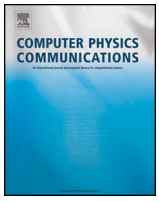
Computer Physics Communications 292 (2023) 108853

Contents lists available at ScienceDirect

Computer Physics Communications

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



Computer Programs in Physics

# CRYSTALpytools: A Python infrastructure for the CRYSTAL code ☆,☆☆

Bruno Camino [a,*], Huanyu Zhou [b], Eleonora Ascrizzi [c], Alberto Boccuni [c], Filippo Bodo [c,d], Alessandro Cossard [c], Davide Mitoli [c], Anna Maria Ferrari [c], Alessandro Erba [c], Nicholas M. Harrison [b,*]

[a] Department of Chemistry, University College London, London WC1E 6BT, United Kingdom
[b] Department of Chemistry and Institute for Molecular Science and Engineering, Imperial College London, White City Campus, 80 Wood Lane, London, W12 0BZ, United Kingdom
[c] Dipartimento di Chimica, Università di Torino, I-10125, Torino, Italy
[d] Department of Chemistry, Southern Methodist University, Dallas, TX, USA

ARTICLE INFO

*Article history:*
Received 14 February 2023
Received in revised form 28 June 2023
Accepted 6 July 2023
Available online 24 July 2023

*Keywords:*
Crystal
Python
Jupyter Notebooks
Pymatgen
ASE
Computational materials science workflows

ABSTRACT

CRYSTALpytools is an open source Python project available on GitHub that implements a user-friendly interface to the CRYSTAL code for quantum-mechanical condensed matter simulations. CRYSTALpytools provides functionalities to: i) write and read CRYSTAL input and output files for a range of calculations (single-point, electronic structure, geometry optimization, harmonic and quasi-harmonic lattice dynamics, elastic tensor evaluation, topological analysis of the electron density, electron transport, and others); ii) extract relevant information; iii) create workflows; iv) post-process computed quantities, and v) plot results in a variety of styles for rapid and precise visual analysis. Furthermore, CRYSTALpytools allows the user to translate CRYSTAL objects (the central data structure of the project) to and from the Structure and Atoms objects of the pymatgen and ASE libraries, respectively. These tools can be used to create, manipulate and visualise complicated structures and write them efficiently to CRYSTAL input files. Jupyter Notebooks have also been developed for the less Python savvy users to guide them in the use of CRYSTALpytools through a user-friendly graphical interface with predefined workflows to complete different specific tasks.

**Program summary**

*Program Title:* CRYSTALpytools
*CPC Library link to program files:* https://doi.org/10.17632/p2bp3fsk86.1
*Developer's repository link:* https://github.com/crystal-code-tools/CRYSTALpytools
*Licensing provisions:* MIT
*Programming language:* Python and Jupyter Notebook
*Nature of problem:* The CRYSTAL code [1,2] is a powerful tool for the calculation of materials properties. It stands out in the condensed matter computational landscape because of the use of local basis sets, heavy parallelisation, efficient implementation of non-local Fock exchange, and extensive use of point and space symmetry. However, it currently lacks an easily programmable interface to access its input/output structure needed to be able to use CRYSTAL calculations within computational materials science workflows. Historically, the use of CRYSTAL in such workflows has been achieved through bespoke scripting implemented in a variety of languages which has hindered code reuse and sharing.
*Solution method:* The CRYSTALpytools project will enable the automation of CRYSTAL calculations in a modular code that can be co-developed by a wide community of users worldwide. It achieves this by transforming standardised input and output files into python objects and providing a suite of functionality to manipulate them. The core implementation is based on a set of data structures denoted the *Crystal_objects*. CRYSTALpytools, in its current implementation, contains a large variety of functions for input/output manipulation, vibrational and thermodynamic analysis, and visualisation.

☆ The review of this paper was arranged by Prof. Blum Volker.
☆☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding authors.
*E-mail addresses:* b.camino@ucl.ac.uk (B. Camino), nicholas.harrison@imperial.ac.uk (N.M. Harrison).

https://doi.org/10.1016/j.cpc.2023.108853

*Additional comments including restrictions and unusual features:* The exploitation of point group and space group symmetry is one of the strengths of the CRYSTAL code, therefore, all the geometry modification functions have been developed to optimise the use of symmetry. When a structure is downloaded from a database or is modified by external code, such as pymatgen [3] or ASE [4], a symmetry analysis is performed before transforming the structure to be an optimised *Crystal_object*. All such objects can be directly transformed to standard structures periodic in 0, 1, 2 or 3 dimensions used by CRYSTAL (ie: .gui or .f34 files).

**References**

[1] Dovesi Roberto, et al., Quantum-mechanical condensed matter simulations with CRYSTAL, WIREs Comput. Mol. Sci. 8(4) (1 July 2018) e1360.
[2] Erba Alessandro, et al., CRYSTAL23: A Program for Computational Solid State Physics and Chemistry, J. Chem. Theory Comput., 2022.
[3] Shyue Ping Ong, et al., Python Materials Genomics (pymatgen): A Robust, Open-Source Python Library for Materials Analysis, Computational Materials Science 68 (2013) 314-319.
[4] Ask Hjorth Larsen, et al., The atomic simulation environment—a Python library for working with atoms, J. Phys.: Condens. Matter, 29 (7 June 2017) 273002.



## 1. Introduction

In the framework of condensed matter physics and chemistry many quantum-mechanical codes are now available to computational materials scientists to calculate the properties of extended systems [1–7]. Among them, the CRYSTAL code stands out because of the use of a local atom centred Gaussian rather than plane wave basis set [8–11]. This facilitates all-electron calculations, periodicity in 0, 1, 2 and 3 dimensions, the efficient calculation of Coulomb and non-local exchange interactions, an effective exploitation of symmetry, and a straightforward local chemical analysis of the electronic structure and related properties. The CRYSTAL code has been developed since the 1970s by the Theoretical Chemistry Group of the University of Torino (Italy) and, from 1974, in collaboration with the Computational Materials Science Group at Daresbury Laboratory and Imperial College London (UK). It has, throughout the years, grown to be developed by a large community of scientists and software developers from many institutions worldwide.

The CRYSTAL code allowed the computational chemistry and materials science communities to achieve groundbreaking results. These range from the early days Roothaan-Hartree-Fock atomic wavefunctions published in 1974 [12], which have been cited more than 5000 times, the first hybrid-exchange calculations performed in periodic systems [13], to the up-to-date self-consistent, as well as perturbative, treatment of spin-orbit coupling [14–16], generalization of the study of chemical bonding to Lanthanides and Actinides [17–19], structural optimization strategies based on molecular dynamics concepts [20], treatment of thermo-elasticity [21–23], description of anharmonic lattice vibrations [24–27], and many others.

Although new and exciting results are constantly obtained by using the CRYSTAL code, the line separating accurate materials simulations from data science and machine learning methods becomes thinner every year. To be able to join these new trends, the user needs to be able to access the computational electronic structure code of choice via a user-friendly and robust programming infrastructure. This is a well-known issue in the computational chemistry and materials science community and as a matter of fact, several interfaces are being developed for other codes [28–31]. These considerations have prompted the beginning of the `CRYSTALpytools` project presented here. The initial goal of the project was to develop an automated way to prepare input files, and quickly analyse output files to be able to develop customisable workflows where information could be extracted efficiently at each step to facilitate the next one. `CRYSTALpytools` is developed in Python as it is a well supported, flexible and easy to use language that has been adopted by many data science and computational materials projects [28,29,32–36] in addition to providing a wide variety of libraries for plotting and numerical analysis [37–39].

Thanks to the object-oriented logic implemented in `CRYSTALpytools`, important information contained in CRYSTAL input and output files are translated into Python objects, hereafter referred to as CRYSTAL objects (COs). Such objects are the central data structure of the project. These allow the user to prepare input files in an automated fashion, quickly analyse output files and extract, post-process and plot relevant information.

In addition, COs can be transformed into the pymatgen Structure [29] and ASE Atoms [28] objects and *vice versa*. This enables the user to access the large range of functions of these distributions to manipulate the geometry of the system within CRYSTAL workflows. For example, pymatgen and ASE can be used to efficiently build defective or adsorbed model structures, instead of having to construct them by hand, which is a cumbersome and error prone process. Conversion functions to transform pymatgen and ASE objects back into COs have also been developed. Furthermore, new functions have been created to export structural COs to commonly used formats such as .cif and .xyz that can be opened by most visualisation tools [40,41].

The `CRYSTALpytools` project also provides extra functionalities for data post-processing and plotting, for the electronic band structure, spatial dependence of mechanical properties, analysis of the electron density and related quantities, visualization of vector fields such as non-collinear magnetisation, orbital and spin current densities, electron transport properties, and thermodynamics.

