Guide to Vaspirin 2.0

Daniel S. Koda[*]
Instituto Tecnológico de Aeronáutica
Group of Semiconductor Materials and Nanotechnology

February 2017

Abstract

Vaspirin is a pre- and post-processsing tool for VASP made in Python. Its goals are to simplify the analysis of simulations, make standard and beautiful figures easily and create a continuous development and centralization of the tool throughout the years for the Group of Semiconductor Materials and Nanotechnology (GMSN). This document describes the capabilities of the software and provides a beginner's guide to its utilization. We cover the installation of the script and demonstrate the use of this tool with examples, such as band structure and density of states plotting.

Contents

[*] `danielskoda@gmail.com`

1 Introduction

1.1 About Vaspirin

Origins: Vaspirin has been created to unify and document scripts made by members of the Group of Semiconductor Materials and Nanotechnology (GMSN) from Instituto Tecnológico de Aeronáutica (ITA), Brazil, in a way which benefits both experienced students with its pragmatic and well-tested approach as well as beginner undergraduate students by lessening the initial difficulties of juggling programming languages, solid state physics, undergraduate studies AND simulations at once.

Authors: Daniel S. Koda (MSc student in 2016–2017) and Ivan Guilhon (PhD student in 2016–2017) are responsible for the beginning of this project and the documentation in 2016. In 2017, Vaspirin was completely rewritten by Daniel S. Koda, giving rise to a second version of Vaspirin, more robust, customizable and complete.

Software: The raw code and the installation package can be found at: `https://`

`github.com/gmsn-ita/vaspirin.git`. New versions will be constantly updated in this link.

1.2 Installation

Requirements: Vaspirin uses Python 3 as standard language. The packages `numpy`, `matplotlib`, `scipy`, `pylab`, and `setuptools` are required in order to take full advantage of Vaspirin. XMGrace is also used to plot `.bfile` scripts generated by Vaspirin. `git` is necessary if you want to contribute to the software. Please use your package manager to install these packages before proceeding with the installation.

Installing: Directly download the Vaspirin code from `https://github.com/gmsn-ita/vaspirin.git` or simply execute:
`git clone https://github.com/gmsn-ita/vaspirin.git`.

Once downloaded, go to the directory `vaspirin/` and execute execute:
`# python setup.py install` or `$ sudo python setup.py install`

If Python 3 is not the default Python language in your operating system (confirm by executing `python --version`), you may want to try:
`# python3 setup.py install`

From this point on, you can execute Vaspirin scripts as normal command-line interface softwares.

2 Vaspirin architecture

From Vaspirin v2.0, the code was completely rewritten to be used as a high-level interface. Basically, the code is made of two parts: core sub-modules, packaged as a single `vaspirin` module, and scripts, which use the modules to execute user-friendly actions. This not only allows programmers to focus onto essential tasks such as reading data from VASP outputs without interfering into the user interface, but also makes scripts easier to be programmed, transferred, modified, tuned and used without worries. This architecture is also more robust to version changes and easier to debug, which makes it pretty much the best option to keep and maintain a complex code such as Vaspirin.

2.1 Core

The core submodule is composed by classes and methods created to digest VASP output and input files into comprehensible information. This series of classes is a toolbox for reading files such as OUTCAR, PROCAR, DOSCAR etc. Each submodule deals strictly with one kind of file as input and, at most, uses auxiliary functions from the other classes.

`outcar`: The `outcar` submodule reads the VASP OUTCAR file, storing information such as band structure, number of bands/electrons, energy gap etc. It is mainly used for plotting band structures.

`procar`: The `procar` submodule reads the VASP PROCAR file, storing information such as band composition, projection onto atomic orbitals and sites. When large cells and heavy calculations are performed, this class is incapable of dealing with these large files. In this case, the `splitter` submodule can be used. It is used for plotting projected band structures.

`poscar`: The `poscar` submodule contains information related to the VASP POSCAR file. It is used as a preprocessing tool, such as straining, moving, rotating etc.

`doscar`: The `doscar` submodule reads the VASP DOSCAR file, storing information of total and projected density of states (DOS). It is used for plotting DOS plots.

`projection`: The `projection` submodule is a user-interface information reader to configure projection onto atomic sites. It allows color and material personalization.

`splitter`: The `splitter` submodule splits PROCAR files into its `.dat` files without opening the entire file at once. This allows bypassing the `datIO` submodule and, therefore, enables big PROCAR files to be broken down into their smaller `.dat` files, compatible with other Vaspirin scripts.

`datIO`: The `datIO` submodule converts information of band structures, projections and DOS into `.dat` files. It also has some extra features, such as marker size personalization and interpolation within band structures.

`graceIO`: The `graceIO` submodule processes `.dat` files generated by the `datIO` module and turns them into beautiful plots.

2.2 Scripts

Scripts are the executors of the tasks in Vaspirin. They contain the user interface and make use of the toolbox provided by the core. By correctly manipulating the information extracted by the core submodules, scripts easily create band structures, density of states and many other figures just with command-line mechanisms. This allow them to be extremely flexible, as well as reusable and robust to debugging and updates. Several possibilities may be designed to complete and automatize tasks within quotidian interfaces, which allow users to create their own configurable script to supply their specific demands.

Parts of a Vaspirin script: Any Vaspirin script is composed by:

- an user-interface section, presented in the form of an argument parser and descriptions;
- several functions to accomplish the goal of the script;
- a main function, sewing together all patches of code into one execution.

Anyone can create a Vaspirin script. If you fell like some script is needed by you, you can study the technical documentation on how the Vaspirin submodules work and then create your own script using one of the scripts already done as an example. If you improve the code, please contribute to its evolution by commiting it to GitHub.

3 Running Vaspirin

Introduction: Vaspirin was created with an eye to flexibility. Therefore, most of its features are activated by running the script with flags starting with a dash `-`. Each script has its individual set of flags to be used and configured on demand. All possibilities are briefly listed here and demonstrated in the Tutorial section. First of all, manual entries within the framework of Vaspirin will be explained. Then, the function of each script will be described here, as well as its options. Detailed explanation of their uses will be illustrated in the Tutorial section. In case of doubt while dealing with the script, a help command is available when the flag `-h` is set.

