express: extensible, high-level workflows for swifter ab initio materials modeling

Qi Zhang\textsuperscript{a}, Chaoxuan Gu\textsuperscript{a}, Jingyi Zhuang\textsuperscript{b,c}, Renata M. Wentzcovitch\textsuperscript{a,b,c,*}

\textsuperscript{a}Department of Applied Physics and Applied Mathematics, Columbia University, New York, NY 10027, USA
\textsuperscript{b}Lamont-Doherty Earth Observatory, Columbia University, Palisades, NY 10964, USA
\textsuperscript{c}Department of Earth and Environmental Sciences, Columbia University, New York, NY 10027, USA

Abstract
In this work, we introduce an open-source Julia project, express, an extensible, high-throughput, high-level workflow framework that aims to automate ab initio calculations for the materials science community. Express is shipped with well-tested workflow templates, including structure optimization, equation of state (EOS) fitting, phonon spectrum (lattice dynamics) calculation, and thermodynamic property calculation in the framework of the quasi-harmonic approximation (QHA). It is designed to be highly modularized so that its components can be reused across various occasions, and customized workflows can be built on top of that. Users can also track the status of workflows in real-time, and rerun failed jobs thanks to the data lineage feature express provides. Two working examples, i.e., all workflows applied to lime and akimotoite, are also presented in the code and this paper.

Keywords: automation, workflow, high-level, high-throughput, data lineage

PROGRAM SUMMARY
Program Title: express
CPC Library link to program files: (to be added by Technical Editor)
Developer’s repository link: https://github.com/MineralsCloud/Express.jl
Code Ocean capsule: (to be added by Technical Editor)
Licensing provisions: The GNU General Public License v3.0 (GPLv3)
Programming language: Julia
Nature of problem: High-performance ab initio calculation is gaining more and more popularity when investigating the physical and chemical properties of materials in the scientific community. There is a lot of ab initio software in the market, but they are, more often than not, not user-friendly to new users because of their intrinsic complexity. Even for familiar users, dealing with daily preparation and post-analysis could be trivial and fallible. There are many workflow software trying to solve the problem. However, most of them cannot meet our expectations due to manifold reasons.
Solution method: We developed a workflow framework that can simplify this process, i.e., most of the mundane work can be replaced by writing a few lines of configurations. We also automated the three most-used work procedures into configurable workflows: equations of state fitting (structural optimization), phonon spectrum calculation, and thermodynamics calculation.

1. Introduction
Materials computations, especially of the ab initio kind, are intrinsically complex. For example, simulations feature many functionalities, eventually imposing a steep learning curve on new users. Much mundane work is performed manually, including preparing required data, writing and checking input, submitting and managing calculation jobs, and analyzing outputs. Such tasks can be time-consuming and prone to human error. Also, if a project is temporarily paused or handed to a new researcher, the lack of complete tracking of previous steps can be another painful experience for some researchers. Luckily, there are multiple ways of mitigating challenges related to these issues. For

\textsuperscript{*}Corresponding author.
E-mail address: rmw2150@columbia.edu

Preprint submitted to Computer Physics Communications September 27, 2021
example, some simulation parameters do not change during the execution cycle, such as cell parameters, atomic positions, and q-points. Some other parameters have a one-to-one correspondence to each input, e.g., pressures of interests in an equation of state calculation and fitting procedure. Such decoupling opens opportunities for concurrent jobs in a distributed system. These difficulties and solutions have inspired us to develop a workflow framework to automate long and extensive ab initio calculations. We name this project express, dedicated to making it direct, stable, and high-speed.

Express relies on previous experiences acquired in similar previous and current projects. One of them is the Virtual Laboratory for the Earth and Planetary Materials (VLab) project[1, 2], funded by the National Science Foundation (NSF) in 2004 at the Minnesota Supercomputing Institute. VLab provided cyberinfrastructure for distributed ab initio computations through a web portal. Users could run predefined, distributed, and interactive workflows on a remote server via web services. Today, it still hosts plenty of online interactive workflows [3], a limited crystal structure input database [4], and a pseudopotentials/PAW datasets library[5] used in previous publications. Unfortunately, as time went by, some of the VLab’s dependencies stopped being maintained[6], making it incompatible with the latest technologies. In the meantime, we developed new and more robust versions of some VLab Java workflows. We chose Julia as our programming language, whose object-orientation, high performance, dynamic nature, and flourishing ecosystem create a pleasant development experience and a satisfying final product.

In this first release of express, we include three workflows: equations of state calculation, phonon spectrum calculation, and property calculations in the framework of the quasi-harmonic approximation (QHA)[7]. Such calculations are performed very frequently, and the methods used are well known and tested. Therefore, these workflows can significantly reduce the need for time-consuming human tasks, e.g., input preparation, job submission, and output analysis.

It is based on the following considerations. The first step of most ab initio calculations is to obtain well converged electronic structure results and static energy-volume curve $E(V)$. The resulting equations of state with different fitting functions are also frequently compared to experimental results to assess the accuracy of the calculation. Therefore, offering an EOS workflow seems a necessary first step. Next, lattice dynamics plays a crucial role in many materials properties that static models cannot solely explain, e.g., thermal expansion and thermal conductivity, especially for minerals at high temperatures and pressures. Phonon modes are also helpful in investigating the phase stability of compounds by inspecting whether imaginary frequencies exist. Last, when predicting thermodynamic properties of materials, the QHA has proved computationally efficient and accurate compared to other statistical mechanics approaches as long as the temperatures of interest are not too high, i.e., close to the melting temperature. QHA results at 300 K can also help us assess the performance of various exchange-correlation functionals, as explained in section 4.4. We drew inspiration from our previously published code, including qha[8] and geothermpy[9], when developing the current workflows. The current code is well-tested, both by actual use and continuous integration systems.

