
Periodic Table Documentation

Release 0.9

Paul Kienzle

December 07, 2010

CONTENTS

1	User's Guide	3
1.1	Periodic Table of Elements	3
1.2	Basic usage	5
1.3	Chemical Composition	7
1.4	Bundling with py2exe	10
1.5	Adding properties	11
1.6	Custom tables	13
1.7	Data Sources	14
1.8	Contributing Changes	14
1.9	License	15
1.10	Disclaimer	15
1.11	Credits	16
2	Reference	17
2.1	Core table	17
2.2	Chemical formula	23
2.3	Covalent radius	27
2.4	Crystal structure	28
2.5	Density	28
2.6	Mass	30
2.7	Neutron scattering potentials	31
2.8	X-ray scattering potentials	40
2.9	Magnetic Form Factor	45
3	Indices and Tables	47
	Python Module Index	49
	Index	51

The periodictable package provides an extensible periodic table of the elements pre-populated with data important to neutron and X-ray scattering experiments. Periodic table is written entirely in Python and does not require any external libraries.

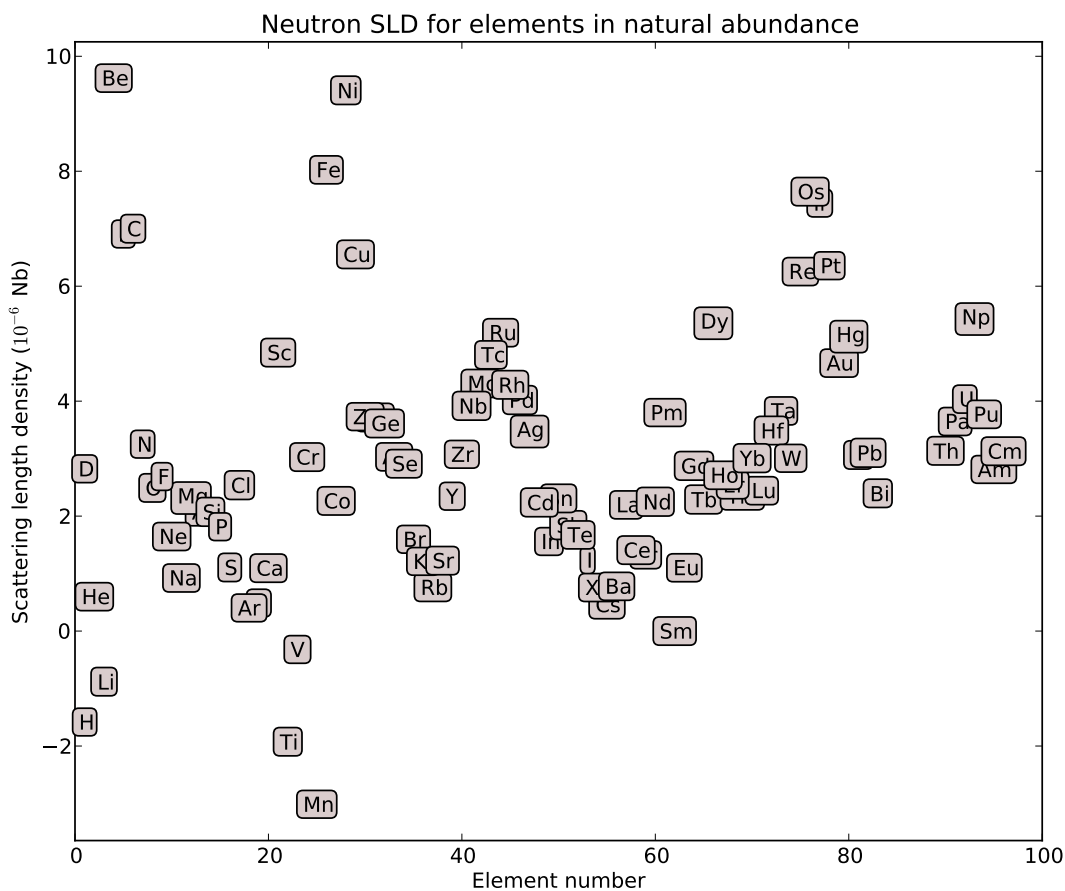
USER'S GUIDE

This section gives an overview and introduction to Periodic Table. Read this to have an idea about what Periodic Table can do for you (and how) and if you want to know in detail about Periodic Table, refer to the *Periodic Table Modules Reference*.

