pyiron – an Integrated Development Environment for *ab initio* Thermodynamics

Dissertation
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)
vorgelegt dem
Department Physik der Fakultät für Naturwissenschaften
an der Universität Paderborn

Jan Janßen

Promotionskommission	
Vorsitzender	Prof. Dr. rer. nat. Cedrik Meier
Gutachter	Prof. Dr. rer. nat. Wolf Gero Schmidt
Gutachter	Prof. Dr. rer. nat. Jörg Neugebauer

Tag der Einreichung: 6. Juli 2021
Tag der mündlichen Prüfung: 2. November 2021

Abstract

From electric cars to human implants, specialized materials and metals in particular continue to drive innovation. Any change of electronic structure by chemical composition, thermodynamic phase stability by temperature, or defect density and grain structure by mechanical processing can affect the macroscopic material properties. The hierarchical nature of materials enables the adjustment of individual material properties and results in application-specific materials design. One of the remaining challenges is to purposefully exploit this structure-property relation and to differentiate long-lasting, energetically stable configurations from metastable and unstable ones. One approach to address this challenge are *ab initio* thermodynamics simulations, which combine parameter free – *ab initio* – models, originated in quantum chemistry with physical models for thermodynamics to calculate temperature, pressure and concentration dependent free energies. Based on these free energies the phase diagram quantifies which thermodynamic phase is energetically stable at a given concentration, pressure and temperature. With this theoretical approach the number of required experimental measurements is reduced, which accelerates the application-specific materials design. Still, the current limitation of *ab initio* thermodynamics is the hierarchy of models as a result of the hierarchical nature of materials. As many of these models are developed in rather different communities, the combination of two models requires the theoretical understanding of both, the practical experience with them and the technical skills to construct the interface between both of them.

To address this technical complexity, the pyiron integrated development environment (IDE) for *ab initio* thermodynamics is developed as part of this thesis. At its core the pyiron IDE consists of a class of objects, which each connect to the user interface, to the resources interface and to the data storage interface. As a consequence of this, these pyiron objects can be combined like building blocks to construct simulation protocols. In this thesis the pyiron IDE is introduced and applied to three current challenges in *ab initio* thermodynamics: (i) The propagation of the uncertainty in density functional theory (DFT) calculations with finite convergence parameters from the representation of the wave function to equilibrium material properties. (ii) The automated calculation of melting temperatures for interatomic potentials with the coexistence approach. (iii) The calculation of a temperature-concentration phase diagram for an interatomic potential with the quasi-harmonic approximation. These applications cover different approximations to compute atomistic energies with different levels of precision, the whole temperature range including the melting temperature and different levels of chemical complexity. Still, they are all based on the same three steps: (a) rapid prototyping of the simulation protocol, (b) up-scaling of the simulation protocol for a parameter study and (c) development of a coarse-grained model for the parameter space. Finally, the resulting coarse-grained models enable studying trends which were previously inaccessible based on the technical complexity. Thereby, the development of the pyiron IDE contributes to the understanding of the physical mechanisms controlling the materials properties at finite temperatures.

Contents

1 Introduction

The continuous development of new materials and in particular new metallic alloys drives innovation in all areas of life. From thinner laptops, to lighter cars and softer bone implants, a change in material properties extends the spectrum of applications. A thinner laptop is more portable, a lighter car drives same range with the less fuel and a bone implant which reproduces the elastic properties of natural bones does not need to be replaced every five to ten years. Therefore, the goal of materials design is not to just aim for light weight, high strength and durability but moreover metallic alloys are designed for specific applications. Two examples developed at the Max-Planck-Institut für Eisenforschung are a flexible non-toxic titanium alloy Ti-Nb for bone implants [1] and a low-cost light-weight ductile magnesium alloy Mg-1Al-0.1Ca for transportation [2]. Commonly, even different components of a product are produced from different materials, each specifically chosen depending on the requirements of the component.

Given the hierarchical nature of materials a small change on one level of the hierarchy can result in a change of the macroscopic material properties. Ranging from the electronic structure and chemical bonding of the individual atomistic species to its thermodynamic phases and the microscopic grain structure, every change of one level has an effect on the other levels. As a consequence, for a long time new alloys were designed top-down starting from a known stable alloy and varying individual parameters in the alloy composition or its processing to create a more suitable alloy for a given application. Still, such a local optimization in parameter space is commonly restricted to remain in the same local minimum. In contrast, a bottom-up approach which includes all possible configurations is capable of identifying the global minimum as well as all local minima. Unfortunately, experimentally this is impractical given the number of possible parameter combinations and the complexity to realize and synthesize them. Therefore, rather than realizing all possible combinations experimentally, theoretical models are employed to identify promising candidates. The predictive capability of such a theoretical model is governed by three factors: (i) the accurate inclusion of the relevant physics, (ii) the uncertainty propagation within the model and (iii) the numerical precision of the model.

One commonly used model to predict alloys is the CALculation of PHAse Diagams (CALPHAD) method [3]. It calculates the Gibbs energy of alloys based on the mixture of the individual phase constituents. For a given temperature, pressure and concentration the Gibbs energy of all possible thermodynamic phases is compared and the phase with the minimal free energy is predicted to be stable. In practise for a specific application, a given thermodynamic phase is selected based on its material properties and the phase diagram is used to predict the required concentration of the chemical species, the pressure and the

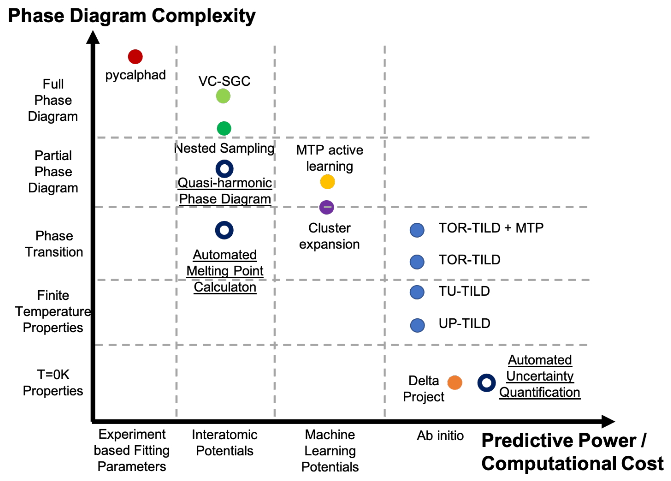


Figure 1.1: Overview of existing methods in the field of *ab initio* thermodynamics. Sorted by the predictive power of the atomistic engine on the x-axis and the phase diagram complexity on the y-axis. The different colours indicate separate developments and the open symbols highlight applications developed in this thesis. The individual methods are briefly introduced in Sec. 2.4.3.

temperature to stabilize this phase. While the CALPHAD method is widely used [4], its uncertainty is limited by the uncertainty of the Gibbs energy of the individual components and the uncertainty of the interpolation. Both of which can be assessed by comparing properties derived from the predicted Gibbs energy, e.g. the heat capacity, of an existing alloy to experimental measurements. As a consequence, the collection and curation of databases of Gibbs energies are the foundation of CALPHAD modelling.

1.1 Ab initio Thermodynamics

The Gibbs energies for CALPHAD modelling are commonly calculated by fitting experimental measurements of the heat capacity, or indirect measurements of derived properties, e.g. entropy and enthalpy, or more recently computed theoretically. Starting from the laws of quantum mechanics and including no parameters other than the fundamental constants the Gibbs energy can be computed using thermodynamics. This combination of parameter free - *ab initio* - methods to calculate the atomistic energies and forces with thermodynamic models to compute the free energy and derive finite temperature properties is referred to as *ab initio* thermodynamics [5]. Both the Ti-Nb alloy [1] and the Mg-1Al-0.1Ca alloy [2] mentioned above were first predicted with *ab initio* methods, before they were validated experimentally, which highlights the capability of this approach. In analogy to the hierarchical nature of a metallic alloy, also the *ab initio* thermodynamics models use a hierarchy of approximations for computing the Gibbs energy. For the calculation of atomistic energies these include the decoupling of the nuclei and the electron motion, the approximation of the electron-electron interaction as an interaction of a single electron with the density of all electrons, the approximation of the core electrons and finally the derivation of a temperature independent approximation for the interactions between the atoms without explicit calculation of the individual electrons. These different approximations to compute the atomistic energies and forces are implemented in specialised simulation codes, which depending on the community who develops the simulation code use different units, different variables names and in general a custom structure of input and output formats. Such a simulation code which calculates energies and forces with a given approximation is referred to as **atomistic engine**. In addition to these atomistic engines also the computing of the Gibbs energy, based on the atomistic energies and forces, requires approximations. These include the quantum nature of the atomic vibrations, the coupling of the electrons with the atomic vibrations, the anharmonic contributions to these vibrations, efficient sampling methods to compute the thermodynamic averages and the Lagrange transformation from the free energy to the Gibbs energy. These approximation in combination with their technical implementation in software utilities, are referred to as **thermodynamics modules**.

In practise the challenge of *ab initio* thermodynamics is the combination of a given atomistic engine and a compatible thermodynamics module to compute the temperature concentration dependent free energy. The visual summary in Fig. 1.1 compares different approaches based on the accuracy of the atomistic engine on the x-axis and the complexity of the thermodynamics modules to compute the free energy on the y-axis. The prediction of the phase diagram from *ab initio* in the upper right corner is the general aim of *ab initio* thermodynamics. Still the colour of the different symbols highlights the segmentation of the current developments. It is commonly a fixed combination of a single atomistic engine with a single thermodynamics module. Already the combination of two existing methods, developed in different communities requires not only a theoretical understanding of both, but also practical experience with the specific implementation in a given simulation software and technical expertise to develop a software interface to combine both methods. While the theoretical foundation of the methods is published in the literature, the details of the technical implementation are not formally documented. This includes the naming of variables, their units, recommended defaults values, required consistency checks and error messages. The required implicit expertise or technical complexity is the limiting factor for combining existing models. Still the combination is required to calculate the phase diagram from *ab initio* and predict new alloys in an automated way, as developing a single simulation code to include all methods is prohibitive. The open symbols in Fig. 1.1 highlight the *ab initio* thermodynamic challenges addressed in this thesis, which cover both a range of different atomistic engines as well as different thermodynamics modules up to the calculation of a phase diagram. This demonstrates the transferability of the new approach which is achieved by reducing the technical complexity.

1.2 pyiron – Integrated Development Environment

The challenge of combining various methods developed for *ab initio* thermodynamics is addressed in this thesis with the development of the pyiron integrated development environment (IDE) for *ab initio* thermodynamics. The pyiron IDE, like other IDEs, combines a source code editor, a build automation system and a debugger, still it is particularly focused on the development of simulation protocols. For this purpose it is based on an abstract class of objects, the pyiron objects, which can be combined like building blocks. The pyiron objects provide unified interfaces independent of the atomistic engine or the thermodynamics module and each of them is connected to the user interface, to the resource interface and to the data storage interface. With this separation the technical complexity is addressed in three steps:

Rapid Prototyping: With the pyiron IDE switching simulation engines or thermodynamics modules is as easy as changing a single variable. So initially, the simulation protocol is developed with a less accurate computationally affordable method and afterwards the precision and the complexity are increased iteratively one step at a time. This dynamic workflow simplifies the process of implementing a first version of a simulation protocol to test a given hypothesis.

Up-Scaling: The programmatic approach to developing simulation protocols in combination with data-science methods like map-reduce [6] enables the iteration over a parameter space with a thousand and more combinations. At the same time these parameter studies are the perfect test to validate that the implemented simulation protocol covers all the relevant physics.

Coarse-Grained Models: Based on the data sets from the parameter study new physical insights are derived to construct coarse-grained models. These models can then either be used for computationally efficient predictions or the systematic comparison of an even larger parameter space.

These three steps are independent of the applied atomistic engine or the thermodynamics module. To highlight the impact of the pyiron IDE and this concept for *ab initio* thermodynamics, this thesis is focused on the application of the pyiron IDE for partitioners: Starting with the theoretical foundation of *ab initio* thermodynamics in Chap. 2, covering density functional theory (DFT) and interatomic potentials as two representative atomistic engines, followed by molecular dynamics and thermodynamics, leading to complex simulation protocols of *ab initio* thermodynamics required to compute free energies. This complexity can be divided in a physical and a technical contribution. Based on identifying the technical contribution of the complexity as the limiting factor of *ab initio* thermodynamics, existing software developments are reviewed in Chap. 3. These software developments range from simulation codes for atomistic simulation, over high-throughput approaches in computational materials science and finally general developments in scientific computing which address the increasing complexity in computer simulation. Following the insight, that the complexity of *ab initio* thermodynamics is not sufficiently addressed by existing software developments in the atomistics community, the pyiron IDE is introduced with the focus on reducing the technical complexity to a minimum. For this purpose Chap. 4 introduces the simulation life cycle and the theoretical concepts of the pyiron IDE based on a series of examples to highlight the application of the pyiron IDE for partitioners. After this introduction the pyiron IDE is applied to three current challenges of *ab initio* thermodynamics:

1. The propagation of uncertainties for material properties calculated with DFT at finite convergence parameters is predicted in Chap. 5 with a coarse-grained model. The coarse-grained model inverts the classical convergence tests and predicts the computationally most efficient convergence parameters to achieve a user-defined level of uncertainty for a given material property. This is achieved in three steps: (a) A simulation protocol for the calculation of the material property at a specific set of convergence parameters is developed. (b) The simulation protocol is applied for a parameter study on a wide range of convergence parameters. (c) Based on the results of the parameter study the coarse-grained model is developed to predict the convergence. The coarse-grained model is then applied to compare the convergence of the bulk modulus for different face-centered-cubic (fcc) metals.
2. The melting temperature for interatomic potentials is predicted in Chap. 6. Again this is achieved in three steps: (a) A simulation protocol is developed to compute the melting temperature with a high precision using the coexistence method. (b) The simulation protocol is applied in a parameter study on an existing database of interatomic potentials. (c) A coarse-grained model is developed for the prediction of the melting temperature for interatomic potentials.
3. A temperature-concentration phase diagram for an interatomic potential is computed with the quasi-harmonic approximation in Chap. 7. Once more the same three steps are applied: (a) A simulation protocol for the quasi-harmonic approximation is developed. (b) It is up-scaled for a parameter study over the whole concentration range. (c) The resulting coarse-grained model, the phase diagram, is compared to existing theoretical predictions and experimental measurements.

While these three applications cover the whole range of atomistic engines and thermodynamics modules, as illustrated in Fig. 1.1, the three steps of (a) rapid-prototyping, (b) up-scaling and (c) developing a coarse-grained model remain the same. This highlights how the pyiron IDE is advancing *ab initio* thermodynamics towards a systematic prediction of new materials. Finally the concept of the pyiron IDE extends beyond *ab initio* thermodynamics as briefly discussed in the outlook of Chap. 8, which is concluding this thesis.

2 Theoretical Background

Ab initio thermodynamics is an interdisciplinary field combining quantum chemistry, physics and mathematics. Following the overview in Fig. 1.1 selected methods for calculating atomistic energies and forces (x-axis) as well as approximation to compute the free energy (y-axis) are introduced. The aim is the reduction of the dependence on experimental measurements in designing new materials. The methods are compared based on their predictive capability and their computational efficiency. For this comparison two types of errors are evaluated:

Accuracy/ Intrinsic Errors: Based on the neglection or approximation of selected contributions, the accuracy of a given method in comparison to experiment is intrinsically limited. For a selected method this error remains constant independent of the available computational resources. Still it varies for different chemical elements or different thermodynamics phases depending on the choice of approximation.

Precision/ Controllable Errors: In contrast to the intrinsic errors the controllable errors can be improved systematically with increasing computational resources. They include discretisation errors, restricted sampling times, numerical errors and the variation of different software implementations of the same method. The impact of the controllable errors and the resulting precision is controlled with convergence parameters. A higher convergence parameter results in higher computational cost and a higher level of precision.

An independent measurement of both types of errors is in practise impossible. It would require the calculation of the sum of intrinsic errors at infinite convergence parameters, by comparing to experiment and then calculating the error at finite convergence parameters, in reference to the infinite convergence parameters. Still based on the coupling of the controllable errors to the computational resources, such a calculation would require infinite computational resources. To separate the two types of errors in practice the controllable error is approximated to decrease continuously for sufficiently large increases in the convergence parameter. So based of a series of calculation at finite convergence parameters the controllable error at infinite convergence parameters is extrapolated. The remaining difference in comparison to experiment is approximated as the sum of intrinsic errors. For multiple convergence errors they can also be separated by extrapolating the contribution at infinite convergence parameters, while a separation of the intrinsic errors is only possible in comparison to other methods, which share selected approximation.

2.1 Density Functional Theory

As a first method to calculate energies and forces for atomistic structures density functional theory (DFT) is introduced. The formalism was developed in 1964 by Hohenberg, Kohn and Sham [7, 8] and has been applied to many material systems including metallic alloys once sufficient computational resources were available [9]. Its popularity in computational materials science is accounted to the accuracy of the calculation in comparison to experiment and other *ab initio* methods at comparably affordable computational cost [10]. In the following, DFT is briefly introduced based on the review articles [11–13], with the focus on the convergence parameters which control the precision of DFT calculation.

2.1.1 Born-Oppenheimer Approximation

Starting with the non-relativistic, time-independent many-body Schrödinger equation for a system with N_{nuc} nuclei and N_{el} electrons and their representative coordinates $\mathbf{R}_I$ and $\mathbf{r}_i$, the wave function can be written as:

$$\Psi_{nuc,\ el}(\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_{N_{nuc}}, \mathbf{r}_1, \mathbf{r}_2, \ldots \mathbf{r}_{N_{el}}) = \Psi_{nuc,\ el}(\{\mathbf{R}_I\}, \{\mathbf{r}_i\})\,, \tag{2.1}$$

and the corresponding Schrödinger equation as:

$$\hat{H}_{nuc,\ el}\Psi_{nuc,\ el}(\{\mathbf{R}_I\}, \{\mathbf{r}_i\}) = E_{nuc,\ el}\Psi_{nuc,\ el}(\{\mathbf{R}_I\}, \{\mathbf{r}_i\})\,. \tag{2.2}$$

In this equation $\hat{H}_{nuc,\ el}$ is the Hamilton operator for the combined many-body system and $E_{nuc,\ el}$ the corresponding energy level for a given electronic state. The Hamilton operator can be rewritten as:

$$\hat{H}_{nuc,\ el} = \hat{T}_{el} + \hat{T}_{nuc} + \hat{V}_{el} + \hat{V}_{nuc} + \hat{V}_{e-n}\,, \tag{2.3}$$

with $\hat{T}_{el}$ being the kinetic energy operator of the N_{el} electrons, $\hat{T}_{nuc}$ the kinetic energy operator of the N_{nuc} nuclei, $\hat{V}_{el}$ the electron-electron repulsion operator and analogous $\hat{V}_{nuc}$ the nucleus-nucleus repulsion operator and finally $\hat{V}_{e-n}$ the electron-nucleus attraction operator. They are defined as:

$$\hat{T}_{el}(\{\mathbf{r}_i\}) = -\sum_{i}^{N_{el}} \frac{\hbar^2}{2m_e}\nabla_i^2\,, \tag{2.4}$$

$$\hat{T}_{nuc}(\{\mathbf{R}_I\}) = -\sum_{I}^{N_{nuc}} \frac{\hbar^2}{2M_I}\nabla_I^2\,, \tag{2.5}$$

$$\hat{V}_{el}(\{\mathbf{r}_i\}) = \frac{1}{2}\sum_{i}^{N_{el}}\sum_{j\neq i}^{N_{el}} \frac{e^2}{4\pi\varepsilon_0|\mathbf{r}_i - \mathbf{r}_j|}\,, \tag{2.6}$$

$$\hat{V}_{nuc}(\{\mathbf{R}_I\}) = \frac{1}{2} \sum_{I}^{N_{nuc}} \sum_{J \neq I}^{N_{nuc}} \frac{Z_I Z_J e^2}{4\pi\varepsilon_0 |\mathbf{R}_I - \mathbf{R}_J|} , \tag{2.7}$$

$$\hat{V}_{e-n}(\{\mathbf{R}_I\}, \{\mathbf{r}_i\}) = -\sum_{i}^{N_{el}} \sum_{I}^{N_{nuc}} \frac{Z_I e^2}{4\pi\varepsilon_0 |\mathbf{r}_i - \mathbf{R}_I|} . \tag{2.8}$$

Here M_I defines the mass of a nucleus I, m_e the mass of an electron and Z_I the proton number of nucleus I. The equations are given in international units with the Planck constants $\hbar$ and vacuum permittivity ε_0. To solve this system, the Born-Oppenheimer approximation is applied to decouple the motion of electrons and the nuclei. It separates the many-body wave function $\Psi_{nuc,\ el}$ in a nucleus wave function Ψ_{nuc} and an electronic wave function Ψ_{el} based on the assumption that $M_I \gg m_e$. The electronic wave function Ψ_{el} is calculated at fixed ionic positions, so the nuclei coordinates are included as parameters, indicated by the semicolon:

$$\Psi_{el}(\{\mathbf{R}_I\}, \{\mathbf{r}_i\}) \rightarrow \Psi_{el}(\{\mathbf{r}_i\}; \{\mathbf{R}_I\}) . \tag{2.9}$$

The corresponding electronic Hamilton operator $\hat{H}_{el}$ is defined as:

$$\hat{H}_{el}(\{\mathbf{r}_i\}; \{\mathbf{R}_I\}) = \hat{T}_{el}(\{\mathbf{r}_i\}) + \hat{V}_{el}(\{\mathbf{r}_i\}) + \hat{V}_{e-n}(\{\mathbf{r}_i\}; \{\mathbf{R}_I\}) . \tag{2.10}$$

Finally the Schrödinger equation for the electrons can then be written as:

$$\hat{H}_{el}(\{\mathbf{r}_i\}; \{\mathbf{R}_I\}) \Psi_{el}(\{\mathbf{r}_i\}; \{\mathbf{R}_I\}) = E_{el}(\{\mathbf{R}_I\}) \Psi_{el}(\{\mathbf{r}_i\}; \{\mathbf{R}_I\}) . \tag{2.11}$$

Here the energy of the electrons $E_{el}(\{\mathbf{R}_I\})$ depends on the position of the nuclei $\mathbf{R}_I$ and is defined as the Born-Oppenheimer potential energy surface (PSE). When solving the Schrödinger equation for the nuclei, the PSE $E_{el}(\{\mathbf{R}_I\})$ is added in the Hamilton operator of the nuclei $\hat{H}_{nuc}$ to account for the immediate response of the electrons on changes of the nuclei positions $\mathbf{R}_I$:

$$\hat{H}_{nuc}(\{\mathbf{R}_I\}) = \hat{T}_{nuc}(\{\mathbf{R}_I\}) + \hat{V}_{nuc}(\{\mathbf{R}_I\}) + E_{el}(\{\mathbf{R}_I\}) , \tag{2.12}$$

$$\hat{H}_{nuc}(\{\mathbf{R}_I\}) \Psi_{nuc}(\{\mathbf{R}_I\}) = E_{nuc} \Psi_{nuc}(\{\mathbf{R}_I\}) . \tag{2.13}$$

Still, as only static DFT calculation with fixed ionic positions $\mathbf{R}_I$ are evaluated in this thesis the electronic wave function is rewritten as $\Psi_{el}(\{\mathbf{r}_i\}) = \Psi_{el}(\{\mathbf{r}_i\}; \{\mathbf{R}_I\})$.

2.1.2 Kohn-Sham Methodology

To solve the electronic Schrödinger equation with DFT the electronic density $n(\mathbf{r})$ and the energy functional of the electronic density $E_{el}[n(\mathbf{r})]$ are introduced:

$$n(\mathbf{r}) = N_{el} \int \mathrm{d}^3\mathbf{r}_2 \cdots \int \mathrm{d}^3\mathbf{r}_{N_{el}}\, \Psi^*_{el}(\{\mathbf{r}_i\})\, \Psi_{el}(\{\mathbf{r}_i\}) , \tag{2.14}$$

$$E_{el}[n(\mathbf{r})] = T_{el}[n(\mathbf{r})] + V_{el}[n(\mathbf{r})] + V_{e-n}[n(\mathbf{r})] . \tag{2.15}$$

In analogy to the electronic Hamilton Operator $\hat{H}_{el}$ the energy functional $E_{el}\left[n\left(\mathbf{r}\right)\right]$ consists of three contributions, the kinetic energy contribution $T_{el}\left[n\left(\mathbf{r}\right)\right]$, the electron-electron repulsion $V_{el}\left[n\left(\mathbf{r}\right)\right]$ and the attraction of the electrons to the nuclei $V_{e-n}\left[n\left(\mathbf{r}\right)\right]$. For this functional the Hohenberg-Kohn theorem [7] states: "The ground-state density of a bound system of interacting electrons in some external potential determines this potential uniquely." This theorem connects the ground state density $n_0(\mathbf{r})$ with the external potential $\hat{V}_{\text{ext}}(\mathbf{r})$:

$$\begin{tikzcd}
\hat{V}_{\text{ext}}(\mathbf{r}) \arrow[d] & n_0(\mathbf{r}) \arrow[l, "HK"'] \\
\Psi_n(\{\mathbf{r}_i\}) \arrow[r] & \Psi_0(\{\mathbf{r}_i\}) \,. \arrow[u]
\end{tikzcd} \tag{2.16}$$

The external potential $\hat{V}_{\text{ext}}(\mathbf{r})$ for a bulk calculation without any external fields, as they are used in this thesis, is defined by the attraction of the electrons to the nuclei $\hat{V}_{e-n}(\mathbf{r})$:

$$\hat{V}_{\text{ext}}(\mathbf{r}) = \hat{V}_{e-n}(\mathbf{r})\,. \tag{2.17}$$

To solve the Schrödinger equation for large numbers of electrons N_{el} Kohn and Sham [8] introduce an auxiliary system of non-interacting electrons with the density of the non-interacting electrons in the ground state $\tilde{n}_0\left(\mathbf{r}\right)$ being equal to the density of the ground state of the interacting electrons:

$$\tilde{n}_0\left(\mathbf{r}\right) = n_0\left(\mathbf{r}\right)\ . \tag{2.18}$$

The functional of the total energy for the interacting electrons $E_{el}\left[n_0\left(\mathbf{r}\right)\right]$ can be compared to the functional of the non-interacting electrons $\tilde{E}_{el}\left[n_0\left(\mathbf{r}\right)\right]$:

$$E_{el}\left[n_0\left(\mathbf{r}\right)\right] = T_{el}\left[n_0\left(\mathbf{r}\right)\right] + V_{el}\left[n_0\left(\mathbf{r}\right)\right] + V_{e-n}\left[n_0\left(\mathbf{r}\right)\right]\ , \tag{2.19}$$

$$\tilde{E}_{el}\left[n_0\left(\mathbf{r}\right)\right] = \tilde{T}_{el}\left[n_0\left(\mathbf{r}\right)\right] + \tilde{V}_{el}\left[n_0\left(\mathbf{r}\right)\right] + \tilde{V}_{e-n}\left[n_0\left(\mathbf{r}\right)\right]\ . \tag{2.20}$$

Based on the equality of the density of the ground state the attraction to the nuclei is equal for both the interacting and non-interacting electrons. In contrast the other contributions differ with the advantage that the electron-electron repulsion operator $\tilde{V}_{el}\left[n_0\left(\mathbf{r}\right)\right]$ for non-interacting electrons, which is identified as the interaction of individual electrons with an effective potential, can be computed for large numbers of electrons N_{el} [14]. The difference of both remaining contributions is summarized as exchange-correlation functional $E_{\text{xc}}\left[n_0\left(\mathbf{r}\right)\right]$:

$$\begin{aligned}
E_{el}\left[n_0\left(\mathbf{r}\right)\right] - \tilde{E}_{el}\left[n_0\left(\mathbf{r}\right)\right] &= T_{el}\left[n_0\left(\mathbf{r}\right)\right] - \tilde{T}_{el}\left[n_0\left(\mathbf{r}\right)\right] + V_{el}\left[n_0\left(\mathbf{r}\right)\right] - \tilde{V}_{el}\left[n_0\left(\mathbf{r}\right)\right] && (2.21)\\
&= E_{\text{xc}}\left[n_0\left(\mathbf{r}\right)\right]\ . && (2.22)
\end{aligned}$$

The exchange-correlation functional $E_{\text{xc}}\left[n_0\left(\mathbf{r}\right)\right]$ is unknown and it is one of the **intrinsic errors** of every DFT calculation. Multiple approximations for the exchange-correlation functional $E_{\text{xc}}\left[n_0\left(\mathbf{r}\right)\right]$ are introduced in Sec. 2.1.3. Based on the exchange-correlation functional $E_{\text{xc}}\left[n_0\left(\mathbf{r}\right)\right]$ the resulting approximation for the functional of the total energy for the interacting electrons $E_{el}\left[n_0\left(\mathbf{r}\right)\right]$ can be written as:

$$E_{el}\left[n_0\left(\mathbf{r}\right)\right] = \tilde{T}_{el}\left[n_0\left(\mathbf{r}\right)\right] + \tilde{V}_{el}\left[n_0\left(\mathbf{r}\right)\right] + E_{\text{xc}}\left[n_0\left(\mathbf{r}\right)\right]\ . \tag{2.23}$$

With the functional of the total energy for the interacting electrons $E_{el}\left[n_0\left(\mathbf{r}\right)\right]$ the effective potential of the non-interacting electrons $\hat{V}_{\text{eff}}(\mathbf{r})$, which leads to the equality of the densities of states for the ground state, is defined as:

$$\hat{V}_{\text{eff}}(\mathbf{r}) = \hat{V}_{\text{ext}}(\mathbf{r}) + \tilde{V}_{el}(\mathbf{r}) + \tilde{V}_{\text{xc}}(\mathbf{r}) \tag{2.24}$$

$$= \hat{V}_{\text{ext}}(\mathbf{r}) + \frac{1}{2}\int \frac{n_0\left(\mathbf{r}'\right)}{|\mathbf{r}_i - \mathbf{r}'|}d\mathbf{r}' + \frac{\delta E_{\text{xc}}\left[n_0\left(\mathbf{r}\right)\right]}{\delta n_0\left(\mathbf{r}\right)}\,. \tag{2.25}$$

Inserting the effective potential $\hat{V}_{\text{eff}}(\mathbf{r})$ in the Schrödinger equation for non-interacting electrons the non-interacting wave functions $\varphi_i(\mathbf{r})$ are determined:

$$\left(-\frac{\hbar^2}{2m_e}\nabla^2 + \hat{V}_{\text{eff}}(\mathbf{r})\right)\varphi_i(\mathbf{r}) = \varepsilon_i\varphi_i(\mathbf{r})\,. \tag{2.26}$$

By summation over the N_{el} electronic wave functions the ground state density of the non-interacting electrons $\tilde{n}_0(\mathbf{r})$ can be calculated, which equals the ground state density of the interacting electrons $n_0(\mathbf{r})$:

$$\tilde{n}_0(\mathbf{r}) = \sum_{i=1}^{N_{el}} f_i|\varphi_i(\mathbf{r})|^2 = n_0(\mathbf{r})\,. \tag{2.27}$$

Here f_i is the electronic "smearing" function for which different approximations are introduced in Sec. 2.1.4. In summary the combination of Eqs. 2.25, 2.26 and 2.27 enables the self-consistent calculation of the ground state energy. Extensions beyond ground state energies are available and required for certain *ab initio* thermodynamics models [15]. Still the thermodynamic models used in this thesis only require ground state energies to be calculated with DFT. So the following sections focus on the error contributions to ground state DFT calculations.

2.1.3 Exchange-Correlation Functional

With the exact exchange-correlation functional $E_{\text{xc}}\left[n_0\left(\mathbf{r}\right)\right]$ remaining unknown, different approximations have been developed for various material systems [9, 16]. They are sorted by complexity in the Jacob's ladder [17, 18]. The first level of the Jacob's ladder is the local density approximation (LDA), followed by the generalized gradient approximation (GGA), the Meta-GGA and the hybrid-methods. The later are primarily developed for molecular system. Still, with increasing complexity also the computational costs increase, which is the primary limitation for the application in *ab initio* thermodynamics. As a consequence commonly, only the first two levels of the Jacob's ladder, LDA and GGA, are considered. For selected elements LDA is found to be underbinding and GGA is found to be overbinding. This applies not only to the equilibrium volume, but also for thermodynamic properties the two approximations form an upper and an lower bound [19]. Both are briefly introduced in the following:

Local Density Approximation

LDA is the simplest approximation with the exchange-correlation energy derived from an uniform electron gas:

$$E_{xc}\left[n_0\left(\mathbf{r}\right)\right] = \int \varepsilon_{xc}\left(n_0\left(\mathbf{r}\right)\right) n_0\left(\mathbf{r}\right) d\mathbf{r}\,. \tag{2.28}$$

Here ε_{xc} is the exchange correlation energy per particle of the uniform electron gas. As a consequence the LDA is only exact for the uniform electron gas. Still it is widely used to this day. This accounts for the realization that the LDA obeys the sum rule that expresses the normalization of the exchange-correlation hole and the computational efficiency in comparison to higher levels of the Jacob's ladder [11].

Generalized Gradient Approximation

In addition to the density $n_0\left(\mathbf{r}\right)$ the GGA following the spirit of a Taylor expansion also includes the density gradient $\nabla n_0\left(\mathbf{r}\right)$:

$$E_{xc}\left[n_0\left(\mathbf{r}\right)\right] = \int \varepsilon_{xc}\left(n_0\left(\mathbf{r}\right), \left|\nabla n_0\left(\mathbf{r}\right)\right|\right) n_0\left(\mathbf{r}\right) d\mathbf{r}\,, \tag{2.29}$$

this accounts for the in-homogeneity of the exchange-correlation hole density. One commonly used GGA functional is the PBE-GGA [20] named after the authors John P. Perdew, Kieron Burke and Matthias Ernzerhof. In this thesis the PBE-GGA functional is used for all calculation, still the derived coarse-grained models are equally applicable to other approximations of the exchange-correlation functional.

2.1.4 Plane Wave Basis Set

Once the exchange-correlation functional $E_{\mathrm{xc}}\left[n_0\left(\mathbf{r}\right)\right]$ is approximated the next challenge is the computationally efficient representation of the wave functions of the non-interacting electrons $\varphi_i(\mathbf{r})$. This is achieved by defining a set of functions $\{\psi_\alpha\left(\mathbf{r}\right)\}_{\alpha=1}^{M}$ to represent the wave functions. Such a set of functions is referred to as **basis set**:

$$\varphi_i(\mathbf{r}) = \sum_{\alpha=1}^{M} c_\alpha \psi_\alpha\left(\mathbf{r}\right)\,. \tag{2.30}$$

Here c_α are the coefficients of the basis set and $\psi_\alpha\left(\mathbf{r}\right)$ are the basis functions of the basis set. The choice of basis set depends on the periodicity of the material system. For the calculations of crystal structures a supercell approach with periodic boundaries is favourable over an orbital based basis set or a real-space basis set. In the following the plane-wave basis set is introduced based on the review articles [21, 22].

Brillouin Zone

For a periodic crystal lattice, each lattice point $\mathbf{R}$ can be identified as a summation over the lattice vectors $\mathbf{a}_i$, multiplied by an integer n_i, with the lattice vectors $\mathbf{a}_i$ spanning the crystallographic unit cell:

$$\mathbf{R} = n_1\mathbf{a}_1 + n_2\mathbf{a}_2 + n_3\mathbf{a}_3 \,. \tag{2.31}$$

In addition based on the periodicity of the crystal structure, the reciprocal lattice is defined as the Fourier transform of the crystal lattice:

$$b_1 = 2\pi \frac{\mathbf{a}_2 \times \mathbf{a}_3}{\det(\mathbf{a}_1\mathbf{a}_2\mathbf{a}_3)} \,. \tag{2.32}$$

With b_2 and b_3 being defined analogously. Finally the unit cell of the reciprocal lattice is the Brillouin zone.

Bloch's Theorem

In accordance with Bloch's theorem, the wave function in a periodic crystal can be separated in a cell-periodic part and a wavelike part:

$$\varphi_i(\mathbf{r}) = \varphi_{n,\mathbf{k}}(\mathbf{r}) = u_{n,\mathbf{k}}(\mathbf{r})e^{i\mathbf{kr}} \,. \tag{2.33}$$

Here the cell-periodic part $u_{n,\mathbf{k}}(\mathbf{r})$ is defined by the periodicity requirement $u_{n,\mathbf{k}}(\mathbf{r}+\mathbf{R}) = u_{n,\mathbf{k}}(\mathbf{r})$. As a consequence it is sufficient to determine the wave function $\varphi_i(\mathbf{r})$ of a periodic crystal in the first Brillouin zone.

K-point Mesh

For this determination the cell-periodic part of the wave function $u_{n,\mathbf{k}}(\mathbf{r})$ is represented in a plane wave basis set:

$$u_{n,\mathbf{k}}(\mathbf{r}) = \sum_{\mathbf{G}} c_{n,\mathbf{k}}(\mathbf{G})\psi_{n,\mathbf{k}}(\mathbf{r}) = \sum_{\mathbf{G}} c_{n,\mathbf{k}}(\mathbf{G})e^{i\mathbf{G}\cdot\mathbf{r}} \,. \tag{2.34}$$

Based on the on the finding that close **k**-points have approximately similar wave functions a small sampling of **k**-points is sufficient. In the following the 3-dimensional **k**-point mesh for a cubic supercell is denoted as $q_x \times q_y \times q_z$ with κ_j **k**-points along each axis $\kappa_j = q_x = q_y = q_z$ and a total of κ_j^3 **k**-points. The **k**-point mesh with the convergence parameter κ_j controls the sampling of the wave function. With increasing κ_j the Brillouin zone sampling increases and the **controllable error** in the wave function and all derived properties decreases.

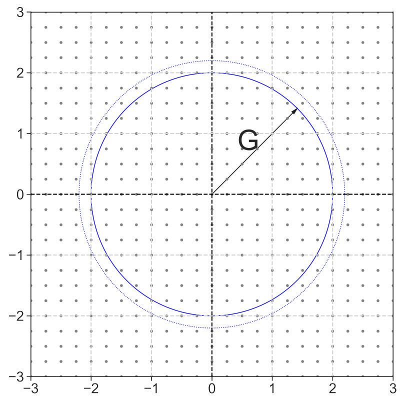


Figure 2.1: **k**-point mesh and ϵ_i: 2D illustration of a 4×4 **k**-point mesh in each Brillouin zone (light-grey squares) of a reciprocal lattice. When the energy cut-off is increased continuously (from the blue circle to the blue dotted circle) additional **k**-points are included. The discretisation of the **k**-point mesh results in a discontinuous increase in the number of plane waves, which causes fluctuation in the energy.

Energy Cut-off

The summation over all wave vectors **G** in Eq. 2.34 results in an infinite sum. So to further increase the computational efficiency a plane wave cut-off or energy cut-off ϵ_i is introduced to restrict the number of wave vectors **G** in the basis set:

$$\frac{\hbar^2}{2m_e}|\mathbf{G}+\mathbf{k}|^2 < \epsilon_i \,. \tag{2.35}$$

Only reciprocal **G** vectors that fulfil this inequality are considered. With increasing energy cut-off ϵ_i higher frequency contributions to the wave functions are included, which results in a more complete representation of the wave function $\varphi_i(\mathbf{r})$. As a consequence of the increasing completeness of the wave functions the total energy decreases. This decrease is homogeneous with increasing energy cutoff ϵ_i at the cost of increasing computational resource requirements. The energy cut-off ϵ_i is identified as the second convergence parameter. Increasing the energy cut-off, improves the the representation of the wave function in the plane wave basis set and decreases the **controllable error** related to the limited number of plane waves.

Plane Wave Jumps

The increase of the **k**-point mesh convergence parameter κ_j and the increase of the plane wave basis set convergence parameter, the energy cutoff ϵ_i, both reduce their corresponding controllable errors contributing to the total controllable error of the wave function. Still in addition to the individual controllable errors, the coupling of both results in a third **controllable error**, the plane wave jumps. The coupling of the energy cutoff and the

k-point mesh is illustrated in Fig. 2.1. The 2D 4×4 **k**-point mesh in each Brillouin zone (light-grey squares) is used to evaluate the wave function with the blue circle indicating the energy cutoff ϵ_i and limiting the maximum wave vector **G**. When the energy cut-off ϵ_i is increased (from the solid blue circle to the dotted blue circle) additional **k**-points are included for the representation of the wave function. As the consequence of the discretisation **k**-point mesh, the increase in the number of plane wave is discontinuous, even when the energy cutoff ϵ_i is increased continuously. This discontinuous increase in the total number of plane waves results in fluctuations in the energy - the plane wave jumps [23].

These fluctuations depend on both convergence parameters ϵ_i and κ_j. For a finer **k**-point mesh the frequency of the increases in the number of plane waves increases. As a consequence the amplitude of the plane wave jumps decreases. In the same way an increase of an already large energy cutoff leads to an inclusion of more discrete **k**-points, resulting in more individual jumps, each with a lower amplitude. So the plane wave jumps can be controlled with both convergence parameters ϵ_i and κ_j independently. In Chap. 5 the convergence of the three controllable errors is analysed in detail to predict the required convergence parameters ϵ_i and κ_j to achieve a predefined convergence goal.

Crystal Symmetry

By aligning the **k**-point mesh with the high symmetry planes and axis of the crystal lattice the number of **k**-points which have to be evaluated can be reduced. This is based on the wave function at symmetrically equivalent **k**-points being identical, so the calculation of the wave function at one of the symmetrically equivalent **k**-points is sufficient. The reduced set of **k**-points are commonly referred to as **irreducible k-points**. For the systematic and computationally efficient construction of the **k**-point mesh the Monkhorst Pack method [24, 25] is applied. It defines the positions of the **k**-points for a cubic crystal lattice with the lattice constant a and a **k**-point mesh with a grid of $q_x \times q_y \times q_z$ by the vector **k**:

$$\mathbf{k} = (k_{x,r}, k_{y,r}, k_{z,r})\pi/a\,. \tag{2.36}$$

With the components $k_{i,r}$ for the axis $\{x, y, z\}$ being a function of the total number of **k**-points in the corresponding direction q_i and an integer r_i from 1 to q_i:

$$k_{i,r} = \frac{2r_i - q_i - 1}{2q_i}\,. \tag{2.37}$$

The construction is illustrated in Fig. 2.2 for an 2D Monkhorst-Pack **k**-point mesh for a hexagonal crystal. The shaded region indicates the irreducible **k**-point mesh based on the crystal symmetry. More recently algorithms from informatics [26–29] have been introduced with the aim to generate computationally more efficient **k**-point meshes and improve the convergence of the controllable **k**-point mesh related error. Still the convergence of the controllable error remains a manual process, which requires human expertise and has to be repeated for every material system. To address this a coarse-grained model is developed in Chap. 5, which for a given convergence goal of the controllable error predicts the required

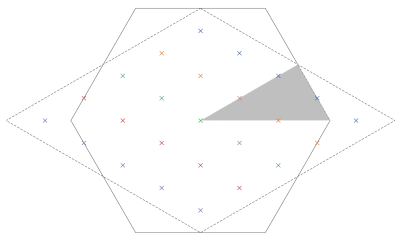


Figure 2.2: Monkhorst-Pack mesh: 2D illustration of a 5×5 $\mathbf{k}$-point mesh for a hexagonal lattice cell following the Monkhorst-Pack construction, with the shaded region highlighting the reduced $\mathbf{k}$-point mesh based on symmetry optimisation. Based on an illustration from [22].

convergence parameters ϵ_i and κ_j. This coarse-grained model is based on Monkhorst Pack method for comparison with existing studies, still it can be extended to include other method as well.

Electronic Smearing

The "smearing" function f_i is introduced for two purposes: On the one hand it represents the temperature dependent "smearing" of the electronic occupation and on the other hand it is introduced to interpolate between the discrete $\mathbf{k}$-points, especially around the Fermi energy. This interpolation improves the numerical stability related controllable error of the electronic self consistency field convergence. At the same time it introduces and additional intrinsic error of the interpolation. For an equilibrium calculation at $T_{\text{el}} = 0\,\text{K}$ the discrete $\mathbf{k}$-points represent delta functions $\delta(\mathbf{k})$ which sample the product of the occupation represented by a step function $S(\varepsilon_i(\mathbf{k}) - E_F)$ and the band energy $\varepsilon_i(\mathbf{k})$. The step function is 1 up to the Fermi energy E_F and 0 above the Fermi energy. This can be written as integration over the first Brillouin zone:

$$I_i = \int_{\text{BZ}} \varepsilon_i(\mathbf{k}) \cdot S(\varepsilon_i(\mathbf{k}) - E_F) d\mathbf{k} = \int_{\text{BZ}} \varepsilon_i(\mathbf{k}) \cdot f(\varepsilon_i(\mathbf{k})) d\mathbf{k}\,. \tag{2.38}$$

In this case I_i is the charge of the band $\varepsilon_i(\mathbf{k})$. Still this integration over the step function $S(\varepsilon_i(\mathbf{k}) - E_F)$ is computationally inefficient as the error in the determination of the Fermi energy E_F is directly related to the density of the $\mathbf{k}$-point mesh. So various "smearing" functions $f(\varepsilon_i(\mathbf{k}))$ have been introduced to interpolate between the discrete $\mathbf{k}$-points, especially around the Fermi energy E_F.

One physically motivated approach is the use of Fermi-Dirac distribution to account for the smearing of the electronic occupations at finite electronic temperatures T_{el} [15, 30]:

$$f(\varepsilon_i(\mathbf{k}_j), T_{el}) = \frac{1}{\exp\left(\left(\varepsilon_i(\mathbf{k}_j) - E_F\right) / \left(k_B T_{el}\right)\right) + 1} . \tag{2.39}$$

Based on the normalisation with the product of electronic temperatures T_{el} and the Boltzmann constant k_B the width of the smearing around the discrete position is related to the electronic temperatures T_{el}. In addition with the normalisation of the Fermi-Dirac distribution the method remains variational with respect to partial occupations. So the Fermi smearing methods is commonly applied for finite temperature calculations.

For equilibrium calculation at $T_{\mathrm{el}} = 0$ K the Methfessel-Paxton smearing [31] is commonly used. It is replaces the delta function of the discrete **k**-points with Hermite polynomials. This has the advantage that for a finite width σ of the distribution the results for $\sigma \to 0$ can be efficiently extrapolated. In analogy to the Fermi smearing it also remains variational with respect to partial occupations and is therefore suitable for crystal structure optimisations, to determine the ground-state structure.

Still for high precision calculation the finite width σ introduces another **controllable error** with convergence parameter σ. To address this limitation and alternative approach is the direct interpolation between the **k**-points in the first Brillouin zone with the tetrahedron method. This does not remain variational with respect to partial occupations and is therefore not suitable for the calculation of atomic forces, still by removing the additional convergence parameter it is the most suitable method for calculating total energies. The only limitation of the tetrahedron method is that the linear interpolation overestimates positive curvature and underestimates negative curvatures. This is improved with the Blöchl correction [32, 33] which leads to faster convergence with increasing convergence parameter κ_j without the requirement of an additional convergence parameter. Based on this comparison the tetrahedron method with Blöchl correction is selected for the uncertainty quantification in Chap. 5. In addition this enables the comparison with other high precision DFT calculation [34], which also use the tetrahedron method with Blöchl correction. A more detailed discussion of the different “smearing” methods and their various application is available in the review article [22].

Pseudopotentials

The selection of the irreducible **k**-points based on the crystal symmetry and the electronic smearing both decrease the required **k**-point mesh density controlled by the convergence parameter κ_j, still the energy cut-off ϵ_i remains unaffected. A high energy cut-off ϵ_i is required to resolve high frequency wave functions. As a consequence, plane wave basis sets are computationally efficient for the delocalized wave functions of the valence band electrons but they are less suitable for the core electrons. With decreasing distance from the nucleus the potential energy decreases and the kinetic energy increases. This increase in kinetic energy results in high frequency oscillations around the core region, which require a higher energy cut-off ϵ_i. At the same time the contribution of the core region, which requires a higher energy cut-off ϵ_i, to the bulk properties is negligible.

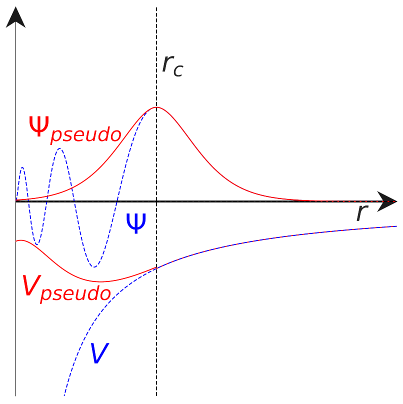


Figure 2.3: Pseudopotential: Comparing the wave function (upper blue dashed line) of the Coulomb potential (lower blue dashed line) with the approximated pseudopotential (lower red line), which neglects the high frequency oscillations of the wave function (upper red line) around the core region indicated by r_c.

To address this limitation pseudopotentials are introduced, which reproduce the potential for the valence and semi-core electrons, while the core electrons are are neglected. Such a pseudopotential is illustrated in Fig. 2.3. The lower blue line represents the Coulomb potential and the upper blue line is the corresponding wave function. In analogy the lower red line represents the pseudopotential and the upper red line the corresponding wave function. For the cut-off radius $r > r_{\text{cut}}$ both potentials and their corresponding wave functions agree, while below the cut-off radius $r < r_{\text{cut}}$ the potentials and the corresponding wave functions differ. The challenging part is to identify the cut-off radius r_{cut} or correspondingly divide the available electrons in core electrons and semi-core or valence electrons. Commonly DFT simulation codes provide multiple pseudopotentials with varying number of core electrons. For the construction of pseudopotentials various approaches are available, two of those are briefly introduced in the following:

- The linearized augmented plane wave plus local orbitals (LAPW+lo) [35] approach divides the real space in two distinct regions: Non-overlapping spheres centered around the positions of the ions, which represent the core region of the potential and are represented by an atomic like basis. The rest, the area outside the spheres, describes the smooth crystal wave function and is represented with a plane wave basis set. The challenge of this approach is to match both wave functions in real space at the edge of the spheres and in energy space which results in a high computational cost. At the same time this separation leads to a good agreement with all electron calculation and the highest accuracy.
- An alternative approach are the projector augmented wave functions (PAW) [36], which combines three different contributions. Here, the first contribution is the smooth wave function developed as plane waves over the full crystal. The second contribution are additional smooth wave functions defined inside the spheres around the positions

of the ions to cancelled the first contribution. This second contribution is developed in the same radial basis as the third contribution the strongly oscillating wave function inside the spheres. By combining the second and the third contribution, which are both defined inside the sphere and based on the same radial basis the matching of the wave function in real space and in energy has to be performed only once.

More recently, pseudopotentials that were primarily developed for calculating the self-energy of many-body systems using the Greens function G and the screened Coulomb interaction W (GW-Calculation) [37], have been identified empirically to be most accurate even for non-GW-calculations [34]. So, while no GW-calculation are used in this thesis the PBE-GGA GW pseudopotentials which were initially parameterized for GW-calculation are selected as default all DFT calculation.

Kinetic Energy Correction

Even with pseudopotentials, it is empirically observed that the total energy of a single DFT calculation over energy cut-off ϵ_i converges slower than the difference of two total energy calculations of the same atomistic structure at different volumes. This effect is based on the contribution of the core region to the convergence of a single DFT calculation, while this contribution cancels for energy differences. To address this, limitation the kinetic energy in the atomic limit at a finite energy cutoff is subtracted. This correction accelerates the convergence of total energy calculations over energy cut-off ϵ_i at the cost of the energy cut-off convergence being no longer homogeneous. Empirically, it is found that this correction addresses 80% of the total error for free electron metals [38]. As a consequence, this approximation is also included in all calculations of this thesis even though the improvement is limited to the convergence of individual energies.

2.2 Interatomic Potentials

Based on the introduction in the previous chapter DFT is widely used for calculation at a temperature of 0 K. This is the result of a low intrinsic error and low controllable errors at affordable computational costs in comparison to other *ab initio* methods [34]. The controllable errors are studied in more detail in Chap. 5. Still, the cubic scaling with the number of particles limits the time and length scales of DFT calculations. As a consequence, the calculation of thermodynamic properties, which requires the sampling of a large number configuration, as well as the development process of thermodynamic simulation protocols are limited. An alternative approach is starting from the Born-Oppenheimer approximation, introduced in Sec. 2.1.1 and predicting the PSE with a coarse grained model based on the atomic positions. So the explicit calculation of the electrons is neglected and the dynamics of the atoms are calculated based on classical mechanics rather than quantum mechanics. One type of these coarse grained models are interatomic potentials, which by restricting the interaction of the interatomic potential to a maximum real space cut-off R_{cut}, scale linearly. This is achieved, as the contributions to the energy E_I of atom I is limited to only the neighbouring atoms J with:

$$|\mathbf{R}_J - \mathbf{R}_I| < R_{\text{cut}}\,. \tag{2.40}$$

This linear scaling of interatomic potentials enables an atomistic understanding of otherwise inaccessible processes, like the coupling of material stiffness and hardness under nanoindentation [39]. These nanoindentation calculations require over a million atoms, which makes them inaccessible for DFT. At the same time the simulation results can be compared to experimental nanoindentations, even though the time scale of indentation in simulations is orders of magnitude smaller than the experimental time scale. The complexity of interatomic potentials is differentiated based on three categories:

Atomic Neighbourhood Descriptor: It provides a unique representation of the atomic coordinates while maintaining the crystal symmetries. These descriptors range from pair interactions to many-body terms [40].

Kernel: It maps the atomic neighbourhood descriptors to the potential energy. The implementation of the kernels range from analytical functions to neural networks [40].

Number of Fitting Parameters: With increasing number of fitting parameters the flexibility increases. .

With increasing flexibility it is possible to reproduce a more complex energy surface, while at the same time the computational cost increases. This results in a Pareto front of precision over computational cost [41–43]. In order to classify the reference data for the fitting of an interatomic potential, three levels are defined [44]:

1. **Level:** These are energies and forces calculated with DFT.
2. **Level:** These are material properties that can be calculated with DFT, e.g. the equilibrium bulk modulus.
3. **Level:** These are material properties that are inaccessible with DFT, but can be computed with interatomic potentials and measured experimentally, e.g. the hardness mearured by nanoindentation.

Historically, interatomic potentials were parameterised based on experimental measurements of the second and third level properties. They are referred to as classical interatomic potentials. In contrast, more recent interatomic potentials are primarily fitted to the first level and are commonly referred to as machine learning potentials based on the complexity of their kernels. As a consequence, the **intrinsic error** of such a machine learning interatomic potential is at best equal to the total error of its DFT reference data, which again highlights the need of quantifying the DFT controllable error, discussed in Chap. 5.