Extensive efforts have already been devoted to implementing ab initio workflows and produced several successful libraries, including but not limited to, Aiiida[10], ASE[11], AFLFlow[12], AbiPy[13], and atomate[14]. However, there is still room for innovation. Some of this software are compatible with others or offer several features, resulting in abstract code. On the other hand, some packages only implement workflows for specific software. By far, QUANTUM ESPRESSO is only supported by a few packages. Even the largest one, Aiiida, covers limited use cases of QUANTUM ESPRESSO. Considering the size of the QUANTUM ESPRESSO community, there is a great need for an advanced and eclectic workflow ecosystem. Some packages mentioned above are implemented in Python, a convenient language when building a prototype project but not the most convenient one when developing a large project due to performance issues. We hope express will be a valuable contribution to the scientific computing community, especially those routinely performing ab initio calculations of thermodynamic properties.

This paper is organized as follows. The next section elaborates on the design philosophy, code composition, and basic functionalities of express. Section 3 focuses on procedures used by the three major workflows available here and how they facilitate typical routine calculations. Section 4 contains the minimal technical documentation introducing the deployment and installation of express, input preparation, command usage, and two working examples, lime and akimotoite. Section 5 offers a closing summary.
2. Code design overview

Running calculations with external software involves many intricacies and requires several considerations. Express aims to provide a high-level interface requiring the slightest effort to learn and user-friendly operations only. Here we briefly discuss some design decisions to accomplish these goals. Some design decisions may also be helpful and easily implemented in other software, e.g., the provenance of workflows and pseudopotentials database in Aiida and atomate. However, the programming interfaces, the modularity of the code, and some convenient functionalities may be unique to this software.

2.1. Three aspects of modularity to enable highly reusable workflows

Though it is conceivable that using express as a monolithic software is preferable by most users, we understand the need for selective installation of its components or tweaking and adopting its functionalities into other code. Even with that flexibility and extensibility disregarded, writing big chunks of code is not advisable. It may result in low readability and unexpected behaviors. Therefore, we pay great attention to the modularity of this code during its development. The current stage of express has three almost orthogonal aspects of modularity, as described below.

Figure 1: Main components of the express project in terms of Julia packages. The switch in this figure means that QuantumESPRESSOExpress.jl is a plugin loosely coupled to Express.jl and can be enabled or disabled flexibly.

Under the project name express is a collection of Julia packages, whose core is Express.jl, managing and dispatching the rest. Figurative relations of some main components of express are shown in Figure 1, where rectangles of the same color are at the same level, and lighter-colored rectangles are at lower levels. High-level packages
QuantumESPRESSO.jl could depend on low-level packages (e.g., QuantumESPRESSOBase.jl), i.e., the high-level ones utilize the functionalities exported by the low-level ones to maximize code reuse. Each package is released individually and has its version number to avoid updating the whole codebase whenever there is a bug fix or a feature enhancement. They also have separated pull request pages for developers and skilled users to discuss and collaborate. Julia’s semantic versioning system manages their compatibility, i.e., compatible packages are downloaded automatically, and no human intervention is needed in most cases. The functionalities of each package are described below. There are several direct dependencies of Express.jl, which provide the basic functionalities of express. They can be executed without relying on each other as well.

- Express.jl provides a high-level interface to all the workflows, including file reading and writing, job creation, submission, monitoring, result retrieving, and data analysis. To work with specific software, install the corresponding plugin, e.g., QuantumESPRESSOExpress.jl for Quantum ESPRESSO.
- ExpressCommands.jl is a user-friendly command-line interface of Express.jl for non-developers. It installs an executable ‘xps’ that can execute code from configuration files provided by users. We will review this in section 4.3.
- EquationsOfStateOfSolids.jl fits energy (or pressure) vs. volume results to equations of state, etc. These features are repetitively used in the equation of state workflow (section 3.1).
- Crystallography.jl calculates a crystal’s primitive cell (or supercell) volume from lattice parameters, finds symmetry operations and generates high symmetry points in the Brillouin zone, etc.
- PyQHA.jl is a Julia wrapper of the Python qha package[8], which can calculate several thermodynamic properties of both single- and multi-configuration crystalline materials in the framework of quasi-harmonic approximation (QHA). The qha code is the foundation of the QHA workflow (section 3.3).
- Geotherm.jl is a Julia interpretation of the Fortran code we used in reference [9], which calculates the isentropic temperature/pressure gradient (geotherm) using thermodynamic properties obtained with the QHA workflow.
- Pseudopotentials.jl presents a database for storing and querying pseudopotentials used in ab initio calculations. We will elaborate on this topic in section 2.4.
- SimpleWorkflows.jl is the skeleton of the workflow system, which defines building blocks, composition rules, and operation order of workflows. See section 2.2.

The second modularity lies in that every workflow is software-neutral. As an example, the QuantumESPRESSOExpress.jl in Figure 1 is a special type of package called “plugin of express” for handling ab initio software such as Quantum ESPRESSO. Other plugins for other software are possible. To do this, Express.jl defines a public protocol that is open to extension, and the package representing the software implements this protocol. These plugins offer a unified interface to users dealing with multiple distinct software without modifying their code. Though currently, only Quantum ESPRESSO is supported, it is straightforward to add new plugins in the future.

The dependencies of QuantumESPRESSOExpress.jl are listed below.

- AbInitioSoftwareBase.jl provides a standard API for some popular ab initio software such as Quantum ESPRESSO.
- QuantumESPRESSOBase.jl declares basic data types and methods for manipulating crystal structures, generating input files for Quantum ESPRESSO, error checking before running, etc.
- QuantumESPRESSOParser.jl parses the input or output files of Quantum ESPRESSO to extract and analyze data.
- QuantumESPRESSOFormatter.jl formats the input files of Quantum ESPRESSO.
• QuantumESPRESSOCommands.jl is a command-line interface that exports the commands QUANTUM ESPRESSO uses in a configurable way.

• QuantumESPRESSO.jl is simply a wrapper of the types, methods, and commands defined in QuantumESPRESSOBase.jl, QuantumESPRESSOParser.jl, QuantumESPRESSOFormatter.jl, and QuantumESPRESSOCommands.jl under a common namespace.

Third, workflows should consist of atomic building blocks that could be added, removed, or reused. It is an intuitive way that reduces the repetition of code and human intervention. We will elaborate on this in section 2.3.

