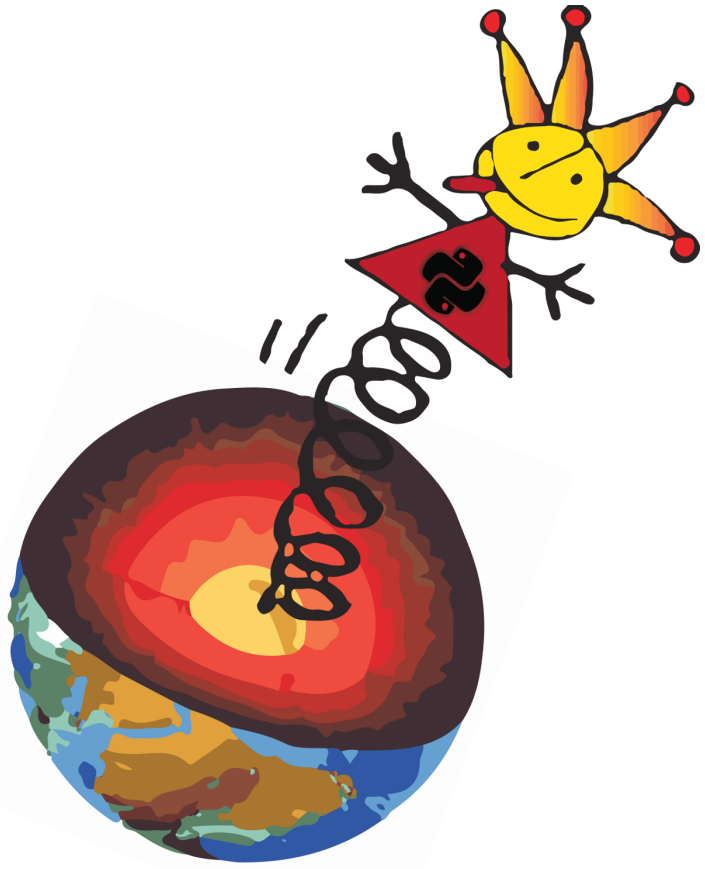
COMPUTATIONAL INFRASTRUCTURE FOR GEODYNAMICS (CIG)

BurnMan

a lower mantle mineral physics toolkit

User Manual

Version 0.8b1



Sanne Cottaar
Timo Heister
Robert Myhill
Ian Rose
Cayman Unterborn

http://geodynamics.org

CHAPTER

ONE

BURNMAN

BurnMan is an open source mineral physics toolbox written in Python which determines seismic velocities for the lower mantle. BurnMan calculates the isotropic thermoelastic moduli by solving the equations-of-state for a mixture of minerals defined by the user. The user may select from a list of minerals applicable to the lower mantle included or easily define one of their own.

Features:

- form composites of arbitrary combination of *Minerals*
- extensive *Mineral database*
- easy plotting and comparison of seismic profiles using matplotlib
- many examples highlighting different features of BurnMan
- different thermoelastic models, choice between second or third order accuracy
- different averaging schemes
- different geotherms
- extensible: all parts can be replaced by user-written modules if desired

Please cite:

- Cottaar S., Heister, T., Rose, I., and Unterborn, C., 2014, BurnMan: A lower mantle mineral physics toolkit, Geochemistry, Geophysics, and Geosystems, 15(4), 1164-1179 (link)

Acknowledgement and Support:

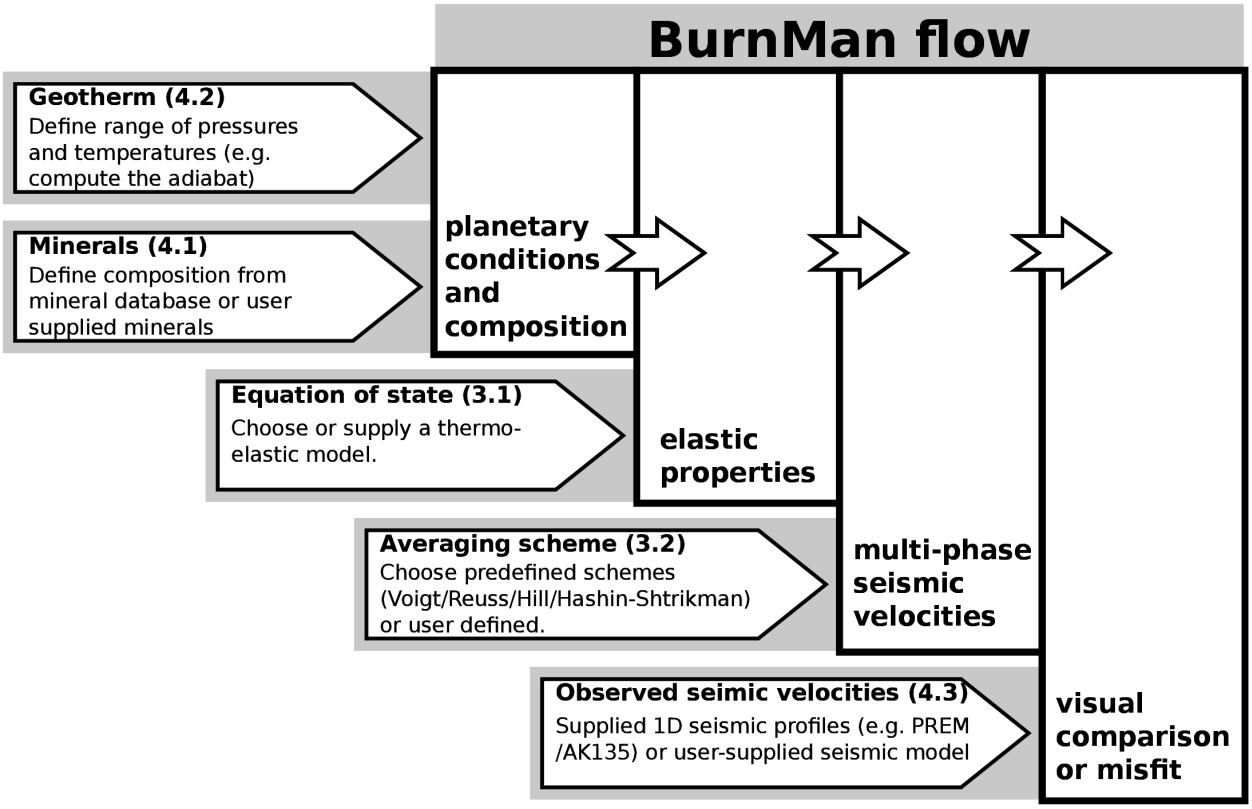
- This project was initiated at, and follow-up research support was received through, Cooperative Institute of Deep Earth Research, CIDER (NSF FESD grant 1135452) – see www.deep-earth.org
- We thank all the fellow members of the CIDER Mg/Si team for their input: Valentina Magni, Yu Huang, JiaChao Liu, Marc Hirschmann, and Barbara Romanowicz.
- We thank Lars Stixrude for providing benchmarking calculations.
- We thank CIG (www.geodynamics.org) for support and accepting our donation of BurnMan as an official project.
- We also welcomed helpful discussions with Zack Geballe, Motohiko Murakami, Bill McDonough, Quentin Williams, Wendy Panero, and Wolfgang Bangerth.

CHAPTER

TWO

OVERVIEW

2.1 Overall Structure



The goal of BurnMan is to calculate seismic velocity profiles for a given mineral composition, geotherm, EoS, and averaging scheme. These calculated seismic velocity profiles can then be compared (either graphically or quantitatively) to profiles computed for other compositions or constrained by seismology. It is written in the Python language and is run from the command line. This allows the library to be incorporated into other projects. BurnMan makes extensive use of SciPy and NumPy, which are widely used Python libraries for scientific computation. Matplotlib is used to display results and produce publication quality figures. The computations are consistently done in SI units, although for this paper we convert units for plotting purposes. A large collection of annotated examples on the usage of BurnMan are provided. Scripts to reproduce the figures in this paper are included in the toolbox. We are happy to accept contributions in form of corrections, examples, or new features.

The figure above shows each of the various steps in BurnMan and the input required at each step. The user sets the composition and temperature and pressure profiles. At later steps, the user can pick from several existing methodologies or supply an alternative implementation. This makes BurnMan very extensible and allows for many combinations and configurations with which to run calculations. Eventually, one can choose or provide a seismic model for comparison.

This flow setup can be used to evaluate the isotropic seismic velocities in a geodynamic model or as the forward problem when inverting seismic profiles or seismic velocity variations for mineralogical compositions and temperature. The modular components of BurnMan can also be used separately or combined in different ways than shown in the figure above. For example, one can input experimental results at certain pressures and temperatures and fit reference elastic moduli for a specific EoS. Additionally, one can implement their own alternatives for each of the existing modules. BurnMan has the potential to expand to other planetary applications. While all the features are modular, everything is available in a single library.

CHAPTER

THREE

METHODS

Here is a bit of background on the currently implemented methods in BurnMan. More detail can be found in the cited papers.

3.1 Calculating Thermoelastic Properties

To calculate the bulk (K) modulus, shear modulus (G) and density (ρ) of a material at a given pressure (P) and temperature (T), optionally defined by a geotherm) and determine the seismic velocities (V_S, V_P, V_Φ), one uses an Equation of State (EoS). Currently the following EoSs are supported in BurnMan: the Birch-Murnaghan formulation (excludes temperature effects, [Poi91]), and the Birch-Murnaghan formulation with a Mie-Grüneisen-Debye temperature correction as formulated by [SLB05]. To calculate these thermoelastic parameters, the EoS requires the user to input three parameters: pressure, temperature, the phases and their molar fractions. These inputs and outputs are further discussed in Section *User input*.

3.1.1 Isothermal calculations: Birch-Murnaghan

The Birch-Murnaghan equation is an isothermal Eulerian finite-strain EoS relating pressure and volume. The negative finite-strain (or compression) is defined as

$$f = \frac{1}{2}\left[\left(\frac{V}{V_0}\right)^{-2/3} - 1\right], \tag{3.1}$$

where V is the volume at a given pressure and V_0 is the volume at a reference state ($P = 10^5$ Pa , T = 300 K). The pressure and elastic moduli are derived from a third-order Taylor expansion of Helmholtz free energy in f and evaluating the appropriate volume and strain derivatives (e.g., [Poi91]). For an isotropic material one obtains for the pressure, isothermal bulk modulus, and shear modulus:

$$P = 3K_0 f\left(1+2f\right)^{5/2}\left[1 + \frac{3}{2}\left(K_0' - 4\right)f\right], \tag{3.2}$$

$$\begin{aligned} K_T = (1+2f)^{5/2}\Big[&K_0 + (3K_0K_0' - 5K_0)f \\ &+ \frac{27}{2}(K_0K_0' - 4K_0)f^2\Big], \end{aligned} \tag{3.3}$$

$$G = (1+2f)^{5/2}\Big[G_0 + (3K_0G_0' - 5G_0)f + (6K_0G_0' - 24K_0 - 14G_0 + \frac{9}{2}K_0K_0')f^2\Big]. \tag{3.4}$$

Here K_0 and G_0 are the reference bulk modulus and shear modulus and K_0' and G_0' are the derivative of the respective moduli with respect to pressure.

BurnMan has the option to use the second-order expansion for shear modulus by dropping the f^2 terms in these equations (as is sometimes done for experimental fits or EoS modeling).

3.1.2 Thermal Corrections

Thermal corrections for pressure, and isothermal bulk modulus and shear modulus are derived from the Mie-Grüneisen-Debye EoS with the quasi-harmonic approximation. Here we adopt the formalism of [SLB05] where these corrections are added to equations (3.2)–(3.4):

The Δ refers to the difference in the relevant quantity from the reference temperature (300 K). γ is the Grüneisen parameter, q is the logarithmic volume derivative of the Grüneisen parameter, η_S is the shear strain derivative of the Grüneisen parameter, C_V is the heat capacity at constant volume, and $\mathcal{U}$ is the internal energy at temperature T. C_V and $\mathcal{U}$ are calculated using the Debye model for vibrational energy of a lattice. These quantities are calculated as follows:

$$\begin{aligned}
C_V &= 9nR\left(\frac{T}{\theta}\right)^3 \int_0^{\frac{\theta}{T}} \frac{e^\tau \tau^4}{(e^\tau - 1)^2} d\tau, \\
\mathcal{U} &= 9nRT\left(\frac{T}{\theta}\right)^3 \int_0^{\frac{\theta}{T}} \frac{\tau^3}{(e^\tau - 1)} d\tau, \\
\gamma &= \frac{1}{6}\frac{\nu_0^2}{\nu^2}(2f+1)\left[a_{ii}^{(1)} + a_{iikk}^{(2)} f\right], \\
q &= \frac{1}{9\gamma}\left[18\gamma^2 - 6\gamma - \frac{1}{2}\frac{\nu_0^2}{\nu^2}(2f+1)^2 a_{iikk}^{(2)}\right], \\
\eta_S &= -\gamma - \frac{1}{2}\frac{\nu_0^2}{\nu^2}(2f+1)^2 a_S^{(2)}, \\
\frac{\nu^2}{\nu_0^2} &= 1 + a_{ii}^{(1)} f + \frac{1}{2} a_{iikk}^{(2)} f^2, \\
a_{ii}^{(1)} &= 6\gamma_0, \\
a_{iikk}^{(2)} &= -12\gamma_0 + 36\gamma_0^2 - 18 q_0 \gamma_0, \\
a_S^{(2)} &= -2\gamma_0 - 2\eta_{S0},
\end{aligned}$$

where θ is the Debye temperature of the mineral, ν is the frequency of vibrational modes for the mineral, n is the number of atoms per formula unit (e.g. 2 for periclase, 5 for perovskite), and R is the gas constant. Under the approximation that the vibrational frequencies behave the same under strain, we may identify $\nu/\nu_0 = \theta/\theta_0$. The quantities γ_0, η_{S0} q_0, and θ_0 are the experimentally determined values for those parameters at the reference state.

Due to the fact that a planetary mantle is rarely isothermal along a geotherm, it is more appropriate to use the adiabatic bulk modulus K_S instead of K_T, which is calculated using

$$K_S = K_T(1 + \gamma\alpha T), \tag{3.5}$$

where α is the coefficient of thermal expansion:

$$\alpha = \frac{\gamma C_V V}{K_T}. \tag{3.6}$$

There is no difference between the isothermal and adiabatic shear moduli for an isotropic solid. All together this makes an eleven parameter EoS model, which is summarized in the Table below. For more details on the EoS, we refer readers to [SLB05].

User Input	Symbol	Definition	Units
V_0	V_0	**Volume at P =** 10^5 Pa , T = 300 K	m ^{3} mol $^{-1}$
K_0	K_0	Isothermal bulk modulus at *P=10^5* Pa, T = 300 K	Pa
Kprime_0	K'_0	Pressure derivative of K_0	
G_0	G_0	Shear modulus at P = 10^5 Pa, T = 300 K	Pa
Gprime_0	G'_0	Pressure derivative of G_0	
molar_mass	μ	mass per mole formula unit	kg mol^{-1}
n	n	number of atoms per formula unit	
Debye_0	θ_0	Debye Temperature	K
grueneisen_0	γ_0	Grüneisen parameter at P = 10^5 Pa, T = 300 K	
q0	q_0	Logarithmic volume derivative of the Grüneisen parameter	
eta_s_0	η_{S0}	Shear strain derivative of the Grüneisen parameter	

3.2 Calculating multi-phase seismic velocities

3.2.1 Averaging schemes

After the thermoelastic parameters (K_S, G, ρ) of each phase are determined at each pressure and/or temperature step, these values must be combined to determine the seismic velocity of a multiphase assemblage. We define the volume fraction of the individual minerals in an assemblage:

$$\nu_i = n_i \frac{V_i}{V},$$

where V_i and n_i are the molar volume and the molar fractions of the i th individual phase, and V is the total molar volume of the assemblage:

$$V = \sum_i n_i V_i. \tag{3.7}$$

The density of the multiphase assemblage is then

$$\rho = \sum_i \nu_i \rho_i = \frac{1}{V} \sum_i n_i \mu_i, \tag{3.8}$$

where ρ_i is the density and μ_i is the molar mass of the i th phase.

Unlike density and volume, there is no straightforward way to average the bulk and shear moduli of a multiphase rock, as it depends on the specific distribution and orientation of the constituent minerals. BurnMan allows several schemes for averaging the elastic moduli: the Voigt and Reuss bounds, the Hashin-Shtrikman bounds, the Voigt-Reuss-Hill average, and the Hashin-Shtrikman average [WDOConnell76].

The Voigt average, assuming constant strain across all phases, is defined as

$$X_V = \sum_i \nu_i X_i, \tag{3.9}$$

where X_i is the bulk or shear modulus for the i th phase. The Reuss average, assuming constant stress across all phases, is defined as

$$X_R = \left(\sum_i \frac{\nu_i}{X_i} \right)^{-1}. \tag{3.10}$$

The Voigt-Reuss-Hill average is the arithmetic mean of Voigt and Reuss bounds:

$$X_{VRH} = \frac{1}{2} \left(X_V + X_R \right). \tag{3.11}$$

The Hashin-Shtrikman bounds make an additional assumption that the distribution of the phases is statistically isotropic and are usually much narrower than the Voigt and Reuss bounds [WDOConnell76]. This may be a poor assumption in regions of Earth with high anisotropy, such as the lowermost mantle, however these bounds are more physically motivated than the commonly-used Voigt-Reuss-Hill average. In most instances, the Voigt-Reuss-Hill average and the arithmetic mean of the Hashin-Shtrikman bounds are quite similar with the pure arithmetic mean (linear averaging) being well outside of both.

It is worth noting that each of the above bounding methods are derived from mechanical models of a linear elastic composite. It is thus only appropriate to apply them to elastic moduli, and not to other thermoelastic properties, such as wave speeds or density.

3.2.2 Computing seismic velocities

Once the moduli for the multiphase assemblage are computed, the compressional (P), shear (S) and bulk sound (Φ) velocities are then result from the equations:

$$V_P = \sqrt{\frac{K_S + \frac{4}{3}G}{\rho}}, \qquad V_S = \sqrt{\frac{G}{\rho}}, \qquad V_\Phi = \sqrt{\frac{K_S}{\rho}}. \tag{3.12}$$

To correctly compare to observed seismic velocities one needs to correct for the frequency sensitivity of attenuation. Moduli parameters are obtained from experiments that are done at high frequencies (MHz-GHz) compared to seismic frequencies (mHz-Hz). The frequency sensitivity of attenuation causes slightly lower velocities for seismic waves than they would be for high frequency waves. In BurnMan one can correct the calculated acoustic velocity values to those for long period seismic tomography [MA81]:

$$V_{S/P} = V_{S/P}^{\text{uncorr.}} \left(1 - \frac{1}{2}\cot(\frac{\beta\pi}{2})\frac{1}{Q_{S/P}}(\omega)\right).$$

Similar to [MBR+07], we use a β value of 0.3, which falls in the range of values of 0.2 to 0.4 proposed for the lower mantle (e.g. [KS90]). The correction is implemented for Q values of PREM for the lower mantle. As Q_S is smaller than Q_P, the correction is more significant for S waves. In both cases, though, the correction is minor compared to, for example, uncertainties in the temperature (corrections) and mineral physical parameters. More involved models of relaxation mechanisms can be implemented, but lead to the inclusion of more poorly constrained parameters, [MB07]. While attenuation can be ignored in many applications [TVV01], it might play a significant role in explaining strong variations in seismic velocities in the lowermost mantle [DGD+12].

3.3 User input

3.3.1 Mineralogical composition

A number of pre-defined minerals are included in the mineral library and users can create their own. The library includes wrapper functions to include a transition from the high-spin mineral to the low-spin mineral [LSMM13] or to combine minerals for a given iron number.

Standard minerals – The 'standard' mineral format includes a list of parameters given in the above table. Each mineral includes a suggested EoS with which the mineral parameters are derived. For some minerals the parameters for the thermal corrections are not yet measured or calculated, and therefore the corrections can not be applied. An occasional mineral will not have a measured or calculated shear moduli, and therefore can only be used to compute densities and bulk sound velocities. The mineral library is subdivided by citation. BurnMan includes the option to produce a LaTeX; table of the mineral parameters used. BurnMan can be easily setup to incorporate uncertainties for these parameters.

Minerals with a spin transition – A standard mineral for the high spin and low spin must be defined separately. These minerals are "wrapped," so as to switch from the high spin to high spin mineral at a give pressure. While not realistic, for the sake of simplicity, the spin transitions are considered to be sharp at a given pressure.