3.1 Standards requiring manual entry

`KPOINTS` header:	To create a band structure specifying symmetry k-points, a header describing which are these special k-points should be written in the `KPOINTS` file. The header has the following format: `SymPoint1 Index1, SymPoint2 Index2, SymPoint3 Index3, ...` A specific example is shown in section 4.1. The script `gen_kpoints.py` automatically generates this header when used.
`PROJECTION` file:	Projected band structures are creating by summing all contributions from atoms which belong to specified materials. Therefore, which atoms are part of each material is an information supplied by the user. The `PROJECTION` file is responsible for conveying that data to `vaspirin`. It is a plain text file with the following format: `Material1Label atomsBelongingToMaterial1` `Material2Label atomsBelongingToMaterial2` ⋮ An example of application of this technique is shown in section 4.3.
`.dat` files:	These files are output from the `datIO` submodule and contain information about the simulation loaded. For band structures (`eigenv.dat`), this file is a double-column table, in which the first column is the normalized k-point (based on the length and the size of the Brillouin zone) and the second is the corresponding eigenvalue. Bands are organized in blocks. In cases such as projection and orbital character, a folder named `bands_character` or `bands_projected` is created, and one `.dat` file is written for each band. This allows plotting special band structures with XMGrace.
`.bfile` files:	This kind of file contain batch instructions for plotting data with XMGrace. They should be executed with this software, for example: `xmgrace -batch bands.bfile`

3.2 General flags for most scripts

`-h` (help)	Shows the help message and exit the software
`-o` (output)	Output file to be written. Its extension must be specified whenever possible.
`-q` (quiet)	Do not display text on the output window
`-v` (version)	Displays the program version number

3.3 Script `band_offsets`

The **`band_offsets`** script is an automated way to draw band offsets using Python. Instead of drawing everything by hand, one may simply input a band alignment with respect to the vaccum and then see the figure drawn as he/she wants.

`input_file` The **`band_offsets`** script requires an input file, defined as ALIGNMENTS as default. This file is a plain text file, which defines the valence band maximum (VBM) and conduction band minimum (CBM) for each material, as well as the dividers and joint lines. The formatting is similar to:

```
Material1Label VBM_1 CBM_1 color_1
divider_sign
Material2Label VBM_2 CBM_2 color_2
⋮
```

The divider sign customize the line style joining together the band alignments. It may be selected between: ’-’ (solid line), ’–’ (dashed line), ’-.’ (dashed-dotted line), ’:’ (dotted line), ’None’, ’ ’ (spaces), ”. Examples of use for this file are provided in the Tutorial section.

`-v` (vacuum) defines the dipole vacuum step (feature not yet implemented).

`-x` (axis) Turns on the axis for the plot (default: False).

`-s` (show) Show the band alignments figure before saving it to the file.

`-n` (number) Number of materials composing the interface (Default: 2). Useful for creating interfaces with more than 2 materials and avoiding the cluttering of labels to interfere with it.

3.4 Script `colored_bands`

`colored_bands` is a post-processing script used to create color gradients onto projected band structures. This allows one to easily follow the contributions from each atomic site to the final band structure.

`-i` (input folder) The `colored_bands` script requires an input folder, which contains the information of projected bands onto two different materials. This folder can be obtained by using the scripts `split_procar` and `plot_bands` together with their optional projections onto atomic sites.

`-b` (bands) Select the range of bands to be plotted. Usually, plotting a large number of bands may be computationally costly, while most bands may not be of interest for the analysis. This flag allows one to bypass these waiting times.

`-x` (x axis) Set the x axis range for the band structure

`-y` (y axis) Set the y axis range for the band structure

`-z` (size) Set the output figure size (in inches)

`-s` (show) Show the band structure figure after saving it to the file.

`-l` (legend) Include a colored legend bar by the figure’s side.

3.5 Script `dos`

`dos` is a post-processing script used to create density of states (DOS) and projected density of states onto atomic orbitals and sites. It reads the density of states from the `DOSCAR` file and projected information, when necessary.

`-o` (orbital)	Generate density of states projected onto atomic orbitals (s,$p_x + p_y$, p_z, d).
`-p` (projected)	Generate density of states projected onto atomic sites (specified within the PROJECTION file).
`-f` (fill)	Fill below the lines of the DOS in order to create a more solid and beautiful plot.
`-y` (y axis)	Set the energy axis range for the plot. Although the energy axis is the x-axis for the DOS plot, the compatibility with band plots is improved if the notation is preserved. Also, energy ends with an Y, which is kind of mnemonic.
`-d` (DOS axis)	Set the maximum value for the DOS axis within the plot. The default minimum is always 0.
`-r` (reference)	Set the reference for the DOS. The default is the Fermi level.

3.6 Script `plot_bands`

`plot_bands` is a post-processing script used to create plain band structures, as well as projected band structures onto atomic orbitals and sites. It reads the eigenvalues from the `OUTCAR` file and projected information from the `PROJECTION` file, when necessary.

`-o` (orbital)	Generate band structures projected onto atomic orbitals (s,$p_x + p_y$, p_z, d).
`-p` (projected)	Generate band structures projected onto atomic sites (specified within the PROJECTION file).
`-i` (ignore)	Ignore the first specified k-points when plotting band structures. Useful for HSE06 calculations, in which the `KPOINTS` file must include k-points with zero weight for band calculations.
`-t` (interpolate)	Interpolate k-points between each pair of k-points via a cubic interpolation. This helps to smoothen the bands when k-point sampling is not adequate enough.
`-m` (marker)	Select the marker size for the plot. Allows bigger and smaller markers to be written into graphs.
`-f` (fill)	Fill the band structure symbol markers in order to create a more solid and beautiful plot.
`-y` (y axis)	Set the energy axis range for the plot (y axis).
`-r` (reference)	Define an arbitrary reference for the 0 eV level in band structures. Recognized arguments are: `vbm`, `efermi`, `e-fermi` and any floating-point number. Default: `vbm`.
`-s` (spin-orbit)	Set whether this file comes from a non-collinear calculation or not. In future implementations, this could be easily done automatically.

3.7 Script `plot_compared_bands`

`plot_compared_bands` is a post-processing script used to create band structures compared with each other. It reads each band from its respective folder, in which `OUTCAR` files can be found.