2.2. A user-friendly interface that simplifies the installation of express and interaction with ab initio software

Due to the diversity of simulation software, programming languages, and computing platforms, complete compatibility between them is difficult or virtually impossible to achieve. If one language or software manages to do it, it will gain the most popularity. Python is one of the candidates, but it has shown its limitations over the years, e.g., limited performance and speed. However, an emerging language, Julia, was designed as an option. It has improved interoperability with other languages, and such versatility is continuously improving. It can now interact directly with programming languages, e.g., C/C++, Java, R, MATLAB, Mathematica, Fortran, etc., covering plenty of practical needs. Users would spare little effort to integrate express into their code if necessary.

Apart from interoperability, the accessibility of express is essential as we understand a steep learning curve usually pushes users away. Generally, users are held back at the moment of installing the software because of compatibility issues. Installing express should be easy for most users, and we will discuss installation details in section 4.1. Another way of flattening the learning curve is to provide a user-friendly yet reusable interface. There are four standard user interfaces: (a) directly running the Julia code in read-eval-print loops (REPLs) or (b) running a curated list of commands covering most use cases with the help of a configuration file; (c) scripts written in one or more programming languages; (d) the graphical user interface (GUI). The first three interfaces are available in express. The first two methods require running from a local or remote terminal session. It is possible to build a GUI, e.g., a website or an application for it in the future. But since the majority of the code is run on remote clusters most of the time, our current focus is on extending functionalities and improving the usability of the other interfaces. They offer almost complete freedom when interacting with express.

While users can benefit from exclusive Julia features and libraries not bound by a workflow-specific API, using them requires some programming skills and familiarities with Julia, which not everyone is expected to have. On the other hand, the configuration file interface seems to be leveraging both customizability and user-friendliness. Thus, it is our recommended way of using express. Therefore, as mentioned in section 2.1, we ship express with an executable ‘xps’ that can handle most operations. Computational settings are written into a human-readable file and fed to xps when running. We will discuss the details of this interface in section 4.2. The advantages and disadvantages of each interface are listed in Table 1 for clarification.

2.3. Graph model for workflow representation

In computer science, a workflow is often abstracted into a directed acyclic graph (DAG), where the vertices symbolize actions and the edges specify orders of these actions. Each vertex, doing only one thing at a time, is called an AtomicJob in express. It could be generating an input, running an executable, reading a file, etc. AtomicJobs can be combined sequentially or in parallel, forming more complex operations. Therefore, we can make any task out of smaller existing ones. Furthermore, any part of the workflow can be cut, branched, or looped according to need.

Apart from the flexibility, splitting a workflow into small pieces instead of having a monolithic chunk of code has other benefits. First, it is compliant with human perception of how jobs are organized, making understanding, checking, and monitoring workflows more intuitive. It also avoids writing repetitive code, a rule of thumb in software engineering. For example, it is often the case that we want to apply the same operation on different input parameters, or the same operation is reused multiple times in a workflow or in different workflows. More importantly, it reduces the time and effort to rerun jobs because of their loose coupling. This property is desirable since running external software, e.g., QUANTUM ESPRESSO is usually the most computationally expensive part.

Tracking job status, input parameters, and output results manually are time-consuming and error-prone. In express, each job’ status (whether it succeeded, failed, is running or pending), input, and output in the workflow
are tracked by an isomorphic DAG. The DAG is saved to a file while running, making it possible to rerun the interrupted or failed steps after restarting or fixing the reported errors. The file can also be stored in a database for future reference or sharing information with colleagues. Figure 2 shows the representation of a simplified running workflow where each ellipse denotes a job and each color labels a status. In this chart, jobs are divided into three concurrent groups, where the third one throws an error in the first step. Therefore, its direct descendants will also fail. Whether we will still get the final result depends on whether the third group is decisively influential. The next step is either to fix the error and rerun the workflow or disregard that group. Real-world cases are usually more complex but follow the same logic.

![Figure 2: Abstract representation of a workflow state, plotted by Express. Two sequential snapshots in time, a and b, are shown. Figure a shows stage one, where some jobs are either finished, running, or pending. In Figure b (stage two), all jobs are finished. The final result is obtained even though one group is failed. The execution order is from top to bottom in both figures, and each color denotes a different status. Light green, pink, yellow, and light purple mean succeeded, failed, running, and pending, respectively.](image)

2.4. Pseudopotential database

As mentioned in section 1, VLab has a limited pseudopotential library. To follow this tradition, we integrate an optionally installable pseudopotential database into package Pseudopotentials.jl. The database lists all available pseudopotentials from PSlibrary[15] with some of their properties. It is a local binary file in JLD2 format (an HDF5-compatible file format) storing a tabular data structure DataFrame (implemented in DataFrames.jl[16]). This combination guarantees it is easy to filter, aggregate, sort, insert, delete, and perform other database operations while being very efficient. No prior knowledge of databases is required to operate. Some Python users may find it has a similar syntax to pandas. Users may also share it with colleagues. Before running calculations, the pseudopotentials required in inputs will be automatically downloaded from the database if the corresponding items exist. If the files are not found either in the database or locally, the workflow being executed will throw an exception immediately. This mechanism highlights the error and avoids wasting computing resources since potential bugs are found before running on ab initio software. We also provide a parser that could parse files of unified pseudopotential format (UPF), which Quantum ESPRESSO, ABINIT, gpaw, etc., adopt. Please see its official documentation[17] for more details.
3. Available workflows

Figure 3 presents a high-level overview of the workflows we include in express. Each light-gray block denotes a workflow we have and will be documented below. The static elasticity block is not currently contained in express but will be released in the near future. A white block means the results from a previous block. Starting from the equation of state (EOS) workflow, we get a series of equilibrium structures. Then we could either run a phonon workflow followed by a QHA workflow to get thermodynamic properties of a material, or we could obtain thermoelasticity from the static elasticity workflow (whose algorithm is proposed in reference [18]) combined with the QHA workflow.

As mentioned, the workflows are highly modularized, which are ready to be separated, chained, and customized according to practical needs. For example, we could join the EOS, phonon, and QHA workflows into one. Some workflows can have more than one implementation, as we will explain in the following subsections. QUANTUM ESPRESSO will be treated as the default target software unless otherwise stated.

