not changed.

After this, if the 'bipolar single carrier' is not selected so that the code computes the bipolar transport coefficients for both *n*- and *p*-type cases, *ElecTra* will replicate these values for the valence band. Importantly, the zero of the Fermi levels always refers to the band edge.

In the case of bipolar transport, the user can choose to change the band gap and enter a desired bandgap value. This can obviate to the difficult in computing reliable band gap values within DFT and generally offers more flexibility to the user. The new band gap must be entered with as a positive value in eV.

4) Parallelization

Finally, the parallelization details are selected. The 'number of parallel cpu' is the number of workers of the parallel pool on which the code will run. This depends on the HPC cluster facilities, and the user has to request a corresponding number of processors when launching the code on the cluster. The user can specify less CPUs than the total in the node, or the user can run two calculations on the same node. The user will also need to request a consistent number of CPUs when launching the code on a supercomputing node. This is done in different ways according to the HPC facility used. In Section 8 we provide an example of the required bash script. If the user specifies a CPU number higher than the one available on the HPC node, then MATLAB® will return an alert and will set this number to 12.

If the other fields, ‘number of nodes’, ‘cluster’, ‘account’, ‘queue’ and ‘wall time’, are left blank, *ElecTra* will teach MATLAB® to open a ‘pool’ of processors on a single node, in ‘local’ configuration. To work on more nodes in a cluster, the other details must be entered. First, an integer greater than 1 must be entered in the number of nodes, then the ‘cluster’, ‘account’, ‘queue’ and ‘wall time’ information must be entered, according to the specifics of the cluster the user is working on; these are extremely cluster-dependent and cannot be generalized.

ElecTra code is parallelized in the energy, each processor of the parallel computes all the quantities referred to that energy, each energy is printed in the .log file during the execution, together with lines telling the stage of the calculation.

The GUI – Advanced Mode

Figure 10 shows the GUI for the Advanced Mode. We explain here only the differences between the ‘Basic’ and ‘Advanced’ modes.

1) General part

The ‘spin resolved bands’ button is selected if the bands are not spin degenerate. This can be the case when the spin-orbit coupling is included in the DFT calculation. Please, note that if the material is not ferromagnetic, the DFT code might return couple of identical bands which are spin resolved. To fasten the calculation, the user can manually remove the identical bands and assume the bands are not spin resolved.

The ‘ π/a ’ button is selected if the k -mesh is in units of π/a , with a being the lattice parameter instead of international system (SI) units, that is 1/m. This can be useful if the user wants to make simulation on artificially built parametric bandstructures rather than DFT computed ones.

The ‘multivalley’ option allows the user to provide different deformation potentials for different initial and final bands. See **section 6** for an example related to this option.

Input file creation app - Advanced

1) General

Material Name Dimensionality Tetrahedra Scan NN Scan

spin resolved bands pi/a units multivalley save relaxation times

2) Scattering mechanisms

ADP ADP IVS ODP IVS

POP screening POP IIS IIS interband

Alloy analytical overlap integrals scattering vector restriction scattering vector cutoff

3) Transport specifics

Temperatures Fermi levels Energy step

Lock Fermi levels electrons holes metal

bipolar transport bipolar single carrier change band gap new band gap

4) Parallelization

number of parallel cpu number of nodes cluster

account queue wall time

Figure 10: GUI to prepare the input file in the ‘Advanced mode’

2) Details of scattering mechanisms

There are only two differences in respect to the ‘Basic’ mode

- The ‘screening POP’ can be selected for the POP, which adds a screening term to the POP scattering rate, as in the case of IIS. This makes the computation much slower and the impact is generally little.

- The '*IVS*', Inter-Valley Scattering, describes the *inter*-band inelastic scattering. It allows several different processes with different scattering strengths (deformation potentials).
- The '*interband IIS*', allows the IIS as *inter*-band in addition to *intra*-band.

NOTE: when the '*multivalley*' option is NOT selected, the ODP is both *intra*- and *inter*-band (as commonly considered in the valence band of silicon). About the IVS, a certain number of equivalent bands can be considered to mimic multi-valley scattering (as for example the six valleys in the CB of Silicon), inserted as multiplicity *Z* in the '*scattering parameters*' file. Thus, computationally, the carrier scatters into the same band, considering the multiplicity entered as degeneracy in the '*scattering parameters*' file. In the case of Si, this allows to simulate the *f*- and *g*- processes of electron transport in Si without the need to identify if the process should be of *f*- or *g*- type according to the location in the BZ of each pair of initial and final states. Instead, only the degeneracy needs to be inserted, in the case of Silicon, because the six valleys belong to the same band, 1/6 for *f*-processes and 4/6 for *g*-processes.

For other materials, for which the valleys are different, like the Ge and GaAs conduction bands, the '*multivalley*' flag must be selected, with the possibility to use different scattering parameters values according to the initial/final bands pair. When the '*multivalley*' option is selected, ODP is *intra*- band only.

3) Transport specifics

Here the '*metal*' option can be selected as well. This describes the case where the system is metallic rather than semiconducting, see **Figure 11** for a schematic. While in a semiconductor the charge transport can occur mainly in the CB with electrons as the main carriers (*n*-type) or in the VB with holes as the main carriers (*p*-type), in a metal the Fermi level

E_F falls in the middle of a band and there is absence of bandgap. In this case the distinction between the conduction and valence band is not made, the bands are not shifted to zero and the inserted Fermi levels are directly used without shift. Thus, when inputting the Fermi levels, the user should know the precise energy range coming from the DFT bandstructure calculation.

Important note: in this case the zero in the Fermi array will correspond to the Fermi energy from the DFT. When working with *semimetals*, please, use this modality. Materials with a bandgap *smaller than* the optical phonon energy, so that an inelastic scattering between the CB and VB can occur, must be treated as *semimetals* and the 'metal' button must be selected.

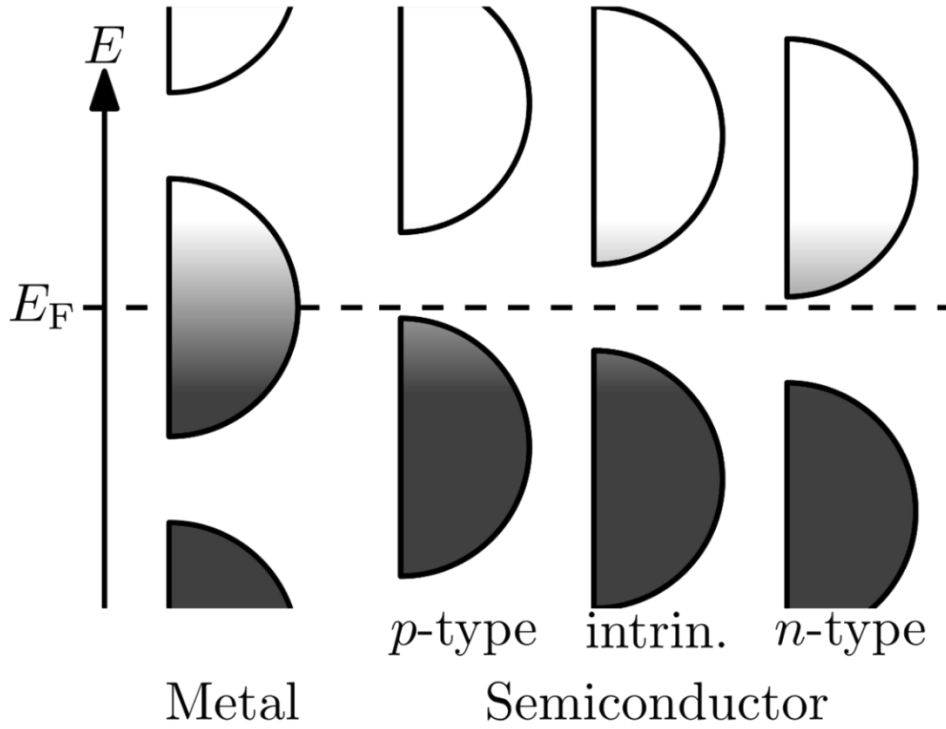


Figure 11: Schematics of the electron energy levels of a metal, left, and of a semiconductor, right, from [19].