The primary advantage of the modern machine learning potentials is, that the process of fitting them is automated. Hence, the selection of the DFT reference calculation, the degree of many-body terms of the atomic neighbourhood descriptors, the parameterisation of the kernel and the cut-off radius R_{cut}, become convergence parameters. So the user of a machine learning potential fitting method, can adjust the corresponding **controllable errors**. In addition the different convergence parameters are coupled to the application of the interatomic potential, as the parameterisation of an interatomic potential for a single thermodynamic phase requires less complexity than the parameterisation of a transferable interatomic potential, which reproduces the full phase diagram. Opposing to the flexibility of the machine learning interatomic, classical interatomic potentials are typically parameterised once by the author and cannot be easily modified by the user of the interatomic potential. So for classical interatomic potentials the total error is not controllable and can be summarized as a combined **intrinsic error**. While the different types of interatomic potentials are discussed in more detail below, already the comparison of the different error types highlights the importance of addressing the technical complexity. By automating the process of fitting an interatomic potential and providing a systematic basis to extend both the descriptors of the atomic neighbourhood and the kernels the intrinsic error of classical interatomic potentials becomes controllable.

2.2.1 Classical Interatomic Potentials

In the subsequent part, selected classes of interatomic potentials for metallic systems are briefly introduced, following the review article [45].

Pair Potentials

The simplest interatomic potentials are the pair or two-body potentials with the potential being defined as a function of the distance between two atoms $\mathbf{R}_{IJ}$, respectively. The Morse type potential [46] is such a pair potential with the advantage that all potential parameters are level two properties, namely the dissociation energy D_0, the equilibrium bond distance R_0 and α, the width of the potential, which is related to the force constant at the equilibrium bond distance R_{eq}:

$$V_{\text{Morse}}(\mathbf{R}_{IJ}) = D_0 \left(e^{-2\alpha(R_{IJ}-R_0)} - 2e^{-\alpha(R_{IJ}-R_0)} \right) . \tag{2.41}$$

As a consequence, the Morse type potential can be parameterised directly with experimental measurements [47]. In addition, the Morse type potential can be used to analyse the dependence of the simulation result on the dissociation energy, equilibrium bond distance and equilibrium bond force constant. This enables parameter studies for level three properties like the hardness measured by nanoindentation [39]. The major limitation of pair potentials is that by construction the elastic constants C_{12} and C_{44} are equal, which is one of the Cauchy relations [48]. As a result, the cohesive energy equals the vacancy formation energy [49]. While for inert gases and ionic crystals these approximations are sufficient, they are incorrect for metals as the metallic nature of the bonds is neglected. Hence, the distance of surface atoms to the bulk is larger than the lattice distance in bulk for simulation with pair potentials while experimentally the opposite is true.

Embedded Atom Method

The embedded atom method (EAM) [50, 51] addresses this limitation of the pair potentials by adding many-body interactions depending on the electron density $\bar{\rho}_I$ to account for the metallic bonds. The total energy of the system is defined as:

$$E_{\text{tot}} = \frac{1}{2}\sum_{I \neq J} V_{\text{Pair}}(\mathbf{R}_{IJ}) + \sum_{I} F_I(\bar{\rho}_I)\,. \tag{2.42}$$

With the embedding function F_I being a function of the sum of the electron densities $\bar{\rho}_I$:

$$\bar{\rho}_I = \sum_{J \neq I} \rho_{\text{atom}}(\mathbf{R}_{IJ})\,. \tag{2.43}$$

The electron density of the surrounding atoms $\rho_{\text{atom}}(\mathbf{R}_{IJ})$ as well as the pair-potential $V_{\text{Pair}}(\mathbf{R}_{IJ})$ are parameterized depending on the specific type of EAM potential. In particular, as the electronic structure is not calculated explicitly for interatomic potentials, the electron density $\rho_{\text{atom}}(\mathbf{R}_{IJ})$ is approximated as a function of the atomic distances $\mathbf{R}_{IJ}$. This approach is sufficient for most metals but fails for directional bonds. In particular, sp-valent semi-conductors and sd-valent transition metals are described inaccurately [45]. In thermodynamic simulations EAM potentials are commonly used to study the temperature dependence of material properties like elastic constants [52–54] and more recently even the calculation of phase diagrams [55, 56]. This is enabled by the computationally affordable analytical form which is already sufficiently complex to include the required physics to predict the stability of different phases.

Modified Embedded Atom Method

To take into account the angular-dependence of directional bonds or three-body interactions, the electron density in the modified embedded atom method (MEAM) is treated as a tensor quantity [57]. With this, the total energy of the system is defined as:

$$E_{\text{tot}} = \frac{1}{2}\sum_{I \neq J} V_{\text{Pair}}(\mathbf{R}_{IJ}) + \sum_{I} U_I(n_I)\,, \tag{2.44}$$

with n_I:

$$n_I = \sum_{J \neq I} \rho_{\text{atom}}(\mathbf{R}_{IJ}) + \sum_{J<K J,K \neq I} f(\mathbf{R}_{IJ}) f(\mathbf{R}_{IK}) g[\cos(\theta_{IJK})]\,, \tag{2.45}$$

and θ_{IJK} as the angle between the bond of atom I and atom J - $\mathbf{R}_{IJ}$ and the bond of atom I and atom K - $\mathbf{R}_{IK}$. The remaining functions f, g as well as U and the pair potential V_{pair} have to be parameterised. The MEAM potentials are an extension to the EAM formalism. Still, the EAM potentials remain more popular which is indicated by the number of EAM potentials in existing potential databases compared to MEAM potentials [58–60].

2.2.2 Machine Learning Potentials

To automate the fitting of interatomic potentials, their construction is abstracted into two parts: The descriptors of the atomic neighbourhood Q_I for each atom I based on the atomic environment of all neighbouring atoms within the cutoff radius R_{cut} and a kernel E_{ML} for mapping the descriptors onto an energy :

$$V_{tot} = \sum_I E_{\mathrm{ML}}(Q_I)\,. \tag{2.46}$$

In this formalism the pair potentials, EAM potentials and MEAM potentials can be identified as analytical kernels of the atomic neighbourhood descriptors: pair distance $\mathbf{R}_{IJ}$, electron density $\bar{\rho}_I$ and angle θ_{IJK}.

Atomic Neighbourhood Descriptors

The atomic neighbourhood descriptors for the interatomic potentials are similar to the structure descriptors used for structure analysis, like the calculation of the Voronoi volume around each atom, the common neighbour analysis (CNA) or the Steinhardt parameters [61]. The aim of the atomic neighbourhood descriptors is to map the 3-dimensional atomistic structure to a scalar, while accounting for symmetry and still being able to differentiate unique configurations. So, the atomic neighbourhood descriptors for machine learning potentials are constructed as basis sets to describe atomic structures with increasing complexity [62].

Kernels

Besides, the more systematic atomic neighbourhood descriptors modern machine learning potentials also implement a wide range of different kernels ranging from linear regression models [43, 62–64] over Gaussian processes [65] to neuronal networks [66]. While potentials are commonly published as a fixed combination of atomic neighbourhood descriptors with the corresponding kernels in general arbitrary combination are possible.

Active Learning

Based on the automation of the fitting process in combination with the expandability of the atomic neighbourhood descriptors and the kernels, machine learning potentials enable the user to control the error of the parameterisation of the interatomic potential. This process can be further automated, by using the atomic neighbourhood descriptor space to differentiate between interpolation and extrapolation. Based on this criteria a given structure can be either evaluated by the interatomic potential or by DFT to extend the reference data set and reparameterise the interatomic potential. This process of continuously updating the interatomic potential with additional DFT calculation is referred to as active learning [67, 68] or on-the-fly force field generation [69].

The development of machine learning interatomic potentials is commonly referred to as one of the primary application of machine learning for atomistic simulation. Still this overview demonstrates that the development of extendable atomic neighbourhood descriptors and the automation of the fitting process also contribute to the success of the machine learning potentials in addition to their kernels. So the automation of existing simulation protocols is going to be one focus of this thesis. While the applications in this thesis focus on classical pair potentials and EAM potentials, the simulation protocols are developed independently of the type of interatomic potential. So they can also be applied to machine learning potentials. The only reason to choose classical potentials over machine learning potentials are their more affordable computational costs.

2.3 Molecular Dynamics

After the introduction of different methods for atomistic engines in the previous two sections, the focus of the next two sections is the calculation of finite temperature properties. For this purpose various thermodynamic approximation, along the y-axis of Fig. 1.1 are introduced, starting with molecular dynamics (MD). Based on the Born-Oppenheimer approximation introduced in Sec. 2.1.1 the motion of the nuclei is treated independently of the movement of the electrons. So the movement of the nuclei can then be approximated classically by solving Newton's equations of motion. This applies to both, the DFT calculation and the calculation with interatomic potentials. While the previous two sections covered the calculation of total energies for atomistic structures, the forces for plane wave DFT calculation can be calculated, based on the Hellmann–Feynman theorem [70] and for interatomic potentials based on the derivative of the potential energy surface. This includes interatomic potentials with complex kernels, e.g. the neuronal network potentials [66]. In the following, molecular dynamics for the sampling of thermodynamic properties is introduced based on the book of Marx and Hutter [71].

2.3.1 Verlet Integration

While solving Newton's equations of motion analytically is not possible for a system with several hundred atoms, it is possible to solve them numerically. By discretizing time into n equal timesteps of length Δt the Verlet integration [72] can be written as:

$$\mathbf{R}_I(t+\Delta t) = 2\mathbf{R}_I(t) - \mathbf{R}_I(t-\Delta t) + \frac{\mathbf{F}_I(t)}{M_I}\Delta t^2 + \mathcal{O}(\Delta t^4)\,. \tag{2.47}$$

With $\mathbf{F}_I(t)$ as the force acting on atom I, which equals a Taylor expansion in time. To improve the sampling of thermodynamic properties, it is reasonable to explicitly include

Abbr.	Nr. part.	Volume	Pressure	Energy	Enthalpy	Temperature
NVE	fixed	fixed		fixed		
NVT	fixed	fixed				Thermostat
NPT	fixed		Barostat			Thermostat
NPH	fixed		Barostat		fixed	
μVT		fixed				Thermostat

Table 2.1: Comparison of the available ensembles in atomistic simulations, their abbreviations (Abbr.) and their implementation with barostats and thermostats. Here Nr. part. defines the number of particles and the enthalpy is defined as $H = E + VT$.

the velocities $\mathbf{v}_I(t)$ in the propagation [73]:

$$\mathbf{R}_I(t+\Delta t) = \mathbf{R}_I(t) + \mathbf{v}_I(t)\Delta t + \frac{1}{2}\frac{\mathbf{F}_I(t)}{M_I}\Delta t^2 + \mathcal{O}(\Delta t^4)\,, \tag{2.48}$$

$$\mathbf{v}_I(t+\Delta t) = \mathbf{v}_I(t) + \frac{1}{2}\left[\frac{\mathbf{F}_I(t) + \mathbf{F}_I(t+\Delta t)}{M_I}\right]\Delta t + \mathcal{O}(\Delta t^3)\,. \tag{2.49}$$

Both variants of the Verlet integration are time invariant [74] and require only the forces $\mathbf{F}_I(t)$ to be calculated for a given configuration $\mathbf{R}_I(t)$. For the calculations in this thesis the timestep Δt is set to 1 fs, which results in a cut-off of the normal modes of frequencies in the THz regime.

2.3.2 Barostats and Thermostats

To equilibrate the atomic system at a finite temperature T, a thermostat is applied and alternatively a barostat can be applied to control the pressure p. Finally, the chemical potential μ can be adjusted by adding or removing atoms. With these, the following thermodynamic ensembles can be realised: microcanonical ensemble (NVE), canconical (NVT), grandcanconical (μVT), isothermal-isobaric (NPT) and isoenthalpic-isobartic (NPH). The different ensembles and their implementation using barostats and thermostats are summarized in Tab. 2.1. In the following, the stochastic Langevin method as well as the Nosé-Hoover method are briefly introduced.

Langevin Thermostat

The Langevin method describes the motion of the atoms as Brownian motion embedded in a sea of much smaller fictional particles [75]. The momenta $\mathbf{p}_i$ are defined as:

$$\frac{d\mathbf{R}_i}{dt} = \frac{\mathbf{p}_i}{M_i}\,. \tag{2.50}$$

With the derivative of the momenta defined as:

$$\frac{d\mathbf{p}_i}{dt} = \frac{\partial V(\mathbf{R})}{\partial \mathbf{R}_i} - \gamma \mathbf{p}_i + \delta p\,. \tag{2.51}$$

Here the damping constant γ represents the drag of the fictional particles and δp the random kicks given from the fictional particles to the atoms. These random kicks follow a Gaussian distribution:

$$\rho(\delta p) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{|\delta p|^2}{2\sigma^2}\right)\,. \tag{2.52}$$

With the standard deviation defined as:

$$\sigma^2 = 2\gamma M_i k_B T\,. \tag{2.53}$$

The stochastic approach of the Langevin method enables efficient computation which lead to its popularity.

Nosé-Hoover Thermostat

The second and more versatile method is the Nosé-Hoover method [76]. It extends the Hamiltonian with a heat bath resulting in an additional degree of freedom s. The resulting Hamiltonian can be written as:

$$\hat{H} = \sum_I \frac{{\mathbf{p}'_I}^2}{2M_I s^2} + \frac{1}{2}\sum_{I\neq J} V(\mathbf{R}'_I - \mathbf{R}'_J) + \frac{p_s^2}{2Q} + g k_B T \ln(s)\ , \tag{2.54}$$

with the virtual coordinates $\mathbf{R}'_I$ and $\mathbf{p}'_I$ defined as:

$$\mathbf{R}'_I = \mathbf{R}_I,\ \mathbf{p}'_I = \frac{\mathbf{p}_I}{s} \text{ and } t' = \int^t \frac{\mathrm{d}\tau}{s}\,. \tag{2.55}$$

and Q as the effective mass associated with s, g as the number of degrees of freedom, commonly $3N_{nuc} + 1$ and p_s as the momentum of s. With the additional degree of freedom the Nóse-Hoover method is often referred as an extended system method. A more detailed explanation with a focus on the implementation is available in the literature [77]. Following the constraints listed in Tab. 2.1 both methods can be used to realise either a thermostat or a barostat. Empirically the Langevin thermostat is found to perform better for small supercells, based on the stochastic damping force and the Nosé-Hoover thermostat is preferred for larger supercells, based on the explicit treatment of the heat bath. As a consequence the Langevin thermostat is mainly applied for DFT MD calculation while for MD calculation with interatomic potentials the Nosé-Hoover thermostat is commonly preferred. This also applies to the MD calculation in this thesis, in particular for the melting temperature calculation in Chap. 6.

2.4 Computing Free Energies

With MD methods from the previous section in combination with a atomistic engines to calculate energies and forces from Sec. 2.1 and Sec. 2.2 it is possible to compute atomistic trajectories at finite temperatures. Still the comparison of multiple phases in a single MD calculation is challenging, the more appropriate approach is to compute the free energies of the individual phases and compare those. This is in analogy to the CALculation of PHAse Diagrams (CALPHAD) method, which is introduced in Sec. 2.4.3. With thermodynamics simulation the energy eigenvalues of a given atomic configuration $E_\xi(V)$ are connected with the free energy $F(V,T)$. Using the change in the classical partition function $Z(V,T)$, which is based on the sum of all energy levels $E_\xi(V)$ in a given volume V at a specific temperature T, as starting point the free energy $F(V,T)$ is defined as:

$$Z(V,T) = \sum_\xi e^{-\beta E_\xi(V)} \text{ with } \beta = (k_B T)^{-1}\,, \tag{2.56}$$

$$F(V,T) = -\,k_B T \ln Z(V,T)\,. \tag{2.57}$$

The Gibbs energy $G(p,T)$ is then calculated from the free energy $F(V,T)$ using a Legendre transformation to compare phases at given temperatures T and pressures p:

$$G(p,T) = F(V,T) + pV\,. \tag{2.58}$$

Still, the limitation of this approach is that the partition function cannot efficiently be computed directly. Thus, two alternative approaches are introduced in the following par. Afterwards with the free energy the stability of a given phase at a given temperature T and pressure p can be derived by comparing the free energy of all relevant phases in the phase diagram, which is going to be the third part of this section. The phase with the lowest free energy is the thermodynamically stable one.

2.4.1 Adiabatic Approach

Under the assumption of the adiabatic theorem [78] it is possible to separate the different contribution of the free energy surface. Following the literature [79, 80] this decoupling can be written as:

$$F(V,T) = E_0(V) + F_{el}(V,T) + F_{qh}(V,T) + F_{ah}(V,T) + F_{mag}(V,T) + F_{cpl}(V,T)\,. \tag{2.59}$$

Starting with the equilibrium energy at a temperature of 0 K $E_0(V)$, the electronic contribution $F_{el}(V,T)$, the quasi-harmonic contribution $F_{qh}(V,T)$, the anharmonic contribution $F_{ah}(V,T)$, the magnetic contribution $F_{mag}(V,T)$ and finally coupling contributions $F_{cpl}(V,T)$ are added. The different contributions are briefly explained in the following:

Ground State Energy

The ground state energy $E_0(V)$ is evaluated at the equilibrium volume at $T = 0$ K, which can either be calculated by optimizing the volume of the supercell iteratively to achieve a

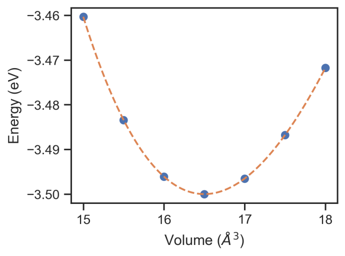


Figure 2.4: Energy-volume curve for a primitive aluminium supercell. The blue dots represent seven DFT calculations equally spaced around the initial guess of the equilibrium volume and the orange dashed line represents the Birch-Murnaghan fit of the seven DFT calculations.

minimal potential energy or by fitting an equation of state (EOS). Based on the plane wave jumps introduced in Sec. 2.1 the second option is preferred for cubic cells. The EOS were initially derived for experimental applications and have been adopted in the DFT community since then, such that they allow to derive the equilibrium properties, equilibrium energy E_0, equilibrium volume V_0, equilibrium bulk modulus B_0 and the derivative of the bulk modulus with respect to the pressure $B_0' = \frac{dB_0}{dp}$, directly from fitting a set of energy-volume pairs $E(V)$. This approach has two advantages for plane wave DFT calculation:

- On the one hand, as the total number of plane waves is correlated to the volume of the simulation cell, the calculations of individual energy volume pairs with varying number of plane waves reduces the Pulay stresses [81].
- On the other hand with the fitting of multiple energies $E(V)$ around the equilibrium volume the plane wave jumps [23] are interpolated, improving the prediction of the equilibrium properties.

In Fig. 2.4 an energy-volume curve for a primitive aluminium supercell is plotted (blue dots) in combination with the fit of the EOS. The commonly used EOSs in the DFT community are the Murnaghan EOS [82], the Birch-Murnaghan EOS [83] and the Vinet EOS [84]:

$$E_{\text{Murnaghan}}(V) = E_0 + B_0 V_0 \left[\frac{1}{B_0'(B_0'-1)} \left(\frac{V}{V_0} \right)^{1-B_0'} + \frac{1}{B_0'} \frac{V}{V_0} - \frac{1}{B_0'-1} \right] , \tag{2.60}$$

$$E_{\text{Birch-Murnaghan}}(V) = E_0 + \frac{9B_0V_0}{16} (\eta^2-1)^2 \left[6 + B_0'(\eta^2-1) - 4\eta^2 \right] , \tag{2.61}$$

$$\begin{aligned} E_{\text{Vinet}}(V) = & E_0 + 2 \cdot \frac{2B_0V_0}{(B_0'-1)^2} \\ & - \frac{2B_0V_0}{(B_0'-1)^2} \left[5 + 3B_0'(\eta-1) - 3\eta \right] \exp\left(- \frac{3(B_0'-1)(\eta-1)}{2} \right) . \end{aligned} \tag{2.62}$$

With $\eta = \left(\frac{V}{V_0} \right)^{\frac{1}{3}}$ being a relative measure for the distance from the equilibrium volume. While for $V = V_0$ all equations result in $E(V_0) = E_0$, the same energy-volume pairs $E(V)$

fitted with the three EOSs yield similar, but not exactly the same equilibrium parameters $\{E_0, V_0, B_0, B_0'\}$. This can be identified as **intrinsic error** of the corresponding EOS, or a **controllable error** in case the EOS can be systematically extended, e.g. a polynomial fit.

On an abstract level the fitting of the EOS is a coarse grained model with a hierarchy of errors. It has internal errors in calculating the equilibrium parameters from a given set of energy volume pairs and external errors which are related to the methodology used to calculate the energies for the energy volume pairs. Finally both error types are coupled, as fluctuation of the energy volume pairs are interpolated by the fit of the EOS. As a consequence, already the determination of the **intrinsic** and **controllable errors** for the equilibrium parameters $\{\Delta E_0, \Delta V_0, \Delta B_0, \Delta B_0'\}$ is challenging. At the same time, based on the hierarchy of the coupling of errors, the calculation of the equilibrium parameters is a prototypical application, which is computationally affordable compared to free energy calculation or the fitting of an interatomic potential. In Chap. 5 the different errors contributing to the uncertainty of the equilibrium parameters are selected to study the propagation of uncertainty systematically. Starting with the rapid prototyping of a simulation protocol, followed by a parameter study and finally a coarse grained model is developed to predict the uncertainty for the equilibrium parameters. With the coarse grained model it is then possible to compare the convergence of different fcc elements as well as calculate the required convergence parameters to achieve a user defined convergence goal.

Electronic Contribution

The second contribution to the free energy $F(V,T)$ is the electronic contribution $F_{el}(V,T)$. With the Fermi "smearing" method [15] the effect of the electronic temperature T_{el} can be separated, by comparing the volume dependence of the ground state energy $E(V)$ at $T_{el} = 0$ from the previous section, with the volume dependence of the energy $\tilde{F}_{el}(V,T)$ at a finite temperature T:

$$F_{el}(V,T) = \tilde{F}_{el}(V,T) - E(V)\,. \tag{2.63}$$

The electronic contribution $F_{el}(V,T)$ can then be interpolated based on the dependence to the electronic entropy $S_{el}(T)$:

$$F_{el}(V,T) \approx -T S_{el}(T)\,. \tag{2.64}$$

Here the electronic entropy is parameterised based on the Fermi distribution $f(\varepsilon_i(\mathbf{k_j}),T)$ and the density of states $D(\varepsilon_i(\mathbf{k_j}),T)$ as [85]:

$$S_{el}(T) = -k_B \int D(\varepsilon_i(\mathbf{k_j}),T) s_{el}(\varepsilon_i(\mathbf{k_j}),T) d\varepsilon\,, \tag{2.65}$$

$$s_{el}(\varepsilon_i(\mathbf{k_j}),T) = [f(\varepsilon_i(\mathbf{k_j}),T)\ln f(\varepsilon_i(\mathbf{k_j}),T) + (1 - f(\varepsilon_i(\mathbf{k_j}),T)\ln(1 - f(\varepsilon_i(\mathbf{k_j}),T))] \tag{2.66}$$

Following the discussion in the previous section, the **intrinsic error** of the electronic contribution is the limitation of the fit of the electronic entropy $S_{el}(T)$ and the **controllable**

errors are related to the convergence parameters of the fit, the number of volumes V and temperatures T which are included. These errors again couple with the errors of the individual total energy calculation for both the ground-state energy and the energy at finite electronic temperatures. With the separation of the different contributions in the adiabatic approach it is possible to identify the importance of the electronic contribution for selected elements in particular for temperatures below $T = 100$ K [86].

Quasi-Harmonic Contribution

The third contribution of the adiabatic approach is the quasi-harmonic contribution. The vibrational energy levels are populated following the Bose-Einstein statistics, instead of a single frequency, like it is used in the Einstein harmonic oscillator, $3N$ frequencies, with N being the number of atoms in the supercell, are used. For the quasi-harmonic approximation the phonons with the frequencies $\omega_i(V)$ are non-interacting but temperature-dependent in contrast to the harmonic approximation which does not contain the volume expansion with temperature, which affects the resulting phonon frequencies:

$$F_{qh}(V,T) = \frac{1}{N}\sum_{i}^{3N}\left\{\frac{\hbar\omega_i(V)}{2} + k_B T \ln\left[1 - \exp\left(-\frac{\hbar\omega_i(V)}{k_B T}\right)\right]\right\}. \tag{2.67}$$

By using the quantum mechanical oscillator rather than the classical oscillator the zero-point energy is included, the energy levels are quantised and equally spaced. The phonon frequencies can be calculated using the finite displacement method, which displaces atoms within the harmonic regime from the equilibrium position at 0 K and measures the resulting forces. The dynamical matrix $D_{k,l}(V,T)$, independent of the electronic excitations, can be written as:

$$D_{k,l}(V,T) := \frac{1}{M}\left[\frac{\partial^2 E_0(V)}{\partial R_k \partial R_l}\right]. \tag{2.68}$$

To include the electron-phonon coupling non-adiabatic contributions are required. For the adiabatic approach these are approximated with the coupling term $F_{cpl}(V,T)$ [87].

Anharmonic Contribution

Similarly, the explicit anharmonic contribution $F_{ah}(V,T)$ cannot be calculated directly. Instead of calculating the explicit anharmonic contributionm, it is necessary to calculate the full vibrational contribution:

$$F_{vib}(V,T) = F_{qh}(V,T) + F_{ah}(V,T). \tag{2.69}$$

The explicit anharmonic contribution is then derived by subtracting the quasi-harmonic contribution $F_{qh}(V,T)$ from the full vibrational free energy $F_{vib}(V,T)$. Nevertheless, based on the complexity to compute the full vibrational free energy $F_{vib}(V,T)$ the anharmonic contributions is commonly neglected. This results in an **intrinsic error** of the adiabatic approach [19].

Magnetic Contribution

This thesis is focused on non-magnetic elements, so the magnetic contributions are also neglected. A detailed comparison of the currently available methodologies and their applications is available in the literature [88, 89].

Coupling Term

The coupling terms of electron-phonon coupling, electron-magnon coupling and magnon-phonon coupling represent the non-adiabatic contribution to the free energy. While it is possible to compute these coupling terms they are commonly neglected in the adiabatic approach for non-magnetic elements. Again resulting in an additional **intrinsic error** of the adiabatic approach [87].

2.4.2 Thermodynamic Integration

A secondary approach is to derive the free energy directly from an DFT molecular dynamics [90] calculation. While this approach does not allow to separate the different contributions, like the adiabatic approach does, it already includes the coupling terms and the anharmonic contribution. Still two challenges remain:

- The quantum mechanical contribution to the vibrational free energy $F_{vib}(V,T)$ and its coupling to the other contributions is missing. This is the result of choosing a classical molecular dynamics approach, introduced in Sec. 2.3, rather than a quantum mechanical path integral [91].
- The molecular dynamics calculation approximate the internal energy, still the free energy cannot be computed directly without a reference system.

To address these limitations thermodynamic integration is applied to calculate the free energy difference $F_{\text{int}}(V,T)$ of the classical harmonic oscillator and the molecular dynamics calculation [92]. The free energy is then calculated as:

$$F(V,T) = E_0(V) + F_{\text{qh, el}}(V,T) + F_{\text{int}}(V,T)\,. \tag{2.70}$$

In this case $F_{\text{qh, el}}(V,T)$ includes both the electronic and the quasi-harmonic contribution and is computed based on the phonons calculated at finite electronic temperatures.

Method

By mixing the forces according to the coupling parameter λ, the internal energy $U(V,T)$ as a function of the coupling parameter can be written as:

$$U_\lambda(V,T) = (1-\lambda)U_0(V,T) + \lambda U_1(V,T)\,. \tag{2.71}$$

The resulting change of free energy ΔF can be calculated directly from the internal energy $U_\lambda(V,T)$:

$$\Delta F = \int_0^1 d\lambda \langle dU_\lambda / d\lambda \rangle_\lambda = \int_0^1 d\lambda \langle U_1 - U_0 \rangle_\lambda \,. \tag{2.72}$$

Combining this with the adiabatic approach the total free energy can be calculated as:

$$F_\text{int}(V,T) = \int_0^1 d\lambda \langle U_\text{DFT} - U_\text{qh, el} \rangle_\lambda \,. \tag{2.73}$$

Here, the DFT internal energy $U_\text{DFT}(V,T)$ is calculated with the corresponding Fermi "smearing" at the corresponding temperature T. It therefore requires a high $\mathbf{k}$-point mesh sampling.

Upsampled Thermodynamic Integration using Langevin Dynamics

While the thermodynamic integration from the quasi-harmonic internal energy with electronic coupling $U_\text{qh, el}$ to the fully anharmonic DFT internal energy U_DFT is in principle possible, it is computationally very expensive given the number of DFT timesteps Δt that have to be computed. The number of timesteps Δt required for getting the free energy contribution of the thermodynamic integration $F_\text{int}(V,T)$ converged depends on the similarity of the two internal energy surfaces. Other alternative references for the thermodynamic integration have been tested like the Einstein crystal [93], still the limitation remains. To address this inefficiency an intermediate reference system is introduced, which can be computed efficiently and which is as close as possible to the DFT reference. One choice is a DFT calculation with low convergence parameters. As a consequence, the thermodynamic integration is divided into two parts [19]:

$$F_\text{int}(V,T) = \int_0^1 d\lambda \langle U_\text{DFT, low} - U_\text{qh, el} \rangle_\lambda + \int_0^1 d\lambda \langle U_\text{DFT, high} - U_\text{DFT, low} \rangle_\lambda \,. \tag{2.74}$$

By integrating from the quasi-harmonic internal energy with electronic coupling $U_\text{qh, el}$ first to a reference DFT internal energy with low convergence parameters $U_\text{DFT, low}$ and afterwards from this intermediate reference to a DFT internal energy with high convergence parameters $U_\text{DFT, high}$ the total computational time can be reduced. The first integration still requires 10^4 steps while for the second less than 100 steps are sufficient [19]. In practice, the second integration is even independent of λ and mainly a constant shift. Therefore, it is sufficient to calculate the shift based on an average over independent samples:

$$\langle \Delta E \rangle_\text{up} = \frac{1}{M} \sum_{u=1}^{M} (E_\text{DFT, low} - E_\text{DFT, high}) \,, \tag{2.75}$$

with M, the number of independent structures. In this new approach, the upsampled thermodynamic integration using Langevin dynamics (UP-TILD) [19], the free energy of a phase is finally calculated as:

$$F(V,T) = E_0(V) + F_{qh,el}(V,T) + \int_0^1 d\lambda \langle U_{DFT,low} - U_{qh,el} \rangle_\lambda - \langle \Delta E \rangle_{\text{up}} \,. \tag{2.76}$$

With the combination of the thermodynamic integration and the up-sampling the DFT calculation with the low convergence parameters require the most computational resources. So the approach can be further improved by identifying an computationally more efficient reference.

Two-stage Sampled Thermodynamic Integration using Langevin Dynamics

Based on the capabilities of the interatomic potentials introduced in Sec. 2.2 the two-stage upsampled thermodynamic integration using Langevin dynamics (TU-TILD) method was introduced [94]. It is using interatomic potentials as a second intermediate reference to accelerate the thermodynamic integration to the free energy with low convergence DFT calculation. The free energy is calculated as:

$$\begin{aligned} F(V,T) = E_0(V) + F_{qh,el}(V,T) &+ \int_0^1 d\lambda \langle U_{\text{pot}} - U_{\text{qh, el}} \rangle_\lambda \\ &+ \int_0^1 d\lambda \langle U_{\text{DFT, low}} - U_{\text{pot}} \rangle_\lambda - \langle \Delta E \rangle_{\text{up}} \,. \end{aligned} \tag{2.77}$$

Here, U_{pot} is the internal energy calculated from the interatomic potential. In order to have a sufficient agreement of the internal energy at a given temperature, it is necessary to fit an interatomic potential specifically to the thermodynamic integration of a given phase at a given concentration. In contrast to other interatomic potentials, these specific interatomic potentials are not optimised for transferability, but rather for this specific use case only. To additionally limit the number of required DFT calculations, only DFT calculations at the highest temperature required for the thermodynamic integration are considered for the parameterisation of the interatomic potential. As a result the parameterisation is less accurate at low temperatures or for calculations that only consider small displacements around the equilibrium position, like the calculation of phonons.

For the parameterisation of the interatomic potential, 1000 uncorrelated DFT snapshots are used. Thereby, the following thermodynamic integration from the quasi-harmonic internal energy with electronic coupling $U_{\text{qh, el}}$ to the internal energy of the potential U_{pot} requires only 150 steps [94], which equals a speed up to a factor 50. Finally, more recently this methodology has been combined with machine learning interatomic potentials introduced in the Sec. 2.2.2. The increased flexibility of these interatomic potentials enables the extension of the chemical space beyond unaries and binaries up to five component high entropy alloys [95]. With these extensions – the UP-TILD and TU-TILD method – as well as combining them with machine learning interatomic potentials, it is now possible to calculate thermodynamic properties up to the melting point as illustrated in Fig. 1.1. The important achievement of the TILD methods is that many of the **intrinsic errors** of the adiabatic approach are replaced by **controllable errors**. This is in analogy to the comparison of classical interatomic potentials and machine learning interatomic potentials in Sec. 2.2.

2.4.3 Phase Diagrams

Following the calculation of the free energy, the Gibbs energies of different thermodynamic phases are compared to identify the one with the lowest Gibbs energy as the stable phase for a given temperature T and pressure p. These *ab initio* predictions can be compared to existing predictions of the phase stability based on the CALculation of PHAse Diagrams (CALPHAD) method [3].

Calculation of Phase Diagrams

The CALPHAD method calculates the Gibbs energy of any phase based on the mixture of the individual chemical elements. For this the Gibbs energy is approximated as:

$$G_m^{\phi} = {}^{\mathrm{ref}}G_m^{\phi} + {}^{\mathrm{cfg}}G_m^{\phi} + {}^{\mathrm{phy}}G_m^{\phi} + {}^{\mathrm{exs}}G_m^{\phi}\,. \tag{2.78}$$

It combines the reference Gibbs energy ${}^{\mathrm{ref}}G_m^{\phi}$ based on the weighted Gibbs energies of the individual endmembers with the configurational Gibbs energy${}^{\mathrm{cfg}}G_m^{\phi}$ and the physical Gibbs energy ${}^{\mathrm{phy}}G_m^{\phi}$ which includes physical effects like magnetism and the additional term ${}^{\mathrm{exs}}G_m^{\phi}$ which includes all other contributions to the Gibbs energy. Based on this interpolation the solution of two elements can be written as the Redlich-Kister polynomial [96]:

$$G_m^{\phi} = x_A {}^{\circ}G_A + x_B {}^{\circ}G_B + RT\left\{x_A \ln x_A + x_B \ln x_B\right\} + x_A x_B \sum_{i=0}^{n_{AB}} L_{A,B:i}(x_A - x_B)^i\,. \tag{2.79}$$

Here x_A is the mole fraction of element A, x_B is the mole fraction of element B, ${}^{\circ}G_A$ is the stability of element A in the sublattice, ${}^{\circ}G_B$ is the stability of element B in the sublattice, R is the molar gas constant and $L_{A,B:i}$ are the excess parameters. Based on a large database of experimental measurements, the CALPHAD method has been successfully applied in both research and industry [4].

Towards ab initio Phase Diagrams

Therefore, one goal of *ab initio* thermodynamics is to extend the existing experimental databases, either by adding DFT calculation for metastable phases which are inaccessible experimentally or ideally by being able to calculate the phase diagram fully from *ab initio* [97, 98]. As illustrated in Fig. 1.1 by the different colours the progress towards *ab initio* phase diagrams is highly segmented. Each approach is a fixed combination of a given **atomistic engine**, a corresponding **thermodynamic approximation** and a **atomistic structure generator**. The approaches range from high precision DFT calculation as part of the Delta project [34], focusing on the equilibrium parameters, over the TILD approaches explained in Sec. 2.4.2, to cluster expansion [99] and machine learning interatomic potentials with active learning [68], up to approaches to calculate the full phase diagram with free energies like variance constrained semi-grand-canonical ensembles [100, 101] or nested sampling [102–104] and finally calculating the phase diagram with the CALPHAD methodology using pycalphad [105]. While the first interatomic potentials are published with their corresponding phase diagrams [45, 55, 56, 106] a systematic evaluation of the **intrinsic** and **controllable errors** for predicted phase diagrams is hindered by the technical complexity.

Atomistic Structure Generators

In particular the addition of the chemical complexity represented by the atomistic structure generators remains challenging, as it drastically increases the number of required calculation. The structure generators already pre select promising candidate structures, either based on their entropy, e.g. special quasirandom structures (SQS) [107, 108] or based on their symmetry [109–112]. Still the combination of the remaining candidates for atomistic structures with the thermodynamic approximation currently prohibits the systematic sampling of the phase diagram.

2.4.4 Melting Temperature

To highlight the complexity of comparing the free energy for two thermodynamic phases, the calculation of the melting temperature for a unary as the comparison of the stability of the solid and liquid phase is introduced [113]. The melting temperature T_{melt} is defined as the point where the Gibbs energies of the solid and the liquid phase are equal:

$$G_{\mathrm{solid}}(V, T_{\mathrm{melt}}) = G_{\mathrm{liquid}}(V, T_{\mathrm{melt}})\,, \tag{2.80}$$

$$G(V,T) = F(V,T) + pV \text{ with } p = -\left(\frac{\partial F}{\partial V}\right)_T . \tag{2.81}$$

The challenge of applying the free energy comparison directly is on the one hand the selection of a reference with a known free energy for the liquid phase and on the other hand the determination of the melting temperature for free energies with similar slopes in the temperature dependence. For a similar slop already a small uncertainty in the free energy results in a large uncertainty of the melting temperature. As a consequence, the uncertainty in the prediction of this approach is commonly in the order of ±100 K [114, 115].

Mechanical Methods

An alternative to calculating the free energy are molecular dynamics simulation, introduced in Sec 2.3, at different temperatures to identify the phase transition of the solid to the liquid phase based on the change of mechanical properties and lattice instabilities [116]. Two commonly used criteria are:

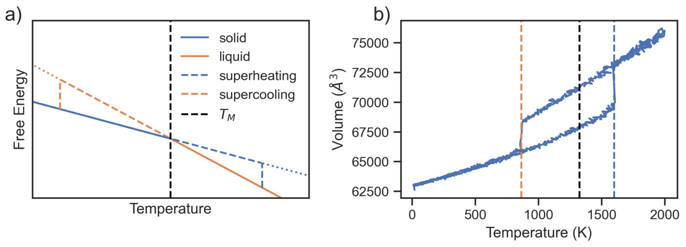


Figure 2.5: The melting temperature T_M defined (black dashed line) based on the free energy (a) and on the volume expansion over temperature (b). The volume expansion results in a hysteresis with the phase transition from solid to liquid denoting the superheating temperature (dashed blue line) and the liquid to solid phase transition denoting the supercooling temperature (dashed orange line).

Lindemann Criterion: It defines the melting temperature T_M based on the thermal root mean square displacement of the atoms in relation to the nearest neighbour distance [117]. Typically, a ratio of $\approx$ 12% [118] is defined as the melting condition. In this case the short range order in the liquid remains similar to the solid, but the long range order vanishes.

Born Criterion: Three conditions are proposed for the elastic constants at the melting temperature: The shear modulus and the diagonal elastic constant vanishes $C_{44} = 0$, the elastic constants become isotope $C_{11} = C_{12}$ like a gel and the sublimation condition $C_{11} + 2C_{12} = 0$. The first condition was confirmed by Born for a bcc crystal and is commonly referred to as the Born criterion [119].

The limitation of both of these criteria is, that the solid can be heated beyond the melting temperature, remains solid and only transitions to the liquid phase at the **superheating temperature**. In analogy the liquid can be cooled below the melting temperature, remains liquid and only solidifies at the **supercooling temperature** [53]. Both effects are illustrated in Fig. 2.5 a) for the free energy with the solid blue line for the solid phase, the dashed blue line for the solid phase during superheating, the orange line the liquid phase and the dashed orange line the supercooling and in analogy b) for the hysteresis of the volume expansion over temperature.

Coexistence Method

With the coexistence approach [120–122], which simulates the interface of a solid and a liquid phase, this can be addressed. The interface acts as precursor for nucleation [123] with already a small gradient in the free energy causing the expansion of the phase with the lower free energy [124]. Finally the coexistence approach can be combined with the calculation of free energies, based on the UP-TILD and TU-TILD methodology, introduced in Sec. 2.4.2, resulting in the two-optimized references thermodynamic integration using Langevin dynamics (TOR-TILD) method [125]. It is based on five steps to calculate an absolute free energy surface for the liquid phase, which is constructed from the free energy of the solid phase at the point of thermal equilibrium with the liquid phase:

Step 0. Following the TU-TILD approach explained in the previous section, TOR-TILD requires the DFT free energy surface for the solid $F^{\text{solid}}_{\text{DFT}}(V,T)$ to be already calculated with the corresponding reference free energy surface $F^{\text{solid}}_{\text{ref 1}}(V,T)$ being calculated with an interatomic potential $V_{\text{ref 1}}$ as well as the free energy surface for the liquid $F^{\text{liquid}}_{\text{DFT}}(V,T)$ and the corresponding reference free energy surface $F^{\text{liquid}}_{\text{ref 2}}(V,T)$ which is parameterized based on a liquid DFT molecular dynamics trajectory.

Step 1. The interatomic potential for the solid phase $V_{\text{ref 1}}$ can then be used in an coexistence approach [120] to determine an initial guess for the melting temperature T^{melt}. In the coexisting approach a simulation cell is constructed which contains both a solid and a liquid phase sharing a common interface. The simulation box is then equilibrated in a NPT ensemble to derive the melting temperature $T^{\text{melt}}_{\text{ref}}$ at zero pressure p. This is achieved by applying multiple strains to the interface structure.

Step 2. At the melting point the free energy of both the solid $F^{\text{solid}}_{\text{ref 1}}(V,T)$ and the liquid $F^{\text{liquid}}_{\text{ref 1}}(V,T)$ are the same, while the corresponding volume is different:

$$V^{\text{melt, liquid}}_{\text{ref 1}} \neq V^{\text{melt, solid}}_{\text{ref 1}} . \tag{2.82}$$

With the free energy of the liquid phase $F^{\text{liquid}}_{\text{ref 1}}(V,T)$ being calculated with the interatomic potential, parameterised with the solid phase $V_{\text{ref 1}}$: it is not the same as the free energy of the liquid phase $F^{\text{liquid}}_{\text{lref 2}}(V,T)$ calculated with the interatomic potential, parameterised with the liquid phase $V_{\text{ref 2}}$. So, the next step is to calculate the free energy of the liquid phase with the second reference potential $V_{\text{ref 2}}$:

$$\begin{aligned} F^{\text{liquid}}_{\text{ref 2}}(V^{\text{melt, liquid}}_{\text{ref 1}}, T^{\text{melt}}_{\text{ref 1}}) =& F^{\text{liquid}}_{\text{ref 1}}(V^{\text{melt, liquid}}_{\text{ref 1}}, T^{\text{melt}}_{\text{ref 1}}) \\ &+ \int_0^1 d\lambda \left\langle U^{\text{liquid}}_{\text{ref 2}} - U^{\text{liquid}}_{\text{ref 1}} \right\rangle . \end{aligned} \tag{2.83}$$

Step 3. By integrating over temperature and pressure, the full free energy surface $F^{\text{liquid}}_{\text{ref 2}}(V,T)$ for the interatomic potential, which is parameterised with the liquid phase $V_{\text{ref 2}}$, can be calculated:

$$F^{\text{liquid}}_{\text{ref 2}}(V, T^{\text{melt}}_{\text{ref 1}}) = F^{\text{liquid}}_{\text{ref 2}}(V^{\text{melt, liquid}}_{\text{ref 1}}, T^{\text{melt}}_{\text{ref 1}}) - \int_{V^{\text{melt, liquid}}_{\text{ref 1}}}^{V} P(V', T^{\text{melt}}_{\text{ref 1}}) dV' , \tag{2.84}$$

$$\frac{F^{\text{liquid}}_{\text{ref 2}}(V,T)}{k_B T} = \frac{F^{\text{liquid}}_{\text{ref 2}}(V, T^{\text{melt}}_{\text{ref 1}})}{k_B T} + \int_{T^{\text{melt}}_{\text{ref 1}}}^{T} d\left(\frac{1}{T'}\right) U(V,T') . \tag{2.85}$$

Both integrations use only interatomic potentials therefore the integrations can be executed efficiently with high precision.

Step 4. Based on the full free energy surface for the interatomic potential's the liquid phase $F^{\text{liquid}}_{\text{ref 2}}(V,T)$, which is parameterised with the liquid phase $V_{\text{ref 2}}$, thermodynamic integration can be used following the TU-TILD method to integrate from the reference potential to DFT with low convergence parameters:

$$F^{\text{liquid}}_{\text{DFT low}}(V,T) = F^{\text{liquid}}_{\text{ref 2}}(V,T) + \int_0^1 d\lambda \langle U^{\text{liquid}}_{\text{DFT low}} - U^{\text{liquid}}_{\text{ref 2}} \rangle_{\lambda,T} \,. \tag{2.86}$$

Step 5. In the last step the UP-TILD method is applied to up-sample the DFT precision from low convergence parameter to high convergence parameter:

$$F^{\text{liquid}}_{\text{DFT high}}(V,T) = F^{\text{liquid}}_{\text{DFT low}}(V,T) + \frac{1}{M}\sum_{u=1}^{M}(E^{\text{liquid}}_{\text{DFT, low}} - E^{\text{liquid}}_{\text{DFT, high}}) \,. \tag{2.87}$$

The TOR-TILD method has been successfully applied to calculate the melting temperature for metals with a **controllable error** within the method of up to ±5K. While the **intrinsic error** of comparing the exchange-correlation functionals LDA and PBE, introduced in Sec. 2.1.3, to experimental measurements is ±150K in contrast to the experimental precision of ±5K for comparing different experimental approaches [126]. This difference in complexity from calculating a single free energy to comparing two free energies, once more highlights that the current development of new methods in the field of *ab initio* thermodynamics is primarily limited by the technical complexity. Many different methods are available, but a systematic comparison is missing. On the one hand most current methods still require human expertise to execute them, which prohibits extensive parameter studies and on the other hand the combination or coupling of two existing methods is not easily possible as a joined interface is missing.

3 Technical Complexity

The theoretical overview in Chap. 2 highlighted three dimensions of complexity for calculating *ab initio* phase diagrams. They range from the complexity of calculating energies and forces from *ab initio* with minimal uncertainty, over the complexity of computing free energies to calculate finite temperature properties, to the chemical complexity. The resulting combination of complexities currently hinders the prototyping of new methods, the systematic comparison of existing methods and finally the development of coarse-grained models. This is demonstrated by the complexity of combining two phases, e.g the solid and liquid phase to calculate the melting temperature, as introduced in Sec. 2.4.4. Still based on the recent achievements of machine learning interatomic potentials, which automate the parametrisation of interatomic potentials, as introduced in Sec. 2.2.2, and the thermodynamic integration using langevin dynamics, which enables the efficient computation of free energies, as introduced in Sec. 2.4.2, the technical complexity is identified as the limiting factor. To evaluate existing solutions to address the technical complexity three prototypical challenges are defined, which are indicated in Fig. 1.1 by open symbols:

- For the complexity of the *ab initio* calculation with minimal uncertainty a DFT parameter study is selected to calculate the bulk modulus, from fitting an EOS, as introduced in Sec. 2.4.1, for a range of combinations of the convergence parameters of energy cut-off ϵ_i and kpoint mesh κ_j.
- For the thermodynamic complexity the calculation of the melting temperature with the coexistence method is selected, as introduced in Sec. 2.4.4. To reduce the other two dimensions, the melting temperature is calculated for interatomic potentials rather than DFT.
- For the chemical complexity a phase diagram is calculated with the quasi-harmonic approximation, again for an interatomic potential.

With these three challenges, the three dimensions of complexity, namely the *ab initio* accuracy, the thermodynamic complexity and the chemical complexity are addressed. At the same time the challenges are selected as prototypical application, each focused on one aspect of complexity. The aim is to identify a simulation framework which is capable to address all three of these challenges.

3.1 Software Implementation

The three challenges combine methods developed in different fields, from quantum chemistry over physics to engineering. So the development of a single simulation code which covers all three fields is prohibitive. Moreover the technical challenge of *ab initio* thermodynamics is the combination of multiple simulation codes developed in different communities. The differences range from different variable names, over different units to different formats for input and output files. So the coupling of two methods not only requires the theoretical understanding of both methods, but also practical experience and the technical expertise to develop an interface. The aim is to identify a software which enables the combination of existing methods like building blocks.

3.1.1 Integration Levels

The different simulation codes and software frameworks are classified based on their ability to support the user in rapid prototyping, up-scaling and sharing complex simulation protocols:

1. **Level:** The most fundamental feature of a simulation code is the functionality of an atomistic engine to compute energies and forces.
2. **Level:** The thermodynamics module takes the energies and forces of a given atomistic engine as input to compute thermodynamic properties.
3. **Level:** The simulation framework includes the required logic to construct feedback loops to dynamically adjust the simulation protocol based on previous results and also handles the calculation and data management.

Each of these levels increases the user comfort by reducing the technical complexity. While it is theoretically possible to compute an *ab initio* phase diagram using just the energies and forces computed from the atomistic engine, this would require the application-specific implementation of both the physical approximations to compute the thermodynamic properties and the technical implementation of the calculation and data management. As a consequence, such a development would be most likely specific to a given application and IT infrastructure. In contrast a simulation framework enables the separation of the atomistic engine, thermodynamics module and technical implementation, so a simulation protocol which is developed on one IT infrastructure can be executed on a different IT infrastructure ideally without any modification. The typical cases for switching the IT infrastructure are on the one hand the transition from rapid prototyping on a local workstation computer to up-scaling on an High-Performance-Computing (HPC) cluster and on the other hand sharing a simulation protocol with a collaborator for reasons of reproducibility and transferability.

3.1.2 Software for Atomistic Simulation

In the following section, a selection of existing simulation codes, thermodynamic modules and simulation frameworks is briefly introduced and classified based on the three levels defined above. This list is not complete but rather representative based on the software used in this thesis:

Vienna ab initio Simulation Package

The Vienna *ab initio* simulation package (VASP) [33, 127, 128] is one of the most popular plane wave DFT simulation codes, as introduced in Sec. 2.1. It is mainly known for its combination of computational efficiency in terms of supporting modern computing libraries and architectures and the user-friendly interface based on human read-able input files consisting mainly of key-value pairs. Beyond the calculation of energies and forces – level one – VASP also supports structure optimisations and molecular dynamics calculation as well as other more complex simulation protocols like nudget elastic band calculations – level two. Each of these can be controlled by specifying additional key-value pairs in the default input files. Still to implement new simulation protocols the user has to modify the VASP source code, which is written in the Fortran programming language. There is no possibility to implement feedback loops or other logical structures directly in the input files. So VASP can be classified between level one and level two, as it supports selected thermodynamic properties while others require the user to execute multiple independent calculations.

S/PHI/nX

To address the complexity of modifying the VASP source code, written in Fortran, the S/PHI/nX [129] DFT code was developed in C++ based on an object oriented implementation of the underlying quantum mechanical operators. This approach is especially suitable for method development. Typical examples include the development of new optimiser for atomistic structures [130] or the inclusion of charged defects in DFT [131–133]. In contrast to VASP, the S/PHI/nX DFT code does not implement molecular dynamics or nudget elastic band calculation, but rather an unix pipeline based interface to evaluate atomistic structures and return energies and forces. In addition, the S/PHI/nX DFT code is available as open-source in contrast to the commercial VASP DFT code. This not only enables the users to contribute to the S/PHI/nX code but also allows them to reuse the core functionality of the S/PHI/nX DFT code, the library of mathematical objects, to address other challenges like the development of an $k \cdot p$ model [134]. As a consequence, the S/PHI/nX DFT code provides two different interfaces for the users: They can either control the S/PHI/nX DFT code with a hierachical input file or develop their own simulation code based on the underlying libraries using the C++ programming language. Hence, the S/PHI/nX DFT code is another simulation code between the levels one and two. In addition, it demonstrates the advantage of open-source software development for scientific software by sharing the core library with multiple projects.

Large-scale Atomic/Molecular Massively Parallel Simulator

Moving from the DFT simulation codes to molecular dynamics with interatomic potentials, the Large-scale Atomic / Molecular Massively Parallel Simulator (LAMMPS) [135] is the most popular molecular dynamics simulation code for interatomic potentials used in computational materials science for solid-state simulations. Like the S/PHI/nX DFT code, LAMMPS is developed in the C++ programming language and released under an open source software licence. Still, the primary feature of the LAMMPS simulation code is the versatility of available simulation protocols in combination with its parallel computing performance. In addition to molecular dynamics with various ensembles, as introduced in Sec. 2.3.2, and structure optimisations with various minimizers the LAMMPS simulation code enables the development of complex simulation protocols by providing the necessary computing logic e.g. if-then statements, loops and the ability to load external files which include further instructions.

This functionality has been extended recently to support loading the LAMMPS simulation code in other programming languages like C++ or even Python as software library. Having access to the functionality of the LAMMPS simulation code from a scripting programming language like Python standardises the development process as it allows coupling LAMMPS to other python software libraries. So the LAMMPS simulation code covers the first two levels and even goes beyond the second level by providing a standard interface to develop simulation protocols either with the internal logic in the input files or by loading the simulation code as a software library in another programming language.

If the same version of the LAMMPS simulation code with the same software extensions is installed on two different IT infrastructures, then a simulation protocol developed on one infrastructure can also be transferred to another infrastructure without major modifications. The same applies to the input files of VASP and S/PHI/nX. Still, those simulation codes require an external software to implement feedback loops which can result in an IT infrastructure dependence.

Phonopy

Beyond the simulation codes a different approach for the second level is the phonopy [136] software toolkit to calculate phonons with the finite displacement method, as introduced in Sec. 2.4.1. Rather than including a specific first level atomistic engine in the phonopy software toolkit, the code can generate input files for 14 different mainly DFT simulation codes including VASP. These input files include the required atomistic structures with the corresponding displacements for a given crystal structure already taking the crystal symmetry into account. In addition to the atomistic structure, the user then has to add additional input files to specify parameters like the electronic convergence, kpoint mesh κ_j, energy cut-off ϵ_i and pseudopotential to execute the calculation. The required inputs for a DFT calculation are introduced in Sec. 2.1. After the successful calculation of the forces with the external quantum engine controlled by the user, phonopy can parse the output files of the supported simulation codes to convert the forces of all codes to the same units, compute the dynamical matrix and calculate the free energy. Phonopy is developed in the Python programming language and released under an open-source licence, which allows other projects to reuse components of phonopy e.g. the symmetry library to determine the required displacements.