Minerals depending on Fe partitioning – The wrapper function can partition iron, for example between ferropericlase, fp, and perovskite, pv. It requires the input of the iron mol fraction with regards to Mg, X_{fp} and X_{pv}, which then defines the chemistry of an Mg-Fe solid solution according to $(\text{Mg}_{1-X_{\text{Fe}}^{\text{fp}}},\text{Fe}_{X_{\text{Fe}}^{\text{fp}}})\text{O}$ or $(\text{Mg}_{1-X_{\text{Fe}}^{\text{pv}}},\text{Fe}_{X_{\text{Fe}}^{\text{pv}}})\text{SiO}_3$. The iron mol fractions can be set to be constant or varying with P and T as needed. Alternatively one can calculate the iron mol fraction from the distribution coefficient K_D defined as

$$K_D = \frac{X_{\text{Fe}}^{\text{pv}}/X_{\text{Mg}}^{\text{pv}}}{X_{\text{Fe}}^{\text{fp}}/X_{\text{Mg}}^{\text{fp}}}. \tag{3.13}$$

We adopt the formalism of [NFR12] choosing a reference distribution coefficient K_{D0} and standard state volume change (Δv^0) for the Fe-Mg exchange between perovskite and ferropericlase

$$K_D = K_{D0} \exp\left(\frac{(P_0 - P)\Delta v^0}{RT}\right), \tag{3.14}$$

where R is the gas constant and P_0 the reference pressure. As a default, we adopt the average Δv^0 of [NFR12] of $2 \cdot 10^{-7}\ m^3 mol^{-1}$ and suggest using their K_{D0} value of 0.5.

The multiphase mixture of these minerals can be built by the user in three ways:

1. Molar fractions of an arbitrary number of pre-defined minerals, for example mixing standard minerals mg_perovskite ($MgSiO_3$), fe_perovskite ($FeSiO_3$), periclase (MgO) and wüstite (FeO).

2. A two-phase mixture with constant or (P, T) varying Fe partitioning using the minerals that include Fe-dependency, for example mixing $(Mg, Fe)SiO_3$ and $(Mg, Fe)O$ with a pre-defined distribution coefficient.

3. Weight percents (wt%) of (Mg, Si, Fe) and distribution coefficient (includes (P,T)-dependent Fe partitioning). This calculation assumes that each element is completely oxidized into its corresponding oxide mineral (MgO, FeO, SiO_2) and then combined to form iron-bearing perovskite and ferropericlase taking into account some Fe partition coefficient.

3.3.2 Geotherm

Unlike the pressure, the temperature of the lower mantle is relatively unconstrained. As elsewhere, BurnMan provides a number of built-in geotherms, as well as the ability to use user-defined temperature-depth relationships. A geotherm in BurnMan is an object that returns temperature as a function of pressure. Alternatively, the user could ignore the geothermal and compute elastic velocities for a range of temperatures at any give pressure.

Currently, we include geotherms published by [BS81] and [And82a]. Alternatively one can use an adiabatic gradient defined by the thermoelastic properties of a given mineralogical model. For a homogeneous material, the adiabatic temperature profile is given by integrating the ordinary differential equation (ODE)

$$\left(\frac{\mathrm{d}T}{\mathrm{d}P}\right)_S = \frac{\gamma T}{K_S}. \tag{3.15}$$

This equation can be extended to multiphase composite using the first law of thermodynamics to arrive at

$$\left(\frac{\mathrm{d}T}{\mathrm{d}P}\right)_S = \frac{T\sum_i \frac{n_i C_{Pi}\gamma_i}{K_{Si}}}{\sum_i n_i C_{Pi}}, \tag{3.16}$$

where the subscripts correspond to the i th phase, C_P is the heat capacity at constant pressure of a phase, and the other symbols are as defined above. Integrating this ODE requires a choice in anchor temperature (T_0) at the top of the lower mantle (or including this as a parameter in an inversion). As the adiabatic geotherm is dependent on the thermoelastic parameters at high pressures and temperatures, it is dependent on the equation of state used.

3.3.3 Seismic Models

BurnMan allows for direct visual and quantitative comparison with seismic velocity models. Various ways of plotting can be found in the examples. Quantitative misfits between two profiles include an L2-norm and a chi-squared misfit, but user defined norms can be implemented. A seismic model in BurnMan is an object that provides pressure, density, and seismic velocities (V_P, V_Φ, V_S) as a function of depth.

To compare to seismically constrained profiles, BurnMan provides the 1D seismic velocity model PREM [DA81]. One can choose to evaluate $V_P, V_\Phi, V_S, \rho, K_S$ and/or G. The user can input their own seismic profile, an example of which is included using AK135 [KEB95].

Besides standardized 1D radial profiles, one can also compare to regionalized average profiles for the lower mantle. This option accommodates the observation that the lowermost mantle can be clustered into two regions, a 'slow' region, which represents the so-called Large Low Shear Velocity Provinces, and 'fast' region, the continuous surrounding region where slabs might subduct [lekic2012]. This clustering as well as the averaging of the 1D model occurs over five tomographic S wave velocity models (SAW24B16: [MegninR00]; HMSL-S: [HMSL08]; S362ANI: [KEkstromD08]; GyPSuM: [SFBG10]; S40RTS: [RDvHW11]). The strongest deviations from PREM occur in the lowermost 1000 km. Using the 'fast' and 'slow' S wave velocity profiles is therefore most important when interpreting the lowermost mantle. Suggestion of compositional variation between these regions comes from seismology [TRCT05][HW12] as well as geochemistry [DCT12][JCK+10]. Based on thermo-chemical convection models, [SDG11] also show that averaging profiles in thermal boundary layers may cause problems for seismic interpretation.

We additionally apply cluster analysis to and provide models for P wave velocity based on two tomographic models (MIT-P08: [LvdH08]; GyPSuM: [SMJM12]). The clustering results correlate well with the fast and slow regions for S wave velocities; this could well be due to the fact that the initial model for the P wave velocity models is scaled from S wave tomographic velocity models. Additionally, the variations in P wave velocities are a lot smaller than for S waves. For this reason using these adapted models is most important when comparing the S wave velocities.

While interpreting lateral variations of seismic velocity in terms of composition and temperature is a major goal [TDRY04][MCD+12], to determine the bulk composition the current challenge appears to be concurrently fitting absolute P and S wave velocities and incorporate the significant uncertainties in mineral physical parameters).

CHAPTER

FOUR

EXAMPLES

BurnMan comes with a small tutorial in the tutorial/ folder, and large collection of example programs under examples/. Below you can find a summary of the different examples. They are grouped into *Tutorial*, *Simple Examples*, and *More Advanced Examples*. We suggest starting with the tutorial before moving on to the simpler examples, especially if you are new to using BurnMan.

Finally, we also include the scripts that were used for all computations and figures in the 2014 BurnMan paper in the misc/ folder, see *Reproducing Cottaar, Heister, Rose and Unterborn (2014)*.

4.1 Tutorial

The tutorial for BurnMan currently consists of three separate units:

- `step 1`,
- `step 2`, and
- `step 3`.

4.1.1 CIDER 2014 BurnMan Tutorial — step 1

In this first part of the tutorial we will acquaint ourselves with a basic script for calculating the elastic properties of a mantle mineralogical model.

In general, there are three portions of this script:

1) Define a set of pressures and temperatures at which we want to calculate elastic properties

2) Setup a composite of minerals (or "rock") and calculate its elastic properties at those pressures and temperatures.

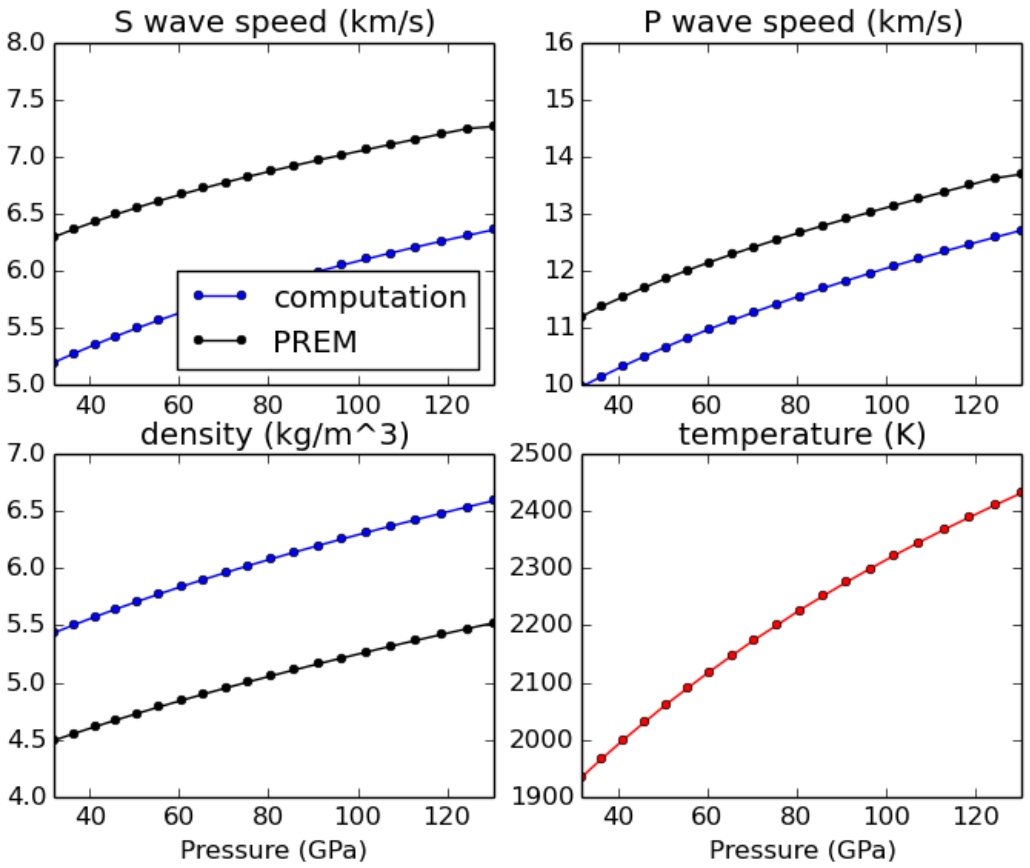
3) Plot those elastic properties, and compare them to a seismic model, in this case PREM

The script is basically already written, and should run as is by typing:

```
python step_1.py
```

on the command line. However, the mineral model for the rock is not very realistic, and you will want to change it to one that is more in accordance with what we think the bulk composition of Earth's lower mantle is.

When run (without putting in a more realistic composition), the program produces the following image:



Your goal in this tutorial is to improve this awful fit....

link to source code: tutorial/step1.py

4.1.2 CIDER 2014 BurnMan Tutorial — step 2

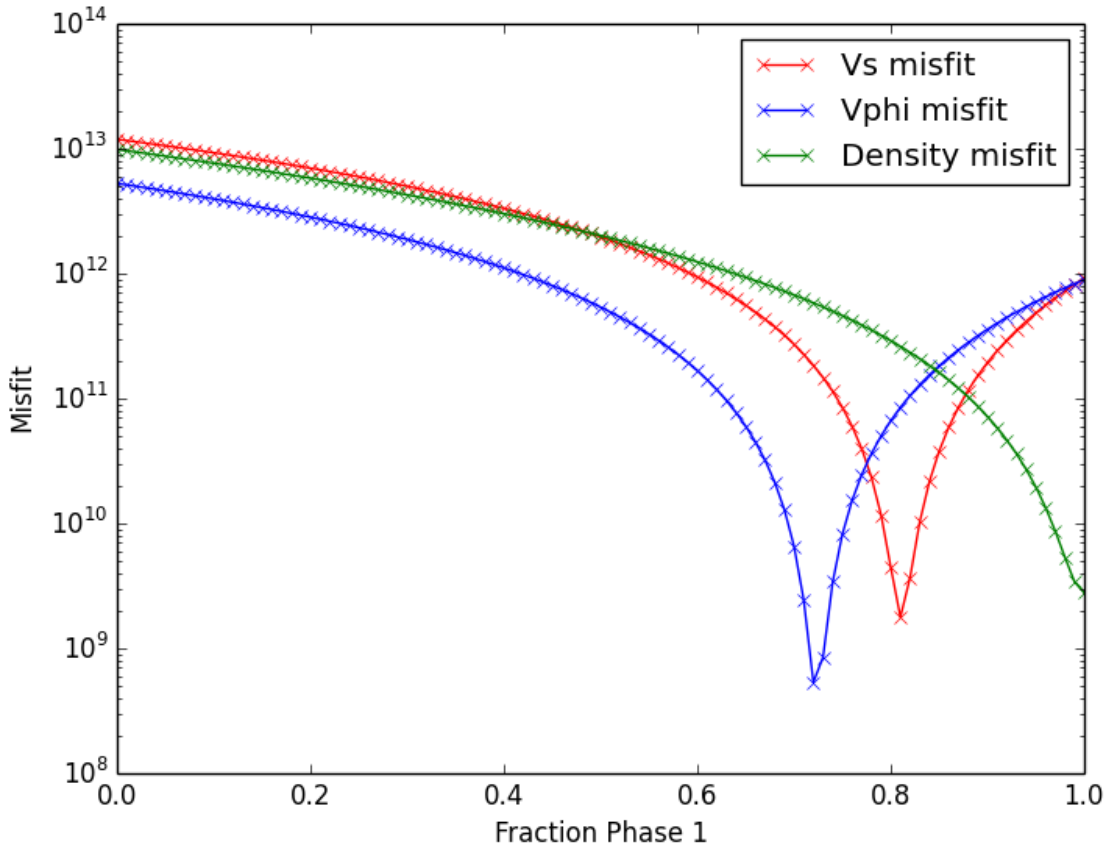
In this second part of the tutorial we try to get a closer fit to our 1D seismic reference model. In the simple Mg, Si, and O model that we used in step 1 there was one free parameter, namely phase_1_fraction, which goes between zero and one.

In this script we want to explore how good of a fit to PREM we can get by varying this fraction. We create a simple function that calculates a misfit between PREM and our mineral model as a function of phase_1_fraction, and then plot this misfit function to try to find a best model.

This script may be run by typing

    python step_2.py

Whithout changing any input, the program should produce the following image showing the misfit as a function of perovskite content:



link to source code: tutorial/step_2.py

4.1.3 CIDER 2014 BurnMan Tutorial — step 3

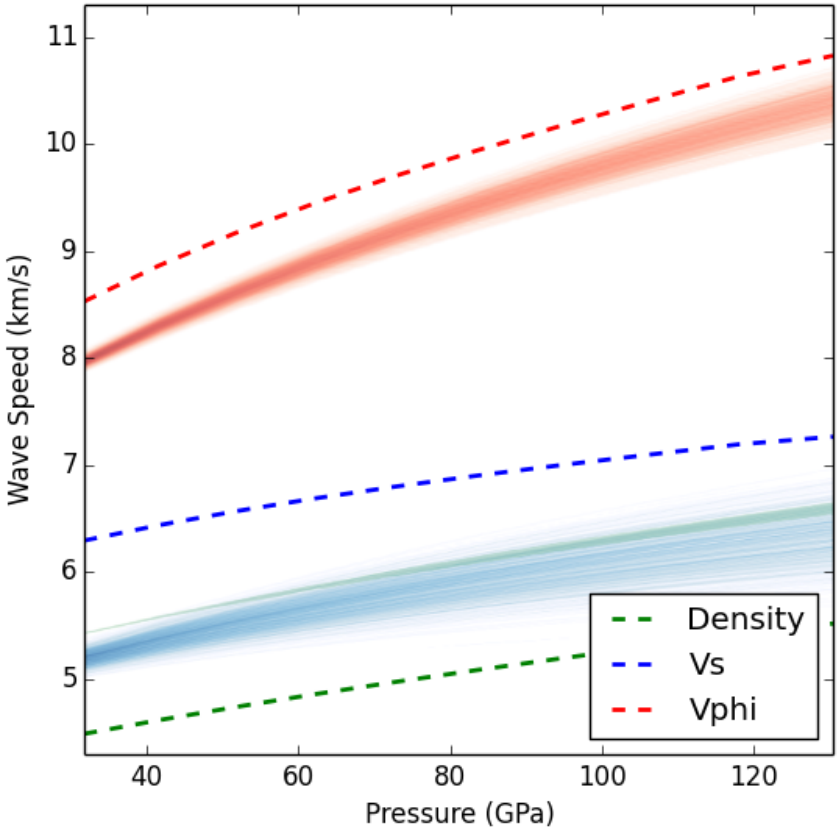
In the previous two steps of the tutorial we tried to find a very simple mineralogical model that best fit the 1D seismic model PREM. But we know that there is consideral uncertainty in many of the mineral physical parameters that control how the elastic properties of minerals change with pressure and temperature. In this step we explore how uncertainties in these parameters might affect the conclusions you draw.

The strategy here is to make many different "realizations" of the rock that you determined was the closest fit to PREM, where each realization has its mineral physical parameters perturbed by a small amount, hopefully related to the uncertainty in that parameter. In particular, we will look at how perturbations to $K_0^{'}$ and $G_0^{'}$ (the pressure derivatives of the bulk and shear modulus, respectively) change the calculated 1D seismic profiles.

This script may be run by typing

```
python step_3.py
```

After changing the standard deviations for $K_0^{'}$ and $G_0^{'}$ to 0.2, the following figure of velocities for 1000 realizations is produced:



4.2 Simple Examples

The following is a list of simple examples:

- example_beginner,
- example_geotherms,
- example_seismic,
- example_composition, and
- example_averaging.

4.2.1 example_beginner

This example script is intended for absolute beginners to BurnMan. We cover importing BurnMan modules, creating a composite material, and calculating its seismic properties at lower mantle pressures and temperatures. Afterwards, we plot it against a 1D seismic model for visual comparison.

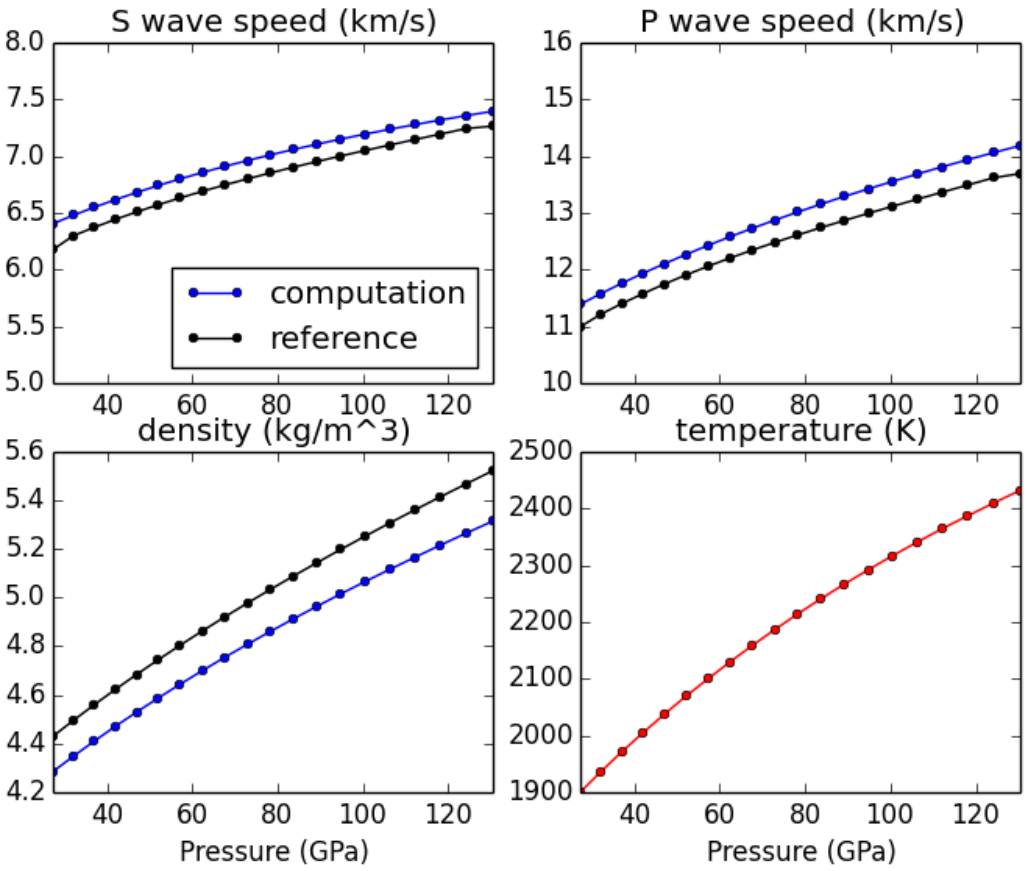
Uses:

- *Mineral database*
- `burnman.composite.Composite`
- `burnman.seismic.PREM`
- `burnman.geotherm.brown_shankland()`
- `burnman.main.velocities_from_rock()`

Demonstrates:

- creating basic composites
- calculating thermoelastic properties
- seismic comparison

Resulting figure:



4.2.2 example_geotherms

This example shows each of the geotherms currently possible with BurnMan. These are:

1. Brown and Shankland, 1981 [BS81]

2. Anderson, 1982 [And82a]
3. Watson and Baxter, 2007 [WB07]
4. linear extrapolation
5. Read in from file from user
6. Adiabatic from potential temperature and choice of mineral (pyrolite in this example)