`-i` (ignore)	Ignore the first specified k-points when plotting band structures. Useful for HSE06 calculations, in which the `KPOINTS` file must include k-points with zero weight for band calculations.
`-t` (interpolate)	Interpolate k-points between each pair of k-points via a cubic interpolation. This helps to smoothen the bands when k-point sampling is not adequate enough.
`-c` (color)	Select the colors for each band structure to be plotted in the comparison.
`-y` (y axis)	Set the energy axis range for the plot (y axis).
`-r` (reference)	Define an arbitrary reference for the 0 eV level in band structures. Recognized arguments are: `vbm`, `efermi`, `e-fermi` and any floating-point number. Default: `vbm`.
`-s` (spin-orbit)	Set whether this file comes from a non-collinear calculation or not. In future implementations, this could be easily done automatically.

3.8 Script `split_procar`

`split_procar` is a post-processing script used to split `PROCAR` files without having to import it to the RAM memory. This allows one to deal with big `PROCAR` files without running out of memory. The script is more slow, but it also plots the band structures accordingly.

`-o` (orbital)	Split `PROCAR` files and generate band structures projected onto atomic orbitals (s,$p_x + p_y$, p_z, d).
`-s` (split)	Split `PROCAR` files and generate band structures projected onto atomic sites (specified within the PROJECTION file).
`-i` (ignore)	Ignore the first specified k-points when plotting band structures. Useful for HSE06 calculations, in which the `KPOINTS` file must include k-points with zero weight for band calculations.
`-m` (marker)	Select the marker size for the plot. Allows bigger and smaller markers to be written into graphs.
`-f` (fill)	Fill the band structure symbol markers in order to create a more solid and beautiful plot.
`-y` (y axis)	Set the energy axis range for the plot (y axis).
`-r` (reference)	Define an arbitrary reference for the 0 eV level in band structures. Recognized arguments are: `vbm`, `efermi`, `e-fermi` and any floating-point number. Default: `vbm`.

3.9 Script `gen_kpoints`

`gen_kpoints` is a pre-processing script used to create `KPOINTS` paths in the reciprocal space. It already has a database of Bravais lattices in order to help the user only input a k-point symmetry path and then use the generated file forthwith. Created specially for HSE06 calculations, in which band structures must have personalized KPOINTS files.

-o (output)	Output name for the generated files.
-n (number)	Number of k-points between symmetry points.
-p (proportional)	Creates a k-point path with the number of points between symmetry points proportional to the length of the path. This allows a more reasonable sampling to be made, at the cost of more computational effort.
-i (IBZKPT)	Include a IBZKPT file at the beginning of the document.
-l (lattice)	Select the Bravais lattice to import the symmetry points dictionary.
-w (weigth)	Weight given to the k-points path created.

3.10 Script move_atoms

move_atoms is a pre-processing script used to manipulate POSCAR files. It allows the creation of sequences of POSCAR files with ease to analyze the effects of moving atoms inside the cell in many calculations. Pretty useful for molecules and van der Waals heterostructures.

-o (output)	Output name for the generated files.
-d (displacement)	The amount to displace the selected set of atoms
-s (step)	Step for the sequence of displacements
-m (atom)	Set of atoms to be displaced in the POSCAR files.
-x (axis)	The direction in which the atoms will be displaced

3.11 Script rotate_molecule

rotate_molecule is a pre-processing script used to manipulate POSCAR files. It allows the creation of sequences of POSCAR files with ease to analyze the effects of rotating atoms inside the cell in many calculations. Pretty useful for molecules.

-o (output)	Output name for the generated files.
-a (angles)	The amount to rotate the selected set of atoms (in degrees).
-s (step)	Step for the sequence of rotations.
-m (atom)	Set of atoms to be rotated in the POSCAR files.
-x (axis)	The axis around which the atoms will be rotated
-r (reference)	The atom which will be static, i.e. will be the origin of the coordinate system.

3.12 Script strain_cell

strain_cell is a pre-processing script used to manipulate POSCAR files. It allows the creation of sequences of POSCAR files with ease to analyze the effects of straining the cell in many calculations.

-o (output)	Output name for the generated files.

`-s` (strain)	The amount to strain the cell.
`-t` (step)	Step for the sequence of strains.
`-x` (x axis)	Allows the first lattice vector to be strained.
`-y` (y axis)	Allows the second lattice vector to be strained.
`-z` (z axis)	Allows the third lattice vector to be strained.

4 Tutorial

In this section, actual examples of calculations will be supplied in order to clarify the use of `vaspirin`. The files for this example are available in the folder `tests/` in the `vaspirin` directory. No VASP simulations will be done at this point.

4.1 Plotting standard band structures

Changing the directory: As a first step, the default configurations from Vaspirin scripts are used. This is the more favorable situation, in which the script is executed in the same folder as the VASP simulation. Therefore, all files have their default name and are all in the same directory. To start using Vaspirin scripts, please change your directory to that of interest.

`KPOINTS` header: The `KPOINTS` file must specify the path made in the band structure calculation, as well as their indexes. In this case, 130 k-points were used to form the entire path, distributed so that the Γ point were the 1st and the 130th points, the M point was the 41th point and the K point was the 70th point. Thus, the `KPOINTS` header should be:

```
G 0, M 40, K 69, G 129
```

Plotting: To plot a simple band structure using Vaspirin scripts, just execute:

```
plot_bands.py
```

This tells the script to acquire band structure data from the default `OUTCAR` file and then plot the correspondent bands. This generate the files `eigenv.dat` and `bands.bfile`.

Generating the bands using XMGrace: Finally, you can create the plot by executing:

```
xmgrace -batch bands.bfile
```

The result of these operations for the given example files is shown in Fig. 1.

Changing the y-axis scale: By default, the script puts the 0 eV reference on the top of the valence band. If the default range of [-3,3] eV is not enough, it is possible to customize these values simply by using the tag `-y`. Continuing the example above with this tag:

```
plot_bands.py -y -2 4
```

This new band structure is shown in Fig. 2.

It should be noted that the script does not care about the order in which the numbers after `-y` are written. Therefore, running `plot_bands.py -y 4 -2` would yield exactly the same result.

