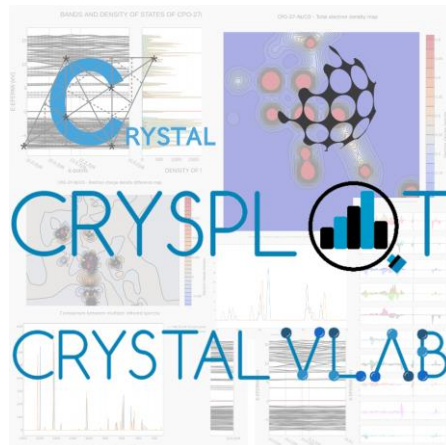




**Università degli Studi di Torino**

PhD Programme in Chemical and Materials Sciences XXXII<sup>nd</sup> Cycle

**Development of web-oriented tools to analyse and visualise properties of crystalline solids**



**Giorgia Beata**

Supervisor:  
Prof. Bartolomeo Civalleri



## **Università degli Studi di Torino**

PhD Programme in Chemical and Materials Sciences XXXII<sup>nd</sup> cycle

### **Development of web-oriented tools to analyse and visualise properties of crystalline solids**

Candidate: **Giorgia Beata**

Supervisor: Prof. **Bartolomeo Civalleri**

Jury Members: Prof. **Piero Ugliengo**  
Università di Torino  
Dipartimento di Chimica

Prof. **Furio Corà**  
University College of London  
Department of Chemistry

Prof. **Alfonso Pedone**  
Università di Modena e Reggio Emilia  
Dipartimento di Scienze Chimiche e Geologiche

Head of the Doctoral School: Prof. Alberto Rizzuti

PhD Programme Coordinator: Prof. Mario Chiesa

Torino, 2019

# Index

1	Introduction.....	5
	Bibliography.....	10
2	CRYSPLOT .....	12
	What it does and how it works .....	12
	CRYSPLOT web page.....	14
	CRYSPLOT: Under the hood .....	17
	CRYSPLOT at work: Selected case studies .....	20
	Bibliography.....	31
3	CRYSTAL VLab .....	33
	CRYSTAL code in short .....	33
	CRYSTAL VLab structure .....	37
	CRYSTAL VLab: BUILD.....	38
	CRYSTAL VLab: COMPUTE .....	49
	CRYSTAL VLab: VIEW.....	54
	CRYSTAL VLab: FILE MANAGER.....	58
	CRYSTAL VLab: Under the hood .....	59
	Bibliography.....	61
4	Conclusions and Outlook.....	62
	Appendix 1 .....	64
	Appendix 2 .....	66
	Appendix 3 .....	67
	Appendix 4 .....	75
	Appendix 5 .....	76
	Appendix 6 .....	91
	Appendix 7 .....	94
	Published Papers .....	102



# 1 Introduction

Visualisation tools and graphical interfaces have become more and more important to analyse, manage, help in understanding, and present scientific data. Nowadays, this is particularly true in the realm of computational molecular and solid-state chemistry for which a paramount number of data can be obtained from more robust and multipurpose computational codes and through the accessibility of powerful computing facilities. With this amount of numbers, scientists need a method to understand them, what they mean and what conclusions take; as quoted by Valle<sup>1</sup>, a way to help scientists is visualisation, because it activates a brain part that it is stopped in the case of pure rational analysis. Then, to use those parts of brain, we have to transform numbers in colours and shapes, because in this way scientists can have new insights and insight is the heart of the cognition process.

A possible definition of visualisation can be "Visualisation is the use of computer-support, interactive, visual representations of data to amplify cognition"<sup>1</sup>; in this concept all visualisation usages are condensed.

- *Computer-supported* means that visualisation can support scientist brain because without them they are limited in their arguments; it is important to note that the definition says *Computer-supported* and not *Computer-based* because visualisation must be used as a support to human reasoning, which must always be present in the cognitive process.
- *Interactive* means that visualisation allows scientists to be able to interact with the images, e.g. rotate them, move, and see their relationship with the space.
- *Visual representation of data* means that converting numbers into graphic objects allows us to see new thing, to have new ideas, to have insights never before.

- *To amplify cognition* means that the ultimate goal of visualisation is not to reproduce nice charts, but rather to stimulate the intuition of the scientist. In this perspective, therefore, even a simple line chart becomes a powerful visualisation tool if it allows the scientist to reach new conclusions.

Visualisation has also an important role in discovery cycle, either in the case in which scientist has an insight or in case he/she has no idea. In the first situation, the process starts with an insight that becomes a model; this model can be visualised and the visualisation can improve it, creating a cycle in which the model gradually improves. The scientist therefore knows what he/she wants to find and directs the visualisation in a precise direction. Instead, in the second instance, scientists do not know what they want to find and then the possibility to visualise computed numbers in shapes and colours can guide cognition processes with a new way to look at the data.

Despite Richard Weinberg says that "Computing, and in particular supercomputing, without visualisation is like assembling a jigsaw puzzle in the dark"<sup>2</sup>, we must not forget that visualisation is not a total substitute for a careful analysis of the data and then scientists must have a critical view of the images produced.

With this in mind, in my PhD I focused on the development of visualisation and graphical tools to be applied in computational solid-state chemistry. In the field of computational solid-state chemistry, several different softwares exist for the ab initio study of physical and chemical properties of periodic systems as polymers, surfaces, and crystalline solids. Among them a prominent role is played by the CRYSTAL code<sup>3-8</sup>, a software developed since mid-70s of the last century by the Theoretical Chemistry Group of the University of Torino and later in collaboration with the Computational Materials Science Group at Daresbury Laboratory (UK). Although CRYSTAL has recently shown a tremendous improvement in terms of advanced and powerful algorithms,<sup>5,7</sup> particularly for speeding up calculations of

the two parallel versions<sup>9-11</sup> and for extending the number of properties that can be computed,<sup>8</sup> the graphical tools for the analysis and visualisation of the predicted results have remained at a less developed stage. The CRYSTAL package performs ab initio calculations of the ground state energy, electronic wave function and properties of periodic systems at the HF and DFT level of theory<sup>8</sup>. Presently, graphical tools available to analyse data computed with the CRYSTAL code are not unique and in some cases obsolete. Some of the computed properties and related data can be plotted by means of Gnuplot<sup>12</sup>. Gnuplot is a widespread tool for plotting scientific data. It was originally created to allow scientists and students to visualise mathematical functions and data interactively, but nowadays it has grown to support many non-interactive uses such as web scripting. For instance, CRYSTAL can create files that can be read by Gnuplot to plot simulated vibrational spectra. Other properties like band structure and density of states can be plotted with the homemade CrGra package whose latest release dates back to 2006. The development of the CrGra suite of programs started almost thirty years ago by the CRYSTAL team to create graphic plots in the PostScript language that could be easily printed. CrGra2006 processes data written by CRYSTAL in the Fortran unit "fort.25"<sup>8</sup>. The CrGra suite is comprised of three modules: maps06 to draw contour maps (e.g. charge and spin density, electrostatic potential), bands06 to plot band structures and doss06 to plot total and projected density of states. However, the code was not updated for plotting data related to other properties that are now available from CRYSTAL (e.g. simulated IR/Raman spectra, topological analysis related quantities and electron conductivity).

In addition to these plotting tools, also other graphical tools are available to visualise data from CRYSTAL such as DLV<sup>13</sup> and XCRYSDEN<sup>14</sup>. Even if they provide a more extended graphical interface to the code (e.g. visualisation of the crystalline structure and related features, animation of vibrational frequencies, and so on, see ref. 13 and 14 for further details) they are currently not fully updated to the last release of the code and not available for all operating systems.

Therefore, for the first part of the PhD project I decided to create an up to date modern web-oriented visualisation tool to be machine independent, easy to use and freely accessible to users from all over the world through Internet browsers. Briefly, CRYSPLOT<sup>15</sup> (<http://crysplot.crystalsolutions.eu/>) extends and improves over the capabilities of CrGra to plot additional data (e.g. simulated IR/Raman spectra) and to manage multiple datasets. It also allows users to customize plots as detailed in the next section.

As a second step, I tried to understand how my project could help a beginner user of CRYSTAL to make more gradual his learning curve in using the code. My conclusion was that a complicate task for a user is the creation of the input deck because one has to define the structure of the examined system, the method and basis set, and other computational details along with several possible keywords to compute different properties. In addition, such keywords must be specified in a precise order in the input file. Furthermore, for a beginner the analysis and the visualisation of computed results are really complex.

Therefore to provide a new tool to help and facilitate the use of CRYSTAL, I designed CRYSTAL VLab (<http://vlab.crystalsolutions.eu/>), an all-purpose application that allows users to create their own input decks, run calculations and visualise the results. For each of these three steps, a wizard with main settings has been designed.

CRYSTAL VLab, as well as CRYSPLOT, is a web-based tool to make it machine independent and freely accessible to users from all over the world through Internet browsers and with a modern, simple and easy to use layout. CRYSPLOT and CRYSTAL VLab are really new in the world of visualisation of computational material science, because a lot of software exists but a few use a web-based approach. Indeed, MaterialsStudio, Avogadro, Ovito, Vesta and P4VASP are very well known as visualisation tools but none of them is web-based and cross-platform at the same time.



The outcomes of my PhD project are then CRYSPLOT and CRYSTAL VLab. In the 1st year and half of 2nd one I created CRYSPLOT, a new web-based graphical tool that allows to visualise results for many different properties (e.g. band structure, density of states, simulated IR and Raman spectra, ...) and customize the resulting plots. In the second half of 2nd year and 3rd one, I worked on the designed and creation of CRYSTAL VLab, a more complex graphical tool that can help users not only to visualise, but also to build/read their own inputs and run the CRYSTAL code. Both tools will be presented and described in the next chapters of the present thesis.

Finally, it is worthy to note that the present research work was carried out within a PhD in Apprenticeship. The PhD activity was a joint project between the University of Torino and Aethia s.r.l. where I worked for my apprenticeship. This project was also supported by Regione Piemonte (Avviso Pubblico approvato con D.D. 537 del 03/08/2016).

## Bibliography

1. M. Valle, *Int. J. Quantum Chem.* **2013**, 113, 2040-2052
2. K. A. Frenkel, *Commun. ACM* **1988**, 31, 111.
3. R. Dovesi, R. Orlando, B. Civalleri, C. Roetti, V. R. Saunders, and C. M. Zicovich-Wilson, *Z. Kristallogr.* **2005**, 220, 571-573
4. R. Dovesi, V. R. Saunders, C. Roetti, R. Orlando, C. M. Zicovich-Wilson, F. Pascale, B. Civalleri, K. Doll, N. M. Harrison, I. J. Bush, P. D'Arco, and M. Llunell, *CRYSTAL09 User's Manual*; University of Torino, Torino, **2009**
5. 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. Noel, M. Causa, M. Rerat, B. Kirtman. *Int. J. Quantum Chem.* **2014**, 114, 1287-1317
6. R. Dovesi, V. R. Saunders, C. Roetti, R. Orlando, C. M. Zicovich-Wilson, F. Pascale, B. Civalleri, K. Doll, N. M. Harrison, I. J. Bush, P. D'Arco, M. Llunell, M. Causà, Y. Noel, *CRYSTAL14 User's Manual*; University of Torino, Torino, **2014**
7. R. Dovesi, A. Erba, R. Orlando, C. M. Zicovich-Wilson, B. Civalleri, L. Maschio, M. Rerat, S. Casassa, J. Baima, S. Salustro, B. Kirtman. *WIREs Comput Mol Sci.* **2018**, 8, e1360
8. R. Dovesi, V. R. Saunders, C. Roetti, R. Orlando, C. M. Zicovich-Wilson, F. Pascale, B. Civalleri, K. Doll, N. M. Harrison, I. J. Bush, P. D'Arco, M. Llunell, M. Causà, Y. Noël, L. Maschio, A. Erba, M. Rerat and S. Casassa, *CRYSTAL17 User's Manual*; University of Torino, Torino, **2017**
9. R. Orlando, M. Delle Piane, I. J. Bush, P. Ugliengo, M. Ferrabone, R. Dovesi, *J. Comput. Chem.* **2012**, 33, 2276
10. A. Erba, J. Baima, I. Bush, R. Orlando, R. Dovesi, *J. Chem. Theory Comput.* **2017**, 13 (10), 5019–5027
11. S. Casassa, A. Erba, J. Baima, R. Orlando. *J. Comput. Chem.* **2015**, 36, 1940–1946

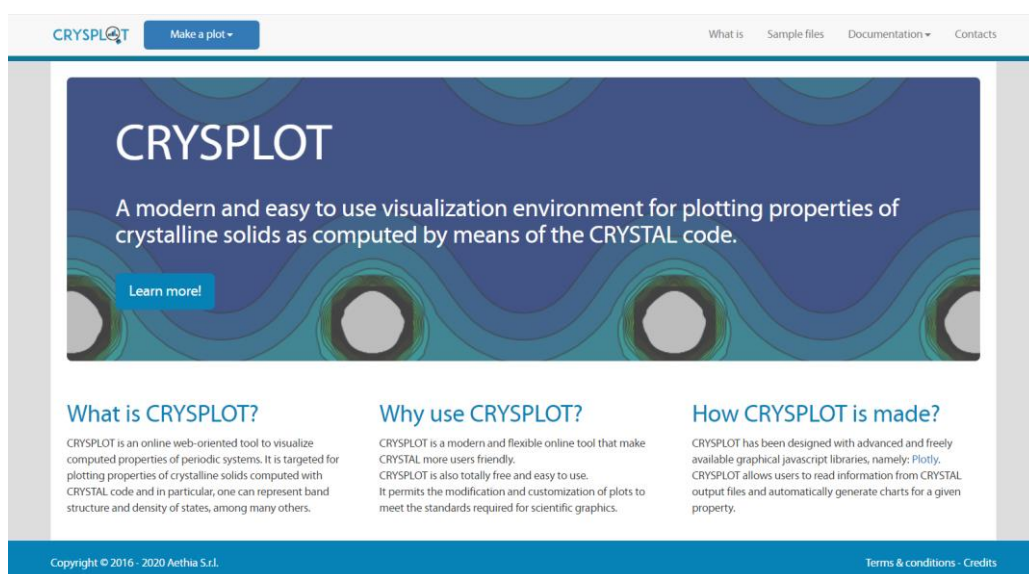
12. T. Williams, C. Kelley, H.-B. Broker, John Campbell, R. Cunningham, D. Denholm, G. Elber, R. Fearick, C. Grammes, L. Hart, L. Hecking, P. Juhasz, T. Koenig, D. Kotz, E. Kubaitis, R. Lang, T. Lecomte, A. Lehmann, A. Mai, B. Markisch, Ethan A Merritt, P. Mikulik, C. Steger, S. Takeno, T. Tkacik, J. Van der Woude, J. R. Van Zandt, A. Woo, J. Zellner, *Gnuplot 5.2.2: An Interactive Plotting Program* **2017**.
13. B.G. Searle, *Comp. Phys. Comm.*, **2001**, 137, 25.
14. A. Kokalj, *Comp. Mater. Sci.* **2003**, 28, 155. Code available from <http://www.xcrysden.org/>.
15. G. Beata, G. Perego, B. Civalleri "CRYSPLOT: a new tool to visualise physical and chemical properties of molecules, polymers, surfaces and crystalline solids", *J. Comput. Chem.*, **2019**, Volume 40, Issue 26, 2329-2338.

## 2 CRYSPLOT

CRYSPLOT<sup>1</sup> is a project born from the idea of giving to CRYSTAL users a simple, user-friendly and intuitive visualisation tool, but also complete and with high graphical performances. In the following, I will discuss in more detail technical aspects and features of CRYSPLOT (i.e. how it works, what it does,...) along with selected examples to show the capabilities of this new graphical tool.

### What it does and how it works

CRYSPLOT is publicly available online at <http://crispplot.crystalsolutions.eu/>.



*Figure 2-1. Screenshot of the CRYSPLOT home page*

In Figure 2-1, a screenshot of the CRYSPLOT homepage is shown. Basically, CRYSPLOT allows users to visualise Band Structure, Density of States, Electron Charge Density and Electrostatic Potential maps, Simulated Vibrational Spectra, Topological Analysis, Phonon Dispersion and Transport Properties computed with CRYSTAL on a machine independent platform. It also aims at offering users with easy-to-use options to modify and customize plots in order to meet standards required for scientific publications. It is worthy to note that, although CRYSPLOT is targeted to CRYSTAL, the same properties as computed from other programs could

be plotted by simply converting raw data to the CRYSTAL format as described in the Appendix of the CRYSTAL User's Manual<sup>2</sup>.

At the design stage of CRYSPLOT, I decided to create a cross platform tool for making it easily accessible to the largest number of users. For this reason I decided to organize CRYSPLOT as a website and develop it as a web application by using HTML5 markup language, CSS3 style sheet language, BOOTSTRAP front-end framework, JavaScript programming language and jQuery JavaScript library.

BOOTSTRAP is a free front-end framework for faster and easier web development that includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation, modals, image carousels and many other, as well as optional JavaScript plugins. In addition, BOOTSTRAP has the ability to easily create responsive designs, i.e. web sites that automatically adjust themselves to look good on all devices, from small smartphones to large desktops. BOOTSTRAP is easy to use and compatible with all modern browsers (Chrome, Firefox, Internet Explorer, Safari, and Opera).

From the programming language point of view, I used jQuery library in addition to just JavaScript language because jQuery makes much easier to use JavaScript on website; indeed it takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that can be called with a single line of code. jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

For the graphical part I have chosen PLOTLY<sup>3</sup> for 2D plots and JSmol<sup>4</sup> for 3D ones. Plotly is a high-level, declarative charting library, built on top of d3.js and stack.gl. In CRYSPLOT, I used its JavaScript version named plotly.js. Plotly.js uses stack.gl for high performance 2D and 3D charting and the charts are shipped with zoom, pan, hover, and click interactions like click-and-drag to zoom into a region, double-click to autoscale, click on legend items to toggle traces. Plotly.js ships with 20

chart types, including 3D charts, statistical graphs, and SVG maps; it abstracts the types of statistical and scientific charts that one would find in packages like matplotlib, ggplot2, or MATLAB. The charts are described declaratively as JSON objects and every aspect of the charts, such as colors, grid lines, and the legend, has a corresponding set of JSON attributes.

JSmol is the HTML5 modality of Jmol. Jmol is a free, open source viewer of molecular and crystalline structures. JSmol can be embedded into web pages and all the functionality of Jmol (as a standalone application) is also present in JSmol. JSmol is an interactive web browser object that can read all the files that Jmol reads and can do all the scripting that Jmol does. JSmol has both a console and a popup menu.

## CRYSPLIT web page

The CRYSPLIT website (see Figure 2-1) has a descriptive part (homepage and “What is”), an operative one (“Make a plot”) and user dedicated services (“Sample file”, “Documentation” and “Contacts”) (See Appendix 1). The “Make a plot” button is the heart of CRYSPLIT: it opens a cascade menu from which user can select the property to be plotted as shown in Figure 2-2.

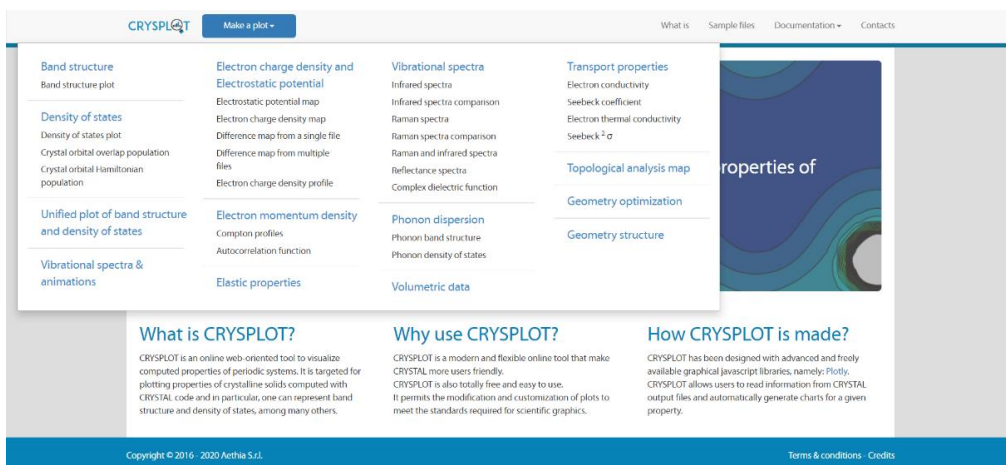


Figure 2-2. “Make a plot” cascade menu

In addition, the header of the web page contains links to the pages described above: "CRYSPLOT" brings to the homepage, "What is" links to a page on the main features of CRYSPLOT, "Sample files" to a page with sample files to visualise some properties plot, "Documentation" to different manuals and "Contacts" to the contact page. Then, under the header there is a tab in which the selected property will be plotted. Figure 2-2 also shows the list of available properties, namely: band structure, density of states, crystal orbital overlap population, crystal orbital Hamiltonian population, electron charge density map and profile, electrostatic potential, Compton profiles, autocorrelation function, infrared, Raman, reflectance and complex dielectric spectra, phonon band structure and density of states, topological analysis, electron conductivity, Seebeck coefficient, electron thermal conductivity and volumetric data analysis. There is also the possibility to plot elastic properties through the link to the Elate web site<sup>5</sup>, a web tool for the analysis of elastic tensors, developed by F. X. Coudert and co-workers.

In each property tab, users may choose the output file from their local PC (see button "Choose file") to upload the data for the selected property as previously computed by CRYSTAL. All data to be plotted are written in a formatted way (i.e. plain ASCII) either on the Fortran unit 25 (fort.25) or in separate files (e.g. BAND.DAT, IRSPEC.DAT, ...). The list of properties and the corresponding name of the auxiliary files is listed in Table 1 in Appendix 2.

Notably, since in crystalline systems many properties are tensors and anisotropic, CRYSPLOT also allows the simultaneous plot of multiple datasets as for the simulated Raman spectra of a single crystal model for which six sets of data are available for the six independent orientation of the Raman tensor (i.e.  $xx$ ,  $yy$ ,  $zz$ ,  $xy$ ,  $yz$ ,  $xz$ ) as will be shown later on.

The “Plot settings” lateral menu (hidden on the left of the CRYSPLOT web page) contains a palette of options for customizing the plot shown on the screen. CRYSPLOT has independent pages for every property and these pages have a specific “Plot settings” set of options.

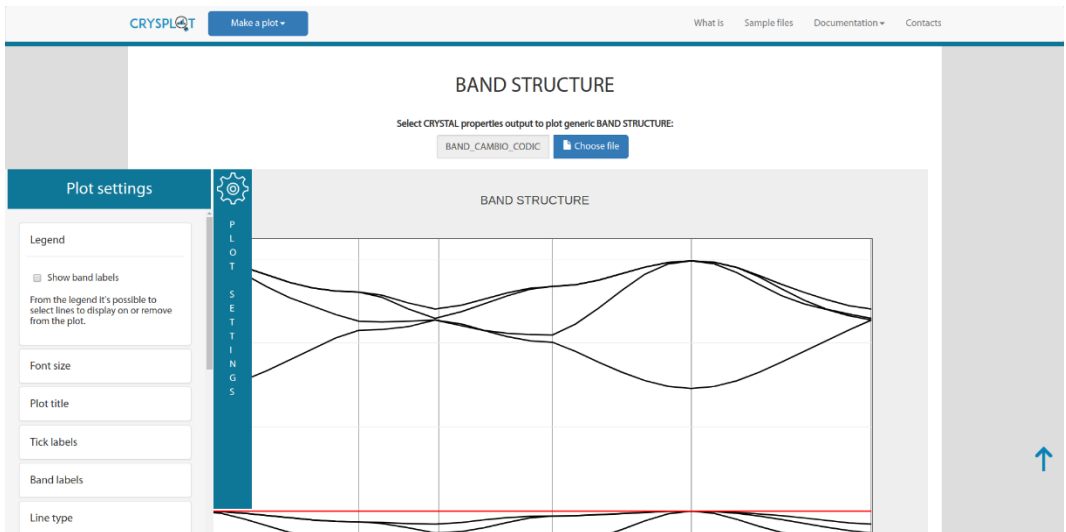


Figure 2-3. Example of plot settings palette as for the band structure plotting

For instance, Figure 2-3 shows an example of the “plot settings” menu for the band structure. Some of the available options to modify the plot are described in the following:

- *Legend* allows to show or hide the legend,
- *Font size* permits to decide the size of the plot and axis titles font, selecting from four sizes,
- *Plot title* permits to change the chart title,
- *Tick labels* allows to change the labels of the k points on the x-axis,
- *Band labels* allows users to change the name of each band (i.e. electronic level),
- *Line type* permits to decide which type of line is adopted for plotting the bands, namely: plain line, only markers or both,
- *Y-axis unit* controls the y-axis unit to be either Hartree (default) or eV,



- *Different layout for open shell case* enables user to change the appearance of the plot to have the band structure for alpha and beta electrons on the same graph or on two separate plots,
- *Fermi Energy line* allows one to show or hide the line of Fermi Energy,
- *Shifting Y-axis* allows to shift values for the value of Fermi Energy,
- *Axis range* allows to selected a sub-region of the plot by axis values,
- *Data on hover* allows to change the visualisation of data on the plot,
- *Grid* allows one to show or hide the grey grid of the plot,
- *Background color* modifies the color of the plot background from grey (default) to white,
- *Plot layout* allows to select the size of the plot.

Of course there are also other customizations created for particular plots, e.g. in the case of the density of states plot users can change line colours, in the case of spin polarized systems one can decide whether painting  $\alpha$  and  $\beta$  lines in pairs or with different colours. Also, total and projected density of states can be plotted either on the same graph or on different charts.

Finally, at the bottom of the page there is an "Export" section to save the actual plot as a picture. After selecting the file format (i.e. jpeg, png and svg) users must simply write the plot name and click on "Download Plot".

## **CRYSLOT: Under the hood**

CRYSLOT has been designed using an object oriented programming approach, where each class implements a different type of property. In particular CRYSLOT has a parent class called "Property" that is a sort of abstract class from which every other property inherits common attributes and methods. This object contains all the functions that are shared by all pages, e.g. function that allows downloading the plot in different format or the one that allows resizing the plot when the browser window is resized. Under this parent class, there are 5 children classes: "PropBand",

"PropDos", "PropSpectra", "PropBrCp" and "PropDensityMap". These classes are shared between different plot types, for example in "PropBand" class are defined some functions that are shared between some CRYSPLOT pages that plots band structure in different ways, i.e. "Band structure plot", "Unified plot of band structure and density of states" and "Phonon band structure". In essence, CRYSPLOT makes great use of inheritance between classes, avoiding unnecessary code repetition.

Regarding the programming language, CRYSPLOT is designed and written in JavaScript. In particular the code parses the input files, checks if the uploaded files are correct and, if they are, it reads the data and organizes them into objects ready to be plotted with plotly.js library. A peculiar feature of CRYSPLOT is the separation of the code dedicated to import data from the one dedicated to plot. Indeed, different import filters for each supported file format have been created, but the code that transfers data to plotly.js is always the same. When importing the data, the code checks whether the file format is correct, and if not, an error message appears. After data have been normalized, it is straightforward to visualise them through the common code for plotting. For instance, data for band structure can be imported either from BAND.DAT (or filename.BAND) file or the Fortran unit fort.25 (or filename.f25). In that case, the code has two different import functions: one for .BAND or .DAT format and another one for fort.25. Therefore, even if data are organized and formatted very differently, after importing them, users get the same JavaScript object that can be passed to plotly.js through the general plotting code for the final rendering of the graph. This structure makes CRYSPLOT modular and flexible so that it is very simple to add other input data formats. In fact, it is just a matter of creating a new import filter that organizes the values in the proper way.

After parsing data, my code passes information to a graphical tool, in particular plotly.js for 2D plots or JSmol for 3D ones.

Plotly.js requires three parameters:

- the id of the element in which the plot will be displayed on the page,
- data to be displayed,
- layout and customization settings of plot.

It's important to note that the method defined above allows to replicate the plot without loading the file.

In other words, if variables are stored server-side, it's possible to replicate the plot in different browsers and embed it in other web pages.

In the case of JSmol, I organized the code in two parts:

- The first one consists of a variable definition, usually called "Info", in which the main plot settings are defined (for example width and height) and those parameters that never change. The most important "Info" field is "script", because it defines the initialization of data. After "Info" definition, JSmol initializes a so-called "Applet" object linked with the "Info" variable, and through a JQuery method all this information is displayed in a specific area of the page.
- The second step consists of calling the previously defined functions that represent the data. These functions make use of Jmol.script() method, that requires two parameters, the name of the applet and a string with the commands to execute. Jmol.script() method automatically updates the plot with the changes written in the specified string.

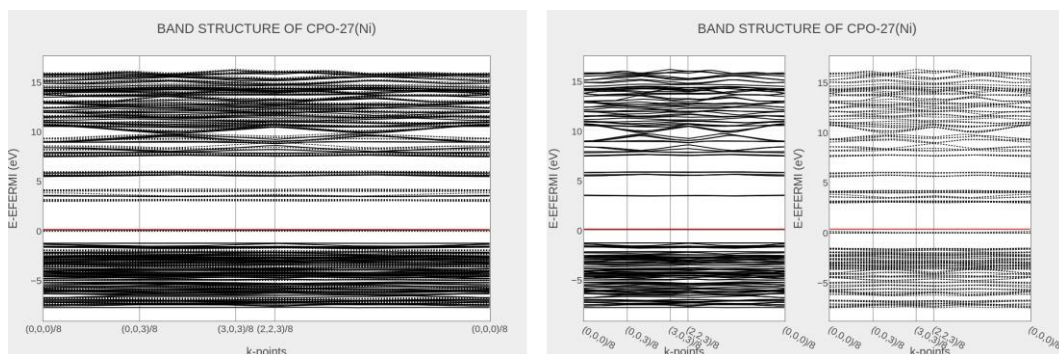
I also defined customization functions that exploit Jmol.script() method in order for example to allow users to change the lattice dimension or the colour of the structure.

Overall, CRYSPLOT is designed only with plotly.js and JSmol libraries and it has been tested with most common browsers on different operating systems by using standard test cases coming from the CRYSTAL code, sample tests included in the CRYSTAL Tutorial project and on-purpose designed tests.