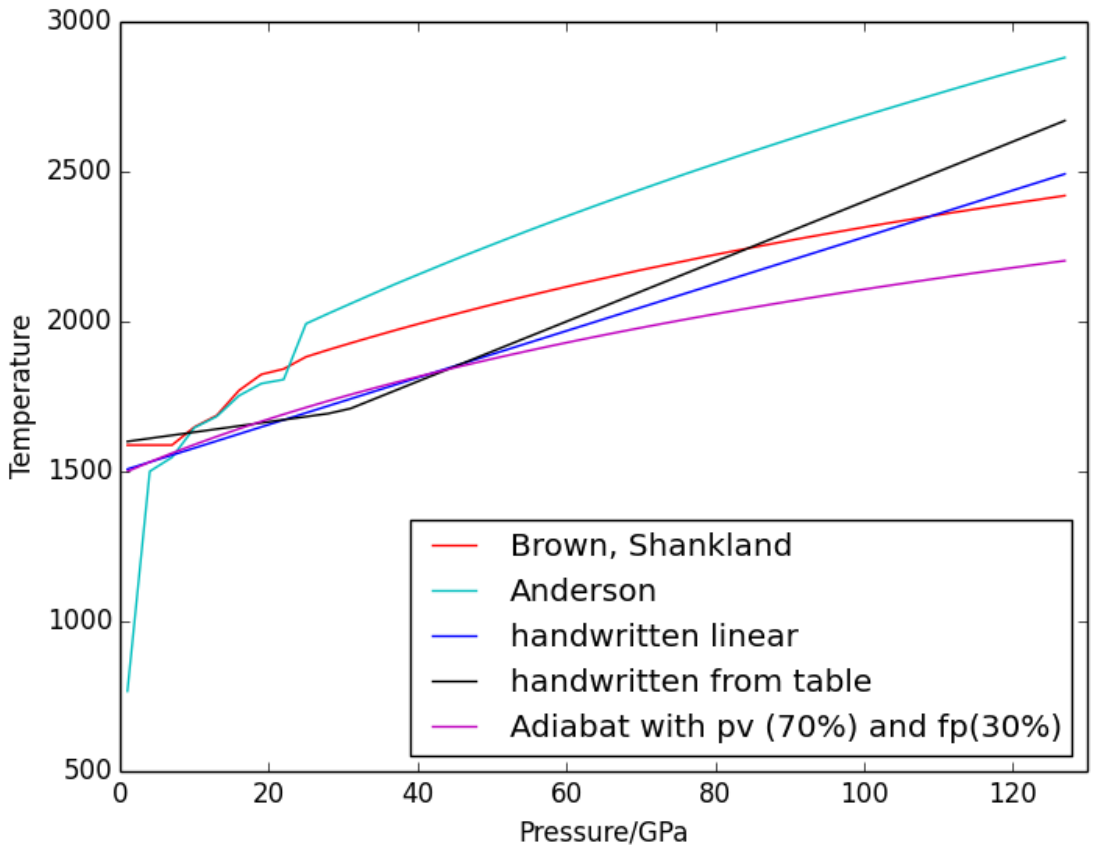
Uses:

- `burnman.geotherm.brown_shankland()`
- `burnman.geotherm.anderson()`
- input geotherm file *input_geotherm/example_geotherm.txt* (optional)
- `burnman.composite.Composite` for adiabat

Demonstrates:

- the available geotherms

Resulting figure:



4.2.3 example_seismic

Shows the various ways to input seismic models (V_s, V_p, V_ϕ, ρ) as a function of depth (or pressure) as well as different velocity model libraries available within Burnman:

1. PREM [DA81]
2. Reference model for fast regions (outside the LLSVP's) in the lower mantle [LCDR12]
3. Reference model for slow regions (LLSVP's) in the lower mantle [LCDR12]

This example will first calculate or read in a seismic model and plot the model along the defined pressure range. The example also illustrates how to import a seismic model of your choice, here shown by importing AK135 [KEB95].

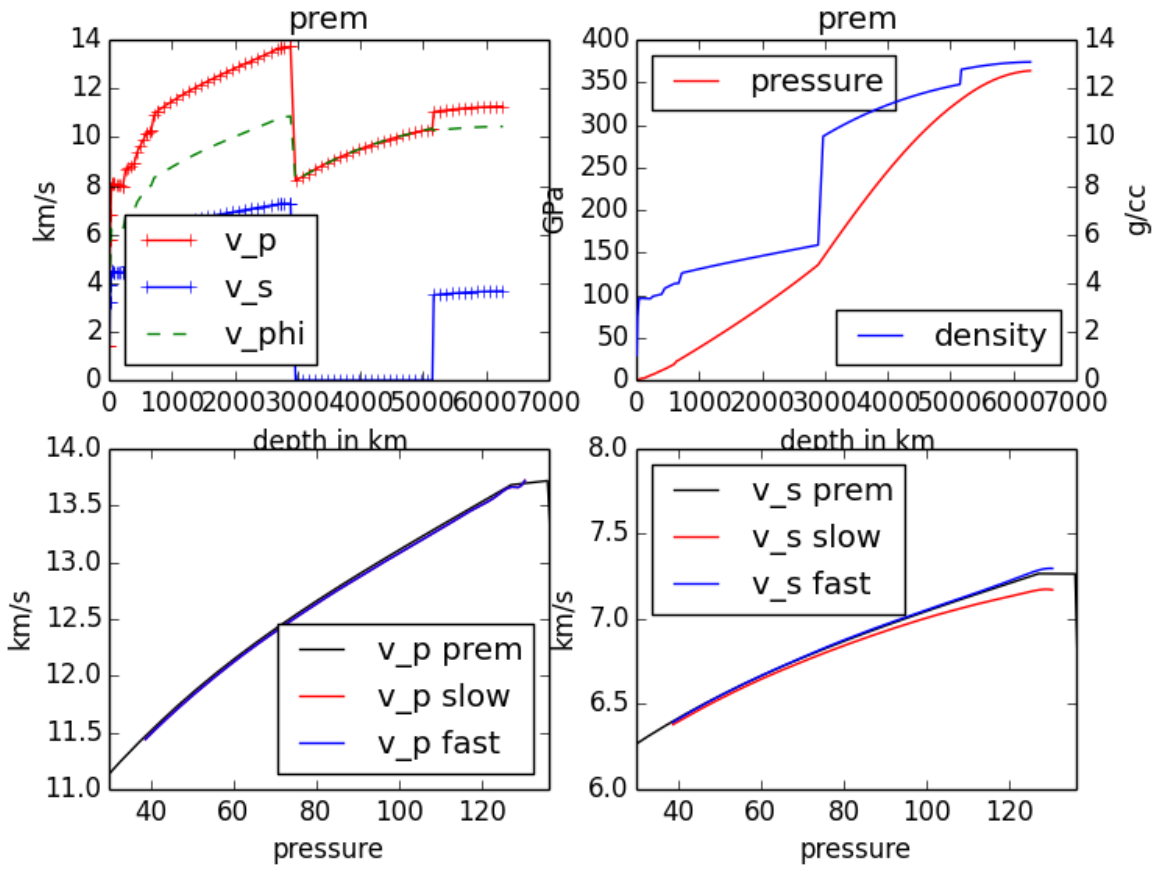
Uses:

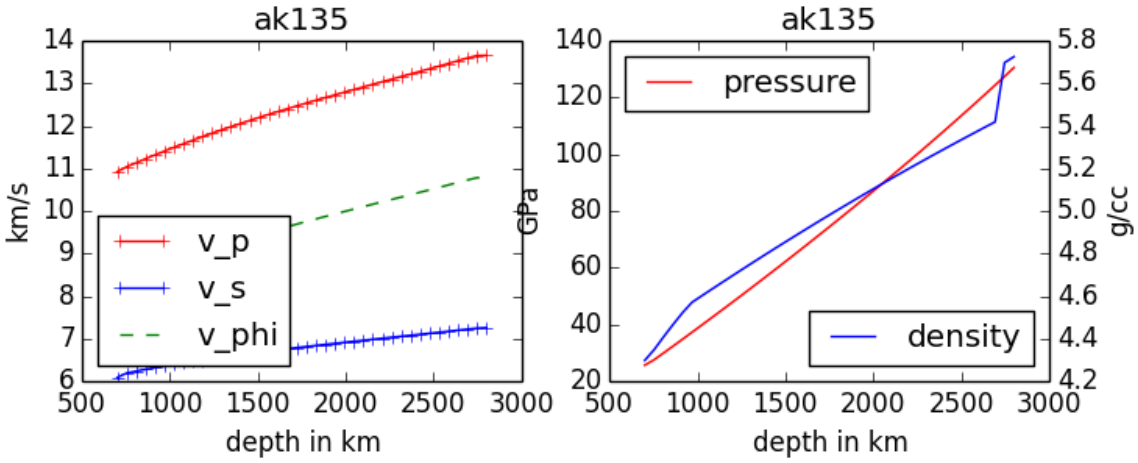
- *Seismic*

Demonstrates:

- Utilization of library seismic models within BurnMan
- Input of user-defined seismic models

Resulting figures:





4.2.4 example_composition

This example shows how to create different minerals, how to compute seismic velocities, and how to compare them to a seismic reference model.

There are many different ways in BurnMan to combine minerals into a composition. Here we present a couple of examples:

1. Two minerals mixed in simple mole fractions. Can be chosen from the BurnMan libraries or from user defined minerals (see example_user_input_material)
2. Two minerals mixed in simple mole fractions with user-defined Fe partitioning
3. The user can input wt% of each cation (Mg, Fe and Si) and BurnMan will calculate Fe partitioning along a P, T profile (see example_partition_coef.py)
4. A mixture of three minerals.

In compositions 2, 3, and 4 of the above inputs, BurnMan will mix the mineral physical paremeters of end member minerals (pure Mg and Fe) of the user's choice using either volumetric (moduli) or molar averaging (all others) at room pressure and temperature (see example_user_input_material.py for information on these parameters).

To turn a method of mineral creation "on" the first if statement above the method must be set to True, with all others set to False.

Note: These minerals can include a spin transition in (Mg,Fe)O, see example_spintransition.py for explanation of how to implement this

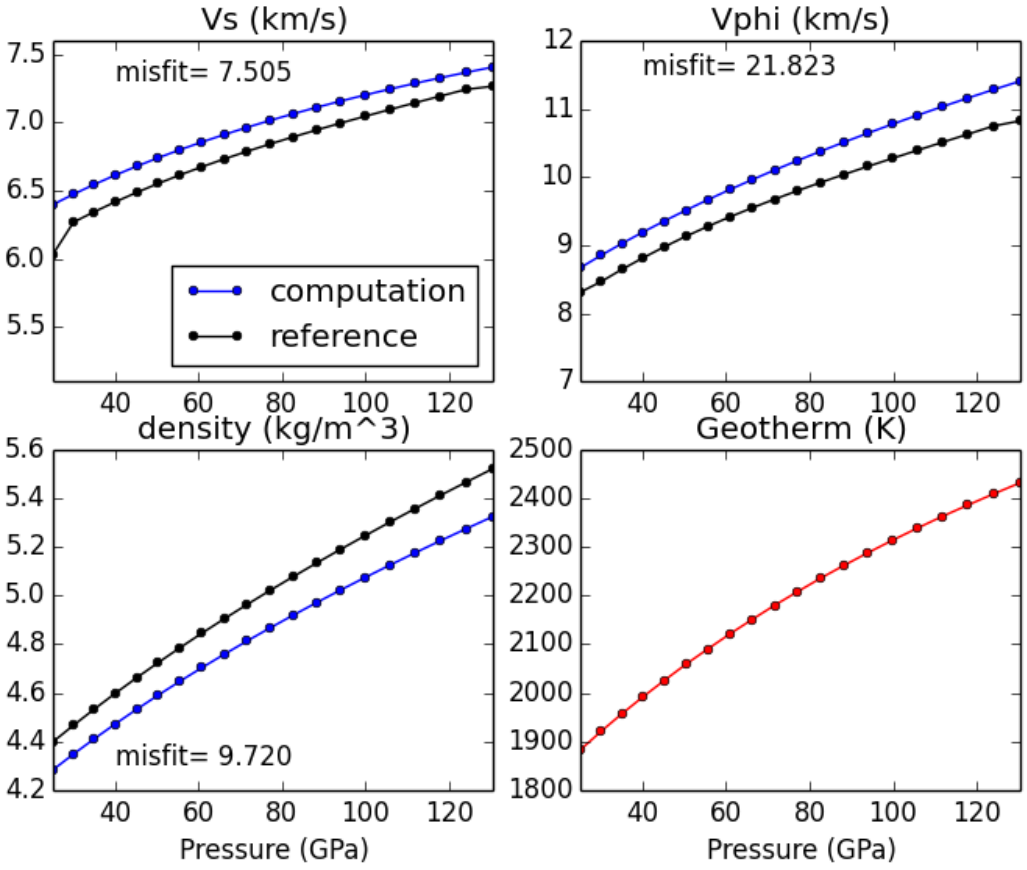
Uses:

- *Mineral database*
- `burnman.composite.Composite`

Demonstrates:

- Different ways to define a composite
- Compare computations to seismic models

Resulting figure:



4.2.5 example_averaging

This example shows the effect of different averaging schemes. Currently four averaging schemes are available:

1. Voight-Reuss-Hill
2. Voight averaging
3. Reuss averaging
4. Hashin-Shtrikman averaging

See [WDOConnell76] Journal of Geophysics and Space Physics for explanations of each averaging scheme.

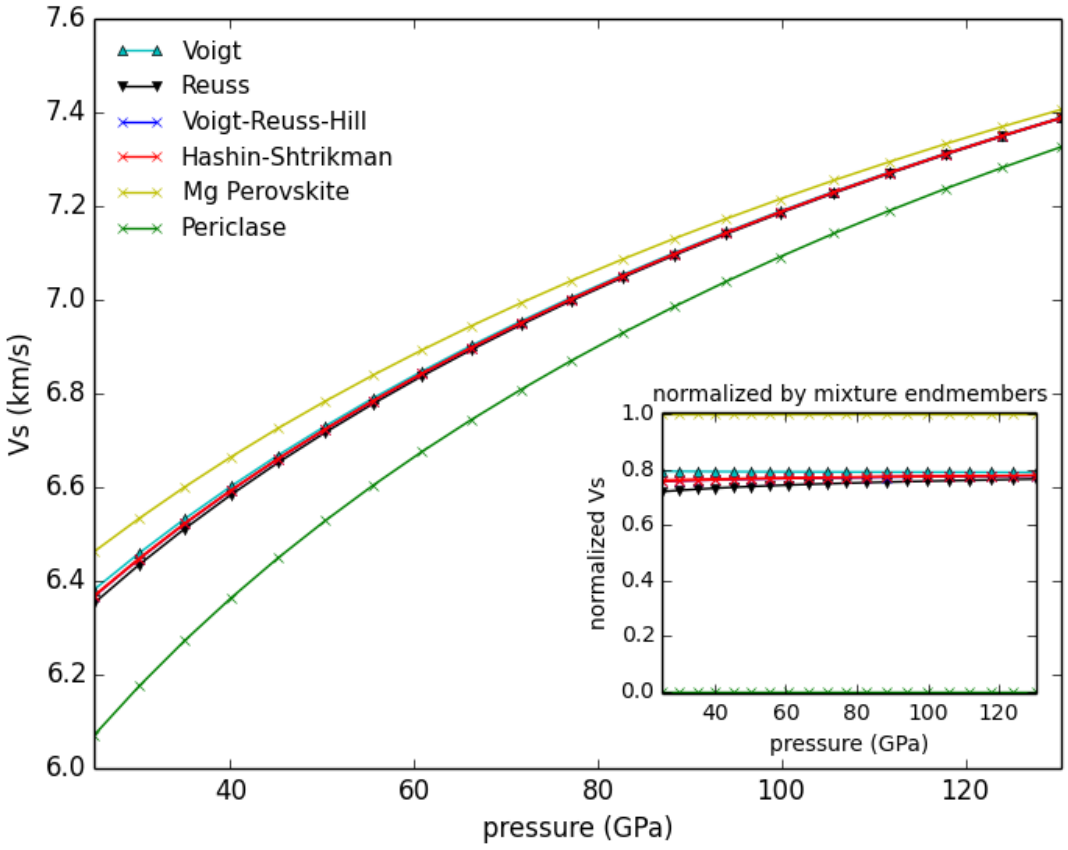
Specifically uses:

- `burnman.averaging_schemes.VoigtReussHill`
- `burnman.averaging_schemes.Voigt`
- `burnman.averaging_schemes.Reuss`
- `burnman.averaging_schemes.HashinShtrikmanUpper`
- `burnman.averaging_schemes.HashinShtrikmanLower`

Demonstrates:

- implemented averaging schemes

Resulting figure:



4.3 More Advanced Examples

Advanced examples:

- example_spintransition,
- example_user_input_material,
- example_optimize_pv, and
- example_compare_all_methods.

4.3.1 example_spintransition

This example shows the different minerals that are implemented with a spin transition. Minerals with spin transition are implemented by defining two separate minerals (one for the low and one for the high spin state). Then a third dynamic mineral is created that switches between the two previously defined minerals by comparing the current pressure to the transition pressure.

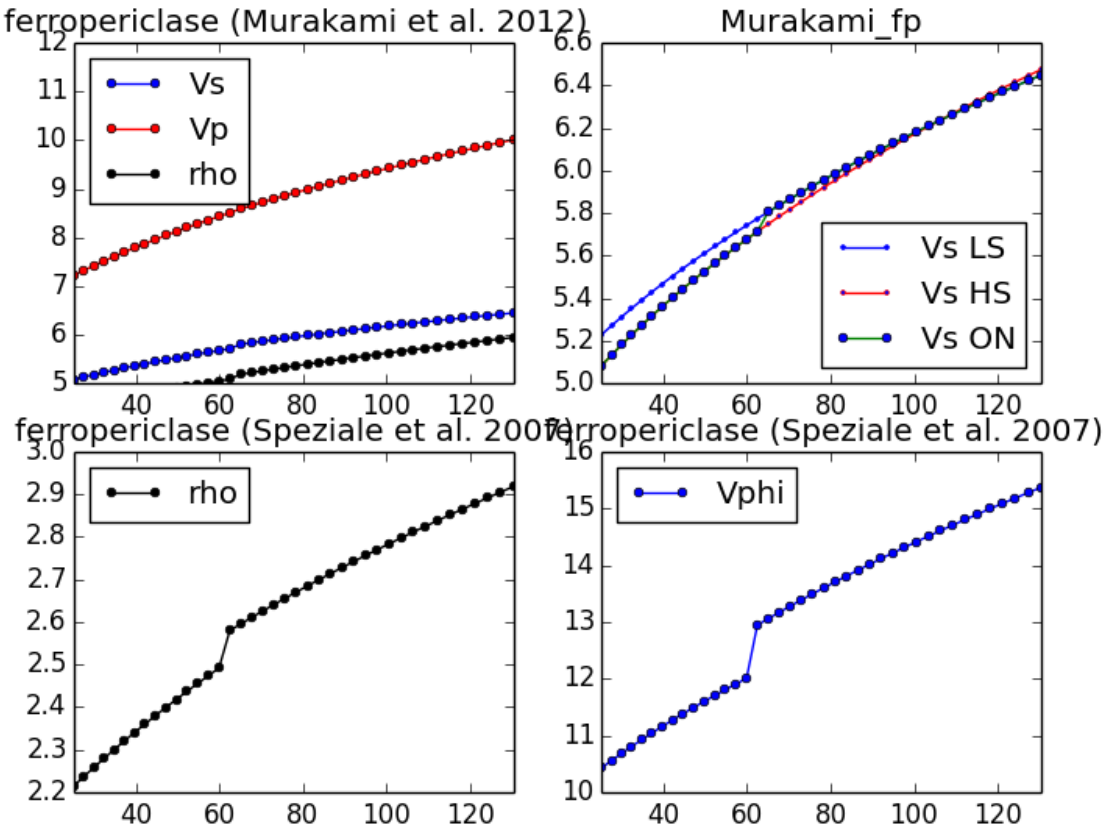
Specifically uses:

- burnman.mineral_helpers.HelperSpinTransition()
- burnman.minerals.Murakami_etal_2012.fe_periclase()
- burnman.minerals.Murakami_etal_2012.fe_periclase_HS()
- burnman.minerals.Murakami_etal_2012.fe_periclase_LS()

Demonstrates:

- implementation of spin transition in (Mg,Fe)O at user defined pressure

Resulting figure:



4.3.2 example_user_input_material

Shows user how to input a mineral of his/her choice without usint the library and which physical values need to be input for BurnMan to calculate V_P, V_Φ, V_S and density at depth.