![Figure 3: High-level schematic representation of the workflows.](image)

3.1. Equation of state (EOS) workflow

Figure 4 shows the EOS workflow, which is a zoomed-in view of the corresponding block in Figure 3, denoted by “equation of state (EOS) block”. Here each gray block (action block) stands for a computational task, and each white block (data block) corresponds to an input or output. Following this notation, a gray stack means a group of concurrent tasks, while a white stack means a list of inputs or outputs distributed to each task. Some blocks are still simplified for legibility since they may contain some trivial steps, especially when interacting with external software like QUANTUM ESPRESSO. For example, the stack “do structure optimizations” actually consists of three substates: input generation, running ab initio simulation, and output analysis (including reading optimized structures, fitting EOS, etc.). The legend shares the same meaning in subsequent workflow graphs. The complete procedure of this workflow is:

![Figure 4: Visual depiction of the “parallel” implementation of the equation of state (EOS) workflow.](image)
1. Given the roughly approximate equation of state parameters and user-desired pressures, estimate the volumes corresponding to these pressures. This is done by numerically finding the solution $V$ of a given EOS at a certain pressure: $P(V) - P_{\text{desired}} = 0$. In this process, isotropic volume expansion or compression is assumed.

2. Generate a series of inputs of self-consistent field (SCF) calculations at these volumes based on a template input file, and send them to the ab initio software. As mentioned in section 2.4, the required pseudopotentials will be automatically downloaded if possible during this process. We will not emphasize this again in the following text.

3. Gather energies and cell volumes from the outputs (using the parser in QuantumESPRESSOParser.jl) and fit them into a refined EOS. Multiple types of EOS can be chosen, as explained in Table 2. Unit conversion is often prone to human errors. Thus, we allow users to specify the units of the initial EOS parameters, and the fitting algorithm will return values with the same units.

4. Save the intermediate results. Then repeat step 1, find the volumes corresponding to user-desired pressures using the new EOS.

5. As in step 2, generate a series of inputs for structure optimizations at these volumes and run ab initio calculations.

6. As in step 3, fit the final EOS and get the equilibrium structures. Save the results to a file for future reference.

There is also a more coupled, sequential way of sampling data points, as shown in Figure 5, i.e., optimizing structures in increasing order of pressures where the output of the $n$th step is used as the input of the $(n+1)$th step. This method may take a longer time than the first one, given its sequential computation nature. Its complete procedure is:

1. Read the template file providing the initial structure and other parameters.
2. Set desired volumes as optimization goals and generate a series of inputs with information from that template.
3. Optimize the structure of the first input, use the output structure as the initial structure for the next input, repeat this step for the remaining volumes until all structures are optimized.
4. Fit an equation of state to energy vs. volume data.

This method is supplementary to the first one. It is often used when a crude EOS is unknown, especially when working on a new crystal structure where no calculations nor experiments have been performed.
3.2. Phonon workflow

As shown in Figure 3, the phonon block is indispensable whether the goal is to obtain thermodynamic or thermoelastic properties. Also, the effects of electronic thermal excitations on phonon frequencies and their implications for the thermodynamic properties are addressed in our recent publication [19]. The code (pgm) will be published and integrated into a future version of express. The procedure of the phonon workflow is:

1. Start from one or more equilibrium crystal structures, which are not necessarily but likely to be optimized by an EOS workflow; define q-points automatically. Generate input files for the next step.

2. Perform multiple distributed SCF and density functional perturbation theory (DFPT) [20, 21] calculations to obtain dynamical matrices at those user-defined q-points. This step often requires distributing jobs over CPU cores since this is usually the most time-consuming step in the phonon workflow. Express distributes jobs by itself. However, it also allows manual configuration due to the complexity of distributed computing.

3. (Call the ab initio software to) perform a Fourier transform to derive force constant matrices from those dynamical matrices.

4. Compute phonon frequencies in the entire Brillouin zone using these force constant matrices. Two types of calculations can be selected: path-mode, where the phonon dispersion is calculated along a high-symmetry q-path; and uniform-mode, where the vibrational density of states (VDOS) is sampled from results on a uniform q-point mesh. Switching between the two modes requires only a keyword in the configuration file.

While the current workflow computes phonon frequencies using DFPT, implementing the small displacement method to compute force constant matrices will be added in the near future.

3.3. QHA workflow

As shown in Figure 7, the QHA block is a key component for calculating thermodynamic properties. The QHA model is computed by considering the free energy at different volumes.

Figure 6: Graphical representation of the phonon workflow.

Figure 7: Schematic representation of the QHA workflow.
In the qha package, we allow calculating the thermodynamic properties of materials with either single or multiple configurations within a user-specified pressure and temperature range. The contribution of multiple configurations is not ignorable when investigating fully or partially disordered phases or order-disorder phase boundaries, e.g., between ice-VIII and ice-VII, high-density phases of ice[22]. The rough steps of a QHA workflow are

1. Determine whether it is a single- or multi-configuration calculation.

2. Obtain VDOS at each statically constrained volume (of each configuration) using the phonon workflow. If there are imaginary frequencies, the workflow will warn the users that QHA might not be applicable in this case.

3. Read input data and compute the free energy, \( F(T, V) \), of each desired temperature \( T \) and volume \( V \) using the qha code. Interpolate \( F(T, V) \) on a finer volume grid for better accuracy during this process. See equations (2) – (8) in reference [8] for a more detailed description.

4. Derive other thermodynamic properties, including entropy, internal energy, enthalpy, Gibbs free energy, thermal expansion coefficient, Grüneisen parameter, bulk modulus, and heat capacity, of the material from \( F(T, V) \). Save all the results to human-readable text files, and plot them in PDF format if desired.

The qha Python code features an entry command ‘qha’, which can run with some input data and a runtime settings file. It has been the inspiration of express’s command-line interface.