Atomic Simulation Environment

The atomic simulation environment (ASE) [137, 138] extends the second level approach by providing python representations of the individual first level atomistic engines. In total, ASE supports more than 40 simulation codes. For each of these simulation codes ASE not only writes the required atomistic structure to the input file like phonopy but also takes the code-specific input from a set of predefined python variables to include them in the input files. The simulation code then only computes the energies and forces and thermodynamic approximations e.g. calculating a molecular dynamics trajectory, are all implemented as code independent python objects in the ASE. This approach is suitable for rapid prototyping of new simulation protocols, as the ASE python objects allow the user to define the simulation protocol in the Python programming language which can be executed on every IT infrastructure, which has the same ASE version and the required simulation codes installed. The limitation of this approach is the up-scaling of existing simulation protocols. On the one hand, ASE provides objects for data storage but these have to be configured manually and on the other hand, ASE does not provide any interface to job schedulers which are required for HPC-calculations. So, it is the task of the users to up-scale their simulation protocols by managing the individual calculation and their data. Still, with the ASE simulation framework being released under an open-source licence the components of the framework can be integrated in other frameworks. In particular, the atomistic structure class of the ASE simulation framework is used in many other second level thermodynamic toolkits like phonopy.

Ab initio Path Integral Molecular Aynamics

The ASE approach of writing input and output files is insufficient for strong coupling of existing simulation codes. This is required for thermodynamic integration, when the forces calculated with two atomistic engines have to be mixed at every timestep of the molecular dynamics trajectory, as introduced in Sec. 2.4.2. For the calculation of a single free energy this can result in several thousand files being written during the sampling of a single trajectory. Such a large number of file writes is prohibitive in particular for IT infrastructures with shared file systems, e.g. HPC cluster. To address this, ASE supports the use of Python compatible software interfaces for simulation codes which support those, e.g. LAMMPS, or the communication via process standard input and output, e.g. VASP. Still, the supported functionality of the different codes vary, as the LAMMPS software library supports the full functionality of the LAMMPS simulation code while the VASP interface is restricted to modifying the atomistic structure and computing energies and forces.

In contrast the *ab initio* path integral molecular dynamics code (ipi-code) [139, 140] modifies the source code of existing simulation codes to add a socket-based interface for coupling them. Currently a total of eleven simulation codes is supported including VASP and LAMMPS. The approach is promising as socket-based communication is faster than file-based communication and reduces the load on the file system, while at the same time introducing an open-source standard other simulation codes can follow. With this, the ipi-code combines the first and second level. Nevertheless, given the technical complexity implementing a new simulation protocol currently requires the modification of the Python source code of the ipi-code open-source toolkit. The user interface is limited to a hierarchical key-value-based input file. In addition, the communication via sockets is commonly prohibited on HPC clusters, which enforce the use of the message passing interface (MPI). As a consequence, the transferability and reproducibility of this solution is limited as it requires special permissions from the HPC cluster administration.

Materials Project

Besides the rapid prototyping of new simulation protocols as it is supported by the ASE simulation framework, the up-scaling of existing simulation protocols is a second challenge. With the available computing power of modern HPC clusters high-throughput DFT calculations are enabled. The impact of data-driven science as fourth pillar [141] next to experiment, theory and simulation is discussed in the next section. Still, some high-throughput projects also release their software environment as open-source software. The software stack of the materials project [142, 143] is such an example. It covers the second and third level, is written in the Python programming language and available as open source. The software stack consists of:

pymatgen An extensive parser originally developed for the VASP simulation code input and output files, which has more recently been extended to other simulation codes, as an interface to level one [144].

atomate A set of predefined simulation protocols for common materials properties like fitting an equation of state, calculating the elastic tensor or electronic structure calculations like calculating the electronic bandstructure. This is their implementation of the second level [145].

fireworks The underlying job and data management – the third level – which handles the internal logic like restarting a failed calculation with adjusted parameters as it is required for high throughput calculation. This is implemented by submitting a worker to the HPC job scheduler and when computing resources are available the worker executes multiple similar calculations in one allocation of the HPC job scheduler [146].

In direct comparison to ASE the materials project focuses on the up-scaling of comparably simple simulation protocols to iterate these over the periodic table, while ASE primarily focuses on the rapid prototyping of new simulation protocols. In practise, the simulation

	ASE	Materials Project	AiiDA	pyiron (new)
Area of Application	Interactive Development	High-throughput calculation	High-throughput calculation	Interactive Development, Up-Scaling
User Interface	Python, Jupyter	Python, Command Line Interface	Python, Command Line Interface	Jupyter as primary Interface
Code Interface	Python, File-based	File-based	File-based	Python, Sockets, File-based
Data Management	User-defined SQLite table	MongoDB (JSON)	SQL table	SQL table HDF5 File
Job Management		Worker-based	Heterogeneous Job Schedulers	Heterogeneous Job Schedulers
Package Distribution	conda, only a few simulation codes included	conda, simulation codes not included	conda, only core package, no plugins	conda including simulation codes

Table 3.1: Comparison of the simulation frameworks ASE, Materials Project and AiiDA with the newly developed pyiron framework, which combines both the interactive development of simulation protocols with the up-scaling towards high-throughput calculations.

protocol can be developed with ASE and has to be rewritten afterwards to up-scale it for high-throughput calculations in an HPC environment.

AiiDA

A second high-throughput project which published their software environment as open-source is the Materials cloud [147] with their software framework AiiDA [148–150]. In comparison to the materials project, the AiiDA software framework is focused on maintaining the provenance of each calculation. Like a tree structure the provenance stores the dependence of individual calculation in a database. With this database it is possible to relate existing calculations to each other. This data-centric approach goes beyond just storing a single number as result of a calculation rather it maintains the whole history which lead to a given result including all approximations used and all required individual calculations. For the user of such a high-throughput database, this does not only simplify the comparison to their own results but it also helps to identify other similar calculations which could be used for comparison. From the *ab initio* thermodynamics perspective, AiiDA in analogy to the materials project covers the levels two and three. Still, the creation of the provenance requires additional implementation effort on the developer side, which hinders the rapid prototyping. As a consequence, most simulation protocols are primarily developed by experts in the field who agreed on a given standard how to calculate a specific materials property.

3.1.3 Comparison

All three levels defined in the previous Sec. 3.1.1 are covered by the existing simulation frameworks in the Sec. 3.1.2 above. With a major separation between the first and the other two levels, as most simulation frameworks provide parsers for existing level one simulation codes rather than re-implementing the underlying atomistic engines. This separation is also visible in terms of programming languages, the simulation codes for level one are primarily written in Fortran or C++ with a particular focus on parallelisation to achieve the optimal performance on a HPC cluster, while the thermodynamic modules of level two and simulation frameworks of level three which orchestrate the individual calculation are primarily developed in Python. Still, the combination of rapid-prototyping and up-scaling remains challenging. Those frameworks which support the user in rapid-prototyping like ASE lack the ability to up-scale the simulation protocols for parameter studies on HPC clusters and the high-throughput frameworks commonly require additional effort in the development of simulation protocols. In addition, the high-throughput simulation frameworks and their simulation protocols are optimised for a given HPC cluster to achieve the optimal performance which reduces the transferability and reproducibility. In summary, this results in two independent developments, rapid prototyping with one framework and afterwards up-scaling the resulting simulation protocol with another framework. A more technical overview of the three simulation frameworks ASE, Materials Project and AiiDA is summarized in Tab. 3.1.

3.1.4 Software for CALPHAD Modelling

Besides these developments on the atomistic scale, also the CALPHAD community introduced in section 2.4.3 is moving towards open source and Python-based software frameworks. These developments originated in pycalphad [105] which implements the CALPHAD method, provides plotting routines as well as direct access to existing CALPHAD databases. The pycalphad approach is extended by the Extensible Self-optimizing Phase Equilibria Infrastructure (ESPEI) which is primarily used to parameterise CALPHAD databases based on experimental data sets and Phase Diagram Uncertainty Quantification (PDUQ). This confirms that the development of open-source Python software frameworks is not limited to the atomistic community. In addition these tools provide an interface for *ab initio* thermodynamics software frameworks to provide their results to a larger audience. As a consequence, a simulation framework for *ab initio* thermodynamics should not only bridge the gap between rapid prototyping and up-scaling, but also work towards an integration with the CALPHAD community.

3.2 Towards Data-Driven Science

With the constant increase in computing power, high-throughput DFT studies are now possible and data-driven science has been established as the fourth pillar next to experiment, theory and simulation [141]. This is based on the experience that up-scaling from a few hundred calculations to several thousand calculations requires a different approach in terms of managing the calculation and analysing them. Once more this increases the technical complexity. At the same time these high-through put studies are capable of identifying trends which are not accessible in smaller data sets. The systematic study of hydrogen diffusion in fourth row elements is just one example [151]. In the following selected high-throughput projects in atomistic community are briefly introduced, followed by benchmarking projects which use high-throughput approaches to validate DFT simulation codes and interatomic potentials. Both result in the challenge of transferability and reproduciblity and demonstrate what can be learned from data-driven studies.

3.2.1 High-Throughput DFT

Based on the precision and accuracy of DFT simulations in comparison to other methods at a comparable small computational cost, DFT is commonly not only used by theoretical groups, but also experimentalists to compare their measurements. Still, handling the convergence of a DFT calculation requires human expertise as explained in Sec. 2.1.4 and using the default convergence parameters is commonly insufficient. This led to the first high-throughput DFT studies: the virtual materials design framework (VMDF) [152], the automatic flow (AFLOW) [153] framework, the materials project [142, 143], the automated interactive infrastructure and database (AiiDA) [148–150] for computational science as part of the materials cloud [147] and many more [60, 154–158]. Many of these high-throughput projects provide extensive web interfaces which allow non-experts to browse the available data. On the one hand these databases addressed the need of experimentalists to get access to well-converged existing data sets and on the other hand the collected data allows studying trends over the periodic table. This led to the idea of not only storing individual material properties calculated from a given set of calculation but moreover creating a repository to potentially store all calculations, as calculation which are unreasonable for one person might contribute to a high-throughput study of another person.

The Nomad repository [159] offers this service to the community and provides findable, accessible, interoperable and reproducible data (FAIR-data) to the community [141]. Users can share their data on the Nomad repository and the Nomad project even converts the output files from a code-specific format to a generic simulation code independent format to improve the interoperability which enables the comparison beyond individual simulation codes. However, in terms of *ab initio* thermodynamics at the current stage none of the existing databases include thermodynamic calculations beyond the harmonic approximation. So the creation of a database of *ab initio* phase diagrams with the corresponding free energies for each phase and temperature could be a valuable extension once the automated calculation of *ab initio* phase diagrams is possible.

3.2.2 Validation Projects

Besides iterating over the periodic table to extend the general understanding of the chemical trends another approach that evolved from the data-driven science perspective is the systematic validation of existing DFT simulation codes and interatomic potentials.

Delta Project

The Delta project [34] is one of these projects. It compares over 15 different DFT simulation codes by calculating an energy volume curve, like it is illustrated in Fig. 2.4, for 71 elemental crystals. Based on seven discrete strains between 94% and 106%, the delta value between two simulation codes a and b is calculated as:

$$\Delta_i(a,b) = \sqrt{\frac{\int_{0.94V_0}^{1.06V_0} (E_{b,i}(V) - E_{a,i}(V))dV}{0.12V_{0,i}}}\,. \tag{3.1}$$

With $E_{a,i}(V)$ as the Birch Murnaghan fit, introduced in Sec. 2.4.1, of the energy volume curve for simulation code a and element i and $E_{b,i}(V)$ analogous. The equilibrium volume V_0 used to equally distribute the strains is included in the formula for normalisation. For the most recent pseudo-potentials the delta project finds good agreement between DFT simulation codes like VASP and all-electron calculations. This demonstrates that the implementation of the VASP DFT simulation code is sufficient to achieve the theoretically predicted precision of DFT and the accuracy being limited to the unknown exchange correlation functional, as introduced in Sec. 2.1.3.

The convergence parameters in terms of energy cutoff and kpoint mesh used in the Delta project are higher than the recommendations in the VASP manual and still differ from the convergence parameters used in the materials project. As even high-throughput DFT projects, which are focused on high precision DFT calculations, disagree in their choice of convergence parameters, this raises the question if it is possible to systematically determine the required convergence parameters for a given convergence goal. This challenge is addressed in more detail in Chap. 5. At the same time the Delta project is authored by a total of 70 authors who contributed to the comparison which is another indicator for the need of human expertise to achieve the optimal convergence with a given DFT simulation code as well as for the presumably manual work involved in this project. Still, it is an remarkable example how the community can benefit if the up-scaling of existing simulation protocols is simplified.

OpenKIM and NIST Repository

The second application is the validation of interatomic potentials to systematically compare the different types of interatomic potentials introduced in Sec. 2.2. While many interatomic potentials are developed, tested and published for a specific application, the community commonly applies them beyond the application the author intended assuming the interatomic potential is transferable. In order to prevent this and to compare the existing interatomic potentials the two major databases for interatomic potentials for solid state simulations the NIST repository [59] and the OpenKIM community [58] started to systematically test all interatomic potentials in their databases. This resulted in the development of atomman [60] for the NIST repository and the OpenKIM processing pipeline [160]. Rather than iterating over the periodic table, both frameworks iterate over all interatomic potentials in a given database and compare the same material properties for potentials which include the same chemical elements. The material properties compared range from equilibrium properties, e.g. the bulk modulus, which were included in the fitting of most interatomic potentials, to more complex properties e.g. comparing the stability of dislocation and grainboundary structures.

In analogy to the Delta project these projects benefit from the recent progress towards the up-scaling of simulation protocols and while all three projects developed their own software frameworks for their specific application this again highlights the general need. A framework which supports both the rapid prototyping, the up-scaling and the reproduciblity of simulation protocols would not only support the data-driven approaches in computational materials science but also the validation of existing approaches. In particular, the ability to validate an interatomic potential with a standardized test suite before publication or to test a new DFT simulation code for consistency with other DFT simulation codes would simplify the corresponding developments. As a consequence, the transferability and reproducibility of such a test suite is another essential aspect. Especially given the need of *ab initio* thermodynamics to combine both, DFT calculation and interatomic potential calculation, a simulation framework which covers both would enable a systematic assessment of existing interatomic potentials in direct comparison to DFT calculations by executing the same simulation protocol for both.

3.2.3 Transferability and Reproducibility

By comparing the high-throughput DFT studies to the validation projects and existing parameter studies the similarity in their software frameworks becomes clear, even though they have different goals. The high-throughput projects and parameter studies are focused on calculating specific materials properties once for a wide range of different parameter combinations and their results are published in their database to be accessible to a large audience. In contrast, the detailed calculation results for the validation projects are less relevant. They are primarily focused on the comparison of applying the same simulation protocols on different simulation codes or different interatomic potentials to validate their functionality. As a consequence, it is less reasonable to store all calculation results as they can be regenerated at any time. So the primary focus is on storing the simulation protocol rather than the result of the simulation protocol.

This approach is motivated based on the continuously increasing computing power available. Still, it requires the simulation protocols to be developed in an IT infrastructure independent way by separating the technical complexity from the scientific steps of the simulation protocol. At the same time this also enables sharing the simulation protocol with collaborators. While for individual simulation codes it is sufficient to share the input files and the required version of the simulation code for an simulation framework with multiple simulation codes used in one simulation protocol the complexity of tracking all dependencies of the software environment increases. At the current point none of the existing simulation frameworks support the user in sharing their simulation protocols. Nevertheless, extending the simulation framework to include the reproducibility in addition to the rapid prototyping and up-scaling of the simulation protocol, is the next step to extend the scientific workflow.

3.3 Scientific Computing

To address the aspects of rapid prototyping, up-scaling and sharing of simulation protocols, recent developments beyond the computational materials science community are analysed to identify suitable technologies. These technologies cover interactive computing environments to document the simulation protocol, the software management to maintain reproducible and transferable software environment and advanced data storage solutions to efficiently store large numerical arrays of atomistic trajectories.

3.3.1 Interactive Computing Environments

The Python programming language [161, 162] gained popularity in the scientific community including materials science by emphasising code readability and supporting multiple programming paradigms to enable the programmatic development of simulation protocols. Today, the majority of simulation frameworks for atomistic simulations are developed in Python as highlighted in Sec. 3.1.2. These simulation frameworks are commonly based on the scientific library scipy [163], the numerical library numpy [164] and the plotting library matplotlib [165] which provide high-level interfaces to underlying mathematical C++ and Fortran software libraries required by scientists. In the same way the simulation frameworks themselves can be identified as high-level interfaces to the underlying simulation codes.

Following the same analogy, the jupyter notebooks are the next generation development interface superseding the development of shell scripts on the command line. They combine an interactive computing interface which can be used with the python programming language and originated from the interactive python project [166]. The interactive interface supports tab completion to automatically fill partly typed commands, the addition of markdown text and LaTeX formulas for documentation, and inline visualisation. So in contrast to a python script with text comments, the documentation of a simulation protocol in a Jupyter notebook can be nicely formatted and the original output is included in the document. Like the python programming language, Jupyter notebooks are already wide spread in the data science community with all the numerical python libraries being integrated. In addition also the bioinformatics community adopted the use of Jupyter notebooks on a large scale. This is beneficial for atomistic simulation in materials science, as tools like NGLView [167], to visualise atomistic structures directly in the Jupyter notebook, are already available.

By addressing both, the rapid prototyping and the reproducibility, Jupyter notebooks are suitable to cover a full simulation protocol in a single document. Starting from the creation of the atomistic structure over the execution of the simulation code to the plotting of the simulation results all steps can be represented in a single Jupyter notebook to include it as supplementary material for a publication. This enables the reader to recreate the same results given that he or she has access to the same software environment. Finally, Jupyter notebooks can also be edited inside Jupyter lab which is an interactive environment running inside the users web browser, which allows combining multiple Jupyter notebooks inside one interactive session. At the same time by accessing the service via a web browser, it is possible to analyse the data where it was generated rather than transferring it. For the case of a simulation protocol being executed on an HPC cluster the data transfer is reduced to just the program code and visualisation in the Jupyter notebook, which also reduces the data segmentation.

3.3.2 Software Management

To further address the aspects of reproducibility and transferability of simulation protocols the scientific python community developed the conda package manager. In contrast to the built-in pip package manager which provides the source files of a given python software packages and compiles the source code locally the conda package manager provides pre-compiled binaries. This not only accelerates the installation but also addresses the issue of linking to C, C++ or Fortran software libraries as these can be distributed as separate conda packages rather than requiring the user to install these separately.

While the conda package manager was originally developed to distribute the numerical python library numpy it has been adopted by a growing number of scientific communities. Most notably the bioinformatics community [168, 169] released over 8000 software packages specific to bioinfromatics, which makes the conda package manager the default choice to distribute bioinformatics software. In addition the general community channel named conda-forge already includes over 14500 software packages for scientific application and beyond. As a consequence most of the dependencies of the software packages specific to materials science are already available. Once a software package is available on the conda package manager, it can be installed with a single command:

```
conda install -c conda-forge numpy
```

This downloads the package `numpy` from the conda community channel and installs it in the current conda software environment. The conda software environment is a self-contained installation. On the one hand this enables the user to have multiple environments to compare different software versions and on the other hand an environment can be exported as a single environment file `environment.yml` which lists all packages with their corresponding versions:

```
conda env export > environment.yml
```

In combination with the Jupyter notebooks this enables the distribution of the whole scientific workflow including all software dependencies encapsulated in a single package which increases the reproduciblity and transferablity of a given scientific workflow. Still, an optimised compilation of a simulation code on a given IT infrastructure can accelerate the execution of the same calculation by a factor of two to three. As a consequence, the use of conda-forge in a HPC environment is debatable. It is recommended to compile frequently used simulation codes manually for optimal performance. So the conda packages should be treated as reference implementation for testing and validation.

3.3.3 Data Storage Solutions

While simulation codes write the data for all simulation parameters on a per time step basis the analysis at the end of the simulation is based on the change of individual properties over time. In addition when working with multiple simulation codes the conversion to a shared generic format in terms of units and variable names is reasonable for comparison. As a consequence, transforming the simulation code output at the end of the simulation simplifies the following analysis. The parsing of the simulation output at the end of the calculation rather than during the analysis is already implemented in the AiiDA simulation framework and the software environment of the materials project. The AiiDA simulation framework uses an SQL database for storing the provenance of the simulation input and output to represent the relationships between the different parameters and the Materials Project uses a JSON based NO-SQL databases for less structured simulation properties. Still both approaches are inefficient for storing molecular dynamic trajectories for *ab initio* thermodynamics as they are optimised to keep all data in memory rather than storing the data on disk, which is prohibitive for large molecular dynamics trajectories.

To address this limitation the data-driven science community developed file-based databases to handle up to several petabyte of data [170] and specific data formats for scientific applications like the Hierarchical Data Format 5 (HDF5) [171]. In contrast to traditional file formats, the HDF5 format prepends an index of the files content at the beginning of the file. With this index reading a specific datasets from the file with multiple datasets does not require reading the whole file, but rather just the individual blocks for the specific dataset. In addition the datasets in the file can be grouped in groups, to create arbitrary levels of hierarchy, hence the name Hierarchical Data Format. In combination with the corresponding Python library h5py compressed data sets are directly saved from memory to the file and reloaded as compressed data sets from the file directly to memory, which is faster than uncompressing the data first and loading it to memory afterwards and directly supports the access from the numerical library numpy [164]. Based on these features, the HDF5 format has already been adopted in the computational materials science community with the introduction of the h5md [172] format. It is a specification for the HDF5 format to store molecular dynamics trajectories, which is supported by a growing number of molecular dynamics simulation codes like LAMMPS.

In contrast to only storing selected properties in the database like current high-throughput frameworks do or alternatively parsing the output file during every iteration like it is done by interactive frameworks like ASE, a HDF5 based generic file format allows parsing the code-specific output files only once at the end of the simulation and storing the output already with the transformed units and variable names. As a consequence, HDF5 is suitable for both, the rapid prototyping and up-scaling simulation protocols. For rapid prototyping the whole output of the simulation code is parsed at the end of the calculation providing the user with the flexibility to compare different output properties to identify correlations, without the need to explicitly define the properties to store before the simulation. While for the up-scaling the user benefits from the index of the HDF5 file, as the data aggregation is not required to load all output properties but rather only selected properties are loaded to accelerate the post-processing. Still, the whole simulation output is available for debugging when required.

4 pyiron – Integrated Development Environment

Based on the physical complexity of *ab initio* thermodynamics, introduced in Chap. 2 and the resulting technical complexity, introduced in Chap. 3, the requirements for a new third level simulation framework are defined in the following. The simulation framework enables the combination of level one atomic engines and level two thermodynamics modules like building blocks to address the whole process of developing a simulation protocol. Starting from the rapid prototyping, over the up-scaling for parameter studies and the publication of the simulation protocol in a reproducible format. The whole process is handled in the same framework.

Leveraging the latest software developments in the general scientific community, the interactive Jupyter environment is identified as a digital format to represent the whole simulation protocols in one document: namely the creation of an atomistic structure, the analysis of the atomistic structures with structure descriptors, the calculation of energies and forces with an atomistic engine, the propagation of this structure in a thermodynamics module, the aggregation of calculation results and finally the creation of figures and tables for publication with the corresponding documentation. The Jupyter notebook can then be shared as supplementary material of the scientific publication in combination with the underlying data stored in the HDF5 format and the software dependencies tracked in a conda environment file. This combination not only enables the readers to reproduce the results but it also reduces the barrier to apply newly developed methods to other material systems, hence fostering the collaboration in the scientific community. Given this overview, the requirements for the new simulation framework are summarized as:

- The ability to scale a simulation protocol from rapid prototyping up to high throughput parameter studies. This includes switching from interactive execution on the local workstation computer to submitting to a HPC cluster by accessing the job scheduler and requires an efficient data format for storing the calculation results like HDF5.
- Defining building blocks based on a standardized generic format, which allows switching from one simulation code to another one with minimal effort, even if these simulation codes are developed in different communities, use different input/output format, variable names and units. This enables using a simulation code with lower accuracy and more affordable computational costs during the interactive development and switching to the more accurate simulation code only once the development of the simulation protocol is finalized.

- Using the python programming language for coupling simulation codes and leverage the Jupyter environment to simplify the interactive development process of complex simulation protocols. This includes reducing the number of software modules the user has to import, using tab completion to reduce the number of commands the user has to remember and render objects for visual representation inside the Jupyter environment.
- Applying a combination of a relational SQL database for structured data and the HDF5 for unstructured data like the charge density of a DFT calculation or the positions of a molecular dynamics trajectory.
- Leveraging the functionality of existing software developments, this includes functionality which is implemented in the simulation codes as well as functionality of existing thermodynamic modules to accelerate the development process. Use the conda package manager to track the environment with all dependencies and allow the user to publish the dependencies as part of the supplementary material to guarantee the reproducibility and transferability of the simulation protocol.

Given these requirements, previous developments in the department for computational materials science at the "Max-Planck-Institut für Eisenforschung" are evaluated. Following the first release of the S/PHI/nX DFT simulation code, introduced in Sec. 3.1.2, the department has started to develop multiple prototypes of a simulation frameworks already in 2008, namely PHInaX [173] and PyCMW [174] to simplify the construction of complex simulation protocols. PHInaX combines various utilities developed as part of the S/PHI/nX DFT code with a central database to store the relation between the individual calculation results. This enables the reduction of duplicate calculations. The limitations of the PHInaX approach are the development in the C++ programming language and the abstract data storage based on numerical database identifiers. These hinder the rapid prototyping and limit the debugging capabilities.

The Python computational materials science workbench (PyCMW) is developed in Python and uses the Python programming language for constructing simulation protocols. Instead of a central database a local database on the users workstations computer is used and calculation results are transferred from the users workstation computer to the HPC cluster and back. The local data storage is the central connection between the data creation, the data analysis and the data visualisation. This enables the user to visually inspect the results of their calculations via the PyCMW graphical user interface (GUI). The GUI in combination with developing simulation protocols in Python increases the flexibility and having direct access to the simulation data improves the debugging capabilities in case a calculation is not executed successfully. Finally by using the HDF5 file format in addition to a SQL database as central index of all simulations both structured and unstructured data can be stored efficiently. The limitations of this approach are the requirement to write the complete simulation protocol before being able to test it instead of step by step interactive execution, the transfer of each calculation to the HPC cluster, which limits the up-scaling of simulation protocols and finally the GUI as primary user interface which does not satisfy the requirements of expert users.

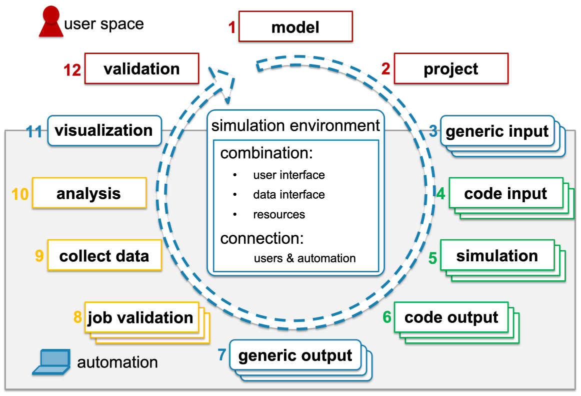


Figure 4.1: The simulation life cycle illustrates the 12 steps from developing a model, executing the simulation, analysing the data and finally validating the model. While the steps 1.–2. and 12. (red boxes) require user input the steps 3.–11. can be automated by a simulation framework (grey box). The simulation frameworks (blue boxes) interacts with the simulation code (green boxes) and the data analysis module (yellow boxes). Finally, the layered boxes for steps 3.–8. indicate the parallel execution of multiple calculations, while the remaining steps 1.–2. and 9.–12. can include input and output of multiple calculations. Illustration based on Fig. 1 in the pyiron release paper [175].

Based on these previous prototypes the current framework – the pyiron, an integrated development environment (IDE) for *ab initio* thermodynamics [175], is developed starting in 2015. The name combining the programming language Python the software is developed in and iron as reference to the “Max-Planck-Institut für Eisenforschung” (iron research) where the development originated. Junior scientists and experts alike contribute to the development of the pyiron IDE, either with their feedback or by extending the functionality. Especially with the pyiron IDE being released as open-source softwareand the pyiron project joining the NumFocus foundation as an affiliated project to promote open practices in research and scientific computing, the world wide pyiron community grows continuously. Still, for the purpose of this thesis the reproducible simulation protocols, the high-throughput computing capabilities and the rapid prototyping of simulation protocols are highlighted as primary features required for *ab initio* thermodynamics.

4.1 Simulation Life Cycle

Given the complexity of the simulation protocols in *ab initio* thermodynamics, introduced in Sec. 2.4 a primary focus of the pyiron IDE is to structure the development of complex simulation protocols. The simulation protocols should be reproducible, easy to up-scale for parameter studies and finally the development of the simulation protocols should be as simple as possible to enable rapid prototyping to address the complexity of *ab initio* thermodynamics. To illustrate this iterative approach, the simulation life cycle is introduced, which describes the process from defining a model, parameterising the simulation, validating the results and updating the model. Starting with one of the most fundamental simulation protocols in *ab initio* thermodynamics, the calculation of an energy volume curve to determine the equilibrium properties: equilibrium Volume V_0, equilibrium bulk modulus B_0, derivative of the equilibrium bulk modulus with respect to the pressure B' and the equilibrium energy E_0. The energy volume curve, introduced in Sec. 2.4.1, typically contains seven energies calculated for seven volumes equally spread around the approximated equilibrium volume, by a uniform deformation of the lattice. The energy volume pairs are fitted with one of the equations of state, introduced in Sec. 2.4.1 to directly derive the equilibrium parameters as results of the fit. The simulation life cycle for the calculation of the equilibrium parameters is illustrated in Fig. 4.1 and it can be structured in the following twelve steps, which can be executed in multiple iterations to achieve self-consistency.

Step 1. The initial step is the definition of the model. In terms of calculating the energy volume curve, this is choosing an equation of state, including an initial guess of the equilibrium volume, specifying the spread of the volume points as well as the parameters for the formalism of calculating the energy. In the case of a DFT calculation, this would be specifying the energy cutoff, kpoint mesh and other convergence parameters. Commonly this initial step is less formalized and the reasoning for a specific choice of parameters is not documented.

Step 2. Given the requirements of the model, a project is defined. The project contains the definition of the individual calculations. For the energy volume curve these are calculations with equal convergence parameters at different volumes.

Step 3. The parameter definition in the project is typically done in a generic format, so the parameters are independent and the choice of the simulation code is just another parameter. This allows iterating over different simulation codes, e.g. the delta project [34]. For the calculation of the energy volume curve the steps 3.-9. have to be executed for each volume point individually.

Step 4. The generic input for each volume point is then converted in a code-specific input. This includes converting the generic variable names and units to the code-specific variable names and units as well as communicating the parameters to the simulation code. In the simplest case this is done by writing an input file which is then read from the simulation code.

Step 5. The simulation code is executed using the code-specific input from the previous step. Depending on the complexity of the calculation it might be necessary to execute the steps 4.–8. on a separate computing system. In that case the specification of the computing resource are additional generic input parameters.

Step 6. After the execution of the simulation code ended, the simulation results are transferred from the simulation code to the simulation framework. In terms of file based communication this includes parsing the output files to have the simulation results available for further processing.

Step 7. In analogy to step 3. the results of the simulation code are then converted from the code-specific format to the generic format. As a consequence, the following steps can be defined independently of the simulation code, which allows the same analysis methods to be applied to different simulation codes.

Step 8. As a first step of the analysis the output of the individual calculation is analysed separately. For a DFT simulation code this analysis of the individual calculation includes confirming that the electronic convergence reached the required precision.

Step 9. The individual calculations are collected and the combined data is aggregated for further analysis. In terms of the energy-volume curve calculation, this aggregated data is an array of volumes in combination with a corresponding array of converged energies for these volumes.

Step 10. The aggregated data is then post processed in the analysis step. For the energy-volume curve calculation this is fitting the selected equation of state to derive the equilibrium parameters for the material system in dependence to the other parameters defined, as the energy cutoff, kpoint mesh, exchange correlation functional and pseudopotential.

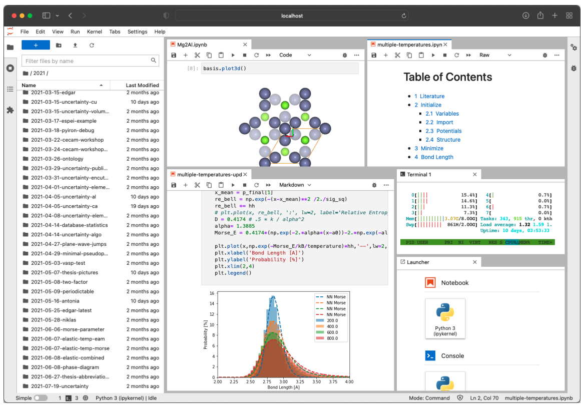
Step 11. To gain a qualitative understanding the data can be visualised for the user. In terms of the energy-volume curve a typical visualisation is plotting the equilibrium volume and equilibrium energy in combination with the other energy volume pairs to validate the selected energy volume pairs are well balanced around the equilibrium volume, as it is shown in Fig. 2.4.

Step 12. Finally, it is the task of the user to validate the results and possibly update the underlying model to trigger another iteration of the simulation life cycle. For the energy-volume curve this could be adjusting the initial guess, alternatively adjusting the convergence parameters energy cutoff and kpoint mesh to analyse the dependence of the equilibrium parameters on the convergence parameters.

While the steps 1.–2. and 12. require user input, the steps 3.–11. can be automated by a simulation framework. In particular the steps 4.–6. can be identified as the communication with the simulation code and the steps 8.–10. as the analysis of the simulation results. Finally the steps 3.–8. have to be executed for each calculation separately while the remaining steps can include input and output of multiple calculations.

4.2 Concepts

Based on the simulation life cycle in Fig. 4.1 the pyiron IDE is developed to support the user during all steps of it. In the following the fundamental concepts of the pyiron IDE are introduced, starting with comparing it to IDEs for software development in Sec. 4.2.1, followed by the pyiron objects, the abstract class of objects which defines the building blocks for the simulation protocols in Sec. 4.2.2 and finally how the pyiron IDE supports analysing high-throughput parameter studies with an MapReduce [6] based approach in Sec. 4.2.3.



Name	Last Modified
2021-03-15-edgar	2 months ago
2021-03-15-uncertainty-cu	10 days ago
2021-03-15-uncertainty-volum...	2 months ago
2021-03-17-espei-example	2 months ago
2021-03-18-pyiron-debug	2 months ago
2021-03-22-cecam-workshop	2 months ago
2021-03-24-cecam-workshop...	2 months ago
2021-03-26-ontology	2 months ago
2021-03-29-uncertainty-publi...	2 months ago
2021-03-31-uncertainty-encut...	2 months ago
2021-04-01-uncertainty-eleme...	2 months ago
2021-04-05-uncertainty-al	10 days ago
2021-04-05-uncertainty-ca	19 days ago
2021-04-08-uncertainty-elem...	2 months ago
2021-04-14-database-statistics	2 months ago
2021-04-14-uncertainty-algo	2 months ago
2021-04-27-plane-wave-jumps	2 months ago
2021-04-29-minimal-pseudop...	2 months ago
2021-05-03-vasp-test	2 months ago
2021-05-07-thesis-pictures	10 days ago
2021-05-08-two-factor	2 months ago
2021-05-09-periodictable	2 months ago
2021-05-16-antonia	10 days ago
2021-05-25-edgar-latest	2 months ago
2021-05-28-niklas	2 months ago
2021-06-06-morse-parameter	2 months ago
2021-06-07-elastic-temp-eam	2 months ago
2021-06-07-elastic-temp-morse	2 months ago
2021-06-08-elastic-combined	2 months ago
2021-06-08-phase-diagram	2 months ago
2021-06-27-thesis-abbreviatio...	2 months ago
2021-07-19-uncertainty	2 months ago

```
[8]: basis.plot3d()
```

```
x_mean = p_final[1]
re_bell = np.exp(-(x-x_mean)**2 /2./sig_sq)
re_bell *= hh
# plt.plot(x, re_bell, ':', lw=2, label='Relative Entrop
D = 0.4174 # .5 * k / alpha^2
alpha= 1.3885
Morse_E = 0.4174*(np.exp(-2.*alpha*(x-a0))-2.*np.exp(-al

plt.plot(x,np.exp(-Morse_E/kB/temperature)*hh,'--',lw=2,
plt.xlabel('Bond Length [A]')
plt.ylabel('Probability [%]')
plt.xlim(2,4)
plt.legend()
```

Table of Contents

- 1 Literature
- 2 Initialize
  - 2.1 Variables
  - 2.2 Import
  - 2.3 Potentials
  - 2.4 Structure
- 3 Minimize
- 4 Bond Length

```
  0[||||          15.4%]   4[|            0.7%]
  1[||||          14.6%]   5[             0.0%]
  2[|||           11.3%]   6[|            0.7%]
  3[||             7.3%]   7[             0.0%]
Mem[|||||||||||3.07G/8.00G] Tasks: 343, 915 thr, 0 kth
Swp[|||||||||   861M/2.00G] Load average: 1.32 1.59 1.
                            Uptime: 10 days, 03:53:33

 PID USER      PRI  NI  VIRT   RES S CPU%▵MEM%   TIME+
```

Figure 4.2: Screenshot of the pyiron IDE running inside Jupyterlab with multiple Jupyter notebooks being executed in a single session. The left bar shows a list of projects created by the user. It acts as a file browser and helps to inspect the project structure visually. The top center figure shows the interactive visualisation of an hcp Mg2Al structure using the NGLview library [167]. In the top right a table of content of a Jupyter notebook is listed. The lower center part shows the analysis of a finite temperature bond distributions, plotted with matplotlib [165]. The center right window shows a terminal inside the Jupyterlab environment and finally the lower right picture shows the application launcher used to open additional applications within the session windows. The whole Jupyterlab environment is accessible directly via the web browser. This applies to both installations of the pyiron IDE on a local workstation computer or on an HPC cluster.

4.2.1 Integrated Development Environment

As an IDE for *ab initio* thermodynamics the pyiron IDE combines a web-based code editor in the form of Jupyter notebooks with an extensive job management for build automation and up-scaling simulation protocols from rapid prototyping to high-throughput parameter studies as well as extensive debugging capabilities. These three components, the code editor, the build automation and the debugger define an IDE as they are commonly used for software development. The same components are required for the development of complex simulation protocols, even though the individual components are different.

Source Code Editor: Jupyter notebooks allow the combination of software source code, comments in rendered text as well as the inclusion of graphical content. With this the Jupyter notebooks are similar to lab books used by scientists to document their process. The functionality of Jupyter notebooks is briefly summarized in Sec. 3.3.1. In contrast the source code editors of IDEs in the field of software development focus only on the source code and in-line comments.

Build Automation: While software developers use the build automation to test and distribute their software for multiple IT infrastructures, scientists are primarily focused on the results of their calculation. Still, with increasing simulation protocol complexity in the field of computational materials science workflows commonly scale beyond just a single computer system. The build automation component of a scientific IDE has to be focused on up-scaling calculation from the interactive prototyping to high-throughput parameter studies. This includes submitting calculation to a job scheduler of an HPC cluster, collecting the results of several individual calculations and finally summarizing the result.

Debugger: Given the requirements of computing on a distributed IT infrastructure, the debugger for a scientific IDE is also different from a debugger used for software development. It has to be able to collect intermediate results of unsuccessful calculation from the distributed IT infrastructure and assist the user in identifying the source of the error. This includes providing access to the input and output files of the specific simulation code, allowing the user to modify the input of the simulation code beyond the generic format of the IDE and finally flexible ways to redo previously unsuccessful parts of a complex simulation protocol.

While existing high-throughput frameworks allow to up-scale simulation protocols, they require the user to define complex computing pipelines which hinders rapid prototyping and debugging, as highlighted in Sec. 3.2.1. The core capability of the pyiron IDE is to support the interactive rapid prototyping of new simulation protocols, the up-scaling to high-throughput parameter studies and the ability to debug the simulation protocols all within the same simulation framework. Hence, the name pyiron an integrated development environment (IDE) for *ab initio* thermodynamics.

4.2.2 pyiron Objects

To address the challenges that come along with rapid prototyping, up-scaling and debugging of complex simulation protocols, an abstract class of objects – the pyiron objects – is introduced. The pyiron objects represent the different components of a simulation protocol, as the project, the atomistic structure, the computing infrastructure and the simulation code job objects for the various kinds of calculations. Each pyiron object has built-in interfaces to the user interface, the data storage and the resources associated with this object. Finally, pyiron objects can be nested inside each other. For example an atomistic structure can be nested in a DFT calculation job and a DFT calculation job can be nested in a job to calculate the energy-volume curve, using the nested DFT calculation job and its structure as a template for calculations at multiple strains. As a consequence, this hierarchy of different building blocks simplifies the construction of complex simulation protocols while maintaining the dependence of the objects.

User Interface

One of the goals of the pyiron IDE is that the user never has to leave the Jupyter notebook environment, as introduced in Sec. 3.3.1. The pyiron IDE aims at creating fully reproducible simulation protocols, which requires recording all the parameters specified by the user, and to record all user inputs it is essential that all functionality is available within the same interface. This is in contrast to existing environments to develop simulation protocols like ASE. While these environments were primarily designed to for the Python programming language, they can also be used inside Jupyter notebooks. They support the user to create atomistic structures and define the simulation code to compute energies and forces, but they do not provide any integration of an HPC job scheduler. So in order, to up-scale a simulation protocol developed in ASE the user has to submit the python script to the HPC job scheduler manually. In the same way the user has to define the data storage manually and specify which data is stored as discussed in Sec. 3.1.2.

In contrast the pyiron IDE addresses this issue by integrating the user interface in each pyiron object. This includes reducing the number of import statements the user has to remember and focusing on tab-completion to support the interactive rapid prototyping in the Jupyter notebook environment. In addition each simulation code job object includes a server object which is used to specify the computing environment where the calculation is executed. Finally each object can be rendered in the Jupyter notebook to give the user a visual representation of the internal structure. This includes the visualisation of atomistic structures in the Jupyter notebook using the NGLview library [167]. Therefore, during the whole process of developing the simulation protocol, submitting an initial set of calculations, analysing these and then continuing the development the user never has to leave the Jupyter notebook and all steps can be documented. An example screenshot of an active pyiron IDE session is pictured in Fig. 4.2. It includes multiple windows all accessible via the Jupyter lab environment with a standard web browser. In particular the pyiron IDE can be either installed locally on the user's workstation computer or directly on the HPC cluster to minimize the data transfer by keeping the simulation results on the HPC cluster instead of transferring them to the client computer and to maintain the reproducibility. This approach is beneficial especially within one group of collaborating researchers who share the same computing infrastructure. By having a single installation of the pyiron IDE directly on the HPC cluster rather than individual installations on the collaborators workstation computers the administrative effort can be drastically reduced. Still, the pyiron IDE supports both configurations for compatibility.

Data Storage Interface

The data storage for *ab initio* thermodynamics is challenging, because different calculations have different data storage requirements. On the one hand for computing the energy-volume curve it is sufficient to store the volumes and the corresponding energies. On the other hand thermodynamic integration calculation requires the forces to be calculated from two different simulation codes at each time step of the molecular dynamics trajectory. So, to reproduce the thermodynamic integration it is necessary to store not only the averaged free energy difference but also the forces of all atoms over the whole trajectory. Different simulation frameworks therefore use different data storage options, compare Tab. 3.1, ranging from relational Structured Query Language (SQL) databases (ASE and AiiDA) to unstructured non SQL (NoSQL) databases (Materials Project). The advantage of NoSQL databases being that they do not require a relational model and are therefore more suitable for unstructured data. The structured and unstructured data of atomistic simulation codes is defined in the following:

Structured Data: This is data which is independent of the simulation code and methodology used. This includes technical details like the location of the calculation on the file system, the status of the calculation and the dependency of the calculation to other calculation as well as physical parameters like the chemical composition. Storing structured data in an SQL database enables the user to efficiently query the database of existing calculation.

Unstructured Data: This is data which is code-specific and strongly depends on the calculation method. It can range from a scalar value of the final energy at the end of an electronic convergence to an atomistic trajectory of a thermodynamic integration calculation which requires storing all positions and forces at every time step. Storing this unstructured data in an SQL database is not efficient, as it leads to sparse databases with poor performance. Instead an unstructured NoSQL database or optimized file formats like HDF5 are more suitable.

To address the combination of structured meta-data for unstructured datasets object-based storage is commonly used. Still, the existing object-based storage systems are neither optimized for scientific data nor for HPC-infrastructures. The pyiron IDE based on the experience with PyCMW implements such an object store based on the HDF5 format introduced in Sec. 3.3.3 and an relational SQL database. This has three advantages:

Generic Path: The internal hierarchy of the HDF5 file is used to extend the hierarchy of the filesystem. This is implemented by using the SQL database as an index of the HDF5 files distributed on the filesystem. The users experience the combined hierarchy as one continuous generic path. They can place pyiron objects at any position of the generic path and reload the same objects using the same generic path. In the background the path is split in a filesystem path and an HDF5 file internal path. A typical example is a pyiron object nested in another pyiron object. In this case the parent object is placed in the root of the HDF5 file and the nested object is placed in a subpath of the same HDF5 file.

Parallel Writes: To prevent multiple compute nodes to write to the same HDF5 file, the status of the pyiron objects is tracked in the SQL database. This prevents the same object to be loaded in write-mode twice. Still, in the case a large calculation is executed in a job scheduler allocation with multiple compute nodes using the message pathing interface (MPI) then the MPI can be used to coordinate the file access within the job scheduler allocation, while for all other compute nodes the object remains read-only.

HPC-Compatibility: Besides the SQL database, the HPC job scheduler and the shared file system, the pyiron IDE does not require any additional daemon processes. While the HPC job scheduler and the shared file system are available on all HPC clusters a database which is accessible and sufficiently scalable to be accessed from all compute nodes in the HPC cluster at the same time can be a limitation. In this case pyiron is backwards compatible and uses the file system meta data to store the calculation dependence. Still the use of an SQL database is highly recommended as an efficient index.

With the direct link between the pyiron objects and the HDF5 file, the pyiron IDE can manage the data storage automatically. So the user never has to save a pyiron object manually instead the object is automatically saved before the calculation is executed and once more after the execution of the calculation. In case the execution is not successful this helps to debug the status of the pyiron object before the execution. By maintaining a direct relation between the pyiron objects and the data storage in the HDF5 file, the pyiron IDE benefits from the efficiency of the HDF5 internal index. In summary, this new development combines the hierarchy of the file system the user is familiar with with the computational efficiency of an object store optimised for scientific data while maintaining compatibility to the IT infrastructure of existing HPC clusters.

Resources Interface

Besides the connection to the HDF5 file, pyiron objects can also be connected to external resources. These external resources can be executables, parameter databases or computing resources like an HPC cluster. The complexity of a modern DFT simulation code in combination with the complexity of thermodynamic simulation protocols emphasises the need for an integrated solution. As a consequence the simulation code interface of the pyiron IDE not only supports quantum engines to compute energies and forces (level one) but also more thermodynamics modules (level two) if these are available in the simulation code. This is required for *ab initio* thermodynamics to calculate molecular dynamics trajectories computationally efficiently especially for interatomic potential simulation codes like LAMMPS, which already include highly optimised parallel implementations for molecular dynamics.

In addition the pyiron IDE is focused on shielding the user from the complexity of interfacing to the simulation code. It uses one central resource configuration directory, which includes links to the executables, scripts for interfacing with the job scheduler as well as parameter databases like interatomic potential files or pseudopotential files. So if multiple users on the same HPC cluster use the pyiron IDE they can share their resources. Also with the links to these executables as well as to the job scheduler of the HPC cluster being based on shell scripts, they are easy to modify for advanced users without the need to modify the source code of the pyiron IDE. At the same time these shell scripts enable the pyiron IDE to support multiple versions of the same executable which simplifies the process of debugging newly released versions or own modifications to the source code of the simulation code. Finally, all open-source software codes integrated in the pyiron IDE are released as conda packages in binary format on the conda-forge community channel, which was introduced before in Sec. 3.3.2. So they can be directly downloaded during the installation to give new the pyiron IDE users a head start with an already configured system, with the required simulation codes already set up. The benefit is twofold: On the one hand this central configuration simplifies the daily use of the pyiron IDE, because the users have to set up the pyiron IDE only once and afterwards there is no need for him to remember the location of the parameter database on the file system or the dependencies which need to be loaded for a specific executable. On the other hand by separating the technical configuration of the resources from the simulation protocol, the simulation protocol is more transferable, because it does not contain any information which is specific to the computing resources used.

It is still possible for malicious users to actively manipulate their own simulation data. This cannot be stopped, but with reference implementation of the open source executables being available as part of the conda-forge channel, it is the choice of the user to either use these executables or validate that their optimized or modified executable agrees with the reference. Therefore, by separating the configuration of the resources from the simulation protocol and by providing reference binaries for all open-source executables, the pyiron IDE offers fully reproducible simulation protocols, which can be added to the publication as supplementary material to give the readers the option to reproduce the same results if they are willing to invest the required computing time. This level of reproducibility previously required a dedicated simulation infrastructure [58] and is now included in the pyiron IDE out of the box. More details how to publish a simulation protocol with the pyiron IDE are provided in Sec. 4.4.4.

4.2.3 Interaction of pyiron Objects

While the individual pyiron objects already address the reproducibility of simulation protocols, the interaction of pyiron objects is key to support the up-scaling for high-throughput parameter studies and especially the rapid prototyping. By being easy to use starting from the development of an individual simulation protocol up to the execution of a high-throughput parameter study the pyiron IDE accelerates the productivity of the scientists using it. Still, it is very difficult to quantify the ease of use, as it strongly depends on personal preference and previous experience. But what can be quantified is the number of lines required to reproduce a given simulation protocol, the number of commands a new user has to learn to construct simulation protocols from scratch and finally the number of required parameters the user has to remember.

The unix approach [176] of modularity and reusability by using specialist tools is a perfect example how a set of self-consistent tools can be combined to address a higher level of complexity. In the same way the pyiron objects are self-consistent and can be combined to enable the construction of even more complex simulation protocols. For this purpose the pyiron objects represent the building blocks of simulation life cycle in Fig. 4.1 from the atomistic structure over the simulation job up to the project level. The goal is to enable switching to the pyiron IDE easy for both types of users: users who are already experienced with python programming as well as those who previously created their simulation protocols primarily on the command line but encountered limitations when it comes to up-scaling complex simulation protocols.

Serialization

HPC cluster offer two main challenges: One is to efficiently distribute the computing tasks and the other is to collect the results from the distributed computing resources after the calculation finished. For the first part, the pyiron IDE communicates with the HPC job scheduler via shell scripts, as part of the resources interface, discussed in the previous section. The second part is commonly addressed in HPC clusters by having one shared file system which is shared between all computing nodes. So in the case a simulation fails to execute successful, the input files can be analysed on a separate computing node. But communication over the file system, by writing input files and parsing output files, is slow in particular when using a shared file system which is accessed by many computing nodes at the same time. Therefore, the pyiron IDE in addition to the file-based communication also supports direct communication of simulation processes and the pyiron objects during the runtime of the simulation code. While this accelerates the communication, it also makes debugging an unsuccessful calculation more complicated, because it is difficult to reconstruct the current state of the calculation at the point when the calculation failed.

To address this issue of distributed memory access on a regular HPC cluster with a shared file system the pyiron IDE leverages the exception handling capabilities of Python and the pyiron data storage, introduced in Sec. 4.2.2, for object serialisation. During the execution of the simulation code, the pyiron IDE maintains a Python process to communicate with the simulation code and track the progress of the simulation code. This Python process remains running even in the case the simulation code fails. So, the pyiron IDE is able to store the current status of the calculation after the failure of the simulation code before ending the Python process. By storing the full status of any pyiron object in the pyiron data storage namely serializing it. The advantage of serialisation is that objects can be stored on one computing node in the pyiron data storage and can be reloaded on another computing node. Especially in the case of debugging an unsuccessful calculation, this is helpful because it allows to reconstruct the last state of the pyiron object and the last state communicated with the simulation code, which might already give an indication for the reason of the unsuccessful calculation. Commonly this kind of distributed memory requires special hardware, in the case of the pyiron IDE the object orientation and serialisation enable this feature on regular HPC clusters with a regular shared file system.

In comparison to the default Python serialisation pickle the pyiron implementation has two advantages. On the one hand in contrast to storing the full memory footprint the pyiron IDE only stores the input and output of a given pyiron object while the object definition is already included in the source code. On the other hand by using the pyiron data storage it is possible to deserialise a nested object from a parent object without reloading the parent object. To achieve this functionality each pyiron object has to implement a serialisation and deserialisation routine, which is not required for pickle. These serialisation and deserialisation routines are implemented in the generic pyiron object class so each object derived from the generic pyiron object class, only needs to adjust the serialisation slightly when a new class of pyiron objects is implemented. Still, from the user side these functions are never called, instead when the user submits a calculation, the pyiron IDE automatically stores the state of the object, hiding this technical complexity. The users can therefore debug calculations which are executed on an HPC cluster just like they debug calculations on his local workstation, which is in contrast to existing simulation frameworks and accelerates the rapid prototyping and up-scaling of complex simulation protocols. This consistency of distributed memory which is achieved in the pyiron IDE by implementing serialisation for each pyiron object is commonly called orthogonal persistence [177], because the object state outlives the process it was created in.

Data Analysis

Reusing the example of calculating the energy-volume curve in Sec. 4.1, the iteration over different elements, requires the up-scaling of the simulation protocol like it is done in the Delta project [34] which iterates over both the elements and different simulation codes. The development of a typical workflow can be divided in three phases:

Rapid Prototyping: The first step is to interactively develop the simulation protocol. In the case of calculating the energy-volume curve this included determining the initial guess for the equilibrium volume to distribute the volumes around and select a equation of state to use. A typical initial guess would be the experimental lattice constant and a equation of state can be selected from Sec. 2.4.1.

Testing: With the initial guess and the equation of state defined, the simulation protocol can then be tested with a subset of parameters to validate the stability and reduce the number of corrections required.

Parameter Study: After the testing, the calculation for the parameter study over all parameters is submitted to a HPC cluster, even when the previous two steps are executed on a local workstation computer.