1.1 Periodic Table of Elements

The periodictable package provides an extensible periodic table with various properties of the elements like name, symbol, mass, density etc and also provides data important to neutron and X-ray scattering experiments. With the elements package you can compute the scattering potential of a compound at a given wavelength.

There is a wealth of possible information that could be stored in such a table, and collecting it all is far beyond the scope of this project. Instead, we provide an extensible table in which third party packages can provide properties in addition to the base properties we define here.



Neutron SLD as a function of element.

1.1.1 Features

Standard properties Name, symbol, `mass` and `density` of elements are built in.

Chemical Formula Parses chemical formula and computes properties such as molar mass.

Isotopes Mass and relative abundance of isotopes are included for known isotopes. Formulas can include isotope composition.

Ions Magnetic form factors and ionic radii for various ions.

Neutron and X-ray Scattering Factors Provides neutron and wavelength dependent X-ray scattering factors for elements, isotopes, and formulas.

Extensible New properties can be added to the `Periodic Table` from outside the package. Specialized tables can be created with alternatives to the standard values.

Data Sources References are available for all information in the table.

1.2 Basic usage

The periodic table is available on PyPI, and can be obtained simply with:

```
easy_install periodictable
```

This will install pyparsing if it is not already available. The numpy package must already be installed.

Access particular elements by name:

```
>>> from periodictable import hydrogen
>>> print "H mass", hydrogen.mass, hydrogen.mass_units
H mass 1.00794 u
```

Access particular elements as symbols:

```
>>> from periodictable import H,B,Cu,Ni
>>> print "B absorption", B.neutron.absorption
B absorption 767.0
>>> print "Ni f1/f2 for Cu K-alpha X-rays", Ni.xray.scattering_factors(wavelength=Cu.K_alpha)
Ni f1/f2 for Cu K-alpha X-rays (25.022929905648375, 0.52493074546535157)
```

Access isotopes using mass number subscripts:

```
>>> print "58-Ni vs 62-Ni scattering", Ni[58].neutron.coherent, Ni[62].neutron.coherent
58-Ni vs 62-Ni scattering 26.1 9.5
```

Access elements indirectly:

```
>>> import periodictable
>>> print "Cd density", periodictable.Cd.density, periodictable.Cd.density_units
Cd density 8.65 g/cm^3
```

Import all elements:

```
>>> from periodictable import *
>>> print periodictable.H
H
>>> print periodictable.H.mass
1.00794
```

Deuterium and tritium are special isotopes named D and T some neutron information is available as 'n':

```
>>> print "D mass",D.mass
D mass 2.014101778
>>> print "neutron mass",n.mass
neutron mass 1.00866491597
```

Process all the elements:

```
>>> import periodictable
>>> for el in periodictable.elements:
...     print el.symbol,el.name
n neutron
H hydrogen
He helium
...
Uuh ununhexium
```

Another example for processing all elements:

```
>>> from periodictable import elements
>>> for el in elements:
...     print el.symbol,el.number
n 0
H 1
He 2
...
```

Process all the isotopes for an element:

```
>>> for iso in periodictable.H:
...     print iso,iso.mass
1-H 1.0078250321
D 2.014101778
T 3.0160492675
4-H 4.02783
5-H 5.03954
6-H 6.04494
```

You can create a unique handle to an individual ion. In addition to storing the ion charge, this can be used to reference the underlying properties of the element or isotope:

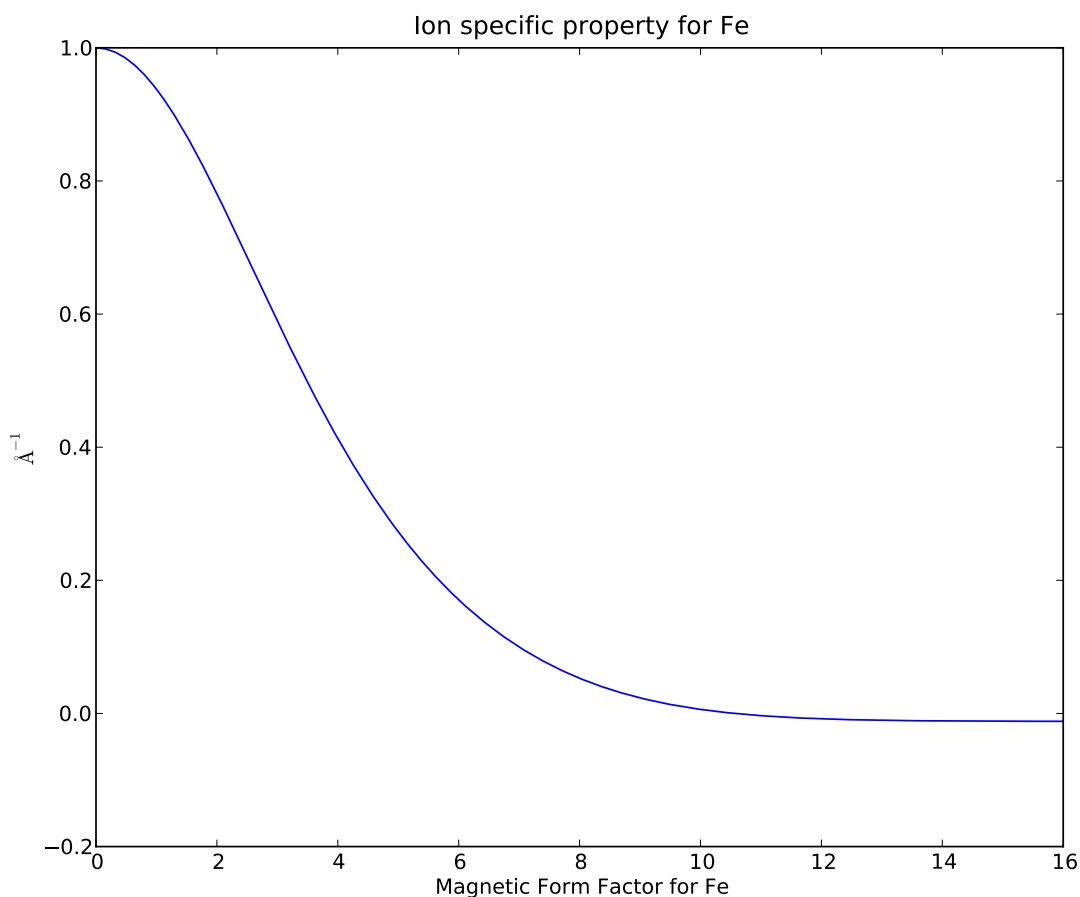
```
>>> Ni58_2 = periodictable.Ni[58].ion[2]
>>> Ni_2 = periodictable.Ni.ion[2]
>>> print "charge for Ni2+",Ni_2.charge
charge for Ni2+ 2
>>> print "mass for Ni[58] and for natural abundance", Ni58_2.mass, Ni_2.mass
mass for Ni[58] and for natural abundance 57.9353479 58.6934
```

The ion specific properties can be accessed from the ion using `ion.charge` for the ion index:

```
>>> import pylab
>>> import periodictable
>>> Fe_2 = periodictable.Fe.ion[2]
>>> print Fe_2.magnetic_ff[Fe_2.charge].M_Q([0,0.1,0.2])
[ 1.          0.99935255  0.99741366]
```

The following is a plot of the magnetic form factor vs. Q :

```
>>> Q = pylab.linspace(0,16,200)
>>> M = Fe_2.magnetic_ff[Fe_2.charge].j0_Q(Q)
>>> pylab.xlabel(r'Magnetic Form Factor for Fe')
>>> pylab.ylabel(r'$\AA^{-1}$')
>>> pylab.title('Ion specific property for Fe')
>>> pylab.plot(Q,M)
```



Missing properties generally evaluate to *None*:

```
>>> print "Radon density",periodictable.Rn.density
Radon density None
```

Specific defined properties related to elements can be accessed in a table format as shown in following example :

```
>>> elements.list('symbol','K_alpha',format="%s K-alpha = %s")
Ti K-alpha = 2.7496
Cr K-alpha = 2.2909
Mn K-alpha = 2.1031
...
Ag K-alpha = 0.5608
```

1.3 Chemical Composition

Some properties are available for groups of elements. Groups are specified as a chemical formula string and either density or cell volume for the crystal structure. While it does not provide any information about molecular structure, a formula does provide complete control over chemical composition.