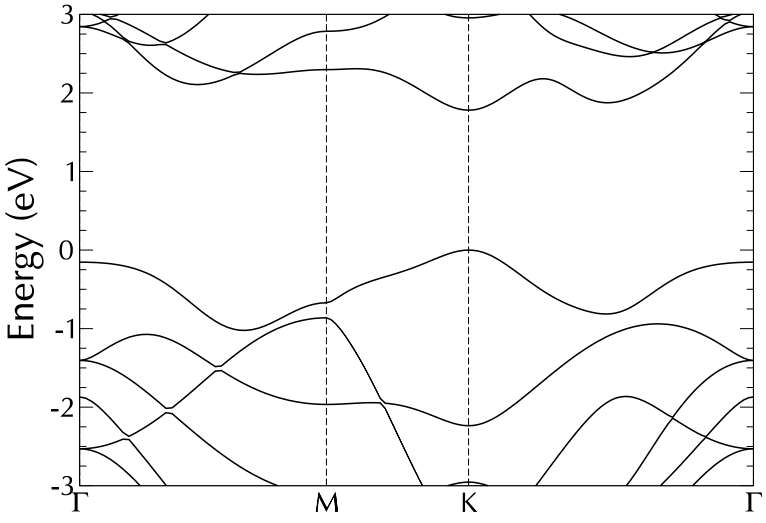

Figure 1: Example of a monolayer MoS_2 band structure plotted using `plot_bands.py`.

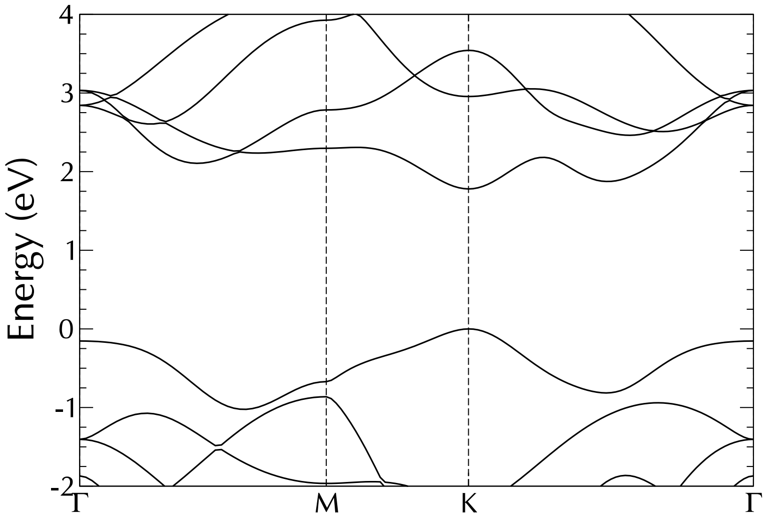

Figure 2: Example of a monolayer MoS_2 band structure plotted using `plot_bands.py -y -2 4`.

4.2 Plotting band structures projected onto orbitals

In many cases, it is useful to analyze the system based on the formation of the bands via orbitals. The Vaspirin toolkit allows this kind of analysis to be done. Creating this requires a `PROCAR` file, which has the contributions of all points from the band structure.

Generating band structures with orbital character: To create band structures with orbital character, `plot_bands.py` should be executed with the `-o` tag active:

```
plot_bands.py -o
```

This loads the default `OUTCAR` and `PROCAR` files from the current directory. The script outputs the data to the folder `bands_character/` and the XM-Grace batch `bandsCharacter.bfile`.

Plotting with XMGrace: To plot the resulting file with XMGrace, execute the following line:

```
xmgrace -batch bandsCharacter.bfile
```

The result is shown in Fig. 3.

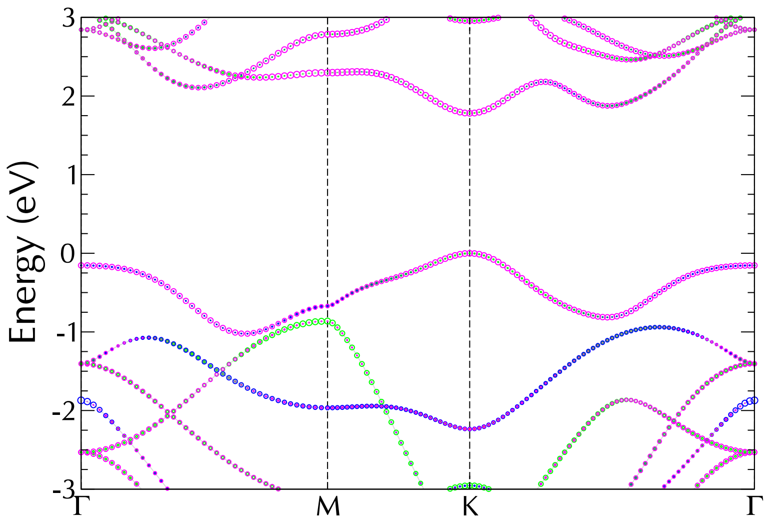

Figure 3: Example of a monolayer MoS_2 band structure plotted using `plot_bands.py -o`. Red, green, blue and magenta circles depict relative contributions from s, $p_x + p_y$, p_z and d orbitals, respectively, to the formation of the band.

Interpreting the figure: Orbital contributions scale the colored circular markers in the band structure. Therefore, bigger circles imply predominance of the specific orbital to the formation of the band exactly in that k-point. Red, green, blue and magenta circles depict contributions from s, $p_x + p_y$, p_z and d orbitals, respectively.

4.3 Plotting band structures projected onto specific ions

Sometimes, as in van der Waals heterostructures, it is useful to know which bands come from each of its constituents. This not only helps visualize effects on electronic structures but also creates a pleasant interpretation to the viewer.

Creating the `PROJECTION` file: The `PROJECTION` file is a user-made text file which specifies which atoms compose each material. In the case of the MoS_2, only two different atoms are simulated, and they are part of a single material. If we wanted to analyze contributions of the molybdenum atom to the bands formation, as well as the sulfur atom to it, we would have to build the following `PROJECTION` file:

```
Mo 1 red
S 2..3 blue
```

A 2×2 supercell of MoS_2, therefore with 12 atoms inside it, would have the following `PROJECTION` file:

```
Mo 1..4 red
S 5..12 blue
```

Another equivalent way of specifying the information above would be:

```
molybdenum 1,2,3,4 red
sulfur 5,6,7..12 blue
```

Atoms must be numbered according to their order in the `POSCAR` file and be separated by commas. The `7..12` entry indicates all atoms between the 7th and the 12th (including these both) belong to the material named `sulfur`. The labels can be chosen arbitrarily and is more like a guide for humans than a useful information. All atoms should be specified in the `PROJECTION` file. Finally, each label may receive a color to be used as its identifier in the plot.

Generating the data with the script: To use the simulation and the specification from the `PROJECTION` file, just run the following command:

```
plot_bands.py -p
```

A folder named `bands_projected` and a XMGrace batch named `bandsProjected.bfile` are created upon the execution of this command.

