The Journal of Open Source Software

GIMS: Graphical Interface for Materials Simulations

Sebastian Kokott[1], Iker Hurtado[1], Christian Vorwerk[2], Claudia Draxl[2], Volker Blum[3], and Matthias Scheffler[1]

1 The NOMAD Laboratory at the Fritz Haber Institute of the Max Planck Society, Berlin, Germany **2** Institut für Physik and IRIS Adlershof, Humboldt-Universität zu Berlin, Berlin, Germany **3** Department of Mechanical Engineering and Materials Science, Duke University, Durham, NC, United States of America

DOI: 10.21105/joss.02767

Software

- Review
- Repository
- Archive

Editor: Jeff Gostick

Reviewers:

- @marshallmcdonnell
- @jgostick

Submitted: 06 October 2020
Published: 07 January 2021

Abstract

GIMS (Graphical Interface for Materials Simulations) is an open-source browser-based toolbox for electronic-structure codes. It supports the generation of input files for first-principles electronic-structure calculations and workflows, as well as the automated analysis and visualization of the results. GIMS is deliberately extendable to enable support for any electronic-structure code. Presently, it supports two different software packages: the numerical atom centered orbital package `FHI-aims` and the LAPW code `exciting`.

Statement of Need

Common workflows for electronic-structure calculations require at least the following steps *1. Generating input files:* This step includes the definition of structural data (e.g. position of atoms) and numerical settings (e.g. basis-set quality, runtime choices, and numerical convergence criteria). *2. Running the calculation(s):* Based on the input files, the electronic-structure code performs the requested calculation. Usually, each calculation produces several output files. *3. Post-Processing:* The output files are parsed, analyzed, and results are finally visualized. Step 1 to 3 can be repeated and connected to workflows.

While step 2 is usually run by an *ab initio* engine on a remote HPC cluster, steps 1 and 3 can be executed on local machines. The electronic-structure community primarily uses the *command line interface* and *text editors* as natural working environments, where workflows are automated using scripts. Each of the above-mentioned steps adds its own technical complexity and, thus, potential barriers for the user.

The objective of *GIMS* is to lower the entry barrier and to provide an easy-to-use, platform-independent, zero-setup toolbox for standard tasks within the framework of first-principles electronic-structure calculations. A running GIMS application can be found here: https://gims.ms1p.org. GIMS is intentionally written and designed to be easily extendable to any electronic-structure code. At present, it supports the `FHI-aims` (Blum et al., 2009) and `exciting` (Gulans et al., 2014) codes.

Software Architecture

The application is designed as web client-server system, but can be run entirely on a local machine. The client side is responsible for the user interaction, file parsing, and data visualization. The primary programming language is JavaScript. Conceptually, the web client is

designed as a single page application: the client application is loaded at the outset and some of the data it displays is dynamically updated at runtime by the server. The server has no User Interaction (UI) logic nor does it maintain an UI state.

The server part is written in python. The communication between web application and web server is realized by using the web server gateway interface (WSGI) framework Flask (Ronacher, 2010). Client requests are interfaced with the ASE package (Larsen et al., 2017) on the server side. The ASE package provides python objects for the code-independent handling of atomic structures (`Atoms object`) and numerical settings (`Calculator`). Thus, the use of ASE will enable to extend GIMS' functionalities to all codes (that is, by the time of writing more than 40 different codes) supported by ASE in a straightforward way. Moreover, we integrate `spglib` (Togo & Tanaka, 2018) to obtain additional symmetry properties for periodic structures.

Overview of Features

GIMS is structured in terms of three separate *elemental* apps that address step 1 (input generation) and step 3 (post-processing) described above. These apps also serve as building blocks for workflows (see below). The current three elemental apps are:

1. **Structure Builder.** This app allows to import, view, manipulate, taking snapshots of, and export structure files for various file formats. The 3D structure viewer is based on the threejs library (Cabello, 2010). The builder enables a user to add, delete, and change properties of atoms, as well as to analyze molecular and periodic structures (e.g. measuring distances between two atoms, angles between three atoms, getting symmetry information for periodic structures). This is a subset of capabilities as found, e.g., in existing visualization and building tools such as Jmol, Avogadro (Hanwell et al., 2012), and PyMol, but intrinsically designed as part of a broader client-server framework in the case of GIMS.
2. **Control Generator.** Another step needed to set up a calculation is the selection of numerical parameters. This process is highly specific to each electronic-structure code. The control generator allows the user to provide the basic parameters for the selected electronic-structure code. The user can select items from a form, where tool-tip help provides code-specific information about the listed keywords. As a final product of this step, the input file for the selected code is created and available for download.
3. **Output Analyzer.** After running the calculation, analysis and post-processing of the output files are needed. The output analyzer facilitates some basic tasks, such as output file identification (that is, automatically identifying the code the output files came from and what kind of output files were provided), file parsing, visualization of the results and numerical convergence of the calculations. Graphs can be interactively modified and downloaded as png picture that can be directly used for a presentation or publication.