## 2. Structure of the project

The `CRYSTALpytools` project is open source and shared through GitHub. The work discussed in this article can be found at the GitHub URL given in Ref. [42], which was created as a container for all current and future open source developments of tools to interact with the CRYSTAL code. Currently, the page contains two main repositories:

- `CRYSTALpytools`: this is a set of Python functions that provide the core functionality for the interaction with the input/output structure of the CRYSTAL code, plotting functionalities, and much more. The presentation of `CRYSTALpytools` is the main goal of this paper;
- `CRYSTALnotebooks`: this repository contains an ever growing set of Python Jupyter Notebooks each designed for a spe-

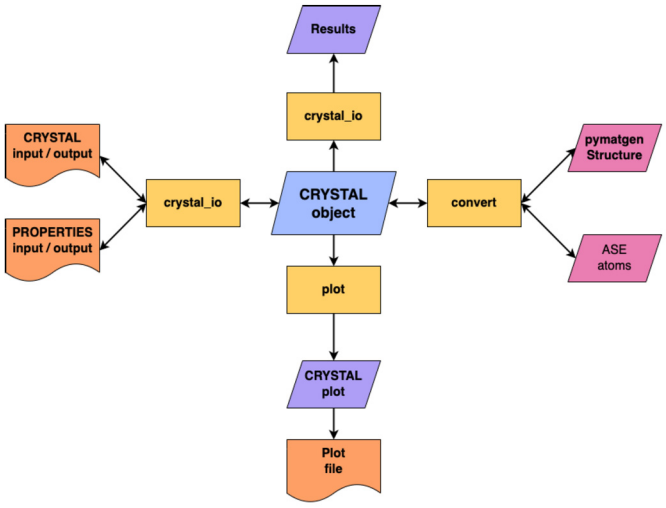



**Fig. 1.** Representation of the flow of data in different modules interacting in CRYSTALpytools. Colour legend: orange=file, yellow=module containing specific functions (*e.g.* the functions labelled *crystal_io* can be found in the file *crystal_io.py* file), light blue = python object (CRYSTAL output), purple= output extracted from a CRYSTAL file, but not yet saved to file (*e.g.* a band structure displayed using Matplotlib), pink = python object (pymatgen Structure or ASE Atoms). (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

cific task, where the functions from CRYSTALpytools are used. Examples of these tasks are plotting electronic band structures, contour maps, computing and plotting mechanical properties, and many others. These Notebooks are developed and made available for the less Python experienced users, as they pre-combine functions in the right sequence, to achieve a given task accurately and efficiently. As a matter of fact, they act as an extra layer between the user and the CRYSTALpytools code. For instance, through a graphical interface, the user can select the files needed for the task while task-specific cells handle the calls to the corresponding CRYSTALpytools functions.

As mentioned above, the central data structure of the project is called CRYSTAL object. A CO can either be generated by reading CRYSTAL input/output files, or by conversion from a pymatgen or ASE object. COs play a central role in the CRYSTALpytools set of functions, where the underlying strategy is that the communication among functions contained in different modules (or even among functions belonging to the same module) passes through the use of COs. This is represented graphically in Fig. 1, where examples of data flows between different modules in CRYSTALpytools and input/output is depicted.

## 3. Reading, writing, and converting

The functions to read/write from/to CRYSTAL files are implemented in the *crystal_io* module of CRYSTALpytools. Their main goal is transforming CRYSTAL input/output files into COs and *vice versa*. They are, therefore, most likely to be used at the beginning and/or at the end of a workflow. The CRYSTAL program suite also contains the PROPERTIES code originally designed to perform post-SCF analysis of the converged wave function of the system. Both CRYSTAL and PROPERTIES have their own input/output files, which is why the *crystal_io* module of CRYSTALpytools has been organized into four main classes: *Crystal_input*, *Crystal_output*, *Properties_input*, *Properties_output*. The type of attributes, methods and functions of each class varies largely depending on the information contained in the file being read or written. For instance, a CO obtained from the *Crystal_output* class can be operated on with a set of methods to extract information found in the CRYSTAL output file such as the final energy, band gap, optimised geometry, etc. An example of how such a CO can be created and used is shown below:

```
1 from CRYSTALpytools import Crystal_output
2 Cout = Crystal_output().read_crystal_output('file')
3
4 ene = Cout.get_final_energy()
5 gap = Cout.get_band_gap()
6
7 cell, atnums, coord = Cout.get_last_geom()
```

In the script above, the first line creates a CO (namely `Cout`) by use of the *read_crystal_output()* method of the *Crystal_output* class, which reads a CRYSTAL output file (`file` in the script above). Lines 3 and 4 extract information from the CO, namely, the final energy and the band gap by acting on it with the two methods *get_final_energy()* and *get_band_gap()*. In this case, the outcome of these operations is stored into variables `ene` and `gap`. In line 6, the *get_last_geom()* method is used to act on the CO, which extracts the final optimized geometry with information on the lattice vectors stored into the `cell` variable, on the atomic numbers of the atoms in the reference cell into the `atnums` variable, and on the Cartesian coordinates of the atoms of the reference cell into the `coord` variable.

The COs from the *Crystal_input* class are characterized by four attributes, corresponding to the four main blocks of a CRYSTAL input file (*i.e.* geometry, basis set, DFT functional, and SCF parameters). These blocks can be generated either by reading an existing CRYSTAL input file (`file1` in the script below), whose path and name are specified as an argument to the *from_file()* method:

```
1 from CRYSTALpytools import Crystal_input
2 Cin = Crystal_input().from_file('file1')
```

or by building it block by block explicitly uing the *from_blocks()* method, which requires the four blocks to passed as its arguments. The same logic applies to COs generated by the *Properties_input* class. Both the *Crystal_input* and *Properties_input* classes also contain writing functionalities. For instance, the CO `Cin` in the script above can be written to a specific file (`file2` in the script below) using the *write_crystal_input()* method as follows:

```
1 Cin.write_crystal_input('file2')
```

The functions contained in the *convert* module were developed in order to convert COs into different data structures characteristic of other Python infrastructures (e.g. pymatgen [29] and ASE [28]), or standard formats, such as .xyz and .cif. The opposite conversion (from other python distributions to COs) is also possible. This allows the user to move back and forth between COs and other data structures in order to take advantage of existing functionalities of other Python projects. For example, a CRYSTAL user might be interested in optimising the geometry of a structure, convert the optimised geometry into a pymatgen Structure object to then use pymatgen functions to modify the volume of the cell, replace or remove atoms via a Python loop, and write the modified geometry back to a CRYSTAL input file (or multiple files if several structures are generated in the process). In the code below, a CRYSTAL output object is converted into a pymatgen Structure object using the *cry_out2pmg()* function:

```
1 from CRYSTALpytools import Crystal_output
2 Cout = Crystal_output().read_crystal_output('file')
3 pmg = cry_out2pmg(Cout,initial=False)
```

The option `initial=False` in the script above specifies that the last geometry found in the CRYSTAL output file should be used to generate the pymatgen Structure object. This refers to both the lattice parameters and atomic positions. Intuitively, `initial=True` would generate a pymatgen Structure object from the initial geometry found in the CRYSTAL output file. In addition, this module

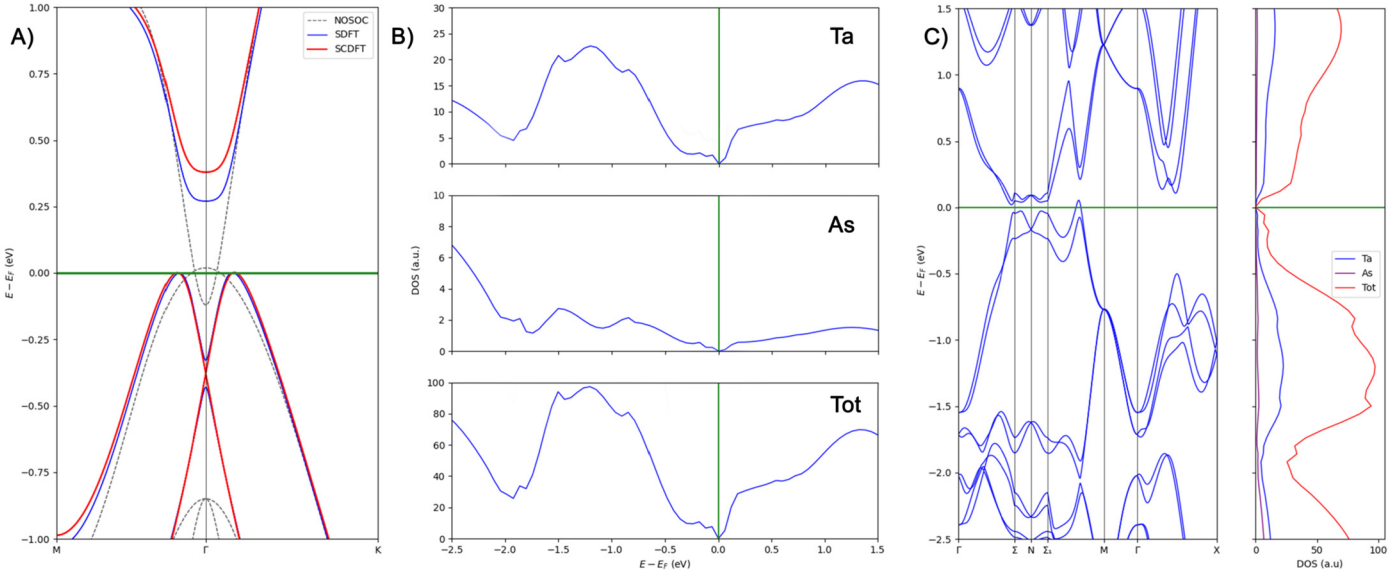



**Fig. 2.** A) Electronic band structure of a Bi (001) bi-layer computed without spin-orbit coupling (dashed grey line), and with spin-orbit coupling as treated within the spin DFT (blue line), and spin current DFT (red line). B) Density-of-States (DOS) of a Weyl semi-metal: the tetragonal phase of TaAs, space group $I4_1md$; total DOS in the bottom panel and contributions from the atomic orbitals of Ta and As atoms in the upper panels. C) A Combined plot of the electronic band structure and DOS of TaAs.

allows the experienced pymatgen or ASE users to save their structures to files in CRYSTAL format.