Plotting with XMGrace: To plot the resulting file with XMGrace, execute the following line:

```
xmgrace -batch bandsProjected.bfile
```

The result is shown in Fig. 4.

Interpreting the figure: Each material contributions scale the colored markers in the band structure. Therefore, bigger markers imply predominance of the specific material to the formation of the band exactly in that k-point. The materials are represented with red, green, blue, yellow, brown, gray, violet, cyan, etc., according to their color specified in the `PROJECTION` file.

4.4 Splitting `PROCAR` files

Suppose the `PROCAR` file specifying the projection content is huge. In this case, using the `plot_bands` script would not be enough to project files onto

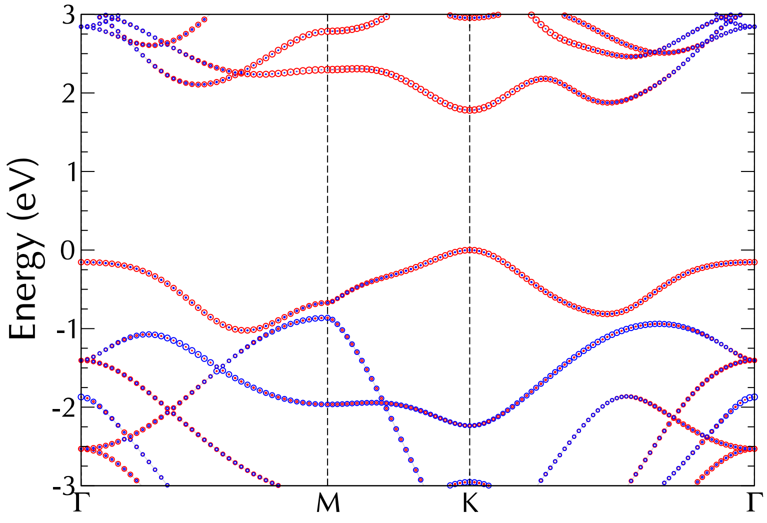

Figure 4: Example of a monolayer MoS_2 band structure plotted using `plot_bands.py -p`. Red and blue circles depict contributions from Mo and S atoms, respectively.

atomic sites or orbitals. To surmount that difficulty, the script `split_procar.py` may be used.

The `PROJECTION` file: The `PROJECTION` file is exactly the same from the section 4.3 and will be reused for this example.

Generating the data with `split_procar`: To use the simulation data and the specification from the `PROJECTION` file, just run the following command:
`split_procar.py -s`

A folder named `bands_projected` and a XMGrace batch named `bandsProjected.bfile` are created upon the execution of this command. Projected bands are identical to that seen in 4. If one wants to create orbital-projected bands (as in Fig. 3), simply run the following command:
`split_procar.py -os`, which is equivalent to `split_procar.py -o -s`

Understanding the `-s` flag: Since splitting large PROCAR files may take time, it is only enabled when the `-s` flag is set. Whenever this option is not specified, only the XMGrace `.bfile` will be created. Thus, users can change plotting options without having to split the PROCAR file again (and wait a lot of time for this task to be completed). However, if the atoms in the `PROJECTION` file are changed, i.e. new projections are specified, the `PROCAR` file must be split again.

4.5 Comparing two different band structures

Comparing two different band structures using Vaspirin scripts is an easy task. It still has a limitation and assumes that only bands executed with the same k-points path will be compared. Otherwise, results cannot have any meaning.

Comparing band structures: To compare band structures, the `plot_compared_bands` script should be executed as follows:
```
plot_compared_bands.py mos2/ ws2/
```

This loads the default `OUTCAR` files from each directory. The script outputs the data to two `.dat` files, nominally `eigenv1.dat` and `eigenv2.dat`, and the XMGrace batch to `bandsComparison.bfile`.

Plotting with XMGrace: To plot the resulting file with XMGrace, execute the following line:
```
xmgrace -batch bandsComparison.bfile
```

An example is shown in Fig. 5.

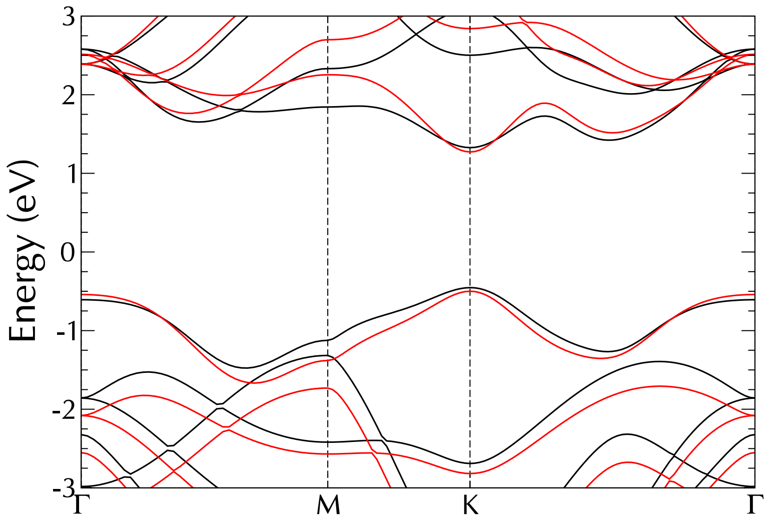

Figure 5: Example of a monolayer MoS_2 band structure compared with a WS_2 band structure plotted using `plot_compared_bands.py mos2/ ws2/`. Black lines are from the first band structure (in this case, MoS_2) and red lines are from the second band structure (WS_2).

Interpreting the figure: Bands from the second OUTCAR are displayed in red, while bands from the first OUTCAR are displayed in black. These depictions could be changed by specifying different colors with the `-c` flag, for example, `-c blue magenta`. Pay attention to the reference: if nothing is specified within the `-r` flag, the Fermi level from both calculations will be set as the 0 eV reference.

4.6 Colored projected bands

Sometimes, it may be more didactic to illustrate projected band structures with colors gradients instead of marker sizes. To perform that, a script named `colored_bands` has been created. It does not splits `PROCAR` and

`OUTCAR` files; instead, it starts from a `bands_projected` folder and plots the file from the available data. From then on, the user receives a simple plot (with no k-points ticks, for example) with the band structure.

Conditions to plot: Only two materials are considered when plotting color gradients in the band structure. Therefore, the original `PROJECTION` file, before splitting the `PROCAR` via `plot_bands` or `split_procar` must specify two sites for projection. Otherwise, contributions will not be computed adequately.