Workflow apps in GIMS combine different *elemental* apps. For instance, the *band-structure* workflow proceeds as follows: First, the user selects the electronic-structure code with which they want to carry out the corresponding band-structure calculation. Second, based on the provided periodic structure and underlying Bravais lattice defined in the *structure builder*, the correct band path is automatically determined (according to the Setyawan-Curtarolo convention (Setyawan & Curtarolo, 2010) as implemented in the ASE package). Third, mandatory keywords to run a band structure calculation are pre-selected in the *control generator*. Fourth, the band path is incorporated into the corresponding input file. Finally, all resulting output files are processed and visualized by the *output analyzer* in the last step of the workflow.

Parsing and visualizing in- and output files in a browser based framework is also an integral part of other projects that make use of electronic structure data, such as the materials databases

NOMAD, AFLOW, and Materials Project. However, the GIMS workflow apps focuses also on the *generation* of input files. Workflow apps help to make the user aware of potential pitfalls, e.g., background sanity checks of the input files and cross checks among all input files (a simple example would be to make it mandatory to define a k-grid when a periodic structure is used).

The manual and a detailed description of all features are available at: https://gims-developers.gitlab.io/gims. Both the client as well as server part are integration tested using jest and pytest, respectively.

Acknowledgements

This work received funding from the European Union's Horizon 2020 Research and Innovation Programme (grant agreement No. 951786), the NOMAD CoE, MS1P e.V., and ERC:TEC1P (No. 740233).

References

Blum, V., Gehrke, R., Hanke, F., Havu, P., Havu, V., Ren, X., Reuter, K., & Scheffler, M. (2009). Ab initio molecular simulations with numeric atom-centered orbitals. *Computer Physics Communications*, *180*(11), 2175–2196. https://doi.org/10.1016/j.cpc.2009.06.022

Cabello, R. (2010). Three.js. In *GitHub repository*. GitHub. https://github.com/mrdoob/three.js

Gulans, A., Kontur, S., Meisenbichler, C., Nabok, D., Pavone, P., Rigamonti, S., Sagmeister, S., Werner, U., & Draxl, C. (2014). Exciting: A full-potential all-electron package implementing density-functional theory and many-body perturbation theory. *Journal of Physics: Condensed Matter*, *26*(36), 363202. https://doi.org/10.1088/0953-8984/26/36/363202

Hanwell, M. D., Curtis, D. E., Lonie, D. C., Vandermeersch, T., Zurek, E., & Hutchison, G. R. (2012). Avogadro: An advanced semantic chemical editor, visualization, and analysis platform. *Journal of Cheminformatics*, *4*(1), 17. https://doi.org/10.1186/1758-2946-4-17

Larsen, A. H., Mortensen, J. J., Blomqvist, J., Castelli, I. E., Christensen, R., Dułak, M., Friis, J., Groves, M. N., Hammer, B., Hargus, C., Hermes, E. D., Jennings, P. C., Jensen, P. B., Kermode, J., Kitchin, J. R., Kolsbjerg, E. L., Kubal, J., Kaasbjerg, K., Lysgaard, S., … Jacobsen, K. W. (2017). The atomic simulation environment — a python library for working with atoms. *Journal of Physics: Condensed Matter*, *29*(27), 273002. https://doi.org/10.1088/1361-648X/aa680e

Ronacher, A. (2010). Flask. In *GitHub repository*. GitHub. https://github.com/pallets/flask

Setyawan, W., & Curtarolo, S. (2010). High-throughput electronic band structure calculations: Challenges and tools. *Computational Materials Science*, *49*(2), 299–312. https://doi.org/10.1016/j.commatsci.2010.05.010

Togo, A., & Tanaka, I. (2018). *Preprint arXiv:1808.01590*.