Specifically uses:

- `burnman.mineral.Mineral`

Demonstrates:

- how to create your own minerals

4.3.3 example_optimize_pv

Vary the amount perovskite vs. ferropericlase and compute the error in the seismic data against PREM. For more extensive comments on this setup, see tutorial/step_2.py

Uses:

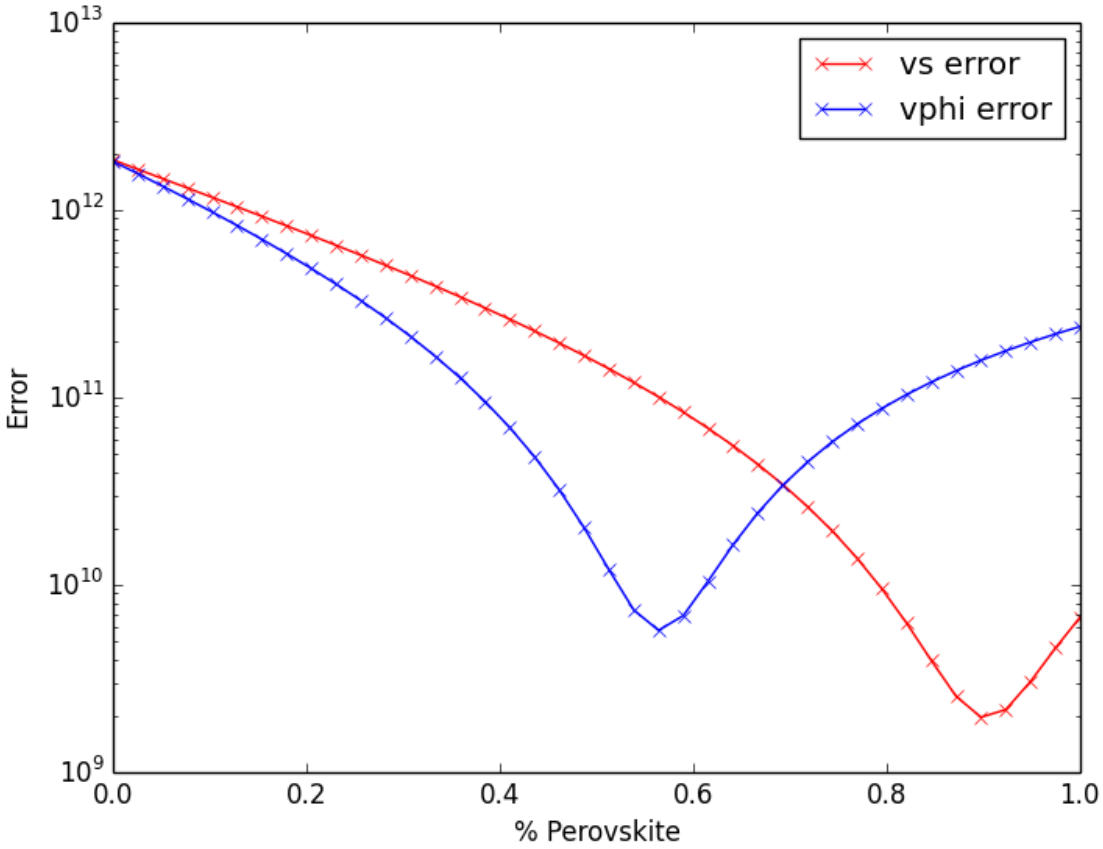
- *Mineral database*
- `burnman.composite.Composite`

- burnman.seismic.PREM
- burnman.geotherm.brown_shankland()
- burnman.main.velocities_from_rock()
- burnman.main.compare_l2()

Demonstrates:

- compare errors between models
- loops over models

Resulting figure:



4.3.4 example_compare_all_methods

This example demonstrates how to call each of the individual calculation methodologies that exist within BurnMan. See below for current options. This example calculates seismic velocity profiles for the same set of minerals and a plot of V_s, V_ϕ and ρ is produce for the user to compare each of the different methods.

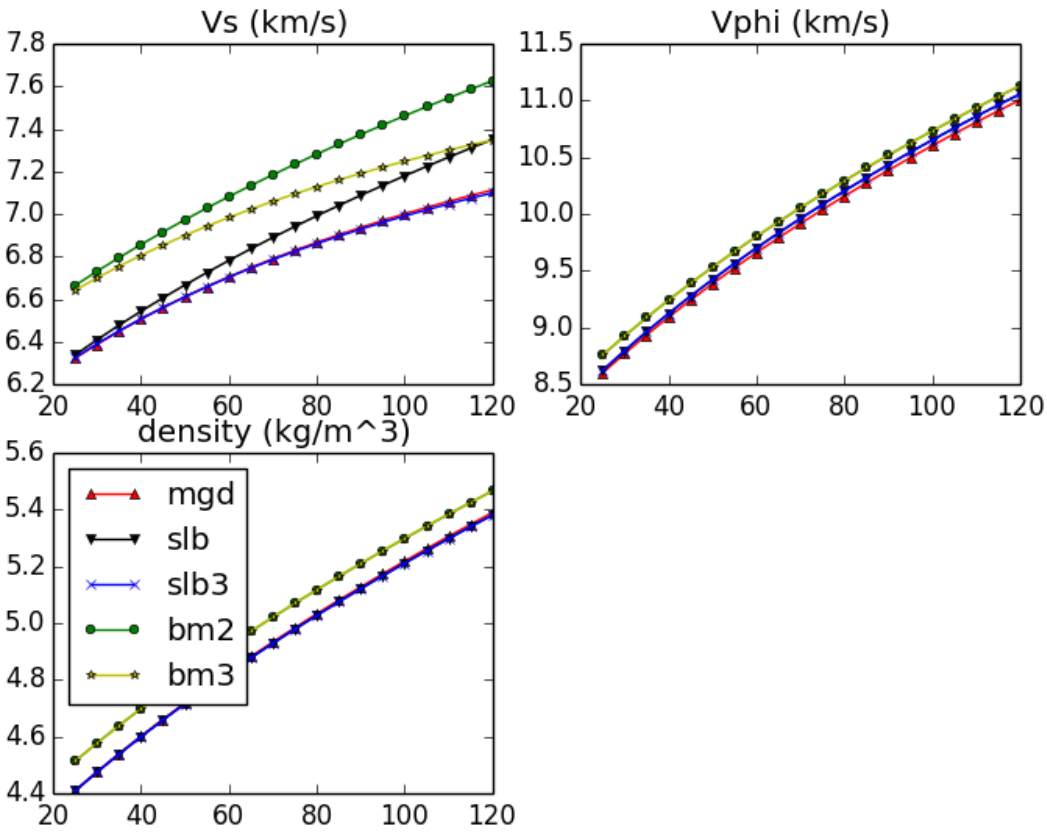
Specifically uses:

- *Equations of state*

Demonstrates:

- Each method for calculating velocity profiles currently included within BurnMan

Resulting figure:



4.4 Reproducing Cottaar, Heister, Rose and Unterborn (2014)

In this section we include the scripts that were used for all computations and figures in the 2014 BurnMan paper: Cottaar, Heister, Rose & Unterborn (2014) [CHRU14]

4.4.1 paper_averaging

This script reproduces [CHRU14], Figure 2.

This example shows the effect of different averaging schemes. Currently four averaging schemes are available: 1. Voight-Reuss-Hill 2. Voight averaging 3. Reuss averaging 4. Hashin-Shtrikman averaging

See [WDOConnell76] for explanations of each averaging scheme.

requires: - geotherms - compute seismic velocities

teaches: - averaging

4.4.2 paper_benchmark

This script reproduces the benchmark in [CHRU14], Figure 3.

4.4.3 paper_fit_data

This script reproduces [CHRU14] Figure 4.

This example demonstrates BurnMan's functionality to fit thermoelastic data to both 2nd and 3rd orders using the EoS of the user's choice at 300 K. User's must create a file with P, T and V_s. See input_minphys/ for example input files.

requires: - compute seismic velocities

teaches: - averaging

`misc.paper_fit_data.`**`calc_shear_velocities`**(*G_0*, *Gprime_0*, *mineral*, *pressures*)

`misc.paper_fit_data.`**`error`**(*guess*, *test_mineral*, *pressures*, *obs_vs*)

4.4.4 paper_incorrect_averaging

This script reproduces [CHRU14], Figure 5. Attempt to reproduce Figure 6.12 from [Mur13]

4.4.5 paper_opt_pv

This script reproduces [CHRU14], Figure 6. Vary the amount perovskite vs. ferropericlase and compute the error in the seismic data against PREM.

requires: - creating minerals - compute seismic velocities - geotherms - seismic models - seismic comparison

teaches: - compare errors between models - loops over models

4.4.6 paper_onefit

This script reproduces [CHRU14], Figure 7. It shows an example for a best fit for a pyrolitic model within mineralogical error bars.

4.4.7 paper_uncertain

This script reproduces [CHRU14], Figure 8. It shows the sensitivity of the velocities to various mineralogical parameters.

4.5 Misc or work in progress

4.5.1 example_compare_enstpyro

This example shows you how to create two materials from wt% determines the optimum mixing between the two to match the seismic model of your choice. Currently it compares two end member meteorite groups among the chondrites: carbonaceous and enstatite. Velocities are calculated for each set of minerals and plotted for comparison.

requires: - geotherms - seismic models - compute seismic velocities - creating minerals

teaches: - weight percent materials

4.5.2 example_parition_coef

This example shows how to vary the distribution coefficient of the perovskite/ferropericlase system. The user sets K_{d0} and BurnMan scales K_d as a function of P and T adopting the formalism of [NFR12]. Specifically we adopt equation 5 of [NFR12] with ΔV_0 = 0.2 cc/mol, and calculating the partition coefficient of Fe in each phase from stoichiometry.

This example will calculate mineral input parameters from Mg and Fe endmembers from Stixrude and Lithgow-bertelloni, 2005 with weighting determined by the calculated partition coefficients. Finally, output plots of X_{Fe} in pv and X_{Fe} in fp our output as well as the user's choice of geotherm

requires: - geotherms -input distribution coefficient K_{d0}

teaches: - creating varying proportions of Fe and its effect on seismic velocities

4.5.3 example_fit_data

This example demonstrates BurnMan's functionality to fit thermoelastic data to both 2nd and 3rd orders using the EoS of the user's choice at 300 K. User's must create a file with P, T and V_s. See input_minphys/ for example input files.

requires: - compute seismic velocities

teaches: - averaging

4.5.4 example_grid

This example shows how to evaluate seismic quantities on a P, T grid.

4.5.5 example_woutput

This example explains how to perform the basic i/o of BurnMan. A method of calculation is chosen, a composite mineral/material (see example_composition.py for explanation of this process) is created in the class "rock," finally a geotherm is created and seismic velocities calculated.

Post-calculation, the results are written to a simple text file to plot/manipulate at the user's whim.

requires: - creating minerals - compute seismic velocities - geotherms

teaches: - output computed seismic data to file

CHAPTER

FIVE

MAIN MODULE

class burnman.main.**ElasticProperties**(*V=None*, *rho=None*, *K=None*, *G=None*, *fraction=None*)

Class that contains volume, density, and moduli. This is generated for different pressures and temperatures using `calculate_moduli()`. Several instances of elastic_properties can be averaged using `average_moduli()`.

Variables

- **V** (*float*) – volume $[m^3]$
- **rho** (*float*) – density $[kg/m^3]$
- **K** (*float*) – bulk modulus K $[Pa]$
- **G** (*float*) – shear modulus G $[Pa]$

burnman.main.**velocities_from_rock**(*rock*, *pressures*, *temperatures*, *averaging_scheme=<burnman.averaging_schemes.VoigtReussHill instance at 0x1027e85a8>*)

A function that rolls several steps into one: given a rock and a list of pressures and temperatures, it calculates the elastic moduli of the individual phases using calculate_moduli(), averages them using average_moduli(), and calculates the seismic velocities using compute_velocities().

Parameters

- **rock** (*burnman.abstract_material*) – this is a rock
- **pressures** (*list of float*) – list of pressures you want to evaluate the rock at. $[Pa]$
- **temperatures** (*list of float*) – list of temperatures you want to evaluate the rock at. $[K]$
- **averaging_scheme** (burnman.averaging_schemes.averaging_scheme) – Averaging scheme to use.

Returns ρ $[kg/m^3]$, V_p, V_s, and V_ϕ $[m/s]$, bulk modulus K $[Pa]$,shear modulus G $[Pa]$

Return type lists of floats

burnman.main.**calculate_moduli**(*rock*, *pressures*, *temperatures*)

Given a composite and a list of pressures $[Pa]$ and temperatures $[K]$, calculate the elastic moduli and densities of the individual phases.

Parameters

- **rock** (*burnman.abstract_material*) – this is a rock
- **pressures** (*list of float*) – list of pressures you want to evaluate the rock at. $[Pa]$
- **temperatures** (*list of float*) – list of temperatures you want to evaluate the rock at. $[K]$

Returns answer – an array of (n_evaluation_points by n_phases) of elastic_properties(), so the result is of the form answer[pressure_idx][phase_idx].V

Return type list of list of `burnman.elastic_properties`

`burnman.main.`**`compute_velocities`**(*moduli*)

Given a list of elastic_properties, compute the seismic velocities V_p, V_s, and V_ϕ $[m/s]$ for each entry in the list.

Parameters **moduli** (list of `ElasticProperties`) – input elastic properties.

Returns lists of V_p, V_s, and V_ϕ $[m/s]$

Return type list of float, list of float, list of float

`burnman.main.`**`average_moduli`**(*moduli_list*, *averaging_scheme=<class burnman.averaging_schemes.VoigtReussHill at 0x102808f58>*)

Given an array of of `elastic_properties` of size: n_evaluation_points by n_phases (e.g. that generated by `calculate_moduli()`), calculate the bulk properties, according to some averaging scheme. The averaging scheme defaults to Voigt-Reuss-Hill (see `burnman.averaging_schemes.voigt_reuss_hill`), but the user may specify, Voigt, Reuss, the Hashin-Shtrikman bounds, or any user defined scheme that satisfies the interface `burnman.averaging_schemes.averaging_scheme` (also see *Averaging Schemes*).

Parameters

- **moduli_list** (list of list of `burnman.elastic_properties`) – List of end-member moduli to be averaged.
- **averaging_scheme** (`burnman.averaging_schemes.averaging_scheme`) – Averaging scheme to use.

Returns A list of n_evaluation_points instances of elastic_properties.

Return type list of `burnman.elastic_properties`

`burnman.main.`**`pressures_for_rock`**(*rock*, *depths*, *T0*, *averaging_scheme=<burnman.averaging_schemes.VoigtReussHill instance at 0x102798248>*)

Function computes the self-consistent pressures (to avoid using the PREM depth pressure conversion) [Cam13]. Only simplification is using g from PREM.

Parameters

- **rock** (*burnman.abstract_material*) – this is a rock
- **depths** (*list of float*) – list of depths you want to evaluate the rock at $[m]$.
- **temperatures** (*list of float*) – list of temperatures you want to evaluate the rock at. $[K]$.

- **averaging_scheme** (burnman.averaging_schemes.averaging_scheme) – Averaging scheme to use.

Returns pressures $[Pa]$

Return type list of floats

CHAPTER

SIX

EQUATIONS OF STATE

6.1 Base class

class burnman.equation_of_state.**EquationOfState**

This class defines the interface for an equation of state that a mineral uses to determine its properties at a given P, T. In order define a new equation of state, you should define these functions.

All functions should accept and return values in SI units.

In general these functions are functions of pressure, temperature, and volume, as well as a "params" object, which is a Python dictionary that stores the material parameters of the mineral, such as reference volume, Debye temperature, reference moduli, etc.

The functions for volume and density are just functions of temperature, pressure, and "params"; after all, it does not make sense for them to be functions of volume or density.

adiabatic_bulk_modulus (*pressure*, *temperature*, *volume*, *params*)

Parameters **pressure** : float

Pressure at which to evaluate the equation of state. $[Pa]$

temperature : float

Temperature at which to evaluate the equation of state. $[K]$

volume : float

Molar volume of the mineral. For consistency this should be calculated using `volume()`. $[m^3]$

params : dictionary

Dictionary containing material parameters required by the equation of state.

Returns **K_S** : float

Adiabatic bulk modulus of the mineral. $[Pa]$

density (*pressure*, *temperature*, *params*)

Calculate the density of the mineral $[kg/m^3]$. The params object must include a "molar_mass" field.

Parameters **pressure** : float

Pressure at which to evaluate the equation of state. $[Pa]$

temperature : float

Temperature at which to evaluate the equation of state. $[K]$

params : dictionary

Dictionary containing material parameters required by the equation of state.

Returns density : float

Density of the mineral. $[kg/m^3]$

grueneisen_parameter(*pressure*, *temperature*, *volume*, *params*)

Parameters pressure : float

Pressure at which to evaluate the equation of state. $[Pa]$

temperature : float

Temperature at which to evaluate the equation of state. $[K]$

volume : float

Molar volume of the mineral. For consistency this should be calculated using `volume()`. $[m^3]$

params : dictionary

Dictionary containing material parameters required by the equation of state.

Returns gamma : float

Grueneisen parameter of the mineral. $[unitless]$

heat_capacity_p(*pressure*, *temperature*, *volume*, *params*)

Parameters pressure : float

Pressure at which to evaluate the equation of state. $[Pa]$

temperature : float

Temperature at which to evaluate the equation of state. $[K]$

volume : float

Molar volume of the mineral. For consistency this should be calculated using `volume()`. $[m^3]$

params : dictionary

Dictionary containing material parameters required by the equation of state.

Returns C_P : float

Heat capacity at constant pressure of the mineral. $[J/K/mol]$

heat_capacity_v(*pressure*, *temperature*, *volume*, *params*)

Parameters pressure : float

Pressure at which to evaluate the equation of state. $[Pa]$

temperature : float

Temperature at which to evaluate the equation of state. $[K]$

volume : float

Molar volume of the mineral. For consistency this should be calculated using `volume()`. $[m^3]$

params : dictionary

Dictionary containing material parameters required by the equation of state.

Returns C_V : float

Heat capacity at constant volume of the mineral. $[J/K/mol]$

isothermal_bulk_modulus (*pressure*, *temperature*, *volume*, *params*)

Parameters pressure : float

Pressure at which to evaluate the equation of state. $[Pa]$

temperature : float

Temperature at which to evaluate the equation of state. $[K]$

volume : float

Molar volume of the mineral. For consistency this should be calculated using `volume()`. $[m^3]$

params : dictionary

Dictionary containing material parameters required by the equation of state.

Returns K_T : float

Isothermal bulk modulus of the mineral. $[Pa]$

shear_modulus (*pressure*, *temperature*, *volume*, *params*)

Parameters pressure : float

Pressure at which to evaluate the equation of state. $[Pa]$

temperature : float

Temperature at which to evaluate the equation of state. $[K]$

volume : float

Molar volume of the mineral. For consistency this should be calculated using `volume()`. $[m^3]$

params : dictionary

Dictionary containing material parameters required by the equation of state.

Returns G : float

Shear modulus of the mineral. $[Pa]$

thermal_expansivity (*pressure*, *temperature*, *volume*, *params*)

Parameters **pressure** : float

Pressure at which to evaluate the equation of state. $[Pa]$

temperature : float

Temperature at which to evaluate the equation of state. $[K]$

volume : float

Molar volume of the mineral. For consistency this should be calculated using `volume()`. $[m^3]$

params : dictionary

Dictionary containing material parameters required by the equation of state.

Returns **alpha** : float

Thermal expansivity of the mineral. $[1/K]$

volume (*pressure*, *temperature*, *params*)

Parameters **pressure** : float

Pressure at which to evaluate the equation of state. $[Pa]$

temperature : float

Temperature at which to evaluate the equation of state. $[K]$

params : dictionary

Dictionary containing material parameters required by the equation of state.

Returns **volume** : float

Molar volume of the mineral. $[m^3]$

6.2 Birch-Murnaghan

class burnman.birch_murnaghan.**BirchMurnaghanBase**

Bases: burnman.equation_of_state.EquationOfState

Base class for the isothermal Birch Murnaghan equation of state. This is third order in strain, and has no temperature dependence. However, the shear modulus is sometimes fit to a second order function, so if this is the case, you should use that. For more see burnman.birch_murnaghan.BM2 and burnman.birch_murnaghan.BM3.

adiabatic_bulk_modulus (*pressure*, *temperature*, *volume*, *params*)

Returns adiabatic bulk modulus K_s of the mineral. $[Pa]$.

density (*pressure*, *temperature*, *params*)
Calculate the density of the mineral $[kg/m^3]$. The params object must include a "molar_mass" field.

Parameters **pressure** : float

Pressure at which to evaluate the equation of state. $[Pa]$

temperature : float

Temperature at which to evaluate the equation of state. $[K]$

params : dictionary

Dictionary containing material parameters required by the equation of state.

Returns **density** : float

Density of the mineral. $[kg/m^3]$

grueneisen_parameter (*pressure*, *temperature*, *volume*, *params*)
Since this equation of state does not contain temperature effects, simply return zero. $[unitless]$

heat_capacity_p (*pressure*, *temperature*, *volume*, *params*)
Since this equation of state does not contain temperature effects, simply return a very large number. $[J/K/mol]$

heat_capacity_v (*pressure*, *temperature*, *volume*, *params*)
Since this equation of state does not contain temperature effects, simply return a very large number. $[J/K/mol]$

isothermal_bulk_modulus (*pressure*, *temperature*, *volume*, *params*)
Returns isothermal bulk modulus K_T $[Pa]$ as a function of pressure $[Pa]$, temperature $[K]$ and volume $[m^3]$.

shear_modulus (*pressure*, *temperature*, *volume*, *params*)
Returns shear modulus G of the mineral. $[Pa]$

thermal_expansivity (*pressure*, *temperature*, *volume*, *params*)
Since this equation of state does not contain temperature effects, simply return zero. $[1/K]$

volume (*pressure*, *temperature*, *params*)
Returns volume $[m^3]$ as a function of pressure $[Pa]$.

class burnman.birch_murnaghan.**BM2**
Bases: burnman.birch_murnaghan.BirchMurnaghanBase

Third order Birch Murnaghan isothermal equation of state. This uses the third order expansion for shear modulus.

adiabatic_bulk_modulus (*pressure*, *temperature*, *volume*, *params*)
Returns adiabatic bulk modulus K_s of the mineral. $[Pa]$.

density (*pressure*, *temperature*, *params*)
Calculate the density of the mineral $[kg/m^3]$. The params object must include a "molar_mass" field.

