property_calcs

- Notes
- Loading
- Constants
  - Properties
  - Type (shared)
  - Type (density)
  - Type (magnetic susceptibility)
  - Type (seismic velocities)
- Property methods
  - calculate_density(mineralogy={}, type=1, density_db={}, molar_mass_db={}, molar_volume_db={})
  - calculate_linear_average_density(mineralogy={}, density_db={})
  - calculate_mass_from_chemistry(molar_mineralogy, molar_mass_db)
  - calculate_volume_from_chemistry(molar_mineralogy, molar_volume_db)
  - calculate_magsus(mineralogy={}, type=1, magsus_db={})
  - calculate_magsus_linear_linear(mineralogy={}, magsus_db={})
  - calculate_magsus_log_log(mineralogy={}, magsus_db={})
  - calculate_magsus_werner_1945(mineralogy, magsus_db)
  - calculate_magsus_mooney_bleifuss_1953(mineralogy, magsus_db)
  - calculate_magsus_grant_west_1965(mineralogy, magsus_db)
  - calculate_magsus_parasnis_1973(mineralogy, magsus_db)
  - calculate_seismic_velocity(mineralogy={}, type=1, properties=1, vp_database={}, vs_database={}, vrh_database={})
  - seismo_linear_average(mineralogy, seismo_database)
  - seismo_time_average(mineralogy, seismo_database)
  - seismo_connolly_vrh(mineralogy, properties, vrh_database)
- Helper methods
  - import_single_property_db(db_name, DelimType = csv.excel)
  - csv_reader(filename)
  - scale_mineralogy(mineralogy={})
- Testing

Notes

Property calcs is the workhorse of the process to add physical properties, and is now a library that can be used within any python script.

This library provides some methods to estimate physical properties based on mineralogy, and some supporting routines.

In this version, there are no attempts to account for properties at various temperatures and pressures, i.e. we are effectively producing the properties of 'lab' samples, although the mineralogy may only be stable at temperatures and pressures beyond that of 'lab' samples. The methods are mainly sourced either from fundamental chemistry, or from the following references:

- Practical Handbook of Physical Properties of Rocks and Minerals, (1989). Carmichael, R.S. ed, CRC Press, Boca Raton, Florida, 741p. ISBN: 0-8493-3703-8
- Ji, S., Wang, Q., Xia, B., (2002). Handbook of Seismic Properties of Minerals, Rocks and Ores, Ecole Polytechnique de Montreal, Canada, 630p.

- Connolly, J., (2007). Calculation of Seismic Velocities with Perple_X: Data Preparation and Computational Algorithm. http://www.perplex.ethz.ch/perplex_seismic_velocity_I.html

The Voigt-Reuss-Hill method of seismic velocity estimation has currently not been implemented, as a compilation of suitable properties for current purposes has not yet been derived. Expected in version 3.

The use of linear averaging in log-log space for magnetic susceptibility determination is unstable and should not be used.

Loading

To use this library, one simply needs to import this script into their own python script. property_calcs.py can be added to the site-packages Python repository, or it could be left in the same directory as the script that you wish to use it with. The library is then loaded as per normal Python:

```
import property_calcs.py
```

Constants

These are used to define various flags for use in the methods.

Properties

prop_vp = 1
: Calculate Vp only.

prop_vs = 2
: Calculate Vs only.

prop_vpvs = 3
: Calculate Vp/Vs ratio only.

prop_all = 4
: Calculate all properties.

Type (shared)

linear_average = 1
: Calculate using a linear average

Type (density)

from_chemistry = 2
: Calculate using chemical calculations

Type (magnetic susceptibility)

log_log_average = 2
: Calculate using a linear average in log-log space.

werner_1945 = 3
: Calculate using Werner (1945)'s method.

mooney_bleifuss_1953 = 4
: Calculate using Mooney & Bleifuss (1953)'s method.

grant_west_1965 = 5
: Calculate using Grant & West (1965)'s method.

parasnis_1973 = 6
: Calculate using Parasnis (1973)'s method.

Type (seismic velocities)

time_average = 2
: Calculate using the time (Backus') averaging.

connolly_vrh = 3
: Calculate using VRH, as implemented by Connolly (CURRENTLY UNUSED!).