Plotting colored band structures: Inside the directory where a `bands_projected` folder is found, simply execute the command, replacing `BAND_MAX` by the last band you want to plot:

```
colored_bands.py -b 1 BAND_MAX -s -l
```

This will produce the result shown in Fig. 6.

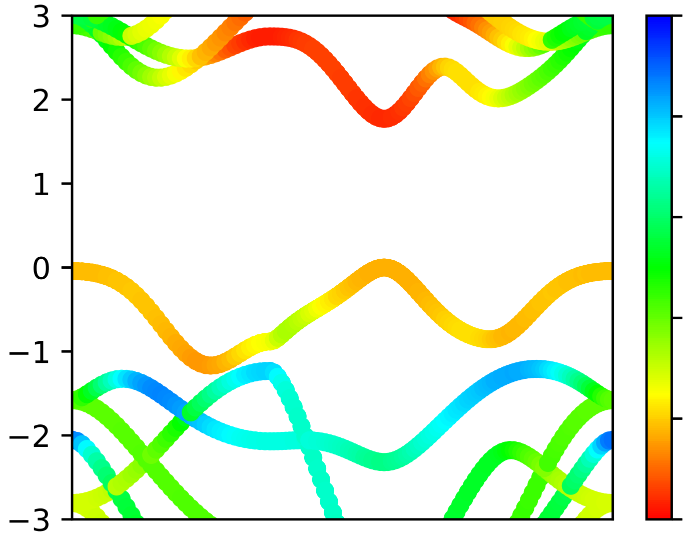

Figure 6: Example of a monolayer WS_2 band structure plotted using `colored_bands.py -b 1 32 -s -l`. Colors close to red (blue) depict greater contributions from W (S) atoms, respectively.

About the figure: The plot does not have any tick markers for the k-points or information in the legend. This is due to the use of the Python `matplotlib` library, which renders results in a different way when compared with XMGrace. A good procedure may be edit the colored bands figure with a software such as Inkscape to conform it to XMGrace axes and legends, obtaining a good figure in return.

4.7 Plotting density of states

Density of states are instantly plotted by using the Vaspirin `dos.py` script. Small and beautiful graphs are created with small and comprehensible commands.

Plotting plain DOS: Inside the directory of interest, with the `DOSCAR` file, one can create density

of states simply by running:
`dos.py`

This loads the default `DOSCAR` file and outputs the data to a `.dat` file, nominally `dos.dat` and the XMGrace batch to `dos.bfile`.

Plotting with XMGrace: To plot the resulting file with XMGrace, execute the following line:
`xmgrace -batch dos.bfile`

An example is shown in Fig. 7.

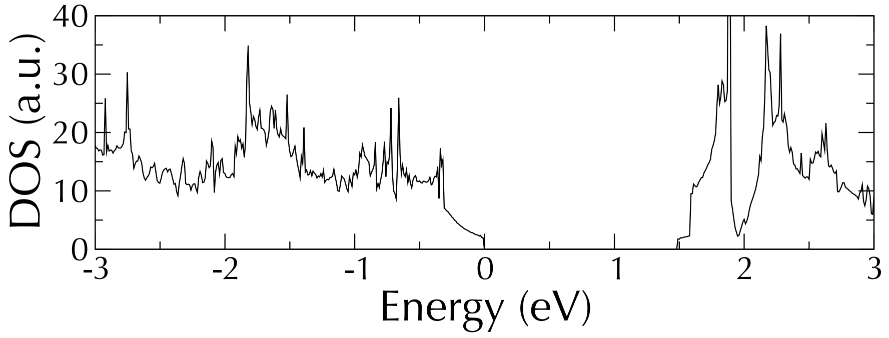

Figure 7: Example of the density of states from a bilayer of hBN/MoSe$_2$ rotated by 19.1° with respect to each other. The DOS was plotted using `dos.py`.

Interpreting the figure: The density of states has the Fermi level as the reference, which can be changed by using the `-r` flag. The DOS may be interpreted with arbitrary units or, according to the VASP manual, in terms of "number of states/unit cell".

Tweaking the plot: To fill below the DOS plot, one may simple use the `-f` flag. To change the DOS axis, one may simply use the `-d` flag. Finally, the energy axis can be altered by means of the `-y`. This way, the `dos` script will create an XMGrace file as shown in Fig. 8.

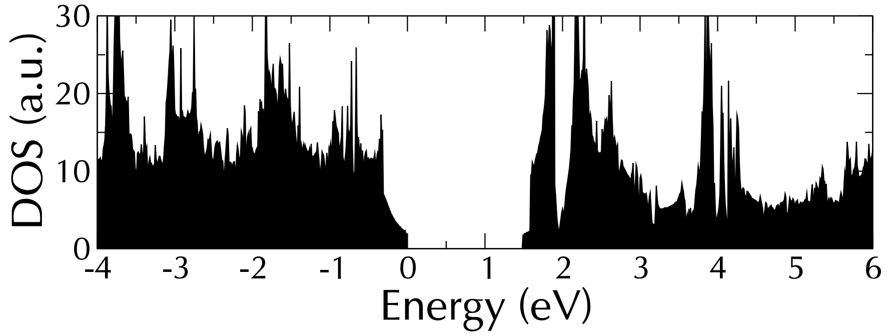

Figure 8: Example of the density of states from a bilayer of hBN/MoSe$_2$ rotated by 19.1° with respect to each other. The DOS was plotted using `dos.py -f -d 30 -y -4 6`.

4.8 Plotting density of states projected onto orbitals

Projected density of states are also plotted by using the Vaspirin `dos.py` script. When plotting onto atomic orbitals, four contributions are set: from s, $p_x + p_y$, p_z and d orbitals.

Plotting projected DOS: Inside the directory of interest, with the `DOSCAR` file, one can create density of states projected onto atomic orbitals simply by running:

```
dos.py -o
```

This loads the default `DOSCAR` file and outputs the data to a `.dat` file, nominally `dosOrbital.dat` and the XMGrace batch to `dosOrbital.bfile`.

Plotting with XMGrace: To plot the resulting file with XMGrace, execute the following line:

```
xmgrace -batch dosOrbital.bfile
```

An example is shown in Fig. 9.