4. Documentations on the express project

4.1. Deployment and installation

Usually, the software that performs \textit{ab initio} calculations is not installed on a local computer but on a remote server or virtual machine. Express can be deployed in all these cases. Thus, there are two situations for the deployment locations of express and the \textit{ab initio} software (for example, QUANTUM ESPRESSO): (i) they are installed and run in the same environment, whether it is a local computer, a remote server, or a virtual machine; (ii) they are installed in different environments, e.g., express is local and QUANTUM ESPRESSO is remote. The latter one is a bit tricky as it requires file transferring and calls between different systems. By now, we recommend installing express in the same environment where the \textit{ab initio} software is installed. In general, this procedure will be smooth since express can be installed wherever Julia and Python can be installed, which covers almost all operating systems and CPU architectures. So it is the assumed method in the following text.

First, you should install Julia. We recommend downloading it from its official website[23]. Versions higher than 1.3, especially 1.6, are strongly recommended. Please follow the detailed instructions on its website if you have to build Julia from source[24]. Some computing centers provide preinstalled Julia. Please contact your administrator for more information in that case.

Next, install Express.jl and choose one plugin for the \textit{ab initio} software. As stated, currently, only the QuantumESPRESSOExpress.jl plugin is available. Please open Julia’s interactive session (REPL) and type the following lines of code:

```julia
using Pkg
Pkg.add("Express")
Pkg.add("QuantumESPRESSOExpress")
Pkg.add("ExpressCommands")
```

Then wait until it is done.

We prepared and ran many unit tests to validate the packages that constitute express. We employed several continuous integration (CI) services (GitHub Actions, AppVeyor, Drone CI, Cirrus CI, and GitLab CI) to run these tests on every Git commit and pull request across multiple platforms and Julia versions so that most incompatibilities and other bugs can be identified before the official release of express.

We host more detailed documentation on our website (reference [25] for stable releases and reference [26] for the development version) for troubleshooting. You are also welcome to report bugs to us [27] or start a discussion if anything is unclear [28].
4.2. Input and output files

As suggested in section 2.2, we regard the configuration file interface as our primary way of interacting with \texttt{express}. Therefore, we will mainly describe how to prepare the input files for it here. The usage of other interfaces, including calling public API, is discussed in our official documentation on GitHub. However, the same input files could be reused by other interfaces. There is no cost for switching interfaces.

We need two types of input files for each workflow: a template input for the \textit{ab initio} software and a configuration file. The template input’ syntax may vary since it completely depends on the software, but all workflows share the configuration file’ syntax. However, it does not mean that all configuration files ask for the same items. For each type of workflow, the items change a little. The complete set of allowed items for each kind of workflow is listed in tables 2–4. The code block below shows a typical configuration file for an EOS workflow. To view more real-world examples, please visit our GitHub repository [29]. When running the workflows, \texttt{express} will automatically generate more input files based on the specifications in the template input and the configuration file. This action saves users a lot of time and labor since each input file usually varies slightly from the other. Most of the input parameters are fixed in a series of calculations, e.g., DFT+U calculations with a constant U. The copying, pasting, and editing procedures are mundane and error-prone. For example, only two variables change when setting pressures of interests and corresponding volumes in an EOS workflow. It is very unproductive if one has to edit those input files one by one.

Different workflows would have different outputs. But generally, they are divided into two categories: raw data returned by the \textit{ab initio} software and processed data, figures, or logs that \texttt{express} prints or plots directly. The first type of file is usually the source of the second type. However, the second file type can also be the source of files of the first type, e.g., inputs, if we chain workflows together. For instance, the optimized structures are extracted and treated as input structures in a phonon workflow after an EOS workflow.

![Code block](image)

4.3. Running workflows

This section shows how to run a configuration file and illustrates how much work can be simplified compared to running calculations manually. After installation, the path to \texttt{xps} (\$HOME/.julia/bin/xps by default) is expected to be added to the PATH environment variable. If not, please refer to our documentation.

Users can interact with the configuration file interface through a terminal session. They can also write a script and submit it to the scheduler on a high performance computing platform after trying out the undermentioned commands on a small sample of inputs. When we say “type” or “run” in the following text, we mean running commands through that session. To carry out a computation, first, users need to prepare the input files introduced in section 4.2. Then type

\begin{verbatim}
xps run <path-to-config-file>
\end{verbatim}

where \texttt{<path-to-config-file>} is the location of the configuration file on the file system. In an EOS workflow, the final results include outputs returned by \texttt{QUANTUM ESPRESSO}, a list of raw data (volume-energy pairs), and a fitted EOS. If something goes wrong, the workflow might be terminated. Its state and the error will be saved in a file for
debugging, as mentioned in section 2.3. Once the bug is fixed, run `xps run <path-to-config-file>` again, and `express` will retry the failed jobs. To print either input or output data in a formatted, readable form, run

```
xps print <file-name>
```

where the allowed extensions of `<file-name>` are `.jls`, `.json`, `.yaml` or `.yml`. The last four extensions correspond to three human-readable data-serialization file formats, i.e., JSON, YAML, and TOML, while `.jls` is a binary serialization format only recognizable to Julia. `Express` can also plot some data, such as the fitted EOS applied to a certain range of volumes along with the raw data. The corresponding command is

```
xps plot <file-name>
```

where `<file-name>` refers to the EOS binary file with extension `.jls`. These are the three most important commands of `express`. These catchy commands cover all the functionalities we have promised, including but not limited to unit conversion, pseudopotential downloading, input validation and generation, calculation monitoring, task distribution, gathering and analysis of results, error handling, logging, and visualization. We hope they can facilitate tedious work as much as possible.

4.4. Test cases

These workflows are well tested, and we want to apply them to a wide range of materials. Here we present two examples, lime, and akimotoite, with different space groups, number of atoms per cell, atom types, etc. We perform the EOS, phonon, and QHA workflows successively to obtain the static and thermal properties of the two materials. The entire procedure is wrapped in one job submitted to a compute node, where a distribution of QUANTUM ESPRESSO is installed. The local-density approximation (LDA) and the Perdew–Burke–Ernzerhof generalized gradient approximation (PBE-GGA) are used for the exchange and correlation functionals. We did not adopt the SCAN and the PBEsol functionals in these examples, but the workflows work equally well for them as well as for DFT+U with constant U. Figures 8 and 9 show both static and thermal equations of state (third-order Birch–Murnaghan) compared with other *ab initio* calculations and experiments. The results calculated with `express` are in relatively good agreement with the experimental data at room temperature, but the LDA functional performs slightly better than PBE. The results are within the margin of error expected for these functionals.