Parameters **pressure** : float

Pressure at which to evaluate the equation of state. $[Pa]$

temperature : float

Temperature at which to evaluate the equation of state. $[K]$

params : dictionary

Dictionary containing material parameters required by the equation of state.

Returns density : float

Density of the mineral. $[kg/m^3]$

gruneneisen_parameter (*pressure*, *temperature*, *volume*, *params*)
Since this equation of state does not contain temperature effects, simply return zero. $[unitless]$

heat_capacity_p (*pressure*, *temperature*, *volume*, *params*)
Since this equation of state does not contain temperature effects, simply return a very large number. $[J/K/mol]$

heat_capacity_v (*pressure*, *temperature*, *volume*, *params*)
Since this equation of state does not contain temperature effects, simply return a very large number. $[J/K/mol]$

isothermal_bulk_modulus (*pressure*, *temperature*, *volume*, *params*)
Returns isothermal bulk modulus K_T $[Pa]$ as a function of pressure $[Pa]$, temperature $[K]$ and volume $[m^3]$.

shear_modulus (*pressure*, *temperature*, *volume*, *params*)
Returns shear modulus G of the mineral. $[Pa]$

thermal_expansivity (*pressure*, *temperature*, *volume*, *params*)
Since this equation of state does not contain temperature effects, simply return zero. $[1/K]$

volume (*pressure*, *temperature*, *params*)
Returns volume $[m^3]$ as a function of pressure $[Pa]$.

class burnman.birch_murnaghan.**BM3**
Bases: burnman.birch_murnaghan.BirchMurnaghanBase

Third order Birch Murnaghan isothermal equation of state. This uses the third order expansion for shear modulus.

adiabatic_bulk_modulus (*pressure*, *temperature*, *volume*, *params*)
Returns adiabatic bulk modulus K_s of the mineral. $[Pa]$.

density (*pressure*, *temperature*, *params*)
Calculate the density of the mineral $[kg/m^3]$. The params object must include a "molar_mass" field.

Parameters pressure : float

Pressure at which to evaluate the equation of state. $[Pa]$

temperature : float

Temperature at which to evaluate the equation of state. $[K]$

params : dictionary

Dictionary containing material parameters required by the equation of state.

Returns density : float

Density of the mineral. $[kg/m^3]$

gruneneisen_parameter (*pressure*, *temperature*, *volume*, *params*)
Since this equation of state does not contain temperature effects, simply return zero. $[unitless]$

heat_capacity_p (*pressure*, *temperature*, *volume*, *params*)
Since this equation of state does not contain temperature effects, simply return a very large number. $[J/K/mol]$

heat_capacity_v (*pressure*, *temperature*, *volume*, *params*)
Since this equation of state does not contain temperature effects, simply return a very large number. $[J/K/mol]$

isothermal_bulk_modulus (*pressure*, *temperature*, *volume*, *params*)
Returns isothermal bulk modulus K_T $[Pa]$ as a function of pressure $[Pa]$, temperature $[K]$ and volume $[m^3]$.

shear_modulus (*pressure*, *temperature*, *volume*, *params*)
Returns shear modulus G of the mineral. $[Pa]$

thermal_expansivity (*pressure*, *temperature*, *volume*, *params*)
Since this equation of state does not contain temperature effects, simply return zero. $[1/K]$

volume (*pressure*, *temperature*, *params*)
Returns volume $[m^3]$ as a function of pressure $[Pa]$.

6.3 Stixrude and Lithgow-Bertelloni Formulation

class burnman.slb.**SLBBase**
Bases: burnman.equation_of_state.EquationOfState

Base class for the finite strain-Mie-Grueneiesen-Debye equation of state detailed in [SLB05]. For the most part the equations are all third order in strain, but see further the burnman.slb.SLB2 and burnman.slb.SLB3 classes.

adiabatic_bulk_modulus (*pressure*, *temperature*, *volume*, *params*)
Returns adiabatic bulk modulus. $[Pa]$

density (*pressure*, *temperature*, *params*)
Calculate the density of the mineral $[kg/m^3]$. The params object must include a "molar_mass" field.

Parameters pressure : float

Pressure at which to evaluate the equation of state. $[Pa]$

temperature : float

Temperature at which to evaluate the equation of state. $[K]$

params : dictionary

Dictionary containing material parameters required by the equation of state.

Returns density : float

Density of the mineral. $[kg/m^3]$

grueneisen_parameter (*pressure*, *temperature*, *volume*, *params*)
Returns grueneisen parameter $[unitless]$

heat_capacity_p (*pressure*, *temperature*, *volume*, *params*)
Returns heat capacity at constant pressure. $[J/K/mol]$

heat_capacity_v (*pressure*, *temperature*, *volume*, *params*)
Returns heat capacity at constant volume. $[J/K/mol]$

isothermal_bulk_modulus (*pressure*, *temperature*, *volume*, *params*)
Returns isothermal bulk modulus $[Pa]$

shear_modulus (*pressure*, *temperature*, *volume*, *params*)
Returns shear modulus. $[Pa]$

thermal_expansivity (*pressure*, *temperature*, *volume*, *params*)
Returns thermal expansivity. $[1/K]$

volume (*pressure*, *temperature*, *params*)
Returns molar volume. $[m^3]$

volume_dependent_q (*x*, *params*)
Finite strain approximation for q, the isotropic volume strain derivative of the grueneisen parameter.

class burnman.slb.**SLB2**
Bases: burnman.slb.SLBBase

SLB equation of state with second order finite strain expansion for the shear modulus. In general, this should not be used, but sometimes shear modulus data is fit to a second order equation of state. In that case, you should use this. The moral is, be careful!

adiabatic_bulk_modulus (*pressure*, *temperature*, *volume*, *params*)
Returns adiabatic bulk modulus. $[Pa]$

density (*pressure*, *temperature*, *params*)
Calculate the density of the mineral $[kg/m^3]$. The params object must include a "molar_mass" field.

Parameters pressure : float

Pressure at which to evaluate the equation of state. $[Pa]$

temperature : float

Temperature at which to evaluate the equation of state. $[K]$

params : dictionary

Dictionary containing material parameters required by the equation of state.

Returns **density** : float

Density of the mineral. $[kg/m^3]$

grueneisen_parameter (*pressure*, *temperature*, *volume*, *params*)
Returns grueneisen parameter $[unitless]$

heat_capacity_p (*pressure*, *temperature*, *volume*, *params*)
Returns heat capacity at constant pressure. $[J/K/mol]$

heat_capacity_v (*pressure*, *temperature*, *volume*, *params*)
Returns heat capacity at constant volume. $[J/K/mol]$

isothermal_bulk_modulus (*pressure*, *temperature*, *volume*, *params*)
Returns isothermal bulk modulus $[Pa]$

shear_modulus (*pressure*, *temperature*, *volume*, *params*)
Returns shear modulus. $[Pa]$

thermal_expansivity (*pressure*, *temperature*, *volume*, *params*)
Returns thermal expansivity. $[1/K]$

volume (*pressure*, *temperature*, *params*)
Returns molar volume. $[m^3]$

volume_dependent_q (*x*, *params*)
Finite strain approximation for q, the isotropic volume strain derivative of the grueneisen parameter.

class burnman.slb.**SLB3**
Bases: burnman.slb.SLBBase

SLB equation of state with third order finite strain expansion for the shear modulus (this should be preferred, as it is more thermodynamically consistent.

adiabatic_bulk_modulus (*pressure*, *temperature*, *volume*, *params*)
Returns adiabatic bulk modulus. $[Pa]$

density (*pressure*, *temperature*, *params*)
Calculate the density of the mineral $[kg/m^3]$. The params object must include a "molar_mass" field.

Parameters **pressure** : float

Pressure at which to evaluate the equation of state. $[Pa]$

temperature : float

Temperature at which to evaluate the equation of state. $[K]$

params : dictionary

Dictionary containing material parameters required by the equation of state.

Returns **density** : float

Density of the mineral. $[kg/m^3]$

grueneisen_parameter (*pressure*, *temperature*, *volume*, *params*)
Returns grueneisen parameter $[unitless]$

heat_capacity_p (*pressure*, *temperature*, *volume*, *params*)
Returns heat capacity at constant pressure. $[J/K/mol]$

heat_capacity_v (*pressure*, *temperature*, *volume*, *params*)
Returns heat capacity at constant volume. $[J/K/mol]$

isothermal_bulk_modulus (*pressure*, *temperature*, *volume*, *params*)
Returns isothermal bulk modulus $[Pa]$

shear_modulus (*pressure*, *temperature*, *volume*, *params*)
Returns shear modulus. $[Pa]$

thermal_expansivity (*pressure*, *temperature*, *volume*, *params*)
Returns thermal expansivity. $[1/K]$

volume (*pressure*, *temperature*, *params*)
Returns molar volume. $[m^3]$

volume_dependent_q (*x*, *params*)
Finite strain approximation for q, the isotropic volume strain derivative of the grueneisen parameter.

CHAPTER

SEVEN

AVERAGING SCHEMES

Given a set of mineral physics parameters and an equation of state we can calculate the density, bulk, and shear modulus for a given phase. However, as soon as we have a composite material (e.g., a rock), the determination of elastic properties become more complicated. The bulk and shear modulus of a rock are dependent on the specific geometry of the grains in the rock, so there is no general formula for its averaged elastic properties. Instead, we must choose from a number of averaging schemes if we want a single value, or use bounding methods to get a range of possible values. The module `burnman.averaging_schemes` provides a number of different average and bounding schemes for determining a composite rock's physical parameters.

7.1 Base class

class `burnman.averaging_schemes.`**`AveragingScheme`**

Base class defining an interface for determining average elastic properties of a rock. Given a list of volume fractions for the different mineral phases in a rock, as well as their bulk and shear moduli, an averaging will give back a single scalar values for the averages. New averaging schemes should define the functions average_bulk_moduli and average_shear_moduli, as specified here.

`average_bulk_moduli` (*volumes*, *bulk_moduli*, *shear_moduli*)

Average the bulk moduli K for a composite. This defines the interface for this method, and is not implemented in the base class.

Parameters **volumes** : list of floats

List of the volume of each phase in the composite. $[m^3]$

bulk_moduli : list of floats

List of bulk moduli of each phase in the composite. $[Pa]$

shear_moduli : list of floats

List of shear moduli of each phase in the composite. $[Pa]$

Returns **K** : float

The average bulk modulus K. $[Pa]$

`average_density` (*volumes*, *densities*)

Average the densities of a composite, given a list of volume fractions and densitites. This is

the only method of burnman.averaging_schemes.AveragingScheme which is implemented in the base class, as how to calculate it is not dependent on the geometry of the rock. The formula for density is given by

$$\rho = \frac{\Sigma_i \rho_i V_i}{\Sigma_i V_i}$$

Parameters **volumes** : list of floats

List of the volume of each phase in the composite. $[m^3]$

densities : list of floats

List of densities of each phase in the composite. $[kg/m^3]$

Returns **rho** : float

Density ρ. $[kg/m^3]$

average_heat_capacity_p (*fractions*, *c_p*)

Averages the heat capacities at constant pressure C_P by molar fractions.

Parameters **fractions** : list of floats

List of molar fractions of each phase in the composite (should sum to 1.0).

c_p : list of floats

List of heat capacities at constant pressure C_P of each phase in the composite. $[J/K/mol]$

Returns **c_p** : float

heat capacity at constant pressure C_P of the composite. $[J/K/mol]$

average_heat_capacity_v (*fractions*, *c_v*)

Averages the heat capacities at constant volume C_V by molar fractions as in eqn. (16) in [IS92].

Parameters **fractions** : list of floats

List of molar fractions of each phase in the composite (should sum to 1.0).

c_v : list of floats

List of heat capacities at constant volume C_V of each phase in the composite. $[J/K/mol]$

Returns **c_v** : float

heat capacity at constant volume of the composite C_V. $[J/K/mol]$

average_shear_moduli (*volumes*, *bulk_moduli*, *shear_moduli*)

Average the shear moduli G for a composite. This defines the interface for this method, and is not implemented in the base class.

Parameters **volumes** : list of floats

List of the volume of each phase in the composite. $[m^3]$

bulk_moduli : list of floats

List of bulk moduli of each phase in the composite. $[Pa]$

shear_moduli : list of floats

List of shear moduli of each phase in the composite. $[Pa]$

Returns **G** : float

The average shear modulus G. $[Pa]$

average_thermal_expansivity(*volumes*, *alphas*)
thermal expansion coefficient of the mineral α. $[1/K]$

7.2 Voigt bound

class burnman.averaging_schemes.**Voigt**
Bases: burnman.averaging_schemes.AveragingScheme

Class for computing the Voigt (iso-strain) bound for elastic properties. This derives from burnman.averaging_schemes.averaging_scheme, and implements the burnman.averaging_schemes.averaging_scheme.average_bulk_moduli() and burnman.averaging_schemes.averaging_scheme.average_shear_moduli() functions.

average_bulk_moduli(*volumes*, *bulk_moduli*, *shear_moduli*)
Average the bulk moduli of a composite K with the Voigt (iso-strain) bound, given by:

$$K_V = \Sigma_i V_i K_i$$

Parameters **volumes** : list of floats

List of the volume of each phase in the composite. $[m^3]$

bulk_moduli : list of floats

List of bulk moduli K of each phase in the composite. $[Pa]$

shear_moduli : list of floats

List of shear moduli G of each phase in the composite. Not used in this average. $[Pa]$

Returns **K** : float

The Voigt average bulk modulus K_V. $[Pa]$

average_density(*volumes*, *densities*)
Average the densities of a composite, given a list of volume fractions and densitites. This is the only method of burnman.averaging_schemes.AveragingScheme which is implemented in the base class, as how to calculate it is not dependent on the geometry of the rock. The formula for density is given by

$$\rho = \frac{\Sigma_i \rho_i V_i}{\Sigma_i V_i}$$

Parameters **volumes** : list of floats

List of the volume of each phase in the composite. $[m^3]$

densities : list of floats

List of densities of each phase in the composite. $[kg/m^3]$

Returns **rho** : float

Density ρ. $[kg/m^3]$

average_heat_capacity_p (*fractions*, *c_p*)

Averages the heat capacities at constant pressure C_P by molar fractions.

Parameters **fractions** : list of floats

List of molar fractions of each phase in the composite (should sum to 1.0).

c_p : list of floats

List of heat capacities at constant pressure C_P of each phase in the composite. $[J/K/mol]$

Returns **c_p** : float

heat capacity at constant pressure C_P of the composite. $[J/K/mol]$

average_heat_capacity_v (*fractions*, *c_v*)

Averages the heat capacities at constant volume C_V by molar fractions as in eqn. (16) in [IS92].

Parameters **fractions** : list of floats

List of molar fractions of each phase in the composite (should sum to 1.0).

c_v : list of floats

List of heat capacities at constant volume C_V of each phase in the composite. $[J/K/mol]$

Returns **c_v** : float

heat capacity at constant volume of the composite C_V. $[J/K/mol]$

average_shear_moduli (*volumes*, *bulk_moduli*, *shear_moduli*)

Average the shear moduli of a composite with the Voigt (iso-strain) bound, given by:

$$G_V = \Sigma_i V_i G_i$$

Parameters **volumes** : list of floats

List of the volume of each phase in the composite. $[m^3]$

bulk_moduli : list of floats

List of bulk moduli K of each phase in the composite. Not used in this average. $[Pa]$

shear_moduli : list of floats

List of shear moduli G of each phase in the composite. $[Pa]$

Returns **G** : float

The Voigt average shear modulus G_V. $[Pa]$

average_thermal_expansivity (*volumes*, *alphas*)
thermal expansion coefficient of the mineral α. $[1/K]$

7.3 Reuss bound

class burnman.averaging_schemes.**Reuss**
Bases: burnman.averaging_schemes.AveragingScheme

Class for computing the Reuss (iso-stress) bound for elastic properties. This derives from burnman.averaging_schemes.averaging_scheme, and implements the burnman.averaging_schemes.averaging_scheme.average_bulk_moduli() and

burnman.averaging_schemes.averaging_scheme.average_shear_moduli()
functions.

average_bulk_moduli (*volumes*, *bulk_moduli*, *shear_moduli*)
Average the bulk moduli of a composite with the Reuss (iso-stress) bound, given by:

$$K_R = \left(\Sigma_i \frac{V_i}{K_i} \right)^{-1}$$

Parameters **volumes** : list of floats

List of the volume of each phase in the composite. $[m^3]$

bulk_moduli : list of floats

List of bulk moduli K of each phase in the composite. $[Pa]$

shear_moduli : list of floats

List of shear moduli G of each phase in the composite. Not used in this average. $[Pa]$

Returns **K** : float

The Reuss average bulk modulus K_R. $[Pa]$

average_density (*volumes*, *densities*)
Average the densities of a composite, given a list of volume fractions and densitites. This is the only method of burnman.averaging_schemes.AveragingScheme which is implemented in the base class, as how to calculate it is not dependent on the geometry of the rock. The formula for density is given by

$$\rho = \frac{\Sigma_i \rho_i V_i}{\Sigma_i V_i}$$

Parameters **volumes** : list of floats

List of the volume of each phase in the composite. $[m^3]$

densities : list of floats

List of densities of each phase in the composite. $[kg/m^3]$

Returns rho : float

Density ρ. $[kg/m^3]$

average_heat_capacity_p (*fractions*, *c_p*)
Averages the heat capacities at constant pressure C_P by molar fractions.

Parameters fractions : list of floats

List of molar fractions of each phase in the composite (should sum to 1.0).

c_p : list of floats

List of heat capacities at constant pressure C_P of each phase in the composite. $[J/K/mol]$

Returns c_p : float

heat capacity at constant pressure C_P of the composite. $[J/K/mol]$

average_heat_capacity_v (*fractions*, *c_v*)
Averages the heat capacities at constant volume C_V by molar fractions as in eqn. (16) in [IS92].

Parameters fractions : list of floats

List of molar fractions of each phase in the composite (should sum to 1.0).

c_v : list of floats

List of heat capacities at constant volume C_V of each phase in the composite. $[J/K/mol]$

Returns c_v : float

heat capacity at constant volume of the composite C_V. $[J/K/mol]$

average_shear_moduli (*volumes*, *bulk_moduli*, *shear_moduli*)
Average the shear moduli of a composite with the Reuss (iso-stress) bound, given by:

$$G_R = \left(\Sigma_i \frac{V_i}{G_i} \right)^{-1}$$

Parameters volumes : list of floats

List of the volume of each phase in the composite. $[m^3]$

bulk_moduli : list of floats

List of bulk moduli K of each phase in the composite. Not used in this average. $[Pa]$

shear_moduli : list of floats

List of shear moduli G of each phase in the composite. $[Pa]$

Returns G : float

The Reuss average shear modulus G_R. $[Pa]$

average_thermal_expansivity (*volumes*, *alphas*)
thermal expansion coefficient of the mineral α. $[1/K]$

7.4 Voigt-Reuss-Hill average

class burnman.averaging_schemes.**VoigtReussHill**
Bases: burnman.averaging_schemes.AveragingScheme

Class for computing the Voigt-Reuss-Hill average for elastic properties. This derives from burnman.averaging_schemes.averaging_scheme, and implements the burnman.averaging_schemes.averaging_scheme.average_bulk_moduli() and burnman.averaging_schemes.averaging_scheme.average_shear_moduli() functions.

average_bulk_moduli (*volumes*, *bulk_moduli*, *shear_moduli*)
Average the bulk moduli of a composite with the Voigt-Reuss-Hill average, given by:

$$K_{VRH} = \frac{K_V + K_R}{2}$$

This is simply a shorthand for an arithmetic average of the bounds given by burnman.averaging_schemes.voigt and burnman.averaging_schemes.reuss.

Parameters **volumes** : list of floats

List of the volume of each phase in the composite. $[m^3]$

bulk_moduli : list of floats

List of bulk moduli K of each phase in the composite. $[Pa]$

shear_moduli : list of floats

List of shear moduli G of each phase in the composite. Not used in this average. $[Pa]$

Returns **K** : float

The Voigt-Reuss-Hill average bulk modulus K_{VRH}. $[Pa]$

average_density (*volumes*, *densities*)
Average the densities of a composite, given a list of volume fractions and densitites. This is the only method of burnman.averaging_schemes.AveragingScheme which is implemented in the base class, as how to calculate it is not dependent on the geometry of the rock. The formula for density is given by

$$\rho = \frac{\Sigma_i \rho_i V_i}{\Sigma_i V_i}$$

Parameters **volumes** : list of floats

List of the volume of each phase in the composite. $[m^3]$

densities : list of floats

List of densities of each phase in the composite. $[kg/m^3]$

Returns rho : float

Density ρ. $[kg/m^3]$

average_heat_capacity_p (*fractions*, *c_p*)
Averages the heat capacities at constant pressure C_P by molar fractions.

Parameters fractions : list of floats

List of molar fractions of each phase in the composite (should sum to 1.0).

c_p : list of floats

List of heat capacities at constant pressure C_P of each phase in the composite. $[J/K/mol]$

Returns c_p : float

heat capacity at constant pressure C_P of the composite. $[J/K/mol]$

average_heat_capacity_v (*fractions*, *c_v*)
Averages the heat capacities at constant volume C_V by molar fractions as in eqn. (16) in [IS92].