A formula string is translated into a formula using `periodictable.formulas.formula()`:

- Formula strings consist of counts and atoms, where individual atoms are represented by periodic table symbol. The atoms are case sensitive, so “CO” is different from “Co”. Here is an example of calcium carbonate:

```
>>> from periodictable import formula
>>> print formula("CaCO3")
CaCO3
```

- Formulas can contain multiple groups separated by space or plus or by using parentheses. Whole groups can have a repeat count. The following are equivalent definitions of hydrated calcium carbonate:

```
>>> print formula("CaCO3+6H2O")
CaCO3(H2O)6
>>> print formula("CaCO3 6H2O")
CaCO3(H2O)6
>>> print formula("CaCO3(H2O)6")
CaCO3(H2O)6
```

- Parentheses can nest, e.g., in polyethylene glycol:

```
>>> print formula("HO ((CH2)2O)6 H")
HO((CH2)2O)6H
```

- Isotopes are represented by index, such as O[18] = ¹⁸O:

```
>>> print formula("CaCO[18]3+6H2O")
CaCO[18]3(H2O)6
```

- Counts can be integer or decimal:

```
>>> print formula("CaCO3+(3HO1.5)2")
CaCO3((HO1.5)3)2
```

- Empty formulas are supported, e.g., for air or vacuum:

```
>>> print formula()
<BLANKLINE>
>>> formula()
formula('')
```

The grammar used for parsing formula strings is the following:

```
number    :: [1-9][0-9]*
fraction  :: (number | [0] | nothing) '.' [0-9]*
count     :: number | fraction | nothing
symbol    :: [A-Z][a-z]*
isotope   :: '[' number ']' | nothing
element   :: symbol isotope count
separator :: '+' | nothing
group     :: count element+ | '(' formula ')' count
grammar   :: group space separator space formula | group | nothing
```

Formulas can also be constructed from atoms or other formulas:

- A simple formula can be created from a bare atom:

```
>>> from periodictable import Ca, C, O, H
>>> print formula(Ca)
Ca
```

- More complex structures will require a sequences of counts and fragments. The fragment itself can be a structure:

```
>>> print formula( [ (1,Ca), (1,C), (3,O), (6, [(2,H), (1,O)]) ] )
CaCO3 (H2O) 6
```

- Structures can also be built with simple formula math:

```
>>> print formula("CaCO3") + 6*formula("H2O")
CaCO3 (H2O) 6
```

- Formulas can be easily cloned:

```
>>> print formula( formula("CaCO3+6H2O"))
CaCO3 (H2O) 6
```

1.3.1 Density

Density can be specified directly when the formula is created, or updated within a formula. For isotope specific formulas, the density can be given either as the density of the formula using naturally occurring abundance if the unit cell is approximately the same, or using the density specific to those isotopes used.

This makes heavy water density easily specified as:

```
>>> D2O = formula('D2O', natural_density=1)
>>> print D2O, "%.4g"%D2O.density
D2O 1.112
```

Density can also be estimated from the volume of the unit cell, either by using the covalent radii of the constituent atoms and assuming some packing factor, or by knowing the lattice parameters of the crystal which makes up the material. Standard packing factors for hcp, fcc, bcc, cubic and diamond on uniform spheres can be used if the components are of about the same size. The formula should specify the number of atoms in the unit cell, which is 1 for cubic, 2 for bcc and 4 for fcc. Be sure to use the molecular mass (M.molecular_mass in g) rather than the molar mass (M.mass in u = g/mol) in your calculations.

Because the packing fraction method relies on the covalent radius estimate it is not very accurate:

```
>>> from periodictable import elements, formula
>>> Fe = formula("2Fe") # bcc lattice has 2 atoms per unit cell
>>> Fe.density = Fe.molecular_mass/Fe.volume('bcc')
>>> print "%.3g"%Fe.density
6.55
>>> print "%.3g"%elements.Fe.density
7.87
```

Using lattice parameters the results are much better:

```
>>> Fe.density = Fe.molecular_mass/Fe.volume(a=2.8664)
>>> print "%.3g"%Fe.density
7.88
>>> print "%.3g"%elements.Fe.density
7.87
```

1.3.2 Mixtures

Mixtures can be created by weight or volume ratios, with the density of the result computed from the density of the materials. For example, the following is a 2:1 mixture of water and heavy water:

```
>>> from periodictable import formula, mix_by_volume, mix_by_weight
>>> H2O = formula('H2O', natural_density=1)
>>> D2O = formula('D2O', natural_density=1)
>>> mix = mix_by_volume(H2O, 2, D2O, 1)
>>> print mix, "%.4g"%mix.density
(H2O)2D2O 1.037
```

Note that this is different from a 2:1 mixture by weight:

```
>>> mix = mix_by_weight(H2O, 2, D2O, 1)
>>> print mix, "%.4g"%mix.density
(H2O)2.2234D2O 1.035
```