## CRYSPLOT at work: Selected case studies

In this section, metal-organic frameworks (MOFs) are used as case studies to show some of the capabilities of CRYSPLOT. MOFs are a relatively new class of hybrid inorganic-organic materials that are comprised of an inorganic cluster (or metal) and an organic linker acting as secondary building units of a tridimensional and usually porous framework.<sup>6</sup> Due to their peculiar structure, MOFs are interesting for several applications ranging from gas adsorption, separation and capture to catalysis and drug delivery. Recently, they have become of interest for other technologically relevant applications such as sensing, lighting and optoelectronics.<sup>7-9</sup>

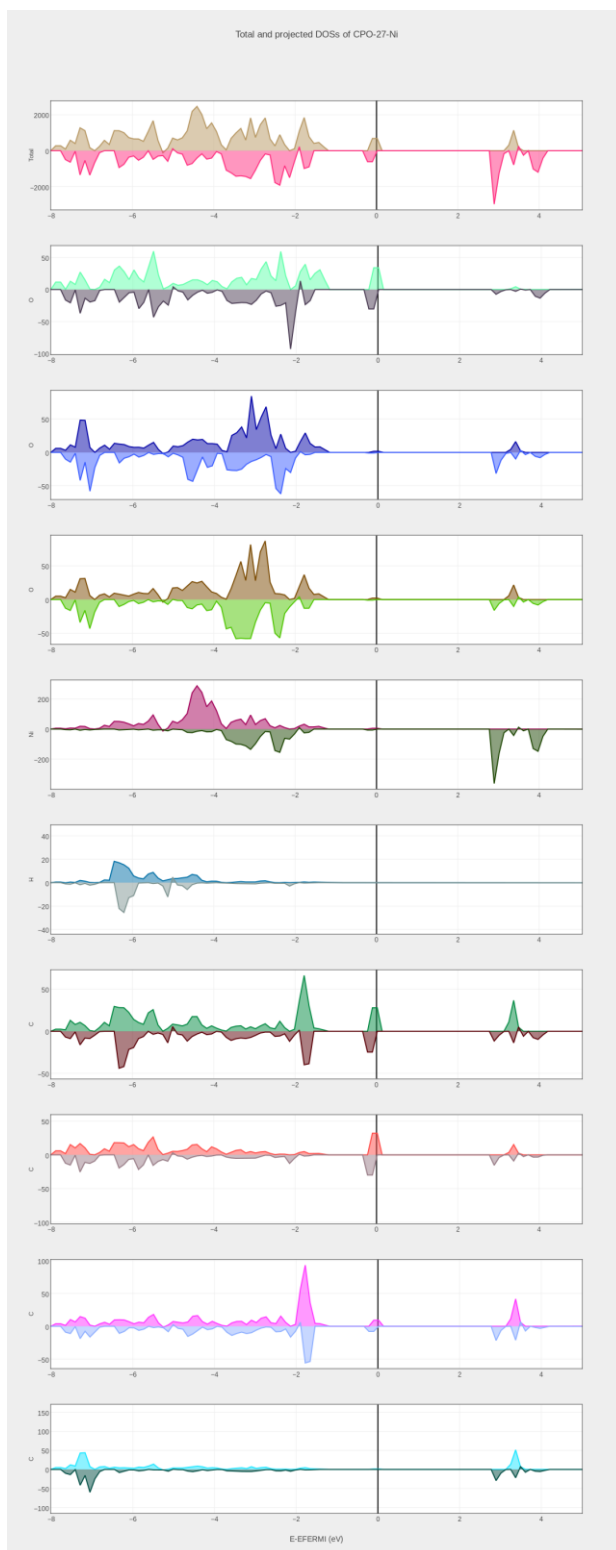
### Band structure and density of states



*Figure 2-4. Example of a band structure plot for the metal-organic framework CPO-27-Ni. Alpha and beta electronic bands are represented as continuous and dashed lines, respectively. The Fermi level is indicated with a red line.*

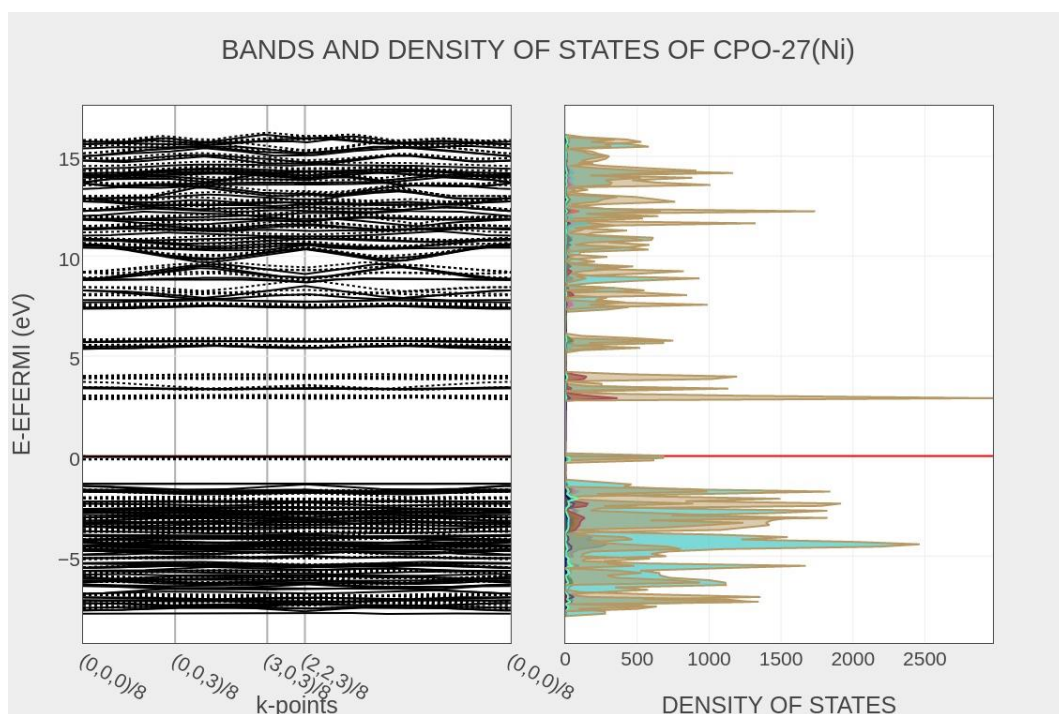
Figure 2-5. Example of total and projected density of states plot for CPO-27-Ni. Projected DOSs are plotted for all non-symmetry related atoms in the unit cell

As a first case study, we discuss the electronic structure of CPO-27-Ni. This MOF consists of metal-containing helical chains connected throughout space by the organic linker (i.e. 2,5-dihydroxyterephthalic acid, empirical formula:  $C_8H_2O_6Ni_2$ ) to form a honeycomb-like framework<sup>10-11</sup>. The resulting structure shows one-dimensional channels in which unsaturated metal sites are exposed at the inner surface. Due to the presence of unpaired electrons on the metal, CPO-27-Ni is a spin-polarized system. Results refer to a ferromagnetic configuration with Ni in a high-spin state. Reported data have been obtained with the M06-D method<sup>12</sup> with a TZVP basis



set<sup>13</sup>. In Figure 2-4, the band structure of CPO-27-Ni is shown. CRYSPLOT can automatically manage spin-polarized systems by plotting band structure for alpha and beta electrons together or separately in two graphs.

The total and the projected density of states have been plotted for all atoms in the asymmetric unit (i.e. 9 projected DOSs) is shown in Figure 2-5. As it can be seen, the coloured filled area plot is very effective in highlighting the role of alpha and beta electrons and multiple plots allow users to compare total and projected DOSs.



*Figure 2-6. Example of a combined band structure and density of states plot for CPO-27-Ni.*

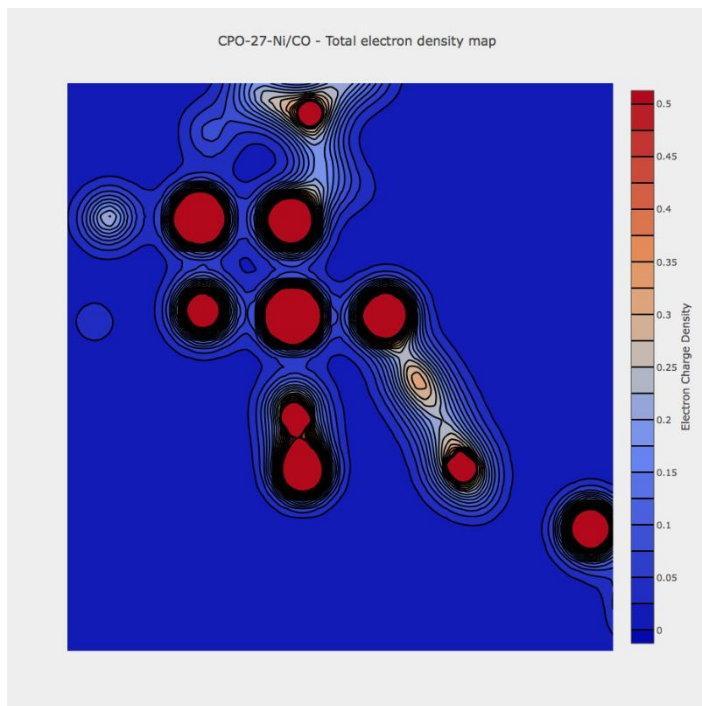
With CRYSPLOT, band structure and density of states can be also easily visualised in a combined plot. Figure 2-6, shows the band structure of CPO-27-Ni along with its total and projected density of states, that is the combination of Figures 2-4 and Figure 2-5. By default, the two plots are aligned to the Fermi level at 0 eV. From this graph, users can easily visualise which atom contributes to the corresponding

electronic band. For instance, for CPO-27-Ni, it can be readily seen that the top of the valence band corresponds to orbitals of the organic linker while the bottom of the conduction band is mainly dominated by the metal, thus suggesting a possible ligand-to-metal electronic transition.

### **Electron charge density maps**

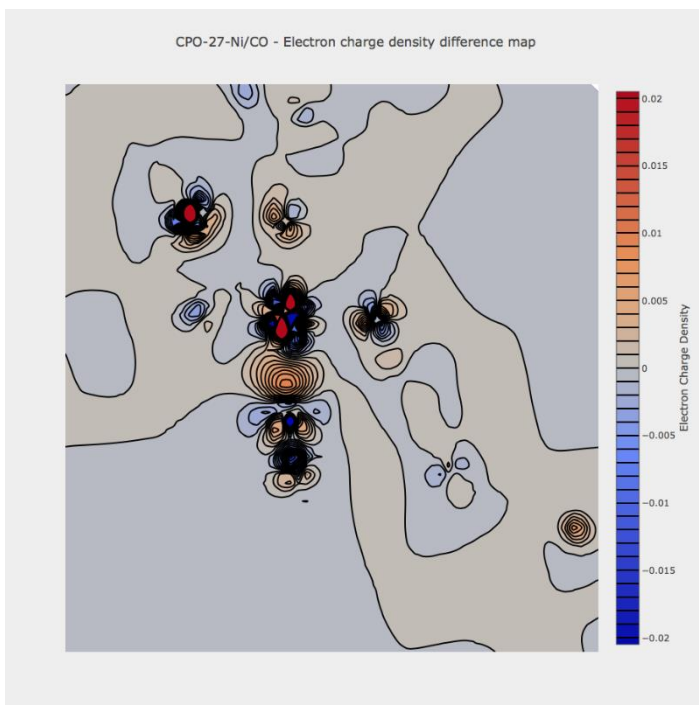
As an example of the plotting of 2D maps, the total electron density of a carbon monoxide molecule adsorbed on the inner surface of CPO-27-Ni is shown in Figure 2-7. Results have been computed at the M06-D/TZVP level of theory. The CO molecule strongly interacts with the Ni atoms that are exposed in the one-dimensional channels oriented along the  $c$ -axis.<sup>13</sup> The interaction is dominated by electrostatics and charge transfer effects due to the back-donation between the diffuse molecular orbitals of carbon monoxide and the unoccupied  $d$ -orbitals of the metal. To better appreciate the redistribution of the charge density upon the formation of the metal-CO interaction, it is then useful to plot the deformation charge density as shown in Figure 2-8.

CRYSPLOT can easily manage multiple sets of data and combine them to obtain a new map that corresponds to the difference (or sum) of the uploaded datasets. For instance, Figure 2-8 shows the deformation of the charge density when a CO molecule is adsorbed on top of the metal in CPO-27-Ni. The plot is obtained as a combination of the total electron charge densities of three systems, namely: the target system (i.e. CPO-27-Ni/CO) and the two separated subsystems (i.e. an array of CO molecules and the CPO-27-Ni alone). The electron density deformation map in Figure 8 shows positive values, in red, that correspond to charge accumulation while negative values, in blue, show where a depletion of charge occurs. As evident from Figure 2-8, it can be seen that the deformation of the charge density is due to back-donation effects related to the interaction between occupied  $d$ -orbitals on Ni and an empty  $\pi$ -acceptor level on CO.



*Figure 2-7. Contour plot of the total electron charge density map of a CO molecule adsorbed in the inner channels of CPO-27-Ni.*

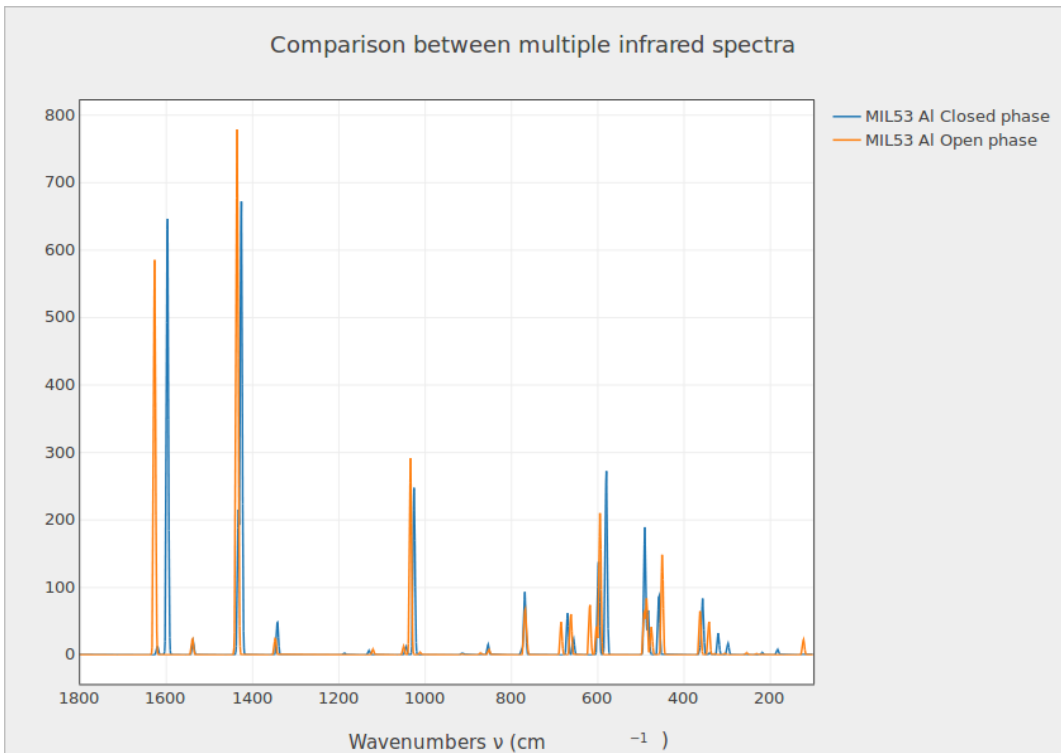
*Figure 2-8. Plot of the electron charge density deformation map of CO molecule adsorbed in CPO-27-Ni. Blue and red lines correspond to negative and positive values, respectively.*



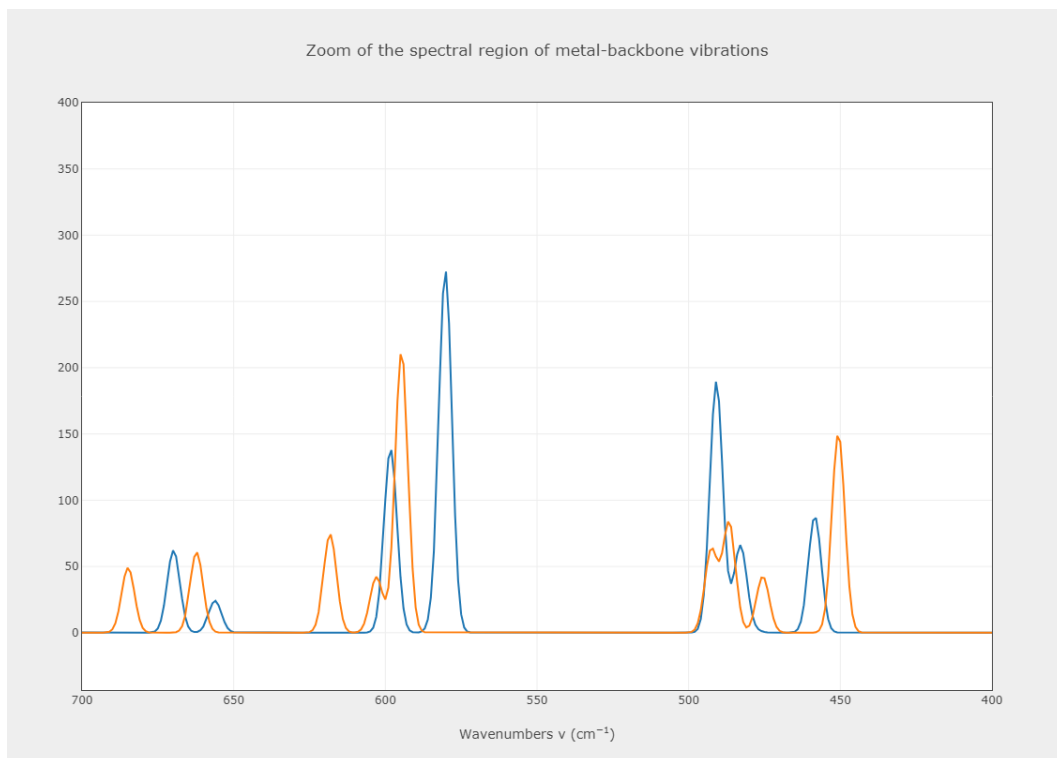


## Simulated vibrational spectra

To show the capabilities of CRYSPLOT in plotting vibrational spectra, we compare the two phases of the bi-stable MOF known as MIL-53-Al.<sup>15-16</sup> The MIL-53 family of MOFs shows a flexible framework that can undertake a phase transition driven by external stimuli as temperature, pressure and gas adsorption.<sup>17</sup> The structure of MIL-53-Al consists of infinite chains of corner-sharing  $\text{AlO}_4(\text{OH})_2$  octahedra linked through the organic ligands (i.e. 1,4-benzendicarboxylic acid, empirical formula:  $\text{C}_8\text{H}_5\text{AlO}_5$ ) to form flexible one-dimensional channels.<sup>15</sup> For MIL-53-Al, it was shown that, by changing the temperature, a phase transition occurs between a low-T narrow-pore (NP) structure and a high-T large-pore (LP) phase.<sup>15-16</sup>



*Figure 2-9. Comparison between IR spectra of the NP (blue) and LP (orange) phases of MIL-53-Al.*



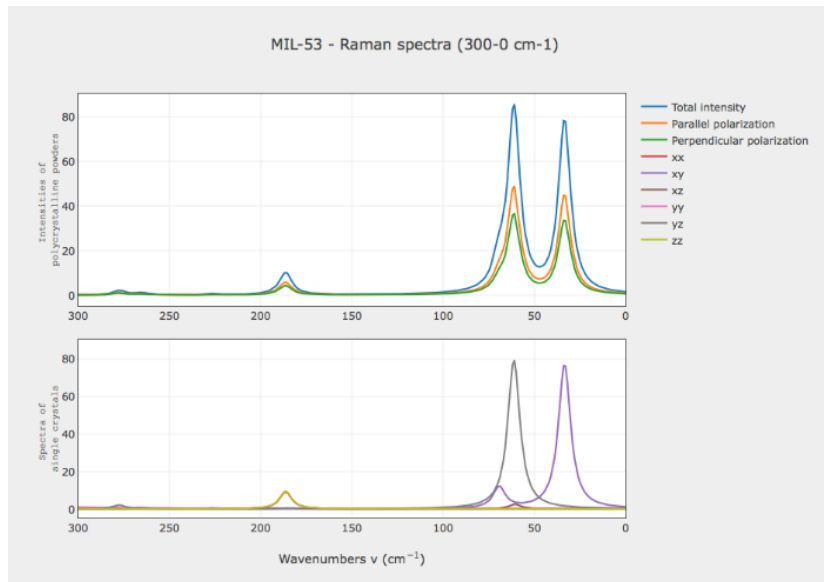
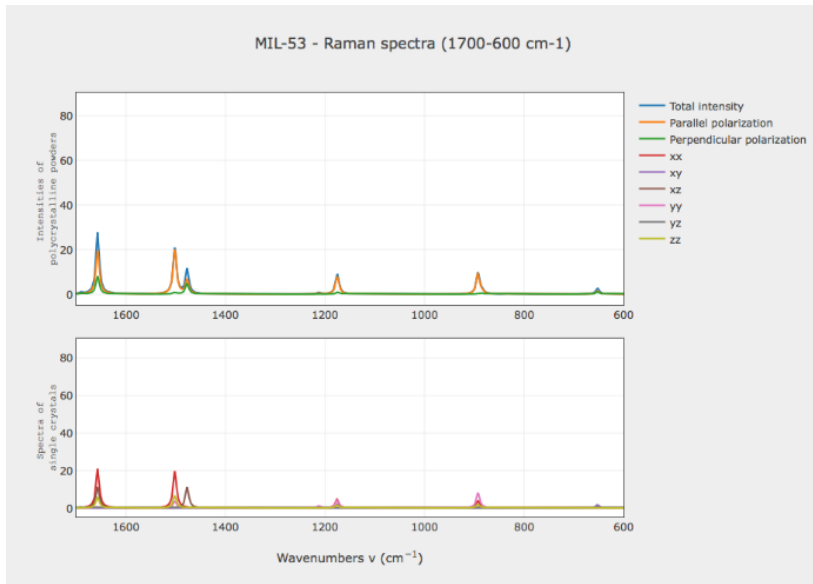
*Figure 2-10. Details of the spectral region between 400 and 700  $\text{cm}^{-1}$  of the IR spectra of the NP (blue) and LP (orange) phases of MIL-53-Al.*

It has been recently shown that the two phases exhibit unique vibrational fingerprints from dielectric function spectra<sup>18</sup> and IR/Raman spectra<sup>19</sup>. With CRYSPLOT simulated IR and Raman spectra as computed by CRYSTAL can be easily visualised. In addition, up to five IR (or Raman) spectra can be plotted together to highlight differences in the vibrational modes and distinguish between different phases. This is very useful when comparing vibrational spectra of different polymorphs as for molecular crystals, or different phases as for the case of MIL-53-Al.

In Figure 2-9 and 2-10 the comparison between the simulated IR spectra, respectively, of the NP and CP phases of MIL-53-Al is shown. Results have been obtained with the B3LYP functional<sup>20-22</sup> augmented with the Grimme's D3 dispersion correction<sup>23-24</sup> in combination with a triple-zeta quality basis set<sup>25</sup>. It can be clearly

seen that the two phases show several spectral shifts of the peaks from a few to twenty wavenumbers. In particular, by zooming into the plot with CRYSPLOT one can also visualise in more detail the spectral regions that show the largest variations because they can reveal the presence of vibrational fingerprints. For instance, in Figure 2-10 the IR region between 400 and 700  $\text{cm}^{-1}$  of the two phases is highlighted. According to a recent work by Hoffman *et al.*<sup>6</sup> this corresponds to the region of metal-oxide backbone vibrations that are more influenced by the phase transition.

As an additional example, the simulated Raman spectra of MIL-53 large pore phase are shown in Figure 2-11. CRYSTAL permits to simulate the Raman spectra of solids either as a polycrystalline powder or a single crystal<sup>26-27</sup>. In the former case, the total spectrum is predicted along with two components in the parallel and perpendicular directions. For the latter, the six spectra originate from the corresponding independent components of the second-order electric susceptibility tensor (i.e.  $xx, xy, xz, yy, yz, zz$ ). For instance, Figure 2-11 has been obtained by using the unscaled vibrational frequencies and a Lorentzian profile with a FWHM of 8  $\text{cm}^{-1}$ . For this kind of plot, CRYSPLOT permits to visualise all spectra simultaneously. A nice feature is that by clicking on the items in the legend, one can hide (or show) the corresponding spectrum in the plot. This option allows users to analyse the data and highlight the Raman spectrum along a given direction. From Figure 2-11 (bottom), it is evident that the two peaks at very low frequency in the THz region (i.e. 300-0  $\text{cm}^{-1}$ ) are polarized in different directions, i.e.  $xy$  and  $xz$ , thus making easier the comparison with single-crystal Raman spectra from experimental works. On passing, it is worthy to note that the simulated total Raman spectrum of MIL-53(LP) as a crystalline powder (see Figure 2-11, top) nicely agrees with the experimental one reported by Hoffman *et al.*<sup>6</sup>



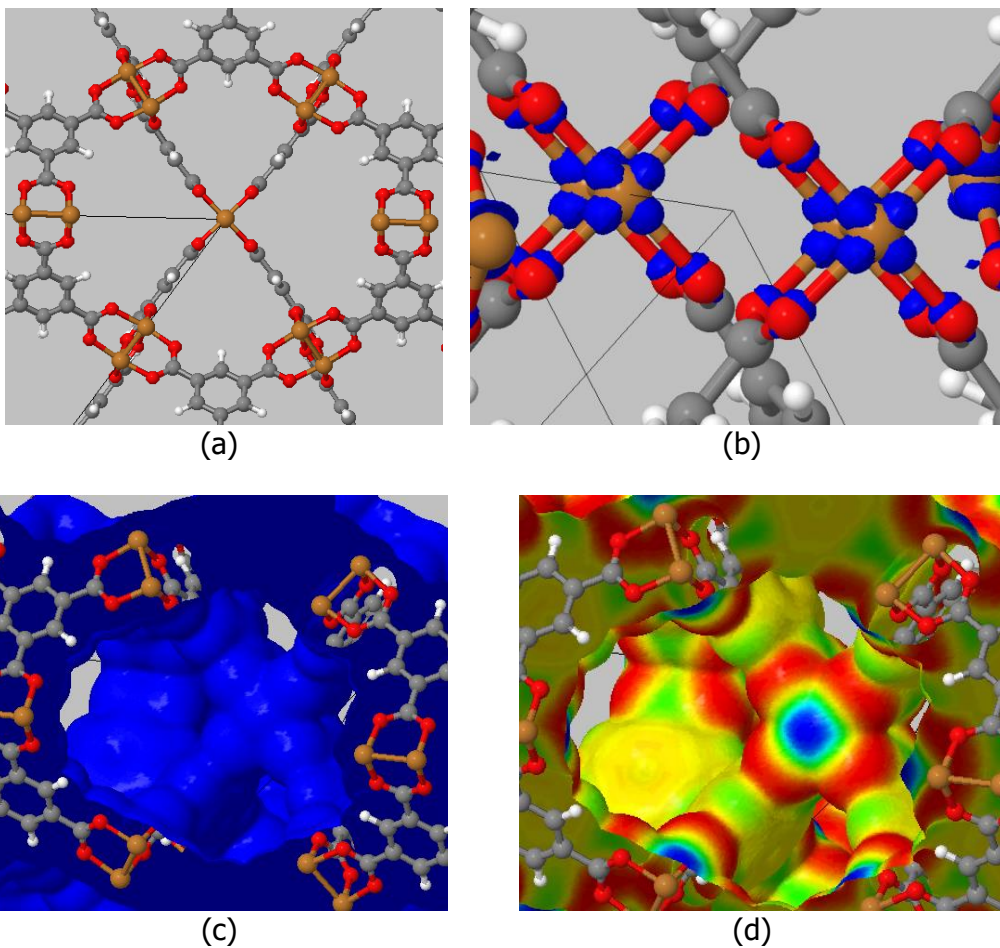
*Figure 2-11. Raman spectra of the LP phase of MIL-53-Al in the region (top) 1700-600  $\text{cm}^{-1}$  and (bottom) 300-0  $\text{cm}^{-1}$ . The two graphs show the simulated Raman spectra as for a polycrystalline material (3 components) and a single crystal (6 components) respectively. See text for details.*

## Analysis of volumetric data

Very recently, the visualisation capabilities of CRYPLOT has been extended by including the plotting of volumetric data through JSmol. This is a very useful option that allows users to visualise properties that can be represented in 3D plots such as isosurfaces. CRYSTAL permits to compute the total electron charge density, the electron spin density and the electrostatic potential on a regular 3D grid of points in the unit cell. Data are then written in a cube format that can be read by the JSmol plug-in.

For example, Figure 2-12 shows volumetric data as computed with CRYSTAL for metal-organic framework HKUST-1. HKUST-1 is comprised of Cu-containing paddle-wheels linked by trimesate ligands (i.e. empirical formula:  $C_{18}H_6Cu_3O_{12}$ , see Figure 2-12 (a)) to form a porous structure with large pores that make the compound a promising material for its adsorption properties. The Cu atom shows a square planar coordination and possesses an unpaired electron. Reported results correspond to the ferromagnetic (open-shell triplet) cubic geometry. As clearly seen from Figure 2-12 (b) that shows the spin density around the copper atoms, the unpaired electrons are located in a  $d(x^2-y^2)$  orbital of Cu with some spin polarization of the oxygens of the paddle-wheel.

Figure 2-12 (c) and (d) show the total electron charge density and electrostatic potential mapped on top of a density isosurface of  $0.003 e$  around the pore of HKUST-1. In particular, the latter allows highlighting positive and negative regions that show where possible adsorption sites can be located. This is a very useful piece of information for microporous materials as MOFs. For HKUST-1, a positive region can be clearly seen around the copper atom that, not unexpectedly, acts as a specific adsorption site to molecules with electron pairs as water or ammonia, while the oxygens around it are negatively charged.