After the calculations the user has to address two challenges: the first is aggregating the data of the calculation which were executed successfully to identify correlations and the second is handling the calculations which failed to identify patterns which led to failures in the calculations. Both of these challenges are addressed by the pyiron IDE. For the aggregation of data the pyiron IDE follows the MapReduce method [6]. The pyirontable object takes a project object as input, in combination with a series functions. Each of these functions takes a job object as input and return a parameter. The functions are then executed on each job object in the corresponding project and the resulting parameters are stored in a pandas table [178] with one row per job. By aggregating the calculation results in one table it is possible to study correlations between the different parameters. In contrast to other solutions the pyiron IDE is not limited to a specific set of predefined parameters. Instead, with the flexibility of user-defined functions, the pyiron IDE can be adjusted for the needs of a specific project. This implementation of the MapReduce model to map a given set of functions to a series of jobs and afterwards reduce the output to a single object is therefore one of the core components of the pyiron IDE to up-scale complex simulation protocols from rapid prototyping to high-throughput parameter studies. In the same way the pyirontable object can also be used to identify patterns in unsuccessful calculations. While the pyiron IDE is by default parsing for known error patterns, this can again be extended by user-defined functions. Parsing the output files is not as efficient as loading the already parsed datasets from the HDF5 files. Still, the same methodology applies.

Factory Pattern

The previous two features addressed the reproducibility and the up-scaling of simulation protocols, the third addresses the rapid prototyping of simulation protocols. Instead of creating the pyiron objects from their classes, which would require the user to import the classes separately, the pyiron IDE is able to create new pyiron objects from existing pyiron objects independently of their classes, which is called factory pattern. This not only reduces the number of lines the user has to write and remember but at the same time it allows the pyiron IDE to track which pyiron object was used to create another pyiron object. With this information it is possible to create the provenance graph which tracks the dependency of the objects, a feature initially proposed in AiiDA, as intoduced in Sec. 3.1.2. In the pyiron IDE the dependency of the objects is stored as structured data in the SQL database, which accelerates the resolution of dependencies.

In practice the project object is the only pyiron object which is created from importing the class. With the project object the users define the location in the file sytem where to execute the calculation. From the project object the atomistic structure is created as well as the pyiron job object to execute a calculation. Finally a job object can again be used to generate more complex job objects. With the factory pattern the hierarchy of the objects is accessible for the pyiron IDE, while at the same time the required inputs from the user side are reduced to a minimum. An example how to create the individual objects is discussed in the next chapter.

The same factory pattern is used when deserialising a job from an HDF5 file. The pyiron IDE first accesses the HDF5 file with a generic pyiron object class. The HDF5 file then contains the information which object type is stored inside the root level of the HDF5 file. So the pyiron IDE automatically loads the required classes in the background to be able to deserialise the corresponding object. As a consequence, the pyiron object on the one hand simplifies the user interface as the user explicitly creats one pyiron object from another one. On the other hand the resulting hierarchy is matched to the pyiron data storage which simplifies the debugging and the data analysis. So, while the underlying concepts of the pyiron IDE are complex, they successfully reduce the technical complexity on the user side. As a result of this, the users can focus on the scientific complexity of their simulation protocols.

4.3 Example Workflow

To highlight the concepts introduced in the previous section, they are now applied to demonstrate the functionality of the pyiron IDE from a practitioner's perspective. Following the simulation life cycle in Fig. 4.1 the necessary pyiron commands for the specific steps are introduced with a focus on the ease of use and the rapid prototyping.

4.3.1 Installation

Comparing the ease of use of a software is always subjective. Still while other software packages require the user to configure the dependencies and compile their software manually, the pyiron IDE can be directly installed via the conda package manager from the conda-forge community channel [168]:

```
conda install -c conda-forge pyiron
```

This command not only downloads the pyiron IDE, but also the required dependencies like spglib for identifying crystal symmetries and h5py the Python interface for the HDF5 file format as well as several others. The advantage of the conda package manager over the pip package manager as well as the general concept of the conda package management are discussed in Sec. 3.3.2. In addition to pyiron also the interatomic potential code LAMMPS and the DFT code S/PHI/nX are downloaded using:

```
conda install -c conda-forge lammps sphinxdft
```

Both of these simulation codes as well as several others and software utilities for computational materials science have been contributed to the conda-forge community as part of this thesis. This results in a total of over 250 software packages and over ten million downloads, which demonstrates the need for lowering the technical barrier in computational materials science. Another optional step is the installation of the visualisation library NGLview to visualise atomistic structures directly in the interactive Jupyter environment:

```
conda install -c conda-forge nglview
```

With a total of three commands the pyiron IDE is installed and configured for all examples in this section. The more extensive application in Chap. 5–7 requires additional software packages. While the conda package manager supports all major operation systems, the majority of the computational materials science simulation codes are primarly developed for the Linux operation system, so the corresponding packages are also restricted. For more advanced configuration options including the addition of the VASP simulation code and the configuration of HPC clusters are available on the pyiron website. By embracing the use of the conda package manager for computational materials science, the pyiron IDE on the one hand enable users to test and compare different simulation codes and on the other hand it improves the reproduciblity of simulation protocols. The publication process is discussed in more detail in Sec. 4.4.4.

4.3.2 Project Object

After the successful installation a folder for pyiron projects is created and the Jupyter notebook environment is started in this folder:

```
mkdir -p ~/pyiron/project
cd ~/pyiron/project
jupyter notebook
```

All the following python commands are then executed in individual cells of the Jupyter notebook. To create the first pyiron project, the project class `Project` is imported from the pyiron module `pyiron` and afterwards used to create an instance of the project class `Project`. The path parameter `path="projectname"` of the project class `Project` defines the folder of this project:

```
from pyiron import Project
pr = Project(path="projectname")  # ~/pyiron/procet/projectname
```

The pyiron IDE has no limits to the number of jobs inside one project and also allows to access and manipulate the same project with multiple simulation protocols. Still, in practice the project-based structure including arbitrary levels of subprojects helps the users to structure their data. It is similar to working with a classical file system-based approach, which helps to map existing workflows to the pyiron simulation protocols. To be able to track the modifications of the user and generate a consistent simulation protocol it is essential for the pyiron IDE that all steps are executed inside the Jupyter notebook and the data is not manipulated outside the Jupyter notebook. However for the intermediate process of migrating a simulation protocol to the pyiron IDE or debugging an unsuccessful calculation the direct access via the file system can be helpful. Following the nomenclature of the HDF5 format, the project object supports the list function for groups `list_groups()` and the list function for nodes `list_nodes()`. In the HDF5 format a group can contain both groups and nodes, while nodes only contain datasets. In the same way the pyiron IDE implements list groups for subprojects and list nodes for pyiron objects. Finally, the path property of the pyiron project object `path` returns the current directory on the file system where the project is located.

```
pr.list_groups()
>>> [] # ["sub_project_a",...]
pr.list_nodes()
>>> [] # ["calculation_1", ...]
pr.path
>>> "~/pyiron/projects/projectname"
```

With these very basic commands the user can already navigate through the current folder and iterate over existing objects, if there are any. In this example there exist no additional objects yet. To create a new project object from an existing project, the pyiron IDE supports the create-group function `create_group()` in analogy to the HDF5 format. After the new group is created the user can navigate to this group using the edge bracket notation which is commonly used in python to access items in another object. To demonstrate this functionality a subproject for the existing project named "new" is created and then accessed via the original project object:

```
pr_new = pr.create_group("new")
pr_new.path
>>> "~/pyiron/projects/projectname/new"
pr_new_2 = pr["new"]
pr_new_2.path
>>> "~/pyiron/projects/projectname/new"
```

Both variables `pr_new` and `pr_new_2` reference the same object. The users are not forced to keep track of all objects in memory, but instead simply create the pyiron objects they need. If these objects already exist, the pyiron IDE reloads the existing copy instead of creating a new one. This flexibility again helps the users to create complex simulation protocols.

4.3.3 Create an Atomic Structure Object

Instead of importing the atoms class to create the atoms object, it is created from the project object following the factory pattern, introduced in Sec. 4.2.3. As the pyiron atoms class is derived from the ASE atoms class, it benefits from the utilities ASE provides to create atomistic structures, with ASE being introduced in Sec. 3.1.2. One of these utilities is the bulk class which creates bulk structures of the equilibrium phase with the experimental lattice constants. Only the symbol of the element has to be selected. In this case it is `"Al"` for aluminium:

```
structure = pr.create.structure.ase.bulk("Al")
```

This creates a primitive fcc aluminium cell with just a single atom. To create the cubic supercell the parameter `cubic=True` is added:

```
structure_cubic = pr.create.structure.ase.bulk("Al", cubic=True)
```

Using the factory pattern for creating the atoms object has two advantages. First the user does not need to remember the path to import the atoms class from, he can simply use tab-completion on the project object and second the atoms object inherits the location on the file system from the project. So, to serialize the atoms object in a separate file path is not required. This automatic inheritance creates the provenance of the objects. It is stored in the Jupyter notebook and on the file system based on the hierarchy of the objects. Finally with the function to create an ASE structure `create.structure.ase.bulk()` directly accessible from the project object the user can use the question mark character to access the built-in documentation. It shows all input parameters, the lattice constant `a`, the crystal structure `crystalstructure` and the ratio of the lattice constants for hcp crystals `c_over_a`, as well as short explanations for each input parameter.

```
pr.create_ase_bulk?
```

So in principle, knowing how to create the first project object in combination with tab-based auto completion and in-line documentation should be sufficient to learn more about the pyiron IDE while using it. After the creation, the atomistic structure is visualised in the jupyter notebook using the NGLview [167] library. For this purpose the supercell is repeated nine times in each direction using the repeat function `repeat()` and afterwards it is visualized using the plot 3D function `plot3d()`. While the repeat function is inherited from the ASE library the plot 3D function is a pyiron IDE specific extension for compatibility with the jupyter environment.

```
structure_repeated = structure.repeat([9,9,9])
structure_repeated.plot3d()
```

This highlights the pyiron approach of reusing existing open source simulation codes and utilities rather than reinventing them with a special focus on compatibility with the interactive jupyter environment.

4.3.4 Job Object

The pyiron job object interfaces to a simulation code. Depending on the functionality of the simulation code the job object can either only calculate energies and forces or execute internal simulation protocols like computing molecular dynamics trajectories. The simulation code internal implementation is computationally more efficient than the python implementation in particular for interatomic potential codes.

Calculate Energy and Forces

To calculate the energies and forces, a pyiron job object is defined. Following the factory pattern the create-job function `create.job.Sphinx()` of the project object is used to create a S/PHI/nX simulation job object `job`. The S/PHI/nX simulation code is introduced in Sec. 3.1.2. The function only requires one argument, the name of the job, which has to be unique in the name space of the project to be able to reload the job object later on. By using tab-completion the job type can be selected without the need to remember the exact class name. Internally, this loads the S/PHI/nX class and sets it as a type of the job object. After the creation of the S/PHI/nX job object the atomic structure object which was created above is assinged and the calculation is executed by calling the run function of the job object `run()`.

```
job = pr.create.job.Sphinx(
    job_name="sphinx_al"
)
job.structure = structure
job.run()
>>> "The job sphinx_al was saved and received the ID: 1"
```

Calling the run function triggers a series of internal steps, namely steps 3.-8. of the simulation life cycle illustrated in Fig. 4.1. First the job object is serialized to the HDF5 file, which is located in the path given by the project object and according to the name of the job object. The serialisation also includes the structure object which was assigned to the job object. It is therefore possible to reload the objects by navigating through the objects using their generic path, similar to navigating through the file system.

```
job_reload = pr["sphinx_al"].to_object()
structure_reload = pr["sphinx_al/input/structure"].to_object()
```

Adding the function `to_object()` returns the object, while by default only a pointer to the object is returned. By using the pointer it is possible to navigate to a property of the object and reload just a single property, rather than the full object. This is useful for data aggregation following the MapReduce methodology [6]. For the user there is no difference between navigating through the file system and navigating through the HDF5 files, both are accessible from the project objects. For the example of accessing the structure inside the job object the generic path is *"sphinx_al/input/structure"* with the job name of the job object *"sphinx_al"* which is represented on the file system by a HDF5 file named `sphinx_al.h5` and inside the group *"structure"* is accessed which is a subgroup of the input group *"input"*. This integration of the HDF5 format as an extension of the file system is unique in the pyiron IDE as introduced in Sec. 4.2.2.

In analogy to the structure object which is part of the generic input – step 3. in the simulation life cycle – also the output – step 7. – can be accessed in the generic format. Starting with the total energy there are three ways to access it: In addition to the HDF5 based access via the job object and the project object, the total energy can be accessed as an attribute of the output object of the job object `job.output.energy_tot`.

```
job["output/generic/energy_tot"]
>>> array([-57.23744137])
pr["sphinx_al/output/generic/energy_tot"]
>>> array([-57.23744137])
job.output.energy_tot
>>> array([-57.23744137])
```

The output object includes the output of the S/PHI/nX calculation in the generic code-independent format. This generic format in the pyiron IDE uses the International System of Units while the S/PHI/nX code internally uses atomic units. This is achieved by converting the units after the calculation to match the generic format. In the same way variable names are converted to follow the typology of the generic format. For the user the generic code-independent format has the advantage that analysis methods can be implemented independent of the simulation code, which enables switching the simulation code during the development of a simulation protocol. This generic format is another essential feature of the pyiron IDE which supports the rapid prototyping because users can gradually increase the complexity of their calculation. They start with an interatomic potential simulation code and switch to a DFT simulation code afterwards. To demonstrate this the previous calculation is repeated with the interatomic potential code LAMMPS:

```
job_lmp = pr.create.job.Lammps(
    job_name="lammps_al"
)
job_lmp.structure = structure
job_lmp.run()
>>> "The job lammps_al was saved and received the ID: 2"
job_lmp.output.energy_tot
>>> array([-3.36000002])
```

The job type in the create-job function `pr.create.job` is changed to LAMMPS and the job name parameter is set to `job_name="lammps_al"`. As no interatomic potential is defined the pyiron IDE automatically selects and interatomic potential based on the atomistic species. In this case the interatomic potential for nickel aluminium [55] is selected. The pyiron IDE also supports the user in choosing a corresponding potential by providing a list-potentials function `list_potentials()` for the LAMMPS interatomic potential code, which lists all potentials compatible with both the simulation code and the current atomistic structure. More detailed information about the available interatomic potentials is available via the view-potentials function `view_potentials()`, which provides additional information for each potential. By providing parameter databases for the individual simulation codes, the pyiron IDE helps new users to get started and at the same time encourages advanced users to compare different interatomic potentials during the development of their simulation protocols.

While the difference in total energy for the LAMMPS simulation code and the S/PHI/nX simulation code demonstrates that the user still has to be careful with his choice of parameters, the following sections are going to show that the energy differences at different volumes reproduce the same trend even though the total energies are different. Being able to switch between different methods seamlessly enables rapid prototyping in the pyiron IDE. Finally, after the successful calculation of the total energy it is confirmed that the atom in the supercell is not moving. In analogy to the total energy also the forces of the calculation can be inspected. For the inspection of the S/PHI/nX calculation the generic path is used:

```
pr["sphinx_al/output/generic/forces"]
>>> array([[[-0., -0., -0.]]])
```

The forces are zero as expected based on the symmetry of the periodic supercell. This demonstrates another feature of the pyiron IDE. Instead of extracting just a single property from the output files like the total energy, the pyiron IDE converts the whole output to the generic format and stores the output in the HDF5 format. This requires more time at the end of the calculation. Still, it is beneficial for rapid prototyping to validate the physical consistency without the need of an additional calculation or additional parsing of the output files.

Calculate Molecular Dynamics Trajectories

To go beyond the calculation of the energy and forces for a static structure, a second LAMMPS job object is created to calculate a MD trajectory for a temperature of 800 K. MD and the corresponding thermodynamic ensembles are introduced in Sec. 2.3. Instead of the single atom supercell used for the previous calculation the repeated structure from above is selected. After assigning the structure the generic MD function `calc_md()` is called with the parameter for the temperature set to 800 K `temperature=800.0`. This generic function is defined for all codes which support MD calculations to allow the user to switch from one code to another.

```
job_md = pr.create.job.Lammps(
    job_name="lammps_md"
)
job_md.structure = structure_repeated
job_md.calc_md(temperature=800.0)
job_md.run()
>>> "The job lammps_md was saved and received the ID: 3"
```

Other generic functions include the static calculation function `calc_static()` which is used by default, the minimization function `calc_minimize()` which allows to relax the structure, as well as DFT specific functions to set the convergence parameters like the function to set the energy cutoff `set_encut()` and the kpoint mesh `set_kpoints()`. A simulation protocol which uses generic functions can be applied to any simulation code which implements the necessary functionality by just changing the job type in the create-job function `create.job()` as demonstrated above for the LAMMPS interatomic potential code and the S/PHI/nX DFT code.

Define Computing Resources

With the increased size of the supercell containing 729 atoms in the $9 \times 9 \times 9$ repeated unit cell, it is reasonable to use multiple CPU cores. For parallel calculation LAMMPS internally uses the MPI. So the same calculation is repeated using the parallel version of LAMMPS with 8 CPU cores and executing the calculation in the background. This allows the user to continue the development of the simulation protocol while the calculation is in progress. Starting with a copy of the previous calculation by using the copy-to function `copy_to()` with the additional parameter to copy only the input and not the calculation results `input_only=True`. Afterwards, the number of CPU cores is increased to 8 cores `job_md_para.server.cores = 8` and set run mode to executing the calculation in the background `job_md_para.server.run_mode.non_modal = True`. Finally, the job object is executed using the run method `run()` in analogy to the previous calculation.

```
job_md_para = job_md.copy_to(
    new_job_name="lammps_md_parallel",
    input_only=True,
)
job_md_para.server.cores = 8
job_md_para.server.run_mode.non_modal = True
job_md_para.run()
>>> "The job lammps_md_parallel was saved and received the ID: 4"
```

By calling the run method the job object is serialised to an HDF5 file, but instead of executing the job object in the same processes like the python environment it is now sent to a background process. The ability to modify the way how a calculation is executed by specifying two additional parameters supports the users in up-scaling their simulation protocols. Typical examples are calculating larger structures or longer trajectories to achieve higher sampling rates. When the local CPU cores are no longer sufficient, the pyiron IDE supports up-scaling the calculation using a HPC job scheduler.

Following the same approach as linking to the executables the pyiron IDE uses shell scripts to link to the different job queues. This is driven from the experience that while modern HPC job schedulers support a wide range of options, they are always restricted by the available hardware. On the hardware side the ratio of available CPU cores to available memory is fixed. As a consequence instead of supporting all possible job scheduler options the pyiron IDE uses shell script templates which are accessible directly from the server object. The list-queues function `list_queues()` lists the available job queue templates and afterwards the user can assign one of the available job queues using the queue parameter of the server object `queue`. In the case of a pyiron IDE installed on a workstation with a connection to a pyiron IDE installed on an HPC cluster the queue names of the HPC cluster are listed and the jobs are transferred automatically in the background.

```
job.server.list_queues()
>>> # ["queue_small", "queue_big"]
# job.server.queue = "queue_small"
```

As the configuration of the job scheduler templates is specific to the job schedulers the user is referred to the pyiron website [179] for more details. In contrast to all previous options the specification of the queue is IT infrastructure specific, so this setting has to be adjusted when reusing the same simulation protocol on a different HPC cluster.

Advanced Input and Output

The generic format allows to switch from one simulation code to the next one with minimal modifications. Still, advanced users might require simulation code-specific functionality which is not available as a generic function inside the pyiron IDE. To address this limitation the pyiron IDE extends the generic interface with the ability to access the input and output files of the simulation code directly. To access the input file of the parallel MD calculation from above, the input object of the LAMMPS calculation which represents the central LAMMPS input file `job_md_para.input.control` is accessed:

```
job_md_para.input.control
>>> "   Parameter         Value"
>>> "0  units             metal"
>>> "1  dimension         3"
>>> "2  boundary          p p p"
>>> "3  atom_style        atomic"
>>> "4  read_data         structure.inp"
>>> "5  include           potential.inp"
>>> "6  fix___ensemble    all nvt temp 800.0 800.0 0.1"
>>> ...
```

The result of the generic MD function `calc_md()` is listed in line 6 where the ensemble is defined as NVT with a constant temperature of 800 K. This LAMMPS-specific input can be modified using the generic notation of the input object:

```
job_md_para.input.control["fix___ensemble"] = \
    "all nvt temp 900.0 900.0 0.1"
```

This notation is generic for all codes but the variables and the units of the variables are code specific. In addition to the input files also the executable can be modified. This demonstrated by comparing the job object of the molecular dynamics calculation with the job object of the parallel molecular dynamics calculation the pyiron IDE automatically switched from the serial executable to the parallel executable using the message passing interface (MPI):

```
job_md_para.executable
>>> "~/resources/lammps/bin/run_lammps_2021.07.01_mpi.sh"
job_md.executable
>>> "~/resources/lammps/bin/run_lammps_2021.07.01.sh"
```

As explained in Sec. 4.2.2 the pyiron IDE is using shell scripts to link to the executables because it simplifies the configuration and debugging. Finally, the pyiron IDE provides the user access to the original output of the simulation code – step 6. of the simulation life cycle.

```
job_md_para.decompress()
job_md_para["log.lammps"]
>>> "LAMMPS (1 Jul 2021)"
>>> "OMP_NUM_THREADS environment is not set. Defaulting to 1 thread."
>>> "  using 1 OpenMP thread(s) per MPI task"
>>> "units metal"
>>> "dimension 3"
>>> "boundary p p p"
>>> "atom_style atomic"
>>> "read_data structure.inp"
>>> "  triclinic box = (0 0 0) to (25.7 22.32 21.0)"
>>> "                 with tilt (12.8 12.8 7.4)"
>>> "  2 by 2 by 2 MPI processor grid"
>>> ...
```

By default the output is compressed to reduce both the data storage space and the number of files on the shared file storage, which commonly is a limitation of HPC clusters as more files increase the complexity of the backup. Inside the pyiron IDE the decompress function of the job object `decompress()` is used to access the output files like any other pyiron object with the edge bracket notation. In the last line of the output of the LAMMPS simulation code is using a $2 \times 2 \times 2$ MPI processor grid, which totals in 8 CPU cores as specified above. This validates the parallel execution of the LAMMPS calculation. Still at the same time this direct access to the output files could also be used to extract properties from the output which are not yet included in the pyiron IDE. Finally, to get a list of all input and output files, the list files function `list_files()` is used. For the parallel LAMMPS MD calculation the list files function returns the following list of files:

```
job_md_para.list_files()
>>> ['NiAl.eam.alloy', 'log.lammps', 'potential.inp', 'control.inp',
     'dump.out', 'error.out', 'structure.inp']
```

The combination of modifying the input files, adding new executables via shell scripts and the ability to manually access the output files from within the pyiron IDe is essential to address the development process of those who want to modify the simulation code, while still benefiting from the simulation management, data storage and user interface of the pyiron IDE.

4.3.5 Data Analysis

After the successful execution of the job object the calculation output can either be inspected using the job object as demonstrated in the previous section or using the project object. The second option has the advantage that it enables the separation of executing the calculation and the analysis of the results. A typical application represents the submission of the calculation to an HPC cluster and the analysis of them once all calculation are completed.

Inspect the Project

The previous sections introduced the generic functions to navigate the hierarchy of pyiron objects, namely the list groups function `list_groups()` and the list nodes functions `list_nodes()`. To simplify the navigation for the user, the pyiron objects by default return a combination of both lists. In the case of the project object `pr` above, it lists the subproject as well as the four calculations created in the previous section:

```
pr
>>> {"groups": ["new"],
     "nodes": ["sphinx_al", "lammps_al", "lammps_md",
               "job_md_parallel"]}
```

Again this functionality applies to all pyiron objects, so the reloaded job object for the S/PHI/nX calculation `job_reload` returns the same list of groups and nodes like the project object when accessing the calculation object `pr["sphinx_al"]`:

```
job_reload
>>> {"groups": ["input", "output"],
      "nodes": ["NAME", "TYPE", "VERSION", "server", "status"]}
pr["sphinx_al"]
>>> {"groups": ["input", "output"],
      "nodes": ["NAME", "TYPE", "VERSION", "server", "status"]}
```

By implementing the same functions for all pyiron objects, the usage of the pyiron IDE is simplified. Depending on the object the way how these functions are implemented internally differs. Still, to the user they resemble an coherent interface, which extends the default Python programming language. Following the same principle the project object also implements the lists file function `list_files()` introduced above for analysing the output files of a job object:

```
pr.list_files()
>>> ["lammps_al.h5", "lammps_md.h5", "sphinx_al.h5",
     "job_md_parallel.h5"]
```

Comparing the list of files with the list of nodes of the same project above, it is visible that by default each job object is associated with a separate HDF5 file to store the unstructured data.

Jobtable

Besides the file system based access to the pyiron objects the same pyiron objects can also be accessed via the database interface. To access the database the project object is used:

```
pr.job_table(
    columns=['id', 'status', 'job', 'project', 'totalcputime'],
    all_columns=False
)
>>> "   id  status    job                 project        totalcputime"
>>> "0   1  finished  sphinx_al           projectname/            3.0"
>>> "1   2  finished  lammps_al           projectname/            0.0"
>>> "2   3  finished  lammps_md           projectname/            2.0"
>>> "3   4  finished  lammps_md_parallel  projectname/            1.0"
```

By default the output of the job table function `job_table()` is filtered by the path of the current project, so it only contains the pyiron objects in this project. In addition the number of columns is restricted in the above example. A list of all available columns follows below. The job table lists the calculation, with their corresponding database identifier, the status of the calculation, the name of the object, the location on the file system and the run time. This is the only SQL database table inside the pyiron IDE. All the remaining unstructured data is stored in the HDF5 files. The SQL database provides four core capabilities:

1. It represents an index of all HDF5 files on the file system, so searching for a specific HDF5 file is accelerated compared to a traditional file system.
2. Based on the job status it is not only visible which calculations are currently in process, but in addition the job status also highlights job objects which are currently being written to the file system to prevent multiple write accesses. The serialisation which is implemented for each pyiron object in combination with this central file index which tracks the access status of the different objects enables the orthogonal persistence and handles the object access in the pyiron data storage.
3. The job table also contains the relation of the individual objects, either based on the path in the file system or by specifying the predecessor and superior object for nested objects, resulting in the full provenance being accessible to the user.
4. The job table is used to track the resource usage of different simulation codes, specific versions of a code and the computing environment, which helps users to optimise their simulation protocols for up-scaling.

To give a more detailed introduction to the job table, the individual columns are explained below:

- `id`: It is an integer identifier, which is mainly used internally to load job objects. The users can load objects based on the job id. Still, the numbering of job ids is continuous and therefore specific to the pyiron installation. So, a simulation protocol which relies on job ids is most likely not transferable.
- `status`: The status of a calculation indicates if a calculation is waiting in the job scheduler of the HPC cluster, is currently running or already finished. Based on the job status jobs which are currently being executed on a given compute nodes can be identified. As a consequence, these cannot be loaded on another compute node. Again this is important to maintain the orthogonal persistence of the objects inside the pyiron data storage.
- `job`: The job column lists the job name defined during the creation of the job object. The job name has to be a unique identifier in the name space of an existing project, which allows reloading job objects based on their job names, as demonstrated above. This has two advantages: first it is more intuitive for the user to remember job names rather than job ids and second the job names are independent of the specific installation of the pyiron IDE, which allows the user to transfer the simulation protocol from one installation to another one.
- `projectpath`: By default, the project path is `pyiron/project` as explained above in the installation Sec. 4.3.1. However, on an HPC cluster the directory structure might be restricted or users prefer to have separate volumes for different scientific projects. The pyiron IDE supports this by allowing the users to add multiple paths to the pyiron configuration.
- `project`: Inside the project path the pyiron IDE creates projects, like the project

created above named `Project(path="projectname")`. So the project column is directly related to the path on the file system and represents the location of the HDF5 file.

- `subjob`: Inside the HDF5 file the subjob column gives the path of the nested job objects. While for large calculation it is efficient to have one HDF5 file per calculations this is no longer efficient for very small calculations. Therefore, the pyiron IDE supports nesting multiple calculations in one HDF5 file to reduce the number of files on the file system.
- `chemicalformula`: To quickly obtain an overview of the available data inside a pyiron project, the chemical formula is stored in the database. This enables filtering by atomistic species. In particular for up-scaling an existing simulation protocol over the periodic table, filtering by chemical formula is used as a progress indicator.
- `computer`: To track the resource used by a given calculation, the pyiron IDE stores the number of cores and the job queue in the job table. Keeping an overview of the computational resource usage helps to improve the simulation protocols by addressing the most expensive calculations first.
- `totalcputime`: Besides the resources the pyiron IDE also tracks the CPU time. While the first version of a simulation protocols is commonly not optimised for efficient resource usage the optimisation is essential for up-scaling simulation protocols from rapid prototyping to high-throughput parameter studies.
- `timestart`: The total CPU time is calculated with the difference of start time and stop time. Still, recording the start time also helps to identify failed calculations, which have been started but never finished, as well as tracking the influence of software bugs. To be able to search for all calculations, which were executed in a given time frame, allows to identify all calculations which might be affected by a software bug no matter if it is a bug in the simulation code, in the simulation framework or any underlying software.
- `timestop`: For completeness also the stop time is recorded in the database.
- `hamilton`: The hamilton column defines the simulation code used. By filtering the database by simulation codes helps to identify the resource allocation in a given project and to enable up-scaling of the simulation protocols.
- `hamversion`: Based on the simulation code, the hamilton version column defines the version of the simulation code which was used. This again helps to track bugs in the simulation codes as well as identify customized executables, which might be required for the reproducibility of a given pyiron projects.
- `parentid`: The parent id defines the predecessor of an existing job by referencing its job id to construct the provenance graph. This allows the pyiron IDE to track the dependency of individual calculations.
- `masterid`: Finally, in analogy to the parent id the master id links to wrapper jobs which execute multiple calculations. Maintaining this relation directly available inside

the database accelerates the resolution of dependencies of separate calculations. The use of master jobs is discussed in more detail in the following section.

With the job table the structured data of job objects is directly accessible, while the unstructured data is stored in the HDF5 file. This combination of two different data storage solutions is one of the key features which enables the up-scaling of simulation protocols in the pyiron IDE based on the pyiron data storage. Besides job objects the same job table could in principle be extended for other object types as well. Still, in practice the primary use case of the job table is keeping track of the job objects, because these require the most computing resources and need to be debugged separately when they fail. All other pyiron objects are directly implemented in the Python programming language, use less computing resources and can be debugged interactive inside the Jupyter notebook session, so tracking them in the database is commonly not necessary but technically possible.

Analyse the Project

The analysis is the 10^{th} step of the simulation life cycle in Fig. 4.1. After the successful execution of the simulation codes and the automatic validation of the calculation, it is the task of the user to analyse and visualise the calculation results. Starting with the LAMMPS molecular dynamics calculation, the project object is used to directly access the output of the calculation, which is stored inside the HDF5 file of the corresponding job object.

```
e_pot = pr["lammps_md/output/generic/energy_pot"]
steps = pr["lammps_md/output/generic/steps"]
```

In this case both the evolution of the potential energy `"energy_pot"` over time as well as the timesteps `"steps"` are extracted. While the simulation codes typically calculate all properties at a given time step and write all properties as one object to the corresponding output files, the pyiron IDE converts this single time series of all parameters to one time series per parameter. For the user this simplifies the visualisation as the time series of individual parameters can be directly visualised using the Python plotting package [165]. The potential energy over time steps is plotted, to visualise the evolution of the potential energy:

```
import matplotlib.pyplot as plt
plt.plot(steps, e_pot)
plt.xlabel("Steps")
plt.ylabel("£Energy_{pot}£ [eV]")
```

By relying on standard Python packages and interfacing them with both the simulation codes and the distributed parallel object storage, the pyiron IDE enables the users to upscale their simulation protocol from rapid prototyping to high-throughput parameter studies towards data-driven science. Besides creating time series for each parameter, the pyiron IDE also converts the code-specific output to a generic format using the international units system. So the same command used for the LAMMPS calculation can also be executed for the S/PHI/nX created above:

```
e_pot = pr["sphinx_al/output/generic/energy_pot"]
```

The S/PHI/nX calculation is a static calculation, so the potential energy time series only contains a single entry, as it was already demonstrated during the calculation of energies and forces. Using this generic format in combination with loading job objects by name rather than job id, creates a generic simulation protocol, which can be transferred from one installation of the pyiron IDE to another one. This highlights the level of abstraction used in the pyiron IDE to separate the technical interface to the specific computing hardware from the physics of the simulation protocol. The technical configuration is stored in the pyiron configuration files and the simulation protocol is stored in the Jupyter notebook. The process of transferring and publishing a simulation protocol is discussed at the end of the next section. While accessing specific datasets from the HDF5 files of the job objects is already more efficient than loading the job object just to extract a single dataset, this feature can also be combined with the job database from the previous section. It enables iterating over all job objects in the current project and plotting the last entry of their potential energy time series:

```
for job in pr.iter_jobs():
    plt.plot(job["output/generic/energy_pot"][-1], label=job.name)
plt.ylabel("£Energy_{pot}£ [eV]")
plt.legend()
```

In the background the pyiron IDE is accessing the SQL database to select the job objects from the pyiron data storage to iterate over them. By default the function to iterate over all job objects in the project `iter_jobs()` loads the full job objects. So the analysis can be accelerates by adding the parameter to disable the conversion to pyiron objects `iter_jobs(convert_to_object=False)`. With this additional parameter the pyiron IDE is only accessing a pointer to the data stored in the pyiron data storage, which can accelerate the data analysis by an order of magnitude. On the one hand the obligation to add just a single parameter to accelerate the performance of the analysis is helpful for the user on the other hand loading the full object by default provides full access for interactive debugging, which is commonly used during the rapid prototyping. Again the pyiron IDE is focused on the up-scaling of simulation protocols from rapid prototyping to high-throughput parameter studies which is required for the development of *ab initio* thermodynamics simulation protocols and the iteration over the periodic table.

4.4 Energy-Volume Curve Calculation

Using the fundamental pyiron objects discussed before the equilibrium volume V_0, the equilibrium bulk modulus B_0, the derivative of the equilibrium bulk modulus B_P and the equilibrium energy E_0 are calculated following the simulation life cycle in Fig. 4.1. It is recommended to execute these calculation in a new Jupyter notebook, so this Jupyter notebook can be reused later.

4.4.1 Defining the Simulation Protocol

Starting with the import of the required modules, the plotting library [165], the numerical library [164] and the pyiron project class are imported.

```
# Import modules
import matplotlib.pyplot as plt
import numpy as np
from pyiron import Project
```

Following the import, in the second step the template objects are created, which are copied afterwards to create multiple calculations with the same settings. The S/PHI/nX DFT simulation code is selected with the default energy cut off and kpoint mesh. So adjusting these convergence parameters is not necessary. In addition the element aluminium is selected and defined as the variable `element = "Al"`. This keeps the simulation protocol as simple as possible. A more extensive discussion of the DFT uncertainty and the dependence of the equilibrium parameter convergence on the energy cut-off and the kpoint mesh follows in the Chap. 5.

```
# Create project
pr = Project("application")

# Parameter
element = "Al"

# Create structure and job reference
structure = pr.create_ase_bulk(element)
job_ref = pr.create.job.Sphinx(
    job_name="job_ref"
)
```

After the creation of the project object, the template job object and the template structure object, a for loop is created to iterate over eleven strains between 90% and 110% of the initial volume. For each strain the structure object template is copied and to achieve the volume deformations from ±10% the simulation cell is equally rescaled in all three directions with the set cell function `set_cell()`. Adding the option `scale_atoms=True` is required to not only rescale the supercell but also to update the positions within the supercell. Afterwards the structure is assigned to a copy of the job template object with an updated job name to include the strain for future reference. Finally, the calculation is executed using the run function `run()`. Again, these calculations are calculated in the background by setting the run mode of the server objects of each job object to non-modal execution `job_strain.server.run_mode.non_modal = True`. As a result, the pyiron IDE creates eleven Python sub-processes, one for each calculation. Depending on the number of available processor cores this level of parallelisation is more or less sufficient. The details of the parallelisation are addressed in the next section. For larger calculations it is recommended to submit each calculation to the job scheduler of an HPC cluster to delegate the job management. Still, in this example it is assumed that the user is executing the pyiron IDE on a local workstation with more than eleven processor cores.

```
# Loop over different strains
job_lst = []
for strain in np.linspace(0.9, 1.1, 11):

    # Strain the simulation cell
    structure_strain = structure.copy()
    structure_strain.set_cell(
        structure.cell * strain ** (1.0 / 3.0),
        scale_atoms=True
    )

    # Calculate Energy
    job_strain = job_ref.copy()
    job_strain.name = "strain_" + str(strain).replace(".", "_")
    job_strain.structure = structure_strain
    job_strain.server.run_mode.non_modal = True
    job_strain.run()
    job_lst.append(job_strain)
```

After submitting all the calculations to the background the Python process has to wait for the calculations to finish. Then the pyiron IDE iterates over all jobs in the project to wait until all calculations are finished using the iterate over job function in combination with the wait-for-job function `wait_for_job()` both of which are available from the project object.

```
for job in pr.iter_jobs():
    pr.wait_for_job(job)
```

Following the successful execution of all calculations, the same functionality is used to iterate over the pointers of the job objects to the pyiron data storage, to collect only the generic output. This is achieved by adding the parameter `path="output/generic"` to the iterate over jobs function, which provides direct access to the HDF5 file and preselects a path in the HDF5 file to access for each job object. So instead of accessing the entries for the volume and the total energy separately as it was done in the previous section, the preselecting of the path simplifies the access of multiple quantities inside the same job object. Afterwards, the Python plotting library is used again to visualise the energy-volume curve.

```
# Collect the energy volume pairs
vol, eng = [], []
for job in pr.iter_jobs(path="output/generic"):
    vol.append(job["volume"])
    eng.append(job["energy_tot"][-1])
```

```
# Fit and plot the energy volume curve
plt.plot(vol, np.poly1d(np.polyfit(vol, eng, 5))(vol), label="fit")
plt.plot(vol, eng, "x", label="calculation")
plt.xlabel("Volume [£\\AA^3£]")
plt.ylabel("Energy [eV]")
plt.legend()
```

To relate the individual steps with the simulation life cycle in Fig. 4.1, the definition of the project and the set up of the calculation up to the waiting for the jobs to finish the execution represent steps 2 and 3 in the simulation life cylce. After step 3 the pyiron IDE automatically executes the steps 4 to 8, namely executing the calculation and validating the successful execution of the calculations. Steps 9 to 11 are described in the last code block, the data of multiple jobs is collected. Afterwards the energy-volume curve is fitted with a 5^{th} order polynomial fit. Finally, both the calculation of the energy-volume pairs and the polynomial fit are visualised. Based on the results, the user can update the model and the project, for example by increasing the DFT convergence parameters to further investigate the calculation of the energy-volume curve, for the current material system of aluminium.

When the same notebook is executed again the pyiron IDE automatically identifies the calculations which have been executed before based on the database and loads the existing calculation results rather than executing the same calculations again. As a consequence it is not possible to modify the calculation input after the calculation finished to maintain the persistence of the calculations. To remove existing calculations, the users can either use the remove-job function `remove_job()` of the project object which takes a job id or job name as an input or remove all existing jobs from a selected project with the remove-jobs function `remove_jobs()`. The later is reasonable for rapid prototyping after a fundamental mistake was identified to have a fresh start for the next iteration of the simulation protocol. Commonly, users of the pyiron IDE focus on reusing existing calculations whenever possible, so deleting calculations should only be the last option.

4.4.2 Up-Scaling Existing Simulation Protocols

Once a simulation protocol as the one in the previous section is defined in a Jupyter notebook a separate Jupyter notebook can be used to submit multiple copies of the same Jupyter notebook. In this example the Jupyter notebook from the previous section is used to calculate the energy-volume curve of multiple elements. To achieve this, the parameter section is updated by accessing the get-external-input function `get_external_input()` of the project object which provides a dictionary with user-defined input parameters.

```
# Parameter
input_dict = pr.get_external_input()
element = input_dict["element"]
```

In this case, a single parameter the element is provided. Still the same functionality could be used to iterate over multiple parameters. Finally, a separate Jupyter notebook is created to submit the previous Jupyter notebook. To clarify the names of the Jupyter notebooks the previous Jupyter notebook, which calculates the energy-volume curve for a given element, is renamed to *"evcurve.ipynb"* and the second newly created Jupyter notebook is renamed as *"submit.ipynb"*. In this second notebook the following lines are added:

```
# Import modules
from pyiron.project import Project

# Create project
pr = Project("multielements")

# Loop over different elements
element_lst = ["Al", "Au"]
for element in element_lst:
    # Set up ScriptJob object
    job = pr.create.job.ScriptJob(
        job_name=element
    )

    # Input parameters
    job.input["element"] = element
    job.script_path = "evcurve.ipynb"

    # Execute job object in background
    job.server.run_mode.non_modal = True
    job.run()
```

The submission notebook starts with the import of the pyiron project object, followed by the creation of a new project object and afterwards a loop over multiple elements. For the example the number of elements is restricted to two, namely aluminium *"Al"* and gold *"Au"*. Inside the loop a script job object is created which takes a jupyter notebook as an input as well as a series of user-defined input parameters. In this case the notebook which calculates the energy-volume curve for an individual element is set as input notebook *"evcurve.ipynb"* and the element is added as an additional input parameter. Again the background execution is used to run the calculation in the background. As the individual calculation of the energy-volume curve consists of eleven calculation and two of these calculations are executed in parallel this results in 22 parallel processes, which are most efficiently executed if more than 22 processor cores are available. More advanced pyiron objects for the parallel execution of job objects are discussed in the Sup. A. The focus of this example is the simplicity of up-scaling an existing simulation protocol from rapid-prototyping to a high-throughput parameter study over the periodic table.

4.4.3 Analysing Parameter Studies

Before analysing this parameter study the pyiron IDE again has to wait until the execution is finished. As the calculations are executed in the background the job table function of the project object `job_table()` can be utilized to check the current progress of the calculations interactively or alternatively the same loop implemented above is used to wait until all calculations are finished. Here the second option is used to guarantee the execution of all calculations is finished before the next step is executed:

```
for job in pr.iter_jobs():
    pr.wait_for_job(job)
```

Following the successful execution of the calculation, the MapReduce method [6], which is implemented in the pyiron table object, is used to analyse the calculation results. The first step is to filter the job objects to select only the S/PHI/nX DFT calculations. This is achieved by a filter function which takes a job object as an input and returns true if it is a S/PHI/nX calculation. The job type can be checked with the name parameter `__name__` which returns the name of the job object class.

```
def get_sphinx_jobs(job):
    return job.__name__ == "Sphinx"
```

Just like the filter function also analysis functions can be defined to be mapped on the different job objects. In this example a function to extract the total energy from the generic output is used. The pyiron table object supports all kind of functions with the only restriction that they take a job object as an input.

```
def get_energy_tot(job):
    return job["output/generic/energy_tot"][-1]
```

In addition to user-defined functions the pyiron table object also supports a wide range of built-in functions. To execute the MapReduce method and to aggregate the output data of the parameter study, a pyiron table object is created using the create table function `create_table()` of the project object. Then the previously created filter function is assigned, two of the built-in functions are selected, namely the get volume function `get_volume()` to obtain the volume of the simulation cells and the get elements function `get_elements()` to extract the chemical elements and finally the user-defined function to extract the total energy from the generic output is added. Afterwards, the data aggregation is executed by calling the run function `run()`. The function to get the total energy from the generic output, which is defined above, is also available as a built-in function `get_energy_tot()`.

```
table = pr.create_table()
table.filter_function = get_sphinx_jobs
# table.add.get_energy_tot
table.add.get_volume
table.add.get_elements
table.add["energy_tot"] = get_energy_tot
table.run()
```

The table object internally is based on a job object. Thus, it is also possible to submit the data aggregation to a job scheduler on an HPC cluster for extended analysis or complex analysis functions. The pyiron table object then generates a list of job objects currently available in a given project and maps the assigned functions on these objects. Whenever new job objects are added to a given project the pyiron table object can be executed again, it automatically analysis only the newly added jobs. Or if new functions are added to the pyiron table object additional executions map only the added functions to the job objects to reduce duplicate executions. This flexible implementation of the MapReduce method is one of the core features of pyiron, because it allows the dynamic aggregation of data depending on the needs of a given simulation protocol. The aggregated data is then available as pandas dataframe [178] which is commonly used for data analysis and machine learning in the Python community. Therefore, the pyiron table object bridges the gap between the simulation data and the Python data science tools, just like the pyiron IDE bridges the gap between the interactive Jupyter notebooks and the simulation codes. To visualise the data for the user can iterate over the list of elements, select the corresponding subset of the data and plot the resulting energy volume curves for aluminium and gold using the integrated plotting library [165].

```
import matplotlib.pyplot as plt
df = table.get_dataframe()
for element in element_lst:
    df_el = df[df[element]=="1"]
    eng = df_el.energy_tot-df_el.energy_tot.min()
    plt.plot(df_el.volume, eng, label=element)
plt.xlabel("Volume")
plt.ylabel("Energy")
plt.legend()
```

To plot both energy-volume curves in the same graph the energy scale is normalised with the corresponding energy minimum. This comparison of two energy-volume curves obviously does not require the complexity of introducing the script job object and the pyiron table object. Still, with these objects it is now possible to up-scale the simulation workflow and iterate over the whole periodic table. In particular, when the underlying model to calculate the energy-volume curve is extended, for example by adding an optimisation of the equilibrium volume before calculating the energy volume curve. With the code above only the simulation workflow for calculating the energy-volume curve has to be adjusted and all the rest can be reused. In contrast by simply duplicating the notebook for each of the different elements this modification would require to change each copy once, which is very error-prone when executed manually. As a consequence, the pyiron IDE helps the users to up-scale their simulation protocols from the rapid-prototyping to high-throughput parameter studies.

4.4.4 Publishing the Simulation Protocol

With the increasing complexity in *ab initio* thermodynamics calculations it is essential to develop methods to accelerate not only the development of new methods but also the knowledge transfer within the community. The pyiron IDE contributes to this twofold:

- First, within the pyiron IDE the process of developing a simulation protocol is simplified for both the rapid prototyping and the up-scaling as discussed above. This enables the users of the pyiron IDE to implement existing methods faster.
- Second, pyiron supports the users to publish their simulation protocols.

In a traditional software environment for developing the simulation protocol on the command line it would in principle be possible to document every step. The users could document the way they compiled the simulation codes, how they setup their simulation environment and finally how they executed their simulations. But the challenging task is to separate the technical implementation for a specific HPC cluster from the physical simulation protocol. This separation is addressed in the pyiron IDE. Reference implementations for the open-source simulation codes are provided in the conda package manager via the conda-forge community channel [168]. The physical steps of the simulation protocol are documented in the jupyter notebook and finally simulation frameworks like the pyiron IDE connect the specific interfaces of the individual simulation codes with a generic interface to improve the readability of the simulation protocol. As a consequence the pyiron simulation protocols are commonly published in combination with the corresponding conda environment.

To demonstrate the publication of a pyiron simulation protocol the example of calculating the energy-volume curve from above is reused. The first step is to download the pyiron publication repository from the pyiron website. The publication template consists of a folder named pyiron for additional resources and finished calculation to include in the publication, an environment file `environment.yml` which defines the conda environment and an example Jupyter notebook `example.ipynb`. In addition to these files the repository also contains a manual file `README.md` a software license file `LISENCE` and scripts for continuous integration but these are less relevant for scientific purposes. The next step after downloading the publication template is exporting the conda environment. Using the command line utility the conda environment can be written to a file using the following command, as introduced in Sec. 3.3.2:

```
conda env export > publication/environment.yml
```

The above command adds all conda packages currently installed in a given environment to the file. This guarantees all packages are included. Still, to simplify the installation process for other users it makes sense to reduce the dependencies. For the example of calculating the energy-volume curve four packages are installed: the python programming language, the pyiron IDE, the Jupyter notebook environment and finally the S/PHI/nX DFT code.

```
channels:
- conda-forge
```

```
dependencies:
- python=3.9.5
- pyiron=0.4.4
- notebook=6.4.0
- sphinxdft=3.0.3
```

It is reasonable to fix the version numbers of the individual packages for maximum reproducibility, as future changes in one of the packages might require modifications of the simulation protocol. Alternatively, also specific resource files which are not available on conda-forge for example modified pseudopotentials could be included in the pyrion folder under resources. In this example the shell script and the pseudopotential database are used from the conda package, so the pyiron folder in the publication template can be removed. In case the user wants to include existing calculation in the publication these can be exported with the packing function `pr.pack()` of the project object. It creates an archive of the existing calculation which can be copied into the publication template. Finally the Jupyter notebooks `submit.ipynb` and `evcurve.ipynb` are copied to the publication template and the previous `example.ipynb` notebook is removed.

With these three steps, exporting the environment, adding existing calculations or modified resources in the pyiron folder and including the Jupyter notebooks the simulation protocol is ready for publication. When the simulation protocol is released on the Github service, then the simulation protocol is executed automatically as part of the Github continuous integration environment. This continuous integration helps to identify broken software dependencies or missing pyiron resources. In addition the same continuous integration is used to render the Jupyter notebook as HTML website which allows other users to browse it directly on the web. Finally, in case the continuous integration was successful the notebook can also be used with the MyBinder cloud service [180], which enables interactive testing for Jupyter notebooks with limited computing resources. The MyBinder cloud internally, just like the continuous integration service, uses conda to create the corresponding computing environment for a given Jupyter notebook. With the publication of a simulation protocol, as supplementary material for a new scientific method the loop of developing a simulation protocol which commonly consists of multiple iterations and extensions of the simulation life cycle is closed.

In addition the pyiron IDE also supports the user in the process of writing the publication by collecting the corresponding references of the methods used in a given simulation protocol. The list publications method of the projects object `list_publications()` returns the references to be included in a publication.

```
pr.list_publications()
```

With this, the pyiron IDE supports the user from rapid prototyping, over up-scaling the simulation protocol for high-throughput parameter studies to publishing the simulation protocol. As a consequence, the pyiron IDE reduces the technical complexity for the user to focus on the physics. More advanced simulation protocols are discussed in Sup. A.

5 Uncertainty Propagation

The theoretical overview in Chap. 2 highlighted that the complexity of *ab initio* thermodynamics can be divided in three dimensions of complexity, namely to control the *ab initio* uncertainty, to evaluate the free energy at finite temperatures and to disentangle the chemical complexity. The first of these challenges is addressed in this chapter, the other follow in the next two chapters. Controlling the *ab initio* uncertainty is a prerequisite for any DFT calculations. To accurately predict materials properties systematic convergence checks are required. The underlying uncertainty analysis has two purposes:

1. Guarantee the precision of the quantity of interest is below a predefined error bar.
2. Minimise the computational resources without sacrificing (1).

Performing such convergence checks manually, requires human expertise, it is a time-consuming and monotonous manual job that requires to perform routine calculations again and again. As a consequence, the chosen convergence parameters therefore often have a human bias. This is particularly true in large-scale high-throughput DFT simulation projects, as they were introduced in Sec. 3.2.1. These projects use simple rules to select the convergence parameters, rather than systematically executing convergence tests for each individual calculation to identify the optimum parameter sets. For qualitatively mapping of a large chemical compound spaces such a semi-empirical approach may be sufficient. However, many recently developed machine-learning approaches, as well as thermodynamic approximation to calculate the free energy, as introduced in Sec. 2.4.2 require target errors that are smaller than commonly used ones and that outliers or larger errors are absent.

For high-throughput calculations, but also for non-experts in DFT-calculations it becomes thus more and more important that convergence parameters have not to be provided by the user. Rather, they should be automatically computed taking as input only the atomic structure, based on atomic positions, cell size and shape, the exchange-correlation functional, which defines the intrinsic error, as introduced in Sec. 2.1.3 and the acceptable error of a given target quantity. In this chapter the fundamental ideas and the mathematical background of uncertainty propagation for plane wave DFT simulation codes are discussed and based on a parameter study a coarse-grained model is developed to predict the uncertainty for a given set of convergence parameters. The parameter study and the coarse-grained model are both realised with the pyiron IDE, introduced in Chap. 4.

5.1 Convergence Parameters

As outlined in Sec. 2.1 the DFT errors can be classified into (i) controllable ones that impact directly the precision of the result and (ii) intrinsic errors that affect accuracy, i.e., agreement with its experimental counterpart. In principle, also the implementation choices of the DFT simulation code, such as the used algorithms, mathematical libraries, compiler for the programming language etc. can affect the result. However, as recently shown in the Delta project, introduced in Sec. 3.2.2, in a systematic comparison of more than 15 simulation codes these errors are small and can be for typical applications neglected [34]. Only the unknown exchange-correlation functional and its approximation define the intrinsic error. All other parameters can be systematically improved to reduce the controllable errors below a predefined target error and are identified as convergence parameters. The pseudopotentials, introduced in Sec. 2.1.4, fall in between these two categories: While in principle their performance can be benchmarked against all-electron calculations, in practice the user has only a limited choice among available ones. In this thesis the focus is on plane wave DFT calculations with pseudopotential, since these provide a good compromise between accuracy and computational efficiency for materials science questions.

Critical convergence parameters are the size or completeness of the basis set, e.g. plane wave energy cut-off, the number of discrete **k**-points to perform the Brillouin-zone integration or the smearing parameter in the electronic occupation, as introduced in Sec. 2.1.4. Other important convergence parameters are related to size convergence, e.g. size of the supercell for point defects or slab and vacuum thickness for surface calculations or the size of the Fourier mesh to describe quantities such as charge densities or potentials. Commonly, the first step in choosing convergence parameters that are optimized both with respect to sufficiently small errors in the target quantity, while at the same time accounting for the computational efficiency, is the calculation of simple bulk quantities, such as equilibrium volume V_0, cohesive energy E_0, bulk modulus B_0 etc. In simulation codes with periodic boundary conditions, these quantities require only the calculation of the elementary bulk unit cell, which reduces the computational cost, compared to most other target quantities, e.g free energies, as introduced in Sec. 2.4.2. In most cases, they can even be calculated with a cell which consists of only a single, e.g. for fcc, or a few atoms. In the following a fully automated approach to determine optimized convergence parameters is derived. Both, the data sets used to derive the algorithm as well as the implementation of resulting coarse-grained model are done using the pyiron IDE. For the derivation the following choices are made:

- As DFT simulation code the plane-wave pseudopotential simulation code VASP, as introduced in Sec 3.1.2, is selected.
- Only cubic bulk systems consisting of a single chemical species will be considered. For this case, the equilibrium quantities, mentioned above, can be easily computed from a discrete set of energy-volume points and fitted with an EOS, as introduced in Sec. 2.4.1.
- As convergence parameters only energy cut-off ϵ_i and **k**-point sampling κ_j are con-

sidered. The size convergence is no issue for the perfect bulk and the Fourier mesh is determined in correlation of the energy cut-off to include all wave vectors up to $2\mathbf{G}$, as introduced in Sec. 2.1.4. To avoid having the electronic "smearing" parameter, as introduced in Sec. 2.1.4, as a third convergence parameter, the tetrahedron method with Blöchl correction is employed.