![Figure 8: Compression curves for lime (CaO). Dashed-dot-dot lines denote static and thermal equations of state calculated with LDA exchange-correlation functionals in this work, while solid lines are calculated with PBE-GGA exchange-correlation functionals. Dashed lines are from a previous *ab initio* study[32]. Diamonds and stars are results measured in two diamond anvil cell (DAC) experiments[33, 34].](image1)

![Figure 9: Compression curves for corundum-type MgSiO$_3$, akimotoite, obtained using different exchange-correlation functionals (LDA and PBE-GGA, dashed and solid curves) and temperatures (static and 300 K) in this work, compared with experimental data (diamonds and stars) from two multi-anvil experiments[35, 36].](image2)

Lime (CaO) has the rock-salt (NaCl) structure. Our LDA calculations (dashed-dot-dot lines) use Vanderbilt ultrasoft pseudopotentials, while PBE calculations (solid lines) use the projector augmented wave (PAW) potentials. We adopt cutoff energies of 90 Ry for LDA and 120 Ry for PBE, and a $4 \times 4 \times 4$ Monkhorst–Pack k-point mesh for both
cases. Each initial structure is optimized at 8 (−10 GPa to 50 GPa) pressures before performing DFPT calculations on a $4 \times 4 \times 4$ q-point mesh. Phonon frequencies are then interpolated on a $30 \times 30 \times 30$ uniform q-mesh in the Brillouin zone. Karki et al.\cite{32} performed LDA simulations with Troullier–Martins pseudopotentials (dashed lines). They used 90 Ry as cutoff energy, integrated the Brillouin zone over 6 special k-points, performed DFPT calculations on a $4 \times 4 \times 4$ q-point mesh. The diamonds and stars show two equations of state of CaO determined by diamond anvil cell (DAC) experiments\cite{33}.

Akimotoite, or corundum-type ($\text{MgSiO}_3$), has trigonal symmetry ($R\bar{3}$). As in the previous example, we carry out the LDA computations with Vanderbilt ultrasoft pseudopotentials and PBE-GGA with PAW potentials. In this case, the plane-wave cutoff energies are 170 Ry for LDA and 120 Ry for PBE, and the Monkhorst–Pack k-point grids are both $3 \times 3 \times 3$. The q-point grids for DFPT and VDOS calculations are $3 \times 3 \times 3$ and $30 \times 30 \times 30$, respectively. Four equations of state (solid and dashed lines) are also fitted using the aforementioned workflows for both exchange-correlation functionals at the static condition and room temperature. We also compare them with two equations of state from multi-anvil experiments\cite{35, 36}.

5. Conclusions

In this work, we take advantage of the experiences gained from developing VLab and other cyberinfrastructures in the past, implement the workflows we use now, and pave the way for implementing new functionalities and integrating more atomistic simulation software for materials modeling into our workflows in the future. Express aims to make research in computational materials science simpler, faster, and more collaborative. It helps users in the preparation of inputs, execution of simulations, and analysis of data. It tracks the steps users performed and can restart interrupted or failed jobs. We consider the users’ difficulty in learning new software seriously, so we tried to make it user-friendly by providing multiple interfaces, detailed documentation, and examples. Express, an open-source program distributed under the GNU General Public License, is under active development, and more features will be available soon. Please follow our project page\cite{37} for future updates.

Acknowledgments

This work was supported by DOE award DE-SC0019759. It used the Extreme Science and Engineering Discovery Environment (XSEDE\cite{38}), which is supported by the National Science Foundation grant number ACI-1548562 and allocation TG-DMR180081. The authors also thank Jiayang Wang, Michel Marcondes, Hongjin Wang, and Pedro da Silveira for their contributions and suggestions to this code.
Table 1: Comparisons between different possible interfaces of `expr`.

| interface                        | implemented | advantages                                           | disadvantages                                           |
|----------------------------------|-------------|------------------------------------------------------|--------------------------------------------------------|
| REPL                             | true        | Highly customizable (It can run any code.)           | • It needs to be run in interactive mode.              |
|                                  |             |                                                      | • Commands are typed one by one, suitable for debugging.|
|                                  |             |                                                      | • It requires some programming skills.                 |
| scripts                          | true        | The same as the REPL interface, except that it can run in non-interactive mode with multiple commands | It requires some programming skills to use.            |
| command-line (with configuration files) | true  | More user-friendly than the previous two interfaces | Less customizable than the previous two                |
|                                  |             | • Highly reusable                                   |                                                        |
| GUI (web or application)         | false       | The most user-friendly interface                    | Less reusable than the above three                     |
Table 2: Recognizable parameters in the configuration file of an EOS workflow in TOML syntax.