*Figure 2-12. Examples of plotting of volumetric data for HKUST-1: (a) structure of HKUST-1, (b) spin density around the copper atoms in the paddle-wheel, (c) total electron charge density and (d) electrostatic potential mapped on top of a density isosurface of 0.003 e around the pore of HKUST-1 (blue and red correspond to positive and negative values).*

## Bibliography

1. G. Beata, G. Perego, B. Civalleri "CRYSPLOT: a new tool to visualise physical and chemical properties of molecules, polymers, surfaces and crystalline solids", *J. Comput. Chem.*, **2019**, Volume 40, Issue 26, 2329-2338.
2. R. Dovesi, V. R. Saunders, C. Roetti, R. Orlando, C. M. Zicovich-Wilson, F. Pascale, B. Civalleri, K. Doll, N. M. Harrison, I. J. Bush, P. D'Arco, M. Llunell, M. Causà, Y. Noël, L. Maschio, A. Erba, M. Rerat and S. Casassa, *CRYSTAL17 User's Manual*; University of Torino, Torino, **2017**
3. Plotly Technologies Inc. Collaborative data science, Montréal, QC **2015**  
<https://plot.ly>
4. Jmol: an open-source Java viewer for chemical structures in 3D.  
<http://www.jmol.org/>
5. R. Gaillac, P. Pullumbi and F.-X. Coudert, *J. Phys. Condens. Matter* **2016**, 28, 275201
6. J.R. Long, O.M. Yaghi, *Chem. Soc. Rev.* 38 (2009) 1213-1214
7. H. Furukawa, K.E. Cordova, M. O'Keeffe, O.M. Yaghi, *Science*, **2013**, 341, 1230444
8. J.C. Tan, B. Civalleri, *CrystEngComm*, **2015**, 17, 197
9. G. Maurin, C. Serre, A. Cooper, G. Férey, *Chem. Soc. Rev.* **2017**, 46, 3104–3107
10. Rosi, N. L.; Kim, J.; Eddaoudi, M.; Chen, B.; O'Keeffe, M.; Yaghi, O. M. J. *Am. Chem. Soc.* 2005, 127, 1504–1518.
11. P.D.C. Dietzel, P. D. C.; Panella, B.; Hirscher, M.; Blom, R.; Fjellvag, H. *Chem. Commun.* 2006, 959–961.
12. Y. Zhao, D.G. Truhlar, *Theor. Chem. Acc.* **2008**, 120, 215-241
13. M. F. Peintinger, D. Vilela Oliveira, and T. Bredow, *J. Comput. Chem.* **2013**, 34, 451-459

14. L. Valenzano, B. Civalleri, K. Sillar, J. Sauer, *J. Phys. Chem. C* **2011**, 115, 21777–21784
15. Y. Liu, J.-H. Her, A. Dailly, A. J. Ramirez-Cuesta, D. A. Neumann, C. M. Brown, *J. Am. Chem. Soc.* **2008**, 130, 11813-11818
16. A. M. Walker, B. Civalleri, B. Slater, C. Mellot-Draznieks, F. Corà, C. M. Zicovich-Wilson, G. Ramon-Perez, J. M. Soler, J. D. Gale *Angew. Chem. Int. Ed.* **2010**, 49, 7501 –7503
17. C. Serre, S. Bourrelly, A. Vimont, N. A. Ramsahye, G. Maurin, P. L. Llewellyn, M. Daturi, Y. Filinchuk, O. Leynaud, P. Barnes, G. Férey, *Adv. Mater.* **2007**, 19, 2246 – 2251
18. K. Titov, Z. Zeng, M. R. Ryder, A.K. Chaudhari, B. Civalleri, C.S. Kelley, M.D. Frogley, G. Cinque, J.C. Tan, *J. Phys. Chem. Lett.*, **2017**, 8, 5035
19. A. E. J. Hoffman, L. Vanduyfhuys, I. Nevjestic', J. Wieme, S. M. J. Rogge, H. Depauw, P. Van Der Voort, H. Vrielinck, V. Van Speybroeck, *J. Phys. Chem. C* **2018**, 122, 2734–2746
20. A. D. Becke, *J. Chem. Phys.* **1993**, 98, 5648–5652
21. C. Lee, W. Yang, R. G. Parr, *Phys. Rev. B* **1988**, 37, 785–789.
22. B. Miehlich, A. Savin, H. Stoll, H. Preuss, *Chem. Phys. Lett.* **1989**, 157, 200–206.
23. S. Grimme, J. Antony, S. Ehrlich, and H. Krieg, *J. Chem. Phys.* **2010**, 132, 154104
24. S. Grimme, S. Ehrlich, and L. Goerigk, *J. Comput. Chem.* **2011**, 32, 1456-1465
25. A. Schäfer, H. Horn, R. Ahlrichs, *J. Chem. Phys.* 1992, 97, 2571.
26. L. Maschio, B. Kirtman, M. Rerat, R. Orlando, R. Dovesi, *J. Chem. Phys.* **2013**, 139, 164101
27. L. Maschio, B. Kirtman, M. Rerat, R. Orlando, R. Dovesi, *J. Chem. Phys.* **2013**, 139, 164102



### 3 CRYSTAL VLab

CRYSTAL VLab (CRYSTAL Virtual Laboratory) is an on-going project born from the idea of giving to CRYSTAL users an easy to use graphical interface to make more gradual CRYSTAL learning curve. For instance, when a user starts playing with CRYSTAL, the first hurdle to overcome is the creation of the input. This is a very difficult step to learn because there is a standard format for the input files that requires a precise order of the information supplied to the code through specific keywords. Therefore, finding a way to simplify the use of CRYSTAL is one of the main purposes of the CRYSTAL VLab graphical interface that has been developed during this PhD work.

To better understand the structure of VLab, the next section is devoted to a short description of the main features of the CRYSTAL code and the input format.

#### **CRYSTAL code in short**

The CRYSTAL program has been jointly developed by The Theoretical Chemistry Group at the University of Torino (Italy) and the Computational Materials Science Group at Daresbury Laboratory (U.K.).

The CRYSTAL package performs ab initio calculations of the ground state energy, electronic wave function and properties of periodic systems. Hartree-Fock or Kohn-Sham Hamiltonians (that adopt an Exchange-Correlation potential following the postulates of Density-Functional theory) can be used. Systems periodic in 0 (molecules, 0D), 1 (polymers, 1D), 2 (slabs, 2D), and 3 dimensions (crystals, 3D) are treated on an equal footing. In each case the fundamental approximation made is the expansion of the single particle wave functions ('Crystalline Orbital', CO) as a linear combination of Bloch functions (BF) defined in terms of local functions (hereafter indicated as 'Atomic Orbitals', AOs). The local functions are, in turn, linear combinations of Gaussian type functions (GTF), whose exponents and coefficients are defined by input. Functions of  $s$ ,  $p$ , and  $d$  symmetry can be used.

Also available are *sp* shells (*s* and *p* shells, sharing the same set of exponents). The use of *sp* shells can give rise to considerable savings in CPU time.

The program can automatically handle space symmetry: 230 space groups, 80 layer groups, 99 rod groups, 45 point groups are available. In the case of polymers it can also treat helical structures (translation followed by a rotation around the periodic axis). Point symmetries compatible with translation symmetry are provided for molecules.

Input tools allow the generation of slabs (2D system) or clusters (0D system) from a 3D crystalline structure, the elastic distortion of the lattice, the creation of a super cell with a defect and a large variety of structure editing.

Previous releases of the software in 1988 (CRYSTAL88), 1992 (CRYSTAL92), 1996 (CRYSTAL95), 1998 (CRYSTAL98), 2003 (CRYSTAL03), 2009 (CRYSTAL09) and 2014 (CRYSTAL14) have been used in a wide variety of research with notable applications in studies of stability of minerals, oxide surface chemistry, and defects in ionic materials. The current version of the code (CRYSTAL17) has been released in Autumn 2017.

CRYSTAL performs two tasks:

program	task
<b><i>crystal</i></b>	wave function and energy calculation, along with other types of calculations (e.g. geometry optimization)
<b><i>properties</i></b>	wave function analysis and one electron properties calculation

## Wave function calculation

The input deck for wave function calculation, an ASCII text file usually denoted with the extension \*.d12, is read by the program **crystal**. The CRYSTAL input to **crystal** includes a title and three sections.

Each section consists of keywords (cases insensitive, written left justified) and numerical parameters (free format). Every section ends with the keyword END (mandatory: 3 characters only are interpreted, any ending is allowed, ENDgeom, ENDbas, etc) or STOP. The latter will cause immediate termination of execution.

The input deck has the following structure:

	Title
input <b>block 1</b>	<b>Geometry input</b> standard geometry input <i>optional geometry optimization and editing keywords</i> <b>END</b>
input <b>block 2</b>	<b>Basis set input</b> standard basis set input <i>optional basis set related keywords</i> <b>END</b>
input <b>block 3</b>	<b>Single particle Hamiltonian</b> and <b>SCF control</b> <b>SHRINK</b> sampling in reciprocal space (for 1D-2D-3D systems only) <i>optional general information and SCF related keywords</i> <b>END</b>

Some parts of the input are mandatory such as the geometry of the studied system, the basis set and the sampling of the reciprocal space for periodic systems (i.e. SHRINK).

Here is a commented example for MgO bulk:

Section	CRYSTAL input	Description
<b>0) Title</b>	MgO bulk	Title
<b>1) Geometry Input</b>	CRYSTAL	Dimensionality of the system
	0 0 0	Crystallographic information (3D only)
	225	Space Group number
	4.21	Lattice parameter(s) (Angstrom)
	2	Number of non equivalent atoms
	12 0. 0. 0.	Conventional atomic number and fractional coordinates of the atoms
	8 0.5 0.5 0.5	
	<i>Optional keywords</i>	
	END	End of the geometry input section
	<b>2) Basis set Input</b>	12 3
1 0 3 2. 0.		Basis set input: code, type, nr. of primitives, formal charge and scale factor of the gaussian exponents in the shell
1 1 3 8. 0.		
1 1 3 2. 0.		
8 2		Oxygen basis set: 2 shells
1 0 3 2. 0.		1:STO-nG; 0,s shell;3 primitives,2 elec; standard
1 1 3 6. 0.		Pople scale factor
99 0		99: end of basis set definition
<i>Optional keywords</i>		
END		End of the basis set input section
<b>3) Hamiltonian, computational parameters and SCF input</b>	<i>Optional keywords</i>	Default choice: Restricted Hartree Fock
	SHRINK	Keyword to specify shrinking factors
	8 8	Reciprocal space integration parameters
	<i>Optional keywords</i>	
	END	End of the SCF input section

As it can be seen, many optional keywords can be specified that allow users to modify the initial geometry and run different types of calculations (e.g. geometry optimization, vibrational frequencies calculations, elastic constants, piezoelectric tensor calculations, ...). Most of them represents a block opened by a given keyword (e.g. OPTGEOM) and closed by END. Moreover, many additional options can be specified as sub-keywords to control/modify (e.g. convergence criteria) the default set up of the main keyword.

As mentioned above, some of them must follow a given order and the correct syntax. It is then clear that an automatic way to select and specify the diverse keywords would be very helpful.

## **Wave function analysis**

The program **properties** reads an input deck, usually denoted with the extension \*.d3, to perform the analysis of the wave function as computed by **crystal**. Again, the input is an ASCII text file as for **crystal**, but it does not comprise different sections. Yet, each option (e.g. band structure, density of states, ...) requires a specific input with several keywords and parameters.

It is out of the scope of the present PhD thesis to enter in more details into the CRYSTAL input format, additional information on input decks for both **crystal** and **properties** can be found on the CRYSTAL Users' Manual and at the CRYSTAL Tutorials web page.

On passing, it is worthy to mention that the CRYSTAL tutorials web site has been renewed and restyled during this PhD work.

## **CRYSTAL VLab structure**

CRYSTAL VLab is basically divided in three parts that represent the main steps that a user takes when using the CRYSTAL code, that is create or edit an input file, run the calculation and visualise the results:

- *Build* : in this part the user can write his input file (.d12 format or .d3 one) either through a simple text area or with the help of a wizard;
- *Compute* : in this part the user can run a calculation. This task is accomplished by means of an auxiliary program (CRYSTAL VLab starter) previously installed on user's PC (see below for more details);
- *View* : in this part the user can visualise computed results either with CRYSPLOT<sup>1</sup> or with JSmol<sup>2</sup>.



Figure 3-1. Screenshot of CRYSTAL VLab homepage

In addition, for registered users, along with the graphical interface, a file repository has been created on a host server in order to store their own files to be used with CRYSTAL VLab. To do that a *file manager* application has been developed to manage users' private profiles.

In the following sections, the main features of the three parts will be presented along with the two auxiliary applications, namely: CRYSTAL VLab starter and CRYSTAL VLab file manager.

## CRYSTAL VLab: BUILD

The "BUILD" part of CRYSTAL VLab manages the CRYSTAL input. It allows one to read, edit and create a new input. It is the most complex part of the GUI because its development required an extensive work to analyze the structure of the CRYSTAL input file, parse the whole input by taking into account that there could be a combination of multiple keywords and put them in the right order.

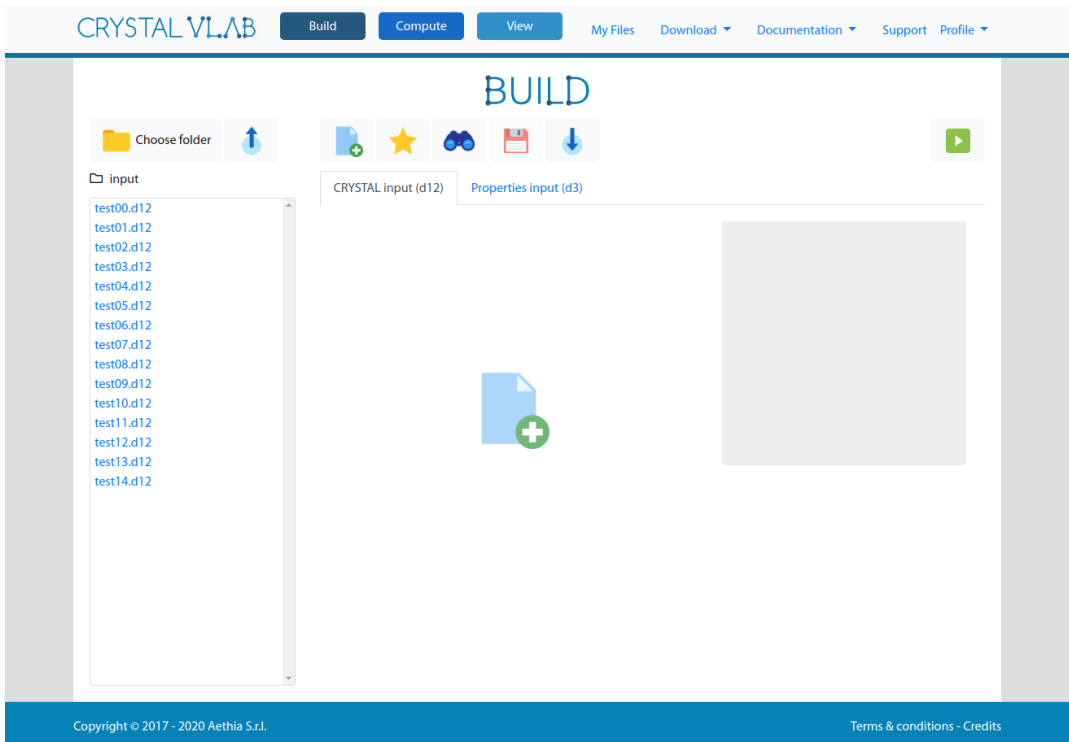






Figure 3-2: Screenshot of CRYSTAL VLab BUILD page

Graphically, the BUILD page is designed with a toolbar on top with different buttons (see Figure 3-2) whose meaning is listed below:

-  : selects a folder between user private folders. After the selection, all the files in folder appear on left.
-  : uploads new files in the selected folder.
-  : writes a new input through a simple text area.
-  : opens the wizard, i.e. an input assistant for a guided input creation.



: refreshes the JSmol block (grey block on the right).



: saves the input in the area below in selected folder.



: downloads the input.



: opens CRYSTAL VLab COMPUTE part, create a new job and then run calculations.

Below the toolbar, there is the heart of the page that is divided in two parts:

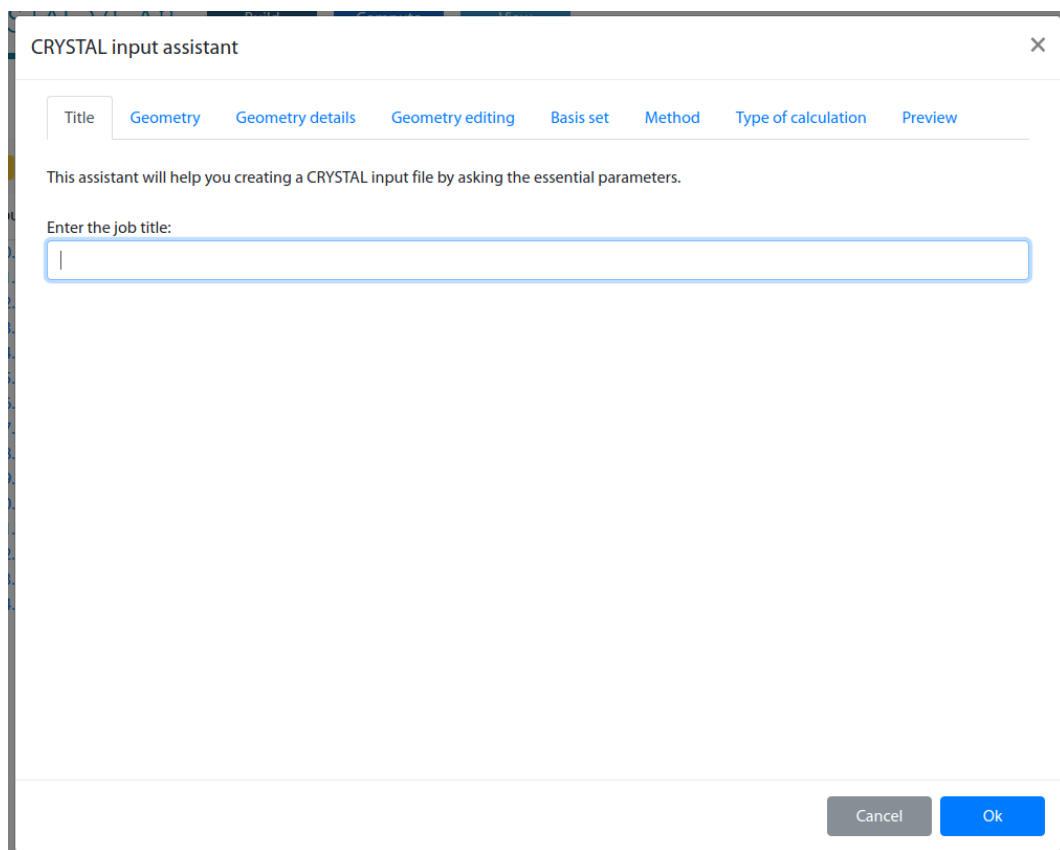
- On the left side there is a window that lists the files contained in the folder selected by the "Choose folder" button. It's important to note that all files in this area are filtered by format, in particular the .d12 and/or .d3 ones.
- In the middle and on the right side there is a text area that shows a preview of the input file and a small graphical box, respectively. In this page, the user can select between two tabs that correspond to the desired input format, that is .d12 (crystal input) and .d3. (properties input). In CRYSTAL VLab it is then possible to create different types of input, in particular .d12 files or .d3 ones, either from scratch or through a wizard. After tab selection, CRYSTAL VLab can set the page in the right way, for example by modifying the contents of the wizard, which obviously changes according to the format of the file that the user wishes to write. In the right part of this area there is a small box (in grey) dedicated to the plot of the structure as written in the input file; this graphical part is managed by JSmol.



The core of the BUILD page is based on two wizards: one dedicated to the input file for **crystal** (i.e. \*.d12 file format (see Appendix 3)) and one for **properties** (i.e. \*.d3 file format (see Appendix 4)).

In the following sections, the two wizards will be discussed by highlighting the main specific settings. Further details are given in Appendix 3 and 4.

## CRYSTAL VLab wizard for .d12 input files



*Figure 3-3. Screenshot of CRYSTAL VLab wizard for .d12 format*

The wizard to create the .d12 input is comprised of several tabs that helps a user in filling the main sections of the input deck that have been shortly described above.

In particular, they allows one to provide all needed information step by step as:

- *Title* requires to enter the title contained in the first row of .d12
- *Geometry* requires to select the structure of the system, importing a file (different formats are supported, e.g. CIF files), selecting a file contained in the personal library on CRYSTAL VLab or selecting a file fort.34.
- *Geometry details* permits to select the dimensionality and lattice parameters of the system, allowing to check the geometry through a quick TESTGEOM calculation that is run on a computer server-side.
- *Geometry editing* requires selecting particular modification of the initial geometry. For example, when a user wants to create a supercell (SUPERCELL keyword) or generate a slab model (SLABCUT).
- *Basis set* permits to specify the basis set of the system. It can be chosen from the predefined basis sets or taken from a library of basis sets for each element of the system. In CRYSTAL VLab, indeed, there exists a database of basis sets divided by element, which corresponds to the basis sets that can be found at the CRYSTAL website.
- *Method* selects the type of Hamiltonian (i.e. Hartree-Fock or Density Functional Theory - DFT) for the calculation, the electron occupancy as for open or closed shell systems and other computational parameters related to the selected method.
- *Type of calculation* allows specifying the type of calculation to be run. In particular, the main options can be set for Geometry optimization (OPTGEOM), Vibrational frequencies calculation (FREQCALC), Coupled Perturbed Hartree-Fock/Kohn-Sham method (CPHF), Equation of state (EOS), Elastic constants calculation (ELASTCON), Piezoelectric tensor (PIEZOCON), Piezoelectric calculation via CPHF/KS (PIEZOCPC), Piezoelectric strain (ELAPIEZO), Photoelastic tensor (PHOTOELA), Quasi-harmonic calculation (QHA) and Anharmonic calculation of frequencies of X-H (X-D) bond stretching (ANHARM). An important feature of the *Type of calculation* tab is that for each option, keywords are subdivided in two sets: Basic and

Advanced. By default, when selecting an option, users can only see Basic keywords (i.e. a selection of the most frequently used keywords), but if one then clicks on the "Advanced options" button, a list of advanced keywords appears. For instance, Figure 3-4 and 3-5 show the tab for the OPTGEOM option with basic keywords (Figure 3-4) and advanced ones (Figure 3-5), respectively.

- *Preview* shows then the current the input file written in the CRYSTAL format.

When the user ends the creation of the input, he/she can see and check the full input in the *Preview* tab and then click on the "Ok" button located in the wizard footer. At that point, a function checks the correctness of the specified data and options. If everything ends correctly, the wizard closes and the final input deck appears at the center of the page. In addition, in the small window on the right of the page, the structure of the system appears for further checking. Instead, if the control function returns an error, a warning message appears with the list errors, the wizard does not close and waits for corrections.

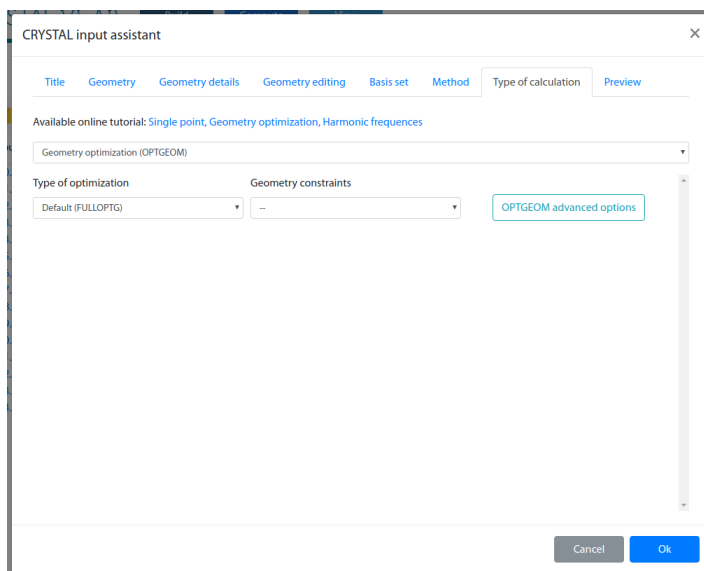


Figure 3-4. Basic keywords of OPTGEOM calculation

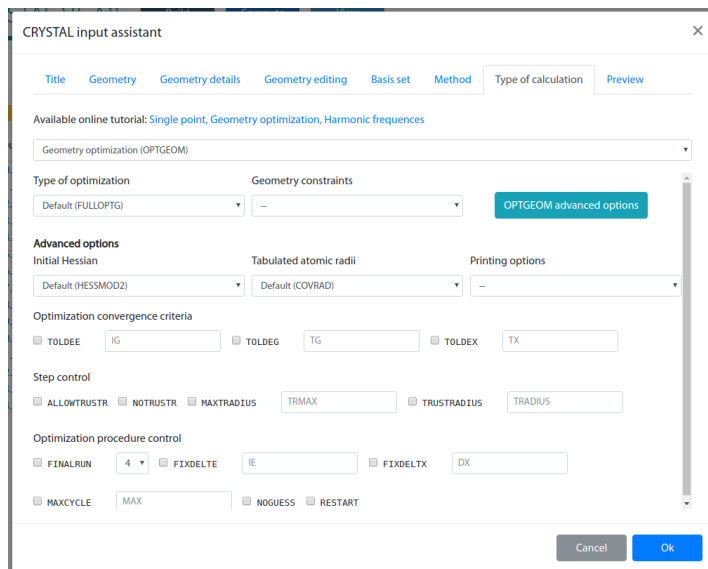


Figure 3-5. Advanced keywords for OPTGEOM calculation

## CRYSTAL VLab wizard for properties calculations

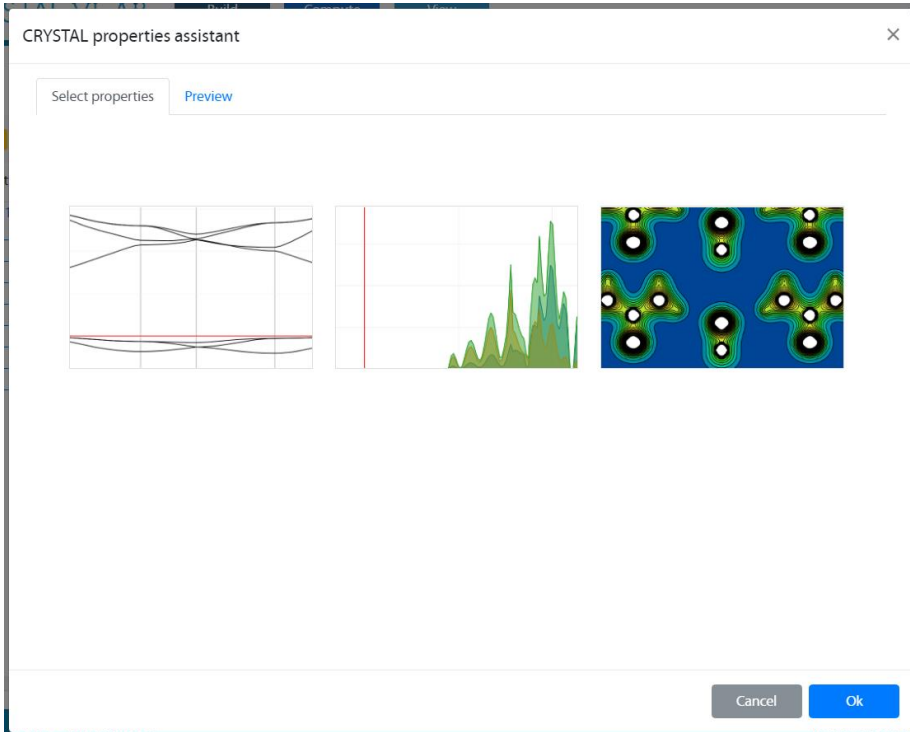


Figure 3-6. Screenshot of CRYSTAL VLab wizard for .d3 format

As mentioned above, a different input deck, usually dubbed as .d3, is needed to compute properties related to the wave function (e.g. band structure, DOSs,...) and charge density. Calculations are carried out with the **properties** module of CRYSTAL.

Presently, the wizard for creating .d3 input files allows users to write the input for Band Structure, Density of States and Electron Charge Density Map. Work is in progress to extend the tool to other properties.

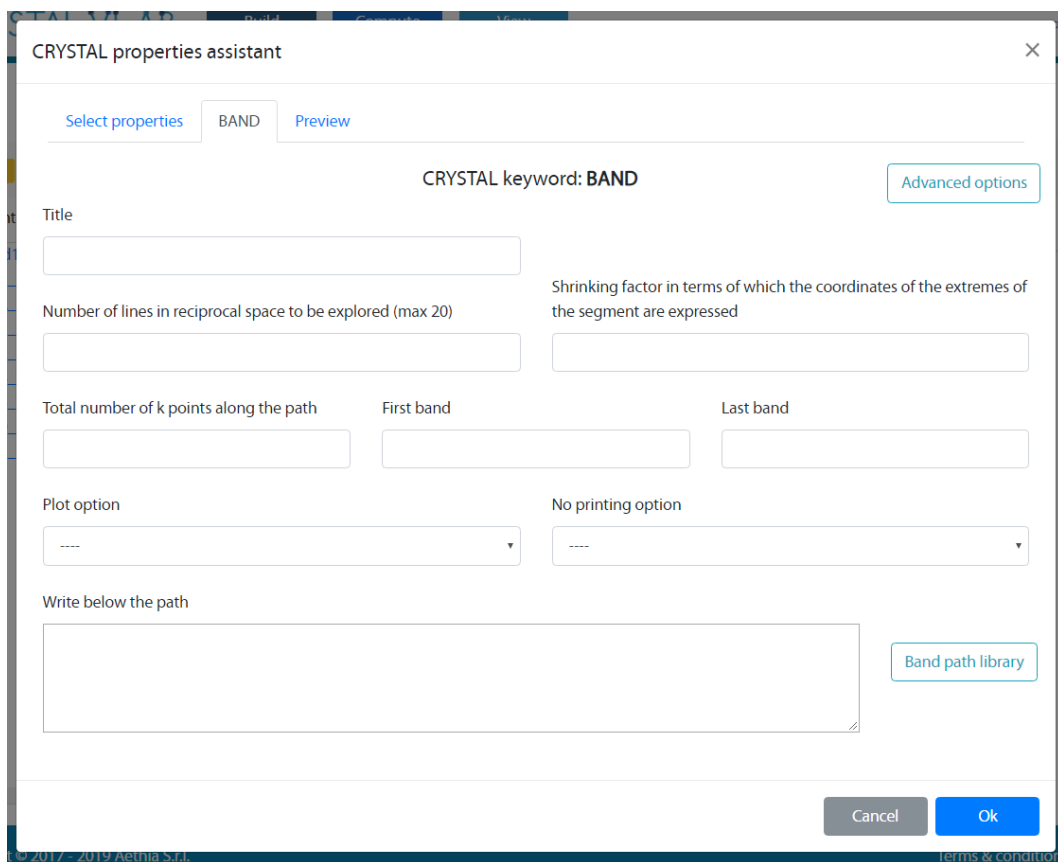


Figure 3-7. Screenshot of wizard for BAND keyword

The wizard is comprised of two tabs called "Select properties" and "Preview" (Figure 3-6). In the first one, the user must click on the image corresponding to the property he/she wants to calculate; it is important to note that he/she can select more than one property. After this selection, as many tabs appear as the number of selected properties; indeed there are dedicated tabs for every property. At the present stage of the development of this part there is a tab for band structure, density of states and electron charge density maps. After filling all required fields, in the "Preview" tab the .d3 input file is written. If everything has been properly set the selected properties are ready to be computed with CRYSTAL.

Graphically, the wizard is then composed by fixed tabs (“Select properties” and “Preview”) and dynamic tabs (“BAND”, “DOSS”, “ECHG”) with the latter appearing only when needed.

For better understanding the use of the wizard, let’s discuss the case of the input for the calculation of band structure. Figure 3-7 shows the fields required for the keyword BAND. For some of them the user has to specify values related to computational parameters of the algorithm implemented in the code and/or to the systems under study, while for other information he/she can select them in a predefined sets of options.