As a reference for the following analysis the recommendations suggested by the VASP-manual [181] are listed:

- Metals require approximately 1000 k-points/per atom for the same accuracy. For problematic cases (transition metals with a steep DOS at the Fermi-level) it might be necessary to increase the number of k-points up to 5000/per atom, which usually reduces the error to less than 1 meV per atom.
- The cut-off which is specified in the POTCAR file will usually result in an error in the cohesive energy which is less than 10 meV.
- Only volume changes of the order of $5 - 10$ % guarantee that the errors introduced by the basis set incompleteness are averaged out.
- For the calculation of the total energy in bulk materials we recommend the tetrahedron method with Blöchl corrections (ISMEAR=-5).
- In fact, we have evidence from comparison with all-electron calculations that the GW potentials are slightly superior even for DFT calculations.

To compare the optimized convergence parameters obtained by the automated approach derived in the following, a comparison with two references will be performed. The first is the Delta project that aims at providing a highly accurate description of the energy-volume curve to compare a large number of popular DFT-codes. In this project exclusively the PBE exchange correlation functional is used. The VASP calculations performed in the Delta project follow the above recommendations except for a few changes: $\mathbf{k}$-point mesh κ_j and energy cut-off ϵ_i are increased beyond the VASP recommendation and non-spherical contributions from the gradient corrections inside the PAW spheres [36] are included. The second reference is the materials project [142, 143], is introduced in Sec. 3.1.2. It uses an element independent energy cut-off ϵ_i for comparing different elements in alloys. To improve computational efficiency regular PBE pseudopotentials, which require a lower energy cut-off, as well as smaller $\mathbf{k}$-point meshes κ_j have been employed in this project.

To construct the energy-volume curve two additional choices have to be made: The number of volume points and the interval over which the are distributed. Interestingly, the two projects employ very different strategies. The Delta project is fitted based on seven equidistant volume points with a total spread of ± 6 %. In contrast, the materials project uses a 21 volume points homogeneously distributed over an interval of approx. ± 30 % around the equilibrium volume. The surprisingly large difference in chosen parameters and strategies for two projects that both aim at high precision indicates that criteria for a rational choice are highly desirable.

5.1.1 Magnitude of Errors

Before getting in an in-depth analysis of how the various errors depend on convergence parameters a simple estimate of their relevance is given. The equilibrium Bulk modulus B_0 is considered as target quantity. As it is related to the second derivative of the energy-volume curve it is rather sensitive to the choice of the convergence parameters. Two sources of error are considered: The convergence with respect to the energy cut-off and the choice of the pseudopotential. To avoid any influence from the other convergence parameters they are chosen such that their errors are negligible. Specifically, a **k**-point mesh of $51 \times 51 \times 51$ is used ($\kappa_j = 51$), which is roughly two orders of magnitude higher than the VASP recommendation. Further, a total of 21 volume points equidistantly spread around $\pm 10\%$ of the equilibrium volume are selected and a fit with an 11^{th}-order polynomial used, to prevent any bias of the EOS. The results of these calculations are shown in Fig. 5.1 for a selection of twelve elements. The pseudopotential related error is given by the difference at maximum energy cut-off to the all-electron result, as dashed gray line. The experimental value is given by the horizontal black dashed line.

For all elements the agreement between the all-electron result and the various pseudopotentials agree within 1–2 GPa. This error is generally much smaller than the non-controllable due to the exchange correlation functional, the difference to the experimental value, which can be as large as 25 GPa as for Ir. This finding agrees with the general observation that errors in elastic constants and bulk modulus are on the order of 10% [182]. Thus, the error due to the pseudopotential is an order of magnitude smaller than the error related to the unknown exchange correlation functional. With respect to the energy cut-off, the results summarized in Fig. 5.1 show the behavior expected for a convergence parameter: For too small values it dominates the error, it is non-monotonous making it difficult to estimate the error from an extrapolation but converges for high, but computationally highly expensive, values to a constant value. Also, as expected the convergence depends on the specific pseudopotential, with potentials including energetically low-lying and thus spatially more localized semi-core states showing with a few exceptions a slower convergence.

5.1.2 Energy Convergence

In the previous discussion the convergence of the equilibrium parameters was analysed. Here the convergence of the total energy and the energy difference for two different volumes is analyzed in Fig. 5.2. All results are calculated for a primitive aluminium supercell using the Al_GW PBE pseudopotential, which is also used in the delta project. The top row shows the convergence over energy cut-off ϵ_i with a linear increase in 20 eV steps. In analogy, the bottom row shows the convergence over **k**-point mesh κ_j with a linear increase in $2 \times 2 \times 2$ steps. In both cases the convergence is compared for the total energy of two different volumes (left), the energy difference of these (middle), and the relative convergence for all three (right). The relative convergence is calculated by subtracting the result at the maximum of the convergence parameters from all other values and is plotted as an absolute norm on a semi-logarithmic scale.

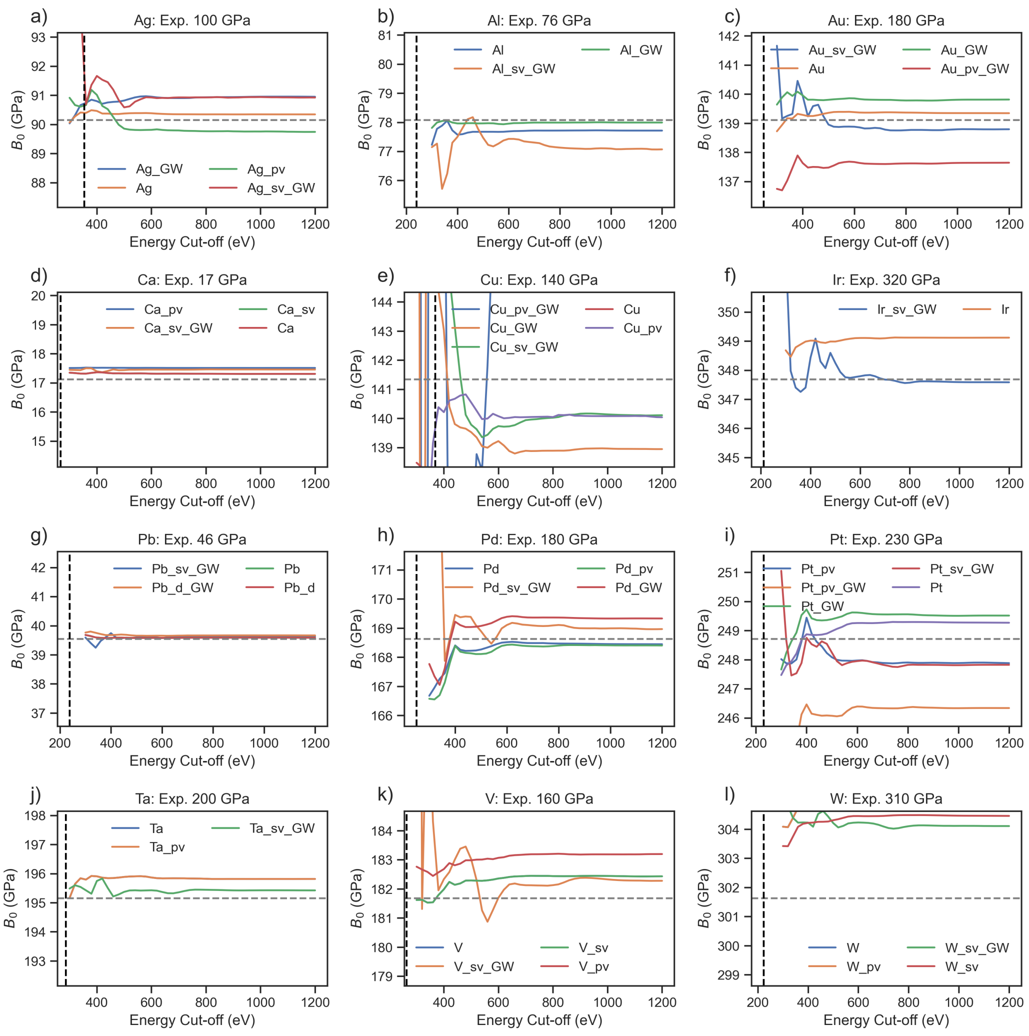


Figure 5.1: Comparison of the convergence of the bulk modulus B_0 over energy cut-off ϵ_i for a total of twelve elements in dependence of the pseudopotential. In addition the experimental bulk modulus is given in the title and the grey dotted line is the all electron reference from the Delta project [34]. The VASP recommended energy cut-off is indicated as vertical black dashed line.

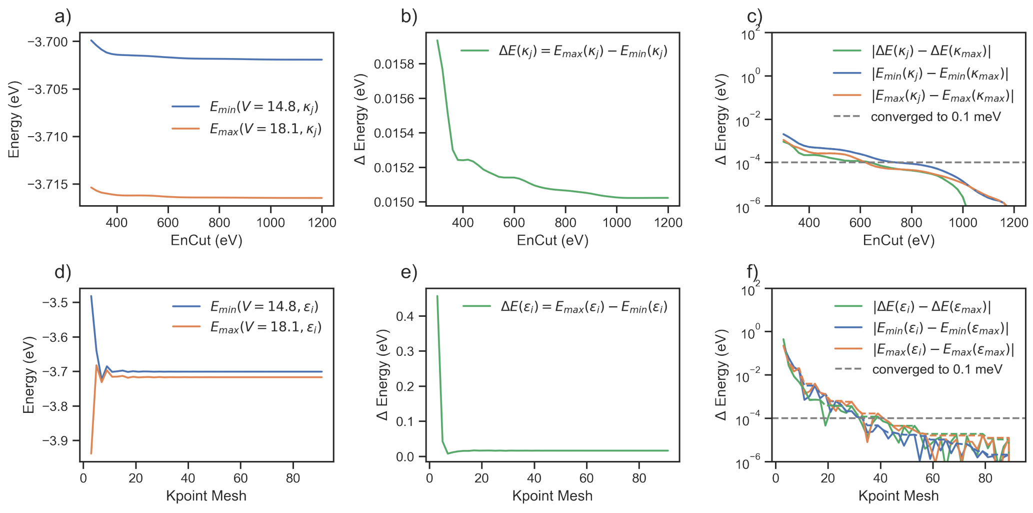


Figure 5.2: Total energy convergence and convergence of energy differences over energy cut-off ϵ_i in the upper row and over **k**-point mesh κ_j in the lower row. The pictures left to right, show the convergence of the total energy for two volumes, the convergence of the energy difference and both combined in an logarithmic plot, by subtracting the fully converged result at maximum convergence parameters.

In contrast to the previous results, which used a high **k**-point mesh when computing the energy cut-off convergence is analysed, here the **k**-point mesh is fixed to the VASP recommendation of $11 \times 11 \times 11$ **k**-point mesh $\kappa_j = 11$ for the energy cut-off convergence. Vice versa, an energy cut-off of $\epsilon_i = 240$ eV is used for the **k**-point mesh convergence. With this their results are comparable with classical convergence tests [23]. The above convergence checks provide two insights:

- On the one hand the semi-logarithmic scale enables a visual quantification of the convergence. It is not a boolean choice of a calculation being converged or not but rather a relative statement, converged up to energy differences below 1 meV in comparison to the maximum convergence parameter (grey line). As a consequence the choice of the maximum convergence parameter impacts the convergence, so it is important to choose a sufficiently high maximum in comparison to the convergence goal.
- On the other hand the top right figure shows the convergence of the total energy and the energy difference being very similar. This is contrast to the the general experience that higher energy plane waves are needed to resolve the energetic states close to the pseudopotential core. Which would result in the total energies converging slower than the energy differences. This is achieved by subtracting the kinetic energy of wave functions in the atomic limit [38], as introduced in Sec. 2.1.4. At the same time this approximation can lead to non-monotonous convergence. While otherwise a higher energy cut-off, results in a higher number of plane waves and a higher number could only result in a lower energy but never a higher energy this is no longer the case with

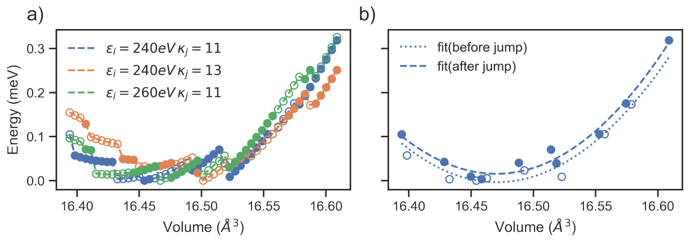


Figure 5.3: Dependence of the plane wave jumps on the energy cut-off and **k**-point mesh. A change in the total number of plane waves is indicated by switching from closed symbols to open symbols. With increasing energy cut-off and increasing **k**-point mesh the frequency of plane wave jumps increases and the amplitude decreases as indicated in a). The resulting difference for the energy-volume curve is illustrated in b) by constructing two energy-volume curves: one based on the points before the jumps and one based on the points after the jumps.

this correction as introduced in Sec 2.1.4.

5.1.3 Plane Wave Jumps

Another aspect that affects the convergence of energy-volume curve calculation and the derived equilibrium properties are the plane-wave jumps in dependence of the volume, as introduced in Sec. 2.1.4. The increase of the volume leads to an increased number of plane waves for the same energy cut-off, as the reciprocal **k**-point mesh decreases in relation to the energy cut-off. Since the **k**-point mesh is discrete the dependence is not continuously, but rather results in jumps. The jump frequency and the amplitude are not systematic. Still, the jump frequency increases with increasing energy cut-off and increasing **k**-point mesh, while the amplitude is decreases. The impact of the plane wave jumps is demonstrated in Fig. 5.3. Following the VASP recommendations first an energy cut-off of $\epsilon_i = 240$ eV and an $11 \times 11 \times 11$ **k**-point mesh $\kappa_j = 11$ is considered. Then, the energy cut-off is once increased to $\epsilon_i = 250$ eV and the **k**-point mesh is once increased to $13 \times 13 \times 13$ ($\kappa_j = 13$). In both cases the frequency of the plane wave jumps increases while the amplitude decreases. To resolve the plane wave jumps a volume range of ± 0.6 % is analysed with 51 energy-volume pairs $E(V_n)$ resulting in an equidistant spacing of 0.004 $\AA^3$.

Beyond the uncertainty of the total energy at a given volume the jumps also affect the equilibrium parameters calculated from such a discontinuous energy-volume curve. When comparing a fit which only includes the points before a jump with a fit, which only includes the points after a jump this becomes visible, like it is shown on the right. While this is the most extreme choice of points, it demonstrates the importance of considering the plane wave jumps. The jumps were regularly considered when the computational resources were more restricted [23], but have been mainly neglected recently. This raises the question if the plane wave jumps impact the DFT precision and how the uncertainty of the rough

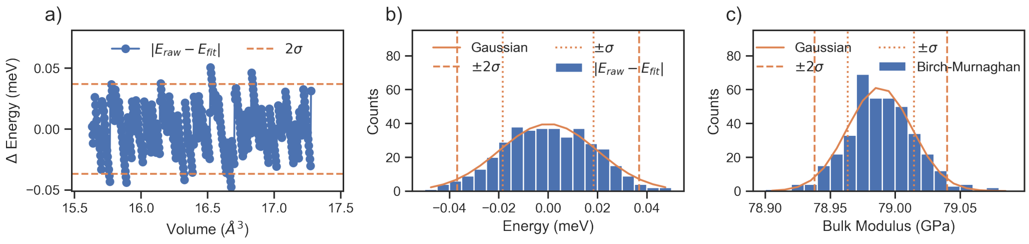


Figure 5.4: Bootstrapping of the statistical error: Based on the energy difference between the fit and the calculated energy-volume curve, the statistical error in the energy is plotted over volume in a) and as histogram in b). Using bootstrapping the error distribution in the bulk modulus is calculated based on the statistical error in the energy in c).

energy surface propagates to the fitting of the equilibrium parameters or other materials properties.

5.2 Equation of State

To quantify the impact of the plane wave jumps on the derived equilibrium properties the volume range is increased to ±5 % maintaining the spacing of 0.004 $\mathring{A}^3$. This is resulting in a total of 385 energy-volume pairs. The energy-volume pairs are fitted with the Birch-Murnaghan EOS, as introduced in Sec. 2.4.1 and the fit energy $E_{\text{fit}}(V_n)$ is subtracted from the energy-volume pairs $E_{\text{calc}}(V_n)$ to separate the plane wave jumps as $\Delta E_{\text{pw-jumps}}(V_n)$:

$$\Delta E_{\text{pw-jumps}}(V_n) = E_{\text{calc}}(V_n) - E_{\text{fit}}(V_n)\,. \tag{5.1}$$

In particular for large numbers of energy-volume pairs $N \to \infty$ the smooth fit function with only four free parameters can be identified as low pass filter, which separates the high frequency plane wave jumps from the underlying energy-volume curve. While the plane wave jumps are deterministic, as illustrated in Fig. 5.3 on the left, the sparse sampling of the energy-volume curve with 21 instead of 385 energy-volume pairs, results in multiple plane wave jumps between two energy-volume pairs, especially for energy cut-offs and **k**-point meshes, above the VASP recommendation. To account for the sparse sampling a statistical approach is chose, which identifies $\Delta E_{\text{pw-jumps}}(V_n)$ as fluctuations or noise. It can be found empirically that this remaining noise follows a normal distribution. This is illustrated in Fig. 5.4: Starting with the difference $\Delta E_{\text{pw-jumps}}(V_n)$ over volume, where the individual plane wave jumps can still be identified, followed by the histogram (center) of $\Delta E_{\text{pw-jumps}}(V_n)$.

5.2.1 Bootstrapping

Based on the empirical finding that the sparse sampling of plane wave jump error follows a normal distribution, it can be identified as a statistical error. Thus, sampling techniques like bootstrapping can be applied to propagate the error in the energy $\Delta E_{\text{pw-jumps}}(V_n)$ to equilibrium parameters and their resulting errors, i.e. errors in the equilibrium energy ΔE_0, volume ΔV_0, bulk modulus ΔB_0 and pressure derivative of the bulk modulus $\Delta B_0'$. Two common bootstrapping methods are the parametric bootstrap and the re-sampling residuals, both can be summarised in three steps:

Calculate residuals: Starting by fitting the model – in this case the energy-volume curve: With the model the energy of a given volume $E_{\text{calc}}(V_n)$ can be predicted as $E_{\text{fit}}(V_n)$ and the residual can be calculated as $\Delta E(V_n)$.

Prediction: The difference in the two approaches is the prediction step. In parametric bootstrapping the residual distribution $\Delta E(V_n)$ is parameterised for efficient re-sampling. So, in the case of the energy differences $\Delta E(V_n)$ the normal distribution is fitted with a Gaussian model and new samples are drawn from the Gaussian distribution. In contrast to this, in the re-sampling residuals the new samples are drawn in a random order from the existing residuals. With the new sample of residuals $\Delta E_{\text{new}}(V_n)$ a synthetic sample of energy-volume pairs $E_{\text{new}}(V_n) = E_{\text{fit}}(V_n) + \Delta E_{\text{new}}(V_n)$ is calculated.

Refit: The synthetic sample is fitted with the same model as the original data set $E_{\text{calc}}(V_n)$. From this fit the equilibrium parameters are calculated, resulting in a second set of synthetic equilibrium parameters.

Finally, the prediction and refit step can be repeated several times to calculate a distribution of the synthetic equilibrium parameters. On the right side of Fig. 5.4 the resulting distribution for the equilibrium modulus B_0 is shown. In this case the number of random samples equals the number of energy-volume pairs N to compare the distribution of the statistical error $\Delta E_{\text{new}}(V_n)$ with the resulting distribution in the bulk modulus ΔB_0. As the number of energy-volume pairs is typically rather limited, the parametric bootstrap is preferred. The advantage of these re-sampling approach is that the fitting of the model which is typically computationally affordable. In contrast the calculation of additional energy-volume pairs requires additional DFT calculations.

5.2.2 Model Error

In addition to the energy fluctuations, caused plane wave jumps, the fitting of the EOS for a given energy-volume curve also depends on the parameterisation of the fit. This includes the number of energy-volume pairs, the volume range and the fitting model used to fit the energy-volume curve. When comparing existing high-throughput projects like the Delta project and the materials project, they use rather different parameters, as introduced in Sec. 5.1.2. The volume range varies from 6 % to 30 % and the number of points vary from seven points to 21 points. Still, both are typically equidistantly spacing. The model range from the Birch-Murnaghan equation, introduced in Sec. 2.4.1, used in the Delta project [34]

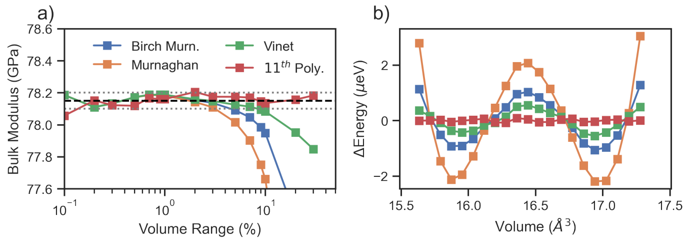


Figure 5.5: Dependence of the bulk modulus B_0 for volume ranges from ± 0.1 % to ± 30 % for three equations of state and a polynomial of degree 11 in a). For a volume range of 5 % the energy difference $\Delta E(V_n)$ is compared for the same equations of state and the polynomial fit in b).

to a total of eight different equations used in the materials project [143]. As a consequence, the volume range, the number of points and the choice of the EOS can be identified as independent parameters for the fitting.

Choice of the EOS: As already discussed in Sec. 2.4.1, those analytical EOS were derived for experimentally easy to measure parameters. In practice, the equilibrium parameters are commonly used. So while the analytical form of the different EOS varies, they all have the same number of fitting parameters and the same polynomial order. This can be confirmed when comparing the volume range dependence for high convergence parameters. Here, an energy cut-off $\epsilon_i = 1040$ eV and a **k**-point mesh $\kappa_j = 91$ are selected to minimise the impact of the plane wave jumps. For volume ranges from ± 0.1 % to ± 30 % the different equations of state, which are introduced in Sec. 2.4.1, are compared in Fig. 5.5. Each energy-volume curve, independent of the volume range, consists of 21 points, so the spacing between the different energy-volume pairs is continuously increasing and the computational cost remains the same. In contrast a constant spacing would result in an increasing computational cost.

From ± 0.1 % to ± 1 % the classical EOS, the Murnaghan equation, the Birch-Murnaghan equation and the Vinet equation, agree while the 11th degree polynomial fit fluctuates with an error of ± 0.05 GPa. Still in contrast to the classical EOS the more flexible polynomial fit remains within the ± 0.05 GPa error bar over all volume ranges above ± 0.1 %. At the same time the classical EOSs starts to diverge at ± 1 %. This results in an error which increases with increasing volume range. This error is an intrinsic bias of the classical EOSs and can be identified as the model error. When comparing the energy difference $\Delta E(V_n)$ for the different EOS at the volume range of ± 5% this effect can be visualised. For the polynomial $\Delta E(V_n)$ is dominated by high frequency fluctuations in the order of ± 0.1 μeV. In contrast, the classical EOSs show low frequency contribution, which can be identified as the 4th order term missing in these EOS. As a consequence, the choice between the different classical EOSs does not matter: All of them are dominated by the same order dependent model error.

A partitioner can always validate their results by comparing the energy difference $\Delta E(V_n)$ to a 4th order polynomial fit. This also explains the recommendation from the VASP man-

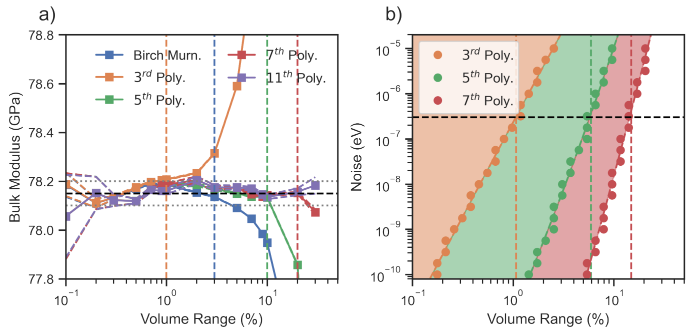


Figure 5.6: Dependence of the volume range used to compute the bulk modulus on the degree of polynomial plotted in a). Based on this dependence the optimal volume range is selected as the volume range when the statistical error equals the systematic error. Finally the uncertainty phase diagram is constructed for different polynomials in b) illustrating the dependence of the optimal volume range for a given polynomial on the standard deviation of the energy noise.

ual to select a volume range between ± 5 % to ± 10 %. Based on the above analysis, for high precision DFT calculation these settings are no longer sufficient when used in combination with classical EOS. It is recommended to use an unbiased polynomial fit rather than the traditional EOS.

5.2.3 Uncertainty Phase Diagram

In the next step the model error is combined with the noise error. While the impact on the equilibrium parameters of a fixed noise error in energy decreases with increasing volume range, the impact of the model error increases. As a consequence for each EOS and a given noise error in the energy there exists an optimal volume range when the statistical noise error and the systematic model error are equal in terms of their contribution to the overall uncertainty to a given equilibrium parameter. This result is summarised on the left side in Fig. 5.6. For polynomials of degree five, seven and eleven and the Birch-Murnaghan equation the calculated bulk modulus B_0 for the same energy-volume curves with varying volume ranges are plotted. As previously, the same high convergence parameters for aluminium are used, compared to Fig. 5.5. Based on these results the bulk modulus B_0 is estimated to be 78.15 ± 0.05 GPa indicated by the black line with the grey line denoting the error bars. Apart from the smallest volume range of ± 0.1 %, the 11$^{\text{th}}$ degree polynomial is always within these error bars (purple line), even when the standard deviation of the statistical noise is larger than the error bar for volume ranges below ± 0.3 %.

In addition, it is worth noting that while the standard deviation for small volume ranges is expected to increase with an increasing order of the polynomial, this is not the case when using the numpy [164] library. The standard deviation increases from the third order polynomial to the fifth order polynomial but remains constant for higher order polynomials.

This is achieved in the numpy library by minimising the Vandermode matrix. This approach is highly beneficial, as long as the number of energy-volume pairs N is sufficiently large and a polynomial of degree $d < \frac{N}{2}$ is selected. For lower order polynomials, the vertical lines denote the transition from the statistical error dominated volume range to the model error dominated volume ranges. For a third order polynomial this is at ± 1 %, for a fifth order polynomial at ± 10 %, and for the seventh order polynomial at ± 20 %. Again these boundaries depend on the noise level. As a consequence, a DFT practitioner can either use a polynomial of degree $d < \frac{N}{2}$ with $N = 21$ points and a volume range of ± 10 %. This settings have been empirically found to be sufficient for the energy noise $\Delta E(V_n)$ related to the plane wave jumps. Alternatively, one may continue to use a traditional EOS and optimise the volume range by comparing the shape of the energy noise $\Delta E(V_n)$, with a higher order polynomial.

To further develop a systematic understanding how the optimal polynomial degree is related to the volume range and the statistical noise of the energy $\Delta E(V_n)$ their dependence for the polynomials with degree three, five and seven is shown on the right side of Fig. 5.6. For these predictions, rather than using DFT calculation, the energy-volume curve at a volume range of ± 30 % is compute with high convergence parameters (ϵ_i, κ_j) once and fitted with a polynomial of degree eleven. Afterwards this fit is used to generate synthetic energy-volume curves at various volume ranges by generating the noise $\Delta E(V_n)$ from normal distributions with varying standard deviations, following the parametric bootstrapping approach introduced in Sec. 5.2.1. For a given noise the volume range dependence is analysed as illustrated in Fig. 5.6 on the left, resulting in one volume range point for each polynomial. At this volume range the statistical noise error is equal to the systematic model error. For high volume ranges the systematic model error is dominant. As a consequence, for larger volume ranges a higher degree of polynomial is required. The transition from the statistical noise error dominated by the volume range can be identified as "phase transition" of the uncertainty and the diagram can be defined as "uncertainty phase diagram". With these "uncertainty phase diagrams" the regions where a given polynomial is dominated by the statistical error – stable region – can be identified.

Finally the prediction of this uncertainty phase diagram can be compared to the previous DFT calculations indicated by the vertical lines. While the agreement is slightly off for the fifth order polynomial the overall predictive capability of this approach is validated. Still, from a practitioner's perspective the optimal choice is a polynomial of degree $d < \frac{N}{2}$ with $N = 21$ for DFT calculation with the convergence parameters higher than the VASP recommendation. The uncertainty phase diagram is only required if either the number of points is restricted, so the polynomial degree cannot be increased or a classical EOS is required. In these cases the volume range has to be adjusted as additional convergence parameter depending on the statistical noise error of the energy, which is again coupled to the other convergence parameters e.g. energy cut-off and **k**-point mesh.

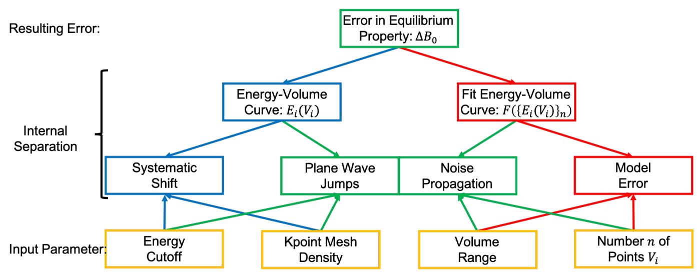


Figure 5.7: Graphical illustration of how the uncertainty in the equilibrium properties is related to the convergence parameters (energy cut-off and **k**-point mesh) and the parameters of the energy-volume curve namely the volume range and the number of energy-volume points. With the plane wave jumps coupling all four parameters it is not possible to separate the convergence when the uncertainty of the plane wave jumps is dominant.

5.3 Reconstruction

The uncertainty in the equilibrium parameters can be attributed to two sources:

- The uncertainty of the fit of the energy-volume curve: It depends on the EOS and its parameterisation in terms of volume range, number of energy-volume pairs N and the order of the EOS, as introduced in Sec. 5.2.
- The uncertainty in the convergence of the energy-volume curve $E(V_i)$: It is typically studied with classical convergence tests and depends on the convergence parameters energy cut-off ϵ_i and **k**-point mesh κ_j, as introduced in Sec. 5.1.

Both of these are coupled by the plane wave jumps. As the plane wave jumps depend on the convergence parameters and at the same time the impact of the resulting statistical noise error $\Delta E_{\text{noise}}(V_n, \epsilon_i, \kappa_j)$ to the equilibrium parameters can be controlled by the choice of the volume range and the number of points N.

This systematic understanding of the different contributions to the uncertainty of the equilibrium parameters is summarised in Fig. 5.7. The systematic model error only depends on the volume range and the number of energy-volume pairs $E(V_i)$. It is independent of the convergence parameters energy cut-off ϵ_i and **k**-point mesh κ_j. At the same time, the systematic energy shift only depends on the energy cut-off ϵ_i and **k**-point mesh κ_j and is independent of the volume range and the number of points. As both the noise propagation of the plane wave jumps and the systematic model error are already combined in the uncertainty phase diagram introduced in Sec. 5.2.3 the next step is to integrate the systematic shift of the energy-volume curve in dependence of the convergence parameters energy cut-off ϵ_i and **k**-point mesh κ_j.

5.3.1 Singular Value Decomposition

To systematically study the dependence of the equilibrium parameters on the convergence parameters $\{E_0(\epsilon_i, \kappa_j), V_0(\epsilon_i, \kappa_j), B_0(\epsilon_i, \kappa_j), B'(\epsilon_i, \kappa_j)\}$, a parameter study is conducted with the pyiron IDE. The **k**-point mesh κ_j is varied in steps of $2 \times 2 \times 2$ starting from $\kappa_{\text{min}} = 3$ to $\kappa_{\text{max}} = 91$. With the calculation of the gamma point – a **k**-point mesh of $1 \times 1 \times 1$ – being excluded because it is impossible in combination with the tetrahedron method and the Blöchl correction in the VASP DFT simulation code. Still, for all other calculations the gamma point is always included as mixing even and uneven **k**-point meshes, which either include the gamma point or not, results in much slower convergence. In addition, the energy cut-off ϵ_i is varied from $\epsilon_{\text{min}} = 200$ eV to $\epsilon_{\text{max}} = 1200$ eV in steps of 20 eV. This results in 2295 combinations. For each combination an energy-volume curve with 21 energy-volume pairs with a volume range of $\pm 10\%$ is calculated resulting in a total of over 48000 calculation for just a single pseudopotential of a single element. Based on this large data set, the dependence of the bulk modulus convergence on the convergence parameters is analysed.

To automatically construct the data set, first 21 equally distant volumes are selected within ± 10 % around the experimental equilibrium volume. These selected volumes are then evaluated at the maximum convergence parameters – energy cut-off $\epsilon_{\text{max}} = 1200$ eV and **k**-point mesh $\kappa_{\text{max}} = 91$. For most cases ± 10 % is sufficient to account for the intrinsic error of the exchange correlation functional. Thus, the equilibrium volume for the given pseudopotential should be included in ± 10 % of the experimental equilibrium volume, otherwise the initial volume range has to be extended. Afterwards the equilibrium volume $V_0(\epsilon_{\text{max}}, \kappa_{\text{max}})$ is calculated by fitting an 11th order polynomial to prevent any bias. Based on the equilibrium volume $V_0(\epsilon_{\text{max}}, \kappa_{\text{max}})$, again 21 equally distant volumes are selected within ± 10 %. This second set of volumes is then evaluated for all convergence parameter combinations. By using the same set of volumes for all convergence parameters, the convergence of individual volumes can be compared.

After the creation of the data set the bulk modulus $B_0(\epsilon_i, \kappa_j)$ is calculated, by fitting an 11th order polynomial to the energy-volume pairs $E(V_n, \epsilon_i, \kappa_j)$ at a given combination of the convergence parameters (ϵ_i, κ_j). The resulting dependence of the bulk modulus $B_0(\epsilon_i, \kappa_j)$ on the convergence parameters in reference to the bulk modulus at maximum convergence parameters $B_0(\epsilon_{\text{max}}, \kappa_{\text{max}})$ is plotted on the right side of Fig. 5.8. The change in colour illustrates the convergence of the bulk modulus as the relative error in comparison to the bulk modulus calculated at maximum convergence parameters $B_0(\epsilon_{max}, \kappa_{max})$. In addition the coloured lines define two sub-samples: The first blue sub-sample starts with the VASP recommended convergence parameters of an energy cut-off of $\epsilon_{\text{VASP}} = 240$eV and a **k**-point mesh of $\kappa_{\text{VASP}} = 11$. This equals more than 1000 **k**-points per atom and includes the gamma point. The second sample in orange is using a **k**-point mesh of $\kappa_{\text{Delta}} = 21$ and an energy cut-off of $\epsilon_{\text{Delta}} = 400$ eV, as recommended by the Delta project [34].

While in general higher convergence parameters improve the convergence of the bulk modulus $B_0(\epsilon_i, \kappa_j)$, the visual comparison on the right side of Fig. 5.8 shows local minima.

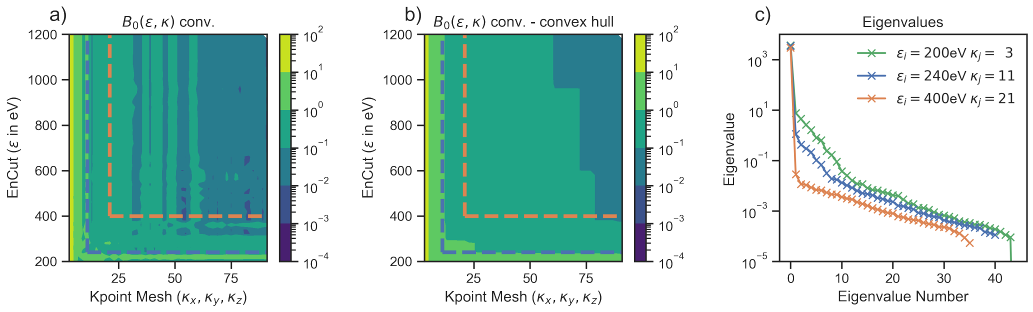


Figure 5.8: Parameter study of the convergence of the bulk modulus in dependence of **k**-point mesh on the x-axis and energy cut-off on the y-axis. The bulk modulus is denoted as the difference to its value at maximum convergence parameters. The calculation results are displayed in a) and a convex hull reconstruction in b). On the calculated bulk modulus values before subtracting the bulk modulus at the maximum convergence parameters or constructing the convex hull is applied and the SVD eigenvalues are plotted in c). The data set is divided in three subsets, starting from the whole data set in green, the data set from the VASP recommended values to the maximum convergence in blue and the data set from the recommendation of the delta project to the maximum convergence in orange.

This indicates that at selected combinations of the convergence parameters, the local bulk modulus agrees well with the bulk modulus at maximum convergence parameters. Still to identify transferable convergence parameters, which also apply to other material properties, monotonous convergence is required. This is achieved by the construction of a convex hull over the bulk modulus convergence $B_0(\epsilon_i, \kappa_j)$. This convex hull is plotted in the center of Fig. 5.8. With the convex hull the convergence parameter for a given convergence goal are indicated by the corresponding contour line.

Beyond this visual inspection of the convergence, singular value decomposition (SVD) is used to decompose the matrix $M_{\epsilon,\kappa}$ in the eigenvectors $u_j(\epsilon_i)$ and $v_i^*(\kappa_j)$ and the matrix of eigenvalues Σ:

$$M_{\epsilon,\kappa} = U_\epsilon \Sigma V_\kappa^* \,. \tag{5.2}$$

Here $M_{\epsilon,\kappa}$ is convergence matrix with the elements $m(\epsilon_i, \kappa_j)$. It is calculated without applying the convex hull or subtracting the final value at the maximum convergence parameters $m(\epsilon_{\max}, \kappa_{\max})$. Each element represents the results of one energy-volume curve fit, e.g. the bulk modulus, with the convergence parameters (ϵ_i, κ_j). For comparison in addition to the whole data set, two sub sets are defined, which both start at higher minimal convergence parameters. One starting at the VASP recommended convergence parameters, indicated as a blue line in Fig. 5.8 and the other at the recommended convergence parameters from the Delta project indicated as the orange line. When comparing the eigenvalues $\sigma_{i,j}$ of the eigenvalue matrix Σ, calculated with the SVD, on a semi-logarithmic scale of eigenvalue over eigenvalue number. The first eigenvalue can be identified as dominant. Already for

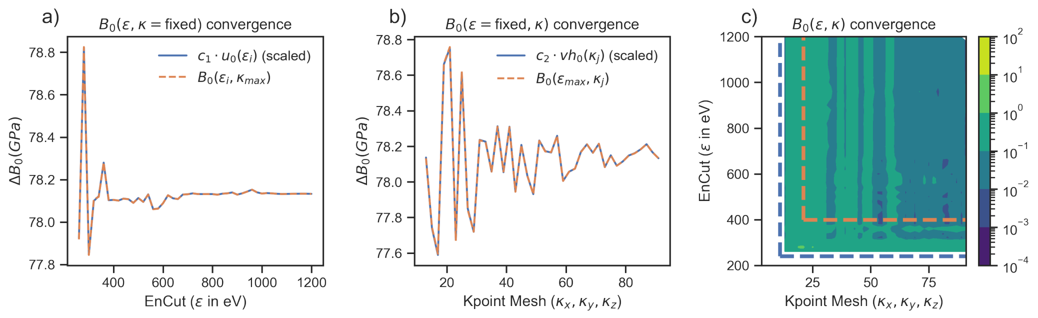


Figure 5.9: Eigenvectors of the SVD in comparison to the convergence at maximum convergence parameters. Starting with the convergence of the bulk modulus over energy cut-off in a) and following by the bulk modulus convergence over **k**-point mesh in b). With the two first eigenvectors and the first eigenvalue the full data set can be approximated, as illustrated by the reconstruction in c).

the full data set, starting at the lowest convergence, the first eigenvalue is more than one order of magnitude larger than the following eigenvalues. This effect becomes even more dominant for the two sub-samples starting at higher convergence parameters (ϵ_i, κ_j). There the difference of the first eigenvalue in comparison to the higher eigenvalues is more than one and a half or even two orders of magnitude.

As a consequence, the first eigenvectors $u_1(\epsilon_i)$ and $v_1^*(\kappa_j)$ seem to contain the most relevant information to approximate the matrix $M_{\epsilon,\kappa}$. The eigenvectors as functions of the convergence parameters (ϵ_i, κ_j) are plotted in Fig. 5.9. On the left side $u_1(\epsilon_i)$ is plotted over the energy cut-off ϵ_i and in the middle $v_1^*(\kappa_j)$ is plotted over the **k**-point mesh κ_j. Finally, the matrix $M_{\epsilon,\kappa}$ can be approximated in first order by:

$$\widetilde{M}_{\epsilon,\kappa} = \begin{pmatrix} u_1(\epsilon_i) \\ 0 \\ \vdots \\ 0 \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & 0 & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \dots & \dots & 0 \end{pmatrix} \begin{pmatrix} v_1(\kappa_j) \\ 0 \\ \vdots \\ 0 \end{pmatrix}^* \approx M_{\epsilon,\kappa} \,. \tag{5.3}$$

The resulting matrix $\widetilde{M}_{\epsilon,\kappa}$ is plotted in Fig. 5.9 on the right. In comparison with Fig. 5.8 on the left it is hard to spot any differences visually. This confirms the dominance of the first eigenvalue. In addition, both eigenvectors can be rescaled to match the convergence of the bulk modulus, with one convergence parameter fixed at the maximum. More explicitly, at the maximum **k**-point mesh κ_{max} the convergence of the bulk modulus only depends on the energy cut-off:

$$\lim_{\epsilon_i \to \epsilon_\text{max}} B_0(\epsilon_i, \kappa_\text{max}) = B_0(\epsilon_\text{max}, \kappa_\text{max}) \,. \tag{5.4}$$

In analogy the **k**-point mesh convergence of the bulk modulus at maximum energy cut-off only depends on the **k**-point mesh:

$$\lim_{\kappa_j \to \kappa_{\max}} B_0(\epsilon_{\max}, \kappa_j) = B_0(\epsilon_{\max}, \kappa_{\max}) \,. \tag{5.5}$$

The dominance of the first eigenvalue in the SVD indicates this separation as:

$$c_{1,\epsilon} \cdot u_1(\epsilon) = \lim_{\epsilon_i \to \epsilon_{\max}} B_0(\epsilon_i, \kappa_{\max}) \;, \tag{5.6}$$

$$c_{1,\kappa} \cdot v_1^*(\kappa) = \lim_{\kappa_j \to \kappa_{\max}} B_0(\epsilon_{\max}, \kappa_j) \,. \tag{5.7}$$

Here the constants $c_{1,\epsilon}, c_{1,\kappa}$ are defined based on the eigenvalue σ_1 as the product $c_{1,\epsilon} \cdot c_{1,\kappa} = \sigma_1$. The above analysis thus shows that the convergence of the energy cut-off ϵ_i and the **k**-point mesh κ_j can be separated once the initial convergence parameters $(\epsilon_{\text{init}}, \kappa_{\text{init}})$ are sufficiently high. The lower the initial convergence parameters, the larger the second and the following eigenvalues in comparison to the first eigenvalue, as illustrated on the right side of Fig. 5.8 and the less accurate is the prediction based only on the first eigenvectors.

5.3.2 Energy Differences

Based on the approximation of the eigenvectors with the bulk modulus at the maximum of one convergence parameter in Eq. 5.6 and 5.7, the total convergence of the bulk modulus can be predicted based on:

$$\Delta_\epsilon B_0(\kappa_j) = |B_0(\epsilon_{\max}, \kappa_j) - B_0(\epsilon_{\max}, \kappa_{\max})| \,, \tag{5.8}$$

$$\Delta_\kappa B_0(\epsilon_i) = |B_0(\epsilon_i, \kappa_{\max}) - B_0(\epsilon_{\max}, \kappa_{\max})| \,, \tag{5.9}$$

$$B_0(\epsilon_i, \kappa_j) \approx B_0(\epsilon_{\max}, \kappa_{\max}) + \Delta_\epsilon B_0(\kappa_j) + \Delta_\kappa B_0(\epsilon_i) \,. \tag{5.10}$$

Still based on the absolute values in the error definition, the error in the bulk modulus is overestimated. At the same time, without the absolute values the error in the bulk modulus is underestimated.

To address this limitation, the differences of the energy volume curves are analysed in the following. Four very special energy-volume curves $E(V_n, \epsilon_i, \kappa_j)$ are considered: $E(V_n, \epsilon_{\min}, \kappa_{\min})$, $E(V_n, \epsilon_{\max}, \kappa_{\min})$, $E(V_n, \epsilon_{\min}, \kappa_{\max})$, $E(V_n, \epsilon_{\max}, \kappa_{\max})$. Each of these energy-volume curves is evaluated at the same set of volumes V_n. This enables the calculation of the energy differences by just subtracting the calculated energies:

$$\Delta_{\epsilon_{\max}} E(V_n, \epsilon_i, \kappa_j) = E(V_n, \epsilon_i, \kappa_j) - E(V_n, \epsilon_{\max}, \kappa_j) \,, \tag{5.11}$$

$$\Delta_{\kappa_{\max}} E(V_n, \epsilon_i, \kappa_j) = E(V_n, \epsilon_i, \kappa_j) - E(V_n, \epsilon_i, \kappa_{\max}) \,. \tag{5.12}$$

The energy differences keep one of the two convergence parameters fixed, while the other one is changed, in analogy to Eq. 5.6 and 5.7: For $\Delta_{\epsilon_{\max}} E(V_n, \epsilon_i, \kappa_j)$ the **k**-point mesh κ_j is fixed while the energy cut-off dependence is analysed by comparing the energy-volume curve at ϵ_i with the energy-volume curve at $\epsilon_{\max}$. In Fig. 5.10 the four energy-volume curves are compared, starting with the energy-volume curves followed by the energy differences

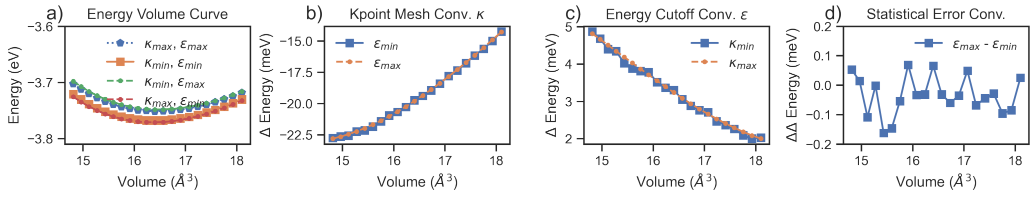


Figure 5.10: Comparison of the energy-volume curves in a), followed by the comparison of the energy differences. In b) the comparison at constant energy cut-off and in c) the comparison at constant **k**-point mesh are plotted. Finally, the difference of the energy differences are compared in d).

$\Delta_{\kappa_{\max}} E(V_n, \epsilon_i, \kappa_j)$ in the second and $\Delta_{\epsilon_{\max}} E(V_n, \epsilon_i, \kappa_j)$ in the third plot. The focus here is the comparison of the energy differences $\Delta_{\epsilon_{\max}} E(V_n, \epsilon_i, \kappa_{\min})$ with $\Delta_{\epsilon_{\max}} E(V_n, \epsilon_i, \kappa_{\min})$ and $\Delta_{\kappa_{\max}} E(V_n, \epsilon_{\min}, \kappa_j)$ with $\Delta_{\kappa_{\max}} E(V_n, \epsilon_{\max}, \kappa_j)$. For aluminium both sets of energy differences are found to agree reasonably well. Further the convergence of the energy-volume curve in dependence of one convergence parameter is independent of the other convergence parameter. This again can be related to just a single dominant eigenvalue in the SVD.

Based on the observed separation of energy cut-off ϵ_i and **k**-point mesh dependence κ_j it is referred to in the following as orthogonal convergence or additive convergence, as both contributions added to calculate the convergence at a given set of convergence parameters (ϵ_i, κ_j). When calculating the difference of the sets of energy differences it can be shown that the second order differences form a closed loop and are equal:

$$\Delta_{\kappa_{\max}} \Delta_{\epsilon_{\max}} E(V_n, \epsilon_i, \kappa_j) = \Delta_{\epsilon_{\max}} E(V_n, \epsilon_i, \kappa_j) - \Delta_{\epsilon_{\max}} E(V_n, \epsilon_i, \kappa_{\max}) \,, \tag{5.13}$$

$$\Delta_{\epsilon_{\max}} \Delta_{\kappa_{\max}} E(V_n, \epsilon_i, \kappa_j) = \Delta_{\kappa_{\max}} E(V_n, \epsilon_i, \kappa_j) - \Delta_{\kappa_{\max}} E(V_n, \epsilon_{\max}, \kappa_j) \,. \tag{5.14}$$

This second order energy difference is plotted in the fourth picture of Fig. 5.10. In contrast to the systematic shift or **systematic error** of the first order energy differences the second order energy differences show a high frequency noise-like error. This high frequency error is related to the plane wave jumps of $E(V_n, \epsilon_{\min}, \kappa_{\min})$. As a consequence, the second order noise like error is again approximated as **statistical error**. By inserting Eq. 5.11 and Eq. 5.12, the equality of Eq. 5.13 and Eq. 5.14 can be validated:

$$\Delta_{\kappa_{\max}} \Delta_{\epsilon_{\max}} E(V_n, \epsilon_i, \kappa_j) = \Delta_{\epsilon_{\max}} \Delta_{\kappa_{\max}} E(V_n, \epsilon_i, \kappa_j) \,. \tag{5.15}$$

This result can again be related to the SVD as the eigenvalues for the second and all following eigenvalues are very close, as illustrated on the right side of Fig. 5.8. Only between the first and the second eigenvalue there is a jump of an order of magnitude. The absence of further strong components indicates an uncorrelated (noisy) data set, once the effect of the first eigenvalue is removed. In the following the uncertainty is separated in a systematic orthogonal contribution Eq. 5.11 and Eq. 5.12 and a remaining statistical contribution Eq. 5.14.

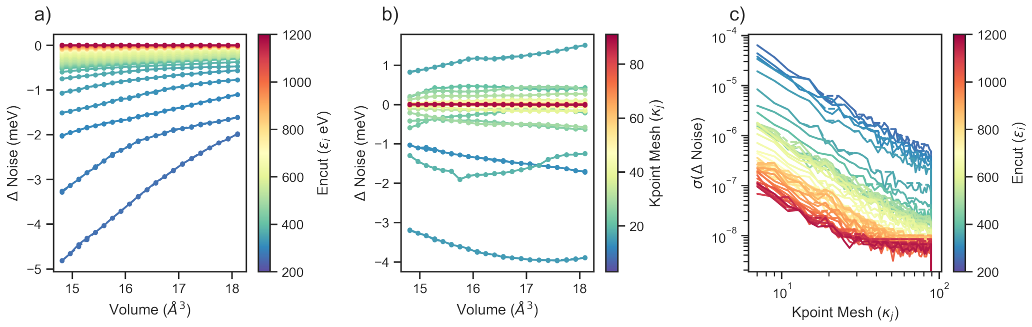


Figure 5.11: Convergence of the energy differences over volume for different energy cut-offs indicated by the different colours in a) and different **k**-point meshes in b). Finally c) shows the convergence of the second order energy difference as standard deviation over **k**-point mesh with the colours indicating the different energy cut-offs.

5.3.3 Convergence of Errors

Based on the separation of the systematic and the statistical convergence, the effect on the resulting energy-volume curve is summarised in Fig. 5.11. Starting with the energy cut-off convergence at the maximum **k**-point mesh $\Delta_{\epsilon_{\max}} E(V_n, \epsilon_i, \kappa_{\max})$ on the left. The energy difference ΔE is plotted over the set of volumes V_n and coloured depending on the energy cut-off ϵ_i. The energy error ΔE continuously decreases with increasing energy cut-off ϵ_i. Still, the vertical shift only affects the equilibrium energy E_0, while the equilibrium volume V_0, equilibrium bulk modulus B_0 and its pressure derivative B' depend on the curvature of the energy-volume curves. Both the energy shift and the correction to the curvature decrease homogeneously with increasing energy cut-off ϵ_i. This is based on the general principle that for higher energy cut-offs more plane waves fit in the same volume, introduced in Sec. 2.1.4.

In contrast to the energy cut-off convergence, the **k**-point mesh convergence is not homogeneous. It is in general recommended to either use even or non-even **k**-point meshes, which either always include the gamma point or never include the gamma point. However, the remaining **k**-points are not fixed with a continuous increase of the **k**-point mesh by $2 \times 2 \times 2$. Thus, a small increase of the **k**-point mesh leads to a different set of **k**-points, which is not a superset of the previous set of **k**-points. This is one of the limitations of the Monkhorst-Pack construction [24] of the **k**-point mesh, introduced in Sec. 2.1.4, which causes the non-monotonous convergence in terms of both the shift of the energy-volume curve as well as its curvature. For the example of aluminium the systematic convergence over energy cut-off ϵ_i and **k**-point mesh κ_j is compared in Fig. 5.11 left and center. The change in terms of the energy shift is in the same order of magnitude. But this depends on the minimal convergence parameters $(\epsilon_{\min}, \kappa_{\min})$. Choosing the **k**-point mesh to start at $\kappa_{\min} = 31$ the systematic shift of the energy cut-off convergence reduces by one order of magnitude.

Finally, the third graph in Fig. 5.11 shows the convergence of the statistical error in dependence of both the **k**-point mesh κ_j on the x-axis and the energy cut-off as colour gradient. To quantify the statistical error the standard deviation of the statistical error is plotted on the y-axis. In the double logarithmic plot the standard deviation scales linearly with the **k**-point mesh κ_j. Thus, a given standard deviation can be achieved by adjusting the **k**-point mesh or the energy cut-off of both. In contrast to the additive behaviour of the systematic convergence the statistical convergence can be expressed as multiplication:

$$\sigma(\epsilon_i, \kappa_j) = \frac{1}{\sigma(\epsilon_{\min}, \kappa_{\min})} \cdot \sigma(\epsilon_i, \kappa_{\min}) \cdot \sigma(\epsilon_{\min}, \kappa_j)\,. \tag{5.16}$$

As a consequence, it is sufficient to once calculate the convergence of the standard deviation dependence on the **k**-point mesh at minimal energy cut-off and once the convergence of the standard deviation dependence on the energy cut-off at minimal **k**-point mesh:

$$\lim_{\epsilon_i \to \epsilon_{\max}} \sigma(\epsilon_i, \kappa_{\min}) = \sigma(\epsilon_{\max}, \kappa_{\min})\,, \tag{5.17}$$

$$\lim_{\kappa_j \to \kappa_{\max}} \sigma(\epsilon_{\min}, \kappa_j) = \sigma(\epsilon_{\min}, \kappa_{\max})\,. \tag{5.18}$$

This dependence is validated empirically as the **k**-point mesh convergence of the statistical error is parallel for the different energy cut-offs.