Parameters fractions : list of floats

List of molar fractions of each phase in the composite (should sum to 1.0).

c_v : list of floats

List of heat capacities at constant volume C_V of each phase in the composite. $[J/K/mol]$

Returns c_v : float

heat capacity at constant volume of the composite C_V. $[J/K/mol]$

average_shear_moduli (*volumes*, *bulk_moduli*, *shear_moduli*)
Average the shear moduli G of a composite with the Voigt-Reuss-Hill average, given by:

$$G_{VRH} = \frac{G_V + G_R}{2}$$

This is simply a shorthand for an arithmetic average of the bounds given by `burnman.averaging_schemes.voigt` and `burnman.averaging_schemes.reuss`.

Parameters volumes : list of floats

List of the volume of each phase in the composite $[m^3]$

bulk_moduli : list of floats

List of bulk moduli K of each phase in the composite Not used in this average. $[Pa]$

shear_moduli : list of floats

List of shear moduli G of each phase in the composite $[Pa]$

Returns **G** : float

The Voigt-Reuss-Hill average shear modulus G_{VRH}. $[Pa]$

average_thermal_expansivity (*volumes*, *alphas*)
thermal expansion coefficient of the mineral α. $[1/K]$

7.5 Hashin-Shtrikman upper bound

class burnman.averaging_schemes.**HashinShtrikmanUpper**
Bases: burnman.averaging_schemes.AveragingScheme

Class for computing the upper Hashin-Shtrikman bound for elastic properties. This derives from burnman.averaging_schemes.averaging_scheme, and implements the burnman.averaging_schemes.averaging_scheme.average_bulk_moduli() and burnman.averaging_schemes.averaging_scheme.average_shear_moduli() functions. Implements formulas from [WDOConnell76]. The Hashin-Shtrikman bounds are tighter than the Voigt and Reuss bounds because they make the additional assumption that the orientation of the phases are statistically isotropic. In some cases this may be a good assumption, and in others it may not be.

average_bulk_moduli (*volumes*, *bulk_moduli*, *shear_moduli*)
Average the bulk moduli of a composite with the upper Hashin-Shtrikman bound. Implements Formulas from [WDOConnell76], which are too lengthy to reproduce here.

Parameters **volumes** : list of floats

List of the volume of each phase in the composite. $[m^3]$

bulk_moduli : list of floats

List of bulk moduli K of each phase in the composite. $[Pa]$

shear_moduli : list of floats

List of shear moduli G of each phase in the composite. $[Pa]$

Returns **K** : float

The upper Hashin-Shtrikman average bulk modulus K. $[Pa]$

average_density (*volumes*, *densities*)
Average the densities of a composite, given a list of volume fractions and densitites. This is the only method of burnman.averaging_schemes.AveragingScheme which is implemented in the base class, as how to calculate it is not dependent on the geometry of the rock. The formula for density is given by

$$\rho = \frac{\Sigma_i \rho_i V_i}{\Sigma_i V_i}$$

Parameters **volumes** : list of floats

List of the volume of each phase in the composite. $[m^3]$

densities : list of floats

List of densities of each phase in the composite. $[kg/m^3]$

Returns rho : float

Density ρ. $[kg/m^3]$

average_heat_capacity_p (*fractions*, *c_p*)

Averages the heat capacities at constant pressure C_P by molar fractions.

Parameters fractions : list of floats

List of molar fractions of each phase in the composite (should sum to 1.0).

c_p : list of floats

List of heat capacities at constant pressure C_P of each phase in the composite. $[J/K/mol]$

Returns c_p : float

heat capacity at constant pressure C_P of the composite. $[J/K/mol]$

average_heat_capacity_v (*fractions*, *c_v*)

Averages the heat capacities at constant volume C_V by molar fractions as in eqn. (16) in [IS92].

Parameters fractions : list of floats

List of molar fractions of each phase in the composite (should sum to 1.0).

c_v : list of floats

List of heat capacities at constant volume C_V of each phase in the composite. $[J/K/mol]$

Returns c_v : float

heat capacity at constant volume of the composite C_V. $[J/K/mol]$

average_shear_moduli (*volumes*, *bulk_moduli*, *shear_moduli*)

Average the shear moduli of a composite with the upper Hashin-Shtrikman bound. Implements Formulas from [WDOConnell76], which are too lengthy to reproduce here.

Parameters volumes : list of floats

List of the volume of each phase in the composite. $[m^3]$

bulk_moduli : list of floats

List of bulk moduli K of each phase in the composite. $[Pa]$

shear_moduli : list of floats

List of shear moduli G of each phase in the composite. $[Pa]$

Returns G : float

The upper Hashin-Shtrikman average shear modulus G. $[Pa]$

average_thermal_expansivity (*volumes*, *alphas*)
thermal expansion coefficient of the mineral α. $[1/K]$

7.6 Hashin-Shtrikman lower bound

class burnman.averaging_schemes.**HashinShtrikmanLower**
Bases: burnman.averaging_schemes.AveragingScheme

Class for computing the lower Hashin-Shtrikman bound for elastic properties. This derives from burnman.averaging_schemes.averaging_scheme, and implements the burnman.averaging_schemes.averaging_scheme.average_bulk_moduli() and burnman.averaging_schemes.averaging_scheme.average_shear_moduli() functions. Implements Formulas from [WDOConnell76]. The Hashin-Shtrikman bounds are tighter than the Voigt and Reuss bounds because they make the additional assumption that the orientation of the phases are statistically isotropic. In some cases this may be a good assumption, and in others it may not be.

average_bulk_moduli (*volumes*, *bulk_moduli*, *shear_moduli*)
Average the bulk moduli of a composite with the lower Hashin-Shtrikman bound. Implements Formulas from [WDOConnell76], which are too lengthy to reproduce here.

Parameters **volumes** : list of floats

List of the volume of each phase in the composite. $[m^3]$

bulk_moduli : list of floats

List of bulk moduli K of each phase in the composite. $[Pa]$

shear_moduli : list of floats

List of shear moduli G of each phase in the composite. $[Pa]$

Returns **K** : float

The lower Hashin-Shtrikman average bulk modulus K. $[Pa]$

average_density (*volumes*, *densities*)
Average the densities of a composite, given a list of volume fractions and densitites. This is the only method of burnman.averaging_schemes.AveragingScheme which is implemented in the base class, as how to calculate it is not dependent on the geometry of the rock. The formula for density is given by

$$\rho = \frac{\Sigma_i \rho_i V_i}{\Sigma_i V_i}$$

Parameters **volumes** : list of floats

List of the volume of each phase in the composite. $[m^3]$

densities : list of floats

List of densities of each phase in the composite. $[kg/m^3]$

Returns **rho** : float

Density ρ. $[kg/m^3]$

average_heat_capacity_p (*fractions*, *c_p*)
Averages the heat capacities at constant pressure C_P by molar fractions.

Parameters **fractions** : list of floats

List of molar fractions of each phase in the composite (should sum to 1.0).

c_p : list of floats

List of heat capacities at constant pressure C_P of each phase in the composite. $[J/K/mol]$

Returns **c_p** : float

heat capacity at constant pressure C_P of the composite. $[J/K/mol]$

average_heat_capacity_v (*fractions*, *c_v*)
Averages the heat capacities at constant volume C_V by molar fractions as in eqn. (16) in [IS92].

Parameters **fractions** : list of floats

List of molar fractions of each phase in the composite (should sum to 1.0).

c_v : list of floats

List of heat capacities at constant volume C_V of each phase in the composite. $[J/K/mol]$

Returns **c_v** : float

heat capacity at constant volume of the composite C_V. $[J/K/mol]$

average_shear_moduli (*volumes*, *bulk_moduli*, *shear_moduli*)
Average the shear moduli of a composite with the lower Hashin-Shtrikman bound. Implements Formulas from [WDOConnell76], which are too lengthy to reproduce here.

Parameters **volumes** : list of floats

List of volumes of each phase in the composite. $[m^3]$.

bulk_moduli : list of floats

List of bulk moduli K of each phase in the composite. $[Pa]$.

shear_moduli : list of floats

List of shear moduli G of each phase in the composite. $[Pa]$

Returns **G** : float

The lower Hashin-Shtrikman average shear modulus G. $[Pa]$

average_thermal_expansivity (*volumes*, *alphas*)
thermal expansion coefficient of the mineral α. $[1/K]$

7.7 Hashin-Shtrikman arithmetic average

class burnman.averaging_schemes.**HashinShtrikmanAverage**

Bases: burnman.averaging_schemes.AveragingScheme

Class for computing arithmetic mean of the Hashin-Shtrikman bounds on elastic properties. This derives from burnman.averaging_schemes.averaging_scheme, and implements the burnman.averaging_schemes.averaging_scheme.average_bulk_moduli() and burnman.averaging_schemes.averaging_scheme.average_shear_moduli() functions.

average_bulk_moduli (*volumes*, *bulk_moduli*, *shear_moduli*)

Average the bulk moduli of a composite with the arithmetic mean of the upper and lower Hashin-Shtrikman bounds.

Parameters **volumes** : list of floats

List of the volumes of each phase in the composite. $[m^3]$

bulk_moduli : list of floats

List of bulk moduli K of each phase in the composite. $[Pa]$

shear_moduli : list of floats

List of shear moduli G of each phase in the composite. Not used in this average. $[Pa]$

Returns **K** : float

The arithmetic mean of the Hashin-Shtrikman bounds on bulk modulus K. $[Pa]$

average_density (*volumes*, *densities*)

Average the densities of a composite, given a list of volume fractions and densitites. This is the only method of burnman.averaging_schemes.AveragingScheme which is implemented in the base class, as how to calculate it is not dependent on the geometry of the rock. The formula for density is given by

$$\rho = \frac{\Sigma_i \rho_i V_i}{\Sigma_i V_i}$$

Parameters **volumes** : list of floats

List of the volume of each phase in the composite. $[m^3]$

densities : list of floats

List of densities of each phase in the composite. $[kg/m^3]$

Returns **rho** : float

Density ρ. $[kg/m^3]$

average_heat_capacity_p (*fractions*, *c_p*)

Averages the heat capacities at constant pressure C_P by molar fractions.

Parameters **fractions** : list of floats

List of molar fractions of each phase in the composite (should sum to 1.0).

c_p : list of floats

List of heat capacities at constant pressure C_P of each phase in the composite. $[J/K/mol]$

Returns **c_p** : float

heat capacity at constant pressure C_P of the composite. $[J/K/mol]$

average_heat_capacity_v (*fractions*, *c_v*)

Averages the heat capacities at constant volume C_V by molar fractions as in eqn. (16) in [IS92].

Parameters **fractions** : list of floats

List of molar fractions of each phase in the composite (should sum to 1.0).

c_v : list of floats

List of heat capacities at constant volume C_V of each phase in the composite. $[J/K/mol]$

Returns **c_v** : float

heat capacity at constant volume of the composite C_V. $[J/K/mol]$

average_shear_moduli (*volumes*, *bulk_moduli*, *shear_moduli*)

Average the bulk moduli of a composite with the arithmetic mean of the upper and lower Hashin-Shtrikman bounds.

Parameters **volumes** : list of floats

List of the volumes of each phase in the composite. [m^3].

bulk_moduli : list of floats

List of bulk moduli K of each phase in the composite. Not used in this average. $[Pa]$

shear_moduli : list of floats

List of shear moduli G of each phase in the composite. $[Pa]$

Returns **G** : float

The arithmetic mean of the Hashin-Shtrikman bounds on shear modulus G. $[Pa]$

average_thermal_expansivity (*volumes*, *alphas*)

thermal expansion coefficient of the mineral α. $[1/K]$

CHAPTER

EIGHT

GEOTHERMS

burnman.geotherm.**adiabatic**(*pressures*, *T0*, *rock*)

This calculates a geotherm based on an anchor temperature and a rock, assuming that the rock's temperature follows an adiabatic gradient with pressure. This amounts to integrating:

$$\frac{\partial T}{\partial P} = \frac{\gamma T}{K_s}$$

where γ is the Grueneisen parameter and K_s is the adiabatic bulk modulus.

Parameters **pressures** : list of floats

The list of pressures in $[Pa]$ at which to evaluate the geotherm.

T0 : float

An anchor temperature, corresponding to the temperature of the first pressure in the list. $[K]$

rock : burnman.composite

Material for which we compute the adiabat. From this material we must compute average Grueneisen parameters and adiabatic bulk moduli for each pressure/temperature.

Returns temperature: list of floats

The list of temperatures for each pressure. $[K]$

burnman.geotherm.**anderson**(*pressure*)

Geotherm from [And82a].

Parameters **pressure** : list of floats

The list of pressures at which to evaluate the geotherm. $[Pa]$

Returns **temperature** : list of floats

The list of temperatures for each of the pressures. $[K]$

burnman.geotherm.**brown_shankland**(*pressure*)

Geotherm from [BS81].

Parameters **pressure** : list of floats

The list of pressures at which to evaluate the geotherm. $[Pa]$

Returns **temperature** : list of floats

The list of temperatures for each of the pressures. $[K]$

burnman.geotherm.**dTdP**(*temperature*, *pressure*, *rock*)

ODE to integrate temperature with depth for a composite material Assumes that the minerals exist at a common pressure (Reuss bound, should be good for slow deformations at high temperature), as well as an adiabatic process. This corresponds to conservation of enthalpy. First consider compression of the composite to a new pressure P+dP. They all heat up different amounts dT[i], according to their thermoelastic parameters. Then allow them to equilibrate to a constant temperature dT, conserving heat within the composite. This works out to the formula:

$$dT/dP = T * \frac{\Sigma_i(X[i] * C_p[i] * \gamma[i]/K[i])}{\Sigma(X[i] * Cp[i])}$$

Where $X[i]$ is the molar fraction of phase i, C_p is the specific heat at constant pressure, γ is the Gruneisen parameter and K is the bulk modulus. This function is called by burnman.geotherm.adiabatic(), and in general it will not be too useful in other contexts.

Parameters **pressure** : float

The pressure at which to evaluate dT/dP. $[Pa]$

temperature : float

The temperature at which to evaluate dT/dP. $[K]$

rock : burnman.composite

Material for which we compute dT/dP.

Returns **dT/dP** : float

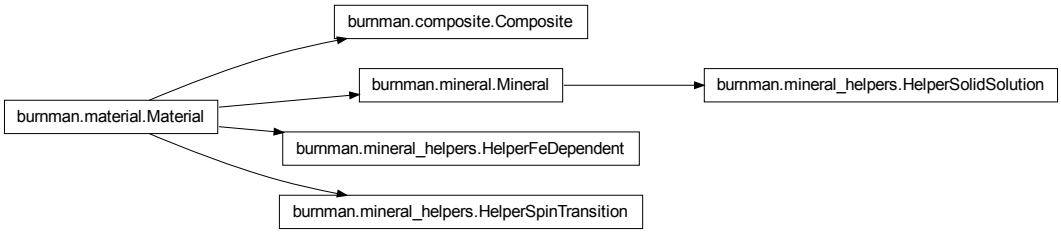
Adiabatic temperature gradient for the composite at a given temperature abd pressure. $[K/Pa]$

CHAPTER

NINE

MINERALS

Burnman operates on rocks (type `Material`) most prominently in form of minerals (`Mineral`) and composites (`Composite`).



9.1 Base Class

class `burnman.material.`**Material**

Base class for all materials. The main functionality is unroll() which returns a list of objects of type `Mineral` and their molar fractions. This class is available as `burnman.Material`.

The user needs to call set_method() (once in the beginning) and set_state() before querying the material with unroll() or density().

Attributes

pressure	(float) The current pressure as set by `set_state()`. [Pa]
temperature	(float) The current temperature as set by `set_state()`. [K]

debug_print(*indent=''*)

Print a human-readable representation of this Material.

density()

Returns the density of this material. Note that the return value of this function may depend on the current state (temperature, pressure). [kg/m^3]

Returns **density** : float

The density of this material in [kg/m^3]

Notes

Needs to be implemented in derived classes.

print_minerals_of_current_state()
Print a human-readable representation of this Material at the current P, T as a list of minerals. This requires set_state() has been called before.

set_method(*method*)
Set the averaging method. See *Averaging Schemes* for details.

Notes

Needs to be implemented in derived classes.

set_state(*pressure*, *temperature*)
Set the material to the given pressure and temperature.

Parameters **pressure** : float

The desired pressure in [Pa].

temperature : float

The desired temperature in [K].

to_string()
Returns a human-readable name of this material. The default implementation will return the name of the class, which is a reasonable default.

Returns **name** : string

Name of this material.

unroll()
Unroll this material into a list of `burnman.Mineral` and their molar fractions. All averaging schemes then operate on this list of minerals. Note that the return value of this function may depend on the current state (temperature, pressure).

Returns **fractions** : list of float

List of molar fractions, should sum to 1.0.

minerals : list of `burnman.Mineral`

List of minerals.

Notes

Needs to be implemented in derived classes.

9.2 Base for individual Minerals

class burnman.mineral.**Mineral**

Bases: burnman.material.Material

This is the base class for all minerals. States of the mineral can only be queried after setting the pressure and temperature using set_state(). The method for computing properties of the material is set using set_method(), which should be done once after creating the material.

This class is available as burnman.Mineral.

If deriving from this class, set the properties in self.params to the desired values. For more complicated materials you can overwrite set_state(), change the params and then call set_state() from this class.

All the material parameters are expected to be in plain SI units. This means that the elastic moduli should be in Pascals and NOT Gigapascals, and the Debye temperature should be in K not C. Additionally, the reference volume should be in m^3/(mol molecule) and not in unit cell volume and 'n' should be the number of atoms per molecule. Frequently in the literature the reference volume is given in Angstrom^3 per unit cell. To convert this to m^3/(mol of molecule) you should multiply by 10^(-30) * N_a / Z, where N_a is Avogadro's number and Z is the number of formula units per unit cell. You can look up Z in many places, including www.mindat.org

adiabatic_bulk_modulus()
Returns adiabatic bulk modulus of the mineral [Pa]

debug_print(*indent=''*)

density()
Returns density of the mineral [kg/m^3]

grueneisen_parameter()
Returns grueneisen parameter of the mineral [unitless]

heat_capacity_p()
Returns heat capacity at constant pressure of the mineral [J/K/mol]

heat_capacity_v()
Returns heat capacity at constant volume of the mineral [J/K/mol]

isothermal_bulk_modulus()
Returns isothermal bulk modulus of the mineral [Pa]

molar_mass()
Returns molar mass of the mineral [kg/mol]

molar_volume()
Returns molar volume of the mineral [m^3/mol]

print_minerals_of_current_state()
Print a human-readable representation of this Material at the current P, T as a list of minerals. This requires set_state() has been called before.

set_method(*method*)
Set the equation of state to be used for this mineral. Takes a string corresponding to any of the predefined equations of state: 'bm2', 'bm3', 'mgd2', 'mgd3', 'slb2', or 'slb3'. Alternatively, you can pass a user defined class which derives from the equation_of_state base class.

set_state(*pressure*, *temperature*)
Update the material to the given pressure [Pa] and temperature [K].

This updates the other properties of this class (v_s, v_p, ...).

shear_modulus()
Returns shear modulus of the mineral [Pa]

thermal_expansivity()
Returns thermal expansion coefficient of the mineral [1/K]

to_string()
Returns the name of the mineral class

unroll()

v_p()
Returns P wave speed of the mineral [m/s]

v_phi()
Returns bulk sound speed of the mineral [m/s]

v_s()
Returns shear wave speed of the mineral [m/s]

9.3 Composite

class burnman.composite.**Composite**(*fractions*, *phases=None*)
Bases: burnman.material.Material

Base class for a static composite material with fixed molar fractions. The elements can be minerals or materials, meaning composite can be nested arbitrarily.

This class is available as burnman.Composite.

debug_print(*indent=''*)

density()
Compute the density of the composite based on the molar volumes and masses

print_minerals_of_current_state()
Print a human-readable representation of this Material at the current P, T as a list of minerals. This requires set_state() has been called before.

set_method(*method*)
set the same equation of state method for all the phases in the composite

set_state(*pressure*, *temperature*)
Update the material to the given pressure [Pa] and temperature [K].

to_string()
return the name of the composite

unroll()

9.4 Mineral helpers

class burnman.mineral_helpers.**HelperSpinTransition**(*transition_pressure*, *ls_mat*, *hs_mat*)

Bases: burnman.material.Material

Helper class that makes a mineral that switches between two materials (for low and high spin) based on some transition pressure [Pa]

debug_print(*indent=''*)

density()

print_minerals_of_current_state()
Print a human-readable representation of this Material at the current P, T as a list of minerals. This requires set_state() has been called before.

set_method(*method*)

set_state(*pressure*, *temperature*)

to_string()
Returns a human-readable name of this material. The default implementation will return the name of the class, which is a reasonable default.