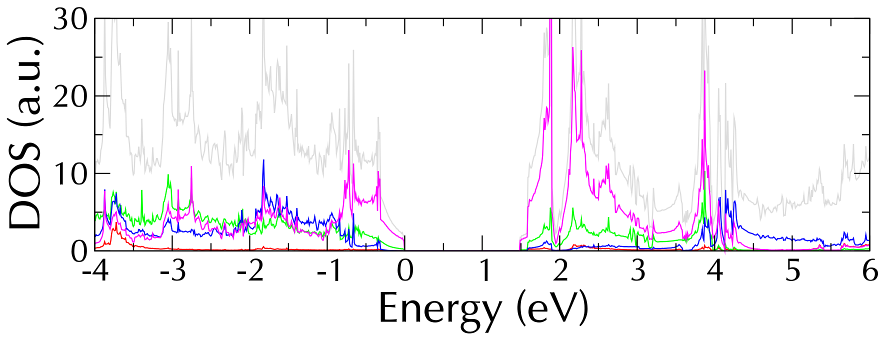

Figure 9: Example of the density of states projected onto atomic orbitals from a bilayer of hBN/MoSe$_2$ rotated by 19.1° with respect to each other. The DOS was plotted using `dos.py -o -d 30 -y -4 6`. Red, green, blue and magenta lines depict DOS projected onto s, $p_x + p_y$, p_z and d orbitals, respectively, while gray lines depict the total DOS.

Interpreting the figure: The density of states projected onto atomic orbitals also has the Fermi level as the reference, which can be changed by using the `-r` flag. The projected DOS has its specific colors, compatible with the creation of band structures projected onto atomic orbitals. The image can always be edited to display only relevant information for the user, specially if one knows how to deal with XMGrace.

4.9 Plotting density of states projected onto sites

Projected density of states onto sites are also plotted by using the Vaspirin `dos.py` script. This requires the creation of a PROJECTION file, specifying ions and projected configurations.

Plotting projected DOS: Inside the directory of interest, with the `DOSCAR` file, a `PROJECTION` file must be created. In this example, the following content is available in the `PROJECTION` file:

```
hBN 1..14 green
MoSe2 15..26 blue
```

which means the first 14 atoms belong to hBN and the other ones to $MoSe_2$. Colors for each material are also specified. From these files, one can create density of states projected onto atomic orbitals simply by running:

```
dos.py -p
```

This loads the default `DOSCAR`, `PROJECTION` file and outputs the data to a `.dat` file, nominally `dosProj.dat` and the XMGrace batch to `dosProjected.bfile`.

Plotting with XMGrace: To plot the resulting file with XMGrace, execute the following line:

```
xmgrace -batch dosProjected.bfile
```

An example is shown in Fig. 10.

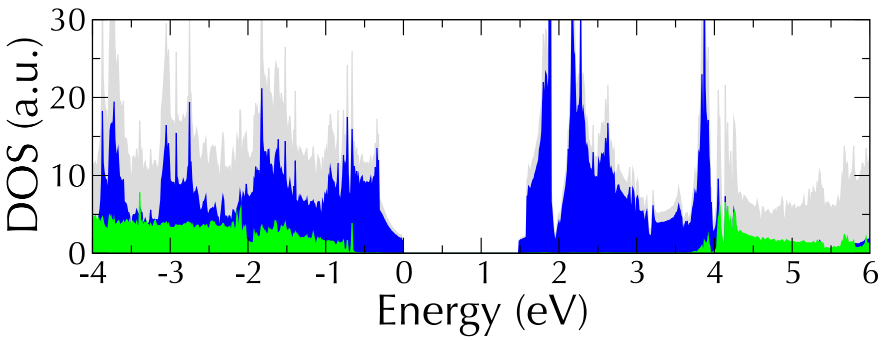

Figure 10: Example of the density of states projected onto atomic orbitals from a bilayer of $hBN/MoSe_2$ rotated by 19.1° with respect to each other. The DOS was plotted using `dos.py -p -f -d 30 -y -4 6`. Green and blue lines represent DOS projected onto hBN and $MoSe_2$, respectively, while gray lines depict the total DOS.

Interpreting the figure: The density of states projected onto atomic sites has the Fermi level as the reference. The projected DOS has its colors specified within the `PROJECTION` file. This input also allows one to personalize which atoms constitute the plot. The image, sometimes, must be edited to display relevant information for the user.

4.10 Drawing band offsets

Band offsets are quite tiresome to draw on scale by hand. The Vaspirin script named `band_offsets` allows one to easily create band diagrams from raw data.

`ALIGNMENTS` file: The input file is referred here with its default name, `ALIGNMENTS`, for the input data to plot band alignments. Its format, in this example, is:

```
hBN -5.65 -0.97 green
:  green
hBN -5.90 -1.22 black
- black
MoSe2 -5.24 -3.75 black
:  blue
MoSe2 -5.24 -3.74 blue
```

To understand how to input data, we shall break the file into its lines:

1. The first line is the default input for a material, with its label (hBN), VBM (-5.65 eV), CBM (-0.97 eV) and its reference color.
2. The second line is a divider. Dividers create the connection between one material and the other, or the interface. Possibilities to choose are:
   - `:` for dotted lines
   - `--` for dashed lines
   - `-.` for dashed-dotted lines
   - `-` for solid vertical lines
   - `None` for empty space

   These dividers also have their colors specified on the second column.
3. The third line is also an input for hBN on the interface.
4. The fourth line represents a divider with solid line. This divider is always drawn on the vertical and automatically creates an interface between two inputs. Its label shall be hBN/$MoSe_2$, since the labels of the adjacent inputs are `hBN` and `MoSe`$_2$.

Creating band offsets: With the `ALIGNMENTS` file done, it is possible to draw the band offsets. To do so, simply run:
`band_offsets.py ALIGNMENTS -s`

This loads the default `ALIGNMENTS` file, displays the figure and then plots it into the `offsets.png` picture. An example is shown in Fig. 11.

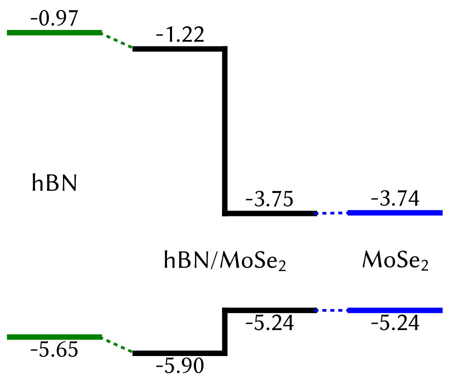

Figure 11: Example of band offsets from a bilayer of hBN/$MoSe_2$ rotated by 19.1° with respect to each other. The diagram was drawn using `band_offsets.py ALIGNMENTS -s`. All energies are in eV.