Property methods

calculate_density(mineralogy={}, type=1, density_db={}, molar_mass_db={}, molar_volume_db={})

Preconditions:

- We have a series of constants - look in the constants_area
- Mineralogy is a dictionary of mineral, proportion. If you're doing a linear average, proportion had better be a vol %. If you're doing it on chemistry, you MUST have proportion as MOLS!
- type is an integer, set by the constants above.
  - linear_average = 1
  - from_chemistry = 2

  Default is linear_average
- density_db (optional), is a dictionary of mineral, density pairs This is needed if you want to calculate a linear averaged density!
- molar_mass_db (optional), is a dictionary of mineral, molar mass pairs. This is needed if you want to calculate a density from chemistry!
- molar_volume_db (optional), is a dictionary of mineral, molar volume pairs. This is needed if you want to calculate a density from chemistry!

Postconditions:
The method returns a float, representing the density of the sample represented by mineralogy, calculated by your choice of method. The method has failed if you get a return value of -1

calculate_linear_average_density(mineralogy={}, density_db={})

Calculates a density from a mineralogy and a database of densities for each mineral

Preconditions:

- Mineralogy is a dictionary of mineral name, vol % pairs
- density_db is a dictionary of mineral name, density pairs

Postconditions:

- Returns -1 if everything falls apart
- Won't return a density equal to zero.
- Density will be returned otherwise

calculate_mass_from_chemistry(molar_mineralogy, molar_mass_db)

Calculates the mass of a given mineralogy.

Preconditions:

- molar_mineralogy is a database of mineral, mols pairs
- molar_mass_db is a database of mineral, molar mass pairs, can be derived from a chemical system file from UT2K?

Postconditions:

- The calculated mass is returned.

calculate_volume_from_chemistry(molar_mineralogy, molar_volume_db)

Calculates the volume of a given mineralogy.

Preconditions:

- molar_mineralogy is a database of mineral, mols pairs
- molar_volume_db is a database of mineral, molar volume for that mineral pair, can be derived from a CSY file derived from UT2K?

calculate_magsus(mineralogy={}, type=1, magsus_db={})

Preconditions:

- We have a series of constants -- go check up in the constants_area
- mineralogy is a dictionary of mineral, vol% pairs
- magsus_db is a dictionary of mineral, magnetic susceptibility pairs

Postconditions:

- Returns a magnetic susceptibility, calculated by the appropriate method, returns -1 if something has gone wrong in the calculation

calculate_magsus_linear_linear(mineralogy={}, magsus_db={})

Calculate magsus using linear averaging in linear (magsus) space - linear magnetic material space. This is an acceptable assumption to calculate magsus from mineralogy, see Figure 27, Carmichael, R.S. (1989), Magnetic Properties of Minerals and Rocks, p. 347. In Carmichael, R.S. (Ed), 1989, Practical Handbook of Physical Propertiers of Rocks and Minerals, CRC Press, 741 p.

Preconditions:

- mineralogy is a dictionary of mineral, volume proportion
- magsus_db is a dictionary of mineral, magsus values

Postconditions:

- The magnetic susceptibility will be returned, in whatever units are used in magsus_db

calculate_magsus_log_log(mineralogy={}, magsus_db={})

Calculate magsus using linear averaging in log (magsus) - log (mag content) space. Idea taken from Figure 23, Carmichael, R.S. (1989), Magnetic Properties of Minerals and Rocks, p. 347. In Carmichael, R.S. (Ed), 1989, Practical Handbook of Physical Propertiers of Rocks and Minerals, CRC Press, 741 p.

Preconditions:

- mineralogy is a dictionary of mineral, volume proportion
- magsus_db is a dictionary of mineral, magsus values

Postconditions:

- The magnetic susceptibility will be returned, in whatever units are used in magsus_db

Notes: This currently does not not work well when you have high concentrations material with a low magnetic susceptibility!

calculate_magsus_werner_1945(mineralogy, magsus_db)

Calculate magnetic susceptibility using the formula of Werner: Werner, S. (1945). Determinations of the magnetic susceptibilities of ores and rocks from Swedish iron ore deposits. Swedish Geological Survey, 39, 1.

Preconditions:

- mineralogy is a dictionary of mineral, volume proportion
- magsus_db is a dictionary of mineral, magsus values

Postconditions:

- The magnetic susceptibility will be returned, calculated in SI values

calculate_magsus_mooney_bleifuss_1953(mineralogy, magsus_db)

Calculate magnetic susceptibility using the formula of Mooney and Bleifuss: Mooney, H.M. and Bleifuss, R. (1953). Magnetic susceptibility measurements in Minnesota. II. Analysis of field results, Geophysics 18, 383.

Preconditions:

- mineralogy is a dictionary of mineral, volume proportion
- magsus_db is a dictionary of mineral, magsus values

Postconditions:

- The magnetic susceptibility will be returned, calculated in SI values

calculate_magsus_grant_west_1965(mineralogy, magsus_db)

Calculate magnetic susceptibility using the formula of Grant and West: Grant, F.S. and West, G.F. (1965). Interpretation theory in applied geophysics, McGraw? -Hill, New York.