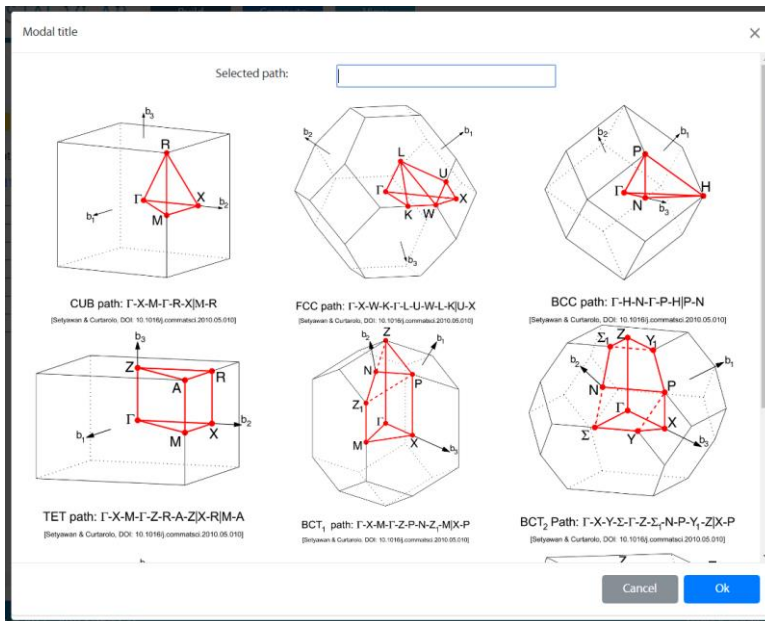


Figure 3-8. Screenshot of predefined paths library for different Brillouin Zones

For instance, at the bottom of the wizard, one has to specify the path within the Brillouin zone for which the code computes the corresponding eigenvalues (i.e. bands). In this case, there are two ways to do it. First, the user can decide to write the input directly in the area using the CRYSTAL format and syntax, that is by specifying the coordinates of the k-points in terms of the shrinking factors defined in the corresponding field (standard route). Second, the path can be chosen in a

graphical way by referring to a library of predefined list of k-points for the different Brillouin zones related to the corresponding Bravais lattices (see Figure 3-8). The usage of this library makes the selection very simple: every image has a label with the path and when the user clicks on an image the predefined path appears at the top of the wizard and is also inserted in the "*Preview*" tab.

### **CRYSTAL VLab input reading**

Along with the two wizards that allow users to create a new CRYSTAL input, another important capability of the GUI is to read an existing input file. This also permits experienced users to take advantage of CRYSTAL VLab for the editing of their files, run calculations and visualizing their results.

The implementation of this feature in the BUILD part has required a lot of work to be able to properly read and parse the CRYSTAL input by taking into account all of the diverse combinations of keywords and sub-keywords, formatted and unformatted input data and additional information.

As mentioned above, this is the most complex and extended part of the GUI. The reason for this lies in the implementation of a Python script that reads and parses an existing input .d12 and then pre-compiles the corresponding fields of the wizard web page with the parameters contained in the input. To do that a careful and deep analysis of each option available in the CRYSTAL code has been carried out.

In more detail, the script starts working when the user has already loaded an input file in the initial page and opens the wizard, for example to modify some parameters. The script reads the input file line by line and rearranges data in a json text format. After the file .json has been created, another piece of code reads all information from .json and controls option by option every field in the wizard and inserts the corresponding parameters and/or activates keywords. Presently, our script can translate and precompile only .d12 file and only basic keywords.



Advanced options are not yet reorganized but if present they are maintained in the input as shown in the “*Preview*” tab.

As an example, in Appendix 5, a piece of the code for this part is reported along with a translated “.json” format of a simple input file.

## CRYSTAL VLab: COMPUTE

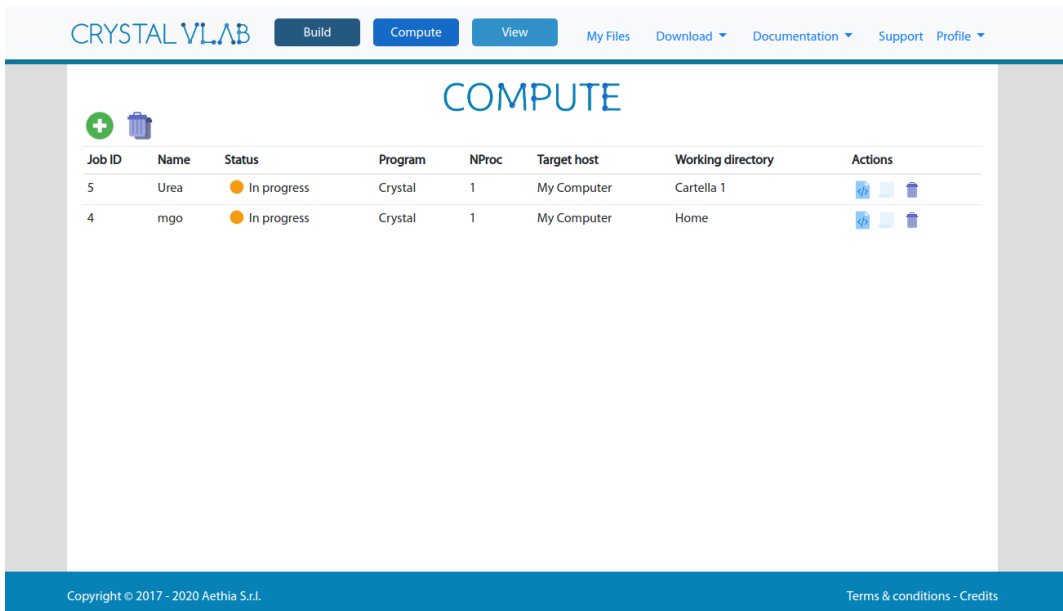


Figure 3-9. Screenshot of CRYSTAL VLab COMPUTE page

The “COMPUTE” part of CRYSTAL VLab (see Figure 3-9) allows users to run a calculation for a newly created or edited or already existing input.

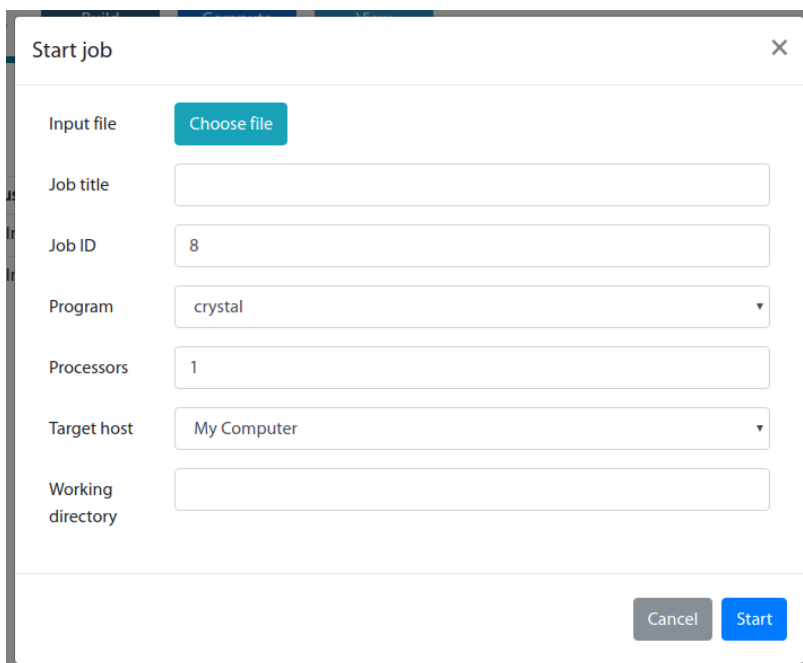
Users can submit a new job by filling a pop-up form that appears when clicking on



As shown in Figure 3-10, in the pop-up window the information required are:

- *Input file:* an input file is selected from the private file manager,
- *Job title:* a string to identify the job must be specified,
- *Job ID:* it is an incremental value that is automatically inserted,

- *Program:* requires selecting which module/version of the CRYSTAL program must be used for the calculation. It can be either a serial or a parallel version of the code: crystal, Pcrystal, MPPcrystal, properties, Pproperties.
- *Processors:* number of processors to be used for parallel calculations. It is set to 1 by default.
- *Target host:* it specifies where the job is going to run. At the moment, just one option is available: "My Computer". It means that the job will run on the local machine. Work is in progress to extend the target hosts to remote machines and cloud servers.
- *Working directory:* this is the name of the directory where the input file is stored. Note that output files, too, will be stored in that folder.




The screenshot shows a 'Start job' dialog box with the following fields and values:

Field	Value
Input file	Choose file (button)
Job title	
Job ID	8
Program	crystal
Processors	1
Target host	My Computer
Working directory	

Buttons: Cancel, Start

*Figure 3-10. Screenshot of pop-up window for a new job*

As mentioned, the current version of the "COMPUTE" part allows a user to run CRYSTAL just on the local user's PC. Therefore, when the user saves information from the pop-up form, the input file is automatically downloaded and opened with an application, dubbed as CRYSTAL VLab Starter, that runs CRYSTAL. The CRYSTAL VLab Starter is a separate program that must be installed on user's PC (see below for more details).

When a user creates an input deck by using the "BUILD" part a shortcut can be used to run immediately the job. Indeed, he/she can use the button  in the "BUILD" page to open the COMPUTE section with the pop-up partially precompiled. In particular, fields precompiled are, namely: *Input file*, *Job title* and *Working directory*.

After clicking on the "Start" button, see Figure 3-10, a new job with status "In progress" is created and a new line corresponding to the submitted job appears in the job list of the COMPUTE page (see Figure 3-9).

For every job in the table, a few information are displayed along with some buttons to manage different actions. In particular, each line shows information present in the pop-up form of Figure 3-10 ("Job ID", "Name", "Status", "Program", "NProc", "Target host" and "Working directory") and allows to:



view input file,



view output file,



archive the job to hide it from the table.

Note that the table shows only not archived jobs.

Finally, at the top of the page, near "Add new job" button, there is another button to archive all completed jobs present in the table.

It is worthy to note that currently the "COMPUTE" part of VLab is designed to run calculations only with *crystal* program (i.e. .d12 input files). In the next future the graphical interface will be extended to allow users to run the *properties* program as well.

### **Crystal VLab Starter**

Crystal VLab Starter is an auxiliary program, written in C++, that runs the CRYSTAL code within the COMPUTE part of CRYSTAL VLab. It must be installed on the local host and is available for Linux, MacOSX and Windows operating systems. Of course, a copy of the CRYSTAL code must be available on the local machine.

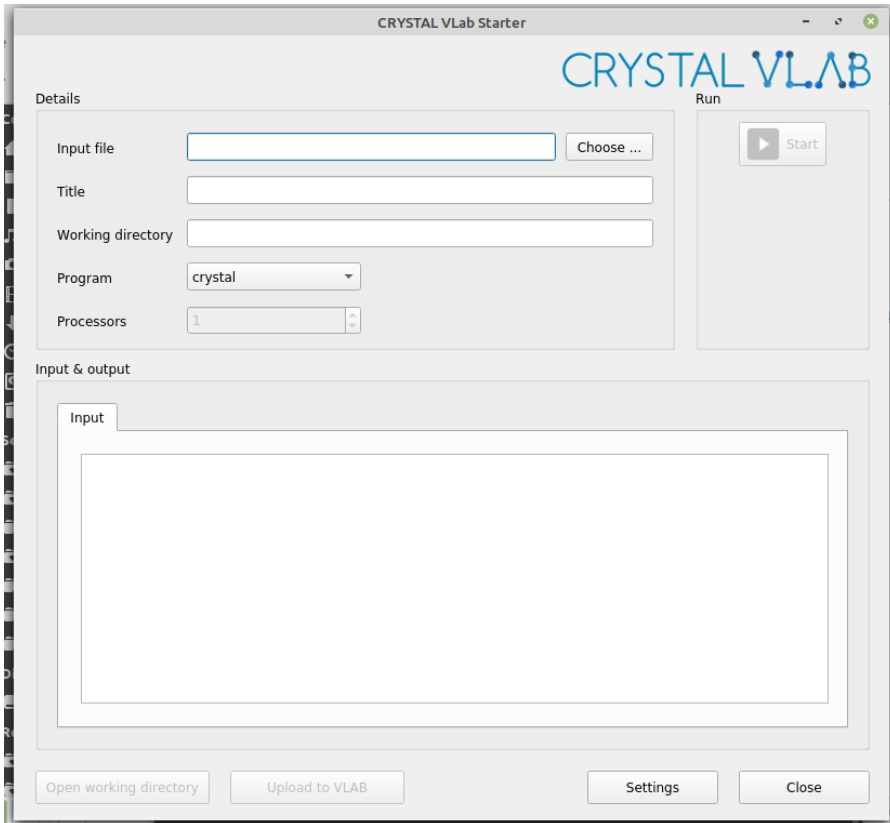
When a user decides to run a job on the local host and clicks the "Start" button (see above) a file with extension .vlabjob is created to be read by CRYSTAL VLab Starter.

During the installation of CRYSTAL VLab Starter, there is an option that selects that program as the default one to open .vlabjob files (see Appendix 6).

As shown in Figure 3-11, information required by CRYSTAL VLab Starter are:

- *Input file* : name of the input file. It can be chosen with the button "Choose...". Note that if the user opens a .vlabjob file this field is already filled in,
- *Title* : displays the title of the calculation as written in the first row of the input file,
- *Working directory* : identifies the directory where both input file and output files will be uploaded at the end of the calculation,
- *Program* specifies which version of the CRYSTAL code must be used (i.e. serial, parallel, massively parallel). This option depends on which executable files are available on the local host. During the installation of the CRYSTAL VLab Starter, users must specify where CRYSTAL binaries can be found,

- *Processors* : specifies the number of processors to be used for the calculation. By default it is set to 1. This means that the serial version of CRYSTAL is utilized. It can be adjusted to a given value when using Pcrystal or MPPcrystal parallel versions.



*Figure 3-11. Screenshot of the CRYSTAL VLab Starter*

In Figure 3-11, the screenshot of the CRYSTAL VLab Starter is shown. In the upper part of the form (Details) there are the fields described above, while in the bottom one (Input & Output) there is a preview of the input. Note that when the calculation is running, in the bottom part a new tab appears where the corresponding output file is displayed. Below the “Input & Output” area, there are some buttons for opening the working directory, contacting VLab web-interface to send obtained output, opening CRYSTAL VLab Starter’s settings and closing the application. It is

worthy to emphasize that users can in every moment terminate the execution of CRYSTAL by clicking on the “Stop” button.

When the calculation has finished, user can click “Upload output” button to automatically contact the web interface to upload output files in the user’s profile folder.

## **CRYSTAL VLab: VIEW**

The “VIEW” part of CRYSTAL VLab is dedicated to the graphical visualisation of computed results (e.g. structure and other properties as band structure, density of states, and so on).

Basically, two graphical tools are utilized: CRYSPLOT for properties visualisation and JSmol for the crystalline structure. It is important to emphasize that the version of CRYSPLOT used in this project is the same as online, indeed we have developed a fully compatible version that can work either as standalone or embedded in CRYSTAL VLab. Therefore, all properties that can be plotted with CRYSPLOT are available for visualisation (see Chapter 2).

To start the visualisation, two options are available:

- reading a file already uploaded in the user’s personal profile on CRYSTAL VLab repository (see Figure 3-12)
- start from a graphical tool, i.e. CRYSPLOT or J-ICE (see Figure 3-13)

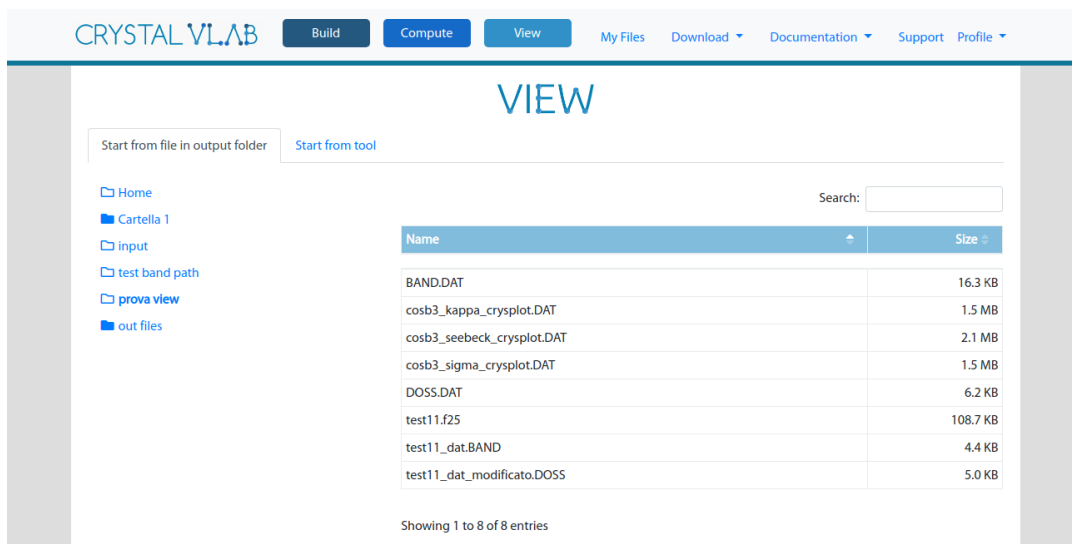


Figure 3-12. Screenshot of VIEW page, in particular "Start from file" option

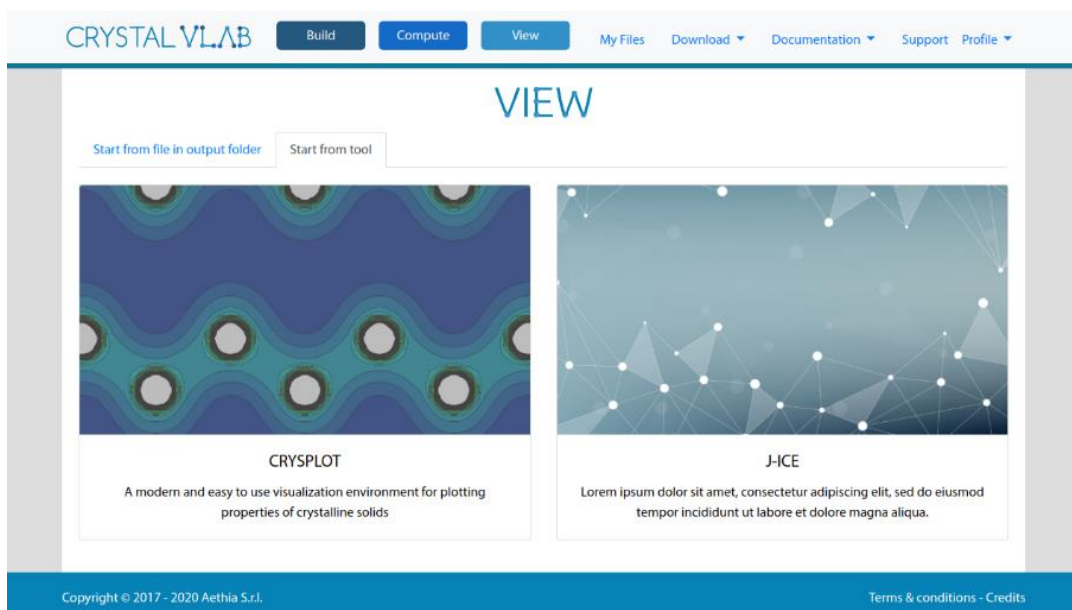
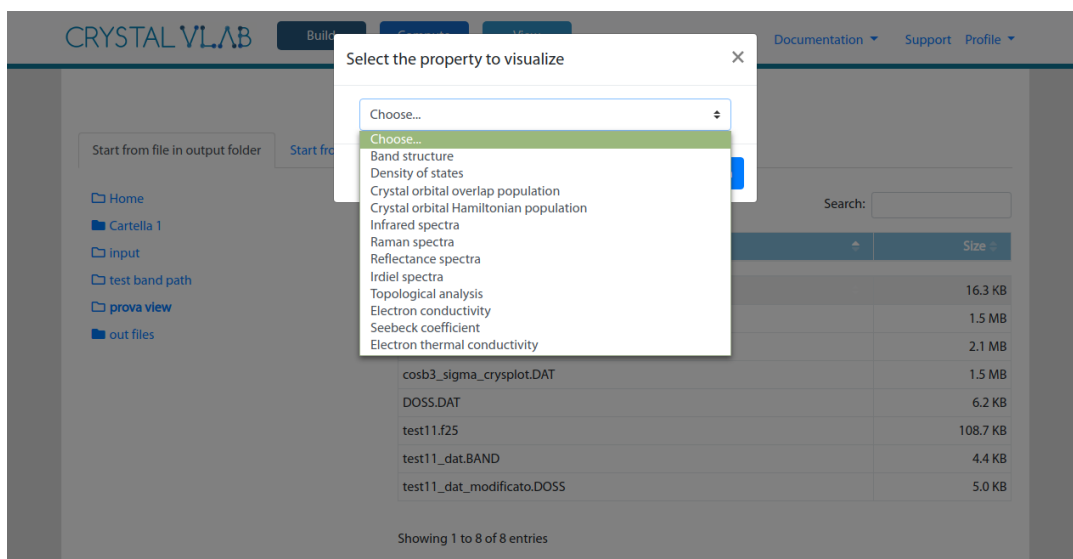


Figure 3-13. Screenshot of VIEW page, in particular "Start from tool" option

In "Start from file" case, when user clicks on the file that he/she wants to visualise, the file content is parsed and sent directly to CRYSLOT by using the mechanism detailed in section "CRYSLOT: Under the hood".

At this point two alternatives are possible, depending on file format:

- if the format is linked to a single property (e.g.: .BAND format), the file is immediately opened
- if the format is shared between different types of property files (e.g.: .f25 file), the user has to select the property he wants to visualise (see Figure 3-14)



*Figure 3-14. Screenshot of modal to select property*

Figure 3-15 shows how the CRYSTAL VLab VIEW page appears once the file has been loaded into CRYSPLOT.

It is important to note that a full-feature version of CRYSPLOT is employed. Presently, there is just a limitation in that users cannot save the plot as an image.

Figure 3-16 shows how the page appears when the file is loaded into JSmol. This option reads directly the .out file format and, at the moment, it shows the structure contained in file.



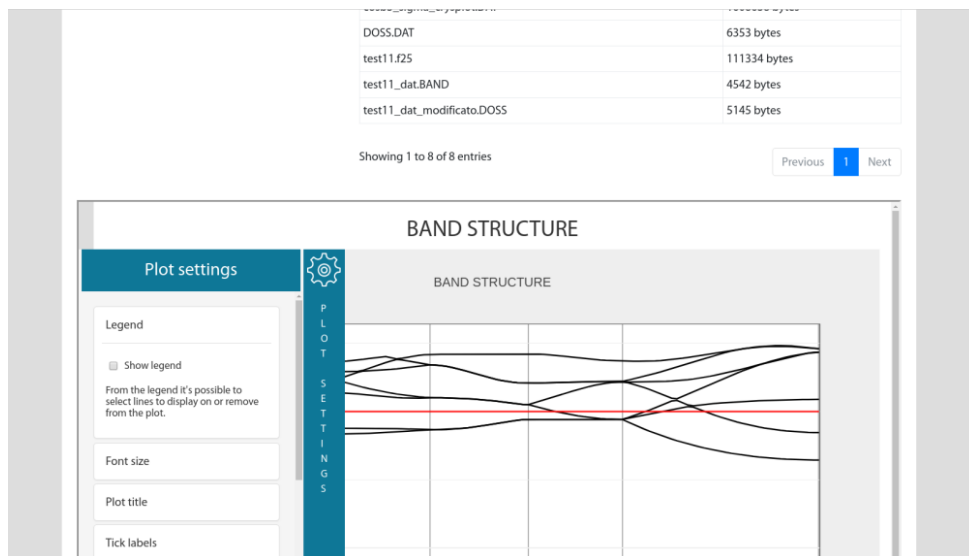


Figure 3-15. Screenshot of CRYSPLOT band page loaded in CRYSTAL VLab

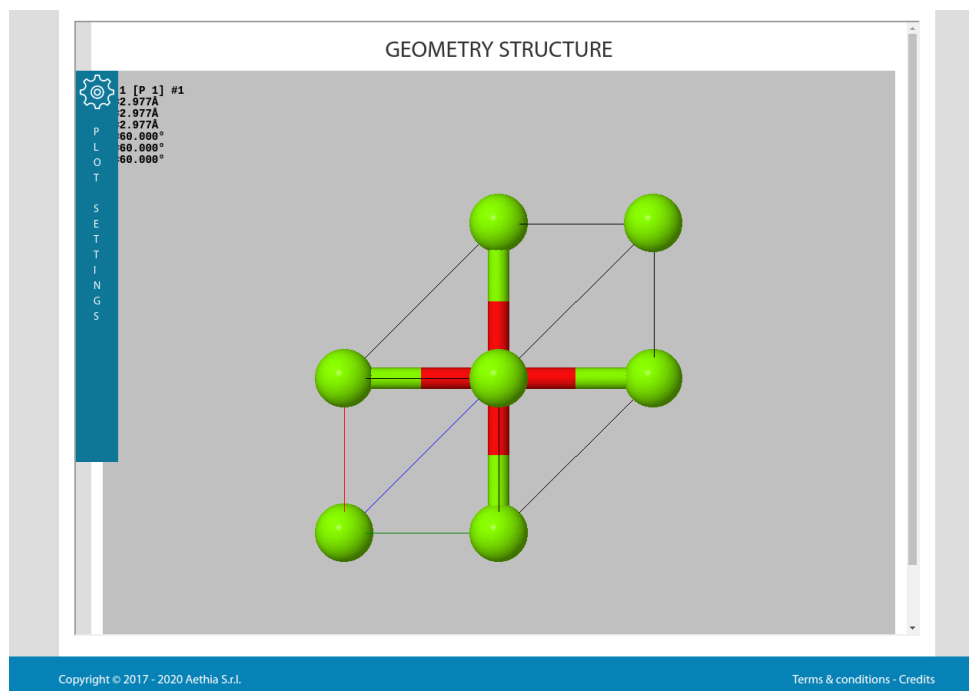


Figure 3-16. Screenshot of JSmol loaded in CRYSTAL VLab

## CRYSTAL VLab: FILE MANAGER

As mentioned before, the CRYSTAL VLab web page is also connected to a repository of inputs and outputs where registered CRYSTAL Users can save their own files. To access this private area, user must login by using the form shown in Figure 3-17.

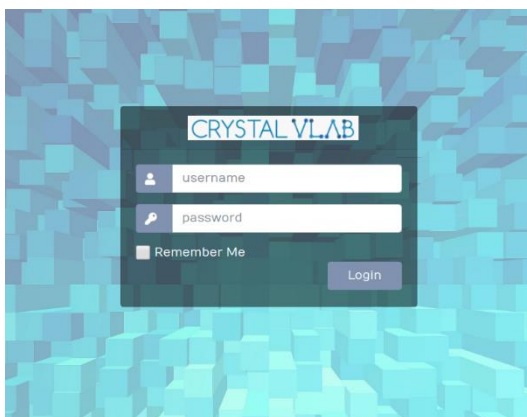


Figure 3-17. Screenshot of the CRYSTAL VLab login

After login, on the top bar of CRYSTAL VLab home page there is an item called "My Files"; which opens the user's personal file library.

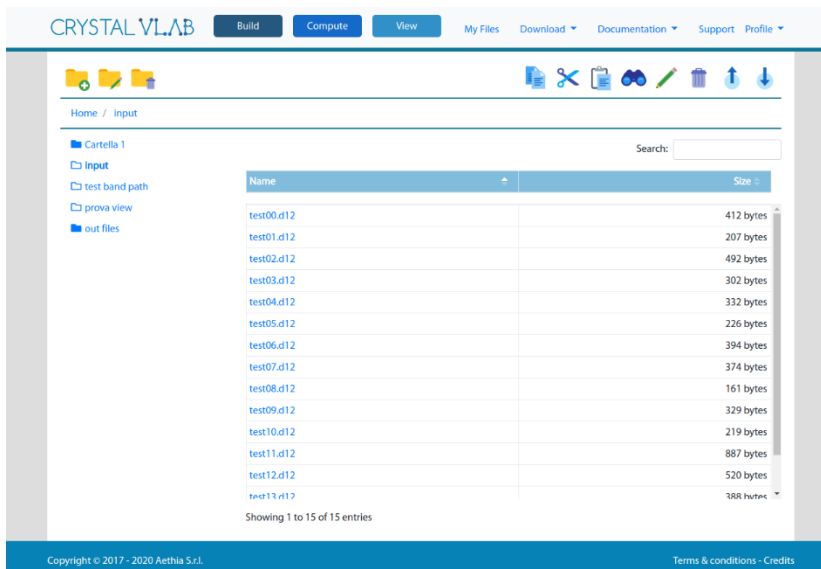


Figure 3-18. Screenshot of the CRYSTAL VLab File manager

A user can then organize its content by adding, renaming or deleting folders, and in each folder he/she can add, rename, view, download, upload or delete files. All the files are saved on a hosting server to which CRYSTAL VLab is connected.

## **CRYSTAL VLab: Under the hood**

After the discussion of the main features of CRYSTAL VLab, let's enter a bit more into the technical details of the web interface.

Basically, CRYSTAL VLab is a web-based application with an architecture made by two parts: a back-end and a front-end.

The back-end part is designed with Django<sup>3</sup> (Version 2.2.7). Django is a free and open-source web framework, written in Python, which follows the model-view-template (MVT) architectural pattern. Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusability and "pluggability" of components, less code, low coupling, rapid development, and the principle of don't repeat yourself. Django also provides an optional administrative "create, read, update and delete" interface that is generated dynamically through introspection and configured via admin models. This administrative interface is also employed for opening, adding, deleting elements in databases. By default, Django manages the databases using SQLite<sup>4</sup>, a self-contained, high-reliability, embedded, full-featured, public-domain SQL database engine (see Appendix 7 for an overview of the database structure). In the future, I expect to use another database engine for the management of databases, in particular PostgreSQL<sup>5</sup>, because it's a production-ready solid DBMS. Finally, I used the Python<sup>6</sup> programming language, in particular version 3.5.2, and JSmol for plotting. JSmol is an HTML5 JavaScript-only extension of the Java-based molecular visualisation applet Jmol.

The front-end is designed with HTML5 markup language, CSS3 style sheet language, BOOTSTRAP<sup>7</sup> front-end-framework and the JavaScript programming

language. These tools have already been used to develop CRYSPLOT, but BOOTSTRAP has been updated to version 4. This version has a more modern look & feel: for example, the so-called "cards" have been introduced, which make the page more dynamic. For the front-end, I used also JQuery, a cross-platform JavaScript library designed to simplify the client-side scripting of HTML. JQuery's syntax is designed to make it easier to navigate a document, select DOM elements, create animations and handle events. The modular approach to the JQuery library allows the creation of powerful dynamic web pages and Web applications. To query the server I used Ajax (short for "Asynchronous JavaScript And XML") that is a set of Web development techniques to create asynchronous Web applications. With Ajax, Web applications can send and retrieve data from a server asynchronously (in the background) without interfering with the display and behaviour of the existing page. By decoupling the data interchange layer from the presentation layer, Ajax allows Web pages, and by extension Web applications, to change content dynamically without the need to reload the entire page. Ajax is not a single technology, but rather a group of technologies. HTML and CSS can be used in combination to mark up and style information. The Web page can then be modified by JavaScript to dynamically display - and allow the user to interact with - the new information. The built-in XMLHttpRequest object within JavaScript is commonly used to execute Ajax on Web pages, allowing Web sites to load content onto the screen without refreshing the page. Ajax is not a new technology, or a different language, but it is a very useful to combine existing technologies in new ways.