Returns **name** : string

Name of this material.

unroll()
return (fractions, minerals) where both are arrays. May depend on current state

class burnman.mineral_helpers.**HelperSolidSolution**(*base_materials*, *molar_fraction*)

Bases: burnman.mineral.Mineral

Class for coming up with a new mineral based based on a solid solution between two or more end member minerals. It is not completely clear how to do this, or how valid this approximation is, but here we just do a weighted arithmetic average of the thermoelastic properties of the end members according to their molar fractions

adiabatic_bulk_modulus()
Returns adiabatic bulk modulus of the mineral [Pa]

debug_print(*indent=''*)

density()
Returns density of the mineral [kg/m^3]

grueneisen_parameter()
Returns grueneisen parameter of the mineral [unitless]

heat_capacity_p()
Returns heat capacity at constant pressure of the mineral [J/K/mol]

heat_capacity_v()
Returns heat capacity at constant volume of the mineral [J/K/mol]

isothermal_bulk_modulus()
Returns isothermal bulk modulus of the mineral [Pa]

molar_mass()
Returns molar mass of the mineral [kg/mol]

molar_volume()
Returns molar volume of the mineral [m^3/mol]

print_minerals_of_current_state()
Print a human-readable representation of this Material at the current P, T as a list of minerals. This requires set_state() has been called before.

set_method(*method*)
Set the equation of state to be used for this mineral. Takes a string corresponding to any of the predefined equations of state: 'bm2', 'bm3', 'mgd2', 'mgd3', 'slb2', or 'slb3'. Alternatively, you can pass a user defined class which derives from the equation_of_state base class.

set_state(*pressure*, *temperature*)

shear_modulus()
Returns shear modulus of the mineral [Pa]

thermal_expansivity()
Returns thermal expansion coefficient of the mineral [1/K]

to_string()
Returns the name of the mineral class

unroll()

v_p()
Returns P wave speed of the mineral [m/s]

v_phi()
Returns bulk sound speed of the mineral [m/s]

v_s()
Returns shear wave speed of the mineral [m/s]

class burnman.mineral_helpers.**HelperFeDependent**(*iron_number_with_pt*, *idx*)
Bases: burnman.material.Material

Helper to implement a rock that does iron exchange (two minerals with changing iron content based on P,T).

Classes deriving from this helper need to implement create_inner_material() and provide a function in __init__ that computes the iron exchange.

create_inner_material(*iron_number*)

debug_print(*indent=''*)

density()

iron_number()

print_minerals_of_current_state()
Print a human-readable representation of this Material at the current P, T as a list of minerals. This requires set_state() has been called before.

set_method(*method*)

set_state(*pressure*, *temperature*)

to_string()
Returns a human-readable name of this material. The default implementation will return the name of the class, which is a reasonable default.

Returns **name** : string

Name of this material.

unroll()
return (fractions, minerals) where both are arrays. May depend on current state

CHAPTER

TEN

SEISMIC

10.1 Base class for all seismic models

class burnman.seismic.**Seismic1DModel**

Base class for all the seismological models.

G(*depth*)

Parameters **depth** : float or array of floats

Shear modulus at given for depth(s) in [Pa].

K(*depth*)

Parameters **depth** : float or array of floats

Bulk modulus at given for depth(s) in [Pa]

density(*depth*)

Parameters **depth** : float or array of floats

Depth(s) [m] to evaluate seismic model at.

Returns **density** : float or array of floats

Density at given depth(s) in [kg/m^3].

depth(*pressure*)

Parameters **pressure** : float or array of floats

Pressure(s) [Pa] to evaluate depth at.

Returns **depth** : float or array of floats

Depth(s) [m] for given pressure(s)

evaluate_all_at(*depth_list*)

Returns the lists of data for a Seismic1DModel for the depths provided

Parameters **depth_list** : array of floats

Array of depths [m] to evaluate seismic model at.

Returns **pressures** : array of floats

Pressures at given depths in [Pa].

density : array of floats

Densities at given depths in [kg/m^3].

v_p : array of floats

P wave velocities at given depths in [m/s].

v_s : array of floats

S wave velocities at given depths in [m/s].

v_phi : array of floats

Bulk sound velocities at given depths in [m/s].

gravity(*depth*)

Parameters **depth** : float or array of floats

Depth(s) [m] to evaluate gravity at.

Returns **gravity** : float or array of floats

Gravity for given depths in [m/s^2]

internal_depth_list()

Returns a sorted list of depths where this seismic data is specified at. This allows you to compare the seismic data without interpolation.

Returns **depths** : array of floats

Depths [m].

pressure(*depth*)

Parameters **depth** : float or array of floats

Depth(s) [m] to evaluate seismic model at.

Returns **pressure** : float or array of floats

Pressure(s) at given depth(s) in [Pa].

v_p(*depth*)

Parameters **depth** : float or array of floats

Depth(s) [m] to evaluate seismic model at.

Returns **v_p** : float or array of floats

P wave velocity at given depth(s) in [m/s].

v_phi(*depth*)

Parameters **depth_list** : float or array of floats

Depth(s) [m] to evaluate seismic model at.

Returns **v_phi** : float or array of floats

bulk sound wave velocity at given depth(s) in [m/s].

v_s (*depth*)

Parameters **depth** : float or array of floats

Depth(s) [m] to evaluate seismic model at.

Returns **v_s** : float or array of floats

S wave velocity at given depth(s) in [m/s].

10.2 Class for 1D Models

class burnman.seismic.**SeismicRadiusTable**

Bases: burnman.seismic.Seismic1DModel

This is a base class that gets a 1D seismic model from a table indexed and sorted by radius. Fill the tables in the constructor after deriving from this class. This class uses burnman.seismic.Seismic1DModel

Note: all tables need to be sorted by increasing radius. Alternatively, you can also overwrite the _lookup function if you want to access with something else.

G (*depth*)

Parameters **depth** : float or array of floats

Shear modulus at given for depth(s) in [Pa].

K (*depth*)

Parameters **depth** : float or array of floats

Bulk modulus at given for depth(s) in [Pa]

density (*depth*)

depth (*pressure*)

evaluate_all_at (*depth_list*)

Returns the lists of data for a Seismic1DModel for the depths provided

Parameters **depth_list** : array of floats

Array of depths [m] to evaluate seismic model at.

Returns **pressures** : array of floats

Pressures at given depths in [Pa].

density : array of floats

Densities at given depths in [kg/m^3].

v_p : array of floats

P wave velocities at given depths in [m/s].

v_s : array of floats

S wave velocities at given depths in [m/s].

v_phi : array of floats

Bulk sound velocities at given depths in [m/s].

gravity(*depth*)

Parameters **depth** : float or array of floats

Depth(s) [m] to evaluate gravity at.

Returns **gravity** : float or array of floats

Gravity for given depths in [m/s^2]

internal_depth_list()

pressure(*depth*)

v_p(*depth*)

v_phi(*depth*)

Parameters **depth_list** : float or array of floats

Depth(s) [m] to evaluate seismic model at.

Returns **v_phi** : float or array of floats

bulk sound wave velocity at given depth(s) in [m/s].

v_s(*depth*)

10.3 Models currently implemented

class burnman.seismic.**PREM**

Bases: burnman.seismic.SeismicRadiusTable

Reads PREM (1s) (input_seismic/prem_table.txt, [DA81]). See also burnman.seismic.SeismicRadiusTable.

G(*depth*)

Parameters **depth** : float or array of floats

Shear modulus at given for depth(s) in [Pa].

K(*depth*)

Parameters **depth** : float or array of floats

Bulk modulus at given for depth(s) in [Pa]

density(*depth*)

depth(*pressure*)

evaluate_all_at(*depth_list*)

Returns the lists of data for a Seismic1DModel for the depths provided

Parameters **depth_list** : array of floats

Array of depths [m] to evaluate seismic model at.

Returns **pressures** : array of floats

Pressures at given depths in [Pa].

density : array of floats

Densities at given depths in [kg/m^3].

v_p : array of floats

P wave velocities at given depths in [m/s].

v_s : array of floats

S wave velocities at given depths in [m/s].

v_phi : array of floats

Bulk sound velocities at given depths in [m/s].

gravity(*depths*)

internal_depth_list()

pressure(*depth*)

v_p(*depth*)

v_phi(*depth*)

Parameters **depth_list** : float or array of floats

Depth(s) [m] to evaluate seismic model at.

Returns **v_phi** : float or array of floats

bulk sound wave velocity at given depth(s) in [m/s].

v_s(*depth*)

class burnman.seismic.**Slow**

Bases: burnman.seismic.SeismicRadiusTable

Inserts the mean profiles for slower regions in the lower mantle (Lekic et al. 2012). We stitch together tables 'input_seismic/prem_lowermantle.txt', 'input_seismic/swave_slow.txt', 'input_seismic/pwave_slow.txt'). See also burnman.seismic.SeismicRadiusTable.

G(*depth*)

Parameters **depth** : float or array of floats

Shear modulus at given for depth(s) in [Pa].

K(*depth*)

Parameters **depth** : float or array of floats

Bulk modulus at given for depth(s) in [Pa]

density(*depth*)

depth(*pressure*)

evaluate_all_at(*depth_list*)

Returns the lists of data for a Seismic1DModel for the depths provided

Parameters **depth_list** : array of floats

Array of depths [m] to evaluate seismic model at.

Returns **pressures** : array of floats

Pressures at given depths in [Pa].

density : array of floats

Densities at given depths in [kg/m^3].

v_p : array of floats

P wave velocities at given depths in [m/s].

v_s : array of floats

S wave velocities at given depths in [m/s].

v_phi : array of floats

Bulk sound velocities at given depths in [m/s].

gravity(*depth*)

Parameters **depth** : float or array of floats

Depth(s) [m] to evaluate gravity at.

Returns **gravity** : float or array of floats

Gravity for given depths in [m/s^2]

internal_depth_list()

pressure(*depth*)

v_p(*depth*)

v_phi(*depth*)

Parameters **depth_list** : float or array of floats

Depth(s) [m] to evaluate seismic model at.

Returns **v_phi** : float or array of floats

bulk sound wave velocity at given depth(s) in [m/s].

v_s (*depth*)

class burnman.seismic.**Fast**
Bases: burnman.seismic.SeismicRadiusTable

Inserts the mean profiles for faster regions in the lower mantle (Lekic et al. 2012). We stitch together tables 'input_seismic/prem_lowermantle.txt', 'input_seismic/swave_fast.txt', 'input_seismic/pwave_fast.txt'). See also burnman.seismic.Seismic1DModel.

G (*depth*)

Parameters **depth** : float or array of floats

Shear modulus at given for depth(s) in [Pa].

K (*depth*)

Parameters **depth** : float or array of floats

Bulk modulus at given for depth(s) in [Pa]

density (*depth*)

depth (*pressure*)

evaluate_all_at (*depth_list*)
Returns the lists of data for a Seismic1DModel for the depths provided

Parameters **depth_list** : array of floats

Array of depths [m] to evaluate seismic model at.

Returns **pressures** : array of floats

Pressures at given depths in [Pa].

density : array of floats

Densities at given depths in [kg/m^3].

v_p : array of floats

P wave velocities at given depths in [m/s].

v_s : array of floats

S wave velocities at given depths in [m/s].

v_phi : array of floats

Bulk sound velocities at given depths in [m/s].

gravity (*depth*)

Parameters **depth** : float or array of floats

Depth(s) [m] to evaluate gravity at.

Returns **gravity** : float or array of floats

Gravity for given depths in [m/s^2]

internal_depth_list()

pressure(*depth*)

v_p(*depth*)

v_phi(*depth*)

Parameters **depth_list** : float or array of floats

Depth(s) [m] to evaluate seismic model at.

Returns **v_phi** : float or array of floats

bulk sound wave velocity at given depth(s) in [m/s].

v_s(*depth*)

10.4 Attenuation Correction

burnman.seismic.**attenuation_correction**(*v_p*, *v_s*, *v_phi*, *Qs*, *Qphi*)

Applies the attenuation correction following Matas et al. (2007), page 4. This is simplified, and there is also currently no 1D Q model implemented. The correction, however, only slightly reduces the velocities, and can be ignored for our current applications. Arguably, it might not be as relevant when comparing computations to PREM for periods of 1s as is implemented here. Called from burnman.main.apply_attenuation_correction()

Parameters **v_p** : float

P wave velocity in [m/s].

v_s : float

S wave velocitiy in [m/s].

v_phi : float

Bulk sound velocity in [m/s].

Qs : float

shear quality factor [dimensionless]

Qphi: float

bulk quality factor [dimensionless]

Returns **v_p** : float

corrected P wave velocity in [m/s].

v_s : float

corrected S wave velocitiy in [m/s].

v_phi : float

corrected Bulk sound velocity in [m/s].

CHAPTER

ELEVEN

MINERAL DATABASE

Mineral database

- Matas_etal_2007
- Murakami_2013
- Murakami_etal_2012
- SLB_2005
- SLB_2011_ZSB_2013
- SLB_2011
- other

11.1 Murakami_2013

Minerals from Murakami 2013 and references therein.

burnman.minerals.Murakami_2013.**fe_bridgmanite**
alias of fe_perovskite

class burnman.minerals.Murakami_2013.**fe_perovskite**
Bases: burnman.mineral.Mineral

class burnman.minerals.Murakami_2013.**ferropericlase**(*fe_num*)
Bases: burnman.mineral_helpers.HelperSolidSolution

burnman.minerals.Murakami_2013.**mg_bridgmanite**
alias of mg_perovskite

burnman.minerals.Murakami_2013.**mg_fe_bridgmanite**
alias of mg_fe_perovskite

class burnman.minerals.Murakami_2013.**mg_fe_perovskite**(*fe_num*)
Bases: burnman.mineral_helpers.HelperSolidSolution

class burnman.minerals.Murakami_2013.**mg_perovskite**
Bases: burnman.mineral.Mineral

class burnman.minerals.Murakami_2013.**periclase**
Bases: burnman.mineral.Mineral

class burnman.minerals.Murakami_2013.**wuestite**
Bases: burnman.mineral.Mineral

11.2 SLB_2011

Minerals from Stixrude & Lithgow-Bertelloni 2011 and references therein

burnman.minerals.SLB_2011.**fe_bridgmanite**
alias of fe_perovskite

class burnman.minerals.SLB_2011.**fe_perovskite**
Bases: burnman.mineral.Mineral

class burnman.minerals.SLB_2011.**ferropericlase**(*fe_num*)
Bases: burnman.mineral_helpers.HelperSolidSolution

class burnman.minerals.SLB_2011.**ferropericlase_pt_dependent**(*iron_number_with_pt*, *idx*)
Bases: burnman.mineral_helpers.HelperFeDependent

create_inner_material(*iron_number*)

burnman.minerals.SLB_2011.**mg_bridgmanite**
alias of mg_perovskite

burnman.minerals.SLB_2011.**mg_fe_bridgmanite**
alias of mg_fe_perovskite

burnman.minerals.SLB_2011.**mg_fe_bridgmanite_pt_dependent**
alias of mg_fe_perovskite_pt_dependent

class burnman.minerals.SLB_2011.**mg_fe_perovskite**(*fe_num*)
Bases: burnman.mineral_helpers.HelperSolidSolution

class burnman.minerals.SLB_2011.**mg_fe_perovskite_pt_dependent**(*iron_number_with_pt*, *idx*)
Bases: burnman.mineral_helpers.HelperFeDependent

create_inner_material(*iron_number*)

class burnman.minerals.SLB_2011.**mg_perovskite**
Bases: burnman.mineral.Mineral

class burnman.minerals.SLB_2011.**periclase**
Bases: burnman.mineral.Mineral

class burnman.minerals.SLB_2011.**stishovite**
Bases: burnman.mineral.Mineral

class burnman.minerals.SLB_2011.**wuestite**
Bases: burnman.mineral.Mineral

11.3 Matas_etal_2007

Minerals from Matas et al. 2007 and references therein. See Table 1 and 2.

burnman.minerals.Matas_etal_2007.**al_bridgmanite**
alias of al_perovskite

class burnman.minerals.Matas_etal_2007.**al_perovskite**
Bases: burnman.mineral.Mineral

burnman.minerals.Matas_etal_2007.**ca_bridgmanite**
alias of ca_perovskite

class burnman.minerals.Matas_etal_2007.**ca_perovskite**
Bases: burnman.mineral.Mineral

burnman.minerals.Matas_etal_2007.**fe_bridgmanite**
alias of fe_perovskite

class burnman.minerals.Matas_etal_2007.**fe_perovskite**
Bases: burnman.mineral.Mineral

burnman.minerals.Matas_etal_2007.**mg_bridgmanite**
alias of mg_perovskite

class burnman.minerals.Matas_etal_2007.**mg_perovskite**
Bases: burnman.mineral.Mineral

class burnman.minerals.Matas_etal_2007.**periclase**
Bases: burnman.mineral.Mineral

class burnman.minerals.Matas_etal_2007.**wuestite**
Bases: burnman.mineral.Mineral

11.4 Murakami_etal_2012

Minerals from Murakami et al. (2012) supplementary table 5 and references therein, V_0 from Stixrude & Lithgow-Bertolloni 2005. Some information from personal communication with Murakami.

burnman.minerals.Murakami_etal_2012.**fe_bridgmanite**
alias of fe_perovskite

class burnman.minerals.Murakami_etal_2012.**fe_periclase**
Bases: burnman.mineral_helpers.HelperSpinTransition

class burnman.minerals.Murakami_etal_2012.**fe_periclase_3rd**
Bases: burnman.mineral_helpers.HelperSpinTransition

class burnman.minerals.Murakami_etal_2012.**fe_periclase_HS**
Bases: burnman.mineral.Mineral

class burnman.minerals.Murakami_etal_2012.**fe_periclase_HS_3rd**
Bases: burnman.mineral.Mineral

class burnman.minerals.Murakami_etal_2012.**fe_periclase_LS**
Bases: burnman.mineral.Mineral

class burnman.minerals.Murakami_etal_2012.**fe_periclase_LS_3rd**
Bases: burnman.mineral.Mineral

class burnman.minerals.Murakami_etal_2012.**fe_perovskite**
Bases: burnman.mineral.Mineral

burnman.minerals.Murakami_etal_2012.**mg_bridgmanite**
alias of mg_perovskite

burnman.minerals.Murakami_etal_2012.**mg_bridgmanite_3rdorder**
alias of mg_perovskite_3rdorder

class burnman.minerals.Murakami_etal_2012.**mg_periclase**
Bases: burnman.mineral.Mineral

class burnman.minerals.Murakami_etal_2012.**mg_perovskite**
Bases: burnman.mineral.Mineral

class burnman.minerals.Murakami_etal_2012.**mg_perovskite_3rdorder**
Bases: burnman.mineral.Mineral

11.5 SLB_2005

Minerals from Stixrude & Lithgow-Bertelloni 2005 and references therein

burnman.minerals.SLB_2005.**fe_bridgmanite**
alias of fe_perovskite

class burnman.minerals.SLB_2005.**fe_perovskite**
Bases: burnman.mineral.Mineral

class burnman.minerals.SLB_2005.**ferropericlase**(*fe_num*)
Bases: burnman.mineral_helpers.HelperSolidSolution

class burnman.minerals.SLB_2005.**ferropericlase_pt_dependent**(*iron_number_with_pt*, *idx*)
Bases: burnman.mineral_helpers.HelperFeDependent

create_inner_material(*iron_number*)

burnman.minerals.SLB_2005.**mg_bridgmanite**
alias of mg_perovskite

burnman.minerals.SLB_2005.**mg_fe_bridgmanite**
alias of mg_fe_perovskite

burnman.minerals.SLB_2005.**mg_fe_bridgmanite_pt_dependent**
alias of mg_fe_perovskite_pt_dependent

class burnman.minerals.SLB_2005.**mg_fe_perovskite**(*fe_num*)
Bases: burnman.mineral_helpers.HelperSolidSolution

class burnman.minerals.SLB_2005.**mg_fe_perovskite_pt_dependent**(*iron_number_with_pt*, *idx*)

Bases: burnman.mineral_helpers.HelperFeDependent

create_inner_material(*iron_number*)

class burnman.minerals.SLB_2005.**mg_perovskite**

Bases: burnman.mineral.Mineral

class burnman.minerals.SLB_2005.**periclase**

Bases: burnman.mineral.Mineral

class burnman.minerals.SLB_2005.**stishovite**

Bases: burnman.mineral.Mineral

class burnman.minerals.SLB_2005.**wuestite**

Bases: burnman.mineral.Mineral

11.6 SLB_2011_ZSB_2013

Minerals from Stixrude & Lithgow-Bertelloni 2011, Zhang, Stixrude & Brodholt 2013, and references therein.

burnman.minerals.SLB_2011_ZSB_2013.**fe_bridgmanite**

alias of fe_perovskite

class burnman.minerals.SLB_2011_ZSB_2013.**fe_perovskite**

Bases: burnman.mineral.Mineral

class burnman.minerals.SLB_2011_ZSB_2013.**ferropericlase**(*fe_num*)

Bases: burnman.mineral_helpers.HelperSolidSolution

class burnman.minerals.SLB_2011_ZSB_2013.**ferropericlase_pt_dependent**(*iron_number_with_pt*, *idx*)

Bases: burnman.mineral_helpers.HelperFeDependent

create_inner_material(*iron_number*)

burnman.minerals.SLB_2011_ZSB_2013.**mg_bridgmanite**

alias of mg_perovskite

burnman.minerals.SLB_2011_ZSB_2013.**mg_fe_bridgmanite**

alias of mg_fe_perovskite

burnman.minerals.SLB_2011_ZSB_2013.**mg_fe_bridgmanite_pt_dependent**

alias of mg_fe_perovskite_pt_dependent

class burnman.minerals.SLB_2011_ZSB_2013.**mg_fe_perovskite**(*fe_num*)

Bases: burnman.mineral_helpers.HelperSolidSolution

class burnman.minerals.SLB_2011_ZSB_2013.**mg_fe_perovskite_pt_dependent**(*iron_number_with_p*
idx)

Bases: burnman.mineral_helpers.HelperFeDependent

create_inner_material(*iron_number*)

class burnman.minerals.SLB_2011_ZSB_2013.**mg_perovskite**
Bases: burnman.mineral.Mineral

class burnman.minerals.SLB_2011_ZSB_2013.**periclase**
Bases: burnman.mineral.Mineral

class burnman.minerals.SLB_2011_ZSB_2013.**stishovite**
Bases: burnman.mineral.Mineral

class burnman.minerals.SLB_2011_ZSB_2013.**wuestite**
Bases: burnman.mineral.Mineral

11.7 Other minerals

class burnman.minerals.other.**Speziale_fe_periclase**
Bases: burnman.mineral_helpers.HelperSpinTransition

class burnman.minerals.other.**Speziale_fe_periclase_HS**
Bases: burnman.mineral.Mineral

Speziale et al. 2007, Mg#=83

class burnman.minerals.other.**Speziale_fe_periclase_LS**
Bases: burnman.mineral.Mineral

Speziale et al. 2007, Mg#=83

References

The functions in the *Main module* operate on *Minerals* which can be combination of different minerals from *Mineral database*.