1.3.3 Derived values

Once a formula has been created, it can be used for summary calculations. The following is an example of hydrated quartz, which shows how to compute molar mass and neutron/xray scattering length density:

```
>>> import periodictable
>>> SiO2 = periodictable.formula('SiO2')
>>> hydrated = SiO2 + periodictable.formula('3H2O')
>>> print hydrated, 'mass', hydrated.mass
SiO2(H2O)3 mass 114.13014
>>> rho, mu, inc = periodictable.neutron_sld('SiO2+3H2O', density=1.5, wavelength=4.75)
>>> print hydrated, 'neutron sld', "%.3g"%rho
SiO2(H2O)3 neutron sld 0.849
>>> rho, mu = periodictable.xray_sld(hydrated, density=1.5,
... wavelength=periodictable.Cu.K_alpha)
>>> print hydrated, 'X-ray sld', "%.3g"%rho
SiO2(H2O)3 X-ray sld 13.5
```

1.4 Bundling with py2exe

When using periodictable as part of a bundled package, you need to be sure to include the data associated with the tables. This can be done by adding a periodictable entry into the *package_data* property of the distutils setup file:

```
import periodictable
...
setup(..., package_data=periodictable.package_data, ...)
```

If you have a number of packages which add package data (for example, periodic table extensions), then you can use the following:

```
import periodictable

package_data = {}
...
package_data.update(periodictable.package_data)
...
setup(..., package_data=package_data, ...)
```

1.5 Adding properties

The periodic table is extensible. Third party packages can add attributes to the table, and they will appear in all of the elements.

In order to add a new property to the table, you need to define a python package which contains the required information, and can attach the information to the periodic table so that it is available on demand. This is done with the function `init(table)` in your table extension.

This example adds the attribute `discoverer` to each element. First create the file `discoverer/core.py`:

```
"""
Partial table of element discoverers.

From http://en.wikipedia.org/wiki/Discoveries\_of\_the\_chemical\_elements.
"""

import periodictable.core

def init(table, reload=False):
    if 'discoverer' in table.properties and not reload: return
    table.properties.append('discoverer')

    # Set the default, if any
    periodictable.core.Element.discoverer = "Unknown"

    # Not numeric, so no discoverer_units

    # Load the data
    for name, person in data.iteritems():
        el = table.name(name)
        el.discoverer = person

data = dict(
    arsenic="Jabir ibn Hayyan",
    antimony="Jabir ibn Hayyan",
    bismuth="Jabir ibn Hayyan",
    phosphorus="H. Brand",
    cobalt="G. Brandt",
    platinum="A. de Ulloa",
    nickel="A.F. Cronstedt",
    magnesium="J. Black",
)
```

Now that we have defined the `init(table)` function, we need a way to call it. The simplest solution is to load it directly when your package is imported. In the current example, this could be done by adding the following line to the end of the file:

```
init(periodictable.core.elements)
```

This would be fine for the current example because the table size is small and load time is fast. For large tables, you may wish to delay loading the table data until it is needed. To do this, we use the `delayed_load` function in our package init file `discoverer/__init__.py`:

```
import periodictable.core

# Delayed loading of the element discoverer information
def _load_discoverer():
    """
```

```
The name of the person or group who discovered the element.
"""
from . import core
core.init(periodictable.core.default_table())
periodictable.core.delayed_load(['discoverer'], _load_discoverer)
```

The first argument to `delayed_load` is the list of all attributes that will be defined when the module is loaded. The second argument is the loading function, whose docstring will appear as the attribute description for each attribute in the first list.

Check that it works:

```
>>> import discoverer
>>> import periodictable
>>> print periodictable.magnesium.discoverer
J. Black
```

Isotope and ion specific data is also supported. In this case we need a data table that contains information for each isotope of each element. The following example uses a dictionary of elements, with a dictionary of isotopes for each. It adds the `shells` attribute to Fe[56] and Fe[58].

Define `shelltable/core.py`:

```
"""
Example of isotope specific extensions to the periodic table.
"""
from periodictable.core import Isotope

def init(table, reload=False):
    if 'shells' in table.properties and not reload: return
    table.properties.append('shells')

    # Set the default. This is required, even if it is only
    # setting it to None. If the attribute is missing then
    # the isotope data reverts to the element to supply the
    # value, which is almost certainly not what you want.
    Isotope.shells = None

    # Load the data
    for symbol, eldata in data.iteritems():
        el = table.symbol(symbol)
        for iso, isodata in eldata.iteritems():
            el[iso].shells = isodata

    # Define the data
    data = dict(
        Fe = {56: "56-Fe shell info",
              58: "58-Fe shell info",
              },
    )
```

Again, we are going to initialize the table with delayed loading. In this case it is very important that we set the `isotope=True` keyword in the `delayed_load` call. If we don't, then the magic we use to return the correct value after loading the new table information fails. Since unknown attributes are delegated to the underlying element, the value for the natural abundance will be returned instead. On subsequent calls the isotope specific value will be returned.