The input_file_ELECTRA_v1.m file

Below we provide an example of the input file for *ElecTra*, when the user prefers to work with a text file rather than a GUI. The example is for TiNiSn, when using a bxsf file for the bandstructure, and considering ADP_IVS, ODP, and IIS scattering. The instructions map exactly what explained in the *Advanced* mode of the GUI. The parallelization instructions are for the case of a run on the Galileo100 cluster from CINECA, using the individual account of one of the authors. *ElecTra* will read and run this script as first step. All these instructions will be contained in a structure type variable named input_info_struct, with a structure field for each instruction and the bandstructure matrix as well.

```
% 1) GENERAL SETTINGS
material_name = 'TiNiSn' ;
file_type = 'bxsf'; % bxsf or mat
% for .bxsf files
bxsf_file_name = 'TiNiSn_fs.bxsf'; %
alat = 0.417; % in nm
SOC_not_magn = 'no'; % yes or no % yes ONLY if use SOC but
system is not ferromagnetic
interpolation_factor = 2 ;

dimensionality='3D'; % 1D, 2D, 3D

scan_type='kScan'; % kScan or DT . This is the way the k(E) is
extracted, kScan represents the 'NN' mode

spin_resolved = 'no';

k_units_SI = 'yes'; % in 1/m , as extracted from bxsf, okay
for DFT bands
k_units_pi_a = 'no'; % usable for numerically built bands

%-----
%-----

% 2) SCATTERING MECHANISMS
ADP = 'no'; % Acoustic Deformation Potential, INTRAValley
ADP_IVS = 'yes'; % INTRA- and INTER- valley
k_restriction = 'no'; % only for the ADP_IVS TO BE EXTENDED
WHEN VALIDATED
```

```

k_restriction_cutoff = 0.2; % Portion of the BZ to be
considered

IVS = 'no'; % intervalley scattering

ODP = 'yes'; % Optical deformation potential

POP = 'no'; % polar optical phonon, constan freq., approx.
screening_POP = 'no'; % this is not necessary and makes the
code much slower

IIS = 'yes'; % this flag makes the IIS calc. much less RAM
consuming but more time consuming
IIS_interband_flag = 'no';

Alloy = 'no';

overlap_integrals_analytical = 'no';

save_RTstruct = 'yes';

multivalley = 'no'; % different types of valleys with
different Deformation Potentials

constant_tau = 'no'; % constant relax. time approx. (CRTA)
tau_const = 1e-14; % in s; value of the scattering time in the
CRTA

%-----
%-----

% 3) TRANSPORT CONDITIONS

% 3.1) FERMI LEVELS AND TEMPERATURES
EF_array = [-0.2,-0.1,-0.06:0.02:0.14,0.2]; % in eV
% Fermi array is in respect to the band edge that will be set
to zero,
% negative values mean Fermi into the gap, positive, into the
band (degen.)

T_array = 300:100:900; % in K
Fermi_shift_flag = 'yes'; % to shift the Fermi level with the
temperature

% 3.2) CARRIERS AND ENERGY
carriers = 'electrons'; % 'holes' or 'electrons'

```

```

bipolar_transport = 'yes';

bipolar_single_carrier = 'no' ;

change_band_gap = 'no' ;
new_band_gap = 1 ; % in eV

metallic = 'no'; % proper metal with Fermi in the bands free
carriers without impurities

Estep = 0.005 ; % energy step in eV, 2 meV is often largely
sufficient

%-----
%-----

% 4) PARALLELIZATION
type_of_parallelization = 'local'; % 'local' (one node) or
'cluster' (more than one node), 'MATLAB Parallel Cloud' is
forecasted
max_number_of_cpu = 10 ; % maximum number of cpu in the single
node
% only for cluster
% number of requested nodes
number_of_nodes = 2;
cluster_name = 'galileo100 R2020b';
account_name = 'IscrC_sd-FRAME';
queue_name = 'g100_usr_prod';
wall_time = '03:40:00';

%-----
% ----- end of the input instructions -----
% -----

save Inputs

```


6. Scattering parameters file preparation

As in the case of the input calculation instructions, another GUI is provided to input the scattering parameters. As for the input file, the user can either input a written .m file and a .mat file created by the GUI. The code will automatically check for both and *will prioritize the .m file*. We remind that *ElecTra* automatically distinguishes between CB and VB, when a metallic bandstructure is used, the user enters only the values for CB.

In the case the user prepares the .m file, this must be a text file with ‘.m’ extension named:

```
'scattering_parameters_',material_name, '.m',
```

e.g.

```
scattering_parameters_TiNiSn.m.
```

GUI

This GUI composes of 4 tabs, two for the ‘General’ mode and two for the ‘Multivalley’ mode. The difference between the two modes is that in the ‘General’, the same deformation potential and phonon energy values are used for all the conduction or valence bands, whereas in the ‘Multivalley’ mode different values can be used for the different pairs of initial and final bands. Each mode has two tabs, one for all the material parameters required for the calculation of the scattering rates, except the alloy scattering, and one for the case where the alloy scattering is considered. In the case of the alloy scattering between two ‘A’ and ‘B’ parental compounds, the parameters in the first tab refer to the parental compound ‘A’ and the additional required parameters, together with the ones for the parental compound ‘B’, are entered in the second tab. The third and fourth tabs correspond to the first two tabs for the multivalley mode, which allows the user to use different scattering parameters for different pairs of initial and final bands. Each band in the E_k variable must be labelled by a number in the ‘bands_type’ entry

and each scattering parameter will be represented by a matrix whose element i,j corresponds to the relative scattering transition between bands i and j .

We first describe the first tab; when the Alloy scattering is not considered, the user needs only the first tab, when the user desires to consider the Alloy scattering as well, also this tab needs to be used as explained below.

The screenshot shows a MATLAB-style GUI window titled 'UI Figure'. It has four tabs: 'General', 'General- Alloy', 'Multivalley', and 'Multivalley - Alloy'. The 'General- Alloy' tab is active. The main area is titled 'Scattering parameters file creation app - Basic'. It contains the following input fields:

- Material Name:
- Dimensionality:
- mass density [Kg/m^3]:
- Bulk modulus [Pa]:
- Share modulus [Pa]:
- sound speed [m/s]:
- ADP, CB [eV]:
- ADP, VB [eV]:
- ODP, CB [eV/m]:
- ODP, VB [eV/m]:
- ODP phonon, CB [eV]:
- ODP phonon, VB [eV]:
- IVS, CB [eV/m]:
- IVS, VB [eV/m]:
- IVS phonon, CB [eV]:
- IVS phonon, VB [eV]:
- POP phonon [eV]:
- static dielectric constant:
- high frequency dielectric constant:
- s.d. array:
- imp. charge:
- thickness (2D) [nm]:

A button labeled 'create scattering parameter file' is located at the bottom right of the form.

Figure 12: Tab for the GUI for the scattering parameters. When the ‘Alloy scattering’ is used, and the alloy is between material “A” and “B”, the data for material “A” are entered in this tab.

The material name and dimensionality operate as in the input GUI, but then a series of material parameters can be inserted, according to the scattering mechanism under consideration. The dielectric constants are inserted as unitless numbers relative to ϵ_0 . CB and

VB refer to the Conduction Band and the Valence Band, respectively. The phonon frequency for ODP and POP in the ‘Basic’ mode are the same for CB and VB.

The speed of sound can be inserted in two ways: the bulk modulus K and the shear modulus G can be inserted to compute the speed of sound, [1] *alternatively*, the speed of sound can be directly inserted; in that case K and G will be ignored. The code allows for more ODP or POP mechanisms. If the more than one of such mechanisms is to be included, the deformation potentials and phonon energies will be inserted one by one and separated by a comma, e.g. 2, 3.5, 4 for ODP deformation potential values and 0.030, 0.032, 0.040 for the phonon energies.

Finally, there are two fields which can also left blank if not necessary. The s.d. array is an array of value which is required when the bands are spin resolved, s.d. = spin degeneracy. The array must contain two different number which identify the two spin orientations, e.g. ‘0 1 0 1 0 1 0 1’ can be an options for the case of eight bands. The order must be the same as the bands appear in the E_k matrix. This is important when spin orbit coupling is considered, and the bands are spin-resolved. The user can use whatever numbers, the code allows the scattering only between bands with the same number in the s.d. array because the considered scattering mechanisms do not cause spin flip. [3] The consideration of spin flip scattering in ferromagnetic material is under development. The thickness, in nanometres, is used only for 2D bandstructures, in this case is mandatory but for 3D bandstructures can be left blank.

When Alloy scattering is considered, we assume an alloy between material “A” and material “B”. The material parameters inserted in the first tab (see **Figure 12** above) are for material “A”, while in this part of the GUI the user enters some parameters for the alloy and the material “B”. This includes the fraction of material “B” and the band gap difference (see Eqs. (4)), and the unit cell volumes of the two materials, in \AA^3 .