The functionalities of `CRYSTALpytools` discussed in this section have been used in a recent work: the study of $Li_2TiS_3$ solid solutions for battery applications [43]. A workflow was generated to read the 4023 symmetry irreducible nuclear configurations generated by CRYSTAL and analyse their local atomic environment (in terms of the Ti-Ti distance) in order to perform a cluster analysis and extract a subset of structures as representative of the whole set.

## 4. Data post-processing and plotting

The `CRYSTALpytools` library provides several functionalities to post-process and plot data obtained from different types of calculations performed with CRYSTAL as well as with its sub-module PROPERTIES. Depending on the specific calculation, different output files are produced by CRYSTAL and PROPERTIES. These can be read via the *crystal_io* module of `CRYSTALpytools` and transformed into COs. In this Section, the main functionalities of the plotting functions are discussed.

### 4.1. Electronic band structure and density-of-states

The visual inspection of the electronic band structure of a material constitutes one of the first preliminary steps of any quantum-mechanical investigation in a condensed matter context. Flexible tools to plot the electronic band structure, density-of-states, and combination of the two were implemented in the *plot* module of `CRYSTALpytools`. These properties are computed by the PROPERTIES module of CRYSTAL and data is stored in files named BAND.DAT and DOSS.DAT. Such files can be read by the *read_cry_band()* and *read_cry_doss()* methods, respectively (as available through the *Properties_output* class in the *crystal_io* module of `CRYSTALpytools`) to generate the corresponding COs. The different types of plots that can be generated are discussed below.

- Electronic Band Structure: A plot of the electronic band structure can be obtained from the *plot_cry_band()* function of the *plot* module of `CRYSTALpytools`. Customised plots can be made through a selection of arguments available in this function. For example, multiple band structures can be superimposed in the same plot. Fig. 2 A) is an example where the electronic band structure of a strained Bi (001) bi-layer is shown at the onset of an insulator to topological insulator phase transition, as computed without and with inclusion of spin-orbit coupling (SOC) [44–47,14,48] as implemented in CRYSTAL23 [11].
  As long as the path on the x axis is consistent, the *plot_cry_band()* function is also able to handle multiple band structures corresponding to different geometries. For instance, one could superimpose the electronic structure of a material as computed at different pressures on the same plot. The plotting function would automatically scale the x coordinate to make them consistent across the different geometries. Technical details on how to customise such plots are reported on the project GitHub page.
- Density-of-States: Plots of the electronic DOS can be generated by using the *plot_cry_doss()* function on the corresponding CO previously generated from the reading of the DOSS.DAT file. As for the electronic structure, plots can be customised through a selection of arguments available in this function. For instance, if the total DOS is computed along with a set of projections on specific atomic orbitals, one can choose to have all data plotted in a single panel or as many panels as there are projections. Fig. 2 B) is an example of the latter case for the DOS of the TaAs Weyl semi-metal [49] where three panels are combined with the total DOS (bottom) and contributions from the atomic orbitals of Ta and As atoms in the upper panels. Technical details on how to customise such plots can be found on the project GitHub page.
- Combined Plot: The electronic band structure and DOS can be combined in a single figure using the *plot_cry_es()* function once the two corresponding COs have been obtained by reading the BAND.DAT and DOSS.DAT files. An example of this plot is presented in Fig. 2 C).

### 4.2. Elastic properties

The elastic response of any material is fully described by its fourth-rank elastic stiffness tensor **C**, whose elements $C_{ijkl}$ (with

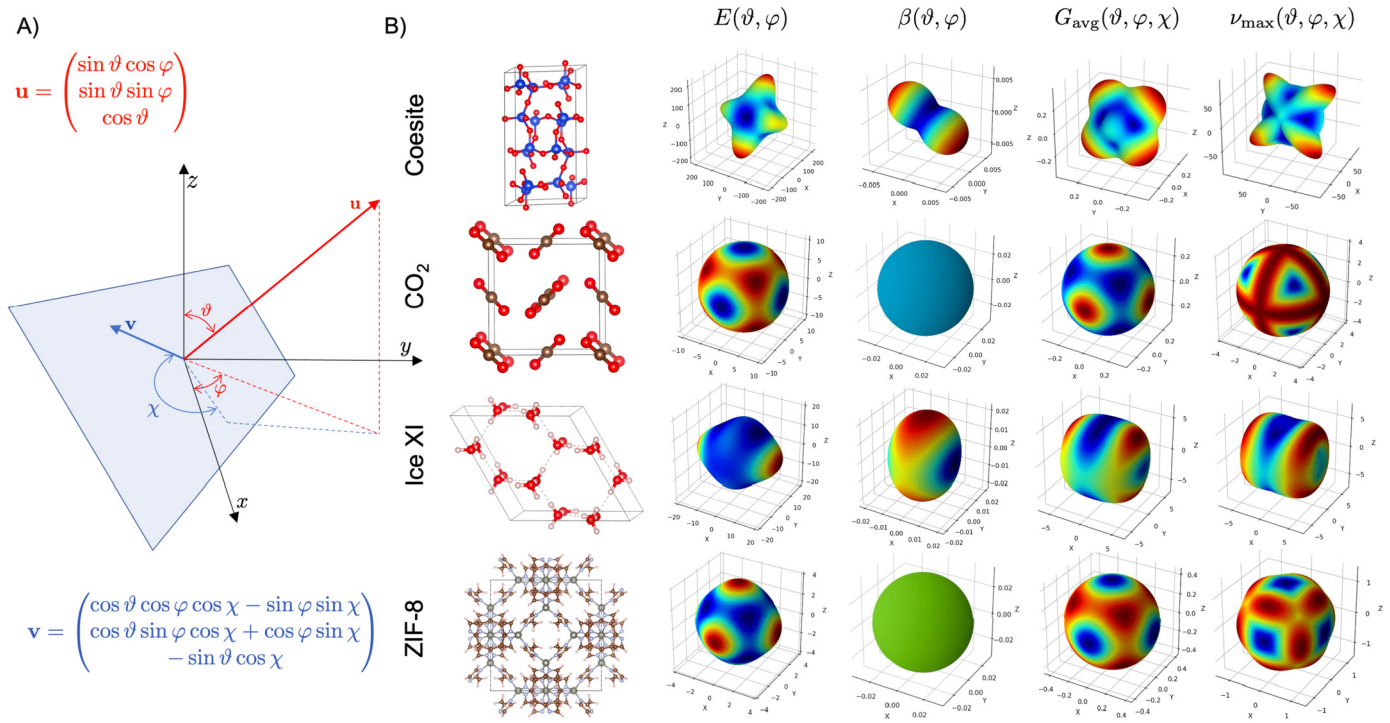



**Fig. 3.** A) Graphical representation of the geometrical parameters needed for the definition of the four mechanical properties: the two directions **u** and **v** and the three angles $\vartheta$, $\varphi$, and $\chi$. B) Young modules $E$, linear compressibility $\beta$, average shear modulus $G_{avg}$ and maximum Poisson's ratio $\nu_{max}$ (all in GPa) for four systems: coesite $SiO_2$, $CO_2$ molecular crystal, water ice XI, and ZIF-8.

$i, j, k, l = x, y, z$) are the so-called elastic constants defined as second energy derivatives with respect to a pair of lattice strains [50]. The CRYSTAL program contains built-in functionalities for the automated numerical evaluation of such derivatives and for the one-shot calculation of the whole elastic tensor at the equilibrium volume [51–53], at a given pressure [54] or temperature [21]. Once the elastic tensor is computed, a variety of mechanical properties of the system can be evaluated. Because materials are generally anisotropic, the value of each mechanical property depends on the particular crystallographic direction along which it is measured. The spatial dependence of these properties can be computed from the elastic constants and requires advanced plotting utilities to be properly visually analysed.