| Keys               | Values (with a default value if applicable)                                                                                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| recipe             | A string that represents the type of the workflow. Allowed value is `eos`. The path to a template input file for a specific software. It should be on the same file system where `express` is deployed.                                                         |
| template           | The trial EOS contains initial values for input files generation and EOS fitting. A string that represents the type of the EOS. Allowed values are murnaghan (Murnaghan), bm2 (Birch–Murnaghan second order), bm3, bm4, vinet (Vinet), pt2 (Poirier–Tarantola second order), pt3, and pt4. |
| trial_eos type     | A vector of strings that specifies each value of the EOS. The default order is $V_0$, $B_0$, $B'_0$, $B''_0$, etc.). Units must be provided.                                                                                                         |
| fixed.pressures    | Specify the pressures or volumes. It can be a vector of numbers, or a string with the syntax "start:step:stop" to form an arithmetic sequence where start, stop, and step are numbers indicating the start, the end, and the common difference of that sequence. |
| fixed.volumes      | The units of pressure or volume. The pressure and volume default units are GPa and angstrom$^3$. Allowed values for volumes are nm$^3$, angstrom$^3$, bohr$^3$, etc. Allowed values for pressures are Pa, GPa, TPa, ..., bar, kbar, ..., atm, and the combinations of eV, Ry, hartree, J, with any unit of volume. |
| unit               | The path of the root directory of output files.                                                                                                                                                                                             |
| files              | It specifies the paths of output directories. A string specifying the naming convention of the output directories. Its default value is `p=`. For example, if fixed.pressures.values is a vector of pressures [10, 20, 30] which represents the relaxations are done from 10 GPa to 30 GPa, then the generated inputs and outputs will be stored in directories p=10, p=20 and p=30. |
| dirs               |                                                                                                                                                                                                                                           |
| root               | The path of the root directory of output files.                                                                                                                                                                                             |
| pattern            | A string specifying the naming convention of the output directories. Its default value is `p=`. For example, if fixed.pressures.values is a vector of pressures [10, 20, 30] which represents the relaxations are done from 10 GPa to 30 GPa, then the generated inputs and outputs will be stored in directories p=10, p=20 and p=30. |
| save status eos    | The path to a binary file that stores the status of the workflow. The path to a binary file that stores the fitted equations of state.                                                                                                           |
| cli                | The command-line tools settings.                                                                                                                                                                                                         |
| mpi                | The configurations of the MPI software.                                                                                                                                                                                                    |
| np                 | An integer indicating the number of processors/cores/CPU's used.                                                                                                                                                                          |
Table 3: Recognizable parameters in the configuration file of a phonon workflow in TOML syntax.

| Keys   | Values (with a default value if applicable) |
|--------|---------------------------------------------|
| recipe | A string that represents the type of the workflow. Allowed values are phonon dispersion (phonon dispersion along a q-path) and vdos (phonon density of states). |
| template |                                        |
| scf | The path to a template input file for an SCF calculation. |
| dfpt | The path to a template input file for a DFPT calculation. |
| q2r | The path to a template input file for a Fourier transform. |
| disp | The path to a template input file for a phonon dispersion/phonon density of states calculation. |
| fixed | The same as that of an EOS workflow. |
| files | The same as that of an EOS workflow. |
| save | The path to a binary file that stores the status of the workflow. |
| cli | The command-line tools settings. |
| mpi | The configurations of the MPI software. |
| np | An integer indicating the number of processors/cores/CPU's used. |

Table 4: Recognizable parameters in the configuration file of a QHA workflow in TOML syntax.

| Keys   | Values (with a default value if applicable) |
|--------|---------------------------------------------|
| recipe | A string that represents the type of the workflow. Allowed values are single qha (single configuration QHA) and multi qha (multiconfiguration QHA). |
| input | A path to the input file of qha. |
| pressures | Pressures on the dense T − P grid. The same as that of an EOS workflow. |
| temperatures | Temperatures on the T − P grid. |
| values | Specify the temperatures. It can be a vector of numbers, or a string with the syntax "start:step:stop" to form an arithmetic sequence. |
| unit | The unit of temperature. Its default value is K. |
| thermo | Specify which thermodynamic properties are supposed to be calculated. Please refer to Table 5 in reference [8]. |
References

[1] M. A. Nacar, M. S. Aktas, M. Pierce, Z. Lu, G. Erlebacher, D. Kigelman, E. F. Bollig, C. R. S. da Silva, B. Sowell, D. A. Yuen, Vlab: collaborative grid services and portals to support computational material science, Concurrency and Computation: Practice and Experience 19 (12) (2007) 1717–1728. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.1199, doi:https://doi.org/10.1002/cpe.1199.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.1199

[2] P. R. da Silva, C. R. da Silva, R. M. Wentzcovitch, Metadata management for distributed first principles calculations in vlab—a collaborative cyberinfrastructure for materials computation, Computer Physics Communications 178 (3) (2008) 186–198. doi:https://doi.org/10.1016/j.cpc.2007.09.001.
URL https://www.sciencedirect.com/science/article/pii/S0010465507003955

[3] J. Z. Qi Zhang. Resources.
URL http://mineralscloud.com/resources/

[4] J. Zhaang. Crystal structure input database.
URL http://mineralscloud.com/resources/mineralsdatabase/

[5] J. Zhaang. Pseudopotentials library.
URL http://mineralscloud.com/resources/mineralsdatabase/pplibrary/

[6] J. Novotny, M. Russell, O. Wehrens, Gridsphere: an advanced portal framework, in: Proceedings. 30th Euromicro Conference, 2004., 2004, pp. 412–419. doi:10.1109/EURMIC.2004.1333997.

[7] R. M. Wentzcovitch, Y. G. Yu, Z. Wu, Thermodynamic properties and phase relations in mantle minerals investigated by first principles quasiharmonic theory, Reviews in Mineralogy and Geochemistry 71 (1) (2010) 59–98. doi:10.2138/rmg.2010.71.4.
URL https://doi.org/10.2138/rmg.2010.71.4.

[8] T. Qin, Q. Zhang, R. M. Wentzcovitch, K. Umemoto, qha: A python package for quasiharmonic free energy calculation for multi-configuration systems, Computer Physics Communications 237 (2019) 199–207. doi:10.1016/j.cpc.2018.11.003.
URL https://www.sciencedirect.com/science/article/pii/S0010465518303953

[9] J. J. Valencia-Cardona, G. Shukla, Z. Wu, C. Houser, D. A. Yuen, R. M. Wentzcovitch. Influence of the iron spin crossover in ferropericlase on the lower mantle geotherm, Geophysical Research Letters 44 (10) (2017) 4863–4871. arXiv:https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1002/2017GL073294, doi:https://doi.org/10.1002/2017GL073294.
URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2017GL073294

[10] G. Pizzi, A. Cepellotti, R. Sabatini, N. Marzari, B. Kozinsky, Aiida: automated interactive infrastructure and database for computational science, Computational Materials Science 111 (2016) 218–230. doi:10.1016/j.commatsci.2015.09.013.
URL https://www.sciencedirect.com/science/article/pii/S0927025615005620

[11] S. R. Baha, K. W. Jacobsen, An object-oriented scripting interface to a legacy electronic structure code, Computing in Science & Engineering 4 (3) (2002) 56–66. doi:10.1109/5992.998641.