UI Figure

General General- Alloy Multivalley Multivalley - Alloy

with Alloy scatt., no multivalley - material "A" data in the tab "Basic"

fraction of material "B" (0 to 1) band gap difference [eV]

cell volume - "A" [Å³] cell volume - "B" [Å³] mass density of "B" [Kg/m³]

Bulk mod. of "B" [Pa] Share mod. of "B" [Pa] sound speed of "B" [m/s]

ADP, CB, material "B" [eV] ADP, VB, material "B" [eV]

ODP, CB, material "B" [eV/m] ODP, VB, material "B" [eV/m]

ODP, CB phonon, material "B" [eV] ODP, VB phonon, material "B" [eV]

IVS, CB, material "B" [eV/m] IVS, VB, material "B" [eV/m]

IVS, CB phonon, material "B" [eV] IVS, VB phonon, material "B" [eV]

POP phonon, material "B" [eV] static dielectric constant, material "B"

high freq. dielectric const., material "B"

Figure 13: Tab for the GUI for the scattering parameters when the 'Alloy scattering' is used. The alloy is formed between "A" and "B". The GUI contains some alloying information and the parameters for material "B".

NOTES on the alloy scattering:

- in the case of more than one ODP or POP processes, the number of processes in the materials "A" and "B" must be the same. If, as in the example above, three ODP processes are considered for material "A", also three ODP processes will be entered for material "B" to have consistency in the calculation of the alloy parameters.
- *ElecTra* does not "mix", or "alloy", the band structures; a single band structure, specific for the alloy, needs to be used as input data. This means that in the alloy between material "A" and material "B", the scattering parameters are linearly interpolated

according to the composition, but not the bandstructure. In this way the code has been validated for the SiGe alloys with details in the validation section. In the case of a spin resolved bandstructure, the s.d. array is entered in the ‘General’ tab only.

Now we present the tabs for the ‘Multivalley’ operation, which map the former two tabs with only two differences.

1) In the first of the two ‘Multivalley’ tabs, **Figure 14**, the ‘bands type’ must be entered, for CB and VB. These are arrays which identify the bands, we remind that *ElecTra* automatically distinguishes between CB and VB. Important: the user must *always enter the labelling numbers for both* if both are present in the Ek or bxsf file, even if the calculation is unipolar. This is important because *ElecTra* labels all the bands contained in the .bxsf file or in the Ek variable. *When a metallic bandstructure is used, the user enters only the values for CB*, but they must cover all the bands in the Ek variable (or in the .bxsf file). Each band must be labelled by a number in the ‘band type’ entry and each scattering parameter will be represented by a matrix whose element corresponds to the relative scattering transition. For example, if we assume as an example that we have five CB bands of three different types, where the second and third are of the same type, and the fourth and fifth are of the same type, then the corresponding band type enter is an array 1,2,2,3,3 – the band number refers to how they appear in the matrix, i.e. their order number in the bxsf file. If there are three different VB bands, the user will edit 1,2,3 in the field even if the run is for electrons only.

2) The scattering parameters are matrixes where the position i,j contains the parameter related to the transition from band i to band j – in the example above these are 3×3 matrixes.

If the element is zero, the transition is read as forbidden, these matrixes must be symmetric for detailed balance reasons. [21] Because the ODP is intra-band only in this operation mode, for

this scattering mechanism the entry is an array where at each position we find the value for the ODP mechanisms in that band, in the example above with five bands of three type, it is a three elements array. In the case of more processes, this entry becomes a bidimensional matrix with a row for each process. For instance, a matrix for ODP with two processes and three types of bands, is entered like 3, 2, 5 ; 2, 5, 4 ; a 3×3 matrix for IVS mechanism can be entered like 0, 5, 6 ; 5, 4, 0 ; 6, 0, 5 .

The tab for the Alloy scattering in the ‘Multivalley’ mode is identical to the one for the ‘General’ mode but must be filled as described above for the ‘Multivalley’ option.

Figure 14: tab to enter the scattering parameters when the ‘Multivalley’ operation is chosen. A partial filling is shown, for the case of five bands of three types, two ODP processes and one IVS process. Note that all the bands (CB and VB) must be labelled.

Via GUI, it is possible to consider more processes only for ODP, while for IVS, if the user wants to use more than one process, he/she must use the scattering parameters file.

The scattering_parameters.m file

Examples of the script for the scattering parameters are provided in the “Data” folder. An example is `D_adp_e` for the deformation potential, where the ‘D’ stands for the deformation potential, ‘adp’ for ADP, and ‘e’ for electrons (CB). The phonon frequency is called, for instance, `hbar_w_odp_h`, where ‘hbar_w’ is for $\hbar\omega$, ‘odp’ for ODP, and ‘h’ for holes in the VB. According to the calculation under consideration for the carrier type, the code will search and use variables ending with ‘_e’ or ‘_h’. If the code does not find them, will search for the variable without the specified ending, and, if not found, an error will be returned and *ElecTra* will stop. The dielectric constant values must be entered for IIS and POP mechanisms, both the static and high frequency values, `k_s` and `k_inf` respectively. The `z_i` variable represents the charge per impurity.

- The “multivalley” case

In this case, each valley, or band, is indexed in the ‘bands_type_CB’ and ‘bands_type_VB’ variables, as described above for the GUI. Because the parameters now are matrixes, as explained above in the GUI section, these have a ‘_M’ termination, e.g. ‘D_odp_e_M’. In addition to the possibility from the GUI, for the IVS mechanisms more processes can be considered when the text file is edited, so that the parameters become three-dimensional matrixes where the third dimension is used for the processes. In example above with three types of bands, these are $3 \times 3 \times n$ matrixes, where n is the number of processes. For example, if we take the example above with three types of bands, and want to consider two IVS processes, we will write, for the deformation potential:

```
D_ivs_e_M = [ 0, 5, 6 ; 5, 4, 0 ; 6, 0, 5 ] *1e10 ;
```

```
D_ivs_e_M( :, :, 2) = [ 2, 4, 0 ; 4, 3, 1 ; 0, 1, 2 ] *1e10 ;
```

This will make `D_ivs_e_M` to be a $3 \times 3 \times 2$ matrix where in the ij position of the 3×3 layers we locate the deformation potentials for the transition from band i to band j and we have two IVS processes. This cannot be entered via GUI.

7. Computed quantities

Main quantities

The main goal of the code is to compute the transport coefficients: mobility, electrical conductivity, Seebeck coefficients, power factor. These are named, respectively, `mu`, `sigma`, `S`, and `PF`. These quantities are tensors, represented as structure type variables where each field is the tensor components, e.g. `sigma.xx`, `sigma.yy`, `sigma.zz`, `sigma.xy`, `sigma.yx`, `sigma.xz`, `sigma.zx`, `sigma.yz`, `sigma.zy`, following the i,j convention in Eqs. (1) to (3). The same holds for `S.xx`, etc. and `PF.xx`, etc.. In the same way, the mobility `mu.xx` etc., is computed as the ratio between the electrical conductivity and carrier density. All these quantities are 2-dimensional matrixes where the first index corresponds on the different Fermi levels and the second index on the different temperatures. Following the same scheme as above, the code also computes the lattice limited quantities, identified with an added ‘_ph’: `sigma_ph.xx`, `sigma_ph.yy`, `sigma_ph.zz`, `sigma_ph.xy`... `S_ph.xx`, `S_ph.yx`... `PF_ph.xx`, `PF_ph.xy`... `mu_ph.xx`, `mu_ph.yy`.... . These include phonon scattering and, if the case, also Alloy scattering.

In addition, the transport coefficients related to each electron-phonon scattering mechanism separately are computed. These are contained in the ‘_sep’ labelled variables, which is for ‘separate’, where there is a field for each mechanism and the nine fields for the nine components of the tensors. For instance: `sigma_sep.ADP.xx`, `sigma_sep.ADP.yy`, ... `S_sep.ADP.xx`, `S_sep.ADP.yx` ... `PF_sep.ADP.xx`, `PF_sep.ADP.xy`... `mu_sep.ADP.xx`, `mu_sep.ADP.yy`....; `sigma_sep.ODP.xx`, `sigma_sep.ODP.yy`, ... `S_sep.ODP.xx`, `S_sep.ODP.yx`... `PF_sep.ODP.xx`, `PF_sep.ODP.xy`... `mu_sep.ODP.xx`, `mu_sep.ODP.yz`... ; `sigma_sep.POP.xx`, `sigma_sep.POP.xy`, ... `S_sep.POP.xx`, `S_sep.POP.yx`... `PF_sep.POP.xx`,

PF_sep.POP.xy... mu_sep.POP.xx, mu_sep.POP.yy...; sigma_sep.Alloy.xx
sigma_sep.Alloy.xz etc..