CHAPTER

TWELVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

[And82a] O. L. Anderson. The earth's core and the phase diagram of iron. *Philos. T. Roy. Soc. A*, 306(1492):21–35, 1982. URL: http://rsta.royalsocietypublishing.org/content/306/1492/21.abstract.

[And82b] OL Anderson. The Earth's core and the phase diagram of iron. *Phil. Trans. R. Soc. Lond., Ser A, Math. Phys. Sci.*, 306(1492):21–35, 1982. URL: http://rsta.royalsocietypublishing.org/content/306/1492/21.short.

[ASA+11] D Antonangeli, J Siebert, CM Aracne, D Farber, A Bosak, M Hoesch, M Krisch, F Ryerson, G Fiquet, and J Badro. Spin crossover in ferropericlase at high pressure: a seismologically transparent transition?. *Science*, 331(6031):64–67, 2011. URL: http://www.sciencemag.org/content/331/6013/64.short.

[BS81] JM Brown and TJ Shankland. Thermodynamic parameters in the Earth as determined from seismic profiles. *Geophys. J. Int.*, 66(3):579–596, 1981. URL: http://gji.oxfordjournals.org/content/66/3/579.short.

[CGDG05] F Cammarano, S Goes, A Deuss, and D Giardini. Is a pyrolitic adiabatic mantle compatible with seismic data?. *Earth Planet. Sci. Lett.*, 232(3):227–243, 2005. URL: http://www.sciencedirect.com/science/article/pii/S0012821X05000804.

[Cam13] F. Cammarano. A short note on the pressure-depth conversion for geophysical interpretation. *Geophysical Research Letters*, 40(18):4834–4838, 2013. URL: http://dx.doi.org/10.1002/grl.50887, doi:10.1002/grl.50887.

[CGR+09] L Cobden, S Goes, M Ravenna, E Styles, F Cammarano, K Gallagher, and JA Connolly. Thermochemical interpretation of 1-D seismic data for the lower mantle: The significance of nonadiabatic thermal gradients and compositional heterogeneity. *J. Geophys. Res.*, 114:B11309, 2009. URL: http://www.agu.org/journals/jb/jb0911/2008JB006262/2008jb006262-t01.txt.

[Con05] JAD Connolly. Computation of phase equilibria by linear programming: a tool for geodynamic modeling and its application to subduction zone decarbonation. *Earth Planet. Sci. Lett.*, 236(1):524–541, 2005. URL: http://www.sciencedirect.com/science/article/pii/S0012821X05002839.

[CHRU14] Sanne Cottaar, Timo Heister, Ian Rose, and Cayman Unterborn. Burnman: a lower mantle mineral physics toolkit. *Geochemistry, Geophysics, Geosystems*, 15(4):1164–1179, 2014. URL: http://dx.doi.org/10.1002/2013GC005122, doi:10.1002/2013GC005122.

[DGD+12] DR Davies, S Goes, JH Davies, BSA Shuberth, H-P Bunge, and J Ritsema. Reconciling dynamic and seismic models of Earth's lower mantle: The domi-

nant role of thermal heterogeneity. *Earth Planet. Sci. Lett.*, 353:253–269, 2012. URL: http://www.sciencedirect.com/science/article/pii/S0012821X1200444X.

[DCT12] F Deschamps, L Cobden, and PJ Tackley. The primitive nature of large low shear-wave velocity provinces. *Earth Planet. Sci. Lett.*, 349-350:198–208, 2012. URL: http://www.sciencedirect.com/science/article/pii/S0012821X12003718.

[DT03] Frédéric Deschamps and Jeannot Trampert. Mantle tomography and its relation to temperature and composition. *Phys. Earth Planet. Int.*, 140(4):277–291, December 2003. URL: http://www.sciencedirect.com/science/article/pii/S0031920103001894, doi:10.1016/j.pepi.2003.09.004.

[DA81] A M Dziewonski and D L Anderson. Preliminary reference Earth model. *Phys. Earth Planet. Int.*, 25(4):297–356, 1981.

[HW12] Y He and L Wen. Geographic boundary of the "Pacific Anomaly" and its geometry and transitional structure in the north. *J. Geophys. Res.-Sol. Ea.*, 2012. URL: http://onlinelibrary.wiley.com/doi/10.1029/2012JB009436/full, doi:DOI: 10.1029/2012JB009436.

[HernandezAlfeB13] ER Hernández, D Alfè, and J Brodholt. The incorporation of water into lower-mantle perovskites: A first-principles study. *Earth Planet. Sci. Lett.*, 364:37–43, 2013. URL: http://www.sciencedirect.com/science/article/pii/S0012821X13000137.

[HHPH13] T. J. B. Holland, N. F. C. Hudson, R. Powell, and B. Harte. New Thermodynamic Models and Calculated Phase Equilibria in NCFMAS for Basic and Ultrabasic Compositions through the Transition Zone into the Uppermost Lower Mantle. *Journal of Petrology*, 54(9):1901–1920, July 2013. URL: http://petrology.oxfordjournals.org/content/54/9/1901.short, doi:10.1093/petrology/egt035.

[HMSL08] C Houser, G Masters, P Shearer, and G Laske. Shear and compressional velocity models of the mantle from cluster analysis of long-period waveforms. *Geophys. J. Int.*, 174(1):195–212, 2008.

[IWSY10] T Inoue, T Wada, R Sasaki, and H Yurimoto. Water partitioning in the Earth's mantle. *Phys. Earth Planet. Int.*, 183(1):245–251, 2010. URL: http://www.sciencedirect.com/science/article/pii/S0031920110001573.

[IS92] Joel Ita and Lars Stixrude. Petrology, elasticity, and composition of the mantle transition zone. *Journal of Geophysical Research*, 97(B5):6849, 1992. URL: http://doi.wiley.com/10.1029/92JB00068, doi:10.1029/92JB00068.

[Jac98] Ian Jackson. Elasticity, composition and temperature of the Earth's lower mantle: a reappraisal. *Geophys. J. Int.*, 134(1):291–311, July 1998. URL: http://gji.oxfordjournals.org/content/134/1/291.abstract, doi:10.1046/j.1365-246x.1998.00560.x.

[JCK+10] Matthew G Jackson, Richard W Carlson, Mark D Kurz, Pamela D Kempton, Don Francis, and Jerzy Blusztajn. Evidence for the survival of the oldest terrestrial mantle reservoir. *Nature*, 466(7308):853–856, 2010.

[KS90] SI Karato and HA Spetzler. Defect microdynamics in minerals and solid-state mechanisms of seismic wave attenuation and velocity dispersion in the mantle. *Rev. Geophys.*, 28(4):399–421, 1990. URL: http://onlinelibrary.wiley.com/doi/10.1029/RG028i004p00399/full.

[KEB95] BLN Kennett, E R Engdahl, and R Buland. Constraints on seismic velocities in the Earth from traveltimes. *Geophys. J. Int.*, 122(1):108–124, 1995. URL: http://gji.oxfordjournals.org/content/122/1/108.short.

[KHM+12] Y Kudo, K Hirose, M Murakami, Y Asahara, H Ozawa, Y Ohishi, and N Hirao. Sound velocity measurements of CaSiO_3 perovskite to 133 GPa and implications for lowermost mantle seismic anomalies. *Earth Planet. Sci. Lett.*, 349:1–7, 2012. URL: http://www.sciencedirect.com/science/article/pii/S0012821X1200324X.

[KEkstromD08] B Kustowski, G Ekström, and AM Dziewoński. Anisotropic shear-wave velocity structure of the Earth's mantle: a global model. *J. Geophys. Res.*, 113(B6):B06306, 2008. URL: http://www.agu.org/pubs/crossref/2008/2007JB005169.shtml.

[LCDR12] V Lekic, S Cottaar, A M Dziewonski, and B Romanowicz. Cluster analysis of global lower mantle tomography: A new class of structure and implications for chemical heterogeneity. *Earth Planet. Sci. Lett.*, 357-358:68–77, 2012. URL: http://www.sciencedirect.com/science/article/pii/S0012821X12005109.

[LvdH08] C Li and RD van der Hilst. A new global model for P wave speed variations in Earth's mantle. *Geochem. Geophys. Geosyst.*, 2008. URL: http://onlinelibrary.wiley.com/doi/10.1029/2007GC001806/full.

[LSMM13] JF Lin, S Speziale, Z Mao, and H Marquardt. Effects of the electronic spin transitions of iron in lower mantle minerals: Implications for deep mantle geophysics and geochemistry. *Rev. Geophys.*, 2013. URL: http://onlinelibrary.wiley.com/doi/10.1002/rog.20010/full.

[LVJ+07] Jung-Fu Lin, György Vankó, Steven D. Jacobsen, Valentin Iota, Viktor V. Struzhkin, Vitali B. Prakapenka, Alexei Kuznetsov, and Choong-Shik Yoo. Spin transition zone in Earth's lower mantle. *Science*, 317(5845):1740–1743, 2007. URL: http://www.sciencemag.org/content/317/5845/1740.abstract, arXiv:http://www.sciencemag.org/content/317/5845/1740.full.pdf.

[MLS+11] Z Mao, JF Lin, HP Scott, HC Watson, VB Prakapenka, Y Xiao, P Chow, and C McCammon. Iron-rich perovskite in the Earth's lower mantle. *Earth Planet. Sci. Lett.*, 309(3):179–184, 2011. URL: http://www.sciencedirect.com/science/article/pii/S0012821X11004018.

[MBR+07] J Matas, J Bass, Y Ricard, E Mattern, and MST Bukowinski. On the bulk composition of the lower mantle: predictions and limitations from generalized inversion of radial seismic profiles. *Geophys. J. Int.*, 170(2):764–780, 2007. URL: http://onlinelibrary.wiley.com/doi/10.1111/j.1365-246X.2007.03454.x/full.

[MB07] J Matas and MST Bukowinski. On the anelastic contribution to the temperature dezpendence of lower mantle seismic velocities. *Earth Planet. Sci. Lett.*, 259(1):51–65, 2007. URL: http://www.sciencedirect.com/science/article/pii/S0012821X07002555.

[MMRB05] E. Mattern, J. Matas, Y. Ricard, and J. Bass. Lower mantle composition and temperature from mineral physics and thermodynamic modelling. *Geophys. J. Int.*, 160(3):973–990, March 2005. URL: http://gji.oxfordjournals.org/cgi/doi/10.1111/j.1365-246X.2004.02549.x, doi:10.1111/j.1365-246X.2004.02549.x.

[MS95] WF McDonough and SS Sun. The composition of the Earth. *Chem. Geol.*, 120(3):223–253, 1995. URL: http://www.sciencedirect.com/science/article/pii/0009254194001404.

[MA81] JB Minster and DL Anderson. A model of dislocation-controlled rheology for the mantle. *Phil. Trans. R. Soc. Lond.*, 299(1449):319–359, 1981. URL: http://rsta.royalsocietypublishing.org/content/299/1449/319.short.

[MCD+12] I Mosca, L Cobden, A Deuss, J Ritsema, and J Trampert. Seismic and mineralogical structures of the lower mantle from probabilistic tomography. *J. Geophys. Res.: Solid Earth*, 2012. URL: http://onlinelibrary.wiley.com/doi/10.1029/2011JB008851/full.

[Mur13] M Murakami. 6 Chemical Composition of the Earth's Lower Mantle: Constraints from Elasticity. In *Physics and Chemistry of the Deep Earth (ed S.-I. Karato)*, pages 183–212. John Wiley & Sons, Ltd, Chichester, UK, 2013. URL: http://books.google.com/books?hl=en&lr=&id=7z9yES2XNyEC&pgis=1.

[MOHH12] M Murakami, Y Ohishi, N Hirao, and K Hirose. A perovskitic lower mantle inferred from high-pressure, high-temperature sound velocity data.. *Nature*, 485(7396):90–94, 2012.

[MSH+07] M Murakami, S Sinogeikin, H Hellwig, J Bass, and J Li. Sound velocity of MgSiO3 perovskite to Mbar pressure. *Earth Planet. Sci. Lett.*, 256(1-2):47–54, April 2007. URL: http://www.sciencedirect.com/science/article/pii/S0012821X07000167, doi:10.1016/j.epsl.2007.01.011.

[MOHH09] Motohiko Murakami, Yasuo Ohishi, Naohisa Hirao, and Kei Hirose. Elasticity of MgO to 130 GPa: Implications for lower mantle mineralogy. *Earth Planet. Sci. Lett.*, 277(1-2):123–129, January 2009. URL: http://www.sciencedirect.com/science/article/pii/S0012821X08006675, doi:10.1016/j.epsl.2008.10.010.

[MegninR00] C Mégnin and B Romanowicz. The three-dimensional shear velocity structure of the mantle from the inversion of body, surface and higher-mode waveforms. *Geophys. J. Int.*, 143(3):709–728, 2000.

[NTDC12] T Nakagawa, PJ Tackley, F Deschamps, and JAD Connolly. Radial 1-D seismic structures in the deep mantle in mantle convection simulations with self-consistently calculated mineralogy. *Geochem. Geophys. Geosyst.*, 2012. URL: http://onlinelibrary.wiley.com/doi/10.1029/2012GC004325/full.

[NFR12] Y Nakajima, DJ Frost, and DC Rubie. Ferrous iron partitioning between magnesium silicate perovskite and ferropericlase and the composition of perovskite in the Earth's lower mantle. *J. Geophys. Res.*, 2012. URL: http://onlinelibrary.wiley.com/doi/10.1029/2012JB009151/full.

[NKHO13] M Noguchi, T Komabayashi, K Hirose, and Y Ohishi. High-temperature compression experiments of CaSiO_3 perovskite to lowermost mantle conditions and its thermal equation of state. *Phys. Chem. Miner.*, 40(1):81–91, 2013. URL: http://link.springer.com/article/10.1007/s00269-012-0549-1.

[NOT+11] R Nomura, H Ozawa, S Tateno, K Hirose, J Hernlund, S Muto, H Ishii, and N Hiraoka. Spin crossover and iron-rich silicate melt in the Earth's deep mantle. *Nature*, 473(7346):199–202, 2011. URL: http://www.nature.com/nature/journal/v473/n7346/abs/nature09940.html.

[PR06] M Panning and B Romanowicz. A three-dimensional radially anisotropic model of shear velocity in the whole mantle. *Geophys. J. Int.*, 167(1):361–379, 2006.

[Poi91] JP Poirier. *Introduction to the Physics of the Earth*. Cambridge Univ. Press, Cambridge, England, 1991.

[RDvHW11] J Ritsema, A Deuss, H. J. van Heijst, and J.H. Woodhouse. S40RTS: a degree-40 shear-velocity model for the mantle from new Rayleigh wave dispersion, teleseismic traveltime and normal-mode splitting function. *Geophys. J. Int.*, 184(3):1223–1236, 2011. URL: http://onlinelibrary.wiley.com/doi/10.1111/j.1365-246X.2010.04884.x/full.

[SZN12] BSA Schuberth, C Zaroli, and G Nolet. Synthetic seismograms for a synthetic Earth: long-period P- and S-wave traveltime variations can be explained by temperature alone. *Geophys. J. Int.*, 188(3):1393–1412, 2012. URL: http://onlinelibrary.wiley.com/doi/10.1111/j.1365-246X.2011.05333.x/full.

[SFBG10] NA Simmons, AM Forte, L Boschi, and SP Grand. GyPSuM: A joint tomographic model of mantle density and seismic wave speeds. *J. Geophys. Res.*, 115(B12):B12310, 2010. URL: http://www.agu.org/pubs/crossref/2010/2010JB007631.shtml.

[SMJM12] NA Simmons, SC Myers, G Johanneson, and E Matzel. LLNL-G3Dv3: Global P wave tomography model for improved regional and teleseismic travel time prediction. *J. Geophys. Res.*, 2012. URL: http://onlinelibrary.wiley.com/doi/10.1029/2012JB009525/full.

[SLB05] L Stixrude and C Lithgow-Bertelloni. Thermodynamics of mantle minerals–I. Physical properties. *Geophys. J. Int.*, 162(2):610–632, 2005. URL: http://onlinelibrary.wiley.com/doi/10.1111/j.1365-246X.2005.02642.x/full.

[SLB11] L Stixrude and C Lithgow-Bertelloni. Thermodynamics of mantle minerals–II. Phase equilibria. *Geophys. J. Int.*, 184(3):1180–1213, 2011. URL: http://onlinelibrary.wiley.com/doi/10.1111/j.1365-246X.2010.04890.x/full.

[SLB12] L Stixrude and C Lithgow-Bertelloni. Geophysics of chemical heterogeneity in the mantle. *Annu. Rev. Earth Planet. Sci.*, 40:569–595, 2012. URL: http://www.annualreviews.org/doi/abs/10.1146/annurev.earth.36.031207.124244.

[SDG11] Elinor Styles, D. Rhodri Davies, and Saskia Goes. Mapping spherical seismic into physical structure: biases from 3-D phase-transition and thermal boundary-layer heterogeneity. *Geophys. J. Int.*, 184(3):1371–1378, March 2011. URL: http://gji.oxfordjournals.org/cgi/doi/10.1111/j.1365-246X.2010.04914.x, doi:10.1111/j.1365-246X.2010.04914.x.

[Tac00] PJ Tackley. Mantle convection and plate tectonics: Toward an integrated physical and chemical theory. *Science*, 288(5473):2002–2007, 2000. URL: http://www.sciencemag.org/content/288/5473/2002.short.

[TRCT05] A To, B Romanowicz, Y Capdeville, and N Takeuchi. 3D effects of sharp boundaries at the borders of the African and Pacific Superplumes: Observation and modeling. *Earth Planet. Sci. Lett.*, 233(1-2):1447–1460, 2005.

[TDRY04] Jeannot Trampert, Frédéric Deschamps, Joseph Resovsky, and Dave Yuen. Probabilistic tomography maps chemical heterogeneities throughout the lower mantle.. *Science (New York, N.Y.)*, 306(5697):853–6, October 2004. URL: http://www.sciencemag.org/content/306/5697/853.full, doi:10.1126/science.1101996.

[TVV01] Jeannot Trampert, Pierre Vacher, and Nico Vlaar. Sensitivities of seismic velocities to temperature, pressure and composition in the lower mantle. *Phys. Earth Planet. Int.*, 124(3-4):255–267, August 2001. URL: http://www.sciencedirect.com/science/article/pii/S0031920101002011, doi:10.1016/S0031-9201(01)00201-1.

[WB07] E. Bruce Watson and Ethan F. Baxter. Diffusion in solid-earth systems. *Earth and Planetary Science Letters*, 253(3–4):307 – 327, 2007. URL: http://www.sciencedirect.com/science/article/pii/S0012821X06008168, doi:http://dx.doi.org/10.1016/j.epsl.2006.11.015.

[WDOConnell76] JP Watt, GF Davies, and RJ O'Connell. The elastic properties of composite materials. *Rev. Geophys.*, 14(4):541–563, 1976. URL: http://onlinelibrary.wiley.com/doi/10.1029/RG014i004p00541/full.

[WJW13] Z Wu, JF Justo, and RM Wentzcovitch. Elastic Anomalies in a Spin-Crossover System: Ferropericlase at Lower Mantle Conditions. *Phys. Rev. Lett.*, 110(22):228501, 2013. URL: http://prl.aps.org/abstract/PRL/v110/i22/e228501.

[ZSB13] Zhigang Zhang, Lars Stixrude, and John Brodholt. Elastic properties of MgSiO3-perovskite under lower mantle conditions and the composition of the deep Earth. *Earth Planet. Sci. Lett.*, 379:1–12, October 2013. URL: http://www.sciencedirect.com/science/article/pii/S0012821X13004093, doi:10.1016/j.epsl.2013.07.034.

b

e

m

t

A

B

C

D

G

H

I

K

M

P

R

S

T

U

V

W