`CRYSTALpytools` contains functions to read the computed elastic tensor from an elastic CRYSTAL calculation and to compute and plot several directional mechanical properties: the Young modulus $E$ (*i.e.* the strain response of a solid to an uniaxial stress, in the direction of this stress), the linear compressibility $\beta$ (*i.e.* the linear strain under the application of an external hydrostatic stress), the shear modulus $G$ (*i.e.* the strain response to a shear stress), and Poisson's ratio $\nu$ (*i.e.* the strain response in the directions orthogonal to a uniaxial stress) [55]. The first two quantities depend on a single direction **u** (and thus on two angles $\vartheta$ and $\varphi$ in spherical coordinates), whereas, the latter two depend on two directions **u** and **v** (if orthogonal, the two directions can be defined in terms of three angles $\vartheta$, $\varphi$, and $\chi$ in spherical coordinates). Fig. 3 A) reports a graphical definition of such geometrical quantities. For each direction (*i.e.* for each pair of $\vartheta$ and $\varphi$ angles), the value of the Young modulus can be computed as:

$$E(\vartheta,\varphi) = \frac{1}{u_i u_j u_k u_l S_{ijkl}} , \tag{1}$$

where Einstein's notation is used according to which indices that are repeated are meant to be summed over, and where $S_{ijkl}$ are the elements of the inverse of the elastic tensor $\mathbf{S} = \mathbf{C}^{-1}$. The value of the directional linear compressibility can be computed as:

$$\beta(\vartheta,\varphi) = S_{ijkk} u_i u_j . \tag{2}$$

The directional shear modulus is computed as:

$$G(\vartheta,\varphi,\chi) = \frac{1}{u_i v_j u_k v_l S_{ijkl}} . \tag{3}$$

Its dependence on three angles makes its graphical representation difficult. Here we follow the same strategy already adopted by other authors [56,57] where for each direction (*i.e.* for each pair of $\vartheta$ and $\varphi$ angles) the shear modulus is computed for all values of the third angle $\chi$ and the maximum, minimum and average values are stored. Three distinct plots are possible where for each direction either the minimum, maximum or average value is reported. The same logic applies to Poisson's ratio:

$$\nu(\vartheta,\varphi,\chi) = -\frac{u_i u_j v_k v_l S_{ijkl}}{u_i u_j u_k u_l S_{ijkl}} . \tag{4}$$

In `CRYSTALpytools`, the *get_elatensor()* method of the *Crystal_output* class reads the computed elastic tensor from the main CRYSTAL output file and stores it into an appropriate CO. The mechanical properties above can be computed and plotted in 3D by use of the *plot_cry_ela()* function. Fig. 3 B) reports some plots produced with this function. In particular, 3D plots of the spatial dependence of the Young modules $E$, linear compressibility $\beta$, average shear modulus $G_{avg}$ and maximum Poisson's ratio $\nu_{max}$ are reported for four systems: coesite $SiO_2$, $CO_2$ molecular crystal, water ice XI, and ZIF-8. Different customised plots can be made through a selection of arguments available in the *plot_cry_ela()* function. For instance, a reference elastic tensor can be provided to define a reference colour scale to be kept constant across multiple plots. Also the angular resolution can be customised.

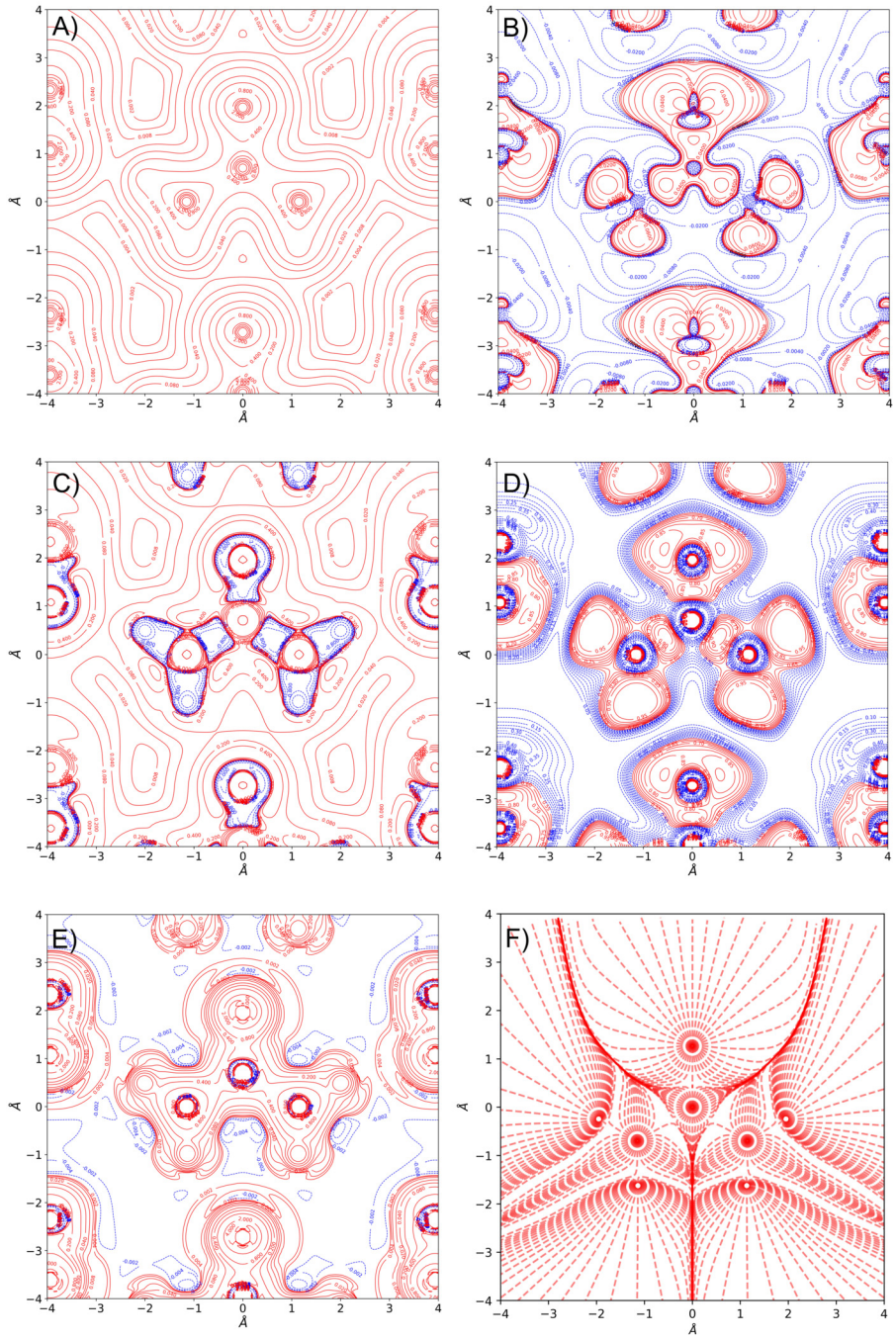



**Fig. 4.** A-E) 2D contour maps of different electronic properties of the Urea molecular crystal: the total electron density $\rho(\mathbf{r})$ (A), the deformation density $\Delta\rho(\mathbf{r})$ (B), the Laplacian of the density $\nabla^2\rho(\mathbf{r})$ (C), the electron localization function ELF($\mathbf{r}$) (D), the kinetic energy density $K(\mathbf{r})$ (E). F) 2D plot of the projected trajectories of the gradient of the electron density $\nabla\rho(\mathbf{r})$ on the molecular plane of the Urea molecule.

### *4.3. Analysis of the electron density and associated properties*

Arguably, the quantum theory of atoms in molecules (QTAIM) represents the most comprehensive formal framework for the analysis of the electron density of a quantum-mechanical system [58,59]. It is based on a topological analysis of the electron density $\rho(\mathbf{r})$, and it is often complemented with a topological analysis of the Laplacian of the electron density $L(\mathbf{r}) = -\nabla^2\rho(\mathbf{r})$ [60]. The QTAIM was implemented in the CRYSTAL package with the TOPOND module [61], which has recently been extended to parallel computing [62] and to high angular momentum basis functions [18,17]. The TOPOND module performs a full topological analysis of both $\rho(\mathbf{r})$ and $L(\mathbf{r})$ looking for their respective critical points and allows for an atomic partitioning of all related properties.