[12] S. Curtarolo, W. Setyawan, G. L. Hart, M. Aykol, H. Tang, J. Neaton, S. P. Ong, K. Persson, A. Jain, Atomate: A high-level interface to generate, execute, and analyze computational materials science workflows, Computational Materials Science 139 (2017) 140–152. doi:10.1016/j.commatsci.2017.07.030.
URL https://www.sciencedirect.com/science/article/pii/S0927025617303919

[13] A. Dal Corso. Pseudopotentials periodic table: From H to Pu, Computational Materials Science 95 (2014) 337–350. doi:https://doi.org/10.1016/j.commatsci.2014.07.043.
URL https://www.sciencedirect.com/science/article/pii/S0927025614005187

[14] J. M. White, B. Kampinski, powerdistribution, M. Bouchet-Valat, S. Garborg, J. Quinn, C. Dubois, H. Harris, K. Squire, A. Arslan, pde, D. Anthoff, D. Kleinschmidt, A. Noack, V. B. Shah, A. Mellnik, T. Arakaki, T. Mohapatra, T. Qin, Q. Zhang, R. M. Wentzcovitch, Atomate: A python package for quasiharmonic free energy calculation for multi-configuration systems, Computer Physics Communications 237 (2019) 199–207. doi:10.1016/j.cpc.2018.11.003.
URL https://www.sciencedirect.com/science/article/pii/S0010465518303953

[15] J. M. White, B. Kampinski, powerdistribution, M. Bouchet-Valat, S. Garborg, J. Quinn, C. Dubois, H. Harris, K. Squire, A. Arslan, pde, D. Anthoff, D. Kleinschmidt, A. Noack, V. B. Shah, A. Mellnik, T. Arakaki, T. Mohapatra, Peter, S. Karpinski, D. Lin, timema, ExpandingMan, F. Oswald, L. White, JuliaData/DataFrames.jl: v0.22.1 (Nov. 2020). doi:10.5281/zenodo.4282946.
URL https://doi.org/10.5281/zenodo.4282946

[16] Q. Zhang. Pseudopotentials.jl documentation.
URL https://mineralscloud.github.io/Pseudopotentials.jl/stable

[17] Z. Wu, R. M. Wentzcovitch, Quasiharmonic thermal elasticity of crystals: An analytical approach, Phys. Rev. B 83 (2011) 184115. doi:10.1103/PhysRevB.83.184115.
URL https://link.aps.org/doi/10.1103/PhysRevB.83.184115

[18] J. Zhaang, H. Wang, Q. Zhang, R. M. Wentzcovitch, Thermodynamic properties of e—f with thermal electronic excitation effects on vibrational spectra, Phys. Rev. B 103 (2021) 144102. doi:10.1103/PhysRevB.103.144102.
URL https://doi.org/10.1103/PhysRevB.103.144102

[19] S. Baroni, P. Giannozzi, A. Testa, Green’s-function approach to linear response in solids, Phys. Rev. Lett. 58 (1987) 1861–1864. doi:10.1103/PhysRevB.83.184115.
[21] X. Gonze, Perturbation expansion of variational principles at arbitrary order, Phys. Rev. A 52 (1995) 1086–1095. doi:10.1103/PhysRevA.52.1086.

[22] K. Umemoto, R. M. Wentzcovitch, S. de Gironcoli, S. Baroni, Order–disorder phase boundary between ice vii and viii obtained by first principles, Chemical Physics Letters 499 (4) (2010) 236–240. doi:https://doi.org/10.1016/j.cplett.2010.09.065.

[23] J. Bezanson, a. Stefan Karpinski, Download Julia. URL https://julialang.org/downloads/

[24] a. Viral B. Shah, Building julia (detailed). URL https://github.com/JuliaLang/julia/blob/master/doc/build/build.md

[25] Q. Zhang, Express.jl documentation. URL https://mineralscloud.github.io/Express.jl/stable

[26] Q. Zhang, Express.jl development documentation. URL https://mineralscloud.github.io/Express.jl/dev

[27] Q. Zhang, Open an issue. URL https://github.com/MineralsCloud/Express.jl/issues/new/choose

[28] Q. Zhang, Welcome to Express.jl Discussions! URL https://github.com/MineralsCloud/Express.jl/discussions

[29] Q. Zhang, Examples. URL https://github.com/MineralsCloud/QuantumESPRESSOExpress.jl/tree/master/examples

[30] J. Sun, A. Ruizsinszky, J. P. Perdew, Strongly constrained and appropriately normed semilocal density functional, Phys. Rev. Lett. 115 (2015) 036402. doi:10.1103/PhysRevLett.115.036402.

[31] J. P. Perdew, A. Ruizsinszky, G. I. Csonka, O. A. Vydrov, G. E. Scuseria, L. A. Constantin, X. Zhou, K. Burke, Restoring the density-gradient expansion for exchange in solids and surfaces, Phys. Rev. Lett. 100 (2008) 136406. doi:10.1103/PhysRevLett.100.136406.

[32] B. B. Karki, R. M. Wentzcovitch, Vibrational and quasiharmonic thermal properties of CaO under pressure, Phys. Rev. B 68 (2003) 224304. doi:10.1103/PhysRevB.68.224304.

[33] J. F. Mammone, H. K. Mao, P. M. Bell, Equations of state of CaO under static pressure conditions, Geophysical Research Letters 8 (2) (1981) 140–142. arXiv:https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/GL008i002p00140. doi:https://doi.org/10.1029/GL008i002p00140.

[34] P. Richet, H.-K. Mao, P. M. Bell, Static compression and equation of state of CaO to 1.35 Mbar, Journal of Geophysical Research: Solid Earth 93 (B12) (1988) 15279–15288. arXiv:https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/JB093iB12p15279. doi:https://doi.org/10.1029/JB093iB12p15279.

[35] Q. Zhang, Express.jl homepage. URL https://mineralscloud.github.io/Express.jl/

[36] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, N. Wilkins-Diehr, Xsede: Accelerating scientific discovery, Computing in Science Engineering 16 (5) (2014) 62–74. doi:10.1109/MCSE.2014.80.