## Bibliography

1. G. Beata, G. Perego, B. Civalleri "CRYSPLOT: a new tool to visualise physical and chemical properties of molecules, polymers, surfaces and crystalline solids", J. Comput. Chem., **2019**, Volume 40, Issue 26, 2329-2338.
2. Jmol: an open-source Java viewer for chemical structures in 3D. <http://www.jmol.org/>
3. Django core team (**2011**). Django: A Web framework for the Python programming language. Django Software Foundation, Lawrence, Kansas, U.S.A. <http://www.djangoproject.com>
4. Hipp, D. R., Kennedy, D., Mistachkin, J., (**2015**) SQLite (Version 3.8.10.2) [Computer software]. SQLite Development Team.
5. Official PostgreSQL website: <https://www.postgresql.org>
6. Python Software Foundation. Python Language Reference, version 3.5.2 Available at: <http://www.python.org>
7. Official Bootstrap website: <http://getbootstrap.com/>

## 4 Conclusions and Outlook

The main purpose of the present PhD work has been the development of graphical tools to analyse and visualise properties of crystalline solids.

Visualisation has a prominent role in intuition by transforming numbers into images, so that visualisation tools can help to see things from a new point of view, while graphical interfaces are important to facilitate the use of a software and make the learning curve of a user less steep.

This is particularly true for a medium-to-high entry-level code as CRYSTAL, which has been the target software in the present work. Regardless of recent developments in the capabilities of the CRYSTAL code, graphical tools for the analysis and visualisation of the predicted results, the creation of CRYSTAL input files and its usage have remained at a less developed stage.

Therefore, the present PhD work has tried to fill this gap by creating up to date modern web-oriented toolboxes to be machine independent, easy to use, and freely accessible to users from all over the world through Internet browsers. Such an ambitious goal has been achieved with two major outcomes: CRYSPLOT and CRYSTAL VLab, which have been presented in this PhD thesis.

**CRYSPLOT** is a web platform that has been designed as a very intuitive graphical tool, a low entry-level interface to all types of users: from beginners (i.e., students) to expert researchers. It is also a sufficiently advanced and powerful tool that enables users to customize graphs in such a way that results can be used in scientific publications.

Since CRYSPLOT reads data from formatted text files, it can be considered a versatile and general graphical tool. Indeed, in principle, it is not limited to visualise data computed with CRYSTAL but one could also obtain results from another

software, reorganize raw data according to the proper format and then plot the results with CRYSPLOT.

Since October 2017 CRYSPLOT is available online at <http://crysplot.crystalsolutions.eu/> and it is currently used by CRYSTAL users from all over the world.

**CRYSTAL VLab** has been designed to be an all-purpose graphical interface to aid users to create input files, run calculations and visualise computed results. In particular, for the latter, CRYSPLOT has been encoded into CRYSTAL VLab. At the moment CRYSTAL VLab is still under development but a preliminary version is available online at <http://vlab.crystalsolutions.eu/>. A beta version is planned to be released by the end of 2020.

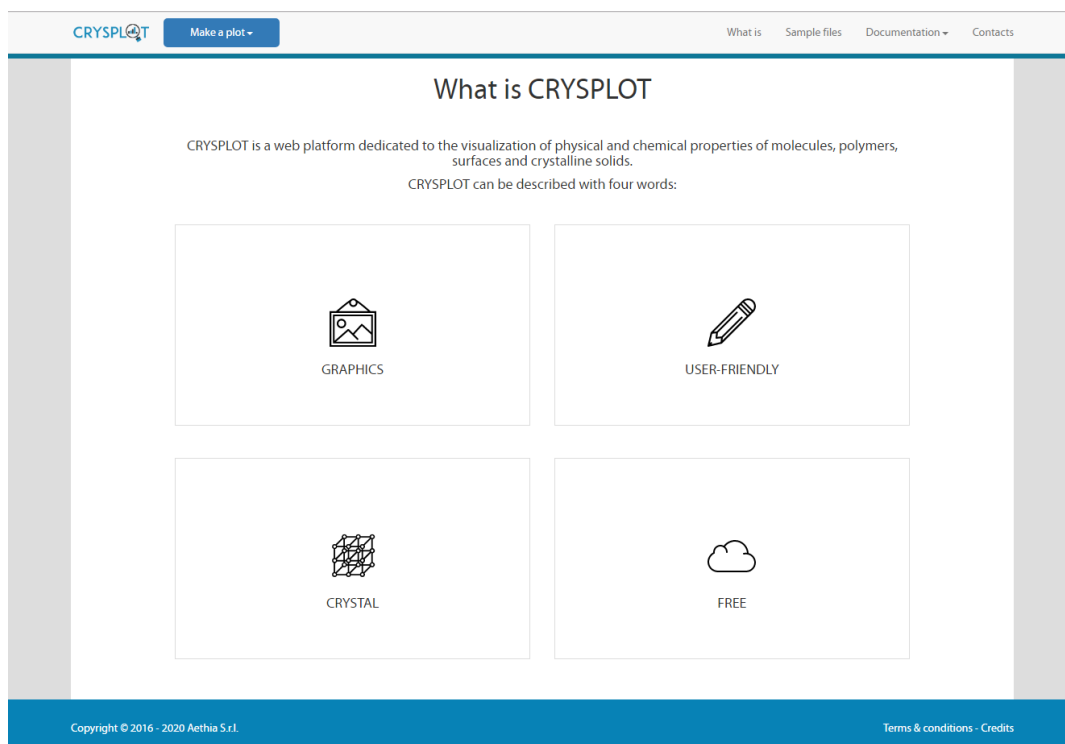
As next developments, I would like to focus on the inclusion of other advanced keywords in the input reading function of BUILD. In addition, I would also like to extend and make interactive some options in the .d3 wizard. For instance, to generate a path of k-points in the reciprocal space by simply clicking on the image of the Brillouin Zone. Furthermore, I plan to connect CRYSTAL VLab to a cloud computing space in which CRYSTAL users could run calculations by simply selecting an option "Cloud" in "Target host" menu of the COMPUTE part.

In conclusion, in this PhD work new graphical toolboxes have been created that allow satisfying some of the roles of visualisation discussed in the Introduction: visualisation as a support for exploration and cognition but also for aiding users in using computational tools, and visualisation as a creator of beautiful images for results dissemination and communication.

# Appendix 1

This section is dedicated to a brief visual guided tour of CRYSPLOT. Each page of the website is shown through an image and a small description.

## “What is” page



This page collects the main features of CRYSPLOT and in particular each of the four blocks in the center of the page shows a keyword: GRAPHICS, USER-FRIENDLY, CRYSTAL and FREE. If the user passes over keywords, a descriptive text appears for each of the 4 words. In the case of GRAPHICS the descriptive text is:

*CRYSPLOT is designed with a modern and innovative tool called Plotly. Plotly is designed for online analytics and data visualisation and it uses stack.gl and an advanced graphical javascript library called D3.js.*



## “Sample files” page

Description	File
Band structure of MgO in .f25 format	mgo_band.f25
Band structure of MgO in .BAND format	mgo_band_dat.BAND
Density of states of MgO in .f25 format	mgo_doss.f25
Density of states of MgO in .DOSS format	mgo_doss_dat.DOSS
Electron charge density map of Urea (with this file is possible to create the difference map from a single file)	urea_for_single_difference.f25
Electron charge density map of Urea (load this file first to create the difference map from multiple files)	urea_for_multiple_difference.f25
Electron charge density map of Urea (load this file per second to create the difference map from multiple files)	urea_multiple.f25
Infrared spectrum of mI33 (closed shell)	mI33_closed_b3-st_tzvp-tzp_fr_nd2_IRSPEC
Infrared spectrum of mI33 (open shell)	mI33_open_b3-st_tzvp-tzp_2_fr_nd2_IRSPEC

This page contains nine different sample files that can be downloaded to try using CRYSPLOT. In particular, there are sample files to plot band structure (one in .f25 format and one in .BAND format), density of states (one in .f25 format and one in .DOSS format), the electron charge density map (using these files it's possible to combine them also for making differences) and for vibrational spectra simulation. After download the user can open these files in corresponding page to see the chart and customize it.

## “Contacts” page

**Theoretical Chemistry Group**  
University of Turin  
Department of Chemistry  
Via Granda, 7  
10125 Torino  
Italy

**Aethia Srl**  
Via Ribes, 5  
10010 Colliareto Giacosa (TO)  
Italy

Name and surname \*

Email \*

Organization

Country

Your Message \*

I agree to the Crystal Solutions privacy policy \*

Send

This page contains a guided procedure to contact with an email the CRYSPLOT developer team.

## Appendix 2

All data to be plotted by CRYSPLOT are written in a formatted way (i.e. plain ASCII) either on the Fortran unit 25 (fort.25) or in separate files (e.g. BAND.DAT, IRSPEC.DAT, ...). The list of properties and the corresponding name of the auxiliary files is listed in Table S1.

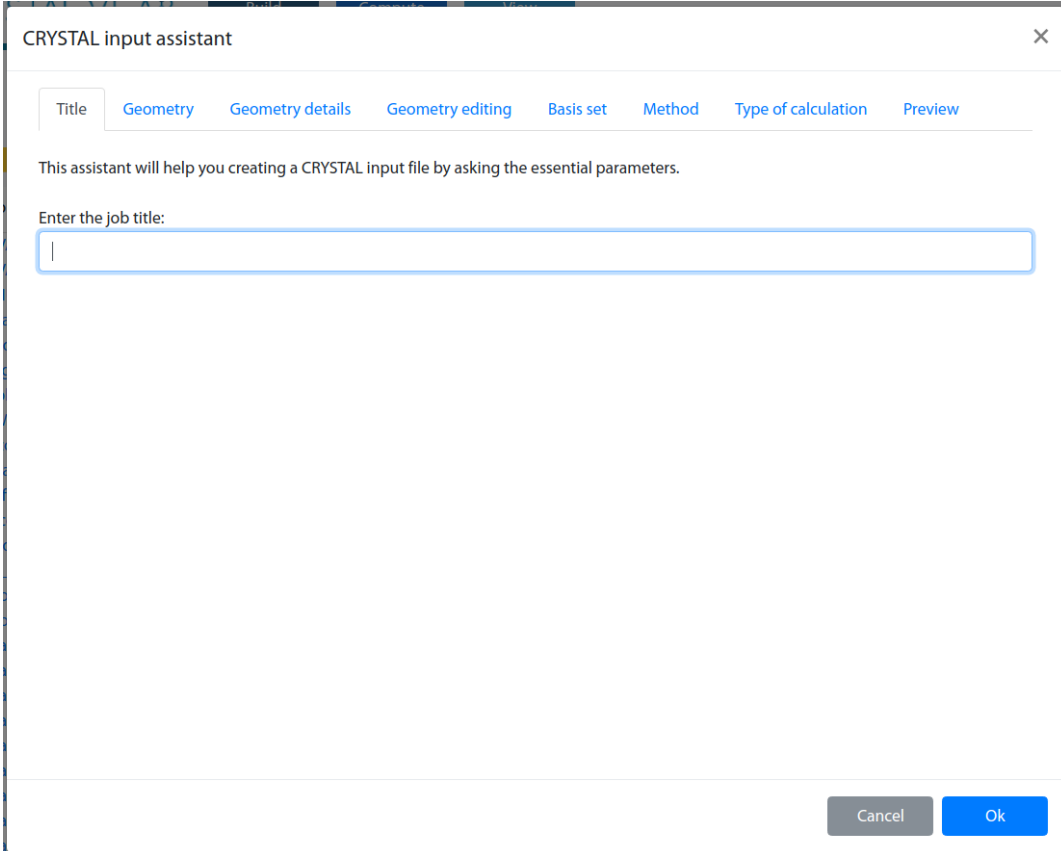
**Table S1.** List of properties in CRYSPLOT with relative supported file format (see the CRYSTAL Users' Manual for details<sup>1</sup>)

Property	File name or file extension
Band structure	BAND.DAT, *.BAND, *.f25
Density of states	DOSS.DAT, *.DOSS, *.f25
Crystal Orbital Hamiltonian Population	COHP.DAT, *.COHP, *.f25
Crystal Orbital Overlap Population	COOP.DAT, *.COOP, *.f25
Electron charge density map	*.f25
Electrostatic potential map	*.f25
Electron charge density profile	RHOLINE.DAT, *.RHOLINE
Compton profiles	*.CP
Autocorrelation function	*.CP
Infrared spectra	IRSPEC.DAT, *.IRSPEC
Raman spectra	RAMSPEC.DAT, *.RAMSPEC
Reflectance spectra	IRREFR.DAT, *.IRREFR
IR dielectric response spectra	IRDIEL.DAT, *.IRDIEL
Phonon band structure	*.f25
Phonon density of states	*.f25
Topological Analysis	*.DAT
Electron conductivity	*.DAT
Seebeck coefficient	*.DAT
Electron thermal conductivity	*.DAT

## Appendix 3

This section is dedicated to a visual guided tour of CRYSTAL VLab wizard for .d12 input files. For each wizard tab there is an image and a short description.

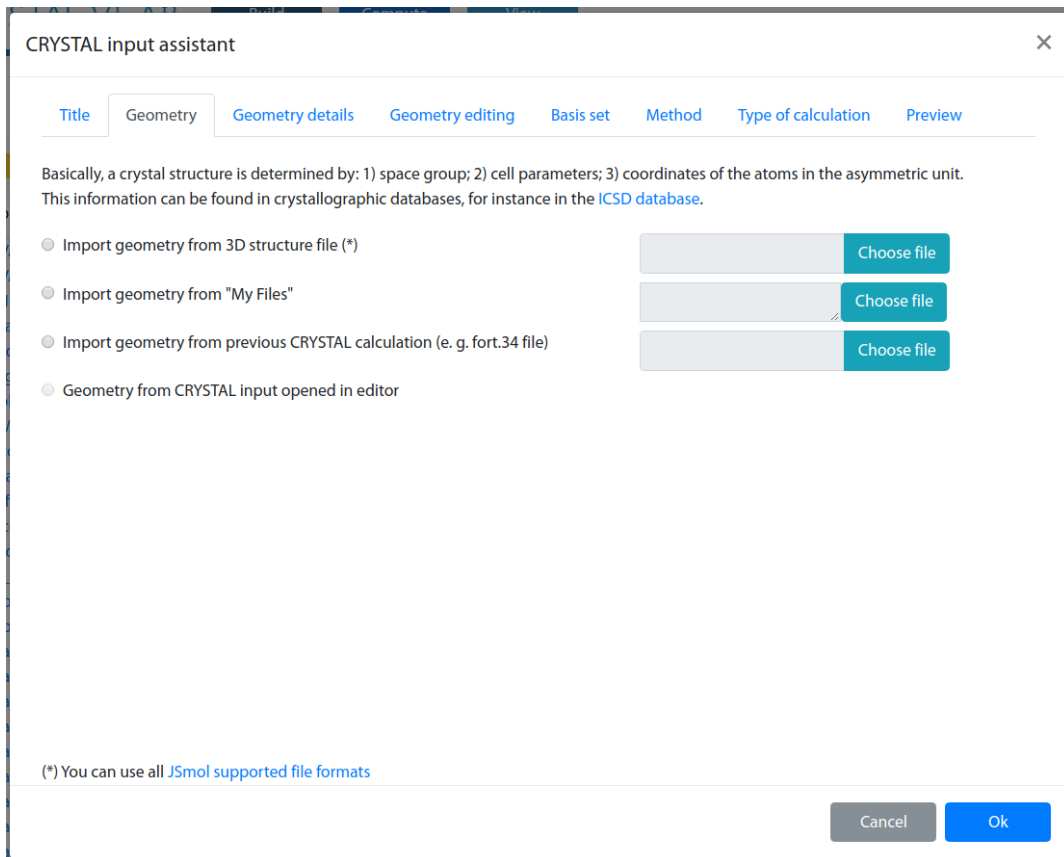
### “Title” tab



The screenshot shows a window titled "CRYSTAL input assistant" with a close button (X) in the top right corner. Below the title bar is a horizontal tab bar with seven tabs: "Title", "Geometry", "Geometry details", "Geometry editing", "Basis set", "Method", and "Type of calculation". The "Title" tab is selected and highlighted. Below the tabs, the text reads: "This assistant will help you creating a CRYSTAL input file by asking the essential parameters." Below this text is a label "Enter the job title:" followed by a large, empty text input field with a blue border and a vertical cursor. At the bottom right of the window are two buttons: "Cancel" (grey) and "Ok" (blue).

In this tab is possible to insert the title of the calculation. Moreover when the user opens this section there is a preselection on the area dedicated to title.

## “Geometry” tab



In this tab the user can load his geometry in different ways:

- From a 3D structure file on his local PC,
- From a file contained in his “My Files” section; in this case the file is not a local one, but it is stored on the server,
- From a fort.34 file on his local PC; in this case if the user decides to run CRYSTAL on his computer, this file is downloaded with the .d12 file.

The tab has also another option but this is the one that is automatic flagged when the user opens the wizard for editing a file already opened in the page.

## “Geometry details” tab

CRYSTAL input assistant

Title Geometry **Geometry details** Geometry editing Basis set Method Type of calculation Preview

Available online tutorial: [CRYSTAL geometry input](#)

Dimensionality of the system: CRYSTAL Lattice type: Triclinic Space group: P 1 (IGR 1)

Setting of the origin: 2nd (default) Lattice parameters: a, b, c,  $\alpha$ ,  $\beta$ ,  $\gamma$

Selected geometry

Atom coordinates

Atom	X	Y	Z
------	---	---	---

Geometry display

Add Remove Edit TESTGEOM

Cancel Ok

In this tab the user has to define the dimensionality of the system and, accordingly, all crystallographic parameters.

In the figure above, an example of a system with 3D dimensionality (i.e. CRYSTAL) is shown. In that case, the required parameters are, namely: lattice type, space group, setting of the origin and lattice parameters.

Below the parameters definition, the atomic positions in the unit cell are reported.

On the left, there is a table denoted as “Atom coordinates” in which the atoms in the asymmetric unit are listed by means of their atomic number and fractional coordinates. This table can be filled both manually, by entering the coordinates of

each atom, or automatically when the user loads the geometry from an existing input. By using "Add", "Remove" and "Delete" it is possible to change it.

On the right, there is a graphical box called "Geometry display" in which the JSmol Applet is loaded to display the specified geometry.

Notably, there is a button called "TESTGEOM" that allows user to quickly check if the geometry of the examined system is correct. In this case the calculation runs on a hosting server rather than on the local user PC (full calculation).

## “Geometry editing” tab

The screenshot shows the 'CRYSTAL input assistant' window with the 'Geometry editing' tab selected. The window has a title bar with a close button (X) and a navigation bar with tabs: Title, Geometry, Geometry details, Geometry editing (active), Basis set, Method, Type of calculation, and Preview. Below the navigation bar, there is a link for an online tutorial: 'Available online tutorial: CRYSTAL geometry input'. The main content area is divided into sections by horizontal lines. The first section is 'SUPERCELL', which is currently disabled (checkbox is unchecked). It contains a 3x3 grid of input fields labeled e11, e12, e13, e21, e22, e23, e31, e32, and e33. The second section is 'SLABINFO', also disabled, with three input fields labeled h, k, and l. The third section is 'SLABCUT', disabled, with three input fields labeled h, k, and l, and two additional fields labeled ISUP and NL. The fourth section is 'ATOMORDE', disabled, with no input fields. The fifth section is 'MOLECULE', disabled, with an input field labeled NMOL and a blue 'Set' button. Below this, there is a label '1:' followed by four input fields labeled ISEED, J COORD, K COORD, and L COORD. At the bottom right of the window are 'Cancel' and 'Ok' buttons.

This tab allows users to modify the initial geometry, in particular they can create a supercell, cut a slab model or extract a molecule.

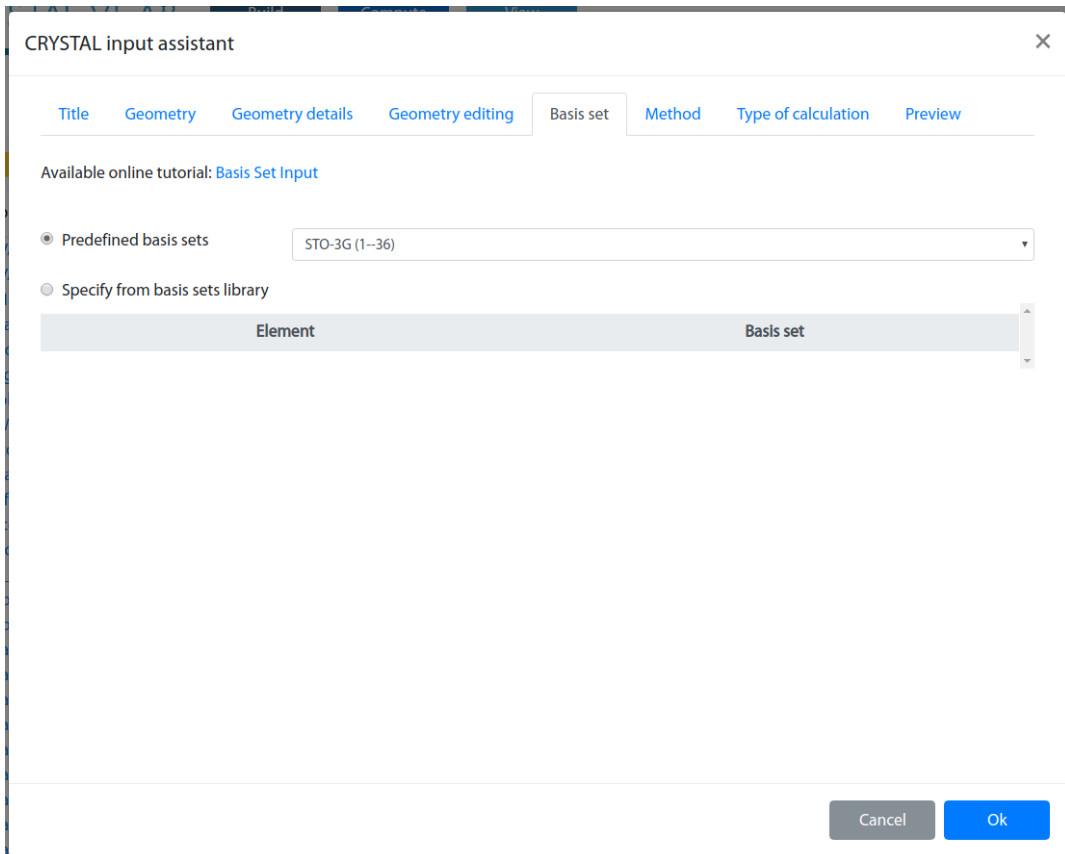
These keywords are enable/disabled depending on the dimensionality of the system to help users in the input creation.

In the case of SUPERCELL, SLABINFO and SLABCUT there are some predefined fields to specify the required parameters

Instead, for the option MOLECULE fields are dynamical because the number of atoms depends by the number of molecules to be isolated.

The ATOMORDE is a standalone keyword, so that no parameters are needed.

## “Basis set” tab



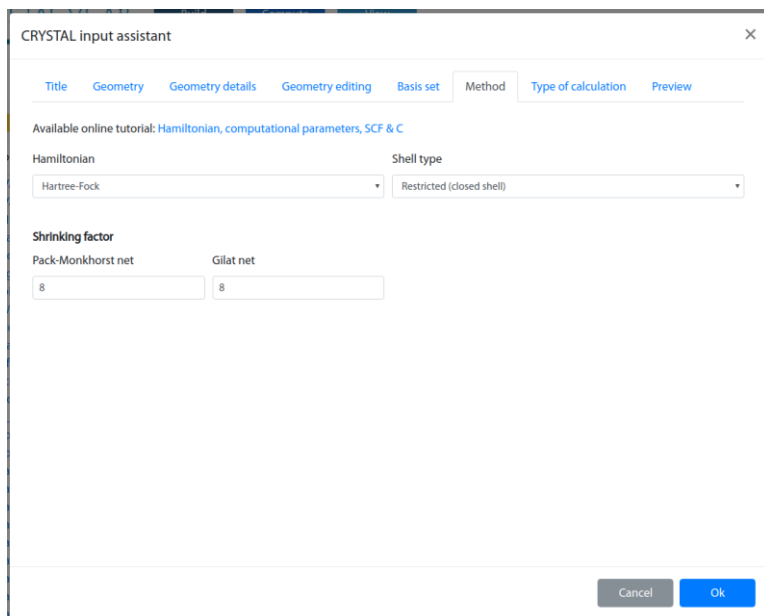
In this tab the user has to specify the basis set for the examined system. This can be done either selecting a predefined basis sets or taking the basis set from a library pre-loaded on VLab that contains basis sets for each element.

In the image above the table of the option “Specify from basis sets library” is empty because no geometry is loaded and there aren’t assigned elements. However, if the user defines a geometry, wizard can capture elements and contact basis sets database on server to obtain basis sets relative to elements in the geometry. The user can also see a preview of the selected basis sets.



## “Method” tab

In this tab the user has to define the method to run calculations, in particular one can select the Hartree-Fock method (first image) which is the default in CRYSTAL, and several DFT methods either through standalone keywords or specifying the the exchange and correlation functionals (second image).



CRYSTAL input assistant

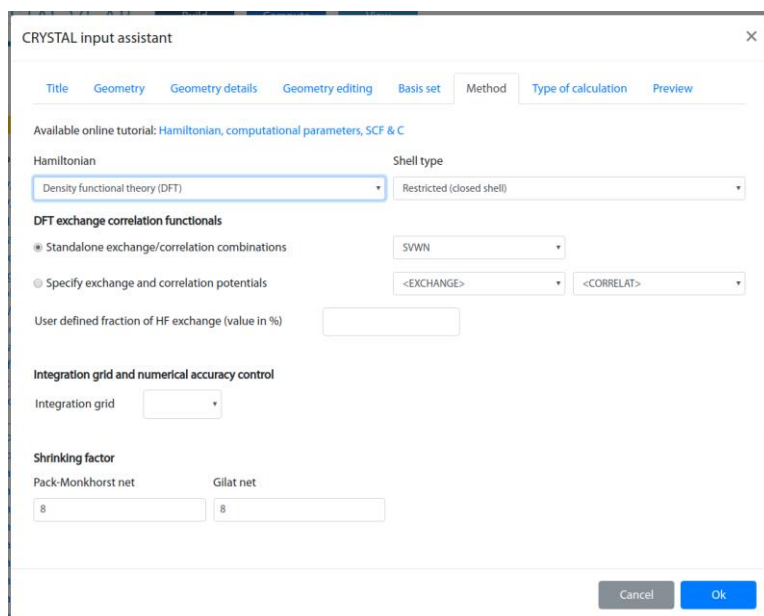
Title Geometry Geometry details Geometry editing Basis set **Method** Type of calculation Preview

Available online tutorial: [Hamiltonian, computational parameters, SCF & C](#)

Hamiltonian: Hartree-Fock  
Shell type: Restricted (closed shell)

Shrinking factor  
Pack-Monkhorst net: 8  
Gilat net: 8

Cancel Ok



CRYSTAL input assistant

Title Geometry Geometry details Geometry editing Basis set **Method** Type of calculation Preview

Available online tutorial: [Hamiltonian, computational parameters, SCF & C](#)

Hamiltonian: Density functional theory (DFT)  
Shell type: Restricted (closed shell)

**DFT exchange correlation functionals**

- Standalone exchange/correlation combinations: SVWN
- Specify exchange and correlation potentials: <EXCHANGE>, <CORRELAT>

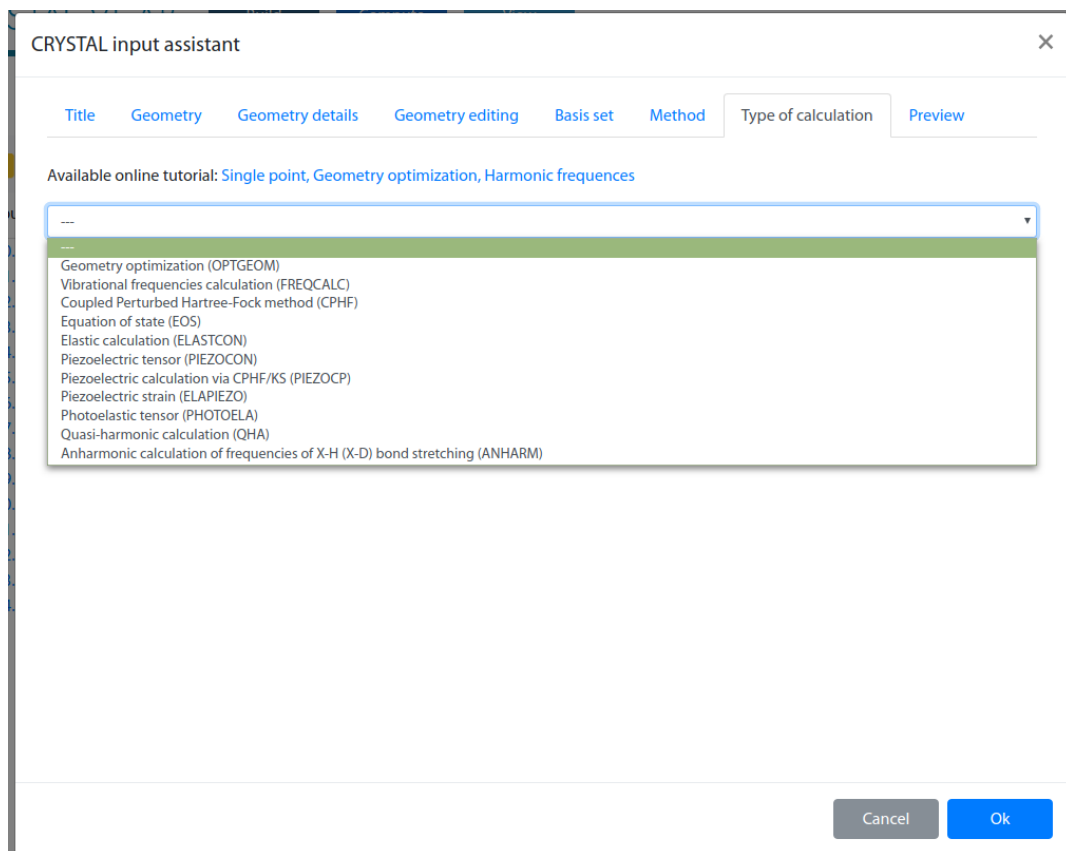
User defined fraction of HF exchange (value in %):

**Integration grid and numerical accuracy control**  
Integration grid:

**Shrinking factor**  
Pack-Monkhorst net: 8  
Gilat net: 8

Cancel Ok

## “Type of calculations” tab



In this tab a user can specify the type of calculations to be run. The main options available in CRYSTAL have been included in the list: OPTGEOM, FREQCALC, CPHF, EOS, ELASTCON, PIEZOCON, PIEZOCP, ELAPIEZO, PHOTOELA, QHA and ANHARM. Every type of calculations has keywords divided in basic and advanced, but in both cases the procedure is guided for user.

# Appendix 4

In this appendix a visual guided tour of the CRYSTAL VLab wizard for .d3 input files is shown. Below there are two screenshots for the pages of the density of states and the electron charge density, respectively.