The above quantities depend on temperature, contained in the one-dimensional array called `T_array`, and Fermi levels, contained in the 2-dimensional matrix `EF_matrix`. The first index of the `EF_matrix`, as in the case of the transport coefficients, correspond to the doping level, and the second to the temperature. This is because the code considers the Fermi level movement with the temperature, thus, for example, in a given row of the matrix there are the Fermi levels which corresponds at the same doping density, at different temperature. [1, 2] For example, if the user enters: `'-0.2, -0.1, -0.05:0.01:0.05, 0.1, 0.2'`, fifteen Fermi levels will be considered, which correspond to 15 doping levels. To have the same doping density at each entered temperature, the 15 entered Fermi levels will be shifted at each temperature to get the same doping density. If 6 temperature values would have been entered, the `EF_matrix` turns out to be a 15×6 matrix. Because the `EF_matrix` maps the impurity density and the carrier density (and this mapping is different in the unipolar and bipolar cases), other two 2-dimensional matrixes exist: `n_carrier` for the carrier density and `N_imp_matrix` for the impurity density. The quantities for the lattice limited transport coefficients are also computed and saved in the following matrixes: `EF_matrix_ph`, `n_carrier_ph`, `N_imp_matrix_ph`.

NOTE: when the calculation is performed in the full bipolar fashion, that is by computing both *p*-type and *n*-type in the same run, the `EF_matrix` is centred at the intrinsic Fermi level.

Secondary quantities

Carrier energy dependent quantities are also computed, for each band separately and then combined. These are the Transport Distribution Function (TDF), the relaxation times (τ_E), the mean-free-path (MFP). The former is labelled according to the *ij* convention: TDF.xx, TDF.xy, etc., following the same notation concept as in the transport coefficients. These are three dimensional matrixes where the first dimension runs along the carrier energy, the second along the Fermi level, and the third along the temperature. The components for each singular bands are labelled as TDF_n.xx, TDF_n.xz, etc. . These are four dimensional matrixes, where the dimensions run along: energy, band index, Fermi level, and temperature.

As above, structure variables ending with _ph, are computed for the lattice limited transport: TDF_ph.xx, TDF_ph_n.xx, etc.. These are normally two-dimensional matrixes (carrier energy, temperature) or three dimensional when the data are for each band, ‘_n’ labelled. But when the charge screening is added in the POP, because the screening depends on the doping level, these are three- and four-dimensions matrixes.

Similarly, they are computed also for each individual considered scattering mechanism, to allow comparison between contributions, with the same nomenclature: TDF_sep.ADP.xx, TDF_sep.ODP.xx, TDF_sep_n.ADP.xx, etc.. As for the phonon limited case, generally these are two dimensions matrixes (carrier energy, temperature) but the POP field has one more dimension, corresponding to the Fermi level/doping density, when the charge screening is added to POP scattering.

The relaxation times (τ_E) and the mean-free-paths (MFP) are computed in a similar way, along x, or y, or z, as in equations (6) and (7). Thus, there will be: $\tau_E.x$, $\tau_E.y$, $\tau_E.z$, $\tau_E.ph.x$, $\tau_E_sep.ADP.x$, $\tau_E_n.x$, $\tau_E_ph_n.x$, $\tau_E_sep_n.ADP.x$... etc. , and MFP.x, MFP.y, MFP_ph .x, MFP_sep.ADP.x, MFP_sep_n.x, MFP_ph_n.x, MFP_sep_n.ADP.x... etc. .

Other quantities, derived from the construction of the constant energy surfaces, the $E(k) \rightarrow k(E)$ process, are also available to the user. These are the energy dependent Density of States (DOS) and the band velocity. They are one dimensional arrays of the size of the energy interval, `E_array`, and are named `DOS_tot` and `V_tot` respectively. The variables named `DOS` and `V` contain the DOS and band velocity information for each band separately.

In the case of bipolar transport, all the band dependent quantities are distinguished between “majority” and “minority”. The energy intervals for these quantities are the `E_majority` or `E_minority` for each band type. In the case of a ‘full’ bipolar run, when both n -type and p -type cases are computed in the same simulation, the ‘majority’ label refers to the conduction band.

All the quantities are stored in a .mat file named `'TE_', material_name, 'scan type, '_', carrier, '.m'`, eventually `'bipolar'` is added in the case of bipolar transport, for example: `TE_TiNiSn_kScan_electrons.mat` or `TE_TiNiSn_kScan_electrons_bipolar.mat`. When a bipolar calculation is run considering both n -type and p -type cases in the same simulation, the ‘majority’ label refers to the conduction band and the word ‘full’ is added to the name: `TE_TiNiSn_kScan_full_bipolar.mat`. For the CRT and CFMP calculations, the corresponding indications are added to the file name. The file is saved in the working directory, where also the ‘run’ file is located.

NOTE: because the conduction and valence bands often have a different number of bands, when a bipolar calculation is run the bands-specific quantities, labeled with ‘_n’ in the name, cannot be computed as global quantities. However, *Electra* code saves them as separated variables for the conduction and valence bands, labelled with ‘_n_majority’ or

'_n_minority' in the name. When a single carrier bipolar calculation is run, the former refers to the chosen main polarity, in the case of a full bipolar run, the former refers to the conduction band.

a. Plotting the results with the plot GUI

In addition to the usual MATLAB® instruction, the data can be plotted using a GUI named 'ElecTra_Plot'. The app plots the selected quantities in a graph inside the app and in an independent Figure. In addition, the app saves the plotted data in a .csv file. The .csv file is designed to be opened with programs like Origin® and contains two or more columns where the first is the x-axis variable and the other(s) is (are) the y-axis quantity. The header contains the column name and eventual additional specification.

The app composes of two tabs, the first is designed to plot the charge transport coefficients whereas the second is conceived to plot the energy dependent quantities.

The first tab, devised to plot a thermoelectric coefficient, TE coefficient, appears as below:

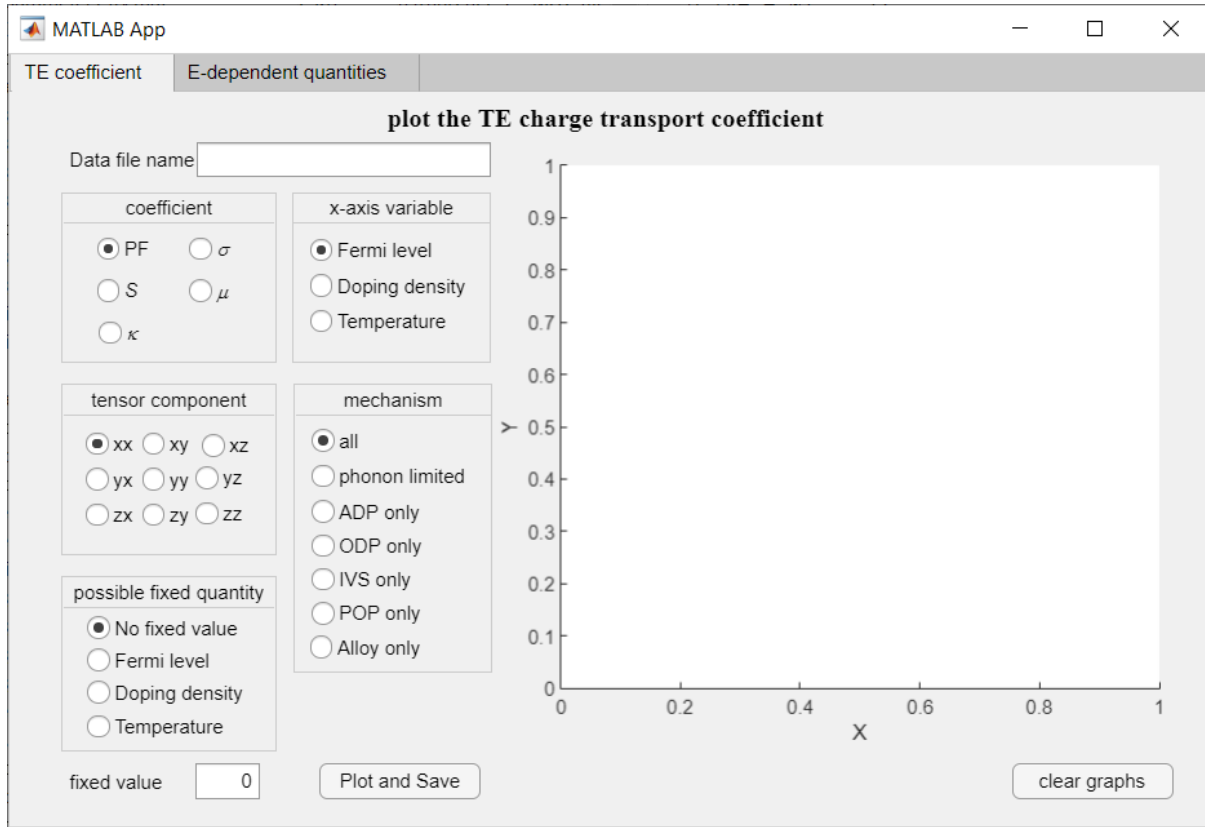


Figure 15: tab to plot the TE coefficients.