4.11 Generating k-point paths

If one wants to generate k-point paths directly in the reciprocal lattice coordinates, e.g. for HSE06 calculations, the `gen_kpoints` script shall be of great help. It allows one to choose his/her main Bravais lattice and immediately create a `KPOINTS` file with its symmetry points.

For example, to generate the path Γ-M-K-Γ used above, in band structure calculations, one could simply run:
`gen_kpoints.py G M K G -n 15`

which creates this path with 15 k-points between each symmetry point for the hexagonal lattice (the default lattice). Another option is, starting from an existing `OUTCAR` file, to make a path with uniform sampling, i.e. the number of k-points between symmetry points is proportional to the path length, with at least the specified number of points between adjacent k-points. This allows one to have a better sampling when plotting band structures, since it takes into account the distances in the first Brillouin zone (1BZ). A graphical comparison of the sampling is shown in Fig. 12.

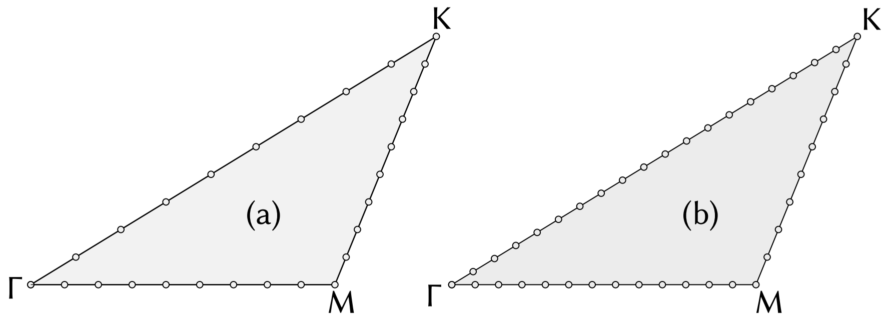

Figure 12: Sampling in the 1BZ for a hexagonal lattice when the path has (a) the same number of points between each symmetry point and (b) a number of points proportional to the distance between the symmetry points. Each circle is a k-point entry in the `KPOINTS` file. The 1BZ path is drawn in terms of reciprocal lattice vectors, which is why its format is not hexagonal.

4.12 Moving atoms in a `POSCAR` file

Instead of manually moving a set of atoms within a `POSCAR` file, one can simply use `move_atoms`. It allows molecules to be translated, 2D solids to be shifted and so on. Let us take, for example, a molecule over a graphene sheet. To move the molecule (last 29 atoms) from a `POSCAR` file, one could simply run:
`move_atoms.py POSCAR -m 201 229 -d 0.1 0.6 -x z -s 0.1`

Understanding the command: This displaces the atoms 201 to 229 from 0.1 to 0.6 Å in steps of 0.1 Å in the z direction and generates a series of corresponding `POSCAR` files. Fig. 13 depicts the operation for the file.

4.13 Straining `POSCAR` files

Straining `POSCAR` files with the Vaspirin script `strain_cell` is pretty straightforward. If one wants, for example, to apply a range of biaxial tensions to the unit cell, it is necessary to run something like:
`strain_cell.py POSCAR -s 0 10 -t 0.5 -x -y`

Understanding the command: The command strains the first and second lattice vectors described in the `POSCAR` cell from 0 to 10% (tensile) in steps of 0.5%. A series of `POSCAR` files will be created to comply with the required strains.

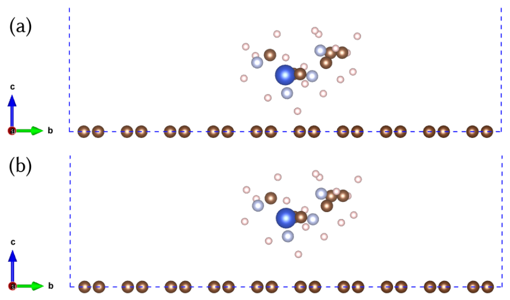

Figure 13: Comparison between (a) the original `POSCAR` file and (b) the `POSCAR` file with the molecule translated by 0.6 Å in the z direction.

A hydrostatic pressure could be done by using the flags `-x -y -z` simultaneously, which is equivalent to using them all together (`-xyz`).

4.14 Rotating atoms in a `POSCAR` file

Instead of manually rotating a set of atoms within a `POSCAR` file, one can simply use `rotate_molecule`. It allows molecules, for example, to be rotated inside a supercell. Let us take, for example, a molecule over a graphene sheet. To move the molecule (last 25 atoms) from a `POSCAR` file, one could simply run:

```
rotate_molecule.py POSCAR -m 201 225 -a 0 90 -x x -s 30 -r 207
```

Understanding the command: This rotates the set of atoms 201 to 225 from 0 to 90 degrees in steps of 30 degrees around the cartesian x axis, taking atom 207 as reference, and generates a series of corresponding `POSCAR` files. Fig. 14 depicts the operation for the file.

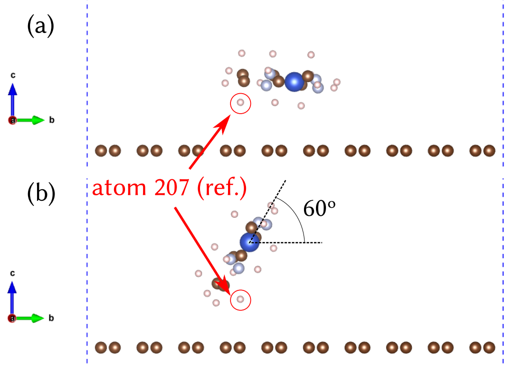

Figure 14: Comparison between (a) the original `POSCAR` file and (b) the `POSCAR` file with the molecule rotated by 60 degrees around the x axis and with respect to the atom 207.

Final words

Vaspirin modules and scripts are in constant improvement. If you feel like you can contribute, please do so! It is important to create a repository of scripts and interfaces and to share them in a most convenient way. This guarantees the continuity of each one's works and helps the students beginning the journey to grow faster and better. Science improves in small steps, after all. If we cannot see further, let us at least lend our shoulders to those who will!

Acknowledgements

The author thanks the help of I. Guilhon and B. Lucatto, for suggestions on writing and debugging of this software.