Furthermore, starting from the ground state wavefunction obtained from the self-consistent field process, the TOPOND module can compute a variety of electronic properties at a 2D grid of points in space, including the electron density itself $\rho(\mathbf{r})$, its Laplacian $L(\mathbf{r})$, local kinetic, potential, and total energy densities $K(\mathbf{r})$, $V(\mathbf{r})$ and $H(\mathbf{r})$, the electron localization function ELF($\mathbf{r}$). These are all scalar fields associating a scalar value to each point of space $\mathbf{r}$. In `CRYSTALpytools`, user-friendly functions to produce 2D contour plots of all these quantities were implemented. The *read_cry_contour()* method of the *Properties_output* class can read the output files generated by TOPOND (such as SURFRHOO.DAT for

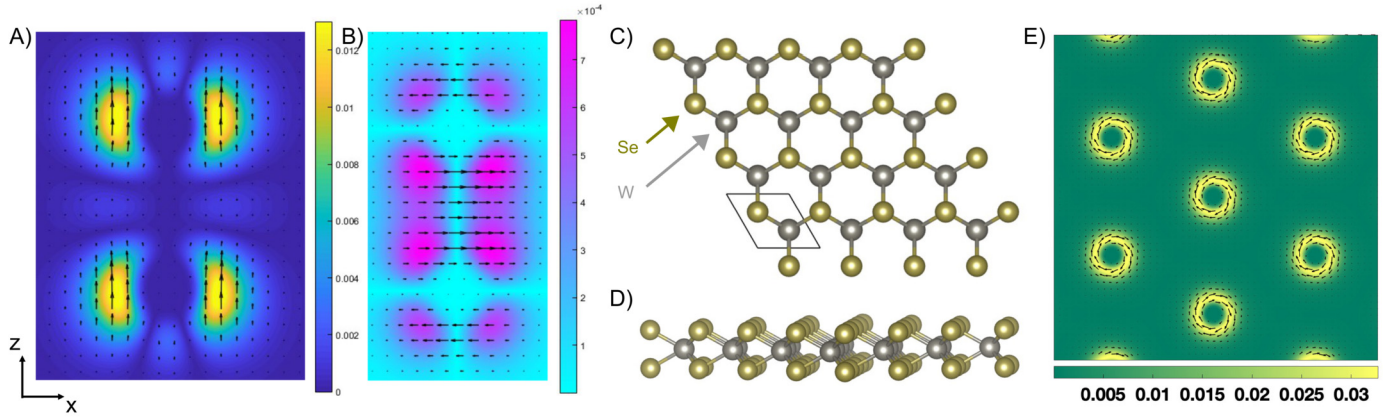



**Fig. 5.** A) 2D representation of the vector field of the magnetization **m** of the $I_2^+$ molecule (with its molecular axis along *z*) in the *xz* plane; B) 2D representation of the vector field of the orbital current density **j** of the $I_2^+$ molecule in the *xz* plane; C-D) top and side view of the $WSe_2$ 2D system; E) 2D representation of the vector field of the spin current density $\mathbf{J}^{\mathbf{z}}$ of the 2D $WSe_2$ in the horizontal plane passing through the W atoms.

$\rho(\mathbf{r})$, SURFLAPP.DAT for the Laplacian, SURFKKIN.DAT for the local kinetic energy density $K(\mathbf{r})$, *etc.*) and generate the corresponding COs. By acting on such COs with the *plot_cry_contour()* function, 2D contour plots can be produced. As an example, Fig. 4 shows 2D contour plots for some of the properties discussed in this section for the molecular crystal of Urea.

In addition, the *plot_cry_difference()* function implemented in `CRYSTALpytools` take two COs obtained from two separate calculations as input and generate difference 2D plots. This is particularly convenient when 2D plots of deformation and interaction densities $\Delta\rho(\mathbf{r})$, or deformation and interaction Laplacians $\Delta L(\mathbf{r})$ are to be analysed. Deformation properties is defined as the difference between the property of the interacting system and that given by the superposition of non-interacting atomic contributions. Interaction properties are defined as the difference between the property of the interacting system and that given by the superposition of non-interacting molecular contributions. The latter can only be defined for molecular crystals. Fig. 4 B) shows an example of deformation density $\Delta\rho(\mathbf{r})$ 2D contour plot for the molecular crystal of Urea. Lastly, the *read_cry_contour()* method and *plot_cry_contour()* function have also been generalised to be able to read and plot the TRAJGRAD.DAT output file from Topond to visualize trajectories of the gradient of the electron density $\nabla\rho(\mathbf{r})$ projected on a plane. Fig. 4 F) shows such trajectories when projected in the molecular plane of the Urea molecule.

### 4.4. Plots of vector fields: non-collinear magnetization, orbital and spin current densities

Both the Hartree-Fock and DFT methods implemented in Crystal have recently been generalised to a two-component spinor basis [46,48]. Such generalization paved the way to an effective self-consistent treatment of the spin-orbit interaction [14]. These extensions allow to compute a variety of advanced electronic properties of materials at any point **r** of space, such as the magnetization vector $\mathbf{m}(\mathbf{r})$ (collinear or non-collinear based on the adopted approach), the orbital current density $\mathbf{j}(\mathbf{r})$, and the three spin current densities $\mathbf{J}^{\mathbf{x}}(\mathbf{r})$, $\mathbf{J}^{\mathbf{y}}(\mathbf{r})$ and $\mathbf{J}^{\mathbf{z}}(\mathbf{r})$. All such properties are vector fields represented by a Cartesian vector of three components at each point **r** of space.

The Properties module of the Crystal suite has been extended to compute all these properties at 2D or 3D grid of points. When the properties are computed on user-defined 2D grids of points on selected planes, data is stored in .f25 format files. These files can be read by the *read_vecfield()* function of the *Properties_output* class. The generated CO can then be plotted. The *plot_vecfield_2D_m()*, *plot_vecfield_2D_j()* and *plot_vecfield_2D_J()* functions are able to produce 2D plots of the properties listed above, where a colour map is used to represent the module of the property at each point of the selected plane. Arrows are superimposed to the plot to show the projections of the local 3D vectors of the property in the plane. Fig. 5 shows selected examples of the obtained graphical representation.

### 4.5. Electron transport properties

A module for the calculation of electron transport properties has recently been implemented in Crystal. Such module is based on the semiclassical Boltzmann transport theory in the constant relaxation-time approximation [63]. Properties such as the Seebeck coefficient matrix **S** (with elements $S_{i,j}$ where $i, j = x, y, z$) and the electron conductivity matrix $\boldsymbol{\sigma}$ (with elements $\sigma_{ij}$ where $i, j = x, y, z$) are computed by the Properties module. Computed values are saved into the output files SEEBECK.DAT and SIGMA.DAT. These can be read by the *read_cry_seebeck()* and *read_cry_sigma()* functions of *Properties_output* class to generate the corresponding COs. These quantities can be plotted in different ways: i) as a function of the chemical potential with functions *plot_cry_seebeck_potential()* and *plot_cry_sigma_potential()* (as displayed in Fig. 6 B); ii) as a function of the charge carrier concentration with functions *plot_cry_seebeck_carrier()* and *plot_cry_sigma_carrier()* (as done in Fig. 6 A); iii) by superimposing data from multiple calculations on a single plot with functions *plot_cry_multiseebeck()* and *plot_cry_multisigma()*. The *plot_cry_powerfactor_potential()* and *plot_cry_powerfactor_carrier()* functions were developed to compute and plot the thermoelectric power factors $PF_{ij} = S_{ij}^2\sigma_{ij}$, as shown in Fig. 6 C) and D).

### 4.6. Thermodynamics

Thanks to the underlying robust implementation of analytical forces [64–68], the Crystal program can efficiently describe the lattice dynamics within the harmonic approximation (HA) [69,70], the quasi-harmonic approximation (QHA) [71–75], and, more recently, some explicit anharmonic approaches [24–27]. From computed phonons, thermodynamic properties of materials at finite temperatures and pressures can be derived via a statistical thermodynamic approach. The *Thermodynamics* module of `CRYSTALpytools` provides functionalities to derive harmonic and quasi-harmonic thermodynamic properties of materials from harmonic phonons computed with Crystal.

In the current implementation, this module comprises three classes: *Mode* to read (from the .out Crystal output file) and store