The user enters first the file name to be opened, which contains the *ElecTra* data. Then, the user selects the transport coefficients to be plotted: PF (Power Factor), σ (electrical conductivity), S (Seebeck coefficient), μ (mobility), κ (electronic thermal conductivity); this will be displayed in the y-axis. The user chooses also a tensor component. After this, the user chooses if displaying the overall transport coefficient or only the contribution from one transport mechanism, e.g., what is the PF if only ADP were considered. When the x-axis is the doping density and the transport simulation are a full bipolar case, the carrier density for *p*-type is displayed as a negative value.

Finally, we note that a transport coefficient usually depends on more than one quantity; it usually depends on both Fermi level position (carrier concentration) and temperature. Thus, if the Fermi level is selected as x-axis variable, a transport coefficient for each temperature used in the simulation will be displayed. If the user wants to see the value for one temperature only,

the ‘possible fixed quantity’ can be selected, in this case ‘Temperature’, and the temperature of interest can be entered in the ‘fixed value’ space. Alternatively, if the x-axis is temperature, a single value of Fermi level or carrier density can be chosen. The app will plot the data for the fixed value closer to the edited one.

NOTE: in the ‘fixed quantity’ field, the Fermi level is in eV, the Doping density in cm^{-3} , the temperature in K.

The ‘Plot and Save’ button will plot the data in the app graph, in an independent Figure, and save the .csv file which will have the name of the material and of the transport coefficients, e.g., ‘TiNiSn_PF_xx.csv’. The ‘clear graphs’ button clears both the graphs.

The second tab to plot the energy dependent quantities, E-dependent quantities, appears as below:

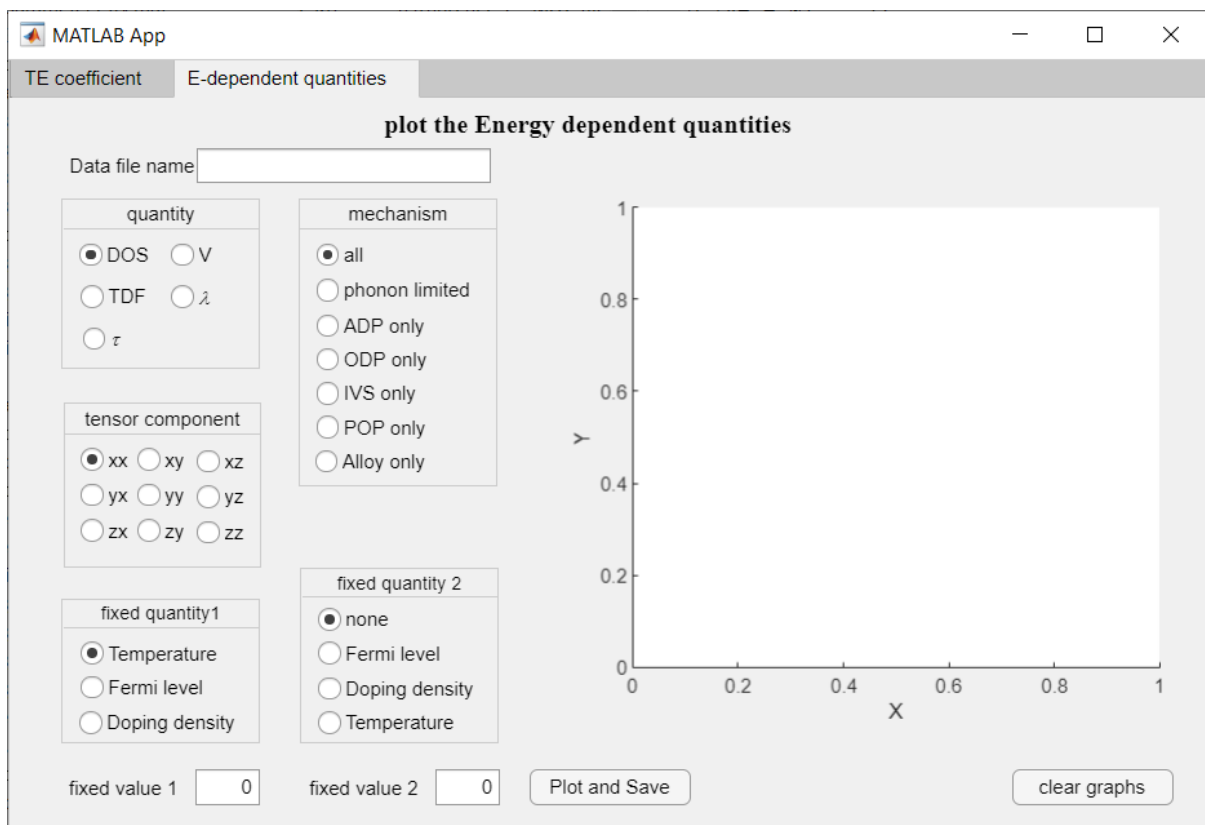


Figure 16: tab to plot the energy dependent quantities.

Like in the other tab, the user first enters the name and chooses the quantity to plot; here these are DOS (density of states), V (band velocity), TDF (transport distribution function, eq. 2), λ (mean free path, eq. 6) and τ (relaxation time, eq. 7). The tensor component is selected as in the other tab, even if only the TDF has a full tensorial character. For λ and τ only the first letter of the tensor component is used, and for DOS and V no tensor component is considered. The scattering mechanism is selected as in the other tab.

In this tab the x-axis is always the carrier energy, so some of these quantities have three variables, as displayed in eq.s (2), (6) and (7). For graphical reason, at least one value must be fixed, which can be the Temperature or the Fermi level (Doping density), and has to be selected in the “fixed quantity 1” menu and defined in the field below. As above, the app will plot the quantities for the fixed values closer to the one entered; the units are as above: K for temperature, eV for Fermi level, cm^{-3} for doping density. Eventually the user can opt for a second fixed value, different from the first. In this way the user will see only one line for the chosen temperature and Fermi level (doping density) values. The DOS and V do not require any fixed value as these are bandstructure quantities.

The ‘Plot and Save’ and ‘clear graphs’ buttons work as for the TE coefficient tab.

8. Run the code

a. on a supercomputing facility

To execute the code on a supercomputing facility, all the code files need to be placed in a directory on a node with MATLAB^(R) 2019a or higher installed. Also, the input files and data need to be placed in that directory, as well as the scattering parameters. Then, a script to launch the code needs to be launched, with an example given by the bash script below (however, the specific HPC cluster will have its own language and specifics):

```
#!/bin/bash

#SBATCH --ntasks=1

#SBATCH --cpus-per-task=24

#SBATCH --mem-per-cpu=2400mb

#SBATCH --mem=90000mb

#SBATCH --time=50:00:00

module load matlab/2019a

matlab -nosplash <ELECTRA_run_Linux.m> run_example.log
```

Please, consider that different HPC clusters might have different requirements to set a parallel calculation with MATLAB®, for instance some clusters require to ask one cpu more than the number of cpu defined in MATLAB®. These are infrastructure-specific, any user can ask to the authors in the case it is necessary to modify ad-hoc the parallelization setting.