5.3.4 Coarse-Grained Model

Based on the insights from the systematic convergence in Sec. 5.3.3 in combination with the uncertainty phase diagram in Sec. 5.2.3 a three step algorithm is proposed to construct a coarse-grained model. It predicts energy-volume curves for all convergence parameters (ϵ_i, κ_j) in a computationally highly efficient matter.

Step 1.

As a first step, based on the additive dependence of the systematic convergence error and the multiplicative dependence of the statistical convergence error, the overall convergence matrix $M_{\epsilon,\kappa}$ can be predicted based on a subset of calculations.

- For the systematic convergence the dependence of the energy-volume curve from the energy cut-off ϵ_i is recorded at maximum **k**-point mesh $\kappa_{\max}$ and the **k**-point mesh dependence κ_j is recorded at maximum energy cut-off $\epsilon_{\max}$. When the other convergence parameter is already at the maximum the contribution of the statistical error is minimal.
- For the statistical convergence error a second set of energy cut-off convergence $\epsilon_{\min}$ and **k**-point mesh convergence $\kappa_{\min}$ is recorded to calculate the plane wave jumps in comparison to the convergence at maximum convergence parameters. While in principle it would be possible to use the minimal convergence parameters independent

of the pseudopotential, it is recommended to use the suggestions from the VASP manual, as otherwise the electronic convergence can be limited.

In total only four convergence series have to be calculated resulting in a sparse convergence matrix:

$$\widetilde{M}_{\epsilon,\kappa} = \begin{pmatrix} 0 & \dots & 0 & \epsilon_{\max}, \kappa_{\min} & \dots & \epsilon_{\max}, \kappa_{\max} \\ \vdots & & \vdots & \vdots & 0 & \vdots \\ 0 & \dots & 0 & \epsilon_{\min}, \kappa_{\min} & \dots & \epsilon_{\min}, \kappa_{\max} \\ 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \dots & 0 & 0 & \dots & 0 \end{pmatrix} . \tag{5.19}$$

For a typical convergence analysis this sparse matrix has only $\pm 10\%$ of the entries of the full matrix $\widetilde{M}_{\epsilon,\kappa}$.

Step 2.

The second step is calculating the predicted energy-volume curves at a given set of convergence parameters (ϵ_i, κ_j). Starting with the systematic convergence, the energy-volume curve can be calculated as:

$$E_{\text{sys}}(V_n, \epsilon_i, \kappa_j) = E(V_n, \epsilon_{\max}, \kappa_{\max}) + \Delta_{\text{sys}} E(V_n, \epsilon_i, \kappa_j) , \tag{5.20}$$

$$\Delta_{\text{sys}} E(V_n, \epsilon_i, \kappa_j) = \Delta_{\epsilon_{\max}} E(V_n, \epsilon_i, \kappa_{\max}) + \Delta_{\kappa_{\max}} E(V_n, \epsilon_{\max}, \kappa_j) . \tag{5.21}$$

The above expressions are basically the fully converged energy-volume curve at maximum convergence parameters $E(V_n, \epsilon_{\max}, \kappa_{\max})$ adjusted by the systematic convergence for both energy cut-off and **k**-point mesh. Analogously the energy-volume curve with only the statistical error contribution can be reconstructed:

$$E_{\text{stat}}(V_n, \epsilon_i, \kappa_j) = E(V_n, \epsilon_{\max}, \kappa_{\max}) + \Delta_{\text{stat}} E(V_n, \epsilon_i, \kappa_j) , \tag{5.22}$$

$$\Delta_{\text{stat}} E(V_n, \epsilon_i, \kappa_j) = \Delta E(V_n, \sigma(\epsilon_i, \kappa_j)) . \tag{5.23}$$

Here $E(V_n, \sigma(\epsilon_i, \kappa_j))$ is the mapping of the normal distribution constructed around the mean of 0.0 and the standard deviation $\sigma(\epsilon_i, \kappa_j)$. This iterative approach enables the separate evaluation of the uncertainty resulting from the systematic error and the statistical error. At the same time both contributions can be combined to calculate the total error:

$$E_{\text{tot}}(V_n, \epsilon_i, \kappa_j) = E(V_n, \epsilon_{\max}, \kappa_{\max}) + \Delta_{\text{sys}} E(V_n, \epsilon_i, \kappa_j) + \Delta_{\text{stat}} E(V_n, \epsilon_i, \kappa_j) . \tag{5.24}$$

With this reconstruction the whole convergence matrix can be approximated.

Step 3.

Finally in the third step the polynomial is fitted to the predicted energy-volume curves to predict the bulk modulus convergence over the whole parameter space. Based on the

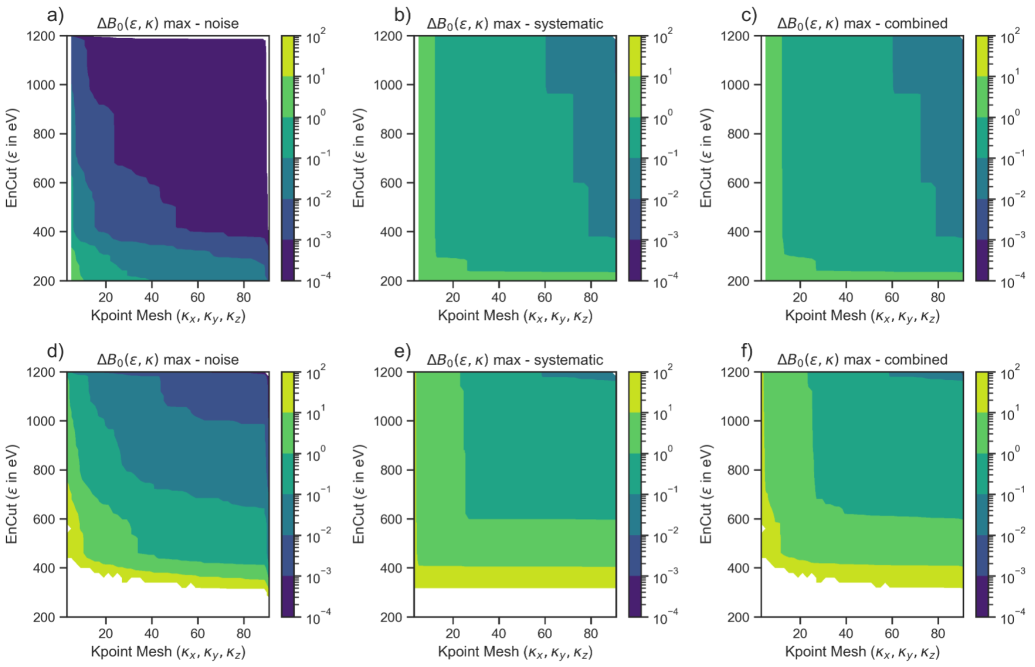


Figure 5.12: Comparison of error contributions for aluminium in the upper row and copper in the lower row. The contributions are divided in the statistical error on the left, the systematic error in the middle and the combination on the right.

experience in the previous section the two error contributions are calculated separately:

$$B_0(\epsilon_i, \kappa_j) = B_0(\epsilon_{\max}, \kappa_{\max}) + \Delta_{\text{sys}} B_0(\epsilon_i, \kappa_j) + \Delta_{\text{stat}} B_0(\epsilon_i, \kappa_j)\,, \tag{5.25}$$

$$\Delta_{\text{sys}} B_0(\epsilon_i, \kappa_j) = B_0(F_{\text{Fit}}(E_{\text{sys}}(V_n, \epsilon_i, \kappa_j))) - B_0(F_{\text{Fit}}(E_{\text{sys}}(V_n, \epsilon_{\max}, \kappa_{\max})))\,, \tag{5.26}$$

$$\Delta_{\text{stat}} B_0(\epsilon_i, \kappa_j) = \sigma(B_0(F_{\text{Fit}}(E_{\text{stat}}(V_n, \epsilon_i, \kappa_j))))\,. \tag{5.27}$$

Here $F_{\text{Fit}}(E(V_n, \epsilon_i, \kappa_j))$ is the polynomial fit of the energy-volume curve $E(V_n, \epsilon_i, \kappa_j)$. From the polynomial fit the bulk modulus $B_0(F_{\text{Fit}}(E(V_n, \epsilon_i, \kappa_j)))$ is calculated and the statistical contribution is approximated using bootstrapping as $\sigma(B_0(F_{\text{Fit}}(E(V_n, \epsilon_i, \kappa_j))))$ the standard deviation from a sample of n calculations of the bulk modulus, sampled from the statistical error in energy predicted for the corresponding convergence parameters.

Comparison of Aluminium and Copper

With these three steps the bulk modulus convergence can be predicted for the full parameter range but requires only a small sub-sample of data points In Fig. 5.12 the different error contributions are compared for aluminium in the top row and copper in the bottom row. For both elements the three errors of the bulk modulus are calculated starting with the statistical error $\Delta_{stat} B_0$, followed by the systematic error $\Delta_{sys} B_0$ and finally the total error combining both. All diagrams use a logarithmic colour scale to quantify the error and the

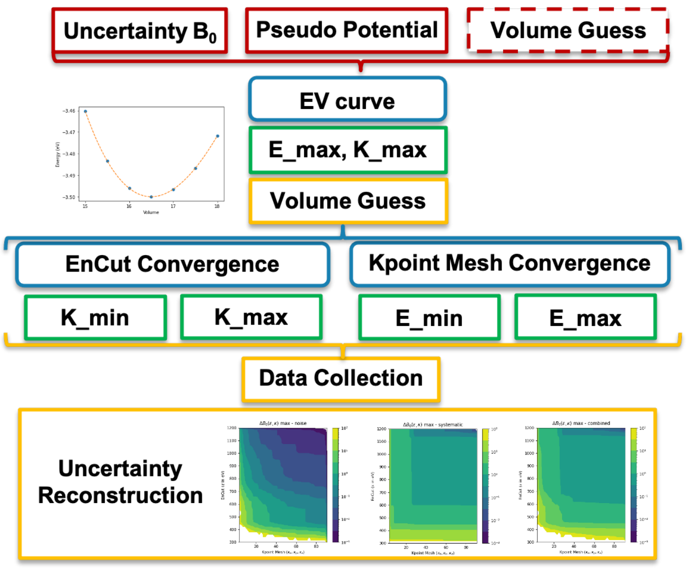


Figure 5.13: Workflow diagram for the uncertainty backwards propagation coarse-grained model. The user inputs are indicated as red boxes, the generic pyiron objects as blue boxes, the calculation with external simulation codes in green boxes and the data analysis steps in yellow boxes.

convex hull reconstruction. The direct comparison of both elements show that the statistical error of aluminium is significantly smaller than the statistical error of copper. This again can be related to the aluminium pseudopotential having less sharp edges than the copper pseudopotential. Thus, less plane waves are required to represent it and the plane waves with higher energies are not required. As a consequence, the plane wave jump amplitude is smaller and the resulting uncertainty of the bulk modulus is smaller as well. The systematic error in the middle of Fig. 5.12 is comparable for both elements with copper requiring higher **k**-point meshes to achieve the same level of convergence. The resulting combined error shows a rather different behavior. For aluminium the statistical error contribution can be neglected, while for copper it plays a dominant role, in particular for calculations up to an energy cut-off of $\epsilon_i = 700$ eV and a **k**-point mesh of $\kappa_j = 41$. Commonly these lower converged calculations are the upper limit for calculating more complex properties like finite temperature properties. This again highlights the importance of separating the statistical and the systematic contribution. Only by combining their behaviours over the entire convergence parameter range of (ϵ_i, κ_j) from a small subset of calculation, it is then possible to suggest the convergence parameters to achieve a user-defined convergence goal. This can be regarded as an uncertainty backwards propagation.

5.4 Simulation Protocol

After the introduction of the uncertainty backwards propagation method for calculating the equilibrium parameters with a plane wave DFT simulation code the next step is to implement and up-scale this method with the pyiron IDE. For this a technical perspective on the method is introduced. While it is in principle possible to run all calculations in serial and create each job object manually, developing a parallel simulation workflow improves the computational efficiency in particular on a HPC cluster. The pyiron IDE supports the user in transitioning from rapid prototyping the simulation protocol to up-scaling it by providing an abstract layer of building blocks: the pyiron objects, as introduced in Sec. 4.2.2. The simulation protocol is implemented based on the workflow diagram in Fig. 5.13. Here, the user input, which indicated in red, consists of the precision goal for one of the equilibrium parameters, the pseudopotential and optionally an initial guess for the equilibrium volume. Based on the uncertainty backwards propagation the precision of all four equilibrium parameters is predicted, so the algorithm is capable of considering multiple precision goals at the same time. For simplicity reasons, only the precision goal for the bulk modulus ΔB_0 is illustrated here. The second mandatory parameter is the pseudopotential. It is based on a specific approximation for the exchange correlation functional and the fixed number of electrons, as introduced in Sec. 2.1.4. Based on the pseudopotential the element, the ground state crystal structure and the lattice constant are selected from experimental references. Alternatively, the user can define these as optional inputs.

5.4.1 Equilibrium Volume Initial Guess

To adjust the initial guess of the lattice constant an initial energy-volume curve is calculated at maximum convergence parameters ($\epsilon_{\max}, \kappa_{\max}$) with a volume range of $\pm 10\%$. The resulting energy-volume is required to parameterise the convergence analysis. As a consequence, the pyiron IDE waits initial energy-volume curve is calculated. In the workflow diagram this is indicated by a blue box, followed by a green and a yellow box. In analogy to the simulation life cycle in Fig 4.1 the red boxes indicate user input, the blue boxes the generic interface of the pyiron objects, the green boxes an external simulation code and finally the yellow box the data analysis. For the calculation of the energy-volume curve it is defined based on the pyiron objects. The calculations are executed with the VASP DFT simulation code. Finally the calculation results are aggregated in a pyiron table, introduced in Sec. 4.3.5, to fit the energy-volume curve and calculate the equilibrium parameters.

5.4.2 Data Aggregation

After calculating the equilibrium volume for the maximum convergence parameters, the convergence step follows a very similar pattern. It is separated in the energy cut-off convergence and the **k**-point mesh convergence, both being executed twice, once at the VASP recommended convergence parameters and once at the maximum convergence parameters.

Following the successful execution, all results are collected in one pyiron table object. The pyiron table object enables different analyses as all energies in dependence of volume, energy cut-off ϵ_i and $\mathbf{k}$-point mesh κ_j are stored in a single table. The workflow diagram indicates that apart from the adjustment of the initial guess for the equilibrium volume the calculations in this workflow are independent from each other and suitable for parallel execution. In addition as only the volume V_n, the energy cut-off ϵ_i and the $\mathbf{k}$-point mesh κ_j change, it is reasonable to define a template job and then only adjust these parameters for the individual calculations. At the same time these three parameters $V_n, \epsilon_i, \kappa_j$, in addition to the total energy $E(V_n, \epsilon_i, \kappa_j)$ define the four mandatory columns in the pyiron table.

5.4.3 Overview

The focus at the current stage is to demonstrate how the graphical representation of the simulation workflow helps to select the required pyiron objects. At the same time, by defining a clear technical implementation, the relevant parameters are visible in the code of the simulation workflow. This is enabled by the separation of the static parameters which remain constant during the workflow and are set once during the definition of the template job and the dynamic parameters which vary between the different calculations and define the columns of the pyiron table, in addition to the calculated total energy. Such an approach is computationally efficient for loosely coupled calculations, as all calculations are submitted to the HPC cluster at once and the HPC job scheduler takes over the distribution of the computational resources.

5.5 Parameter Study

The implementation of the uncertainty backwards propagation as a pyiron IDE simulation protocol enables a wide range of different parameter studies. From the comparison of different approaches to generate $\mathbf{k}$-point meshes, as introduced in Sec. 2.1.4, over the comparison of convergence for different crystal structures to the comparison of different classes of pseudopotentials for the same element to provide a quantification of the computational cost. Still the most relevant question is the dependence of the convergence for a given pseudopotential. Does the convergence depend on the total number of electrons or does the core of the pseudopotential impact the convergence of the equilibrium parameters? To address this question nine fcc elements are analysed.

They are all metals ranging from the alkaline earth metal calcium (Ca), from the s-block, over the transition metals copper (Cu), palladium (Pd), silver (Ag), Iridium (Ir), platinum (Pt) and gold (Au), in the d-block, to the aluminium (Al) and lead (Pb) in the p-block. In Fig. 5.14 the predicted absolute uncertainty in the bulk modulus for the different elements over the convergence parameters (ϵ_i, κ_j) are illustrated as contour plots. The contour lines for the convergence goals of 5 GPa, 1 GPa, 0.5 GPa and 0.1 GPa are additionally marked as red lines with different line styles. These convergence goals are selected based on the reference of 1 GPa used in the delta project to define the DFT precision. On these lines

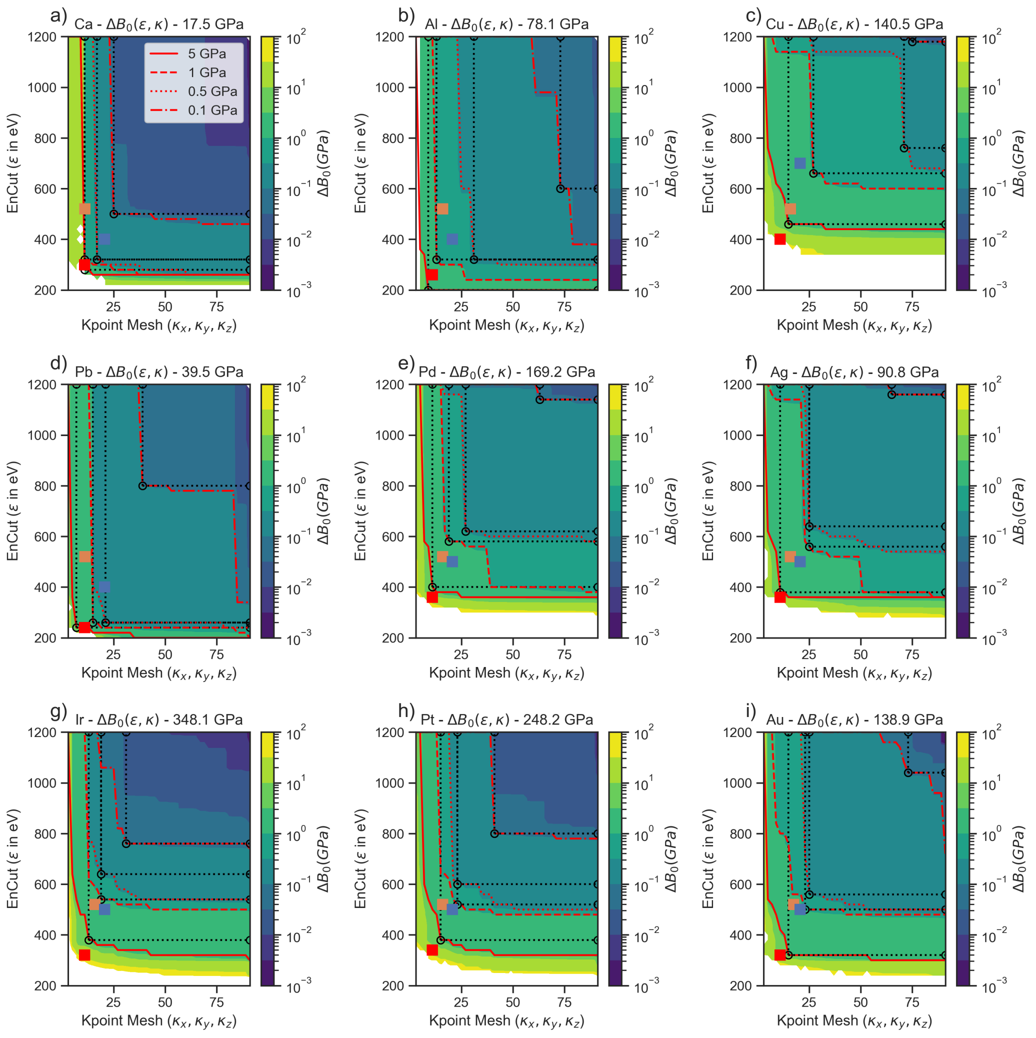


Figure 5.14: Convergence study to predict the uncertainty in the bulk modulus for nine fcc elements using the coarse-grained model. The VASP recommended convergence parameters are marked with a red dot, the recommendation of the delta project with a blue dot and the recommendation of the materials project with an orange dot. The predicted convergence parameters from the algorithm are indicated as open circles on the red lines based on the different convergence goals.

the black open circles indicate the convergence parameters suggested by the uncertainty backwards propagation method. These points are determined by maximising the area of a rectangle between the contour line and the convergence goal. This area is indicated by the black dotted lines. Finally, the coloured dots indicate the recommended convergence parameters by the VASP manual (red), the materials project (orange) and the delta project (blue).

5.5.1 Relative Convergence

As a first step of the analysis the comparison of the convergence of calcium (Ca) and iridium (Ir) demonstrates that the convergence of the bulk modulus precision is independent of the absolute value of the bulk modulus. Calcium (Ca) has the lowest bulk modulus of the selected elements of 17.50 GPa and iridium (Ir) has the highest bulk modulus with 348.08 GPa. Still, while Calcium (Ca) achieves the highest convergence goal of 0.1 GPa with the lowest convergence parameters ($\epsilon_i = 500$ eV, $\kappa_j = 25$) in comparison to the other elements, iridium (Ir) scores second following calcium (Ca) with the convergence parameters ($\epsilon_i = 760$ eV, $\kappa_j = 31$).

In contrast copper with the 4th highest bulk modulus of 140.48 GPa requires the highest convergence parameters ($\epsilon_i = 1180$ eV, $\kappa_j = 75$) to achieve the same convergence. With these parameters being close to the maximum convergence parameters ($\epsilon_i = 1200$ eV, $\kappa_j = 91$) it is not even confirmed if the uncertainty backwards propagation is converged for copper and it would be recommended to further increase the maximum convergence parameters ($\epsilon_{\max}, \kappa_{\max}$) to validate the convergence goal of 0.1 GPa for copper. The same applies for silver (Ag) and palladium (Pd). Still, the results are sufficient to confirm that the convergence of the bulk modulus precision is not related to the absolute value of the bulk modulus. In contrast the DFT error for the equilibrium volume in reference to the experimental equilibrium volume is inversely proportional to the bulk modulus [182].

5.5.2 Transition Metals

As a second step the convergence of the transition metals can be related to the quantum numbers. From copper (Cu) over silver (Ag) to gold (Au) the principle quantum number increases and in particular the energy cut-off required to achieve a given convergence goal decreases. The same applies for the orbital quantum number from iridium (Ir) over platinum (Pt) to gold (Au) the required **k**-point mesh to achieve a given convergence goal increases, in correlation to the orbital quantum number. At the same time the number of semi-core electrons in the selected pseudo potential increases from 17 for iridium (Ir) to 19 gold (Au). However, while the dependence of the uncertainty on the quantum numbers can be approximated for the transition metals, the current data set is not sufficient to explain the convergence for the rest of the periodic table.

5.5.3 Limitations

As a third result of this parameter study the convergence goal of 0.1 GPa can be identified as an upper limit for the convergence uncertainty. While for elements like calcium (Ca) and iridium (Ir) it is possible to reach this limit, for other elements like copper (Cu) it cannot be confirm that this level of convergence is reached within the selected convergence parameter space $(\epsilon_{\max}, \kappa_{\max})$. As a consequence the automated algorithm is limited to minimal convergence goals of 0.1 GPa.

5.5.4 Comparison

The aim of an automated algorithm to predict the uncertainty backwards propagation is to provide the user with recommendations for convergence parameters (ϵ_i, κ_j) which can afterwards be used for further calculations. Thus, an important benchmark for such an algorithm is the comparison to human experts. In Fig. 5.15 the same nine elements are compared once in terms of energy cut-off on the left and once in terms of the **k**-point mesh on the right. In both cases the coloured lines represent the results of the uncertainty backwards propagation and the black lines are reference calculations from other projects. In detail the blue line indicates a convergence goal of 5 GPa, the orange line the 1 GPa convergence goal, the green line the 0.5 GPa convergence goal and the red line the 0.1 GPa convergence goal. For each colour the dotted lines indicate the results from the previous contour lines in Fig. 5.14. In addition to the dotted lines the error bars are calculated as interpolation of a total of 51 convergence goals ranging from 1 GPa to 5 GPa.

For each convergence goal the recommended convergence parameters (ϵ_i, κ_j) are again constructed by maximising the area of the rectangle to the maximum convergence parameters $(\epsilon_{\max}, \kappa_{\max})$. The series of 51 convergence parameters per element are used to smoothen fluctuations in the convergence parameters and are then fitted linearly over the inverse convergence goal. The resulting interpolation is then used to flatten fluctuations between the series of 51 convergence parameters and to construct error bars to fit an interpolation over all elements. This second interpolation is plotted a as solid line in Fig. 5.15. The x-axis in Fig. 5.15 is based on the elements sorted by the VASP recommended energy cut-off (dash-dotted line). When comparing to the delta project (dashed lines) and the materials project (dotted lines), it becomes clear that the delta project reproduces a similar trend like the VASP recommendations, with an on average 200 eV higher energy cut-off. In contrast to the delta project the materials project recommends a constant energy cut-off of 500 eV. This constant energy cut-off is chosen to enable the comparison of different elements at the cost of reaching different levels of convergence for different elements.

The VASP recommendation is close to the convergence goal of ± 5 GPa and the delta project recommendation is close to the ± 1 GPa convergence goal. The same applies for the **k**-point mesh convergence, here both the delta project and the VASP manual recommend a constant number of **k**-points for the selected elements. To allow for a direct comparison, the recommendations of the uncertainty backward propagation is fitted to a constant. In con-

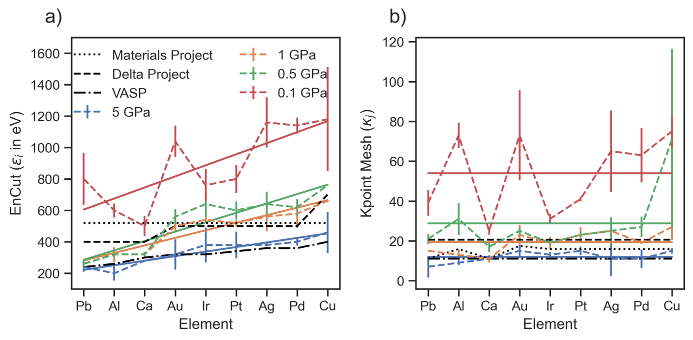


Figure 5.15: Comparison of the convergence limits for the different elements for the energy cut-off on the left and the **k**-point mesh on the right. The convergence limits are coloured based on the uncertainty in the bulk modulus. For comparison the recommendations from the Delta Project and the Materials Project are indicated.

trast to the constant **k**-point mesh, the **k**-point mesh in the materials project is normalised by the equilibrium volume, to maintain a constant **k**-point mesh density. Still, while the constant energy cut-off selected by the materials project is higher then the recommendation from the Delta project (except for copper), the **k**-point mesh recommendation of the materials project is lower then that of the Delta project. In general it mainly follows the recommendation of the VASP manual.

Based on the error bars for both the energy cut-off and the **k**-point mesh, which are calculated by a linear interpolation of 51 individual convergence goal 0.1 GPa can again be identified as an upper boundary. With the error bars reaching beyond ± 200 eV in the energy cut-off and $\pm 20 \times 20 \times 20$ in the **k**-point mesh the computational efficiency is limited. In contrast for the three lower convergence goals 5 GPA, 1 GPa and 0.5 GPa the only calculations with an unproportionally large error bar is copper at 0.5 GPa. This is most likely related to the limited convergence at the maximum convergence parameters ($\epsilon_{\text{max}}, \kappa_{\text{max}}$) in combination with the convex hull reconstruction. At the same time, the large error bar indicates that the interpolation of convergence goals recommends a much lower **k**-point mesh.

In summary, the comparison of the uncertainty backwards propagation with existing high-throughput studies demonstrates that the algorithm achieves the same level of convergence as human experts. The equilibrium parameters calculated as part of the Delta project agree within the predicted error bars with the results of the uncertainty backwards propagation. While the uncertainty backwards propagation requires more computational resources than a human expert, the level of automation enables the integration in high-throughput calculations. Specifically, it allows to systematically asses the contribution of the convergence, the plane wave jumps and the model error to the overall DFT uncertainty. The approach outlined in this chapter is general and can be straightforwardly applied to other DFT properties.

6 Melting Temperature

Following the calculation of the equilibrium parameters with DFT precision in Chap. 5, the next dimenson of complexity to address is the thermodynamic complexity, followed by the chemical complexity in the next chapter. Based on the theoretical overview of *ab initio* thermodynamics in Sec. 2.4, the calculation of the melting temperature is selected as a prototypical example for the thermodynamic complexity, as introduced in Sec. 2.4.4. To address the thermodynamic complexity, the melting temperature is calculated for interatomic potentials using the coexistence method. This is based on the experience that with modern machine learning potentials, as introduced in Sec. 2.2.2, the DFT free energy can be approximated with high precision. The extension of the melting temperature calculated with interatomic potentials to DFT precision, with the TOR-TILD method, is again introduced in Sec. 2.4.4. In addition the calculation are restricted to unaries, to reduce the chemical complexity to a minimum.

Based on the achievements of the previous chapter in addressing the technical complexity, the aim of this chapter is to develop an automated simulation protocol to calculate the melting temperature for an interatomic potential. This simulation protocol is then applied in a parameter study to develop a coarse-grained model for predicting the melting temperature in a computationally efficient way. The pyiron IDE is selected for the development of this simulation protocol, as it supports both the rapid prototyping of simulation protocols as well as the up-scaling for high-throughput parameter studies. Afterwards the simulation protocol can be applied to validate interatomic potentials, by comparing predicted melting temperatures to experiment, as a first step to calculate *ab initio* melting temperatures with the TOR-TILD method, or to identify correlations of the melting temperature with other material properties.

6.1 Experimental Correlation

As a reference to validate the predictions based on interatomic potentials, a literature study is conducted on experimental databases, to identify material properties which are correlated with the melting temperature. The experimental databases range from two general databases, namely the Mendeleev Python package [183] and the Wolfram Alpha knowledge base [184] to specific literature which summarises the thermodynamic results [185]. While more databases are available, the selected ones are accessible within the Python programming language and include the most common material properties. Still, the access to experimental data remains a challenge, which is currently addressed by open data initiatives in materials science e.g. Nomad [141] as introduced in Sec. 3.2.1.

Material Property	Correlation [-1,1]	Nr. of Samples	Source
Cohesive energy	0.95	43	[185]
Vaporization heat	0.94	79	[184]
Evaporation heat	0.93	88	[183]
Boiling Temperature	0.90	95	[183]
Heat of formation	0.87	87	[183]
Proton affinity	0.80	32	[183]
Gas basicity	0.80	32	[183]
Young's modulus	0.75	57	[184]
Bulk modulus	0.72	64	[184]
Shear modulus	0.71	54	[184]
Density	0.67	79	[184]
Liquid density	0.64	64	[184]
Volume	-0.58	58	[184]
Density	0.54	94	[183]
Thermal expansion	-0.53	59	[184]
⋮	⋮	⋮	⋮
Volume · Young's modulus	0.82	48	[184]
Volume · Shear modulus	0.76	48	[184]
Volume · Bulk modulus	0.71	49	[184]

Table 6.1: List of material properties and their correlation with the melting temperature calculated as Pearson correlation coefficients.

The experimental correlations are summarised in Tab. 6.1. For each material property the Pearson correlation coefficient ρ_{MP,T_M} with the melting temperature is calculated using the following formula:

$$\rho_{MP,T_M} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \,. \tag{6.1}$$

In this case x_i is a material property for a given element i, $\bar{x}$ is the mean of the material property over all elements, y_i is the melting temperature for element i and $\bar{y}$ is the mean of the melting temperature. As the Pearson correlation coefficient varies with the number of samples and not all properties are available for all uniaries, the number of samples is added as additional column in Tab. 6.1.

The cohesive energy, the vaporization heat, the evaporation heat and the boiling temperature all have a Pearson correlation coefficient above 0.9. While vaporization heat and the boiling temperature are both related to the transition from the liquid to the gas phase the evaporation heat just like the melting temperature is related to solid liquid phase-transition. As a consequence all three properties require extensive calculations and are less suitable to predict the melting temperature in a computational efficient way. In contrast, the cohesive energy can be calculated by fitting an EOS, as introduced in Sec. 2.4.1 and discussed extensively in the previous chapter. Therefore, the cohesive energy is commonly used to compare *ab initio* calculation with experimental results [186]. For the same reason the cohesive energy is typically included in the fitting process of interatomic potentials and it can be expected that most interatomic potentials are capable of reproducing the cohesive energy.

Other material properties which are computationally affordable and typically included in the fitting process of interatomic potentials are the elastic moduli, namely the Young's modulus E, the bulk modulus B_0 and the shear modulus μ. Each of them is correlated with the melting temperature with a Pearson correlation coefficient of above 0.7. For the shear modulus μ and the Young's modulus E this correlation can be further improved by multiplying with the equilibrium volume V_0, while for the bulk modulus B_0 this effect is negligible.

In the literature for normalising material properties [187, 188] additional correlations can be found, namely:

$$E = \frac{c_E k_B T_M}{V_0}\,, \tag{6.2}$$

$$\mu = \frac{c_\mu k_B T_M}{V_0}\,, \tag{6.3}$$

with the constants $c_E = 100$ and $c_\mu = 44$, the Boltzmann constant k_B and the melting temperature T_M. Still the limitation of this approach is the assumption of a constant Poisson's ratio ν:

$$\nu = \frac{E}{2\mu} - 1 = \frac{c_E}{2c_\mu} - 1 = \frac{3}{22} = 0.1\overline{36}\,. \tag{6.4}$$

This is not the case. Therefore these correlations have to be treated carefully. In particular as the average Poisson's ratio for the data set above is $\bar{\nu} = 0.29 \pm 0.07$. To improve the approximation the constants c_E and c_μ can be calculated directly from the data set by fitting Eq. 6.2 and 6.3 to the data set. The fitted constants are $c_E = 90.6$ and $c_\mu = 36.2$ which then results in a Poisson's ratio of $\nu = 0.25$. This Poisson's ratio agrees within the error bars with the calculated mean.

For the validation of interatomic potentials the cohesive energy and the product of bulk modulus B_0 with the equilibrium volume V_0 are selected, as these material properties can be calculated directly from the energy-volume curve by fitting an EOS as introduced in Sec. 2.4.1. In addition the equilibrium parameters are typically included in the fitting process of interatomic potentials, so a high correlation of the calculated equilibrium parameters with the experimental measurements is expected.

6.2 Simulation Protocol

Following the discourse of analysing experimental correlation of material properties with the melting temperature, the next step is a parameter study on an database of interatomic potentials to assess the ability of the interatomic potentials to reproduce the correlations found for experiment.

6.2.1 Simulation Protocol

The coexistence method is implemented in the pyiron IDE to address two challenges. On the one hand, the dynamic choice of parameters should be automated to remove the need of human expertise and enable a parameter study over a database of interatomic potentials. On the other hand, the simulation protocol should be implemented independently of the simulation code, so it is transferable and can be combined with other methods in the future. In analogy to the simulation protocol to predict the DFT uncertainties in Fig. 5.13, the simulation protocol to calculate the melting temperature is illustrated in Fig. 6.1. Again the red boxes indicate the user input, the green boxes the calculation, the blue boxes the generic pyiron objects and the yellow boxes the data analysis, following the simulation life cycle introduced in Fig. 4.1. In comparison to the DFT uncertainty simulation protocol, which consists of a linear order of steps, the simulation protocol for the coexistence approach consists of three feedback loops. With the feedback loops the parameters of the simulation protocol are dynamically adjusted and the simualtion protocol is restarted from the beginning.

Step 0. Starting from the interatomic potential and the element as mandatory user inputs, the bulk structure is generated based on the experimental crystal structure and the experimental lattice constant. Based on convergence tests with different types of interatomic potentials a cubic bulk structure with more than 4000 atoms is recommended [189]. In addition, a first guess of the melting temperature for the interatomic potential can accelerate the convergence of the melting temperature calculation. If no suggestion for the melting temperature is provided, the experimental melting temperature for the element is selected as initial guess.

Step 1. As a first calculation, the bulk structure is heated in a NPT ensemble to the suggested melting temperature, resulting in the volume expansion of the simulation cell. Under the assumption that the suggested melting temperature is below the superheating temperature, the structure should remain solid. If this is not the case and the structure becomes liquid, the first feedback loop is executed and the suggested melting temperature is reduced by 100 K. The solid structure and the liquid structure are differentiated using the adaptive common neighbour analysis (CNA) [61]. The analysis is explained in more detail in the next section.

Step 2. After the heating of the solid structure, it is duplicated along the z-axis and the duplicated half of the structure is fixed using selective dynamics. Both steps are executed based on the generic pyiron objects independent of the simulation code. By

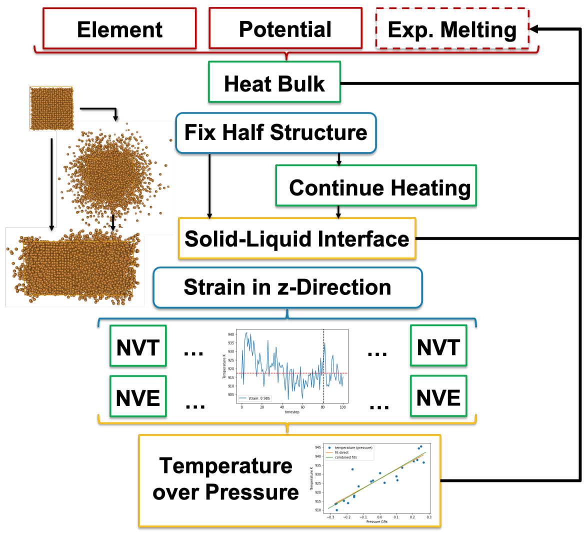


Figure 6.1: Simulation protocol to calculate the melting temperature with the coexistence approach. The user input is indicated by red boxes, the individual calculation by green boxes, the generic pyiron objects by blue boxes and the data analysis by yellow boxes.

fixing half of the structure, the xy-plane is fixed and only the z-axis remains flexible.

Step 3. With half of the structure fixed, the combined structure is heated for 1000 K beyond the suggested melting temperature, again using a NPT ensemble. This results in the melting of the part that is not fixed and in an expansion along the z-axis. Afterwards, the structure is again cooled down to the suggested melting temperature, while the xy-plane remains fixed, resulting in a stable solid-liquid interface structure, which is required for the coexistence approach. Again the NPT ensemble is used to relax the z-axis.

Step 4. After the cooling of the combined structure it is possible that the structure becomes fully solid. This is the case when the suggested melting temperature is below the supercooling temperature, as introduced in Sec. 2.4.4. For this case, the second feedback loop again uses the adaptive CNA analysis to identify the solidified structures and increase the suggested melting temperature by 100 K in case of solidification.

Step 5. With the stabilised solid-liquid interface, the selective dynamics option removed, to again include all atoms in the atomistic simulation. For the next step multiple strains along the z-axis are applied. While this relaxes the z-axis, the xy-plane remains fixed until the guess of the suggested melting temperature is updated.

Step 6. For each of the strained structures the NVT ensemble is applied with the volume defined by the applied strain and the temperature defined by the suggested melting temperature. A compressive strain results in an increase of the internal pressure, while the elongation results in a decrease of the internal pressure.

Step 7. Following the NVT ensemble, the NVE ensemble is applied, while maintaining the velocities of the simulation. By fixing the total energy the temperature is equilibrated in addition to the pressure.

Step 8. In the last and final step, the strained structures are evaluated to check if the solid liquid interface remains stable for the different strains. Afterwards, the temperature is fitted over pressure to predict the melting temperature at a pressure of 0 GPa. Finally, a last feedback loop compares the suggested melting temperature with the predicted melting temperature and repeats the whole simulation protocol if the difference is larger than 1 K. This repetition relaxes the xy-plane, while the loop over strains along the z-axis in the steps 5. to 8. only relaxes the z-axis.

While the above workflow addresses the challenge of creating a stable solid-liquid interface structure by using selective dynamics, the remaining challenge is primarily the last step. To fit the temperature over the pressure dependence, it is necessary to validate at the end of the NVT and NVE calculation that both, the solid and the liquid structure, remain. To address these challenges it is necessary, to quantify the solid-liquid ratio, to identify voids and finally to adjust the simulation parameters dynamically. These challenges are addressed in the following section.

6.2.2 Automation

As the simulation of the liquid phase in a periodic simulation cell results in a rough surface at the edge of the supercell, it is challenging to combine it with the rather sharp surface of the solid structure even at elevated temperatures. By using selective dynamics to construct the solid-liquid interface, the process of combining a solid and a liquid supercell is no longer required. The disadvantage of the selective dynamics is the computational cost of calculating a supercell double the size as both phases are included in one simulation supercell rather than calculating them individually [125, 190]. Still, by keeping the solid structure fixed during the heating of the liquid structure the challenge of combining both after the heating and cooling of the liquid structure in step 3 is removed. Addressing the challenge of combining the solid-liquid interface is the kind of rethinking required to automate existing thermodynamics simulation protocols. Being guided by the experimental setup, of heating one part of the sample and cooling the other part, in addition to identifying selective dynamics as the most extreme form of cooling, physical insights are used to automate the simulation protocol. This is in contrast to the common assumption that automation is only focused on the technical complexity.

Solid-Liquid Detector

The next step is the validation that the interface remains intact during the simulation. This requires the development of a solid-liquid detector to track the interface during the simulation. One option would be to use the Lindemann criterion, introduced in Sec. 2.4.4, to track the mean-square displacement of the individual atoms to separate the ones that

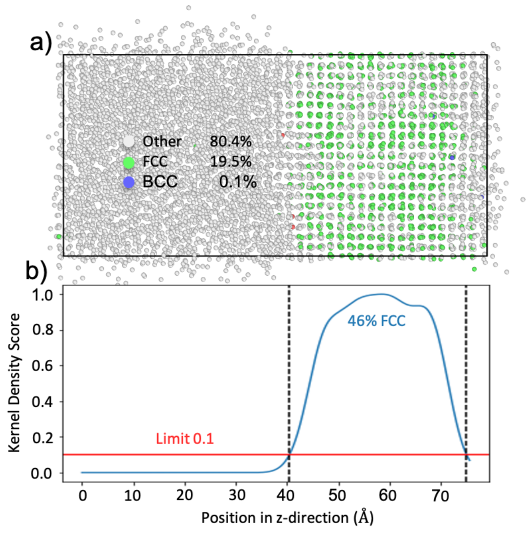


Figure 6.2: Cut along the xz-plane of an aluminium solid-liquid interface structure, with the interface along the xy-plane (top). The right half of the sample is solid, which can be identified based on the crystalline structure with equally spaced columns and rows, while the left half is liquid with no crystallographic structure. Nevertheless, the common neighbour analysis identifies only 19.4% of the atoms as face centered cubic (fcc). The result can be improved by applying a kernel density analysis (bottom), in which 46% can be identified as solid.

fluctuate around their lattice sites from those that move freely in the liquid phase. Still, this would require tracking the position of the atoms over time.

An alternative approach is to only use the final structure of the combined molecular dynamics simulation at the end of step 7 and quantify the solid-liquid ratio for this structure. In Fig. 6.2 on the top such a final structure is analysed with the adaptive CNA detector. Based on the visual inspection the solid structure can be identified as approx. 50 % on the right and the liquid structure as 50 % on the left. This identification is primarily based on the long range order of the atoms in the solid phase being aligned along the positions of the crystal lattice. While the CNA detector is able to identify the face centered cubic (fcc) structure in the area of the solid phase, it predicts a ratio of only 19.5 % solid fcc marked in green, 0.1 % solid basis centered cubic (bcc) marked in blue and the remaining 80.4 % of the atoms are not identified and marked as others in grey. The limitation of the CNA detector is that over 50 % of the solid atoms are not correctly identified as fcc.

To correct this limitation in the detection of the solid phase and based on the physical insight that the solid-liquid interface is oriented orthogonal to the z-axis, the kernel density score is calculated. It interpolates along the xy-plane using Gaussian kernels and reduces the 3-dimensional problem to a 1-dimensional problem of calculating the density of fcc along the z-axis. With a cut-off of the kernel density score at 10 % the solid-liquid interface can be located as illustrated in the lower part of Fig 6.2. In a range from 41 Å to 74 Å the kernel density score is above 10 % resulting in a total of 46 % of the structure predicted to be solid, which agrees well with the visual inspection. In analogy to the creation of the structure, again the physical insight that the solid-liquid interface is orthogonal to the z-axis enables the automation of this previously manual inspection to approximate the solid-liquid ratio. Furthermore, a limit of 25 % is introduced and structures with less than 25 % solid or more than 75 % solid are excluded from the fitting of the temperature over pressure dependence to interpolate the melting temperature. In addition to the adaptive CNA detector a diamond structure detector [191] is used to identify diamond structures in addition to fcc, bcc and hexagonal close packed (hcp).

Void Detector

Another challenge in automating the coexistence approach is the identification of voids forming in the simulation cell. Simulation cells which contain voids no longer follow the pressure strain dependence of structures without voids. Instead of the linear dependence of pressure with strain as illustrated in Fig. 6.3 (top) for an hcp magnesium solid-liquid interface structure with strains ranging from -5 % to 1 %, voids cause fluctuations, which are visible for strains ranging from 1 % to 5 %. Therefore, the structures with voids have to be removed from the analysis, when fitting the temperature-pressure dependence. As mentioned before in previous works [125, 190] the selection of suitable structures was a manual task. To automate the identification of voids at the end of step 7, when the strained structures are equilibrated in a NVE ensemble, the Voronoi analysis is used. It constructs the Wigner-Seitz cell [192] for each atom and calculates the volume of each of these cells. Based on the Voronoi volume of all atoms the maximum is compared with the mean. It is found empirically that void formation is visible when the maximum Voronoi volume is larger than twice the mean Voronoi volume. Based on this the following criterion is defined:

$$\max(\{V_{\text{Voro.}}\}) \leq 2\overline{V_{\text{Voro.}}}\,. \tag{6.5}$$

Here $\{V_{\text{Voro.}}\}$ as the group of all Voronoi volumes and $\overline{V_{\text{Voro.}}}$ as the mean of those. Once this condition is no longer fulfilled, the corresponding calculation is excluded from the fitting of the temperature over pressure interpolation in step 8.

Dynamic Adjustment of Parameters

By separating the convergence along the z-axis in step 5. to 8. and the convergence along the xy-plane in step 1. to 4., the algorithm typically requires multiple iterations. As a consequence, the selection of the control parameters for the equilibration namely the time step length and the total number of time steps have to be adjusted accordingly. In the beginning a larger time step is used in combination with a smaller number of total time steps and a larger strain along the z-axis in step 5. Once the simulation is converged, the time step as well as the strain are reduced to include higher frequencies and measure

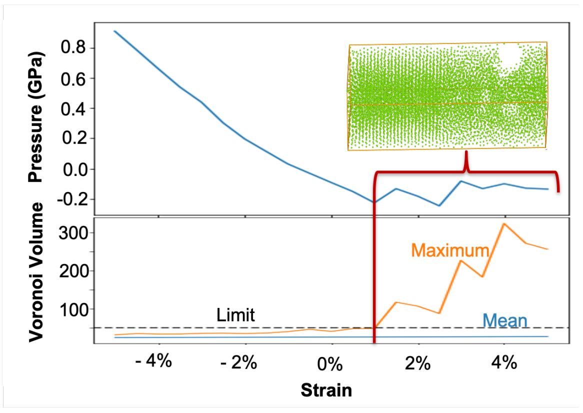


Figure 6.3: Void formation in the liquid phase of the solid-liquid interface structure for hexagonal close packed (hcp) magnesium. The pressure over strain curve(top) shows a linear dependence for strains from -5 % to 1 % and is dominated by fluctuations for strains from 1 % to 5 %. These fluctuations are caused by the formation of voids as illustrated with the inset and they can be identified by measuring the Voronoi volume of the individual atoms and comparing the maximum with the mean (bottom). Once the maximum reaches the limit of twice the mean Voronoi volume, these calculations are removed from the analysis.

more precisely around the melting temperature and the length of the trajectory is extended to improve the convergence. Typical parameters are a time step between 1 fs to 2 fs, a number of total time steps between 20000 to 40000 and a strain from ± 5 % to ± 1 %. The time step and strain are reduced and the number of total time steps is increased when the melting temperature prediction is within the range of different temperatures achieved by a given strain in step 8. Increasing the time step beyond 2 fs also increases the risk of atoms colliding. As a consequence, the maximum time step is material specific.

6.2.3 Implementation

Based on the introduction of the coexistence method in Sec. 6.2.1 and the improvements required for the automation of the simulation protocol introduced in Sec. 6.2.2, the simulation protocol is implemented in the pyiron IDE. While a human expert is able to select the physically “reasonable” structures in step 8 manually, for the automation it is necessary to further define “reasonable” structures. For the interface structures, this includes measuring the solid-liquid ratio as well as identifying voids. Still, these steps are typically not included in scientific publications, which makes it challenging to learn a method based purely on the scientific publication. To address this limitation, it is recommended to publish the simulation protocol as supplementary material, as demonstrated in Sec. 4.4.4.

When comparing the simulation protocol to the calculation of DFT uncertainties in Sec. 5.4.3 there are two major differences: On the one hand the feedback loops in the simu-

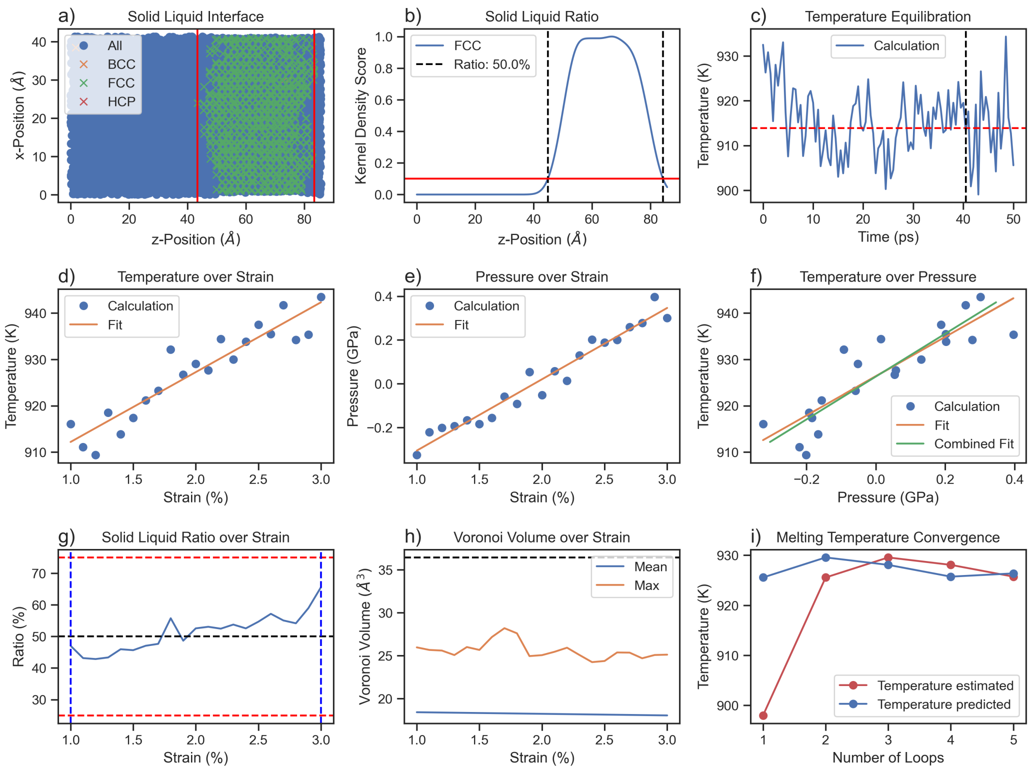


Figure 6.4: Automated analysis during the melting temperature calculation for fcc aluminium: upper row inner loop starting with a) solid-liquid interface structure, b) solid-liquid ratio and c) temperature over time each for a strain of 98.5% and a temperature of 927.75K. Middle and bottom row outer loop: d) temperature over strain, e) pressure over strain and f) temperature over pressure to interpolate the melting temperature at zero pressure. g) solid-liquid ratio over strain to validate both the solid and the liquid phase exist independent of the strain, h) the results of the Voronoi detector for different strains confirming the absence of voids in the structure and i) the convergence of the outer loop of the predicted melting temperature and the calculated melting temperature.

lation protocol, which trigger additional iterations in case the calculation is not converged, and on the other hand the dependence of the simulation on the previous step. For the DFT uncertainty propagation the energy cut-off convergence and the **k**-point mesh convergence both depend on the calculation of the guess for the equilibrium volume at maximum convergence parameters but the rest of the calculation are independent from each other. They can all be executed at the same time, as illustrated in Fig. 5.13. In contrast, in the melting temperature workflow in Fig 6.1 only the calculation for the different strains are independent from each other, while the rest of the calculation depends on the previous steps.

In addition to the interatomic potential calculation even though these are molecular dynamics calculations with several thousand time steps they are still computationally more affordable than the static single atom DFT calculation with high convergence parameters. As a consequence, rather than submitting all calculations to the HPC job scheduler individually the melting temperature simulation protocol for a given set of input parameters is submitted as one pyiron script job. For this the melting temperature simulation protocol is implemented in one Jupyter notebook and the Jupyter notebook is submitted with the pyiron script job object introduced in Sec. 4.4.2 to the HPC job scheduler. In addition the individual molecular dynamics calculation are executed with eight CPU cores using the MPI parallelisation in LAMMPS. But there is no parallelisation for the different strain calculations as this would result in idle computing resources during the other steps, namely steps 1 to 4. The comparison of the two simulation workflows demonstrates that depending on the atomistic engine and as a consequence of the computational cost of an individual calculation different levels of parallelisation are required.