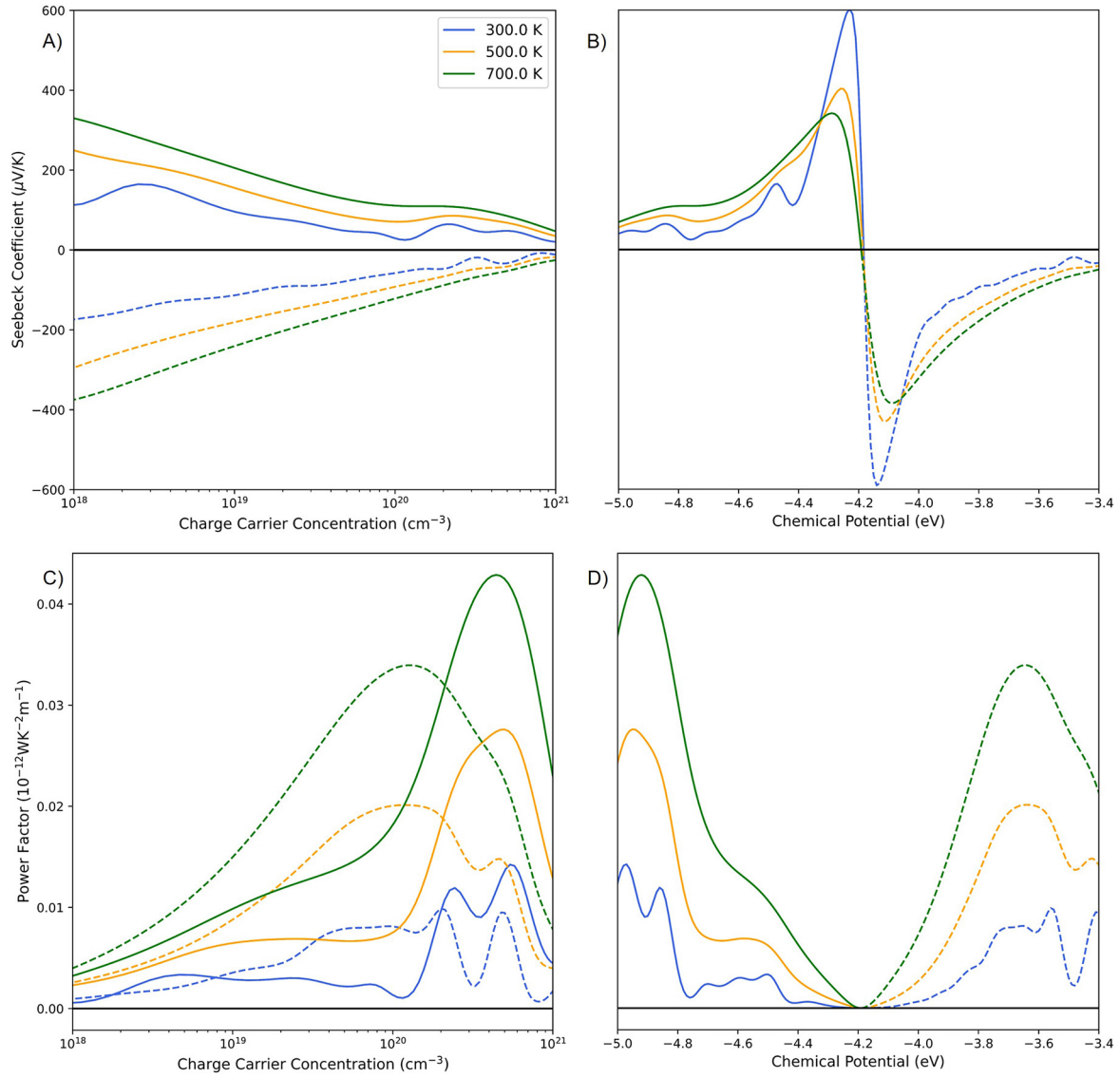



**Fig. 6.** Electron transport coefficients of $Bi_2Te_3$ for $i = x$ and $j = x$ at three different temperatures. A) Seebeck coefficient $S_{xx}$ as a function of charge carrier concentration; B) Seebeck coefficient $S_{xx}$ as a function of chemical potential; C) Power Factor $PF_{xx}$ as a function of charge carrier concentration; and D) Power Factor $PF_{xx}$ as a function of chemical potential. To differentiate transport coefficients due to n-type or p-type conduction (electrons or holes as majority carriers) dashed and solid lines are used, respectively. Data from Ref. [63].

frequencies and eigenvectors of harmonic vibrational modes, *Harmonic* to compute harmonic thermodynamic properties of a system (*i.e.* at constant volume), and *Quasi_harmonic* to compute quasi-harmonic thermodynamic properties of a system.

For the HA, relevant thermodynamic properties include zero-point energy, phonon vibration contribution to internal energy, Helmholtz free energy, Gibbs free energy, vibration entropy, and constant-volume specific heat. The following script shows how to calculate and print harmonic thermodynamic properties at different user-specified temperatures (in K):

```
Cha = Harmonic(temperature=[298,400],write_out=True,
               filename='file2').from_file('file1')
```

Harmonic phonons are read from a CRYSTAL .out file (`file1` in the example above), thermodynamic properties are computed at 298 and 400 K and results printed in a specified file (`file2` in the example above).

The class *Quasi_harmonic* is capable of more comprehensive jobs at the QHA level, such as computing thermal expansion curves and getting high-pressure high-temperature thermodynamic properties. Crucially, a QHA approach relies on harmonic phonons computed at different volumes. The *Quasi_harmonic* class can either read a single QHA CRYSTAL output file or a set of independent harmonic CRYSTAL calculations at different volumes. The latter approach may lead to numerical noise when running calculations with default settings because of overlap-based truncation thresholds for bielectronic Coulomb and exchange integrals [76,77]. More details and possible solutions are addressed in [78]. The main QHA working equation expresses the Helmholtz free energy of the system as a function of temperature and volume as:

$$F(T,V) = E(V) + \frac{1}{2}\sum_i \hbar\omega_i(V) + k_B T \sum_i \left[\ln\left(1 - e^{-\frac{\hbar\omega_i(V)}{k_B T}}\right)\right] \quad (5)$$

where $E(V)$ is the zero-temperature internal energy of the crystal (excluding the vibrational zero-point energy) and $\omega_i(V)$ are the

harmonic phonon frequencies at volume $V$. The Gibbs free energy of the system is:

$$G(T, V, p) = F(T, V) + pV \tag{6}$$

The equilibrium volume at a given temperature and pressure, $V(T, p)$, can be obtained by minimising Eq. (6) with respect to the volume $V$, keeping $T$ and $p$ fixed. Two different strategies have been devised for the calculation of quasi-harmonic thermodynamic properties, implemented in two methods: I) *thermo_freq*; and II) *thermo_eos*.

The first strategy, implemented in method *thermo_freq*, involves three steps: i) fitting the static internal energy $E(V)$ to a variety of equations of state (EoS), as implemented in pymatgen [29], with third-order Birch-Murnaghan being the default one [79]; ii) individual polynomial fitting of each harmonic frequency as a function of the volume, $\omega_i(V)$, where the order of the polynomial can be set by the user, with cubic as a default (note that this step relies on the identification of matching phonons at different volumes, as described in [72]); iii) the minimization of $G(T, V, p)$ with the BFGS and the L-BFGS-B minimisation algorithms implemented in SciPy [38], with the unbounded BFGS being the default option. The following code shows how quasi-harmonic thermodynamic properties of a system can be computed at given temperatures and pressures from a single .out file of a QHA CRYSTAL calculation (`file` in the example below) via strategy I) implemented in method *thermo_freq*:

```
from CRYSTALpytools.thermodynamics import Quasi_harmonic
import numpy as np

Cqha = Quasi_harmonic().from_QHA_file('file')
Cqha.thermo_freq(temperature=np.linspace(10,1000,11),
                 pressure=np.linspace(0,10,6))

print(Cqha.equilibrium_volume[0,:])
```

In this case, quasi-harmonic thermodynamic properties are computed at 11 equally-spaced temperatures in the range [10-1000] K and at 6 equally-spaced pressures in the range [0-10] GPa. The last line of the script prints the computed equilibrium volumes at the 11 temperatures at zero pressure.

The second strategy, implemented in method *thermo_eos*, involves three steps: i) fitting the Helmholtz free energy $F(V;T)$ to a variety of EoS, as implemented in pymatgen [29], with third-order Birch-Murnaghan being the default one [79], or to polynomials of different orders; ii) getting the thermal pressure analytically from $\tilde{p}(V;T) = -\partial F(V;T)/\partial V$; iii) the equilibrium volume at a given temperature and pressure $V(T, p)$ is obtained by minimizing the square root residue of fitted pressure: $V(T, p) = \min\left\{\sqrt{\left[\tilde{p}(V;T) - p\right]^2}\right\}$. The following code illustrates how to get quasi-harmonic thermodynamic properties from four separate harmonic phonon CRYSTAL calculations (each performed at a different volume) by fitting the Helmholtz free energy $F(V;T)$ to the Birch-Murnaghan EoS, via strategy II) implemented in method *thermo_eos*:

```
from CRYSTALpytools.thermodynamics import Quasi_harmonic
import numpy as np

Cqha = Quasi_harmonic().from_HA_files(['file1', 'file2', '
    file3','file4'], sort_phonon=False)
Cqha.thermo_eos(eos_method='birch_murnaghan',
                temperature=np.linspace(10,300,7),
                pressure=np.linspace(0,0.5,6))
```

Strategy II) is preferable for medium-to-large sized systems with low symmetry where the one-to-one identification of matching phonons at different volumes of strategy I) may prove critical. Table 1 shows the comparison between the zero pressure thermal expansion of Form I of the paracetamol crystal as obtained with the two strategies presented above.