CRYSTAL properties assistant

Select properties DOSS ECHG Preview

CRYSTAL keyword: DOSS

Number of projected densities (if 0 only total DOS is calculated)

Number of uniformly spaced energy values where DOSs are calculated, from bottom of first band to top of last one

Projected density records

First band Last band Number of printing options to switch on

Plot option Number of Legendre polynomials used to expand DOSs (max 25)

Cancel Ok

*Density of States*

Keyword: DOSS

CRYSTAL properties assistant

Select properties DOSS ECHG Preview

CRYSTAL keyword: ECHG

Order of the derivatives Number of point along the B-A segment

Coordinates definition

MARGINS (width of the margins that are added to the window, in order AB, CD, AD, BC)

AB: CD: AD: BC:

Cancel Ok

*Electron Charge  
Density*

Keyword: ECHG

# Appendix 5

In the section on “CRYSTAL VLab input reading”, the code that parses .d12 input file in .json format has been briefly discussed.

In this appendix, the parser function `ia_converter_crystal2json`, is presented in more detail. For sake of conciseness, the code is explained only for a 3D system (CRYSTAL keyword). For the other cases with different dimensionality, the structure of the code is very similar.

After the description of the function there is an example of a .d12 file, in particular MgO.d12, in the two formats, .d12 and .json.

```
def ia_converter_crystal2json(request):  
  
    # initialization of variables, in particular numbers, arrays or Boolean flags  
    groupGeom = []  
    groupLayer = []  
    groupRod = []  
    groupPoint = []  
    n = 0  
    path = ""  
    blockOffset = 0  
    d12 = {}  
    numb_atom = 0  
    atom_counter = 0  
    numb_atomslab = 0  
    atom_counter_slab = 0  
    numb_atompoly = 0  
    atom_counter_poly = 0  
    numb_atommol = 0  
    atom_counter_molecule = 0  
    bs_elem_cnt = 0  
    bs_elem_bsblock_num = 0  
    bs_elem_bsblock_cnt = 0  
    bs_elem_bsblock_gtf_num = 0  
    bs_elem_bsblock_gtf_cnt = 0  
  
    counter_scelphono = 0  
    counter_supercell = 0  
    counter_slabcut = 0  
    counter_molecule_geom_edit = 0  
  
    flag_optgeom_intredun_end = False  
  
    flag_optgeom_fixdef_first_line = True  
    flag_optgeom_fixcoord_first_line = True  
    flag_optgeom_fragment_first_line = True  
    flag_optgeom_lngsfrozen_first_line = True  
    flag_optgeom_angsfrozen_first_line = True  
    flag_optgeom_freezdih_first_line = True  
  
    flag_optgeom_tsopt_modefollow = True  
    flag_optgeom_tsopt_pathfollow = True  
    flag_optgeom_tsopt_fittopath = True
```

```

flag_optgeom_intredun = False
flag_optgeom_intredun_angtodouble = True
flag_optgeom_intredun_dbanglist = True
flag_optgeom_intredun_dbanglist_first_line = True
flag_optgeom_intredun_deflngs = True
deflngs_n1_value = 0
counter_deflngs = 0

flag_optgeom_intredun_defangls = True
defangls_n1_value = 0
counter_defangls = 0

flag_optgeom_intredun_modintcoor = True
flag_optgeom_intredun_modintcoor_second_line = False
flag_optgeom_intredun_modintcoor_third_line = False
modintcoor_nmodi_value = 0
counter_modintcoor = 0

flag_optgeom_fixing_int_coord = True
flag_optgeom_scanredu = False

flag_optgeom_lngsfrozen = False
counter_lngsfrozen_mu= 0
lngsfrozen_mu_value= 0

flag_optgeom_angsfrozen = False
counter_angsfrozen_n1=0
angsfrozen_n1_value=0

flag_optgeom_freezint = False
flag_optgeom_freezdih = False

flag_freqcalc_ramexp = True

flag_eos_range = True

flag_piezocp_tolalpha = True

flag_photoela_tolalpha = True

flag_qha_temperat = True

flag_anharm_label = False

SVWN = ["SVWN", "BLYP", "PBEXC", "PBESOLXC", "SOGGAXC", "B3PW", "B3LYP", "PBE0",
        "PBESOL0", "B97H", "B1WC", "WC1LYP", "RSHXLDA", "HSE06", "HISS", "wB97",
        "wB97X", "LC-wPBE", "LC-wPBESOL", "LC-wBLYP", "M05", "M052X", "M06L", "M06",
        "M062X", "M06HF", "PBE0-13"]

# initialization of an empty array of 230 elements for CRYSTAL Space Group and upload
of values from a .csv file
for i in range(0, 230):
    groupGeom.append('')

with open('spacegroup_crystal.csv', 'r') as csvfile:
    groupGeom.append(0)
    spacegroup_crystal = csv.reader(csvfile, delimiter=',', quotechar='')
    for row in spacegroup_crystal:
        groupGeom[int(row[0])] = row[1]

# initialization of an empty array of 80 elements for SLAB Layer Group and upload of
values from a .csv file

```

```

for j in range(0, 80):
    groupLayer.append('')
with open('layergroup_slab.csv', 'r') as csvfile:
    groupLayer.append(0)
    layergroup_slab = csv.reader(csvfile, delimiter=',', quotechar='')
    for row in layergroup_slab:
        groupLayer[int(row[0])] = row[1]

# initialization of an empty array of 99 elements for POLYMER Rod Group and upload of
values from a .csv file
for k in range(0, 99):
    groupRod.append('')

with open('rodgroup_polymer.csv', 'r') as csvfile:
    groupRod.append(0)
    rodgroup_polymer = csv.reader(csvfile, delimiter=',', quotechar='')
    for row in rodgroup_polymer:
        groupRod[int(row[0])] = row[1]

# initialization of an empty array of 47 elements for MOLECULE Point Group and upload
of values from a .csv file
for m in range(0, 47):
    groupPoint.append('')

with open('pointgroup_molecule.csv', 'r') as csvfile:
    groupPoint.append(0)
    pointgroup_molecule = csv.reader(csvfile, delimiter=',', quotechar='')
    for row in pointgroup_molecule:
        groupPoint[int(row[0])] = row[1]

# here starts the reading of the input file
f = request.POST['content_file'].split('\n')

while n < len(f):
    line = f[n]
    # Title
    if n == 0:
        d12['title'] = line
        path = "BLOCK1"

    #=====
    # START Block 1 (geometry)
    #=====

    elif path.find("BLOCK1") != -1:
        if path == "BLOCK1":

            # different .json structure for different system dimensionality-
            CRYSTAL case
            if line == 'CRYSTAL':
                # initialization of .json for CRYSTAL and initialization of path
                variable, path variable allows to define how the .d12 is structured
                d12['geometry'] = {'external': 'no',
                    'dimensionality': {'type': 'CRYSTAL', 'params': {}},
                    'latticeType': '', 'spaceGroup': '',
                    'latticeParameters': {}, 'atomsParameters': [],
                    'typeOfCell': '', 'geometry_adv': [], 'numbAtom': ''},
                    'type_of_calculation': {'type': 'ENERGY', 'parameters': []},
                    'geometry_editing': {}}
                path = "BLOCK1.CRYSTAL"
            elif line == 'SLAB':
                d12['geometry'] = {...}

```

```

        path = "BLOCK1.SLAB"

elif line == 'POLYMER':
    d12['geometry'] = {...}
    path = "BLOCK1.POLYMER"

elif line == 'MOLECULE':
    d12['geometry'] = {...}
    path = "BLOCK1.MOLECULE"

else:
    print('invalid file format...')
    sys.exit(1)

# initialization of .json section for basis set and computation
# parameters: note that these parts are the same for different
# dimensionality

d12['basis_set'] = {'type': ''}

d12['computation'] = {'hamiltonian': '', 'shell_type': '',
                    'shrink_parameters': {}, 'computationParameters': {}}
#=====#
# CASO CRYSTAL (See CRYSTAL MANUAL page 18)
#=====#
# first three parameters: IFLAG, IFHR ed IFSO
elif path == "BLOCK1.CRYSTAL":
    ci = line.split()
    d12['geometry']['full_block_geometry'] = []
    d12['geometry']['dimensionality']['params']['IFLAG'] =
        int(ci[0])
    d12['geometry']['dimensionality']['params']['IFHR'] =
        int(ci[1])
    d12['geometry']['dimensionality']['params']['IFSO'] =
        int(ci[2])
    if int(ci[1]) == 0:
        d12['geometry']['dimensionality']['typeOfCell'] =
            'hexagonal'
    else:
        d12['geometry']['dimensionality']['typeOfCell'] =
            'rhombohedral'
    # note that at the end of every step, path variable is
    # updated
    path = "BLOCK1.CRYSTAL.PARAMS"

# lattice type (triclinic, monoclinic, orthorhombic, tetragonal,
# trigonal, hexagonal, cubic) using space number
elif path == "BLOCK1.CRYSTAL.PARAMS":
    group_str = line.split()
    group = int(group_str[0])
    if group <= 2:
        d12['geometry']['dimensionality']['latticeType'] =
            'triclinic'
    elif group >= ... and group <= ...:
        d12['geometry']['dimensionality']['latticeType'] =
            '...'

# space group chosen from uploaded variable groupGeom
d12['geometry']['dimensionality']['spaceGroup'] =
    groupGeom[group]

path = "BLOCK1.CRYSTAL.PARAMS.SPACE"

```

```

# lattice parameters depending from lattice type
elif path == "BLOCK1.CRYSTAL.PARAMS.SPACE":
    lattice = line.split()
    if d12['geometry']['dimensionality']['latticeType'] ==
        'triclinic':
        d12['geometry']['dimensionality']
        ['latticeParameters']['a'] = float(lattice[0])
        d12['geometry']['dimensionality']
        ['latticeParameters']['b'] = float(lattice[1])
        d12['geometry']['dimensionality']
        ['latticeParameters']['c'] = float(lattice[2])
        d12['geometry']['dimensionality']
        ['latticeParameters']['alfa'] = float(lattice[3])
        d12['geometry']['dimensionality']
        ['latticeParameters']['beta'] = float(lattice[4])
        d12['geometry']['dimensionality']
        ['latticeParameters']['gamma'] = float(lattice[5])
    elif d12['geometry']['dimensionality']['latticeType'] == ...:
        ...
    path = "BLOCK1.CRYSTAL.PARAMS.SPACE.LATTICE"

# number of non equivalent atoms in the system
elif path == "BLOCK1.CRYSTAL.PARAMS.SPACE.LATTICE":
    numb_atomlist = line.split()
    numb_atom = int(numb_atomlist[0])
    d12['geometry']['dimensionality']['numbAtom'] = numb_atom
    path = "BLOCK1.CRYSTAL.PARAMS.SPACE.LATTICE.NUMB_ATOM"

# "if" condition to read rows with type of calculation parameters
elif path in ['BLOCK1.CRYSTAL.PARAMS.SPACE.LATTICE.NUMB_ATOM',
'BLOCK1.ADVANCED'] and atom_counter == numb_atom:
    geometry_kw = line
    if geometry_kw == 'END':
        path = "BLOCK2"
    else :
        path = "BLOCK1.ADVANCED"

        # if row starts with BASISSET, path is equal to
        # BLOCK2 and code knows that basis set part starts
        if geometry_kw == "BASISSET":
            path = "BLOCK2"
            n -= 1

        # if that control if in the input file there are
        # some geometry editing
        elif geometry_kw == 'SUPERCELL':

            d12['geometry']['type_of_calculation']
            ['geometry_editing'].update({
            'SUPERCELL': []})
            path = 'BLOCK1.ADVANCED.SUPERCELL'

        elif geometry_kw == 'SLABINFO':
            d12['geometry']['type_of_calculation']
            ['geometry_editing'].update({
            'SLABINFO': []})
            path = 'BLOCK1.ADVANCED.SLABINFO'

        elif geometry_kw == 'SLABCUT':

```



```

        d12['geometry']['type_of_calculation']['geometry_editing'].update({
            'SLABCUT': {'Miller': [], 'layer': []}})
        path = 'BLOCK1.ADVANCED.SLABCUT'

elif geometry_kw == 'ATOMORDE':

    d12['geometry']['type_of_calculation']['geometry_editing'].update({
        'other_keywords': 'ATOMORDE'})
    path = 'BLOCK1.CRYSTAL.PARAMS.SPACE.LATTICE.NUMB_ATOM'

elif geometry_kw == 'MOLECULE':

    d12['geometry']['type_of_calculation']['geometry_editing'].update({
        'MOLECULE': {'nmol': '', 'atoms': []}})
    path = 'BLOCK1.ADVANCED.MOLECULE'

# if row starts with SCELPHONO, code knows that
# starts a type of calculation part: path is now equal
# to BLOCK1.ADVANCED.SCELPHONO and type_of_calculation
# dictionary is updated with an element SCELPHONO
elif geometry_kw == 'SCELPHONO':
    d12['geometry']
    ['type_of_calculation'].update(
        {'SCELPHONO': []})
    path = 'BLOCK1.ADVANCED.SCELPHONO'

# if row starts with OPTGEOM, code knows that starts
# a type of calculation part: path is now equal to
# BLOCK1.ADVANCED.OPTGEOM and type_of_calculation
# dictionary is updated with some elements
# corresponding to optgeom parameters
elif geometry_kw == 'OPTGEOM':
    d12['geometry']['type_of_calculation']
    ['type'] = 'OPTGEOM'
    d12['geometry']['type_of_calculation']
    ['parameters'] = {'type_optimization': '',
        'geometry_constraints': '',
        'convergence_criteria': '',
        'other_keywords': []}
    path = 'BLOCK1.ADVANCED.OPTGEOM'

elif geometry_kw == ...:
    d12['geometry']['type_of_calculation']
    ['type'] = ...
    d12['geometry']['type_of_calculation']
    ['parameters'] = {...}
    path = 'BLOCK1.ADVANCED...'

# atomic number and atoms coordinates
elif path == "BLOCK1.CRYSTAL.PARAMS.SPACE.LATTICE.NUMB_ATOM":
    atoms = line.split()
    d12['geometry']['dimensionality']
    ['atomsParameters'].append({'atomic_number': int(atoms[0]),
        'x_coord': float(atoms[1]),
        'y_coord': float(atoms[2]),
        'z_coord': float(atoms[3])})

# atoms counter that increases depending on number of lines
read

```

```

        atom_counter += 1

#=====
# END CAS0 CRYSTAL (See CRYSTAL MANUAL page 18)
#=====
# START type of calculation
#=====
#example of SCELPHONO option
elif path == "BLOCK1.ADVANCED.SCELPHONO":
    d12['geometry']['full_block_geometry'].append(line)
    line = line.split()
    counter_scelphono += 1
    # if SCELPHONO part is finished, path changes and in the
    # next step of the while cycle, code will stop before because
    # can be present other type of calculation options
    if counter_scelphono == 4:
        path = 'BLOCK1.CRYSTAL.PARAMS.SPACE.LATTICE.
              NUMB_ATOM.ADVANCED'
        n -= 1
    # if SCELPHONO part is not finished, code stores the values
    else:
        d12['geometry']['type_of_calculation']
        ['SCELPHONO'].append([line[0], line[1], line[2]])
#example of OPTGEOM option
elif path == 'BLOCK1.ADVANCED.OPTGEOM':
    d12['geometry']['full_block_geometry'].append(line)

    # in the first two cases, code passes to basis set block
    if line == "BASISSET":
        path = "BLOCK2"
        n -= 1

    elif line == 'END' or line == 'ENDOPT':
        path = "BLOCK2"

    elif line in
    ["FULLOPTG","ATOMONLY","CELLONLY","INTREDUN","TSOPT"]:
        d12['geometry']['type_of_calculation']
        ['parameters']['type_optimization'] = line

    elif line == ...:
        d12['geometry']['type_of_calculation']
        ['parameters'].update({...})
elif path == 'BLOCK1.ADVANCED...:
    ...
#=====
# END type of calculation
#=====
# END Block 1 (geometry)
#=====
# START Block 2 (basis set)
#=====
elif "BLOCK2" in path:
    if path == "BLOCK2":
        kw_basisset = line.split()
        if kw_basisset[0] == 'END':
            pass
        # condition for implicit basisset

```

```

elif kw_basisset[0] == "BASISSET":
    d12['basis_set']['type'] = 'implicit'
    path = "BLOCK2.BASISSET"
# condition for explicit basisset
else:
    d12['basis_set']['type'] = 'explicit'
    d12['basis_set']['elements'] = []
    path = "BLOCK2.ELEM"
    n -= 1

# implicit basisset → code passes to block 3
elif path == "BLOCK2.BASISSET":
    name_basisset = line.split()
    d12['basis_set']['name'] = name_basisset[0]
    path = "BLOCK3"

# explicit basisset → code stores the explicit basisset for every
# element, in particular in the following "else" stores some parameters
# and in the next elif the numbers of basisset
elif path == "BLOCK2.ELEM":
    kw_basisset = line.split()
    if len(kw_basisset) == 2 and kw_basisset[0] == '99' and
        kw_basisset[1] == '0':
        path = "BLOCK2.END"
else:
    d12['basis_set']['elements'].append(
        {'atomic_number': kw_basisset[0],
         'block_number': kw_basisset[1],
         'bs_blocks': [], 'bs_full_blocks': line})
    bs_elem_bsbblock_num = int(kw_basisset[1])
    bs_elem_bsbblock_cnt = 0
    path = "BLOCK2.ELEM.BS_BLOCK_START"

elif path == "BLOCK2.ELEM.BS_BLOCK_START":
    kw_basisset = line.split()
    if len(kw_basisset) == 2 and kw_basisset[0] == '99' and
        kw_basisset[1] == '0':
        path = "BLOCK2.END"
else:
    d12['basis_set']['elements']
    [bs_elem_cnt]['bs_blocks'].append({
        'ITYB': kw_basisset[0],
        'LAT': kw_basisset[1],
        'NG': kw_basisset[2],
        'CHE': kw_basisset[3],
        'SCAL': kw_basisset[4], 'GTF_lines': []})

    d12['basis_set']['elements']
    [bs_elem_cnt]['bs_full_blocks'] += "\n" + line

    bs_elem_bsbblock_gtf_num = int(kw_basisset[2])
    bs_elem_bsbblock_gtf_cnt = 0
    path = "BLOCK2.ELEM.BS_BLOCK_DATA"

elif path == "BLOCK2.ELEM.BS_BLOCK_DATA":
    d12['basis_set']['elements'][bs_elem_cnt]
    ['bs_blocks'][bs_elem_bsbblock_cnt]['GTF_lines'].append(line)

    d12['basis_set']['elements'][bs_elem_cnt]
    ['bs_full_blocks'] += "\n" + line

```

```

        bs_elem_bsblock_gtf_cnt += 1
        if bs_elem_bsblock_gtf_cnt == bs_elem_bsblock_gtf_num:
            bs_elem_bsblock_cnt += 1
            if bs_elem_bsblock_cnt == bs_elem_bsblock_num:
                bs_elem_cnt += 1
                path = "BLOCK2.ELEM"
            else:
                path = "BLOCK2.ELEM.BS_BLOCK_START"

    elif path == "BLOCK2.END":
        path = "BLOCK3"

#=====
# END Block 2 (basis set)
#=====

#=====
# START Block 2 (Hamiltonian and SCF computational parameters)
#=====
elif path == "BLOCK3":
    # method definition
    third_block_kw = line.split()
    if third_block_kw[0] == 'DFT':
        d12['computation']['hamiltonian'] = 'dft'
        d12['computation']['shell_type'] =
            'ia_inputShellTypeClosedShell'
        path = 'BLOCK3.DFT'
    elif third_block_kw[0] == 'UHF':
        d12['computation']['hamiltonian'] = 'hf'
        d12['computation']['shell_type'] =
            'ia_inputShellTypeOpenShell'
        path = 'BLOCK3.UHF'
    else:
        d12['computation']['hamiltonian'] = 'hf'
        d12['computation']['shell_type'] =
            'ia_inputShellTypeClosedShell'
        n -= 1
        path = 'BLOCK3.UHF'
    if line.split()[0] in SVWN:
        svwn_value = line.split()[0]
        d12['computation']['computationParameters']
            ['svwn_parameter'] = svwn_value

elif path == 'BLOCK3.DFT':
    if line.split()[0] == 'EXCHANGE':
        path = 'BLOCK3.DFT.EXCHANGE.START'

elif path == '...':
    ...

#=====
# END Block 2 (basis set)
#=====
# n counter of while cycle increases
n += 1

# d12 string is formatted in .json format
d12_json = json.dumps(d12, indent=4, sort_keys=True)

return JsonResponse(d12, safe=False)

```

## MgO.d12 input file

.d12 format	.json format
<pre>#GEOMETRY BLOCK# TEST11 - MGO BULK CRYSTAL 0 0 0 225 4.21 2 12 0.  0.  0.  8 0.5 0.5 0.5 END</pre>	<pre>{   "title": "TEST11 - MGO BULK",   "geometry": {     "dimensionality": {       "atomsParameters": [         {           "atomic_number": 12,           "x_coord": 0.0,           "y_coord": 0.0,           "z_coord": 0.0         },         {           "atomic_number": 8,           "x_coord": 0.5,           "y_coord": 0.5,           "z_coord": 0.5         }       ]     },     "geometry_adv": [],     "latticeParameters": {       "a": 4.21     },     "latticeType": "cubic",     "numbAtom": 2,     "params": {       "IFHR": 0,       "IFLAG": 0,       "IFSO": 0     },     "spaceGroup": "F m -3 m",     "type": "CRYSTAL",     "typeOfCell": "hexagonal"   },   "external": "no",   "full_block_geometry": [],</pre>

<pre> #BASIS SET BLOCK# 12 4 0 0 8 2.0 1.0 68370.0 0.0002226 9661.0 0.001901 2041.0 0.011042 529.6 0.05005 159.17 0.1690 54.71 0.36695 21.236 0.4008 8.791 0.1487 0 1 5 8.0 1.0 143.7 -0.00671 0.00807 31.27 -0.07927 0.06401 9.661 -0.08088 0.2092 3.726 0.2947 0.3460 1.598 0.5714 0.3731 0 1 1 2.0 1.0 0.688 1.0 1.0 0 1 1 0.0 1.0 0.28 1.0 1.0 8 4 0 0 8 2. 1. 8020.0 0.00108 1338.0 0.00804 255.4 0.05324 69.22 0.1681 23.90 0.3581 </pre>	<pre> "type_of_calculation": {   "geometry_editing": {},   "parameters": [],   "type": "ENERGY" } },  "basis_set": {   "elements": [     {       "atomic_number": "12",       "block_number": "4",       "bs_blocks": [         {           "CHE": "2.0",           "GTF_lines": [             " 68370.0 0.0002226",             " 9661.0 0.001901",             " 2041.0 0.011042",             " 529.6 0.05005",             " 159.17 0.1690",             " 54.71 0.36695",             " 21.236 0.4008",             " 8.791 0.1487"           ]         },         {           "ITYB": "0",           "LAT": "0",           "NG": "8",           "SCAL": "1.0"         }       ],     },     {       "CHE": "8.0",       "GTF_lines": [         " 143.7 -0.00671 0.00807",         " 31.27 -0.07927 0.06401", </pre>
---	---

<pre> 9.264 0.3855 3.851 0.1468 1.212 0.0728 0 1 4 6. 1. 49.43 -0.00883 0.00958 10.47 -0.0915 0.0696 3.235 -0.0402 0.2065 1.217 0.379 0.347 0 1 1 0. 1. 0.4567 1.0 1.0 0 1 1 0. 1. 0.1843 1.0 1.0 99 0 END </pre>	<pre> " 9.661 -0.08088 0.2092", " 3.726 0.2947 0.3460", " 1.598 0.5714 0.3731" ], "ITYB": "0", "LAT": "1", "NG": "5", "SCAL": "1.0" }, { "CHE": "2.0", "GTF_lines": [ " 0.688 1.0 1.0" ], "ITYB": "0", "LAT": "1", "NG": "1", "SCAL": "1.0" }, { "CHE": "0.0", "GTF_lines": [ " 0.28 1.0 1.0" ], "ITYB": "0", "LAT": "1", "NG": "1", "SCAL": "1.0" } }, "bs_full_blocks": "12 4\n0 0 8 2.0 1.0\n 68370.0 0.0002226\n 9661.0 0.001901\n 2041.0 0.011042\n 529.6 0.05005\n 159.17 0.1690\n 54.71 0.36695\n 21.236 0.4008\n 8.791 0.1487\n0 1 5 8.0 1.0\n 143.7 -0.00671 0.00807\n 31.27 -0.07927 0.06401\n 9.661 - 0.08088 0.2092\n 3.726 0.2947 0.3460\n 1.598 </pre>
---	--

	<pre> 0.5714 0.3731\n0 1 1 2.0 1.0\n 0.688 1.0 1.0\n0 1 1 0.0 1.0\n 0.28 1.0 1.0"     },     {       "atomic_number": "8",       "block_number": "4",       "bs_blocks": [         {           "CHE": "2.",           "GTF_lines": [             "8020.0 0.00108",             "1338.0 0.00804",             " 255.4 0.05324",             " 69.22 0.1681",             " 23.90 0.3581",             " 9.264 0.3855",             " 3.851 0.1468",             " 1.212 0.0728"           ],           "ITYB": "0",           "LAT": "0",           "NG": "8",           "SCAL": "1."         },         {           "CHE": "6.",           "GTF_lines": [             " 49.43 -0.00883 0.00958",             " 10.47 -0.0915 0.0696",             " 3.235 -0.0402 0.2065",             " 1.217 0.379 0.347"           ],           "ITYB": "0",           "LAT": "1",           "NG": "4",           "SCAL": "1."         }       ],     },   ], } </pre>
--	---



	<pre> {   "CHE": "0.",   "GTF_lines": [     " 0.4567 1.0 1.0"   ],   "ITYB": "0",   "LAT": "1",   "NG": "1",   "SCAL": "1." }, {   "CHE": "0.",   "GTF_lines": [     " 0.1843 1.0 1.0"   ],   "ITYB": "0",   "LAT": "1",   "NG": "1",   "SCAL": "1." } ], "bs_full_blocks": "8 4\n0 0 8 2. 1.\n8020.0 0.00108\n1338.0 0.00804\n 255.4 0.05324\n 69.22 0.1681\n 23.90 0.3581\n 9.264 0.3855\n 3.851 0.1468\n 1.212 0.0728\n0 1 4 6. 1.\n 49.43 -0.00883 0.00958\n 10.47 -0.0915 0.0696\n 3.235 -0.0402 0.2065\n 1.217 0.379 0.347\n0 1 1 0. 1.\n 0.4567 1.0 1.0\n0 1 1 0. 1.\n 0.1843 1.0 1.0" } ], "type": "explicit" }, </pre>
--	---

<pre>#METHOD BLOCK# SHRINK 8 8 FMIXING 30 PPAN END</pre>	<pre>"computation": {   "computationParameters": {},   "full_block_computation": [     "FMIXING",     "30",     "PPAN"   ],   "hamiltonian": "hf",   "shell_type": "ia_inputShellTypeClosedShell",   "shrink_parameters": {     "ShrinkGilat": "8",     "ShrinkPack": "8"   } }</pre>
--	---

## Appendix 6

When a user decides to run calculations, he has to create a new job using the pop-up window of Figure 3-10. When he clicks on “Start” button, a file in format .vlabjob is automatically created and downloaded; this file is in a format that is a bridge between the CRYSTAL VLab web-interface and the Starter.

In this appendix, I will show the structure of this particular file. Let’s start with an example showed below:

```
{
  "input_file_content": "b'TEST11 - MGO BULK\\n'b'CRYSTAL\\n'b'0
0 0\\n'b' 225\\n'b'4.21\\n'b'2\\n'b' 12 0.    0.    0.\\n'b' 8
0.5  0.5  0.5\\n'b'END\\n'b'12 4\\n'b'0 0 8 2.0 1.0\\n'b'
68370.0 0.0002226\\n'b' 9661.0 0.001901\\n'b' 2041.0
0.011042\\n'b' 529.6 0.05005\\n'b' 159.17 0.1690\\n'b' 54.71
0.36695\\n'b' 21.236 0.4008\\n'b' 8.791 79458\\n'b'0 1 5 8.0
1.0\\n'b' 143.7 -0.00671 0.00807\\n'b' 31.27 -0.07927
0.06401\\n'b' 9.661 -0.08088 0.2092\\n'b' 3.726 0.2947
0.3460\\n'b' 1.598 0.5714 0.3731\\n'b'0 1 1 2.0 1.0\\n'b' 0.688
1.0 1.0\\n'b'0 1 1 0.0 1.0\\n'b' 0.28 1.0 1.0\\n'b'8 4\\n'b'0 0 8
2. 1.\\n'b'8020.0 0.00108\\n'b'1338.0 0.00804\\n'b' 255.4
0.05324\\n'b' 69.22 0.1681\\n'b' 23.90 0.3581\\n'b' 9.264
0.3855\\n'b' 3.851 0.1468\\n'b' 1.212 0.0728\\n'b'0 1 4 6.
1.\\n'b' 49.43 -0.00883 0.00958\\n'b' 10.47 -0.0915 0.0696\\n'b'
3.235 -0.0402 0.2065\\n'b' 1.217 0.379 0.347\\n'b'0 1 1 0.
1.\\n'b' 0.4567 1.0 1.0\\n'b'0 1 1 0. 1.\\n'b' 0.1843 1.0
1.0\\n'b'99  0\\n'b'END\\n'b'SHRINK\\n'b'8
8\\n'b'FMIXING\\n'b'30\\n'b'PPAN\\n'b'END\\n'",
  "input_file_name": "mgo_clear_testgeom.d12",
  "job_id": 40,
  "ncpus": 1,
  "program": 0,
  "title": "mgo_clear_testgeom",
  "user_token":
  "b'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJqb2JfaWwQiojQwFQ.wUGNbR4
Kj-rWXctHh7ksqgp-SA_HqsoZK2-6wiifuYk'",
  "workdir": "Home",
  "user_id": 0,
}
```

A .vlabjob file is basically a text file structured in json format, i.e. like a dictionary composed by keys (for example "workdir") and corresponding values (for example "Home").

The key-value pairs are:

- "input\_file\_name": the value of this key is the name of the input file. The Starter will show this name in his interface, in a proper field.
- "input\_file\_content": the value of this key is the content of the input file properly formatted. The Starter can read this value and pass it to **crystal** program and the user can see this content in a dedicated area.
- "job\_id": the value of this key is the id assigned automatically to the job in the VLab web interface.
- "ncpus": the value of this key is the number of processors that the user wants to use for his/her calculations.
- "program": the value of this key identifies the type of program to use for running calculations; there is a correspondence between a number and a type of program:
  - "0" stands for crystal
  - "1" stands for Pcrystal
  - "2" stands for MPPpcrystal
  - "3" stands for properties
  - "4" stands for Pproperties
- "title": the value of this key identifies job title, that corresponds to the title in the Starter.
- "user\_token": the value of this key is a string of letters and numbers, which uniquely identifies a job and makes VLab-Starter communication as safe as possible. This token is created by VLab web-interface using a proper Python library called jwt; this library uses the encryption algorithm HS256. The token is passed to the Starter and, when uploading the results after the end of the calculations, it will pass also this token. This passage

allows to identify whether the call to the VLab server has a secure provenance or not.

- `"workdir"`: the value of this key is the name of the folder in VLab private file manager where the input file is stored. When the Starter will send output files, they will be stored in this folder.
- `"user_id"`: the value of this key is the id assigned to the user by VLab. This id is important when uploading the output as it allows to match the job with the right user that launched it.