This also explains why frameworks designed for high-throughput DFT calculations are commonly not suitable for molecular dynamics calculations. Worker-based approaches like fireworks for the materials project as introduced in Sec. 3.1.2 are optimised for many calculations with similar computational requirements, but they are less suitable for heterogeneous computational loads which commonly appear in simulation protocols for *ab initio* thermodynamics. In the pyiron IDE this challenge is addressed by supporting the job scheduler of the HPC clusters directly in addition to the ability of submitting simulation protocols as script jobs. Finally, advantage of using script jobs for simulation protocols is the ability to follow the execution of the simulation protocol. This is helpful for debugging in case a calculation failed and at the same time for improving the efficiency of the simulation protocol.

In Fig. 6.4 the different types of graphs generated from the melting temperature simulation protocol are summarised in one figure. It starts with the solid-liquid ratio calculation in the top left, once as two dimensional representation of the xz-plane and next to it as kernel density score over the z-axis. The third plot in the first row shows the equilibration of temperature over simulation time, with the vertical black line indicating the last 20% of the simulation time, which is used to calculate the time averaged temperature indicated by the red line. In the second row the results of step 8 are compared, starting with the temperature over strain on the left, followed by the pressure over strain and finally the temperature over pressure. From the last plot in the second row the melting temperature is predicted for a pressure at 0 GPa by linear interpolation. The prediction of multiple iterations is compared to the suggestion of the corresponding iteration in the last figure in the bottom row. With the suggested temperature in red and the predicted temperature in blue, already after four iterations both agree with a precision below 1 K. For completeness the last two figures in the bottom row show the solid liquid ratio for different strains on the left and the mean Voronoi volume in comparison to the maximum Voronoi volume over strain in the middle. Both confirm that there is no need to exclude any calculation from the given fit.

In summary, with the implementation of the simulation protocol in the pyiron IDE it is not only possible to automate the calculation of the simulation protocol, but at the same time by including the analysis in the simulation protocol with the corresponding explanation, it becomes an instructive resource for others who want to go beyond just to apply the method. This level of documentation requires additional time. Still, it is a valuable extension as it increases the transferability and reproducibility of a given simulation protocol, which is one of the primary challenges in *ab initio* thermodynamics. As a consequence, the pyiron IDE supports the user in all steps of the development process of a simulation protocol, from the rapid prototyping over the up-scaling to the sharing of the final simulation protocol. The specific steps for sharing a simulation protocol with pyiron are introduced in Sec. 4.4.4.

6.3 Parameter Study

With the simulation protocol implemented in the pyiron IDE multiple applications are enabled:

- The melting temperature can be included in the fitting process of interatomic potentials, as introduced in Sec. 2.2. Previously only selected interatomic potentials were published including their melting temperatures. [193, 194].
- The calculation of the melting temperature for interatomic potentials is also an essential step of the TOR-TILD method to calculate the melting temperature with DFT precision, as introduced in Sec. 2.4.4.
- With a data set of melting temperature calculated for interatomic potentials, the experimental correlations introduced in Sec. 6.1 can be analysed in more detail, to develop a systematic understanding of material properties correlated with the melting temperature.

The focus of this thesis is the third aspect, with the aim to use the systematic understanding of the correlations to to develop a coarse-grained model for predicting the melting temperature of interatomic potentials. To construct the required data set the simulation protocol is iterated over the NIST database of interatomic potentials [59]. An alternative option would be the interatomic potential database of the OpenKIM project [58], as the pyiron IDE supports both databases. Both databases for interatomic potentials and their work on

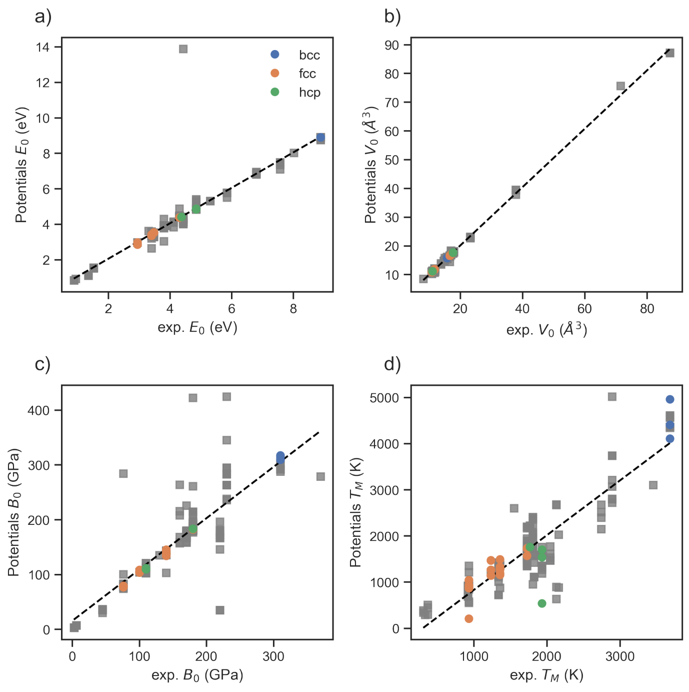


Figure 6.5: Comparison of a) the cohesive energy E_0, b) the equilibrium volume V_0, c) the equilibrium bulk modulus B_0 and d) the melting temperature T_M calculated for multiple interatomic potentials with experimental results. The interatomic potentials which agree with the experimental equilibrium parameters (a-c) are colored by crystalstructure, the rest are marked in grey.

validating interatomic potentials are introduced in Sec. 3.2.2. From the NIST database of interatomic potentials a subset of 200 interatomic potentials is selected. The interatomic potentials are sorted by crystal structure with a focus on bcc, fcc and hcp. As interatomic potentials can implement interaction for more than just a single chemical element, a total of 260 melting temperatures is calculated from the 200 interatomic potentials. With an average of five iterations of the melting temperature simulation protocol outer loop over the relaxation of the xy-plane and over 850.000 molecular dynamics steps in one iteration, this results in more than a billion molecular dynamics steps in this parameter study.

As a first test, the melting temperatures predicted with the coexistence simulation protocol for the different interatomic potentials are compared with the experimental melting temperatures of the corresponding elements. This comparison is shown in the bottom right of Fig. 6.5. While the different interatomic potentials reproduce the general trend with a Pearson correlation coefficient of 0.9063, the deviation can still be larger than ± 1000 K for selected elements. Based on the experimental correlations in Sec. 6.1, the interatomic potentials that are able to reproduce the equilibrium parameters, namely the equilibrium energy E_0, the equilibrium volume V_0 and the equilibrium bulk modulus B_0, should also

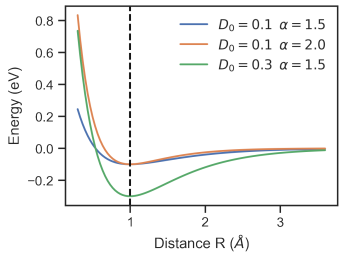


Figure 6.6: Energy over distance r for three Morse pair potentials depending on the potential parameters α and D_0, with a fixed equilibrium distance $R_0 = 1.0$ indicated by the black dashed line.

have a higher correlation with the experimental melting temperatures. To validate this hypothesis the correlations for the equilibrium parameters are also included in Fig. 6.5. The interatomic potentials which reproduce simultaneously the equilibrium energy with an accuracy of $\Delta E_0 = \pm 0.1$ eV, the equilibrium volume with an accuracy of $\Delta V_0 = \pm 0.1$ Å^3 and the bulk modulus with an accuracy of $\Delta B_0 = \pm 10$ GPa are coloured by crystal structure, while the rest are indicated in grey. This selection improves the correlation for the equilibrium parameters, for the equilibrium energy E_0 from 0.9265 to 0.9989, for the equilibrium volume V_0 from 0.9990 to 0.9999 and for the equilibrium bulk modulus B_0 from 0.8585 to 0.9984. Still, the correlation for the melting temperature only improves slightly to 0.9337.

In addition, when analysing the results in Fig. 6.5 in more detail, there are interatomic potentials which reproduce the melting temperature with good agreement to the experiment, while they fail to reproduce the equilibrium parameters within the selected parameter ranges. At the same time there are interatomic potentials which reproduce all equilibrium parameters within the selected parameter ranges but still result in an error in the melting temperature of over ± 1000 K. Consequently, knowing the accuracy in determining the equilibrium parameters is not sufficient to predict the accuracy of the melting temperature calculation. This highlights the importance of including properties like the melting temperature in the fitting process of interatomic potentials to achieve transferability over the whole temperature range. Fitting an interatomic potential only to the equilibrium parameters is not sufficient to reproduce the finite temperature properties like the melting temperature.

6.4 Coarse-Grained Models

In addition to improving the computational efficiency in predicting melting temperatures, the development of coarse-grained models to predict the melting temperature also helps to develop a systematic understanding of phase transitions in general. Three different approaches are compared in the following, starting with the prediction of the melting temperature for the Morse pair potential, followed by the prediction based on the temperature

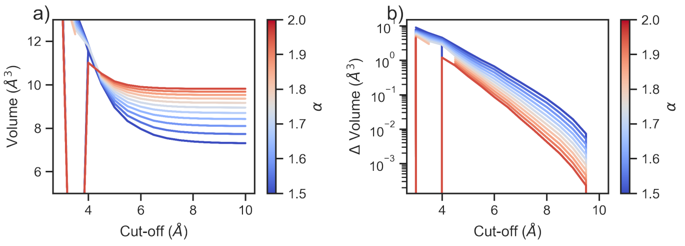


Figure 6.7: Dependence of the equilibrium volume per atom V_0 on the cut-off radius for a Morse potentials that is not restricted to the next nearest neighbors.

dependence of the elastic constants and finally the prediction based on the superheating and supercooling temperature.

6.4.1 Morse Potential

The Morse pair-potential introduced in Sec. 2.2.1 is defined by three potential parameters. Based on the analytical form of the Morse potential the energy contribution $E(R)$ of a neighbouring atom at distance R is defined as:

$$E(R) = D_0 \left[e^{-2\alpha(R-R_0)} - 2e^{-\alpha(R-R_0)} \right] \; ; \text{ with } R > R_{cut} \,, \tag{6.6}$$

with R_0 as the position of the energy minimum of the interaction, R_{cut} as the cut-off radius of the interatomic potential, D_0 as the depth of the potential at the minimum R_0 and α as the width of the potential. The potential is plotted in Fig. 6.6 for three combinations of D_0 and α at the same position of the energy minimum $R_0 = 1.0$ and without any restriction of the cut-off radius R_{cut}. When the α parameter increases, the potential becomes stiffer and as a result the bulk modulus B_0 increases. With this, the potential parameters of the Morse potential are directly correlated to the equilibrium parameters, which have been identified to be correlated to the melting temperature for experimental measurements in Sec. 6.1. Commonly for bulk calculations the Morse potential is cut-off at the nearest neighbour distance. Still, for calculating the melting temperature, such a sharp cut-off radius at the the nearest neighbour distance of the solid phase is not sufficient for calculating the liquid phase. As a consequence, the interactions cannot be fixed to only the nearest neighbours. By extending the cut-off radius beyond the nearest neighbours the equilibrium volume per atom V_0 is no longer only dependent on R_0 but in addition also depends on the cut-off radius R_{cut}.

The impact of the cut-off radius R_{cut} for a fcc crystal with a constant R_0 is illustrated in Fig. 6.7. The equilibrium volume V_0 seems to be converged for cut-off radii larger than 6 Å in Fig. 6.7 on the left. But when subtracting the equilibrium volume V_0 at a cut-off radius of 10 Å from the lower cut-off radii, the convergence becomes continuous on a semi-logarithmic scale in Fig. 6.7 on the right. This effect depends on the stiffness of the Morse potential defined by the α parameter. In Fig. 6.7 the α parameter is indicated by the colour

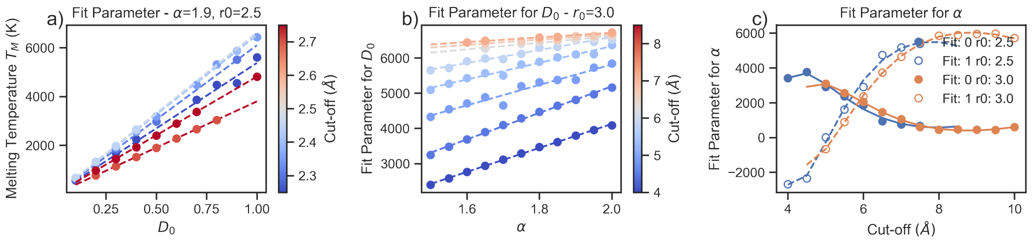


Figure 6.8: Analytical model to predict the melting temperature T_M for a Morse potential based on the parameters D_0, α, R_{cut} and the minimum of the potential R_0.

gradient ranging from $\alpha = 1.5$ in dark blue to $\alpha = 2.0$ in red. The lower the α parameter, the more flexible the potential and as a consequence the more the equilibrium volume V_0 depends on the cut-off radius R_{cut}.

Based on this extension of the Morse pair potential to include higher order nearest neighbours the melting temperature is calculated using the simulation protocol for the coexistence approach developed in Sec. 6.2.3. A total of 1995 melting temperatures are calculated for fcc crystals, with the aim to parameterise an analytical model that predicts the melting temperature T_M based on the parameters of the Morse potential. For fixed positions of the energy minimum at $R_0 = 2.5$ Å and $R_0 = 3.0$ Å, the cut-off radius is varied from a minimum of $R_{cut} = 6.0$ Å to a maximum of $R_{cut} = 10.0$ Å, the potential width is varied from a minimum of $\alpha = 1.5$ to a maximum of $\alpha = 2.0$ and finally the potential depth is varied from a minimum of $D_0 = 0.1$ to a maximum of $D_0 = 0.3$. These parameter ranges are based on the recommendations for cubic metals [47].

The melting temperature is highly correlated with the potential depth D_0, which can be identified as the primary parameter controlling the cohesive energy, as demonstrated in Fig. 6.6. As the experimental analysis in Sec. 6.1 suggests a high correlation of the cohesive energy with the melting temperature, a correlation of the melting temperature calculated for the Morse potential with the potential depth D_0 is expected. A linear correlation is found as illustrated in Fig. 6.8 on the left for $\alpha = 1.9$ and $R_0 = 2.5$ Å without any constant offset. As a consequence, a fictitious Morse potential with a potential depth of $D_0 = 0.0$ has a melting temperature of 0 K or alternatively a Morse potential without an attractive contribution cannot form a solid phase, not even at 0 K. To construct a coarse-grained model, this dependence can be summarised as:

$$T_M(\alpha, D_0, R_0, R_{cut}) = f(\alpha, R_0, R_{cut}) \cdot D_0 \,. \tag{6.7}$$

The melting temperature of a Morse potential $T_M(\alpha, D_0, R_0, R_{cut})$ depends linearly on the potential depth D_0 and a function $f(\alpha, R_0, R_{cut})$, which defines the factor for the linear scaling. In the next step the dependence of $f(\alpha, R_0, R_{cut})$ on the potential width α and the cut-off radius R_{cut} is analysed in Fig. 6.8 in the middle. While $f(\alpha, R_0, R_{cut})$ increases linearly with increasing potential width α, this dependence vanishes for increasing cut-off radii R_{cut}. This empirical result can be interpreted as the melting temperature of a Morse potential with infinite cut-off radius $R_{cut} = \infty$ being independent of the potential width α. Still, for computational efficiency the dependence of potential width α is included as:

$$f(\alpha, R_0, R_{cut}) = g(R_0, R_{cut}) \cdot \alpha + h(R_0, R_{cut}) \,. \tag{6.8}$$

The function $f(\alpha, R_0, R_{cut})$ defines the factor for the linear scaling with the potential depth D_0 as a linear function with the factor $g(R_0, R_{cut})$ and a constant defined by $h(R_0, R_{cut})$. Both functions $g(R_0, R_{cut})$ and $h(R_0, R_{cut})$ can be parameterised as third order polynomials of the cut-off radius R_{cut} with the polynomial parameters depending on R_0 as illustrated in Fig. 6.8 on the right. As a consequence, the melting temperature $T_M(\alpha, D_0, R_0, R_{cut})$ can be calculated as:

$$T_M(\alpha, D_0, R_0, R_{cut}) = [g(R_0, R_{cut}) \cdot \alpha + h(R_0, R_{cut})] \cdot D_0 \,. \tag{6.9}$$

Only the functions $g(R_0, R_{cut})$ and $h(R_0, R_{cut})$ need to be parameterised. While the dependence on the minimum position R_0 remains challenging, this analysis demonstrates that an analytical model to predict the melting temperature $T_M(\alpha, D_0, R_0, R_{cut})$ can be derived based on the linear dependence on the potential depth D_0 and the potential width α at a fixed potential minimum r_0 and with a fixed cut-off radius R_{cut}. These results agree with the experimental results in Sec. 6.1, the melting temperature $T_M(\alpha, D_0, R_0, R_{cut})$ depends primarily on the potential depth D_0 and secondarily on the bulk modulus B_0 which depends on the potential stiffness α. This is an important result as typically pair potentials are neglected for metals, as they fail to reproduce the Cauchy criterion, as introduced in Sec. 2.2.1. Once more this highlights the importance of including the melting temperature in the fitting of interatomic potentials.

6.4.2 Temperature Dependence of the Elastic Constants

Given the success of predicting the melting temperature T_M for the Morse potential in the previous section, the Born Criterion, introduced in Sec. 2.4.4, is used to predict the melting temperature based on the temperature dependence of the elastic constants. This coarse-grained model is computationally efficient, as the temperature dependence of the elastic constants requires only solid bulk calculations. At the same time predicting a first order phase transition based on only a single phase should be impossible, unless the descriptor, in this case the temperature dependence of the elastic constants, includes information of the second phase.

Method

In the literature [52–54, 195] the temperature dependence of the elastic constants is calculated with different thermodynamic ensembles. Four different approaches are defined for comparison. They all start by calculating the thermal volume expansion using a NPT ensemble at the given temperature T and the pressure 0 GPa. Afterwards, the structure is strained along one axis, the structure is equilibrated and the elastic constants are calculated

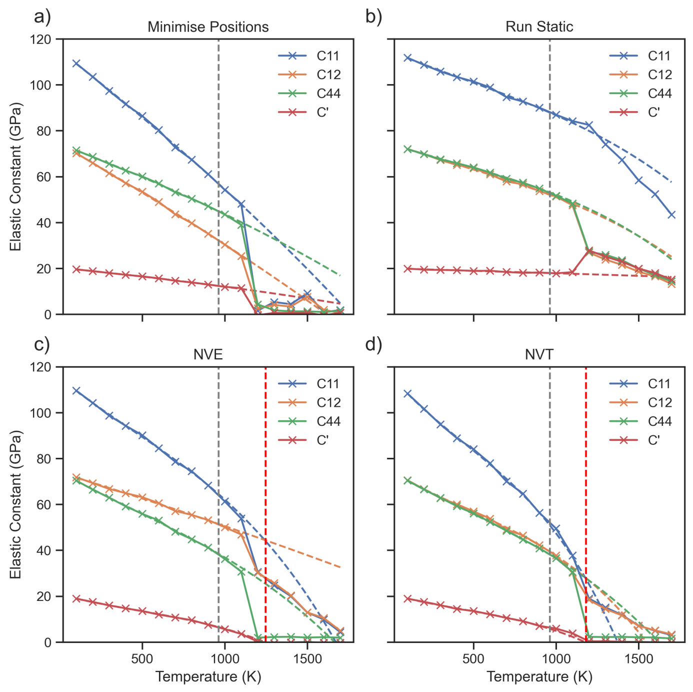


Figure 6.9: Comparison of multiple approaches of calculating the temperature dependence of the elastic constants C_{11} (blue), C_{12} (orange), C_{44} (green) and C' (red), each plotting the elastic constants over temperature for a Morse potential.

based on the change in pressure resulting from the strain after equlibration. The difference in the methods is the equilibration after straining the structure:

(a) At a fixed volume, the positions are minimised to calculate the pressure based on only the displacement of the atoms rather than their thermal fluctuations.

(b) Without additional equilibration the pressure is measured directly after straining the supercell. This equals the immediate response of the thermally expanded crystal.

(c) Using a NVE ensemble the energy after the volume expansion is fixed and the pressure is measured as a response of the strain. As the temperature is not controlled by the ensemble, it increases under compression and decreases under elongation.

(d) Using a NVT ensemble the temperature is fixed in analogy to the experimental setup and the dependence of the energy on an external strain is measured.

To compare the different methods a Morse potential is selected with the potential depth $D_0 = 0.2$ eV, the potential width $\alpha = 1.9$, the potential minimum distance $R_0 = 2.5$ Å and the cut-off radius $R_{cut} = 4.5$ Å. The elastic constants C_{11}, C_{12}, C_{44} and C' are plotted in Fig. 6.9 for each method, with C' being defined as:

$$C' = \frac{1}{2}(C_{11} - C_{12})\,. \tag{6.10}$$

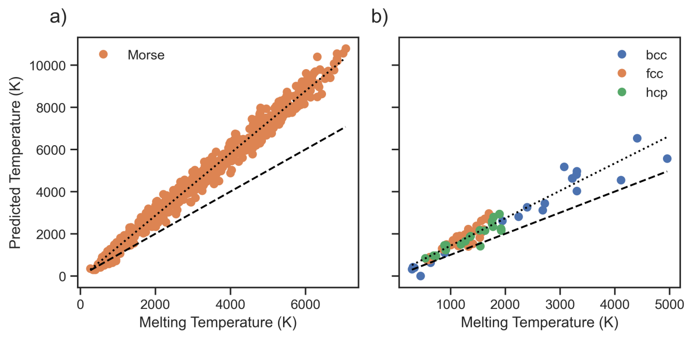


Figure 6.10: From the extrapolation of the temperature dependence of the elastic constants $C_{11}(T)$ and $C_{12}(T)$ below the melting temperature, the superheating can be approximated as the point where $C_{12}(T) > C_{11}(T)$. The resulting superheating temperature is plotted over the melting temperature. Once for the Morse potential in a) and once for potentials from the NIST database in b).

As already introduced in Sec. 2.4.4 the elastic constants remain stable beyond the melting temperature indicated by the grey line and the crystal stability only vanishes at the superheating temperature indicated by the red line. Independent of the ensemble used for equilibration of the strained supercell this confirms results of previous studies [53]. Still the various approaches differ in terms of the predicted temperature dependence of the elastic constants. This also affects the Cauchy relation for pair potentials [48] which is defined as:

$$\frac{C_{12}}{C_{44}} = 1\,, \tag{6.11}$$

with the elastic constants C_{12} being orthogonal to the applied strain and C_{44} being diagonal to the applied strain. This relation is temperature-independent. Thus, the first and the third approach can be neglected, as they do not reproduce the Cauchy relation. In addition, the constant energy in the NVE ensemble in the third approach results in an increase of temperature for the compressed supercell and a decrease of temperature for the strained supercell. This does not represent the experimental bulk conditions as the time scale of thermal equilibration is short in comparison to the straining of the sample in experiments. Finally, as the second approach without equilibration does not fulfill any of the Born criteria, while for the fourth approach with the NVT ensemble the diagonal elastic constant C_{44} vanishes as well as C'. Instead, the second approach only fulfills the combined criteria:

$$C' - C_{44} = 0\,. \tag{6.12}$$

Based on these findings the fourth approach is selected to calculate the temperature dependence of the elastic constants for a series of interatomic potentials to predict the melting temperature T_M.

Morse Pair Potential

Based on the results in Sec. 6.3 the prediction of the melting temperature T_M based on the temperature dependent elastic constants is tested for the Morse pair potential. As explained in Sec. 2.4.4 the Born criterion [119] predicts the superheating temperature T_+ [53]. So starting from 100 K the elastic constants are calculated in steps of 100 K and the temperature dependence of the elastic constants is fitted up to the melting temperature T_M. Based on these fits $C'(T)$ is extrapolated to predict the superheating temperature T_+ as $C'(T_+) = 0$. The prediction for superheating temperature T_+ of the Morse potentials from Sec. 6.3 is plotted in Fig 6.10 a). It is correlated with the melting temperature T_M as indicated by the black dotted line in comparison to the black dashed line for the melting temperature. Still, the error in predicting the superheating temperature T_+ from the extrapolation of the elastic constants is in the order of ± 1000 K. So for predicting the melting temperature T_M of the Morse potential the analytical model developed in Sec. 6.3 is more suitable.

EAM Potentials

The advantage of using the temperature dependence of the elastic constants as the coarse-grained model is that this approach is independent of the type of interatomic potential. In Fig. 6.10 on the right the same interpolation is used to predict the melting temperature for interatomic potentials from the NIST database of interatomic potentials [59]. Again the superheating temperature T_+ is correlated with the melting temperature T_M as indicated by the black lines. This applies to all three crystal structures, bcc indicated in blue, fcc indicated in orange and hcp in green. However, in analogy to the Morse potential the primary limitation of this approach is the prediction of the superheating temperature T_+ based on the elastic constants, which again results in an error in the order of ± 1000 K. The physical explanation for this error is that the temperature dependence of the elastic constants up to the melting temperature T_M contains only limited information about is temperature dependence beyond the melting temperature T_M up to the superheating temperature T_+. In analogy to the phonons introduced in Sec. 2.4.1 the temperature dependence of the elastic constants can be used to measure thermal instabilities [116]. While the extrapolation of the elastic constants over temperature seems to provide a sufficient prediction in the case of Fig. 6.9 this does not seem to be the case for all parameter combinations of the Morse potential a) or interatomic potentials b) in general.

6.4.3 Superheating and Supercooling

The correlation of the superheating temperature T_+ and supercooling temperature T_- with melting temperature T_M is analysed in the following. As illustrated in Fig. 2.5 on the right the superheating temperature T_+ and supercooling temperature T_- are commonly calculated by measuring the volume expansion over temperature. When the solid structure transitions to the liquid structure at the superheating temperature T_+, the volume increases significantly and in analogy the volume decreases when the liquid structure transitions to the solid structure at the supercooling temperature T_-. The advantage of computing the superheating temperature T_+ and supercooling temperature T_- to predict the melting temperature T_M rather than calculating the melting temperature T_M directly is that they do not require the same level of complex equilibration as discussed in Sec. 6.2.2.

Based on the previous results for predicting the melting temperature T_M from the elastic constants for the NIST interatomic potentials database [59] the superheating temperature T_+ and supercooling temperature T_- are calculated for the same interatomic potentials and plotted over the melting temperature T_M in Fig. 6.11 on the left. The superheating temperature T_+ is indicated by open symbols and the supercooling temperature T_- is indicated by filled symbols, with the colour of the symbols again being based on the crystal structure. The superheating temperature T_+ and the supercooling temperature T_- are both fitted linearly over the melting temperature T_M which results in a deviation of ± 100 K for the individual calculation to the fit over all calculations. This error is one order of magnitude smaller than the error of the prediction, based on the temperature dependence of the elastic constants. At the same time the comparison to the melting temperature highlights that the melting temperature T_M is not the arithmetic mean of the superheating temperature T_+ and the supercooling temperature T_- as it is commonly falsely assumed:

$$\tilde{T} = \frac{T_+ + T_-}{2} \,. \tag{6.13}$$

Previous studies for the Sutton Chen many-body interatomic potentials [196] suggest the melting temperature T_M can be predicted as [197, 198]:

$$T_M = T_+ + T_- - \sqrt{T_+ \cdot T_-} \,. \tag{6.14}$$

This construction can be identified as the assumption that the distance of the melting temperature T_M to the arithmetic mean $\tilde{T}$ is equal to the distance of the arithmetic mean $\tilde{T}$ to the geometric mean $\sqrt{T_+ \cdot T_-}$:

$$T_M - \tilde{T} = \tilde{T} - \sqrt{T_+ \cdot T_-} \,. \tag{6.15}$$

This empirical finding is based only on a single type of interatomic potentials and only on fcc crystals structures. In addition discrete temperature steps of $\Delta T = 50$ K are used in contrast to the $\Delta T = 5$ K steps used for the high throughput study in Fig. 6.11 a). For comparison reasons the results for the Sutton-Cheng interatomic potentials [198] are included in Fig. 6.11 as black symbols. When the superheating temperature T_+ and the supercooling temperature T_- are plotted over the melting temperature T_M the difference between the previous study and the current analysis is negligible.

For a more detailed analysis the superheating coefficient Θ^+ is defined as the ratio of superheating temperature T_+ and the melting temperature T_M:

$$\Theta^+ = \frac{T_+}{T_M} - 1 \,. \tag{6.16}$$

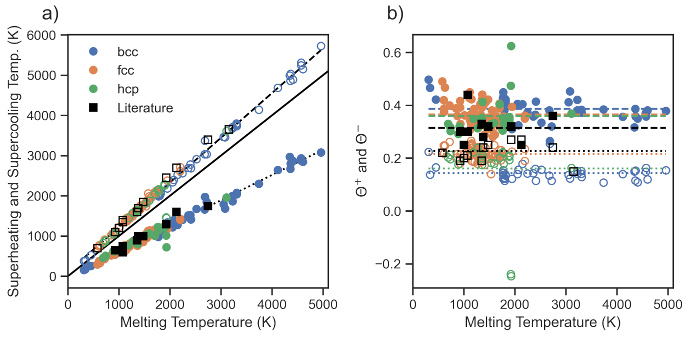


Figure 6.11: Superheating and supercooling temperature over the melting temperature for all interatomic potentials separated by crystal structure in a) in comparison to results from the literature. Superheating and supercooling coefficients over melting temperature, again in comparison to results from the literature in b).

In analogy, the supercooling coefficient Θ^- is defined as the ratio of the supercooling temperature T_+ and the melting temperature T_M:

$$\Theta^- = 1 - \frac{T_-}{T_M}\,. \qquad (6.17)$$

The superheating coefficients Θ^+ and the supercooling coefficients Θ^- are plotted in Fig. 6.11 b). The superheating coefficients Θ^+ are marked as open symbols and the supercooling coefficients Θ^- as filled symbols. The superheating coefficients Θ^+ for fcc agree with the previous calculation, as highlighted by the averaged results illustrated by the dotted black line and the dotted orange line. In contrast to this, the supercooling coefficient Θ^- does not agree. In the previous publication [197] the supercooling temperature T_- is defined as the initial deviation from the liquid phase, while in the current study the solidification is measured. This difference results in lower supercooling temperatures T_- in the current study and as a consequence higher supercooling coefficients Θ^-. When comparing the superheating coefficients Θ^+ for the different crystal structures the superheating coefficient Θ^+ for fcc is found to be notably higher than the superheating coefficient of bcc and hcp. For hcp this is related to the outlier around 2000 K. If this outlier removed the results for fcc and hcp agree well. In contrast, the shift in bcc is systematic. The lower superheating coefficient Θ^+ is caused by a lower superheating temperature T_+ in comparison to the melting temperature T_M. This difference can be related to the lower packing density of the bcc structure compared to fcc and hcp and the effect is independent of the melting temperature. Still, for the supercooling coefficient Θ^- there is no notable difference between the different crystal structures.

In summary, the high-throughput study of the superheating temperature T_+ and the supercooling temperature T_- by calculating the volume expansion over temperature in combination with the high precision melting temperature T_M calculation provides new insights:

- Existing approximations can be analysed systematically to confirm that the arithmetic mean is not sufficient to predict the melting temperature T_M.
- The superheating temperature T_+ for the bcc structure can be identified to be systematically lower than the superheating temperature T_+ for the fcc and hcp structure.
- Based on the simulation protocol to calculate the temperature dependent volume expansion, in combination with the results from the high-throughput study for the precise melting temperature T_M calculation, it is now possible to efficiently predict the melting temperature T_M with an uncertainty of ± 100 K.

These developments are once more enabled by the pyiron IDE. With the pyiron IDE, it is possible to rapidly prototype new simulation protocols and encourage researchers to compare existing methods. Afterwards, these simulation protocols are used for parameter studies to identify general correlations and finally use the resulting data sets to parameterise coarse-grained models. These coarse-grained models provide new insights and enable computationally efficient calculations, which are required for *ab initio* thermodynamics.

7 Phase Diagram

The uncertainty propagation for plane wave DFT calculations in Chap. 5 and the calculation of melting temperatures in Chap. 6 both highlight the versatility of the pyiron IDE. As an outlook, the pyiron IDE is applied to calculate a temperature-concentration phase diagram, which demonstrates the chemical complexity, as the third dimension of complexity of *ab initio* thermodynamics. For computational efficiency an interatomic potential simulation code is selected for the calculation of the phase diagram, still the simulation protocol can be extended to DFT later on. As the number of interatomic potentials, which are published with their corresponding phase diagrams [55, 56, 106], is limited, the nickel-chromium interatomic potential [56] is selected due to the simplicity of the phase diagram and the competition of the bcc and the fcc phases for the different endmembers.

As introduced in Sec. 2.4.3, the phase diagram is computed based on the competition of the different phases in terms of free energy. Based on this introduction, the quasi harmonic approximation is selected. As a consequence, the calculation of the liquid phase is excluded from the phase diagram and only the solid phases are considered. However the simulation protocol is again developed in terms of building blocks, so the approximation for the free energy can be extended at a later stage. As a consequence, the focus of the current simulation protocol is the generation of candidate structures and the comparison of their corresponding free energies. The goal of calculating phase diagrams from *ab initio* is to be able to determine the temperature-composition range for the stability of any particular phase [199, 200] which can lead to the design of new materials [1, 2] and continue to drive innovation.

7.1 Simulation Protocol

In analogy to the simulation protocol to predict the DFT uncertainties in Fig. 5.13 and the simulation protocol to calculate the melting temperature in Fig. 6.1, the calculation of the phase diagram with the quasi-harmonic approximation is illustrated in Fig 7.1. Based on the introduction of the phase diagram calculation in Sec. 2.4.3 the simulation protocol can be divided in three general steps: The generation of atomic candidate structures, the evaluation of a thermodynamic approximation to calculate the free energy and finally an atomistic engine to compute the required energies and forces.

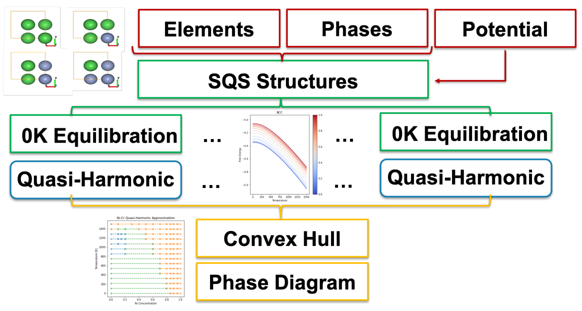


Figure 7.1: Overview of the simulation protocol to calculate phase diagrams with the quasi-harmonic approximation. The user input is indicated in red, the individual calculation in green, the generic building blocks in blue and finally the data analysis steps in yellow.

7.1.1 Free Energy Simulation Protocol

For the current algorithm the candidate phases are required as user inputs marked in red in Fig 7.1. The other user inputs are the chemical elements and the interatomic potential. Based on the number of elements to consider, special quasi random structures (SQS) are generated for the individual phases using a SQS structure generator. For each of the structures the ground state energy E_0 as well as the free energy in the quasi-harmonic approximation F_{qh} are calculated. For the calculation of ground state energy E_0 a simulation code is used so the step is marked in green, while the calculation of the quasi-harmonic approximation requires a separate simulation protocol, which is illustrated in Fig. 7.2, so this step is marked in blue, as it is defined with the simulation code independent generic pyiron objects. Finally, the calculation results are aggregated in the data analysis steps in yellow. Starting with the construction of the convex hull and followed by the plotting of the final phase diagram. Based on the visualisation of the temperature concentration dependent free energies over temperature in a phase diagram, the calculation results can be directly compared to the CALPHAD approach based on experimental databases.

7.1.2 Quasi-Harmonic Approximation

To illustrate the full complexity of the phase diagram workflow, the implementation of the quasi-harmonic simulation protocol is visualised in Fig. 7.2: Starting with the generation of the strained atomistic structures by straining based on the generic pyiron objects and followed by the bulk calculations for each strained structure and afterwards the phonon calculation. As the phonon calculation again requires multiple displacements, especially for SQS structures with a low degree of symmetry, this again results in multiple structures for each strained structure. The creation of these structures is implemented based on the pyiron structure objects and marked in Fig 7.2 in blue. Afterwards, each of these atomistic displacements is then evaluated with a atomistic engine illustrated in green. Finally, the

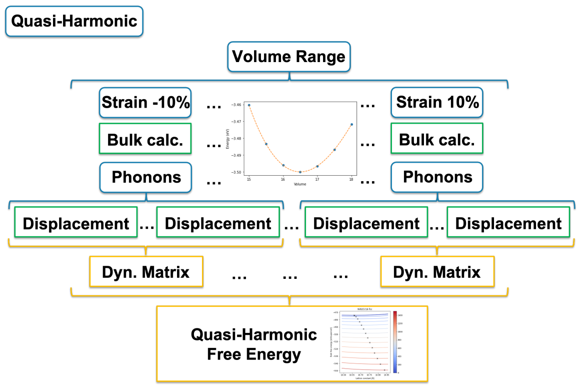


Figure 7.2: Simulation protocol for the quasi-harmonic approximation, starting with the creation of strained structures, followed by bulk calculations and phonon calculations. Both the bulk calculation and the phonon calculation are independent of the atomistic engine. Finally, the data for the individual free energy calculation is combined in the quasi-harmonic prediction of the free energy.

calculation results of each strained structure based on their individual displacements and their individual free energies are combined to calculate the quasi-harmonic free energy, in the analysis steps coloured in yellow.

7.1.3 Implementation

In comparison to the previous simulation protocols, the individual steps of the simulation protocol require a large number of calculations, but are all encapsulated. As a consequence, neither the aggregation of all calculations in a pyiron table, as introduced for the DFT uncertainty propagation workflow in Sec. 5.4, nor the development of the simulation protocol as script job as it is introduced for the calculation of the melting temperature in Sec. 6.2.3 is suitable. Instead, as introduced in more detail in Sup. A.3, a pyiron pipeline enables this level of encapsulation. In combination with the interactive interface of the simulation code, introduced in Sup. A.2, such a pipeline can be implemented in a computationally efficient way.

The individual steps for the phase diagram calculation simulation protocol are explained in the following based on the example of calculating the phase diagram for the nickel chromium interatomic potential [56]:

1. Starting with generating SQS structures for both the bcc phase and the fcc phase for a total of eleven concentrations each, ranging from 10 % Cr up to 90 % Cr and accordingly 90 % Ni to 10 % in steps of 10 %. The bcc structures each contain 108 atoms while the fcc structures contain 128 atoms.
2. Optimising the equilibrium volume V_0 at 0 K and 0 GPa external pressure for each concentration and both phases bcc and fcc. It is important to constrain the shape to

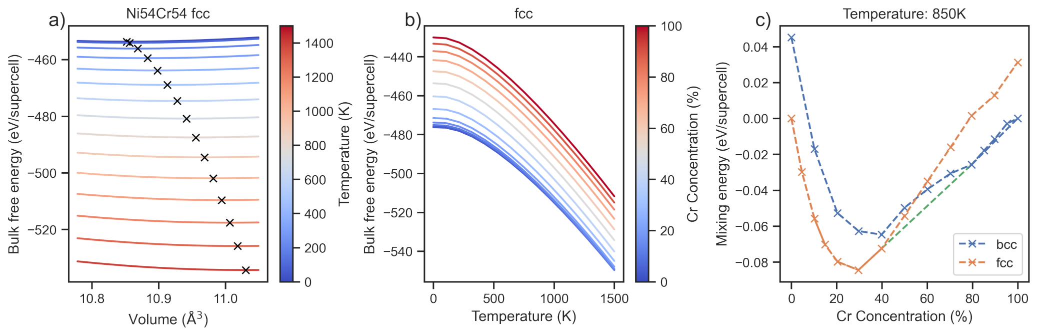


Figure 7.3: In the quasi-harmonic approximation the free energy is calculated in dependence of the volume, the temperature, the phase and the concentration. In a) the volume and temperature dependence of the free energy for the fcc phase are calculated for a fixed concentration of 54 Ni atoms and 54 Cr atoms. For each temperature the optimal volume with the lowest free energy for a given temperature is determined by fitting the bulk, free energy – marked by black crosses. In b), the volume optimised free energy for the fcc phase is plotted for different temperatures and concentrations. Finally, in c) the resulting mixing energy for both phases, the bcc phase, the fcc phase and the resulting convex hull, are plotted in dependence of the concentration for a fixed temperature of 850 K.

remain cubic during the optimisation to prevent a phase transition from bcc to fcc phase even for those concentrations at which the opposing phase is favourable.

3. Calculating the 0 K equilibrium energy E_0 at 21 volumes equally spread around the optimised equilibrium volume V_0 at 0 K and 0 GPa external pressure for each concentration and both phases bcc and fcc.
4. Calculating the phonons using the finite displacement approach implemented in the phonopy software package [136] for each volume for each optimised SQS structure for the given concentrations and both phases. While for the four uniary phases bcc Ni, fcc Ni, bcc Cr and fcc Cr only a single displacement is required, given the high degree of internal symmetry, the SQS structures require many more displacements, resulting in a total of over 150000 individual displacements.
5. Calculating the vibrational contribution E_{vib} of the free energy dependent on the volume, the concentration and the temperature using the harmonic approximation with the force constants determined from the finite displacements in the previous step.
6. Adding the vibrational contribution of the free energy E_{vib} and the equilibrium energy E_0, following the adiabatic appraoch introduced in Sec. 2.4.1. The resulting free energy for a fixed concentration of 50 %/50 % for the fcc phase in dependence of the volume and the temperature is plotted in Fig. 7.3 on the left.

7. Determining the the temperature-dependent equilibrium volume $V_0(T)$ for a given phase and concentration following the quasi-harmonic approximation. The resulting free energy for the temperature-dependent equilibrium volume is plotted for the fcc phase in dependence of the temperature and concentration in Fig. 7.3 in the middle.
8. Adding the configurational entropy contribution to the free energy for the given phase and concentration. Afterwards, to calculate the mixing energy by normalising the free energy with the free energy of the end members.
9. Constructing a convex hull for each temperature to identify the concentrations which lead to solid solutions and those which lead to phase separation. In Fig. 7.3 on the right the resulting convex hull is plotted as dashed green line for a temperature of 800 K in addition with the mixing energy for both the fcc phase in orange and the bcc phase in blue for comparison.
10. Combining the resulting temperature-dependent convex hull reconstructions in a temperature concentration phase diagram. The resulting phase diagram is plotted in Fig. 7.4 on the left and can be compared to the experimental phase diagram in Fig. 7.4 on the right. Only the points on the convex hull are included in the phase diagram.

This implementation of the simulation protocol to calculate the phase diagram with the quasi-harmonic approximation highlights how the technical complexity is addressed in the pyiron IDE by encapsulating the individual steps. On the one hand this enables reusing them in other simulation protocols and on the other hand it allows replacing specific steps in a given simulation protocol to compare the accuracy or efficiency of a given method. For the development of the simulation protocol the LAMMPS simulation code for interatomic potentials is used but the same workflow could be extended to any simulation code which calculates energies and forces.

7.2 Comparison to Experiment

The quasi-harmonic phase diagram, is compared to the phase diagram published with the interatomic potential, to experimental measurements and to a phase diagram calculated with the CALPHAD method based on the experimental measurements in addition to previous CALPHAD assessments. For the specific case of the nickel-chromium phase diagram, experimental measurements and CALPHAD assessments from the literature [201–209] are combined using the ESPEI [210] framework, which is developed based on pycalphad [105]. In Fig. 7.4 the experimental measurements are marked as blue symbols and the CALPHAD prediction is marked as orange dotted line. Based on the experimental measurements in combination with additional experimental measurements for further thermodynamic properties the Redlich-Kister polynomials [96] are fitted and plotted as yellow and blue dotted lines representatively. As a second reference the calculation of the phase boundaries included in the publication [56], which used the variance constrained semi-grand-canonical sampling [100, 101], are added in green. In comparison to the experimental measurements, the solubility limit of bcc in fcc for interatomic potential is extended to higher nickel con-

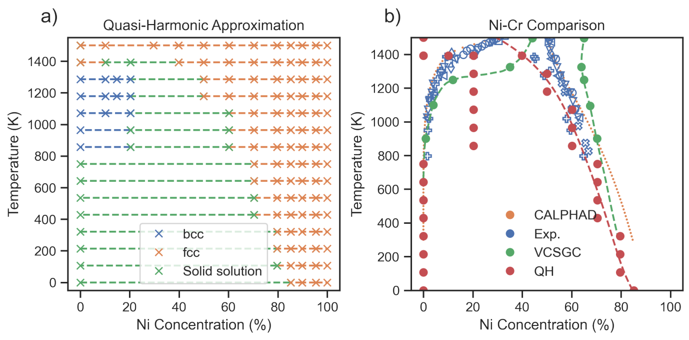


Figure 7.4: Comparison of the phase diagram calculated with the quasi-harmonic approximation with the phase diagram published with the corresponding interatomic potential, the experimental measurements and the phase diagram calculated with the CALPHAD method.

centrations at high temperatures and the solubility limit of fcc in bcc decreases to nickel concentrations of up to 40 % already at 1200 K compared to the 1400 K in experiment. Based on this comparison and the general limitation of quasi-harmonic approximations at high temperatures the quasi-harmonic calculations are limited to a maximum temperature of 1500 K.

The results from the quasi-harmonic approximation are included in Fig. 7.4 as red symbols. The solubility limit of bcc in fcc is predicted to be reached at lower nickel concentrations for the quasi-harmonic approximation in comparison to the variance constrained semi-grand-canonical sampling, it is even reached at lower nickel concentrations than predicted by the CALPHAD model. Still both are the phase boundary predicted from the CALPHAD model and the quasi-harmonic phase boundary are nearly parallel up to 1000 K. Nevertheless, the difference between both theoretical predictions illustrates the importance of a systematic comparison of the different methods. The results for the solubility limit of fcc in bcc highlight the disagreement even more drastic. This is due to the variance constrained semi-grand-canonical sampling method already predicting a lower temperature for the solubility limit of fcc in bcc and the effect is further enhanced by the quasi-harmonic approximation. The quasi-harmonic approximation predicts the solubility limit of fcc in bcc for concentrations up to 20 % nickel concentration for a temperature range from 800 K to 1400 K. In contrast, the variance constrained semi-grand-canonical sampling method predicts the solubility limit of the bcc phase at 1200 K and finally in the experiment the stability beyond 10 % nickel concentration is only found at temperatures above 1400 K.

When comparing the experimental measurements indicated as blue and yellow dots, with the corresponding predictions from the CALPHAD model, then the only visual difference is the prediction of solubility limit of bcc in fcc for temperatures below 1000 K. For the rest the CALPHAD model and the experimental measurements agree within the fluctuation of the experimental measurements. In contrast the quasi-harmonic approximation and the variance constrained semi-grand-canonical sampling method for the same interatomic potential is much larger. This can be related to the intrinsic error of either of the methods or the controllable errors. An indication for an controllable error in the quasi-harmonic approximation is the concentration dependence of the mixing energy for both phases in Fig. 7.3 on the right. This could be related to the supercell size or the displacement distance during the phonon calculation.

As a consequence, it highlights the need for a systematic comparison of various methods applied to existing interatomic potentials which are published with the corresponding phase diagrams. Such a comparison is possible with the pyiron IDE but is beyond the scope of this thesis, as the above example already demonstrates that the pyiron IDE can be applied to address the chemical complexity of calculating phase diagrams.

8 Conclusion

In this thesis, the intrinsic complexity of *ab initio* thermodynamics is addressed to extend the use of simulations for materials design. This complexity is the result of the hierarchical nature of materials and the use of methods developed in different scientific communities.

8.1 Summary

Based on the evaluation of various physical problems, three dimensions of complexity have been identified in this thesis:

Ab initio Uncertainty To predict material properties from plane wave density functional theory (DFT) simulations, the hierarchy of uncertainties has been investigated. For the prototypical calculation of the bulk modulus, the uncertainty has been categorised as intrinsic and as controllable errors. Based on these categories a coarse-grained model has been developed to automatically identify the required convergence parameters to achieve a predefined convergence goal for the total controllable error. With the coarse-grained model the dependence of the controllable error on the plane wave energy cut-off and the **k**-point mesh has been examined: For the selected transition metals and their corresponding pseudo potentials, the required energy cut-off has turned out to depend on the principle quantum number and the required **k**-point mesh has turned out to depend on the orbital quantum number. This is consistent with the physical intuition. In addition, the comparison to convergence tests in the literature has demonstrated the reliability of the coarse-grained model to automate DFT convergence tests, which have been executed manually for over 50 years (Chap. 5).

Thermodynamic Complexity The calculation of thermodynamic properties, in contrast to ground state properties, requires extensive sampling resulting in high computational costs. To highlight how to resolve this complexity a coarse-grained model has been developed for the calculation of the melting temperature for interatomic potentials. It predicts the melting temperature with a precision of ± 100 K based on a linear relation to the superheating and supercooling temperature. Both can be computed efficiently from the hysteresis of the volume expansion over temperature during a heating and cooling cycle. Still, the coarse-grained model has to be parameterised in dependence to the thermodynamic phase, as the ratio of the superheating temperature to the melting temperature for the bcc phase is found to be systematically lower than for the close-packed fcc and hcp phases (Chap. 6).

Chemical Complexity In analogy to the complexity of *ab initio* uncertainties and the complexity to calculate finite temperature properties, the chemical complexity has been addressed by the calculation of a temperature-concentration phase diagram. It illustrates the competition of the stability for different thermodynamic phases at different temperatures and concentrations. The phase diagram for an interatomic potential has been calculated with the quasi-harmonic approximation and compared to other theoretical predictions and experimental models (Chap. 7).

The physical insights of the relation of the DFT convergence parameters to the quantum numbers and the relation of the ratio of superheating temperature to melting temperature to the thermodynamic phase are enabled by the systematic parameter studies. These parameter studies themselves depend on the automation of the simulation protocols, which requires an understanding of the underlying physics to disentangle the intrinsic complexity. Finally, the level of automation in the simulation protocols is achieved by the pyiron integrated development environment (IDE).

From a technical perspective the pyiron IDE implements a class of generic objects which can be combined like building blocks. Each of these pyiron objects is connected to an interactive Jupyter notebook based user interface, a resource interface to communicate with simulation codes as well as high-performance computing (HPC) clusters and a data storage interface optimised for multi-dimensional arrays (Chap. 4). This combination separates the technical complexity of rapid prototyping and up-scaling simulation protocols from the underlying physics and enables new physical insights and the development of coarse-grained models as summarised above.

8.2 Outlook

Based on the success of the pyiron IDE by addressing the technical complexity of *ab initio* thermodynamics and its release as an open-source software, it is applied by a growing number of users from the atomistic community and beyond. Recent studies which use the pyiron IDE are briefly introduced to highlight its versatility.

DFT Method Development From the implementation of generalized dipole corrections [133] over the development of a potentiostat [211, 212] to atomic relaxation around defects in magnetically disordered materials [213] and the electronic structure in unconventional superconductors [214], the pyiron IDE is commonly used to address the technical complexity in DFT method development. This includes the construction of feedback loops to control multiple DFT calculations, parameter studies and the publication of newly developed methods as self-contained simulation protocols.

Interatomic Potentials From the fitting of interatomic potentials [95, 215] over their validation [44] to parameter studies with them [216], the computational efficiency of interatomic potentials enables sampling a large parameter space. As a consequence, the pyiron IDE is applied to address the technical complexity of rapid prototyping and up-scaling simulation protocols.

Beyond atomistic simulations and based on the close collaboration with experimental colleagues at the Max-Planck-Institut für Eisenforschung, the post-processing pipelines for experimental methods, e.g. atom probe tomography, are identified to have similar requirements as the development of simulation protocols [217]. So, with the ability to implement interfaces to additional external software, as introduced in Sup. A.4, first prototypes are currently tested to execute parameter studies for post-processing pipelines of experimental measurements.

Another aspect of the pyiron IDE is the community service. On the one hand, this includes the education and training in virtual workshops [218], in which the participants can use pyiron directly via their web browser. On the other hand, the pyiron IDE joined the NumFocus foundation to collectively promote open-science principles in the materials science community and foster the exchange with software developments in other scientific communities.

In summary, the development of the pyiron IDE has a positive impact for and beyond *ab initio* thermodynamics. By disentangling the technical complexity from the physical complexity, the rapid prototyping and up-scaling of simulation protocols are enabled. This results in new physical insights and the development of coarse-grained models, which cover previously inaccessible parameter spaces.

A Supplement

After the initial release of the pyiron IDE [175] the user feedback led to a constantly growing number of contributors who not only use the pyiron IDe for their research but also continue the development of the pyiron IDE by contributing advanced features for other users. Three of these advanced features, which are developed based on the user feedback, are introduced in the following. Starting with the automation of commonly used calculations by introducing separate job classes for parallel calculation, followed by the introduction of an interactive interface to exchange information between simulation codes during the execution of the simulation code and finally by simplifying the addition of new simulation codes.

A.1 Parallel Job Objects

In the previous simulation protocol, the calculation of the energy-volume curve can be replaced with the corresponding parallel job object. The Murnaghan job object, named after the Murnaghan equation, loops over the different strains and fits the energy-volume curves after the calculation finished successfully. Just like the previous simulation code job objects it is created with the create job function `create.job()` of the project object by selecting the job type `Murnaghan`. Alternatively, the Murnaghan job can be directly created from the reference job by calling the create job method of the reference job. In this case the reference job is automatically assinged.

```
# Loop over different strains
murn = job_ref.create.job.Murnaghan(
    job_name="murnaghan"
)
```

By default the Murnaghan job uses eleven strain rates ranging within ±10% just like the example in Sec. 4.3.4. The users can adjust these by modifying the input object of the Murnaghan job object. For example to reduce the volume range to ±5% the volume range parameter `murn.input["vol_range"]` can be updated and afterwards the input object is displayed to validate the change:

```
murn.input["vol_range"] = 0.05
murn.input
>>> "   Parameter        Value"
>>> "0  num_points       11"
>>> "1  fit_type         polynomial"
>>> "2  fit_order        3"
>>> "3  vol_range         0.05"
```

Besides simplifying the construction of the simulation protocol by reusing existing functionality, the parallel job class also supports the users in up-scaling their simulation protocols. By default only a single calculation is executed at a given time, but by increasing the number of processor cores assigned to the Murnaghan job object to eleven while maintaining the number of processor cores for the individual calculation restricted to a single processor core, the level of parallel execution can be controlled:

```
murn.server.cores = 11
murn.run()
```

In contrast to the previous parallel execution of multiple job objects in the background, the parallel job object is optimised to handle the scheduling of the calculations. So, if the number of cores for the Murnaghan job is restricted to eight it is going to execute the first eight calculations followed by another three in a second iteration. Finally, the Murnaghan job object also provides a built-in plot function `murn.plot()` to visualise the result for the user.

```
# Plot the energy-volume curve
murn.plot()
```

Alternatively the user can access the aggregated calculation results directly from the Murnaghan job object and plot them in a custom routine. For reusing the plotting function defined in the previous section, the collection of the simulation supercell volume and the extraction of the last energy is replaced with the following lines.

```
# Collect the energy volume pairs
vol, eng = murn["output/volume"], murn["output/energy"]
```

This demonstrates how specific job objects can simplify the development of complex simulation protocols by providing a set of default parameters, commonly used functionality like the rescaling of the simulation cell and finally the technical utilities to handle the parallel execution of the calculations. These job types which include multiple job objects are called meta jobs and they are used to define standardised building blocks to construct complex simulation protocols. Following the methodology of the pyiron IDE, the meta job objects behave just like regular job objects. This enables nesting multiple meta jobs to construct pipelines and even more complex simulation protocols. These simulation pipleines are introduced in Sec. A.3.