Note on the memory managing. *ElecTra* is parallelized in the energy; when the relaxation times are saved, each processor of the parallel returns the scattering rates of all the \mathbf{k} -states at that energy. For a bandstructure containing $\sim 200,000$ k-points and a few bands, if the “NN scan”

option is selected, 2 Gb per CPU and a total of ~ 60 to 90 Gb are largely sufficient. In the example above 24 CPUs are requested with 2.4 Gb each, but the total requested memory is 90 Gb, more than the sum of the memory in the CPUs, which is 57.6 Gb. More memory is required when the relaxation times are saved because the master CPU of the parallel calculation will collect a large amount of data. A good practice is to leave at least 20 Gb not occupied in the parallel execution, i.e. the total requested memory in the node should be at least 20 Gb larger than the sum of the memory used in the individual CPUs (32.4 Gb in the example above). If the “Tetrahedron” method is chosen in 3D band-structures, much more memory can be required, even 5 Gb per CPU and > 100 Gb in the node.

Differently, when the relaxation times are not saved, each processor of the parallel returns directly the TDF and the other energy dependent quantities, say that equations (2), (6) and (7) are computed inside the parallel loop onto each processor and the \mathbf{k} -state dependent scattering times are not saved. This allows a much easy managing of the memory because the master processor does not collect tens of Gb anymore, everything is generally of the order of 500 Mb. Besides, the impossibility to access to the state dependent scattering rates makes difficult possible debugging.

b. via the dedicated GUI

The user can run the code directly using the run application, named ‘Run_ELECTRA_App_W’ or ‘Run_ELECTRA_App_L’ for Windows or Linux systems. This can be used to run *ElecTra* on own’s pc while a GUI operation on a HPC infrastructure via a remote desktop connection is under development. This app is used exactly as the app to prepare the input file, but the final button, rather prepare an input .mat file, run the code. If the user runs *ElecTra* in this way, he/she does not have to prepare the input file. The

‘Run_ELECTRA_App_W’ and ‘Run_ELECTRA_App_L’ apps can be downloaded from <https://github.com/PatrizioGraziosi/ELECTRA> . By double-clicking on them, an installation procedure will start. After this, the app will be run from the MATLAB® Apps tab:

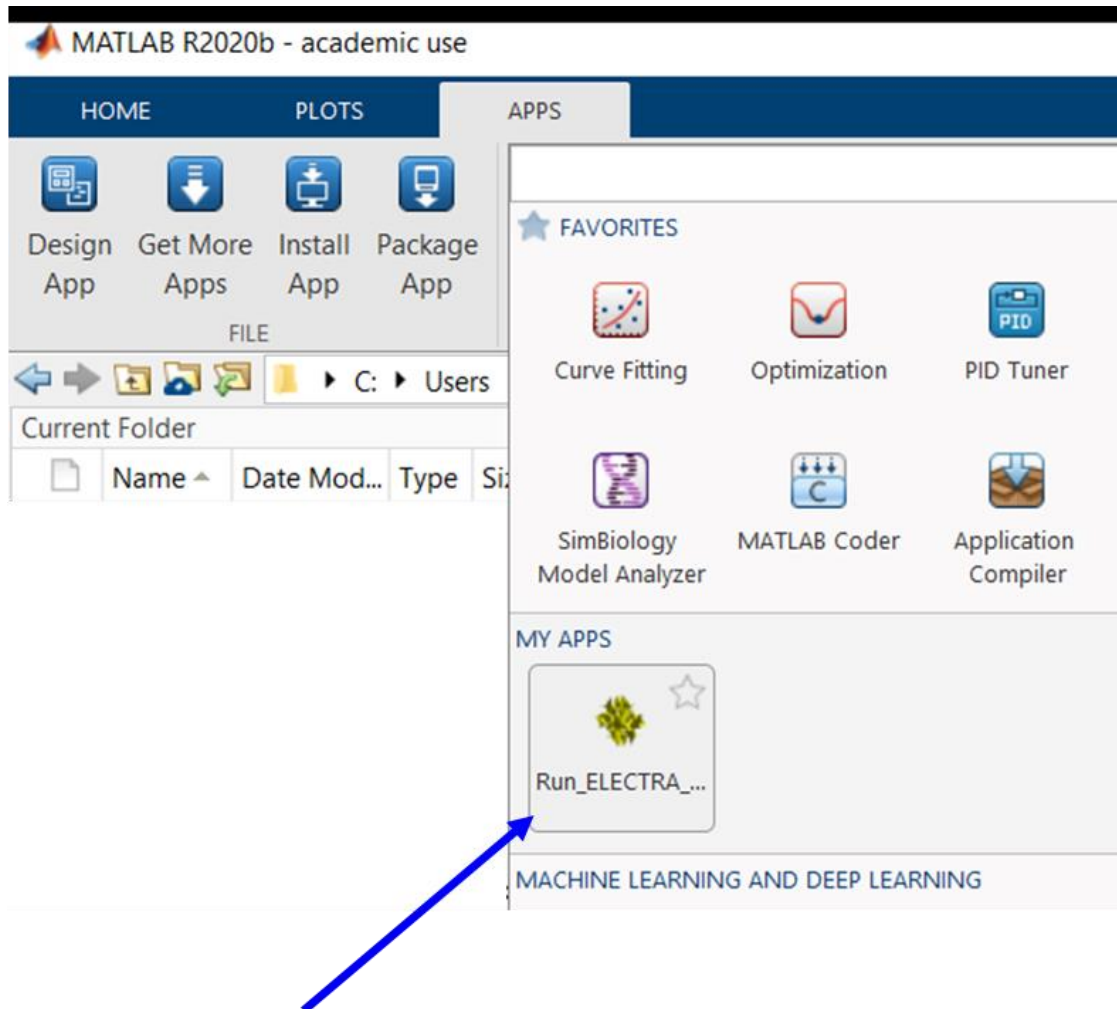


Figure 17: screenshot of the ‘APPS’ bar in the MATLAB® window with the *ElecTra* app installed.

Everything works as explained in **Section 5**, except that the extraction from the bxsf file must be done before running *ElecTra*, and that after having filled the fields, the final button directly launches the code.

First, the user must extract the data from the ‘.bxsf’ file using the first tab and the ‘Interface’ button.

The screenshot shows a software window titled "UI Figure" with three tabs: "DFT interface", "Basic", and "Advanced". The "Basic" tab is active. The main title of the interface is "input bandstructure file format interface". Under the heading ".bxsf format", there are several input fields: "Material Name" (text box), ".bxsf file name" (text box), "alat [nm]" (text box with value 0), "dimensions" (text box with value 0), "SOC in non-magnetic system" (radio buttons for "no" and "yes", with "no" selected), and "interpolation factor" (text box with value 1). An "Interface" button is located at the bottom right of the window.

Figure 18: tab to grab the data from bxsf files and to set them in the matrix format needed by *ElecTra*.

Then, the user run *ElecTra*, in the two modes detailed above. In the figures below, the fields are filled as to perform a bipolar run on TiNiSn, as in reference [2]. Thus, for the basic configuration an example is in **Figure 19** and for the Advanced case in **Figure 20**.

UI Figure

DFT interface Basic Advanced

ELECTRA run app - Basic

1) General

Material Name Dimensionality

2) Scattering mechanisms

constant time [fs]

scattering vector cutoff

3) Transport specifics

Temperatures Fermi levels Energy step

new band gap

4) Parallelization

number of parallel cpu number of nodes cluster

account queue wall time

Figure 19: GUI to directly run *ElecTra*, with the fields edited to compute a bipolar calculation for TiNiSn. [2]

ELECTRA run app - Advanced

1) General

Material Name: Dimensionality:

2) Scattering mechanisms

scattering vector cutoff:

3) Transport specifics

Temperatures: Fermi levels: Energy step:

new band gap:

4) Parallelization

number of parallel cpu: number of nodes: cluster:

account: queue: wall time:

Figure 20: example of the GUI in advanced mode filled as to run a bipolar calculation on TiNiSn; in respect to the case of **figure 19**, now the IIS is considered also as inter-band, and the bandgap is artificially changed to 0.5 eV.

c. running many materials in series

It is possible to launch many materials, one after the other, by launching the `ELECTRA_multiple_run_Windows` or `ELECTRA_multiple_run_Linux` functions. In this case the run function needs an input file with the list the materials, so it is launched as `ELECTRA_multiple_run_Linux('materials_launch')`, where

`materials_launch` is a file with the list of the materials and of the alat, in the case the bandstructures are provided in bxsf format:

```
materials_list = { 'Mg3Sb2', 'TiNiSn' } ;  
alat = [0.46,0.42] ; % in nm, only needed for .bxsf type files
```

In the example above for the HPC, the final line should be changed as

```
matlab -nosplash <multiple_run.m> run_example.log
```

Where the `multiple_run` file is a text file with .m extension which contains the bare line

`ELECTRA_multiple_run_Linux('materials_launch')`. An example is provided in the *ElecTra* folder.