**Table 1**
Computed thermal expansion at zero pressure of Form I of paracetamol crystal with the two strategies, I) and II), described in the text in the temperature range [0-300] K.

| $T$ (K) | $V$ (Å$^3$) | |
|---|---|---|
| | I) | II) |
| 0 | 773.52 | 773.30 |
| 50 | 775.06 | 774.63 |
| 100 | 778.29 | 777.70 |
| 150 | 780.79 | 780.30 |
| 200 | 782.54 | 782.22 |
| 250 | 783.80 | 783.64 |
| 300 | 784.74 | 784.73 |

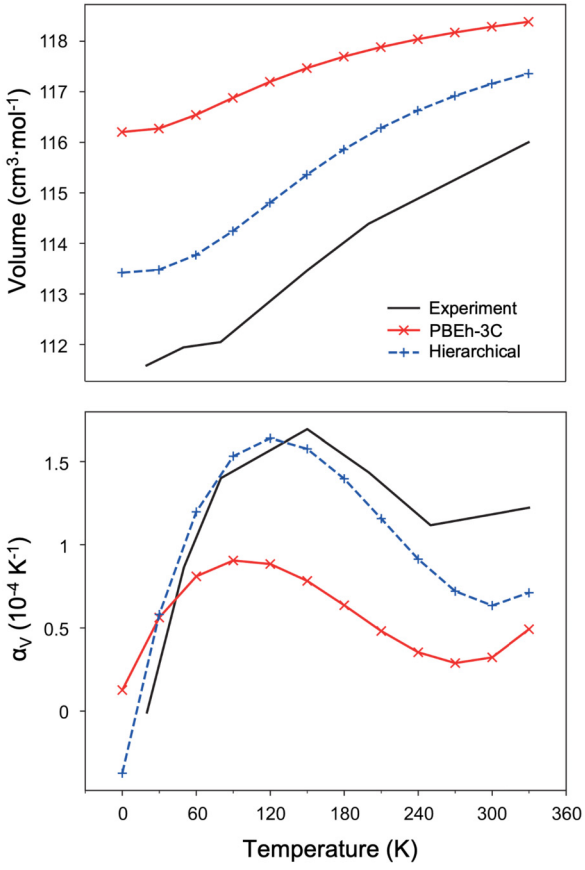



**Fig. 7.** Volume (upper panel) and thermal expansion coefficient (lower panel) of Form I paracetamol crystal.

The implementation of the *Thermodynamics* module of `CRYSTALpytools` is flexible enough to explore new computational strategies. For instance, in the following example, a hierarchical method is adopted to determine the thermal expansion of Form I paracetamol crystal, which combines the static electronic energy from plane-wave DFT (PW-DFT) calculations and phonons from CRYSTAL. PW-DFT is free from the well-known basis set superposition error (BSSE) of atom-centered Gaussian orbitals, while CRYSTAL uses local Gaussian orbitals but implements various cost-effective semi-empirical corrections to the DFT [80]. Both factors are critical to predicting the thermodynamic properties of weakly bound materials such as molecular crystals.

Here, the thermal expansion of Form I paracetamol crystal is computed in two ways: i) with a consistent CRYSTAL calculation of both the electronic and phonon contributions with the PBEh-3c composite method [81]; and ii) with a PW-DFT calculation of the electronic energy by QUANTUM-ESPRESSO [1] (with the B86bPBE-XDM functional, built-in PAW, and 1100 eV energy cut-off), and

phonons from the PBEh-3c CRYSTAL calculation. Results obtained with the two approaches are shown in Fig. 7 and compared to experimental data [82].

## 5. Conclusions

In this paper, the `CRYSTALpytools` open source Python project, which implements a user-friendly interface to the CRYSTAL code for quantum-mechanical condensed matter simulations has been presented. The project provides functionalities to write and read CRYSTAL input and output files for several types of calculations, and to extract relevant information. It implements tools to translate CRYSTAL objects (the central data structure of the project) to and from the Structure and Atoms objects of pymatgen and ASE, respectively. Several computed properties, including elastic constants and phonons, can be post-processed to get extra insight. All these functionalities make `CRYSTALpytools` a powerful tool to design computationally effective workflows to interact with the CRYSTAL program. Additionally, several tools have been developed to plot results in a variety of styles for rapid and precise visual analysis.

## Declaration of competing interest

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

## Data availability

Data will be made available on request.

## Acknowledgements

This research has received funding from the Project CH4.0 under the MUR program “Dipartimenti di Eccellenza 2023-2027” (CUP: D13C22003520001).

## References

[1] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G.L. Chiarotti, M. Cococcioni, I. Dabo, et al., J. Phys. Condens. Matter 21 (39) (2009) 395502.
[2] J. Hafner, G. Kresse, in: Properties of Complex Inorganic Solids, Springer, 1997, pp. 69–82.
[3] X. Gonze, F. Jollet, F.A. Araujo, D. Adams, B. Amadon, T. Applencourt, C. Audouze, J.-M. Beuken, J. Bieder, A. Bokhanchuk, et al., Comput. Phys. Commun. 205 (2016) 106–131.
[4] P. Blaha, K. Schwarz, F. Tran, R. Laskowski, G.K. Madsen, L.D. Marks, J. Chem. Phys. 152 (7) (2020) 074101.
[5] S.J. Clark, M.D. Segall, C.J. Pickard, P.J. Hasnip, M.I. Probert, K. Refson, M.C. Payne, Z. Kristallogr. Cryst. Mater. 220 (5–6) (2005) 567–570.
[6] E. Apra, E.J. Bylaska, W.A.D. Jong, N. Govind, K. Kowalski, T.P. Straatsma, M. Valiev, H.J.J. van Dam, Y. Alexeev, J. Anchell, et al., J. Chem. Phys. 152 (18) (2020) 184102.
[7] J. Hutter, M. Iannuzzi, F. Schiffmann, J. VandeVondele, Wiley Interdiscip. Rev. Comput. Mol. Sci. 4 (1) (2014) 15–25.
[8] R. Dovesi, R. Orlando, B. Civalleri, C. Roetti, V.R. Saunders, C.M. Zicovich-Wilson, Z. Kristallogr. 220 (5–6) (2005) 571–573.
[9] R. Dovesi, R. Orlando, A. Erba, C.M. Zicovich-Wilson, B. Civalleri, S. Casassa, L. Maschio, M. Ferrabone, M. De La Pierre, P. D'Arco, Y. Noël, M. Causà, M. Rérat, B. Kirtman, Int. J. Quant. Chem. 114 (19) (2014) 1287–1317.
[10] R. Dovesi, A. Erba, R. Orlando, C.M. Zicovich-Wilson, B. Civalleri, L. Maschio, M. Rérat, S. Casassa, J. Baima, S. Salustro, B. Kirtman, Comput. Mol. Sci. 8 (4) (2018) e1360.
[11] A. Erba, J.K. Desmarais, S. Casassa, B. Civalleri, L. Doná, I.J. Bush, B. Searle, L. Maschio, L. Edith-Daga, A. Cossard, et al., J. Chem. Theory Comput. (2022).
[12] E. Clementi, C. Roetti, At. Data Nucl. Data Tables 14 (3) (1974) 177–478.
[13] J. Muscat, A. Wander, N. Harrison, Chem. Phys. Lett. 342 (3–4) (2001) 397–401.
[14] J.K. Desmarais, J.-P. Flament, A. Erba, Phys. Rev. B 101 (23) (2020) 235142.
[15] J.K. Desmarais, A. Erba, J.-P. Flament, B. Kirtman, J. Chem. Theory Comput. 17 (8) (2021) 4697–4711.
[16] J.K. Desmarais, A. Erba, J.-P. Flament, B. Kirtman, J. Chem. Theory Comput. 17 (8) (2021) 4712–4732.
[17] A. Cossard, J.K. Desmarais, S. Casassa, C. Gatti, A. Erba, J. Phys. Chem. Lett. 12 (7) (2021) 1862–1868.
[18] A. Cossard, S. Casassa, C. Gatti, J.K. Desmarais, A. Erba, Molecules 26 (14) (2021) 4227.
[19] J. Desmarais, A. Erba, R. Dovesi, Theor. Chem. Acc. 137 (2018) 28.
[20] C. Ribaldone, S. Casassa, AIP Adv. 12 (1) (2022) 015323.
[21] M. Destefanis, C. Ravoux, A. Cossard, A. Erba, Minerals 9 (2019) 16.
[22] P.A. Banks, J. Maul, M.T. Mancini, A.C. Whalley, A. Erba, M.T. Ruggiero, J. Mater. Chem. C 8 (2020) 10917–10925.
[23] J. Maul, D. Ongari, S.M. Moosavi, B. Smit, A. Erba, J. Phys. Chem. Lett. 11 (20) (2020) 8543–8548.
[24] A. Erba, J. Maul, M. Ferrabone, P. Carbonniére, M. Rérat, R. Dovesi, J. Chem. Theory Comput. 15 (2019) 3755–3765.
[25] A. Erba, J. Maul, M. Ferrabone, R. Dovesi, M. Rérat, P. Carbonnière, J. Chem. Theory Comput. 15 (2019) 3766–3777.
[26] J. Maul, G. Spoto, L. Mino, A. Erba, Phys. Chem. Chem. Phys. 21 (48) (2019) 26279–26283.
[27] R.G. Schireman, J. Maul, A. Erba, M.T. Ruggiero, J. Chem. Theory Comput. 18 (2022) 4428–4437.
[28] A.H. Larsen, J.J. Mortensen, J. Blomqvist, I.E. Castelli, R. Christensen, M. Dułak, J. Friis, M.N. Groves, B. Hammer, C. Hargus, E.D. Hermes, P.C. Jennings, P.B. Jensen, J. Kermode, J.R. Kitchin, E.L. Kolsbjerg, J. Kubal, K. Kaasbjerg, S. Lysgaard, J.B. Maronsson, T. Maxson, T. Olsen, L. Pastewka, A. Peterson, C. Rostgaard, J. Schiøtz, O. Schütt, M. Strange, K.S. Thygesen, T. Vegge, L. Vilhelmsen, M. Walter, Z. Zeng, K.W. Jacobsen, J. Phys. Condens. Matter 29 (27) (2017) 273002.
[29] S.P. Ong, W.D. Richards, A. Jain, G. Hautier, M. Kocher, S. Cholia, D. Gunter, V.L. Chevrier, K.A. Persson, G. Ceder, Comput. Mater. Sci. 68 (2013) 314–319.
[30] X. Shao, O. Andreussi, D. Ceresoli, M. Truscott, A. Baczewski, Q. Campbell, M. Pavanello, QEpy: quantum ESPRESSO in python, https://gitlab.com/shaoxc/qepy, 2022.
[31] S.P. Huber, S. Zoupanos, M. Uhrin, L. Talirz, L. Kahle, R. Häuselmann, D. Gresch, T. Müller, A.V. Yakutovich, C.W. Andersen, et al., Sci. Data 7 (1) (2020) 1–18.
[32] A.W. Sousa da Silva, W.F. Vranken, BMC Res. Notes 5 (1) (2012) 1–8.
[33] M. Ceriotti, J. More, D.E. Manolopoulos, Comput. Phys. Commun. 185 (3) (2014) 1019–1026.
[34] H.L. Röst, U. Schmitt, R. Aebersold, L. Malmström, Proteomics 14 (1) (2014) 74–77.
[35] D. Lampert, M. Wu, Environ. Model. Softw. 68 (2015) 166–174.
[36] S. Kundu, S. Bhattacharjee, S.-C. Lee, M. Jain, Comput. Phys. Commun. 233 (2018) 261–268.
[37] C.R. Harris, K.J. Millman, S.J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N.J. Smith, R. Kern, M. Picus, S. Hoyer, M.H. van Kerkwijk, M. Brett, A. Haldane, J.F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T.E. Oliphant, Nature 585 (2020) 357–362.
[38] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C.J. Carey, İ. Polat, Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, P. van Mulbregt, Nat. Methods 17 (2020) 261–272.
[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., J. Mach. Learn. Res. 12 (2011) 2825–2830.
[40] K. Momma, F. Izumi, J. Appl. Crystallogr. 41 (3) (2008) 653–658.
[41] M.D. Hanwell, D.E. Curtis, D.C. Lonie, T. Vandermeersch, E. Zurek, G.R. Hutchison, J. Cheminform. 4 (1) (2012) 1–17.
[42] https://github.com/crystal-code-tools, 2023.
[43] R. Rocca, M. Sgroi, B. Camino, M. D'Amore, A. Ferrari, Nanomaterials 12 (2022) 1832.
[44] W.P. Comaskey, F. Bodo, A. Erba, J.L. Mendoza-Cortes, J.K. Desmarais, Phys. Rev. B 106 (20) (2022) L201109.
[45] J.K. Desmarais, J.-P. Flament, A. Erba, J. Phys. Chem. Lett. 10 (13) (2019) 3580–3585.
[46] Jacques K. Desmarais, Jean-Pierre Flament, Alessandro Erba, J. Chem. Phys. 151 (7) (2019) 074107.
[47] J.K. Desmarais, J.-P. Flament, A. Erba, Phys. Rev. B 102 (23) (2020) 235118.
[48] J.K. Desmarais, S. Komorovsky, J.-P. Flament, A. Erba, J. Chem. Phys. 154 (20) (2021) 204110.
[49] F. Bodo, J.K. Desmarais, A. Erba, Phys. Rev. B 105 (12) (2022) 125108.
[50] J.F. Nye, et al., Physical Properties of Crystals: Their Representation by Tensors and Matrices, Oxford University Press, 1985.
[51] W.F. Perger, J. Criswell, B. Civalleri, R. Dovesi, Comput. Phys. Commun. 180 (2009) 1753–1759.
[52] A. Erba, A. Mahmoud, R. Orlando, R. Dovesi, Simulations 41 (2014) 151–160.
[53] A. Erba, D. Caglioti, C.M. Zicovich-Wilson, R. Dovesi, J. Comput. Chem. 38 (2017) 257–264.
[54] A. Erba, A. Mahmoud, D. Belmonte, R. Dovesi, Grossular Andradite Silicate Garnets 140 (2014) 124703.