This is demonstrated in `shelltable/__init__.py`:

```
import periodictable.core

# Delayed loading of the element discoverer information
def _load():
    """
    The name of the person or group who discovered the element.
    """
    from . import core
    core.init(periodictable.core.default_table())
periodictable.core.delayed_load(['shells'], _load,
                               isotope=True, element=False)
```

Check that it works:

```
>>> import shelltable
>>> import periodictable
>>> print periodictable.Fe[56].shells
56-Fe shell info
>>> print periodictable.Ni[58].shells
None
```

Ion specific data is more complicated, particularly because of the interaction with isotopes. For example, `Ni[58].ion[3]` should have the same mass as `Ni[58]` (the mass of the electron is negligible), but a different mass from `Ni.ion[3]`. However, the `f0` scattering factors for X-rays are dependent on the ionization state, so `Ni[58].ion[3].xray.f0(Q)` and `Ni.ion[3].xray.f0(Q)` are the same but different from `Ni.xray.f0(Q)`.

Current support for ion dependent properties is awkward. The X-ray table `periodictable.xsf` creates a specialized structure for each ion as it is requested. The magnetic form factor table `periodictable.magnetic_ff` does not try to support `ion.magnetic_ff` directly, but instead the user must request `ion.magnetic_ff[ion.charge]`. Properties dependent on both isotope and ion can probably be implemented, but there are no examples yet.

Be sure to use the `ion=True` keyword for `delayed_load` when the table extension contains ion specific information.

1.6 Custom tables

You can create your own private instance of `PeriodicTable` and populate it with values.

Example:

```
>>> import periodictable
>>> from periodictable import core, mass, density, elements, formula

>>> mytable = core.PeriodicTable(table="H=1")
>>> mass.init(mytable)
>>> density.init(mytable)

>>> # Reset mass to H=1 rather than C=12
>>> scale = elements.H[1].mass
>>> for el in mytable:
...     el._mass /= scale
...     if hasattr(el, '_density') and el._density != None:
...         el._density /= scale
...     for iso in el:
...         iso._mass /= scale
```

```
>>> print mytable.H[1].mass, mytable.C[12].mass
1.0 11.9068286833
>>> print periodictable.H[1].mass, periodictable.C[12].mass
1.0078250321 12.0
>>> print formula('2H[1]', table=mytable).mass
2.0
```

You will need to add individual properties by hand for all additional desired properties using `module.init(elements)`.

The table name ($H=I$ above) must be unique within the session. If you are pickling elements from a custom table, you must create a custom table of the same name before attempting to restore them. The default table is just a custom table with the name *public*.

Support for custom tables could be made much smoother by delegating all properties not defined in the custom table back to the base table, much like is currently done for `Isotopes` and `Ions`. That way you only need to replace the properties of interest rather than defining all new properties.

The alternative to using custom tables is to replace a dataset in the base table using e.g., `custom_mass.init(elements, reload = True)`, where `custom_mass` is your own version of the periodic table values. Be aware, however, that this will have a global effect, changing the mass used by all packages, so you are strongly discouraged from doing so.

1.7 Data Sources

Physical constants [NIST Physics Laboratory - Constants, units and uncertainty](#)

Atomic and isotope mass [NIST Physics Laboratory - Atomic weights and isotope composition](#)

Atomic density [ILL Neutron Data Booklet](#)

Covalent Radii [Cordero, et al., Dalton Trans., 2008, 2832-2838, doi:10.1039/801115j](#)

Magnetic form factors [Brown. P. J., In International Tables for Crystallography, Volume C, Wilson. A. J. C \(ed\), section 4.4.5](#)

Neutron scattering factors [Atomic Institute for Austrian Universities](#)

X-ray scattering factors [Center for X-Ray Optics](#)

Crystal structure [Ashcroft and Mermin](#)

1.8 Contributing Changes

The best way to contribute to the periodic table package is to work from a copy of the source tree in the revision control system.

The source is available via git:

```
git clone https://github.com/pkienzle/periodictable.git
cd periodictable
python setup.py develop
```

By using the *develop* keyword on `setup.py`, changes to the files in the package are immediately available without the need to run `setup.py install` each time.