9. Validation

The code has been validated versus parabolic and non-parabolic bands, where an analytical solution is known, and for some well-known semiconductors for which the transport coefficients are well assessed. A validation for the density-of-states for both parabolic band and complex bandstructure materials can be found in reference [1]. **Figure 21** reports the TDF Ξ , equation (2), and the electrical conductivity σ , equation (1a), for an anisotropic parabolic band with $m_x = 1$, $m_y = 0.5$ and $m_z = 0.1$, in units of electron rest mass, and considering ADP scattering with deformation potential of 10 eV. The xx tensorial component as well as the average of the three diagonal components is reported. It can be observed that the matching is nearly perfect. Here, the analytical results are obtained following the approach detailed in reference [6].

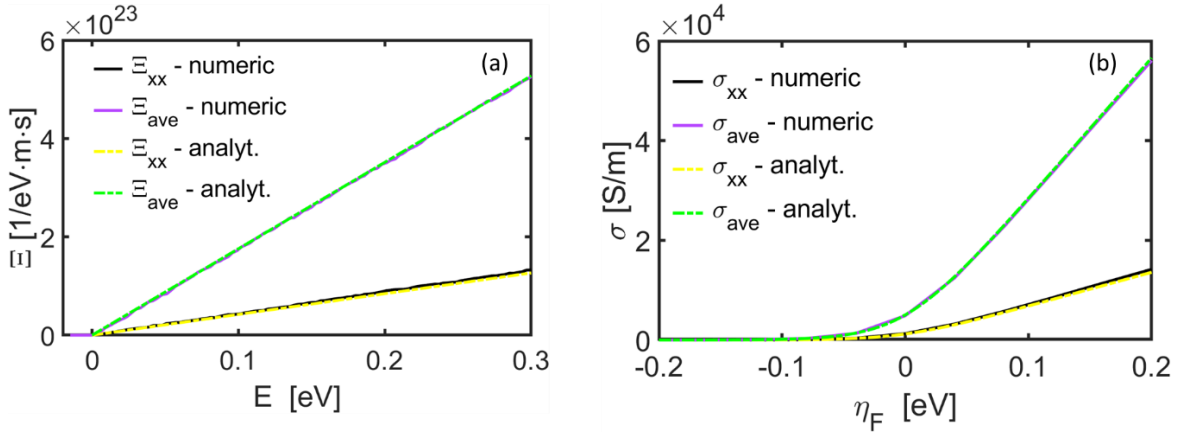


Figure 21: (a) TDF Ξ and (b) electrical conductivity σ , for an anisotropic parabolic band with $m_x = 1$, $m_y = 0.5$ and $m_z = 0.1$, in units of electron rest mass, and considering ADP scattering with deformation potential of 10 eV. The quantities labelled with “ xx ” refer to that tensorial component, the quantities labelled “ave” are the average of the three diagonal tensor components, the quantities identified as “numeric” are computed within the *ElecTra* scheme while the ones identified as “analyt.” are computed using analytical equations known to be exact for parabolic bands.

The capability to describe the charge transport has been validated under several aspects. The CRT functionality of *ElecTra* has been compared with the results obtained with BoltzTraP, [20] a CRT code widely used in the thermoelectric community, for the complex bandstructure material TiCoSb. The electrical conductivity σ and the power factor PF are reported in **Figure 22a** and **22b**, respectively. Here the blue solid lines are for the BoltzTraP results, the lines with circles are for the *ElecTra* results, dark yellow for the conductivity and dark green for the PF. The agreement is quite satisfactory with a maximum difference in the PF of below 10 %. This difference is likely due to a different numeric, for instance BoltzTraP performs volume integrals with a smeared gaussian to represent the delta function whereas *ElecTra* performs surface integrals over the constant energy surfaces defined by the delta function.[1]

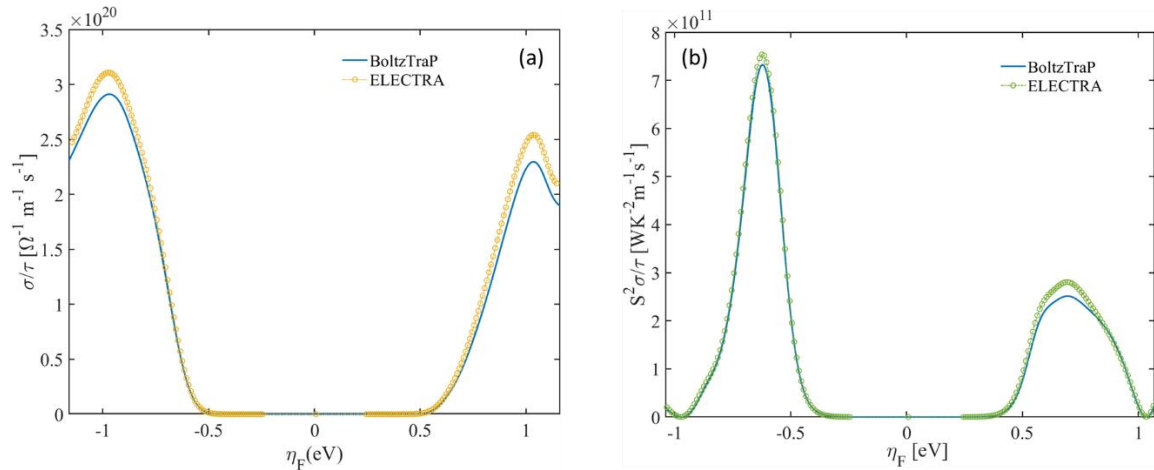


Figure 22: validation of the *ElecTra* code (yellow and green circles) versus BoltzTraP (blue lines) for TiCoSb; (a) electrical conductivity and (b) PF in units of τ .

To validate the treatment of the scattering rate, we apply the *ElecTra* scheme to the conduction bands of the following material systems: Silicon, GaAs, Ge, SiGe. The bandstructures have been built numerically by following the parameters which can be found in literature, [4, 21, 22] and compared with experimental results from the literature. [21-25] The scattering treatment is basically the one described in [3, 4] with the addition that in Ge the IIS

has been considered to be inter-valley as well as. For the SiGe alloys, the definition of the scattering deformation potentials has been based on [10]. The results are shown in **Figure 23**, the satisfactory agreement corroborate the validity of the implemented scattering scheme. The discrepancies at the higher doping can be ascribed to the electron-electron scattering, [4] which is not included here. In addition, **Figure 23c** shows the effect of the spacing in the mesh, in units of π/a , being a the lattice parameter, where a spacing of $0.01 \pi/a$ corresponds to a $200 \times 200 \times 200$ mesh in the employed bandstructures. A spacing of $0.02 \pi/a$ can be sufficient to capture general trends in the transport properties, so it looks enough when screening, or comparing, different materials. However, in order to be more accurate in specific situations, a spacing at least $\sim 0.015 \pi/a$ is recommended. This is the spacing used in **Figures 23a, 23b, and 23d**. For the case of SiGe alloys, **Figure 23d**, a low doping $\sim 10^{14} \text{ cm}^{-3}$ has been considered.