A.2 Interactive Calculation

By default the pyiron IDE communicates with the simulation codes by writing the code-specific input files, executing the simulation code and afterwards parsing the code-specific output files, like discussed above in the simulation life cycle in Fig. 4.1. Still, this level of overhead is not sufficient for complex thermodynamic simulation protocols like thermodynamic integration, which requires coupling two simulation codes at every time step during the calculation of a MD trajectory. As a consequence, the pyiron IDE implements interactive execution modes for all simulation code job objects which enables exchanging information during the run time. In contrast to other solutions, the pyiron IDE does not require modifying the executable of the simulation code to enable interactive coupling but instead it supports a wide range of interactive simulation protocols and can easily be extended. Preferably, the internal interactive interface is based on Python bindings provided by the simulation code, e.g. the one implemented for LAMMPS, which simplifies the development of the interface. Alternatively, the pyiron IDE also supports communicating via named pipes which is implemented in the S/PHI/nX code or via the standard process input and output which is implemented in the VASP code. To demonstrate the advantage of the interactive coupling the optimisation of an atomistic structure is selected. While most simulation codes provide internal minimisation routines and external minimizer helps to compare the different simulation codes. In this case the simulation code is just used as a atomistic engine, which calculates the energy and forces for a given structure. Starting with the import of the optimiser class of the scientific Python library [12] and the import of the pyiron project class:

```
# Import the scipy optimize module and pyiron
from scipy import optimize
from pyiron import Project
```

The pyiron project class is then used to create a new pyiron project object:

```
# Create Project
pr = Project("interactive")
```

Inside the project object a cubic aluminium structure is created using experimental parameters for the lattice constant. This structure contains a total of four atoms. So the first atom is displaced by 0.01 Å in x-direction, by adding the displacement to the position of the atomistic structure object.

```
# Create structure
structure = pr.create.structure.ase.bulk("Al", cubic=True)
structure.positions[0,0] += 0.01
```

Following the creation of the structure object, a LAMMPS job object is created and the aluminium structure object is assigned. In analogy to the previous LAMMPS calculation the interatomic potential is selected automatically from the internal database. These settings are analogue to the configuration of the non-interactive calculation in the previous section. To enable the interactive execution on the run mode of the server object needs to be set to interactive `job.server.run_mode.interactive`.

```
# Create Lammps job object
job = pr.create.job.Lammps(
    job_name="lmp_interactive"
)
job.structure = structure
```

```
# Set run mode interactive
job.server.run_mode.interactive = True
```

For the user switching from the file based interface to the Python bindings is reduced to changing a single variable, just like switching from in-line execution to executing the calculation in the background in Sec. 4.3.4. This principle of reducing complex technical changes of the simulation protocol to the change of a single variable enables rapid prototyping in the pyiron IDE. To connect the pyiron job to the scipy minimizer a function to calculate the total energy for a given set of coordinates `x` is defined. Internally the function assigns the updated coordinates to the structure object of the job object and calls the run function of the job object `run()`. Afterwards these updated coordinates are communicated to the simulation codes and the object is updated internally. Finally, the function returns the total energy of the job object.

```
# Get energy for positions
def get_energy(x):
    job.structure.positions = x.reshape(-1, 3)
    job.run()
    return job.output.energy_pot[-1]
```

The transformation of the coordinates is required to be compatible to the multi-dimensional minimization function of the scientific library. The function for conjugate gradient based minimization `fmin_cg()` can access the pyiron job object using the function to calculate the energy `get_energy()` defined above.

```
# Use scipy optimise
_ = optimize.fmin_cg(
    x0=job.structure.positions.flatten(),
    f=get_energy,
)
```

Being able to couple simulation codes with existing python libraries interactively by reusing the same objects used for static calculations previously demonstrates the flexibility of the pyiron IDE. For the given calculation the output of the minimisation is ignored and instead the job object is analysed separately in the following. However before the analysis it is recommended to close the interactive interface to stop the execution of the simulation code.

```
# Close interative calculation
job.interactive_close()
```

After stopping the execution of the simulation code, the change in total energy is inspected by comparing the first and the last total energy calculated. While the change in total energy is small, this example demonstrates the flexibility of the interactive interface implemented in the pyiron IDE.

```
# Compare first and last total energy
job['output/generic/energy_tot'][0]
```

```
>>> -13.439
job['output/generic/energy_tot'][-1]
>>> -13.440
```

Again it is possible to switch the simulation protocol from the interatomic potential simulation code LAMMPS to the DFT simulation code S/PHI/nX by just changing the job type in the create job function `create.job()`. In the same way the pyiron IDE provides wrapper classes for the optimisers implemented in the Python scientific library as well as other wrapper classes which can be used in combination with the interactive simulation jobs.

```
# Use Scipy Minimizer object
minim = job.create.job.ScipyMinimizer(
    job_name='mini_lmp')
minim.run()
```

In comparison to the previous example the Minimizer meta job object behaves like the Murnaghan job object from Sec. A.1. Just like the Murnaghan job object it is created from the existing reference job object, so the reference job is automatically assigned. When calling the run function of the Minimizer object `run()` it internally couples the Minimizer of the scientific library with the interactive simulation code object and closes the interactive simulation code object after the successful minimization. Again the strength of the pyiron IDE is abstracting the technical complexity and focusing on the physics implemented in the simulation protocol, using building blocks the user is already familiar with.

A.3 Calculation Pipelines

Combining both the meta job objects and the interactive execution of the job objects even more complex calculation pipelines can be constructed. These calculation pipelines consist of a series of job objects and meta job objects and are executed as one joint simulation protocol by calling the run method `run()` of the top-level meta job object. In contrast to simulation protocols defined in Jupyter notebooks which can be reused using the script job objects, simulation protocols defined in pipelines form a more modular structure at the cost of a more complex implementation of the individual building blocks. Typical use cases for calculation pipelines are high-throughput parameter studies of a large number of individual calculations with a fixed dependence. The detailed implementation of a custom job class inside the pyiron IDE is demonstrated in the next section. For this section the focus is on combining existing meta job objects to construct calculation pipelines. While in principle the same pipeline could be constructed without using the interactive execution, in practice the interactive execution commonly accelerates the calculation by reducing the communication overhead in particular when using Python bindings. The quasi-harmonic approximation for a fcc aluminium structure is selected as an example. This includes three steps: first the minimization of the structure, followed by the calculation of an energy volume curve around the optimised volume of the minimisation and finally the calculation of the phonon spectrum at each strain of the energy-volume curve calculation. Again, the new Jupyter notebook is started with the import of the pyiron project class.

```
# Import pyiron
from pyiron import Project
```

After the import of the project class the individual steps of the pipeline are defined as separate functions, so they can be combined later on. Each function takes a job object as an input and uses the copy to function `copy_to()` to duplicate the job object as a template for the internal calculations. In the first step, the minimisation the job object is copied and afterwards the minimisation function of the job object `calc_minimize()` is called with the additional parameter `pressure=0` to optimise the simulation lattice with respect to the external pressure of 0 bar. At the end of the first function the resulting job object is returned without executing it.

```
# Define Calculation Functions
# 1. Step: Minimize Structure
def calc_minimize(job):
    job_mini = job.copy_to(
        new_job_name=job.job_name + '_mini'
    )
    job_mini.calc_minimize(pressure=0)
    return job_mini
```

The job names of the individual jobs are constructed by adding a suffix to the job name of the initial job object. The suffix of the minimisation is *'_mini'*. In analogy the suffix for the second step the calculation of the energy-volume curve with the Murnaghan meta job object it *'_murn'*. In the second step the create job function `create.job()` function of the job object is used to create the meta job object which includes the job object. At the end of the function the Murnaghan meta job object is returned.

```
# 2. Step: Calculate Energy-Volume Curve
def calc_murnaghan(job):
    job_murn = job.copy_to(
        new_job_name=job.job_name + '_murn_lmp'
    )
    return job_murn.create.job.Murnaghan(
        job_name=job.job_name + '_murn'
    )
```

In the third and final step the function name includes the for each structure suffix in the function name `for_each_structure` which inside the pyiron pipelines corresponds to a many-to-many relationship. While all other functions are interpreted as a one-to-one relationship. Inside the function the job object is again copied, used to create the meta job object, for calculating the phonons, the phonopy job `PhonopyJob` and afterwards the meta job object is nested in another meta job object, which maps it to a list of structures, the structure list master meta job `StructureListMaster`. While these meta job objects have not been introduced previously their implementation is analogue to the calculation of the energy-volume curve with the Murnaghan meta job object. The phonopy job `PhonopyJob` calculates the phonons for a given structure with the phonopy thermodyanmics toolkit [136] and the structure list master `StructureListMaster` rather than receiving a single atomistic structure as input receives a list of atomistic structures and evaluates each of them with its reference job. In this case for each atomistic structure which is assigned to the structure list master a separate phonon calculation is executed. The structure list master `StructureListMaster` can also be used to calculate energy-volume curves with a list of strained structures being required rather than a single structure.

```
# 3. Step: Calculate Phonons in Quasi-harmonic Approximation
def calc_phonons_for_each_structure(job):
    job_phono = job.copy_to(
        new_job_name=job.job_name + '_phono_lmp'
    )
    phono = job_phono.create_job(
        job_type=pr.job_type.PhonopyJob,
        job_name=job.job_name + '_phono'
    )
    return phono.create_job(
        job_type=pr.job_type.StructureListMaster,
        job_name='struct_master'
    )
```

After the definition of these functions, the project object is created. While in principle it is possible to define the functions later, in practice defining the functions in the beginning of the simulation protocol prevents accidentally overwriting existing functions and simplifies the migration of the functions from the simulation protocol to an external Python module. Such a module can then be reused in multiple simulation protocols, which again simplifies the up-scaling of an existing simulation protocol.

```
# Create Project
pr = Project('calculation')
```

Rather than choosing the default interatomic potential, the potential as well as the element are defined as variables to modify them later. Here, the interatomic potential for nickel aluminium alloys [55] is selected and aluminium is selected as atomic species.

```
# Parameter
potential = "2004--Mishin-Y--Ni-Al--LAMMPS--ipr1"
element = "Al"
```

Also following the previous example the FCC aluminium structure object with a cubic supercell is created using experimental lattice parameters.

```
# Structure
structure = pr.create_ase_bulk(element, cubic=True)
```

Following the structure object the LAMMPS job object is created as a template for all calculations in the pipeline. All calculations are going to use exactly the same parameters to produce coherent results. Still, it is also possible to modify the job object in the individual calculation steps, even though it is not recommended. Afterwards the structure object and the interatomic potential are assigned to the job object.

```
# Job Template
job = pr.create_job(
    job_type=pr.job_type.Lammps,
    job_name="lmp"
)
job.structure = structure
job.potential = potential
job.server.run_mode.interactive = True
```

In addition the interactive interface is enabled by switching the run mode of the server object to accelerate the calculation and the job object is combined with the functions defined above using the create pipeline function `create_pipeline()` of the job object. It connects the individual functions and constructs the simulation pipeline.

```
# Construct Pipeline
job_lst_master = job.create_pipeline(
    step_lst=[
        calc_minimize,
        calc_murnaghan,
        calc_phonons_for_each_structure,
    ]
)
```

Finally the pipeline is executed by calling the run function `run()` just like any other job and meta job object. In this example the simulation pipeline is executed directly in-line. In practice it would be submitted to the job scheduler of a HPC cluster. It again benefits from the parallel execution implemented at the core of the meta job objects.

```
# Execute Pipeline
job_lst_master.run()
```

With the option to start with rapid prototyping by interactively developing a simulation protocol and then being able to up-scale the same simulation protocol using the same framework by only slightly modifying the simulation protocol is one of the core features of the pyiron IDE. It is enabled by implementing a set of object derived from the same class, the pyiron objects. So the users can extend and accelerate their simulation protocols depending on their needs.

A.4 Extendability

To address the expert users who either develop their own simulation codes or want to use the pyiron IDE with simulation codes which are currently not yet implemented in the pyiron IDE, the implementation of custom classes is simplified. In this example a simple job object is created which copies the input from the input file to the output file. Again a new jupyter notebook is used to develop this example and test it. Afterwards, the migration of the newly implemented class to a separate module is explained and it is shown how to make it accessible within the pyiron IDE. Given the simplicity of the task – copying data from the input to the output – the job template class `TemplateJob` is used to derive the new class. This new class includes an input object, an executable object, a server object and the pyiron data storage interface which is already configured to store the input automatically. For more complex classes the advanced user is referred to the pyiron website and the implementation of the simulation codes LAMMPS, VASP and S/PHI/nX which are already included in the pyiron IDE.

```python
# Import Template Class
from pyiron.base.job.template import TemplateJob
```

In the first implementation a file-based interface is developed. For this two main functions are implemented, one to write the input files `write_input()` and one to collect the output `collect_output()`. The write input function just calls write input on the single input object defined for this class. In a more complex example the write input function could also include consistency checks for the input values or write multiple input objects. Following the writing of the input, the executable is executed in the same directory. In this example this calls a shell script to copy the input to the output file. Again, the executable interface is extendable and can link to a shell script in the pyiron resources directory. Finally, the collect output function opens the output file, extracts the value copied from the input file and stores it in the HDF5 file.

```python
# Create Custom Job Class
class ToyJob(TemplateJob):
    def __init__(self, project, job_name):
        super(ToyJob, self).__init__(project, job_name)
        self.input['input_energy'] = 100
        # Copy input to output in a shell script
        self.executable = "cat input > output"

    # Write Input Files
    def write_input(self):
        self.input.write_file(
            file_name="input",
            cwd=self.working_directory
        )
        # Collect Output Files
        def collect_output(self):
            file = join(self.working_directory, "output")
            with open(file) as f:
                line = f.readlines()[0]
            energy = float(line.split()[1])
            with self.project_hdf5.open("output/generic") as h5out:
                h5out["energy_tot"] = energy
```

With this very fundamental definition of a job class and this job class can already be used in the same notebook. In this example pyiron is imported, a project and a job object are created, the input of the job object is printed, the execution is triggered and finally the output stored in the pyiron data storage is printed.

```
    from pyiron import Project
    pr = Project('toyclass')
    job = pr.create_job(
        job_type=ToyJob,
        job_name="toy"
    )
    print(job.input)
    >>> "   Parameter       Value"
    >>> "0  input_energy    100"
    job.run()
    >>> "The job toy was saved and received the ID: 10"
    job['output/generic/energy_tot']
    >>> "100.0"
```

In contrast to the previous examples in the create job function `create.job()` is called directly with the job class defined as an additional job type parameter `job_type`. This is enabled by deriving the job class from the job template class `TemplateJob`. Encapsulating custom classes inside pyiron job classes helps to structure the simulation protocols and provides direct access to the job management system implemented inside the pyiron IDE. Commonly these custom classes are also used to wrap Python functions which require submission to the job scheduler of an HPC cluster. For such a pure Python class it is not efficient to communicate via the file-based interface. Instead of this, the python only job flag `self._python_only_job = True` is set and the run static function `run_static()` is defined, which executes the calculation and stores the results in the HDF5 file without the need to write input files and collect output files.

```
    # Create Custom Job Class
    class ToyJob(TemplateJob):
        def __init__(self, project, job_name):
            super(ToyJob, self).__init__(project, job_name)
            self.input['input_energy'] = 100
            self._python_only_job = True
    # This function is executed
    def run_static(self):
        with self.project_hdf5.open("output/generic") as h5out:
            h5out["energy_tot"] = self.input["input_energy"]
        self.status.finished = True
```

In analogy to the previous example above the input values are simply copied to the output. Finally, to integrate a custom class in the pyiron simulation protocol and have it listed in the job type list of the project object `pr.job_type` the newly developed class is moved to a separate module. In this example a folder named toy is created and the class definition is placed in a Python file inside this folder *"toy/toy.py"*. Besides the Python file, the python module file *"__init__.py"* is added with the following two lines:

```
from pyiron.base.job.jobtype import JOB_CLASS_DICT
JOB_CLASS_DICT['ToyJob'] = 'toy.toy'
```

These lines import the job type definition from the pyiron base class and extend it by adding the newly developed job class. This capability of being able to add new job classes inside the pyiron IDE following the same workflow used for developing simulation protocols, helps the users of the pyiron to evolve to developers who actively contribute to the development of the pyiron IDE. The users of the pyiron IDE start by writing simple simulation protocols, executing one command at a time, then they summarise commonly used commands in functions, these functions grow to modules and finally they create their own job classes based on these functions to systematically up-scale their workflow. With this standardised process the pyiron IDE goes beyond of just publishing existing simulation workflows, because it enables the users to define higher level interfaces for those who want to reuse an existing simulation protocol as a building block for a more complex simulation protocols.

A.5 List of Abbreviations

Abbreviation	Description
ASE	Atomic Simulation Environment
BCC	Body-Centered Cubic
CALPHAD	CALculation of PHAse Diagrams
CNA	Common Neighbour Analysis
DFT	Density Functional Theory
DOS	Density of States
EAM	Embedded Atom Method
EOS	Equation of State
ESPEI	Extensible Self-optimizing Phase Equilibria Infrastructure

Table A.1: List of Abbreviations (A-E)

Abbreviation	Description
FAIR	Findable Accessible Interoperable Reusable
FCC	Face-Centered Cubic
FFT	Fast Fourier Transform
GGA	Generalized Gradient Approximation
GUI	Graphical User Interface
GW	Greens Function and the screened Coulomb Interaction
HCP	Hexagonal Close-Packed
HDF5	Hierarchical Data Format version 5
HPC	High-Performance Computing
IDE	Integrated Development Environment
KIM	Knowledge base of Interatomic Models
LAMMPS	Large-scale Atomic/Molecular Massively Parallel Simulator
LAPW+lo	Linearized Augmented Plane Wave plus Local Orbital
LDA	Local-Density Approximation
MD	Molecular Dynamics
MEAM	Modified Embedded Atom Method
MPI	Message Passing Interface
NIST	National Institute of Standards and Technology
NPH	Isoenthalpic-Isobaric Ensemble
NPT	Isothermal-Isobaric Ensemble
NVE	Microcanonical Ensemble
NVT	Canconical Ensemble
PAW	Projector Augmented Wave Function
PBE	Perdew-Burke-Ernzerhof
PDUQ	Phase Diagram Uncertainty Quantification
PSE	Potential Energy Surface
SQL	Structured Query Language
SQS	Special Quasi-random Structures
SVD	Singular Value Decomposition
TILD	Thermodynamic Integration using Langevin Dynamics
TOR-TILD	Two-Optimized References TILD
TU-TILD	Two-stage Upsampled TILD
UP-TILD	Upsampled TILD
VASP	Vienna Ab initio Simulation Package
μVT	Grandcanconical Ensemble

Table A.2: List of Abbreviations (F-Z)

Acknowledgements

The development of a scientific software such as the pyiron IDE requires the combination of three aspects: scientific expertise, practical experience in software development and a collaborative environment.

So, I would like to express my gratitude to Prof. Dr. Jörg Neugebauer for not only covering all three aspects to the highest extent, but for establishing a department which continues to embrace these values. I feel truly honoured to take part in this journey, learning from leading experts in the field of ab initio thermodynamics, fostering a systematic scientific understanding and receiving constructive feedback in any situation. The countless hours of discussion with Prof. Dr. Jörg Neugebauer made me the scientist I am today.

I am fully aware that such an opportunity can by no means be taken for granted and I am grateful to my supervisor Tilmann Hickel. He continuously supported me on my journey, encouraged me and was always available for a second opinion. This thesis would not be possible without him.

In addition, I would like to thank the members of the computational materials design department at the Max-Planck-Institut für Eisenforschung. Especially Christoph Freysoldt, Blazej Grabowski, Fritz Körmann, Mira Todorova and Li-Fang Zhu for the extensive discussions, Liam Huber, Marvin Poul, Sudarsan Surendralal and Osamu Waseda for the joined development of the pyiron IDE and my office colleagues Lekshmi Sreekala and Su-Hyun Yoo.

Finally, I would like to thank my collaborators outside the institute:

- Ralf Drautz, Thomas Hammerschmidt, Yury Lysogorskiy and Sarath Menon from Interdisciplinary Centre for Advanced Materials Simulation in Bochum (Germany) and Karsten Albe, Alexander Stukowski and Niklas Leimeroth from the Technical University of Darmstadt (Germany) for collaborating on the evaluation of interatomic potentials,
- Alexander Shapeev and Edgar Makarov from the Skolkovo Institute of Science and Technology in Moscow (Russia) for the collaboration on the quantification of uncertainties for density functional theory,
- Thomas Swinburne from the Center of Interdisciplinary Nano Science in Marseille (France) for the collaboration on coarse-grained models as part of the long-term program on Complex High-Dimensional Energy Landscapes at the Institute for Pure and Applied Mathematics at the University of California in Los Angeles (United States),

- Zi-Kui Liu, Richard Otis and Brandon Bocklund from the Pennsylvania State University (United States) for the collaboration on calculating phase diagrams,
- Shoji Ishibashi from the National Institute of Advanced Industrial Science and Technology in Tsukuba (Japan) for the collaboration on calculating melting temperatures and
- Wolf Gero Schmidt from the University of Paderborn (Germany) for refereeing my thesis.

Apart from the scientific support I thank my family for laying the foundation of my curiosity, my education and my values. Furthermore, I am grateful for the lucky coincidence that I met Antonia, she helps me to maintain a science-life balance.

Bibliography

[1] M. Friák, W. A. Counts, D. Ma, B. Sander, D. Holec, D. Raabe, and J. Neugebauer, Materials **5**, 1853–1872 (2012).

[2] S. Sandlöbes, M. Friák, S. Korte-Kerzel, Z. Pei, J. Neugebauer, and D. Raabe, Scientific Reports **7**, 10458 (2017).

[3] L. Kaufman and H. Bernstein, *Computer calculation of phase diagrams with special reference to refractory metals* (Academic Press Inc, 1970).

[4] U. R. Kattner, Tecnol. Metal. Mater. Min. **13**, 3–15 (2016).

[5] T. Hickel, B. Grabowski, F. Körmann, and J. Neugebauer, Journal of Physics: Condensed Matter **24**, 053202 (2012).

[6] J. Dean and S. Ghemawat, Commun. ACM **51**, 107–113 (2008).

[7] P. Hohenberg and W. Kohn, Phys. Rev. **136**, B864–B871 (1964).

[8] W. Kohn and L. J. Sham, Phys. Rev. **140**, A1133–A1138 (1965).

[9] R. J. Maurer, C. Freysoldt, A. M. Reilly, J. G. Brandenburg, O. T. Hofmann, T. Björkman, S. Lebègue, and A. Tkatchenko, Annual Review of Materials Research **49**, 1–30 (2019).

[10] A. E. Mattsson, P. A. Schultz, M. P. Desjarlais, T. R. Mattsson, and K. Leung, Modelling and Simulation in Materials Science and Engineering **13**, R1–R31 (2004).

[11] W. Kohn, Rev. Mod. Phys. **71**, 1253–1266 (1999).

[12] R. O. Jones and O. Gunnarsson, Rev. Mod. Phys. **61**, 689–746 (1989).

[13] K. Capelle, **36**, Provided by the SAO/NASA Astrophysics Data System, 1318–1343 (2006).

[14] D. R. Hartree, Mathematical Proceedings of the Cambridge Philosophical Society **24**, 89–110 (1928).

[15] N. D. Mermin, Phys. Rev. **137**, A1441–A1443 (1965).

[16] M. G. Medvedev, I. S. Bushmarinov, J. Sun, J. P. Perdew, and K. A. Lyssenko, Science **355**, 49–52 (2017).

[17] J. P. Perdew and K. Schmidt, AIP Conference Proceedings **577**, 1–20 (2001).

[18] J. P. Perdew, A. Ruzsinszky, J. Tao, V. N. Staroverov, G. E. Scuseria, and G. I. Csonka, The Journal of Chemical Physics **123**, 062201 (2005).

[19] B. Grabowski, L. Ismer, T. Hickel, and J. Neugebauer, Phys. Rev. B **79**, 134106 (2009).

[20] J. P. Perdew, K. Burke, and M. Ernzerhof, Phys. Rev. Lett. **77**, 3865–3868 (1996).

[21] M. C. Payne, M. P. Teter, D. C. Allan, T. A. Arias, and J. D. Joannopoulos, Rev. Mod. Phys. **64**, 1045–1097 (1992).

[22] P. Kratzer and J. Neugebauer, Frontiers in Chemistry **7**, 106 (2019).

[23] P. G. Dacosta, O. H. Nielsen, and K. Kunc, Journal of Physics C: Solid State Physics **19**, 3163–3172 (1986).

[24] H. J. Monkhorst and J. D. Pack, Phys. Rev. B **13**, 5188–5192 (1976).

[25] J. D. Pack and H. J. Monkhorst, Phys. Rev. B **16**, 1748–1749 (1977).

[26] P. Wisesa, K. A. McGill, and T. Mueller, Phys. Rev. B **93**, 155109 (2016).

[27] W. S. Morgan, J. J. Jorgensen, B. C. Hess, and G. L. Hart, Computational Materials Science **153**, 424–430 (2018).

[28] G. L. W. Hart, J. J. Jorgensen, W. S. Morgan, and R. W. Forcade, Journal of Physics Communications **3**, 065009 (2019).

[29] W. S. Morgan, J. E. Christensen, P. K. Hamilton, J. J. Jorgensen, B. J. Campbell, G. L. Hart, and R. W. Forcade, Computational Materials Science **173**, 109340 (2020).

[30] N. Marzari, D. Vanderbilt, and M. C. Payne, Phys. Rev. Lett. **79**, 1337–1340 (1997).

[31] M. Methfessel and A. T. Paxton, Phys. Rev. B **40**, 3616–3621 (1989).

[32] P. E. Blöchl, O. Jepsen, and O. K. Andersen, Phys. Rev. B **49**, 16223–16233 (1994).

[33] G. Kresse and J. Furthmüller, Computational Materials Science **6**, 15–50 (1996).

[34] K. Lejaeghere et al., Science **351**, `10.1126/science.aad3000` (2016).

[35] D. Singh, Phys. Rev. B **43**, 6388–6392 (1991).

[36] P. E. Blöchl, Phys. Rev. B **50**, 17953–17979 (1994).

[37] M. S. Hybertsen and S. G. Louie, Phys. Rev. B **34**, 5390–5413 (1986).

[38] A. M. Rappe, K. M. Rabe, E. Kaxiras, and J. D. Joannopoulos, Phys. Rev. B **41**, 1227–1230 (1990).

[39] G. Ziegenhain and H. M. Urbassek, Philosophical Magazine **89**, 2225–2238 (2009).

[40] A. P. Bartók and G. Csányi, International Journal of Quantum Chemistry **115**, 1051–1057 (2015).

[41] A. Hernandez, A. Balasubramanian, F. Yuan, S. A. M. Mason, and T. Mueller, npj Computational Materials **5**, 112 (2019).

[42] Y. Zuo, C. Chen, X. Li, Z. Deng, Y. Chen, J. Behler, G. Csányi, A. V. Shapeev, A. P. Thompson, M. A. Wood, and S. P. Ong, The Journal of Physical Chemistry A **124**, `10.1021/acs.jpca.9b08723` (2020).

[43] Y. Lysogorskiy, C. v. d. Oord, A. Bochkarev, S. Menon, M. Rinaldi, T. Hammerschmidt, M. Mrovec, A. Thompson, G. Csányi, C. Ortner, and R. Drautz, npj Computational Materials **7**, 97 (2021).

[44] Y. Lysogorskiy, T. Hammerschmidt, J. Janssen, J. Neugebauer, and R. Drautz, Modelling and Simulation in Materials Science and Engineering **27**, 025007 (2019).

[45] M. Y., “Interatomic potentials for metals”, in *Handbook of materials modeling*, edited by Y. S. (Springer, Dordrecht, 2005) Chap. 2.2, pp. 459–478.

[46] P. M. Morse, Phys. Rev. **34**, 57–64 (1929).

[47] L. A. Girifalco and V. G. Weizer, Phys. Rev. **114**, 687–690 (1959).

[48] C. S. G. Cousins, Journal of Physics C: Solid State Physics **4**, 1117–1123 (1971).

[49] F. Ercolessi, M. Parrinello, and E. Tosatti, Philosophical Magazine A **58**, 213–226 (1988).

[50] M. S. Daw and M. I. Baskes, Phys. Rev. B **29**, 6443–6453 (1984).

[51] M. W. Finnis and J. E. Sinclair, Philosophical Magazine A **50**, 45–55 (1984).

[52] D. Wolf, P. R. Okamoto, S. Yip, J. F. Lutsko, and M. Kluge, Journal of Materials Research **5**, 286–301 (1990).

[53] Z. H. Jin, P. Gumbsch, K. Lu, and E. Ma, Phys. Rev. Lett. **87**, 055703 (2001).

[54] K. Moriguchi and M. Igarashi, Phys. Rev. B **74**, 024111 (2006).

[55] Y. Mishin, Acta Materialia **52**, 1451–1467 (2004).

[56] C. A. Howells and Y. Mishin, Modelling and Simulation in Materials Science and Engineering **26**, 085008 (2018).

[57] T. J. Lenosky, B. Sadigh, E. Alonso, V. V. Bulatov, T. D. de la Rubia, J. Kim, A. F. Voter, and J. D. Kress, Modelling and Simulation in Materials Science and Engineering **8**, 825–841 (2000).

[58] E. B. Tadmor, R. S. Elliott, J. P. Sethna, R. E. Miller, and C. A. Becker, JOM **63**, 17 (2011).

[59] C. A. Becker, F. Tavazza, Z. T. Trautt, and R. A. B. de Macedo, Current Opinion in Solid State and Materials Science **17**, Frontiers in Methods for Materials Simulations, 277–283 (2013).

[60] L. M. Hale, Z. T. Trautt, and C. A. Becker, Modelling and Simulation in Materials Science and Engineering **26**, 055003 (2018).

[61] A. Stukowski, Modelling and Simulation in Materials Science and Engineering **20**, 045021 (2012).

[62] R. Drautz, Phys. Rev. B **99**, 014104 (2019).

[63] A. Thompson, L. Swiler, C. Trott, S. Foiles, and G. Tucker, Journal of Computational Physics **285**, 316–330 (2015).

[64] A. V. Shapeev, Multiscale Modeling & Simulation **14**, 1153–1173 (2016).

[65] A. P. Bartók, M. C. Payne, R. Kondor, and G. Csányi, Phys. Rev. Lett. **104**, 136403 (2010).

[66] J. Behler and M. Parrinello, Phys. Rev. Lett. **98**, 146401 (2007).

[67] K. Gubaev, E. V. Podryabinkin, and A. V. Shapeev, The Journal of Chemical Physics **148**, 241727 (2018).

[68] K. Gubaev, E. V. Podryabinkin, G. L. Hart, and A. V. Shapeev, Computational Materials Science **156**, 148–156 (2019).

[69] R. Jinnouchi, F. Karsai, and G. Kresse, Phys. Rev. B **100**, 014105 (2019).

[70] R. P. Feynman, Phys. Rev. **56**, 340–343 (1939).

[71] D. Marx and J. Hutter, *Ab initio molecular dynamics: basic theory and advanced methods* (Cambridge University Press, 2009).

[72] L. Verlet, Phys. Rev. **159**, 98–103 (1967).

[73] W. C. Swope, H. C. Andersen, P. H. Berens, and K. R. Wilson, The Journal of Chemical Physics **76**, 637–649 (1982).

[74] J. Thijssen, “Molecular dynamics simulations”, in *Computational physics*, 2nd ed. (Cambridge University Press, 2007), pp. 197–262.

[75] T. Schneider and E. Stoll, Phys. Rev. B **17**, 1302–1322 (1978).

[76] W. G. Hoover and B. L. Holian, Physics Letters A **211**, 253–257 (1996).

[77] W. Shinoda, M. Shiga, and M. Mikami, Phys. Rev. B **69**, 134103 (2004).

[78] M. Born and V. Fock, Zeitschrift für Physik **51**, 165–180 (1928).

[79] X. Zhang, B. Grabowski, F. Körmann, C. Freysoldt, and J. Neugebauer, Phys. Rev. B **95**, 165126 (2017).

[80] X. Zhang, B. Grabowski, T. Hickel, and J. Neugebauer, Computational Materials Science **148**, 249–259 (2018).

[81] G. P. Francis and M. C. Payne, Journal of Physics: Condensed Matter **2**, 4395–4404 (1990).

[82] F. D. Murnaghan, Proceedings of the National Academy of Sciences **30**, 244–247 (1944).

[83] F. Birch, Phys. Rev. **71**, 809–824 (1947).

[84] P. Vinet, J. R. Smith, J. Ferrante, and J. H. Rose, Phys. Rev. B **35**, 1945–1953 (1987).

[85] B. Grabowski, P. Söderlind, T. Hickel, and J. Neugebauer, Phys. Rev. B **84**, 214107 (2011).

[86] A. Gupta, B. T. Kavakbasi, B. Dutta, B. Grabowski, M. Peterlechner, T. Hickel, S. V. Divinski, G. Wilde, and J. Neugebauer, Phys. Rev. B **95**, 094307 (2017).

[87] X. Zhang, B. Grabowski, F. Körmann, A. V. Ruban, Y. Gong, R. C. Reed, T. Hickel, and J. Neugebauer, Phys. Rev. B **98**, 224106 (2018).

[88] F. Körmann, A. A. H. Breidi, S. L. Dudarev, N. Dupin, G. Ghosh, T. Hickel, P. Korzhavyi, J. A. Muñoz, and I. Ohnuma, physica status solidi (b) **251**, 53–80 (2014).

[89] F. Körmann, T. Hickel, and J. Neugebauer, Current Opinion in Solid State and Materials Science **20**, 77–84 (2016).

[90] R. Car and M. Parrinello, Phys. Rev. Lett. **55**, 2471–2474 (1985).

[91] R. P. Feynman, Rev. Mod. Phys. **20**, 367–387 (1948).

[92] G. J. Ackland, Journal of Physics: Condensed Matter **14**, 2975–3000 (2002).

[93] V. L. Moruzzi, J. F. Janak, and K. Schwarz, Phys. Rev. B **37**, 790–799 (1988).

[94] A. I. Duff, T. Davey, D. Korbmacher, A. Glensk, B. Grabowski, J. Neugebauer, and M. W. Finnis, Phys. Rev. B **91**, 214311 (2015).

[95] B. Grabowski, Y. Ikeda, P. Srinivasan, F. Körmann, C. Freysoldt, A. I. Duff, A. Shapeev, and J. Neugebauer, npj Computational Materials **5**.

[96] O. Redlich and A. T. Kister, Industrial & Engineering Chemistry **40**, 345–348 (1948).

[97] Y. Wang, S. Curtarolo, C. Jiang, R. Arroyave, T. Wang, G. Ceder, L.-Q. Chen, and Z.-K. Liu, Calphad **28**, 79–90 (2004).

[98] Z.-K. Liu, Acta Materialia **200**, 745–792 (2020).

[99] J. M. Sanchez, Phys. Rev. B **48**, 14013–14015 (1993).

[100] B. Sadigh, P. Erhart, A. Stukowski, A. Caro, E. Martinez, and L. Zepeda-Ruiz, Phys. Rev. B **85**, 184203 (2012).

[101] A. Stukowski, B. Sadigh, P. Erhart, and A. Caro, Modelling and Simulation in Materials Science and Engineering **17**, 075005 (2009).

[102] L. B. Pártay, A. P. Bartók, and G. Csányi, The Journal of Physical Chemistry B **114**, PMID: 20701382, 10502–10512 (2010).

[103] R. J. N. Baldock, L. B. Pártay, A. P. Bartók, M. C. Payne, and G. Csányi, Phys. Rev. B **93**, 174108 (2016).

[104] R. J. N. Baldock, N. Bernstein, K. M. Salerno, L. B. Pártay, and G. Csányi, Phys. Rev. E **96**, 043311 (2017).

[105] R. Otis and Z.-K. Liu, Journal of Open Research Software **5**, 1 (2017).

[106] A. P. Bartók, J. Kermode, N. Bernstein, and G. Csányi, Phys. Rev. X **8**, 041048 (2018).

[107] A. Zunger, S.-H. Wei, L. G. Ferreira, and J. E. Bernard, Phys. Rev. Lett. **65**, 353–356 (1990).

[108] S.-H. Wei, L. G. Ferreira, J. E. Bernard, and A. Zunger, Phys. Rev. B **42**, 9622–9649 (1990).

[109] D. J. Wales and J. P. K. Doye, The Journal of Physical Chemistry A **101**, 5111–5116 (1997).

[110] A. R. Oganov and C. W. Glass, The Journal of Chemical Physics **124**, 244704 (2006).

[111] Y. Wang, J. Lv, L. Zhu, and Y. Ma, Computer Physics Communications **183**, 2063–2070 (2012).

[112] P. Avery and E. Zurek, Computer Physics Communications **213**, 208–216 (2017).

[113] J. Mei and J. W. Davenport, Phys. Rev. B **46**, 21–25 (1992).

[114] O. Sugino and R. Car, Phys. Rev. Lett. **74**, 1823–1826 (1995).

[115] L. Vočadlo and D. Alfè, Phys. Rev. B **65**, 214105 (2002).

[116] G. Grimvall, B. Magyari-Köpe, V. Ozolins, and K. A. Persson, Rev. Mod. Phys. **84**, 945–986 (2012).

[117] F. Lindemann, Physikalische Zeitschrift **11**, 609–612 (1910).

[118] J. J. Gilvarry, Phys. Rev. **102**, 308–316 (1956).

[119] M. Born, The Journal of Chemical Physics **7**, 591–603 (1939).

[120] J. R. Morris, C. Z. Wang, K. M. Ho, and C. T. Chan, Phys. Rev. B **49**, 3109–3115 (1994).

[121] D. Alfè, M. J. Gillan, and G. D. Price, The Journal of Chemical Physics **116**, 6170–6177 (2002).

[122] U. R. Pedersen, F. Hummel, G. Kresse, G. Kahl, and C. Dellago, Phys. Rev. B **88**, 094101 (2013).

[123] S. Menon, G. Díaz Leines, R. Drautz, and J. Rogal, The Journal of Chemical Physics **153**, 104508 (2020).

[124] D. Alfè, Phys. Rev. B **68**, 064423 (2003).

[125] L.-F. Zhu, B. Grabowski, and J. Neugebauer, Phys. Rev. B **96**, 224202 (2017).

[126] L.-F. Zhu, F. Körmann, A. V. Ruban, J. Neugebauer, and B. Grabowski, Phys. Rev. B **101**, 144108 (2020).

[127] G. Kresse and J. Hafner, Phys. Rev. B **47**, 558–561 (1993).

[128] G. Kresse and J. Furthmüller, Phys. Rev. B **54**, 11169–11186 (1996).

[129] S. Boeck, C. Freysoldt, A. Dick, L. Ismer, and J. Neugebauer, Computer Physics Communications **182**, 543–554 (2011).

[130] C. Freysoldt, Computational Materials Science **133**, 71–81 (2017).

[131] C. Freysoldt, J. Neugebauer, and C. G. Van de Walle, Phys. Rev. Lett. **102**, 016402 (2009).

[132] C. Freysoldt and J. Neugebauer, Phys. Rev. B **97**, 205425 (2018).

[133] C. Freysoldt, A. Mishra, M. Ashton, and J. Neugebauer, Phys. Rev. B **102**, 045403 (2020).

[134] O. Marquardt, S. Schulz, C. Freysoldt, S. Boeck, T. Hickel, E. P. O'Reilly, and J. Neugebauer, Optical and Quantum Electronics **44**, 183–188 (2012).

[135] S. Plimpton, Journal of Computational Physics **117**, 1–19 (1995).

[136] A. Togo and I. Tanaka, Scripta Materialia **108**, 1–5 (2015).

[137] S. R. Bahn and K. W. Jacobsen, English, Comput. Sci. Eng. **4**, 56–66 (2002).

[138] A. H. Larsen et al., Journal of Physics: Condensed Matter **29**, 273002 (2017).

[139] M. Ceriotti, J. More, and D. E. Manolopoulos, Computer Physics Communications **185**, 1019–1026 (2014).

[140] V. Kapil et al., Computer Physics Communications **236**, 214–223 (2019).

[141] C. Draxl and M. Scheffler, MRS Bulletin **43**, 676–682 (2018).

[142] A. Jain, G. Hautier, C. J. Moore, S. P. Ong, C. C. Fischer, T. Mueller, K. A. Persson, and G. Ceder, Computational Materials Science **50**, 2295–2310 (2011).

[143] A. Jain, S. P. Ong, G. Hautier, W. Chen, W. D. Richards, S. Dacek, S. Cholia, D. Gunter, D. Skinner, G. Ceder, and K. A. Persson, APL Materials **1**, 011002 (2013).

[144] S. P. Ong, W. D. Richards, A. Jain, G. Hautier, M. Kocher, S. Cholia, D. Gunter, V. L. Chevrier, K. A. Persson, and G. Ceder, Computational Materials Science **68**, 314–319 (2013).

[145] K. Mathew, J. H. Montoya, A. Faghaninia, S. Dwarakanath, M. Aykol, H. Tang, I.-h. Chu, T. Smidt, B. Bocklund, M. Horton, J. Dagdelen, B. Wood, Z.-K. Liu, J. Neaton, S. P. Ong, K. Persson, and A. Jain, Computational Materials Science **139**, 140–152 (2017).

[146] J. Anubhav, O. S. Ping, C. Wei, M. Bharat, Q. Xiaohui, K. Michael, B. Miriam, P. Guido, R. Gian-Marco, H. Geoffroy, G. Daniel, and P. K. A., Concurrency and Computation: Practice and Experience **27**, 5037–5059 (2015).

[147] L. Talirz, S. Kumbhar, E. Passaro, A. V. Yakutovich, V. Granata, F. Gargiulo, M. Borelli, M. Uhrin, S. P. Huber, S. Zoupanos, C. S. Adorf, C. W. Andersen, O. Schütt, C. A. Pignedoli, D. Passerone, J. VandeVondele, T. C. Schulthess, B. Smit, G. Pizzi, and N. Marzari, Scientific Data **7**, 299 (2020).

[148] G. Pizzi, A. Cepellotti, R. Sabatini, N. Marzari, and B. Kozinsky, Computational Materials Science **111**, 218–230 (2016).

[149] S. P. Huber et al., Scientific Data **7**, 300 (2020).

[150] A. V. Yakutovich, K. Eimre, O. Schütt, L. Talirz, C. S. Adorf, C. W. Andersen, E. Ditler, D. Du, D. Passerone, B. Smit, N. Marzari, G. Pizzi, and C. A. Pignedoli, Computational Materials Science **188**, 110165 (2021).

[151] U. Aydin, L. Ismer, T. Hickel, and J. Neugebauer, Phys. Rev. B **85**, 155144 (2012).

[152] T. R. Munter, D. D. Landis, F. Abild-Pedersen, G. Jones, S. Wang, and T. Bligaard, Computational Science & Discovery **2**, 015006 (2009).

[153] S. Curtarolo, W. Setyawan, G. L. Hart, M. Jahnatek, R. V. Chepulskii, R. H. Taylor, S. Wang, J. Xue, K. Yang, O. Levy, M. J. Mehl, H. T. Stokes, D. O. Demchenko, and D. Morgan, Computational Materials Science **58**, 218–226 (2012).

[154] K. Mathew, A. Singh, J. Gabriel, K. Choudhary, S. Sinnott, A. Davydov, F. Tavazza, and R. Hennig, English (US), Computational Materials Science **122**, 183–190 (2016).

[155] T. Mayeshiba, H. Wu, T. Angsten, A. Kaczmarowski, Z. Song, G. Jenness, W. Xie, and D. Morgan, Computational Materials Science **126**, 90–102 (2017).

[156] K. Choudhary, F. Y. P. Congo, T. Liang, C. Becker, R. G. Hennig, and F. Tavazza, Scientific Data **4**, `10.1038/sdata.2016.125` (2017).

[157] K. Choudhary, I. Kalish, R. Beams, and F. Tavazza, Scientific reports **7**, 5179 (2017).

[158] A. Goyal, P. Gorai, H. Peng, S. Lany, and V. Stevanović, Computational Materials Science **130**, 1–9 (2017).

[159] C. Draxl and M. Scheffler, Journal of Physics: Materials **2**, 036001 (2019).

[160] D. S. Karls, M. Bierbaum, A. A. Alemi, R. S. Elliott, J. P. Sethna, and E. B. Tadmor, The Journal of Chemical Physics **153**, 064104 (2020).

[161] G. Van Rossum and F. L. Drake Jr, *Python reference manual* (Centrum voor Wiskunde en Informatica Amsterdam, 1995).

[162] G. Van Rossum and F. L. Drake, *Python 3 reference manual* (CreateSpace, Scotts Valley, CA, 2009).

[163] E. Jones, T. Oliphant, P. Peterson, et al., *SciPy: open source scientific tools for Python*, 2001.

[164] S. van der Walt, S. C. Colbert, and G. Varoquaux, Computing in Science Engineering **13**, 22–30 (2011).

[165] J. D. Hunter, Computing in Science Engineering **9**, 90–95 (2007).

[166] F. Perez and B. E. Granger, Computing in Science Engineering **9**, 21–29 (2007).

[167] H. Nguyen, D. A. Case, and A. S. Rose, Bioinformatics **34**, 1241–1242 (2018).

[168] conda-forge community, *The conda-forge project: community-based software distribution built on the conda package format and ecosystem*, 2015.

[169] B. Grüning, R. Dale, A. Sjödin, B. A. Chapman, J. Rowe, C. H. Tomkins-Tinch, R. Valieris, J. Köster, and T. B. Team, Nature Methods **15**, 475–476 (2018).

[170] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, in 2010 ieee 26th symposium on mass storage systems and technologies (msst) (2010), pp. 1–10.

[171] T. H. Group, *Hierarchical data format, version 5*, http://www.hdfgroup.org/HDF5/, 1997.

[172] P. de Buyl, P. H. Colberg, and F. Höfling, Computer Physics Communications **185**, 1546–1553 (2014).

[173] T. Uchdorf, MA thesis (FH Jülich, 2008).

[174] U. Aydin, "Interstitial solution enthalpies derived from first-principles : knowledge discovery using high-throughput databases", PhD thesis (Universität Paderborn, 2016).

[175] J. Janssen, S. Surendralal, Y. Lysogorskiy, M. Todorova, T. Hickel, R. Drautz, and J. Neugebauer, Computational Materials Science **163**, 24–36 (2019).

[176] E. S. Raymond, *The art of unix programming* (Addison-Wesley Professional, 2003).

[177] M. Atkinson and R. Morrison, The VLDB Journal **4**, 319–401 (1995).

[178] W. McKinney, in Proceedings of the 9th python in science conference, edited by S. van der Walt and J. Millman (2010), pp. 51–56.

[179] *Pyiron*, `http://pyiron.org`.

[180] P. Jupyter, M. Bussonnier, J. Forde, J. Freeman, B. Granger, T. Head, C. Holdgraf, K. Kelley, G. Nalvarte, A. Osheroff, M. Pacer, Y. Panda, F. Perez, B. Ragan-Kelley, and C. Willing, in Proceedings of the 17th Python in Science Conference, edited by F. Akici, D. Lippa, D. Niederhut, and M. Pacer (2018), pp. 113–120.

[181] G. Kresse, *Vasp manual*, https://www.vasp.at, 2021.

[182] K. Lejaeghere, L. Vanduyfhuys, T. Verstraelen, V. V. Speybroeck, and S. Cottenier, Computational Materials Science **117**, 390–396 (2016).

[183] *Mendeleev – a python resource for properties of chemical elements, ions and isotopes, ver. 0.3.6*, 2014.

[184] I. Wolfram Research, *Mathematica, Version 12.1*, Champaign, IL, 2020.

[185] D. Gupta, in *Diffusion processes in advanced technological materials*, 1st ed. (Springer-Verlag Berlin Heidelberg, 2005), p. 532.

[186] K. Sankaran, S. Clima, M. Mees, C. Adelmann, Z. Tökei, and G. Pourtois, in Ieee international interconnect technology conference (2014), pp. 193–196.

[187] H. Frost and M. Ashby, *Deformation-mechanism maps: the plasticity and creep of metals and ceramics* (Elsevier Science Limited, 1982).

[188] G. Grimvall and S. Sjödin, Physica Scripta **10**, 340–352 (1974).

[189] L.-F. Zhu, J. Janssen, S. Ishibashi, F. Körmann, B. Grabowski, and J. Neugebauer, Computational Materials Science **187**, 110065 (2021).

[190] B.-J. Lee, W.-S. Ko, H.-K. Kim, and E.-H. Kim, Calphad **34**, 510–522 (2010).

[191] E. Maras, O. Trushin, A. Stukowski, T. Ala-Nissila, and H. Jónsson, Computer Physics Communications **205**, 13–21 (2016).

[192] E. Wigner and F. Seitz, Phys. Rev. **43**, 804–810 (1933).

[193] K. Ozaki, S. Fukutani, and K. Honda, JSME International Journal Series A Solid Mechanics and Material Engineering **44**, 199–206 (2001).

[194] G. P. P. Pun and Y. Mishin, Phys. Rev. B **95**, 224103 (2017).

[195] J. Lian, S.-W. Lee, L. Valdevit, M. I. Baskes, and J. R. Greer, Scripta Materialia **68**, 261–264 (2013).

[196] A. P. Sutton and J. Chen, Philosophical Magazine Letters **61**, 139–146 (1990).

[197] S.-N. Luo, T. J. Ahrens, T. Ça ğ ın, A. Strachan, W. A. Goddard, and D. C. Swift, Phys. Rev. B **68**, 134206 (2003).

[198] S.-N. Luo, A. Strachan, and D. C. Swift, The Journal of Chemical Physics **120**, 11640–11649 (2004).

[199] H. I. Sözen, S. Ener, F. Maccari, K. P. Skokov, O. Gutfleisch, F. Körmann, J. Neugebauer, and T. Hickel, Phys. Rev. Materials **3**, 084407 (2019).

[200] R. Freitas, R. E. Rudd, M. Asta, and T. Frolov, Phys. Rev. Materials **2**, 093603 (2018).

[201] C. J. Bechtoldt and H. C. Vacher, Transactions of the Metallurgical Society of AIME **221**, 14–18 (1961).

[202] M. J. Collins, Materials Science and Technology **4**, 560–561 (1988).

[203] W. a. Dench, Transactions of the Faraday Society **59**, 1279 (1963).

[204] E. R. Jette, V. H. Nordstrom, B. Queneau, and F. Foote, Trans. AIME **111**, 361–373 (1934).

[205] C. H. M. Jenkins, E. H. Bucknall, C. R. Austin, and G. A. Mellor, Journal of the Iron and Steel Institute **136** (1937).

[206] L. Karmazin, Materials Science and Engineering **54**, 247–256 (1982).

[207] V. N. Svechnikov and V. M. Pan, Sb. Nauchn. Rabot. Inst. Metallofiz., Akad. Nauk. Ukr. SSR. **15**, 164–178 (1962).

[208] A. Taylor and R. W. Floyd, Journal of the Institute of Metals **80**, 577–587 (1952).

[209] Q. Zhang and J.-C. Zhao, Journal of Alloys and Compounds **604**, 142–150 (2014).

[210] B. Bocklund, R. Otis, A. Egorov, A. Obaied, I. Roslyakova, and Z.-K. Liu, MRS Communications **9**, 618–627 (2019).

[211] S. Surendralal, M. Todorova, M. W. Finnis, and J. Neugebauer, Phys. Rev. Lett. **120**, 246801 (2018).

[212] S. Surendralal, M. Todorova, and J. Neugebauer, Phys. Rev. Lett. **126**, 166802 (2021).

[213] O. Hegde, M. Grabowski, X. Zhang, O. Waseda, T. Hickel, C. Freysoldt, and J. Neugebauer, Phys. Rev. B **102**, 144101 (2020).

[214] F. Lochner, I. M. Eremin, T. Hickel, and J. Neugebauer, Phys. Rev. B **103**, 054506 (2021).

[215] A. Ferrari, M. Schröder, Y. Lysogorskiy, J. Rogal, M. Mrovec, and R. Drautz, Modelling and Simulation in Materials Science and Engineering **27**, 085008 (2019).

[216] H. Zhao, L. Huber, W. Lu, N. J. Peter, D. An, F. De Geuser, G. Dehm, D. Ponge, J. Neugebauer, B. Gault, and D. Raabe, Phys. Rev. Lett. **124**, 106102 (2020).

[217] M. Kühbach, P. Bajaj, H. Zhao, M. H. Celik, E. A. Jägle, and B. Gault, npj Computational Materials **7**, `10.1038/s41524-020-00486-1` (2021).

[218] L. Koschmieder, R. Altenfeld, J. Eiken, B. Böttger, and G. J. Schmitz, Education Sciences **11**, `10.3390/educsci11010005` (2021).

List of Publications

1. L.F. Zhu, J. Janssen, S. Ishibashi, F. Körmann, B. Grabowski and J. Neugebauer, “A fully automated approach to calculate the melting temperature of elemental crystals”, Comp. Mat. Sci. **187**, 110065 (2021).
2. T.D. Swinburne, J. Janssen, M. Todorova, G. Simpson, P. Plechac, M. Luskin, and J. Neugebauer, “Anharmonic free energy of lattice vibrations in fcc crystals from a mean field bond”, Phys. Rev. B **102**, 100101(R) (2020).
3. J. Janssen, S. Surendralal, Y. Lysogorskiy, M. Todorova, T. Hickel, R. Drautz and J. Neugebauer, “pyiron: an integrated development environment for computational materials science”, Comp. Mat. Sci. **161**, 24 (2019).
4. Y. Lysogorskiy, T. Hammerschmidt, J. Janssen, J. Neugebauer and R. Drautz, “Transferability of interatomic potentials for molybdenum and silicon”, Model. Simul. Mater. Sci. Eng. **27**, 2 (2019).
5. J. Janssen, N. Gunkelmann and H. M. Urbassek, “Influence of C concentration on elastic moduli of α'-Fe(1-x)Cx alloys” Phil. Mag. **96**, 1448 (2016).