Preconditions:

- mineralogy is a dictionary of mineral, volume proportion
- magsus_db is a dictionary of mineral, magsus values

Postconditions:

- The magnetic susceptibility will be returned, calculated in SI values

calculate_magsus_parasnis_1973(mineralogy, magsus_db)

Calculate magnetic susceptibility using the formula of Parasnis: Parasnis, D.S. (1973). Mining geophysics, revised ed., Elsevier, Amsterdam.

Preconditions:

- mineralogy is a dictionary of mineral, volume proportion
- magsus_db is a dictionary of mineral, magsus values

Postconditions:

- The magnetic susceptibility will be returned, in the units as used in magsus_db

calculate_seismic_velocity(mineralogy={}, type=1, properties=1, vp_database={}, vs_database={}, vrh_database={})

Preconditions:

- mineralogy is a dictionary of mineral, vol% pairs
- type is an integer, set by the constants up above
- properties is an integer, set by the constants up above

- vp_database (optional), is a dictionary of mineralogy, vp pairs. This is needed for calculating vp, vpvs, or all, unless you are calculating by VRH!
- vs_database (optional), is a dictionary of mineralogy, vp pairs. This is needed for calculating vp, vpvs, or all, unless you are calculating by VRH!
- vrh_database (optional), is a dictionary of mineralogy, [mineralogy_parameters list] pairs. This is needed if you wish to calculate by VRH!
- Method also requires access to the following other methods:
    - seismo_linear_average() if you wish to calculate via linear average.
    - seismo_time_average() if you wish to calculate by time (Backus) average.
    - seismo_connolly_vrh() if you wish to calculate using Connolly's VRH implementation.

seismo_linear_average(mineralogy, seismo_database)

Postconditions:
The method either returns a float, representing either vp, vs or vp/vs ratio, depending on desired calculation type, or it returns a tuple of vp, vs, vp/vs if you require all properties.

Preconditions:
- We have a series of constants -- go check up in the constants_area
- mineralogy is a dictionary of mineral, vol% pairs
- seismo_database is a dictionary of mineral, property pairs. This is whatever property you wish to calculate by linear average

Postconditions:
The method either returns a float, representing the linear average of the velocities

seismo_time_average(mineralogy, seismo_database)

Uses Backus' averaging method.
This is:
(velocity)^1 = volprop1 / mineral_velocity_1 + volprop2 / mineral_velocity_2 + ...

Preconditions:
- We have a series of constants -- go check up in the constants_area
- mineralogy is a dictionary of mineral, vol% pairs
- seismo_database is a dictionary of mineral, property pairs. This is whatever property you wish to calculate by linear average

Postconditions: The method either returns a float, representing the time average of the velocities

seismo_connolly_vrh(mineralogy, properties, vrh_database)

Currently not implemented.

Helper methods

import_single_property_db(db_name, DelimType? = csv.excel)

This is the method to import a database of a single property. Pretty simple, given a filename, with or without path, it opens a file and spits back a dictionary of mineral, property pairs.

Preconditions:
- Requires the csv module to have been imported
- Expects a dialect called TabDelim? to have been created

- Database format is "mineral nameTABsome_property"

Postconditions:
- Database file is closed
- A dictionary of mineral, velocity pairs has been returned.

csy_reader(filename)

This is a way of reading in the CSY file and is to be used until we handle the proper pmdPyRT method of storing the abstracted chemical system.

Note we store the mineral names internally in lower case, this makes comparisons easier.

Preconditions:
- Requires a filename to be fed to it (name will do, with a path is even better)
- Requires CSV to have been imported, and a SpaceDelim? dialect created
- Needs some constants (see top of file)

Postconditions:
- Spits out a list of dictionaries
  - First dictionary is the molar volume dictionary
  - Second dictionary is the molar mass dictionary
  - Each output dictionary is a list of mineral, property pairs

scale_mineralogy(mineralogy={})

Scales the mineralogy to total 1, i.e. represent mineralogy as proportion.

Preconditions:
- mineralogy is a dictionary of mineral, some_value pairs.

Postconditions:
- A dictionary of mineralogy, proportion pairs will be returned.

Testing

This library includes rudimentary unit testing methods. The tests examine each calculation library, as well as database import and export methods. The testing routines are be located in the testing area of the program. To test, simply call the library as a python script:

```
python property_calcs.py
```