When the calculation has finished, the Starter will send the results through a HTTP POST call to the server that passes these parameters:

- `"user_id"`
- `"workdir"`
- `"user_token"`
- `"job_id"`
- `"output_file_name"`: the name of the output files
- `"output_file_content"`: the content of the output files, base64 encoded

CRYSTAL VLab receives the parameters of this POST call, extracts `"output_file_name"` and `"output_file_content"` (decoded from base64 format) and saves the output files on the server in the user private file manager area.

The user can then visualise his/her results through the VIEW section of CRYSTAL VLab.

## Appendix 7

This section is dedicated to the structure of Django database under CRYSTAL VLab. In order to investigate the structure of apps and models it's necessary to take a deeper look into them. For this purpose, a description with code examples follows.

When using Django, the primary concept from which everything starts is the "project", that is (generally) understood to be a single, complete web site (or potentially a family of sites). A site might have several different functionalities — a blog, a forum, a store, a help chat. Each one of these is called an "application" (or just "app"). So a project is a collection of apps.

In the case of my Django project, that is called CRYSTAL\_VLab, the directory structure looks like this:

```
- /CRYSTAL_VLab/  
  - CRYSTAL_VLab /  
  - AppModel /  
  - AppCompute /  
  - AppView /  
  - AppFileMan /  
  - ImportBs /
```

CRYSTAL\_VLab is the project folder and all the other folders belongs to apps that manage different steps in VLab. Commands needed to create the project and the apps are:

```
~/django-projects > django-admin startproject CRYSTAL_VLab  
~/django-projects/CRYSTAL_VLab > python manage.py startapp  
AppFileMan
```

Django apps have something close to a Model-View-Controller architecture, which Django documentation sometimes calls **Model-View-Template**.

This leads to an easy-to-adopt, iterative development pattern:

- work on the **Model**, defining what types of data are going to be tracked and how they relate to each other;
- work on the **View**, defining how data should be accessed and what to do when data is manipulated;
- work on the **Template**, defining how data looks when displayed to the user or consumed by another application.

## MODEL

Building new features usually begins with creating models. Models are classes which define the various objects in the system and how they relate to each other.

For example in my AppFileMan, which organizes the management of files on the user's personal profile, the structure of folder is:

```
- /CRYSTAL_VLab/AppFileMan/  
  - __init__.py  
  - admin.py  
  - apps.py  
  - migrations/  
    - __init__.py  
  - models.py  
  - tests.py  
  - views.py
```

I defined two models, Folder and File, that are written in the models.py file like:

```
# LIBRARY IMPORT  
from django.conf import settings  
from django.db import models  
from django.db.models.signals import pre_delete  
from django.dispatch.dispatcher import receiver  
import os  
  
# FOLDER MODEL  
class Folder(models.Model):  
    name = models.CharField(max_length=100)  
    user = models.ForeignKey(settings.AUTH_USER_MODEL,  
                             on_delete=models.CASCADE,null=True, blank=True)
```

```

parent = models.ForeignKey("self", null=True, blank=True,
                           related_name='subfolders',
                           on_delete=models.CASCADE)

# DEFINITION OF METHOD as_tree FOR FOLDER MODEL
def as_tree(self):
    folders = list(self.subfolders.all())
    branch = bool(folders)
    yield branch, self
    for f in folders:
        for next in f.as_tree():
            yield next
    yield branch, None

# RE-DEFINITION OF METHOD __str__ FOR FOLDER MODEL
def __str__(self):
    return self.name

# FILE MODEL
class File(models.Model):
    file = models.FileField(upload_to=settings.UPLOAD_DIR,
                            max_length=100)
    name = models.CharField(max_length=100)
    folder = models.ForeignKey(Folder, null = True,
                              blank = True, related_name='files',
                              on_delete=models.CASCADE )
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
                            on_delete=models.CASCADE, null=True, blank=True)

# RE-DEFINITION OF METHOD __str__ FOR FILE MODEL
def __str__(self):
    return self.name

@receiver(pre_delete, sender=File)
def mymodel_delete(sender, instance, **kwargs):
    instance.file.delete(False)

```

In the example above, we see that it's possible to define in `models.py` methods relating to particular models, e.g. `as_tree`.

Django has its own Object Relational Mapper (ORM), which provides a layer of abstraction to the database. This allows user to define data model without reference to a specific database management system, and without having to write SQL.



As user adds models, he/she has to do “migrations”. A migration is an automatically-generated set to SQL commands that alters the database design to match the models; Django’s ORM translates user models into SQL migrations. Before doing this, the user is required to set up the database and connect to it.

After setting up the database, he/she can migrate data in this way:

```
~/django-projects > python manage.py makemigrations
~/django-projects > python manage.py migrate
```

## VIEW

In Django, the “View” isn’t a front-end HTML page (which is called instead “Template”). What Django calls View is a function which receives an HTTP Request as an argument, and returns a Response. Typically, the response is the content of a web page which is then accessible from a Template. A response can also be a redirect, an error message, a file, or something else.

The most common scenario is the one in which you want to send a Request for a particular piece of content (for example a folder), and then get back all the data associated with that piece of content (all files that belong to that folder).

Below there is an example of a view that receives in input a folder id and responds with all the files contained in that folder:

```
def file_man(request, folder_id):
    folder_id = int(folder_id)
    request.session['current_folder_id'] = folder_id
    folders = Folder.objects.filter(parent__isnull=True)

    # NEXT IF CONDITION CONTROLS IF THE FOLDER ID IS "0" OR NOT
    # AND THEN RESPONDS WITH A DICTIONARY OF FILES
    if folder_id is not None and folder_id != 0:
        files = File.objects.filter(folder__id=folder_id)
    else:
        files = File.objects.filter(folder__isnull=True)

    context = { 'title': 'CRYSTAL VLab',
                'files' : files ,
                'folders' : folders}

    # RETURN IS LINKED TO A TEMPLATE PAGE AND THE ELEMENTS IN
```

```
# context ARE NECESSARY TO COMPLETE THE TEMPLATE RENDERING
return render(request, 'AppFileMan/file_man_table.html',
              context)
```

After function definition in `views.py` file, user just needs to map his views to URLs. In particular a file called `urls.py` exist inside `AppFileMan` folder and it's structured in this way:

```
from django.conf.urls import url
from django.contrib.auth import views as auth_views
from django.contrib.auth.decorators import login_required

# IMPORT OF CODE INSIDE FILE .views.py
from .views import *

urlpatterns = [
    url(r'^folder/(?P<folder_id>[0-9]+)/files$',
        login_required(file_man), name='file_man'),
]
```

The above pattern match the `/fileman/folder/folder_id/files/` URL pattern to the `file_man` view, which calls its template `'AppFileMan/file_man_table.html'`. It's important to note that URL pattern contains a parameter, `folder_id`, that is defined with a regular expression in `urls.py` file and that represents the ID of the folder for which the user wishes to have all the files.

## TEMPLATE

The last step to close the Model-View-Template Django pattern is the creation of a template in html to display the content from the view. In fact it's possible to call variables from the view in the html template only using `{{...}}`.

For example, if the user wants to insert the value of the variable `'title'` included in the `context` (from the `file_man` view response), he/she has just to include in the HTML code `{{ title }}` and when the view is called the field in templates it's automatically substituted with its value.

# Ringraziamenti

Vorrei ringraziare innanzitutto la Regione Piemonte per aver supportato il mio progetto di Dottorato (Avviso Pubblico approvato con D.D. 537 del 03/08/2016), senza di loro non sarebbe stato possibile arrivare ai risultati ottenuti.

Ma oltre alla Regione Piemonte mi sento di voler ringraziare tutte quelle persone che mi hanno sostenuta in questo percorso così intenso ma così pieno di soddisfazioni, tutte quelle persone che mi hanno detto che sarei riuscita a finire tutto perché ero in gamba, tutte quelle persone che mi hanno tranquillizzato quando la mia solita ansia bussava alla spalla.

Grazie CRYSTAL Team, grazie per avermi accolta come una di famiglia e per aver avuto la pazienza di spiegare ogni passaggio ad una persona totalmente digiuna di chimica. I vostri sorrisi nel vedere i grafici con un vestito nuovo mi hanno ripagato di tutto il tempo impiegato per sciogliere la matassa complessa di CRYSTAL.

Grazie Mimmo, grazie perché sei stato il primo sostenitore dei miei risultati, colui che ha visto nascere e crescere il progetto, colui che l'ha pensato prima ancora che esistesse, colui che nonostante tutti gli impegni c'è sempre stato e ha sempre trovato il tempo per darmi retta. Grazie.

Grazie a Robert Hanson e a Yves Noël, grazie per aver dedicato del tempo al mio progetto: il vostro aiuto è stato per me prezioso. Grazie.

Grazie Aethia, grazie per avermi insegnato cosa vuol dire lavorare insieme, cosa vuol dire lavorare in una famiglia in cui ci si aiuta nei momenti difficili e ci si conosce davvero, pregi e difetti. Grazie.

Grazie Paolo, grazie perché la tua pazienza è stata una colonna portante di questo lavoro, grazie per avermi sempre sostenuto trovando un momento per aiutarmi,

grazie per avermi insegnato tutto ciò che ad oggi posso dire di saper fare. Grazie per aver scommesso su di me. Grazie.

Grazie a Marina, grazie per i tuoi modi gentili e il tuo sorriso, grazie per aver deciso, insieme a Paolo, di voler puntare su di me, su una persona che tre anni fa non sapeva fare nulla di tutto ciò che oggi fa. Grazie.

Grazie a Cadigia, grazie per il sostegno che mi hai sempre dato, grazie per essere ormai un'amica per me. Grazie.

Grazie a Vincenzo, grazie per essere stato mio compagno in tutti quei momenti di creatività, dal muro binario di Aethia all'albero di Natale. Grazie.

Grazie a Dan, grazie perché ormai sei il mio compagno di banco, quello che mi aiuta ogni volta in cui non ce la faccio, grazie per avere la pazienza di spiegarmi le cose mille e mille volte ma anche per ridere delle cose sceme che ti passo. Grazie.

Grazie a Alessia, grazie per l'empatia che hai saputo trasmettermi e per i sorrisi in quei momenti in cui ne avevo proprio bisogno. Grazie.

Grazie a Cecilia, grazie per essere diventata in così poco tempo una parte importante della mia vita, grazie per le chiacchiere e per aver condiviso con me delle parti di te, anche quelle più nascoste e fragili. Grazie.

Grazie a Marco, grazie per essere la parte calma e posata del nostro team, la parte che ad oggi ci mancava. Grazie.

Ed ora la mia famiglia. In inglese si dice *last but not least*, ebbene in questo caso non poteva calzare meglio, loro sono al fondo ma sono la cosa più importante che potessi avere nella mia vita.

Grazie a mamma Anna e papà Ezio, grazie per avermi sempre sostenuta e supportata, grazie per essermi sempre venuti incontro e per avermi sempre detto che ce l'avrei fatta. Grazie.

Grazie ai miei nonni Assunta e Giorgio che non sono mai venuti ad una mia discussione di tesi per la paura che mi avrebbero potuta bocciare. Vi adoro. Grazie.

Grazie allo zio Fabrizio che non ha mai perso un solo momento della mia vita, ognuno è stato immortalato con una foto e sono certa che anche di questo giorno avremo un ricordo per sempre. Grazie.

Grazie ad Ale e Stefy che mi hanno regalato la cosa più bella che potessi ricevere, una piccola peste che con i suoi occhioni mi ha conquistato dal primo sguardo. Grazie.

Ed ora il grazie più intenso di tutti perché rivolto a quella persona che ormai da tre anni vive con me ogni giorno, sostenendomi sempre. Grazie Alberto, grazie per aver accettato di essere la mia nuova famiglia diventando mio marito, grazie per tutto l'amore che mi dai, grazie per tutti gli abbracci e i sorrisi, grazie per aver imparato ad amare le cose per me importanti. Grazie per esserci sempre e per aver modellato te su di me. Grazie.

*...There's something in the water, That makes me love you like*

## **Published Papers**

# CRYSLOT: A New Tool to Visualize Physical and Chemical Properties of Molecules, Polymers, Surfaces, and Crystalline Solids

Giorgia Beata,<sup>\*[a,b]</sup> Gianpaolo Perego,<sup>[b]</sup> and Bartolomeo Civalleri <sup>\*[a]</sup>

CRYSLOT is a web-oriented tool (<http://crysplot.crystalsolutions.eu>) to visualize computed properties of periodic systems, in particular, as computed with the CRYSTAL code. Along with plotting, CRYSLOT also permits the modification and customization of plots to meet the standards required for scientific graphics. CRYSLOT has been designed with advanced and freely available graphical Javascript libraries as Plotly. The programming language used is Javascript. The code parses the input files, reads the data,

and organizes them into objects ready to be plotted with the plotly.js library. It is modular and flexible so that it is very simple to add other input data formats. The new graphical tool is presented in details along with selected applications on metal–organic frameworks to show some of its capabilities. © 2019 Wiley Periodicals, Inc.

DOI: 10.1002/jcc.25858

## Introduction

Visualization tools are becoming more and more important to analyze, help in understanding, and present scientific data.<sup>[1]</sup> Nowadays, this is particularly true in the realm of computational molecular and solid-state chemistry for which a paramount number of data can be obtained from more robust and multipurpose computational codes and through the accessibility of powerful computing facilities. In the world of computational solid-state chemistry, several different software exist for the ab initio study of physical and chemical properties of periodic systems as polymers, surfaces, and crystalline solids. Among them a prominent role is played by the CRYSTAL code,<sup>[2–7]</sup> a software developed since mid-1970s of the last century by the Theoretical Chemistry Group of the University of Torino and later in collaboration with the Computational Materials Science Group at Daresbury Laboratory (UK). Although CRYSTAL has recently shown a tremendous improvement in terms of advanced and powerful algorithms,<sup>[4,6]</sup> particularly, for speeding up calculations of the two parallel versions<sup>[8–10]</sup> and for extending the number of properties that can be computed,<sup>[7]</sup> the graphical tools for the analysis and visualization of the predicted results have remained at a less developed stage. To fill this gap, a new project, CRYSLOT, has started to provide end users with a modern, web-oriented, machine independent, quick, and easy-to-use visualization environment. In the following, we discuss in details technical aspects and features of CRYSLOT (i.e., how it works, what it does, etc.) along with selected examples to show the capabilities of the new graphical tool.

## Why A New Visualization Tool?

The CRYSTAL package performs ab initio calculations of the ground state energy, electronic wave function, and properties of periodic systems at the HF and DFT level of theory.<sup>[7]</sup> Presently, graphical tools available to analyze data computed with the CRYSTAL code are not unique and in some cases obsolete. Some of the computed

properties and related data can be plotted by means of Gnuplot.<sup>[11]</sup> Gnuplot is a widespread tool for plotting scientific data. It was originally created to allow scientists and students to visualize mathematical functions and data interactively, but nowadays it has grown to support many non-interactive uses such as web scripting. For instance, CRYSTAL can create files that can be read by gnuplot to plot simulated vibrational spectra. Other properties like band structure and density of states can be plotted with the homemade CrGra package whose latest release dates back to 2006. The development of the CrGra suite of programs started almost 30 years ago by the CRYSTAL team to create graphic plots in the PostScript language that could be easily printed. CrGra2006 processes data written by CRYSTAL in the Fortran unit “fort.25.”<sup>[7]</sup> The CrGra suite is comprised three modules: maps06 to draw contour maps (e.g., charge and spin density, electrostatic potential), bands06 to plot band structures and doss06 to plot total and projected density of states. However, the code was not updated for plotting data related to other properties that are now available from CRYSTAL (e.g., simulated IR/Raman spectra, topological analysis-related quantities, and electron conductivity).

In addition to these plotting tools, also other graphical tools are available to visualize data from CRYSTAL such as DLV<sup>[12]</sup> and XCRYSDEN.<sup>[13]</sup> Even if they provide a more extended graphical interface to the code (e.g., visualization of the crystalline structure and related features, animation of vibrational frequencies, etc., see refs. [12,13] for further details), they are currently not fully updated

[a] G. Beata, B. Civalleri  
Dipartimento di Chimica, Università di Torino, I-10125, Torino, Italy  
E-mail: giorgia.beata@aethia.com or bartolomeo.civalleri@unito.it

[b] G. Beata, G. Perego  
Aethia Srl, 10010, Collettero Giacosa, Italy  
E-mail: giorgia.beata@aethia.com

Contract Grant sponsor: Regione Piemonte; Contract Grant number: Avviso Pubblico approvato con D.D. 537 del 03/08/2

© 2019 Wiley Periodicals, Inc.

to the last release of the code and not available for all operating systems.

Therefore, we decided to create an up to date modern web-oriented visualization tool to be machine independent, easy to use, and freely accessible to users from all over the world through Internet browsers. Briefly, CRYSPLOT extends over the capabilities of CrGra to plot additional data (e.g., simulated IR/Raman spectra) and to manage multiple datasets. It also allows users to customize plots as detailed in the next section.

## Crystplot: What It Does and How It Works

CRYSPLOT is available online at <http://crystplot.crystalsolutions.eu/>.

In Figure 1, a screenshot of the CRYSPLOT web page is shown. Basically, CRYSPLOT allows users to visualize band structure, density of states, electron charge density and electrostatic potential maps, simulated vibrational spectra, topological analysis, phonon dispersion, and transport properties computed with CRYSTAL on a machine independent platform. It also aims at offering users with easy-to-use options to modify and customize plots to meet standards required for scientific publications. It is worthy to note that, although CRYSPLOT is targeted to CRYSTAL, the same properties as computed from other programs could be plotted by simply converting raw data to the CRYSTAL format as described in the Appendix of the CRYSTAL User's Manual.<sup>[7]</sup>

At the design stage of CRYSPLOT, we decided to create a cross platform tool for making it easily accessible to the largest number of users. For this reason, we decided to organize CRYSPLOT as a website and develop it as a web application by using HTML5 markup language, CSS3 style sheet language,

BOOTSTRAP front-end framework, and the Javascript programming language. BOOTSTRAP is a free front-end framework for faster and easier web development that includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation, modals, image carousels, and many other, as well as optional JavaScript plugins. In addition, BOOTSTRAP has the ability to easily create responsive designs, that is, web sites that automatically adjust themselves to look good on all devices, from small smartphones to large desktops. BOOTSTRAP is easy to use and compatible with all modern browsers (Chrome, Firefox, Internet Explorer, Safari, and Opera). For the graphical part, we have chosen PLOTLY.<sup>[14]</sup> It is a high-level, declarative charting library, built on top of d3.js and stack.gl. In CRYSPLOT, we use its javascript version named plotly.js. Plotly.js uses stack.gl for high-performance 2D and 3D charting and the charts are shipped with zoom, pan, hover, and click interactions like click-and-drag to zoom into a region, double-click to auto-scale, click on legend items to toggle traces. Plotly.js ships with 20 chart types, including 3D charts, statistical graphs, and SVG maps; it abstracts the types of statistical and scientific charts that one would find in packages like matplotlib, ggplot2, or MATLAB. The charts are described declaratively as JSON objects and every aspect of the charts, such as colors, grid lines, and the legend, has a corresponding set of JSON attributes.

## CRYSPLOT web page

The CRYSPLOT website (see Fig. 1) has a descriptive part (home-page and "What is"), an operative one ("Make a plot"), and user dedicated services ("Contacts"). The "Make a plot" button is the

The screenshot shows the CRYSPLOT website interface. At the top, there is a navigation bar with a 'Make a plot' button and links for 'What is', 'Sample files', 'Documentation', and 'Contacts'. The main content area features a large blue banner with the CRYSPLOT logo and the text 'A modern and easy to use visualization environment for plotting properties of crystalline solids as computed by means of the CRYSTAL code.' Below the banner, there are three columns of text: 'What is CRYSPLOT?', 'Why use CRYSPLOT?', and 'How CRYSPLOT is made?'. The footer contains copyright information for 2016-2018 Aethia S.r.l. and a link to 'Terms & conditions - Credits'.

Figure 1. Screenshot of the CRYSPLOT home page. [Color figure can be viewed at [wileyonlinelibrary.com](http://wileyonlinelibrary.com)]



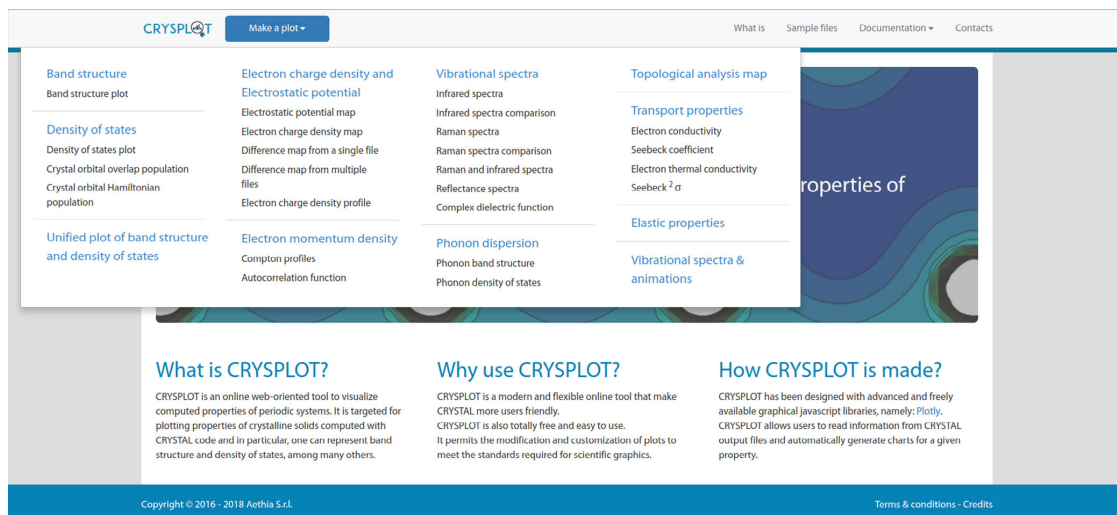


Figure 2. "Make a plot" cascade menu. [Color figure can be viewed at wileyonlinelibrary.com]

heart of CRYSPLOT. It opens a cascade menu from which user can select the property to be plotted as shown in Figure 2.

In addition, the header of the web page contains links to the pages described above: "CRYSPLOT" brings to the homepage, "What is" links to a page on the main features of CRYSPLOT, and "Contacts" to the contact page. Then, under the header, there is a tab in which the selected property will be plotted. Figure 2 also shows the list of available properties, namely: band structure, density of states, crystal orbital overlap population, crystal orbital Hamiltonian population, electron charge density maps and

profiles, electrostatic potential maps, Compton profiles, autocorrelation function, infrared, Raman, reflectance and complex dielectric spectra, phonon band structure and density of states, topological analysis, electron conductivity, Seebeck coefficient, and electron thermal conductivity. There is also the possibility to plot elastic properties through the link to the Elate web site,<sup>[15]</sup> a web tool for the analysis of elastic tensors, developed by F. X. Coudert and co-workers.

In each property tab, users may choose the output file from their local PC (see button "Choose file") to upload the data for

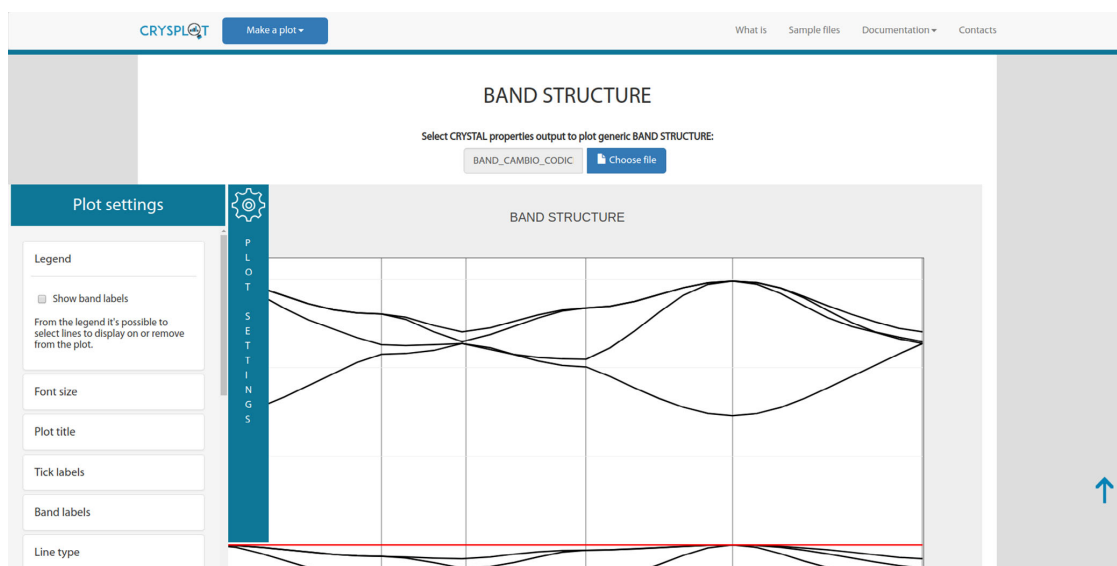


Figure 3. Example of plot settings palette as for the band structure plotting. [Color figure can be viewed at wileyonlinelibrary.com]

the selected property as previously computed by CRYSTAL. All data to be plotted are written in a formatted way (i.e., plain ASCII) either on the Fortran unit 25 (fort.25) or in separate files (e.g., BAND.DAT, IRSPEC.DAT, etc.). The list of properties and the corresponding name of the auxiliary files is listed in Table S1 in the Supporting Information.

Notably, because in crystalline systems, many properties are tensors and anisotropic, CRYSPLOT also allows the simultaneous plot of multiple datasets as for the simulated Raman spectra of a single crystal model for which six sets of data are available for the six independent orientation of the Raman tensor (i.e.,  $xx$ ,  $yy$ ,  $zz$ ,  $xy$ ,  $yz$ ,  $xz$ ) as will be shown later on.

The "Plot settings" lateral menu (hidden on the left of the CRYSPLOT web page) contains a palette of options for customizing the plot shown on the screen. CRYSPLOT has independent pages for every property and these pages have a specific "Plot settings" set of options.

For instance, Figure 3 shows an example of the "plot settings" menu for the band structure. Some of the available options to modify the plot are described in the following:

- *Legend* allows one to show or hide the legend,
- *Font size* permits to decide the size of the plot and axis titles font, selecting from four sizes,
- *Plot title* permits to change the chart title,
- *Tick labels* allows to change the labels of the  $k$  points on the  $x$ -axis,
- *Band labels* allows users to change the name of each band (i.e., electronic level),
- *Line type* permits to decide which type of line is adopted for plotting the bands, namely: plain line, only markers, or both,
- *Y-axis unit* controls the  $y$ -axis unit to be either Hartree (default) or eV,
- *Different layout for open shell case* enables user to change the appearance of the plot to have the band structure for alpha and beta electrons on the same graph or on two separate plots,
- *Shifting Y-axis* allows to shift values for the value of Fermi Energy,

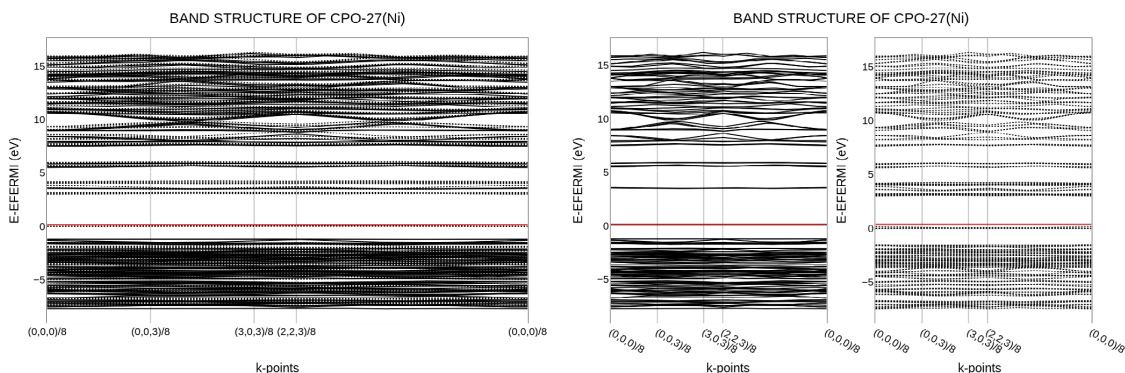
- *Fermi Energy line* allows one to show or hide the line of Fermi Energy,
- *Axis range* allows to selected a sub-region of the plot by axis values,
- *Data on hover* allows to change the visualization of data on the plot
- *Grid* allows one to show or hide the gray grid of the plot
- *Background color* modifies the color of the plot background from gray (default) to white
- *Plot layout* allows to select the size of the plot

In the case of the density of states, users can change line colors, and for spin polarized systems, one can decide whether painting  $\alpha$  and  $\beta$  lines in pairs or with different colors. Also, total and projected density of states can be plotted either on the same graph or on different charts.

Finally, at the bottom of the page there is an "Export" section to save the actual plot as a picture. After selecting the file format (i.e., jpeg, png, and svg), users must simply write the plot name and click on "Download Plot."

#### CRYSPLOT: Under the hood

The programming language used for CRYSPLOT is javascript. Our code parses the input files, checks if the uploaded files are correct and, if they are, it reads the data and organizes them into objects ready to be plotted with plotly.js library. A peculiar feature of CRYSPLOT is to keep separate the import code from the plotting one. Indeed, different import filters for each supported file format have been created, but the code that transfers data to plotly.js is always the same. When importing the data, the code checks whether the file format is correct, if not, an error message appears. After data have been re-organized, it is straightforward to visualize them through the common code for plotting. For instance, data for band structure can be imported either from BAND.DAT (or filename.BAND) file or the Fortran unit fort.25 (or filename.f25). In that case, the code has two different import functions: one for .BAND or .DAT format and another one for fort.25. Therefore, even if data are organized and formatted very differently, after importing them,



**Figure 4.** Example of a band structure plot for the metal–organic framework CPO-27-Ni. Alpha and beta electronic bands are represented as continuous and dashed lines, respectively. The Fermi level is indicated with a red line. [Color figure can be viewed at wileyonlinelibrary.com]

users get the same javascript object that can be passed to `plotly.js` through the general plotting code for the final representation of the graph.

This structure makes CRYSPLOT modular and flexible so that it is very simple to add other input data formats. In fact, it is just a matter of creating a new import filter that organizes the values in the proper way.

Finally, it is worth noting that CRYSPLOT does not need any third party libraries. It has been tested with most common browsers on different operating systems by using standard test cases of the CRYSTAL code, sample tests included in the CRYSTAL Tutorial project and on-purpose designed tests.