Track updates to the original package using:

```
git pull
```

If you find you need to modify the periodic table package, please update the documentation and add tests for your changes. We use doctests on all of our examples that we know our documentation is correct. More thorough tests are found in test directory. Using the the nose test package, you can run both sets of tests:

```
easy_install nose
python2.5 tests.py
python2.6 tests.py
```

When all the tests run, generate a patch and send it to the DANSE Project mailing list at danse-dev@cacr.caltech.edu:

```
git diff > periodictable.patch
```

Alternatively, create a project fork at github and we can pull your changes directly from your repository.

Windows user can use [TortoiseGit](#) package which provides similar operations.

Building the package documentation requires a working sphinx installation and in addition, a copy of [MathJax](#) to view the equations. Download and unzip the MathJax package into the doc/sphinx directory to install MathJax. You can then build the documentation as follows:

```
(cd doc/sphinx && make clean html pdf)
```

You can see the result by pointing your browser to:

```
periodictable/doc/sphinx/_build/html/index.html
periodictable/doc/sphinx/_build/latex/PeriodicTable.pdf
```

As of this writing, the \AA LaTeX command for the Angstrom symbol is not available in the MathJax distribution. We patched `jax/input/TeX/jax.js` with the additional symbol AA using:

```
// Ord symbols
S:          '00A7',
+ AA:       '212B',
aleph:      ['2135', {mathvariant: MML.VARIANT.NORMAL}],
```

If you are using unusual math characters, you may need similar patches for your own documentation.

ReStructured text format does not have a nice syntax for superscripts and subscripts. Units such as $\text{g}\cdot\text{cm}^{-3}$ are entered using macros such as `|g/cm^3|` to hide the details. The complete list of macros is available in

```
doc/sphinx/rst_prolog
```

In addition to macros for units, we also define `cdot`, `angstrom` and `degrees` unicode characters here. The corresponding latex symbols are defined in `doc/sphinx/conf.py`.

1.9 License

This package is in the public domain. Individual files may hold separate copyright and licensing terms. See the file header for details.

1.10 Disclaimer

This data has been compiled from a variety of sources for the user's convenience and does not represent a critical evaluation by the authors. While we have made efforts to verify that the values we use match published values, the values themselves are based on measurements whose conditions may differ from those of your experiment.

1.11 Credits

Periodictable was written by Paul Kienzle and is developed and maintained by the [DANSE](#) project.

We are grateful for the existence of many fine open source packages such as [Pyparsing](#), [NumPy](#) and [Python](#) without which this package would be much more difficult to write.

REFERENCE

2.1 Core table

2.1.1 `periodictable.core`

Core classes for the periodic table.

- **PeriodicTable** The periodic table with attributes for each element.
 - Note:** `PeriodicTable` is not a singleton class. Use `periodictable.element` to access the common table.
- **Element** Element properties such as name, symbol, mass, density, etc.
- **Isotope** Isotope properties such as mass, density and neutron scattering factors.
- **Ion** Ion properties such as charge.

Elements are accessed from a periodic table using `table[number]`, `table.name` or `table.symbol` where *symbol* is the two letter symbol. Individual isotopes are accessed using `element[isotope]`. Individual ions are references using `element.ion[charge]`. There are presently no properties specific to both ion and isotope.

Helper functions:

- `delayed_load()` Delay loading the element attributes until they are needed.
- `get_data_path()` Return the path to the periodic table data files.
- `define_elements()` Define external variables for each element in namespace.
- `isatom()`, `iselement()`, `isisotope()`, `ision()` Tests for different types of structure components.
- `default_table()` Returns the common periodic table.
- `change_table()` Return the same item from a different table.

See Also:

Adding properties for details on extending the periodic table with your own attributes.

Custom tables for details on managing your own periodic table with custom values for the attributes.

class `periodictable.core.Ion` (*element, charge*)

Bases: `object`

Periodic table entry for an individual ion. An ion is associated with an element. In addition to the element properties (*symbol, name, atomic number*), it has specific ion properties (*charge*). Properties not specific to the ion (i.e., *charge*) are retrieved from the associated element.

xray

X-ray scattering properties for the elements.

Reference: *Center for X-Ray optics. Henke. L., Gullikson. E. M., and Davis. J. C.*

class `periodictable.core.Isotope` (*element, isotope_number*)

Bases: `object`

Periodic table entry for an individual isotope. An isotope is associated with an element. In addition to the element properties (*symbol, name, atomic number*), it has specific isotope properties (*isotope number, nuclear spin, relative abundance*). Properties not specific to the isotope (e.g., *x-ray scattering factors*) are retrieved from the associated element.

abundance

Natural abundance.