[55] R.E. Newnham, Properties of Materials: Anisotropy, Symmetry, Structure, Oxford University Press on Demand, 2005.
[56] A. Marmier, Z.A. Lethbridge, R.I. Walton, C.W. Smith, S.C. Parker, K.E. Evans, Comput. Phys. Commun. 181 (12) (2010) 2102–2115.
[57] R. Gaillac, P. Pullumbi, F.-X. Coudert, J. Phys. Condens. Matter 28 (27) (2016) 275201.
[58] R.F. Bader, T. Nguyen-Dang, Quantum Theory of Atoms in Molecules–Dalton Revisited, Advances in Quantum Chemistry, vol. 14, Elsevier, 1981, pp. 63–124.
[59] C. Gatti, Z. Kristallogr. 220 (5–6) (2005) 399–457.
[60] P. Popelier, Coord. Chem. Rev. 197 (1) (2000) 169–189.
[61] C. Gatti, V. Saunders, C. Roetti, J. Chem. Phys. 101 (12) (1994) 10686–10696.
[62] S. Casassa, A. Erba, J. Baima, R. Orlando, J. Comput. Chem. 36 (26) (2015) 1940–1946.
[63] G. Sansone, A. Ferretti, L. Maschio, J. Chem. Phys. 147 (11) (2017) 114101.
[64] K. Doll, Comput. Phys. Commun. 137 (2001) 74–88.
[65] K. Doll, V. Saunders, N. Harrison, Int. J. Quant. Chem. 82 (2001) 1–13.
[66] K. Doll, R. Dovesi, R. Orlando, Theor. Chem. Acc. 112 (5–6) (2004) 394–402.
[67] K. Doll, R. Dovesi, R. Orlando, Theor. Chem. Acc. 115 (5) (2006) 354–360.
[68] K. Doll, Mol. Phys. 108 (3–4) (2010) 223–227.
[69] C.M. Zicovich-Wilson, F. Pascale, C. Roetti, V.R. Saunders, R. Orlando, R. Dovesi, J. Comput. Chem. 25 (2004) 1873–1881.
[70] J. Baima, M. Ferrabone, R. Orlando, A. Erba, R. Dovesi, Phys. Chem. Miner. 43 (2016) 137–149.
[71] A. Erba, J. Chem. Phys. 141 (2014) 124115.
[72] A. Erba, M. Shahrokhi, R. Moradian, R. Dovesi, J. Chem. Phys. 142 (2015) 044114.
[73] A. Erba, J. Maul, R. Demichelis, R. Dovesi, Phys. Chem. Chem. Phys. 17 (2015) 11670–11677.
[74] A. Erba, J. Maul, M. De La Pierre, R. Dovesi, J. Chem. Phys. 142 (2015) 204502.
[75] A. Erba, J. Maul, B. Civalleri, Chem. Commun. 52 (2016) 1820–1823.
[76] R. Dovesi, C. Pisani, C. Roetti, V. Saunders, Phys. Rev. B 28 (10) (1983) 5781.
[77] V. Saunders, C. Freyria-Fava, R. Dovesi, L. Salasco, C. Roetti, Mol. Phys. 77 (4) (1992) 629–665.
[78] In CRYSTAL, the truncation thresholds of bielectronic Coulomb and exchange integrals is based on overlap (the TOLINTEG keyword). Finite changes in lattice and atomic coordinates make the consistent definition of truncation thresholds rather difficult, which might lead to numerical noises on the energy-volume curve. So far, the FIXINDEX option, which fixes the thresholds according to a reference geometry, is implemented for the CRYSTAL QHA module and not available for multiple independent harmonic phonon calculations. The current implementation of CRYSTALpytools is compatible with outputs of CRYSTAL QHA calculations so the users can still utilise its flexibility without the problem of inconsistent truncation. Nevertheless, it should also be noted that this problem can be alleviated by increasing TOLINTEG and exactly computing the bielectronic integrals (NOBIPOLA). Both methods reduce the loss of significant Coulomb or exchange serials which might be cut off or approximated otherwise.
[79] F. Birch, Phys. Rev. 71 (1947) 809–824.
[80] L.M. LeBlanc, A. Otero-de-la Roza, E.R. Johnson, J. Chem. Theory Comput. 14 (4) (2018) 2265–2276.
[81] S. Grimme, J.G. Brandenburg, C. Bannwarth, A. Hansen, J. Chem. Phys. 143 (5) (2015) 054107.
[82] C. Wilson, Z. Kristallogr. 215 (11) (2000) 693–701.