## Cryplot At Work: Selected Case Studies

In this section, metal–organic frameworks (MOFs) are used as case studies to show some of the capabilities of CRYSPLOT. MOFs are a relatively new class of hybrid inorganic–organic materials that are comprised an inorganic cluster (or metal) and an organic linker acting as secondary building units of a tridimensional and usually porous framework.<sup>[16]</sup> Due to their peculiar structure, MOFs are interesting for several applications ranging from gas adsorption, separation and capture to catalysis and drug delivery. Recently, they have become of interest for other technologically relevant applications such as sensing, lighting, and optoelectronics.<sup>[17–19]</sup>

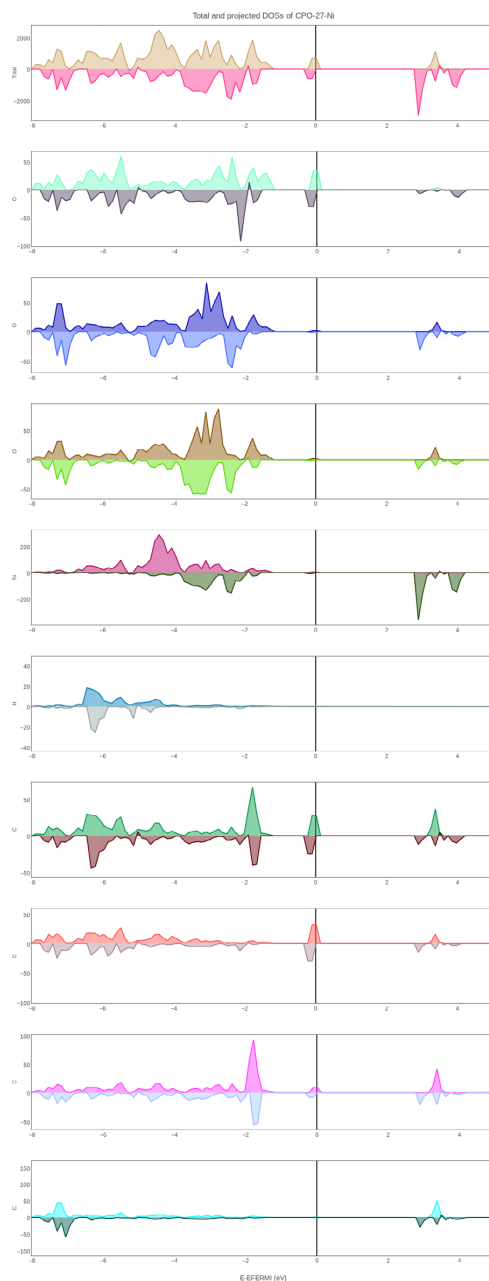
### Band structure and density of states

As a first case study, we discuss the electronic structure of CPO-27-Ni. This MOF consists of metal-containing helical chains connected throughout space by the organic linker (i.e., 2,5-dihydroxyterephthalic acid) to form a honeycomb-like framework.<sup>[20,21]</sup> The resulting structure shows one-dimensional channels in which unsaturated metal sites are exposed at the inner surface. Due to the presence of unpaired electrons on the metal, CPO-27-Ni is a spin-polarized system. Results refer to a ferromagnetic configuration with Ni in a high-spin state. Reported data have been obtained with the M06-D method<sup>[22]</sup> with a TZVP basis set.<sup>[23]</sup>

In Figure 4, the band structure of CPO-27-Ni is shown. CRYSPLOT can automatically manage spin-polarized systems by plotting band structure for alpha and beta electrons together or separately in two graphs.

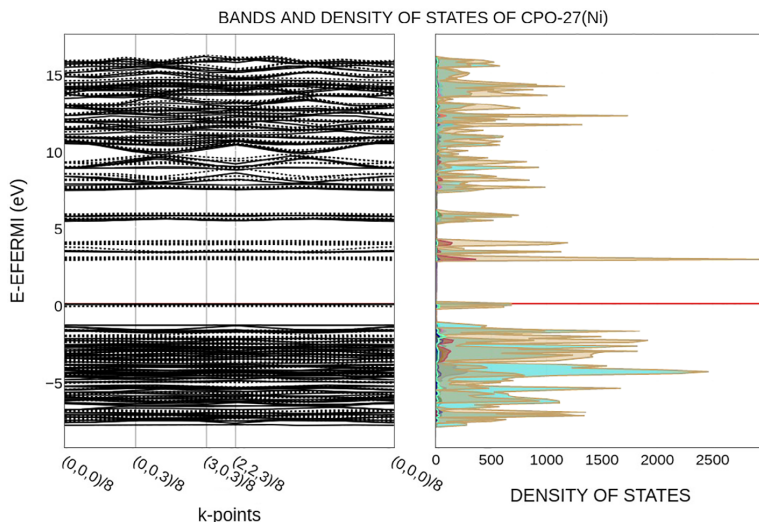
The total and the projected density of states have been plotted for all atoms in the asymmetric unit (i.e., nine projected DOSs) is shown in Figure 5. As it can be seen, the colored filled area plot is very effective in highlighting the role of alpha and beta electrons and multiple plots allow users to compare total and projected DOSs.

With CRYSPLOT, band structure and density of states can be also easily visualized in a combined plot. Figure 6 shows the band structure of CPO-27-Ni along with its total and projected density of states, that is, the combination of Figures 4 and 5. By default, the two plots are aligned to the Fermi level at 0 eV. From this graph, users can easily visualize which atom contributes to the corresponding electronic band. For instance, for CPO-27-Ni, it can be readily seen that the top of the valence



**Figure 5.** Example of total and projected density of states plot for CPO-27-Ni. Projected DOSs are plotted for all non-symmetry related atoms in the unit cell. [Color figure can be viewed at [wileyonlinelibrary.com](http://wileyonlinelibrary.com)]

band corresponds to orbitals of the organic linker, whereas the bottom of the conduction band is mainly dominated by the metal, thus suggesting a possible ligand-to-metal electronic transition.

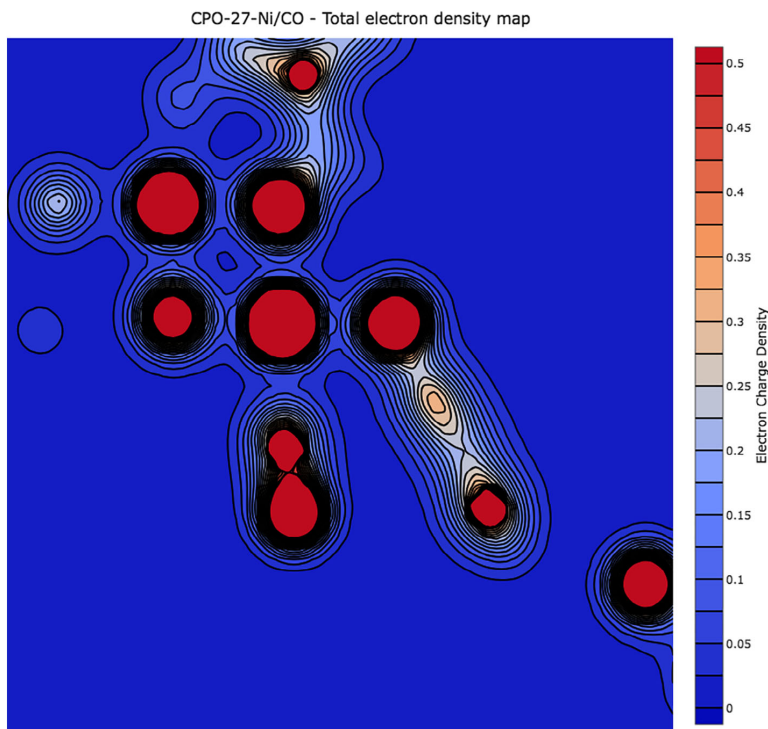


**Figure 6.** Example of a combined band structure and density of states plot for CPO-27-Ni. [Color figure can be viewed at [wileyonlinelibrary.com](http://wileyonlinelibrary.com)]

### Electron charge density maps

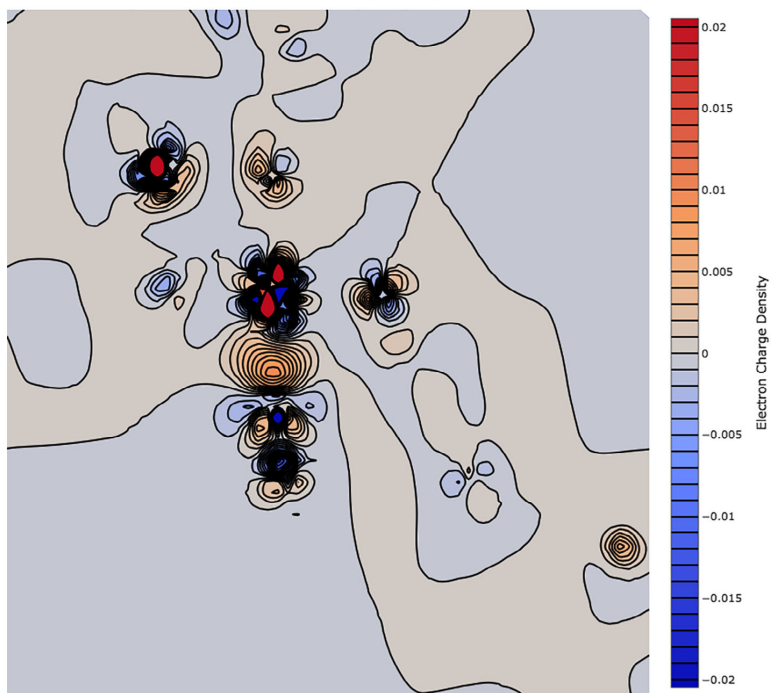
As an example of the plotting of 2D maps, the total electron density of a carbon monoxide molecule adsorbed on the inner surface of CPO-27-Ni is shown in Figure 7. Results have been computed at the M06-D/TZVP level of theory. The CO molecule strongly interacts with the Ni atoms that are exposed in the one-dimensional

channels oriented along the *c*-axis.<sup>[24]</sup> The interaction is dominated by electrostatics and charge transfer effects due to the back-donation between the diffuse molecular orbitals of carbon monoxide and the unoccupied *d*-orbitals of the metal. To better appreciate the redistribution of the charge density upon the formation of the metal-CO interaction, it is then useful to plot the deformation charge density as shown in Figure 8.



**Figure 7.** Contour plot of the total electron charge density map of a CO molecule adsorbed in the inner channels of CPO-27-Ni. [Color figure can be viewed at [wileyonlinelibrary.com](http://wileyonlinelibrary.com)]

CPO-27-Ni/CO - Electron charge density difference map



**Figure 8.** Plot of the electron charge density deformation map of CO molecule adsorbed in CPO-27-Ni. Blue and red lines correspond to negative and positive values, respectively. [Color figure can be viewed at [wileyonlinelibrary.com](http://wileyonlinelibrary.com)]

CRYSPLOT can easily manage multiple sets of data and combine them to obtain a new map that corresponds to the difference (or sum) of the uploaded datasets. For instance, Figure 8 shows the deformation of the charge density when a CO molecule is adsorbed on top of the metal in CPO-27-Ni. The plot is obtained as a combination of the total electron charge densities of three systems, namely: the target system (i.e., CPO-27-Ni/CO) and the two separated subsystems (i.e., an array of CO molecules and the CPO-27-Ni alone). The electron density deformation map in Figure 8 shows positive values, in red, that correspond to charge accumulation while negative values, in blue, show where a depletion of charge occurs. As evident from Figure 8, upon interaction the electron charge density of the CO molecule is polarized toward the metal with a significant charge transfer between CO and Ni.

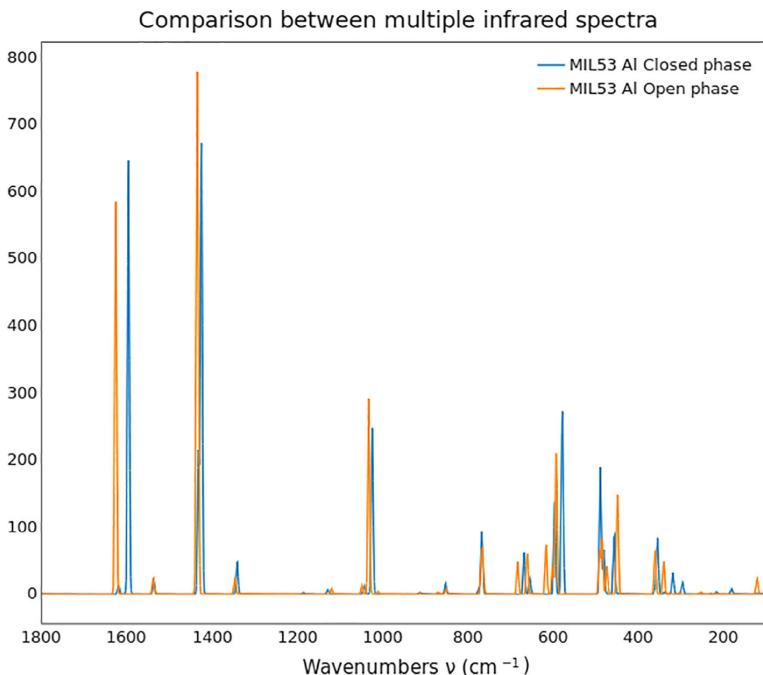
### Simulated vibrational spectra

To show the capabilities of CRYSPLOT in plotting vibrational spectra, we compare the two phases of the bi-stable MOF known as MIL-53-Al.<sup>[25,26]</sup> The MIL-53 family of MOFs shows a flexible framework that can undertake a phase transition driven by external stimuli as temperature, pressure, and gas adsorption.<sup>[27]</sup> The structure of MIL-53-Al consists of infinite chains of corner-sharing  $\text{AlO}_4(\text{OH})_2$  octahedra linked through the organic ligands (i.e., 1,4-benzendicarboxylic acid) to form flexible one-dimensional channels.<sup>[25]</sup> For MIL-53-Al, it was shown that, by

changing the temperature, a phase transition occurs between a low-T narrow-pore (NP) structure and a high-T large-pore (LP) phase.<sup>[25,26]</sup>

It has been recently shown that the two phases exhibits unique vibrational fingerprints from dielectric function spectra<sup>[28]</sup> and IR/Raman spectra.<sup>[29]</sup> With CRYSPLOT simulated IR and Raman spectra as computed by CRYSTAL can be easily visualized. In addition, up to five IR (or Raman) spectra can be plotted together to highlight differences in the vibrational modes and distinguish between different phases. This is very useful when comparing vibrational spectra of different polymorphs as for molecular crystals, or different phases as for the case of MIL-53-Al.

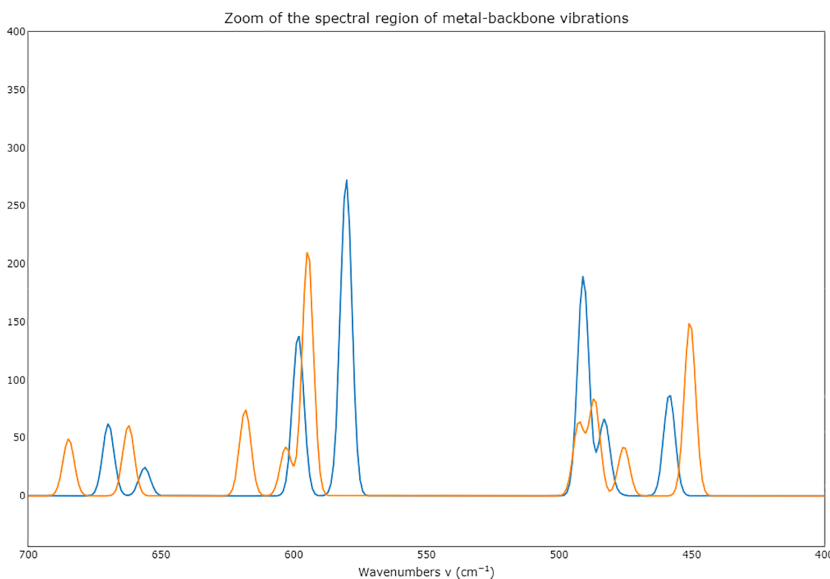
In Figures 9 and 10, the comparison between the simulated IR spectra, respectively, of the NP and CP phases of MIL-53-Al is shown. Results have been obtained with the B3LYP functional<sup>[30–32]</sup> augmented with the Grimme's D3 dispersion correction<sup>[33,34]</sup> in combination with a triple-zeta quality basis set.<sup>[35]</sup> It can be clearly seen that the two phases show several spectral shifts of the peaks from a few to 20 wavenumbers. In particular, by zooming into the plot with CRYSPLOT one can also visualize in more detail the spectral regions that show the largest variations because they can reveal the presence of vibrational fingerprints. For instance, in Figure 10, the IR region between 400 and 700  $\text{cm}^{-1}$  of the two phases is highlighted. According to a recent work by Hoffman et al.,<sup>[16]</sup> this corresponds to the region of metal-oxide backbone vibrations that are more influenced by the phase transition.



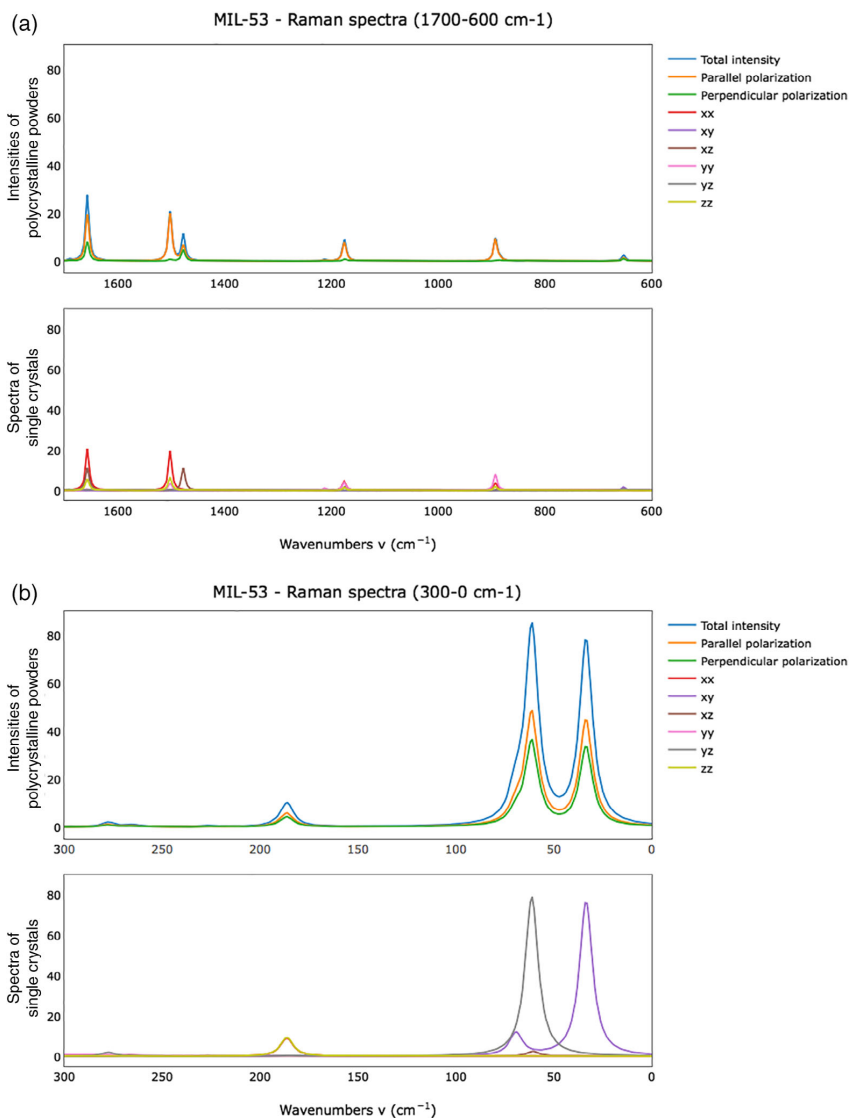
**Figure 9.** Comparison between IR spectra of the NP (blue) and LP (orange) phases of MIL-53-Al. [Color figure can be viewed at [wileyonlinelibrary.com](http://wileyonlinelibrary.com)]

As an additional example, the simulated Raman spectra of MIL-53 large pore phase are shown in Figure 11. CRYSTAL permits to simulate the Raman spectra of solids either as a polycrystalline powder or a single crystal.<sup>[36,37]</sup> In the former case, the total spectrum is predicted along with two components in the parallel and perpendicular directions. For the latter, the six spectra originate from the corresponding independent components of the second-order

electric susceptibility tensor (i.e.,  $xx$ ,  $xy$ ,  $xz$ ,  $yy$ ,  $yz$ ,  $zz$ ). For instance, Figure 11 has been obtained by using the unscaled vibrational frequencies and a Lorentzian profile with a FWHM of  $8\text{ cm}^{-1}$ . For this kind of plot, CRYSPLOT permits to visualize all spectra simultaneously. A nice feature is that by clicking on the items in the legend, one can hide (or show) the corresponding spectrum in the plot. This option allows users to analyze the data and highlight the Raman



**Figure 10.** Details of the spectral region between  $400$  and  $700\text{ cm}^{-1}$  of the IR spectra of the NP (blue) and LP (orange) phases of MIL-53-Al. [Color figure can be viewed at [wileyonlinelibrary.com](http://wileyonlinelibrary.com)]



**Figure 11.** Raman spectra of the LP phase of MIL-53-Al in the region (top) 1700–600  $\text{cm}^{-1}$  and (bottom) 300–0  $\text{cm}^{-1}$ . The two graphs show the simulated Raman spectra as for a polycrystalline material (three components) and a single crystal (six components), respectively. See text for details. [Color figure can be viewed at [wileyonlinelibrary.com](http://wileyonlinelibrary.com)]

spectrum along a given direction. From Figure 11 (bottom), it is evident that the two peaks at very low frequency in the THz region (i.e., 300–0  $\text{cm}^{-1}$ ) are polarized in different directions, that is,  $xy$  and  $xz$ , thus making easier the comparison with single-crystal Raman spectra from experimental works. On passing, it is worthy to note that the simulated total Raman spectrum of MIL-53(LP) as a crystalline powder (see Fig. 11, top) nicely agrees with the experimental one reported by Hoffman et al.<sup>[16]</sup>

## Conclusions

CRYSPLIT is a user-friendly web-oriented tool that is freely available to CRYSTAL users. It allows the visualization of many

properties in a few seconds by reading information from auxiliary files created by CRYSTAL.

CRYSPLIT has been highly optimized with just a common function that is called multiple times and different import filters for each property to be plotted. In this respect, it is also highly modular because it can be easily modified to visualize new kinds of property that will be available in next release of the CRYSTAL code. Because CRYSPLOT reads data from formatted text files, it can be considered a versatile and general tool because, in principle, it is not limited to visualize data computed with CRYSTAL. Indeed, one could obtain results from another software, reorganize raw data according to the proper format (see the CRYSTAL User's Manual<sup>[7]</sup>), and then plot the results with CRYSPLOT.

CRYSPLOT is an easy-to-use web platform that has been designed as a very intuitive graphical tool, a low entry-level interface to all types of users: from beginners (i.e., students) to expert researchers. It is also a sufficiently advanced and powerful tool that enables users to customize graphs in such a way that results could be used in scientific publications. This has been shown for selected applications taken from the study of metal-organic frameworks. Its ease-of-use makes CRYSPLOT also very useful for educational purposes.


From a general perspective, CRYSPLOT is under development and a bunch of other features will be included in future releases. It will be extended to visualize more computed properties (e.g., simulated X-ray powder diffraction and inelastic neutron scattering spectra<sup>[7]</sup>) and some facilities on the server side. In particular, we would like to provide each user with a personal page in which they can save and store plots created with CRYSPLOT. A more advanced graphical user interface for CRYSTAL is also under development with enhanced capabilities like a structure visualizer, a wizard to prepare input files and tools to manage calculations. When ready, CRYSPLOT will be fully encoded in the new graphical interface to enable users to promptly analyze and visualize computed properties.

## Acknowledgments

Giorgia Beata would like to acknowledge Regione Piemonte for supporting her PhD project (Avviso Pubblico approvato con D.D. 537 del 03/08/2016). Bartolomeo Civalleri thanks Lorenzo Donà for his help in running calculations on CPO-27-Ni.

**Keywords:** GUI · CRYSTAL · solid state · periodic systems

How to cite this article: G. Beata, G. Perego, B. Civalleri. *J. Comput. Chem.* **2019**, 9999, 1–10. DOI: 10.1002/jcc.25858

 Additional Supporting Information may be found in the online version of this article.

- [1] M. Valle, *Int. J. Quantum Chem.* **2013**, 113, 2040.
- [2] R. Dovesi, R. Orlando, B. Civalleri, C. Roetti, V. R. Saunders, C. M. Zicovich-Wilson, *Z. Kristallogr.* **2005**, 220, 571.
- [3] R. Dovesi, V. R. Saunders, C. Roetti, R. Orlando, C. M. Zicovich-Wilson, F. Pascale, B. Civalleri, K. Doll, N. M. Harrison, I. J. Bush, P. D'Arco, M. Llunell, CRYSTAL09 User's Manual, University of Torino, Torino, **2009**.
- [4] 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. Rerat, B. Kirtman, *Int. J. Quantum Chem.* **2014**, 114, 1287.
- [5] R. Dovesi, V. R. Saunders, C. Roetti, R. Orlando, C. M. Zicovich-Wilson, F. Pascale, B. Civalleri, K. Doll, N. M. Harrison, I. J. Bush, P. D'Arco, M. Llunell, M. Causà, Y. Noël, CRYSTAL14 User's Manual, University of Torino, Torino, **2014**.
- [6] R. Dovesi, A. Erba, R. Orlando, C. M. Zicovich-Wilson, B. Civalleri, L. Maschio, M. Rerat, S. Casassa, J. Baima, S. Salustro, B. Kirtman, *WIREs Comput. Mol. Sci.* **2018**, 8, e1360.
- [7] R. Dovesi, V. R. Saunders, C. Roetti, R. Orlando, C. M. Zicovich-Wilson, F. Pascale, B. Civalleri, K. Doll, N. M. Harrison, I. J. Bush, P. D'Arco, M. Llunell, M. Causà, Y. Noël, L. Maschio, A. Erba, M. Rerat, S. Casassa, CRYSTAL17 User's Manual, University of Torino, Torino, **2017**.
- [8] R. Orlando, M. Delle Piane, I. J. Bush, P. Ugliengo, M. Ferrabone, R. Dovesi, *J. Comput. Chem.* **2012**, 33, 2276.
- [9] A. Erba, J. Baima, I. Bush, R. Orlando, R. Dovesi, *J. Chem. Theory Comput.* **2017**, 13, 5019.
- [10] S. Casassa, A. Erba, J. Baima, R. Orlando, *J. Comput. Chem.* **2015**, 36, 1940.
- [11] T. Williams, C. Kelley, H.-B. Broker, John Campbell, R. Cunningham, D. Denholm, G. Elber, R. Fearick, C. Grammes, L. Hart, L. Hecking, P. Juhasz, T. Koenig, D. Kotz, E. Kubaitis, R. Lang, T. Lecomte, A. Lehmann, A. Mai, B. Markisch, Ethan A Merritt, P. Mikulik, C. Steger, S. Takeno, T. Tkacik, J. Van der Woude, J. R. Van Zandt, A. Woo, J. Zellner, Gnuplot 5.2.2: An Interactive Plotting Program **2017**.
- [12] B. G. Searle, *Comp. Phys. Comm.* **2001**, 137, 25.
- [13] A. Kokalj, *Comp. Mater. Sci.* **2003**, 28, 155. Code available from: <http://www.xcrysden.org/>.
- [14] Plotly Technologies Inc, Collaborative Data Science, Montréal, QC, **2015**. <https://plot.ly>.
- [15] R. Gaillac, P. Pullumbi, F.-X. Coudert, *J. Phys. Condens. Matter* **2016**, 28, 275201.
- [16] J. R. Long, O. M. Yaghi, *Chem. Soc. Rev.* **2009**, 38, 1213.
- [17] H. Furukawa, K. E. Cordova, M. O'Keeffe, O. M. Yaghi, *Science* **2013**, 341, 1230444.
- [18] J. C. Tan, B. Civalleri, *CrstEngComm* **2015**, 17, 197.
- [19] G. Maurin, C. Serre, A. Cooper, G. Férey, *Chem. Soc. Rev.* **2017**, 46, 3104.
- [20] N. L. Rosi, J. Kim, M. Eddaoudi, B. Chen, M. O'Keeffe, O. M. Yaghi, *J. Am. Chem. Soc.* **2005**, 127, 1504.
- [21] P. D. C. Dietzel, B. Panella, M. Hirscher, R. Blom, H. Fjellvag, *Chem. Commun.* **2006**, 959.
- [22] Y. Zhao, D. G. Truhlar, *Theor. Chem. Acc.* **2008**, 120, 215.
- [23] M. F. Peintinger, D. Vilela Oliveira, T. Bredow, *J. Comput. Chem.* **2013**, 34, 451.
- [24] L. Valenzano, B. Civalleri, K. Sillar, J. Sauer, *J. Phys. Chem. C* **2011**, 115, 21777.
- [25] Y. Liu, J.-H. Her, A. Dailly, A. J. Ramirez-Cuesta, D. A. Neumann, C. M. Brown, *J. Am. Chem. Soc.* **2008**, 130, 11813.
- [26] A. M. Walker, B. Civalleri, B. Slater, C. Mellot-Draznieks, F. Corà, C. M. Zicovich-Wilson, G. Ramon-Perez, J. M. Soler, J. D. Gale, *Angew. Chem. Int. Ed.* **2010**, 49, 7501.
- [27] C. Serre, S. Bourrelly, A. Vimont, N. A. Ramsahye, G. Maurin, P. L. Llewellyn, M. Daturi, Y. Filinchuk, O. Leynaud, P. Barnes, G. Férey, *Adv. Mater.* **2007**, 19, 2246.
- [28] K. Titov, Z. Zeng, M. R. Ryder, A. K. Chaudhari, B. Civalleri, C. S. Kelley, M. D. Frogley, G. Cinque, J. C. Tan, *J. Phys. Chem. Lett.* **2017**, 8, 5035.
- [29] A. E. J. Hoffman, L. Vanduyfhuys, I. Nevjestic, J. Wieme, S. M. J. Rogge, H. Depauw, P. Van Der Voort, H. Vrielinck, V. Van Speybroeck, *J. Phys. Chem. C* **2018**, 122, 2734.
- [30] A. D. Becke, *J. Chem. Phys.* **1993**, 98, 5648.
- [31] C. Lee, W. Yang, R. G. Parr, *Phys. Rev. B* **1988**, 37, 785.
- [32] B. Miehlich, A. Savin, H. Stoll, H. Preuss, *Chem. Phys. Lett.* **1989**, 157, 200.
- [33] S. Grimme, J. Antony, S. Ehrlich, H. Krieg, *J. Chem. Phys.* **2010**, 132, 154104.
- [34] S. Grimme, S. Ehrlich, L. Goerigk, *J. Comput. Chem.* **2011**, 32, 1456.
- [35] A. Schäfer, H. Horn, R. Ahlrichs, *J. Chem. Phys.* **1992**, 97, 2571.
- [36] L. Maschio, B. Kirtman, M. Rerat, R. Orlando, R. Dovesi, *J. Chem. Phys.* **2013**, 139, 164101.
- [37] L. Maschio, B. Kirtman, M. Rerat, R. Orlando, R. Dovesi, *J. Chem. Phys.* **2013**, 139, 164102.

Received: 6 March 2019

Revised: 18 April 2019

Accepted: 24 April 2019