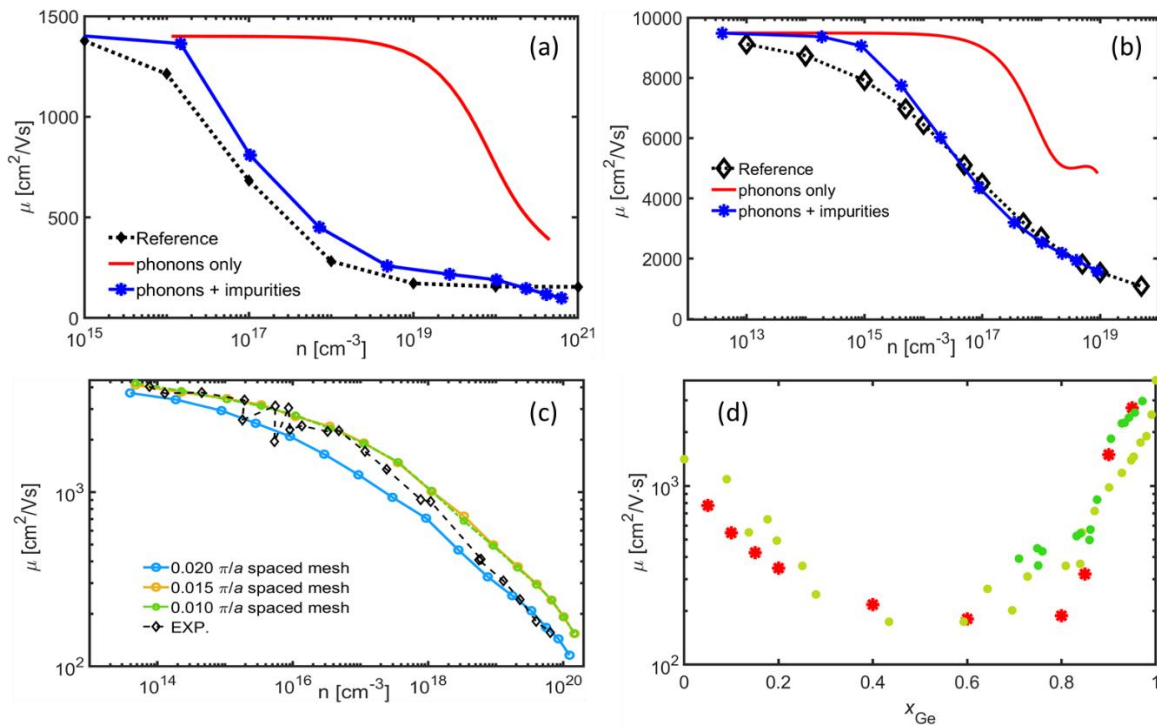


Figure 23: validation of the scattering scheme implemented in *ElecTra* in respect of known semiconductor; electron mobility for (a) Silicon, (b) GaAs, (c) Ge, (d) SiGe alloys. In (a) and (b) the red lines refer to the phonon-limited mobility, the blue lines to the phonon + IIS

mobilities and the black diamonds are the experimental reference values. In (c) the colour circle lines are for the phonon + IIS mobility for k -mesh of different spacing; the black diamonds are the experimental values. In (d) the lattice mobility versus composition for different compositions is reported, the circles are the reference experimental values while the calculated points are reported in red asterisks.

10. Example of using bandstructure from a DFT code

In this section we aim at providing guidance on how prepare the DFT bandstructure output data for the *ElecTra* code. The following example is for the case where the bands are computed with the Quantum Espresso code. [26]

Because of a relatively dense k -mesh is needed, we suggest to perform a self-consistent field (scf) calculation followed by a non-self-consistent field (nscf) calculation on a denser mesh. Below we provide examples of the input files.

I) scf calculation – input

```
&control
  prefix='Material',
  pseudo_dir = './',
  outdir='./'
  wf_collect=.true.
  etot_conv_thr = 1.0d-9,
/

&system
 ibrav= 2,
  celldm(1) = 8.50565729,
  nat= 3,
  ntyp= 2,
  ecutwfc = 300.0,
/

&electrons
  conv_thr = 1.0d-9,
  mixing_beta = 0.7,
/

ATOMIC_SPECIES
at1  24.305  at1.upf
at2  28.085  at2.upf
ATOMIC_POSITIONS crystal
at1 0.25    0.25    0.25
at2 0.75    0.75    0.75
```

```

at1  0      0      0
K_POINTS automatic
    21 21 21 0 0 0

```

launch command

```
mpirun pw.x -npool 24 -input scf.in > Material.out
```

II) nscf calculation – input

```

&control
    prefix='Material',
    pseudo_dir = './',
    outdir='./'
    wf_collect=.true.
    etot_conv_thr = 1.0d-8,
    calculation = 'nscf',
/
&system
   ibrav= 2,
    celldm(1) = 8.50565729,
    nat= 3,
    ntyp= 2,
    ecutwfc = 60.0,
    occupations = 'tetrahedra',
/
&electrons
    conv_thr = 1.0d-8
    mixing_beta = 0.7
/
ATOMIC_SPECIES
at1  24.305  at1.upf
at2  28.085  at2.upf
ATOMIC_POSITIONS crystal
at1 0.25    0.25    0.25

```

```

at2 0.75    0.75    0.75
at1 0        0        0
K_POINTS automatic
    101 101 101 0 0 0

```

launch command

```
mpirun pw.x -npool 24 -input nscf.in > Material_nscf.out
```

III) bxsf file composition – input

```

&fermi
    outdir = './',
    prefix = 'Material',
    deltaE = 3
/

```

launch command (fs.x is launched as serial in the example below)

```
fs.x -input fs.in > Material_fermi.out
```

After the 3rd step, a `Material_fs.bxsf` file is created. This file, together with the `alat` value in the `.out` file from the quantum espresso calculation in I), can be used by *ElecTra* as input bandstructure.

Following this approach, the user can use every DFT code capable of writing a `.bxsf` file.

Acknowledgement

The *ElecTra* code has been developed under the Marie Skłodowska-Curie Actions under the Grant agreement ID: 788465 (GENESIS - Generic semiclassical transport simulator for new generation thermoelectric materials). The authors acknowledge support, ideas, and patience, from Zhen Li, Chathurangi Kumarasinghe, Laura de Sousa Oliveira, Vassilios Vargiamidis, Omosede Osafire.

References

1	P. Graziosi, C. Kumarasinghe, N. Neophytou, J. Appl. Phys. 126, 155701 (2019)
2	P. Graziosi, C. Kumarasinghe, N. Neophytou, ACS Appl. Energy Mater. 3, 5913 (2020)
3	B. R. Nag, <i>Electron Transport in Compound Semiconductors</i> (Springer-Verlag Berlin Heidelberg, New York, 1980)
4	M. Lundstrom, <i>Fundamentals of Carrier Transport</i> (Cambridge University Press, 2000)
5	N. Neophytou, <i>Theory and Simulation Methods for Electronic and Phononic Transport in Thermoelectric Materials</i> (Springer) 2020
6	P. Graziosi, C. Kumarasinghe, N. Neophytou, J. Phys. Chem. C 124, 18462 (2020)
7	G. Lehmann and M. Taut, phys. Stat. Sol. (b) 54, 469 (1972)
8	P. Graziosi, Z. Li, N. Neophytou, <i>Computer Physics Communications, under review.</i>
9	N. Neophytou, H. Karamitaheri, and H. Kosina, J. Comput. Electron. 12(4), 611–622 (2013)
10	M. V. Fischetti and S. E. Laux, J. Appl. Phys. 80, 2234 (1996)
11	J. S. Pedersen, Adv. Colloid Interface Sci. 70, 171 (1997)
12	N. Neophytou and H. Kosina, Phys. Rev. B 83(24), 245305 (2011)
13	G. Samsonidze, B. Kozinsky, Adv. Energy Mater. 8, 1800246 (2018)
14	M. Combescot, R. Combescot and J. Bok, EPL 2, 31 (1986)
15	P. Graziosi, Z. Li, N. Neophytou, Appl. Phys. Lett. 120, 072102 (2022)
16	S. Wang, Z. Wang, W. Setyawan, N. Mingo, and S. Curtarolo Phys. Rev. X 1, 021012 (2011)
17	P. Yu and M. Cardona, <i>Fundamentals of Semiconductors</i> (Springer, 1999)
18	G. Ottaviani, L. Reggiani, C. Canali, F. Nava, and A. Alberigi-Quaranta Phys. Rev. B 12, 3318 (1975)
19	https://en.wikipedia.org/wiki/Semimetal#/media/File:Band_filling_diagram.svg
20	G. K. H. Madsen and D. J. Singh, Comput. Phys. Commun. 175(1), 67–71 (2006)
21	C. Jacoboni, C. Canali, G. Ottaviani, A. Alberigi Quaranta, Solid-State Electronics 20, 77, (1977)
22	W Fawcett and E G S Paige, J. Phys. C: Solid St. Phys., 4, 1801, 1971
23	V. I. Fistul, M. I. Iglitsyn, and E. M. Omelyanovskii, Sov. Phys. Solid State 4, 784 (1962)
24	Landoldt-Bornstein, Numerical Data and Functional Relationships in Science and Technology , New Series Group III, Vol. 17a, Springer, Berlin, 1982
25	M. Sotoodeh, A. H. Khalid, and A. A. Rezazadeh, J. Appl. Phys. 87, 2890 (2000)
26	https://www.quantum-espresso.org/