Parameters *isotope* : Isotope

Returns *abundance* : float | %

Reference: *Coursey. J. S., Schwab. D. J, and Dragoset. R. A., NIST Atomic Weights and Isotopic Composition Database.*

density

Element density for natural abundance. For isotopes, return the equivalent density assuming identical inter-atomic spacing as the naturally occurring material.

Parameters

iso_el [isotope or element] Name of the element or isotope.

Returns *density* : float | g·cm⁻³

Reference: *ILL Neutron Data Booklet, original values from CRC Handbook of Chemistry and Physics, 80th ed. (1999).*

mass

Atomic weight.

Parameters *isotope* : Isotope

Returns

mass [float | u] Atomic weight of the element.

Reference: *Coursey. J. S., Schwab. D. J, and Dragoset. R. A., NIST Atomic Weights and Isotopic Composition Database.*

neutron

Neutron scattering factors, *nuclear_spin* and *abundance* properties for elements and isotopes.

Reference: *Rauch. H. and Waschkowski. W., ILL Neutron Data Booklet.*

class `periodictable.core.Element` (*name, symbol, Z, ions, table*)

Bases: `object`

Periodic table entry for an element. An element is a name, symbol and number, plus a set of properties. Individual isotopes can be referenced as `element[isotope_number]`. Individual ionization states can be referenced by `element.ion[charge]`.

add_isotope (*number*)

Add an isotope for the element.

Parameters

number [integer] Isotope number, which is the number protons plus neutrons.

Returns None

K_alpha

X-ray emission lines for various elements, including Ag, Pd, Rh, Mo, Zn, Cu, Ni, Co, Fe, Mn, Cr and Ti.
K_alpha is the average of *K_alpha1* and *K_alpha2* lines.

K_alpha_units

X-ray emission lines for various elements, including Ag, Pd, Rh, Mo, Zn, Cu, Ni, Co, Fe, Mn, Cr and Ti.
K_alpha is the average of *K_alpha1* and *K_alpha2* lines.

K_beta1

X-ray emission lines for various elements, including Ag, Pd, Rh, Mo, Zn, Cu, Ni, Co, Fe, Mn, Cr and Ti.
K_alpha is the average of *K_alpha1* and *K_alpha2* lines.

K_beta1_units

X-ray emission lines for various elements, including Ag, Pd, Rh, Mo, Zn, Cu, Ni, Co, Fe, Mn, Cr and Ti.
K_alpha is the average of *K_alpha1* and *K_alpha2* lines.

covalent_radius

covalent radius: average atomic radius when bonded to C, N or O.

covalent_radius_uncertainty

covalent radius: average atomic radius when bonded to C, N or O.

covalent_radius_units

covalent radius: average atomic radius when bonded to C, N or O.

crystal_structure

Add *crystal_structure* property to the elements.

Reference: *Ashcroft and Mermin.*

density

Element density for natural abundance. For isotopes, return the equivalent density assuming identical inter-atomic spacing as the naturally occurring material.

Parameters

iso_el [isotope or element] Name of the element or isotope.

Returns *density* : float | g·cm⁻³

Reference: *ILL Neutron Data Booklet, original values from CRC Handbook of Chemistry and Physics, 80th ed. (1999).*

interatomic_distance

Estimated interatomic distance from atomic weight and density. The distance between isotopes is assumed to match that between atoms in the natural abundance.

Parameters

element [Element] The element whose interatomic distance is to be calculated.

Returns

distance [float | Å] Estimated interatomic distance.

Interatomic distance is computed using:

$$d = (m / (\rho_m N_A 10^{-24}))^{1/3}$$

with units:

$$((\text{g} \cdot \text{mol}^{-1})/((\text{g} \cdot \text{cm}^{-3})(\text{atoms} \cdot \text{mol}^{-1})(10^{-8}\text{cm} \cdot \text{\AA}^{-1})^3))^{1/3} = \text{\AA}$$

isotopes

List of all isotopes

magnetic_ff

Magnetic Form Factors. These values are directly from CrysFML.

Reference: *Brown. P. J.(Section 4.4.5) International Tables for Crystallography Volume C, Wilson. A.J.C.(ed).*

mass

Atomic weight.

Parameters *isotope* : Isotope

Returns

mass [float | u] Atomic weight of the element.

Reference: *Coursey. J. S., Schwab. D. J, and Dragoset. R. A., NIST Atomic Weights and Isotopic Composition Database.*

neutron

Neutron scattering factors, *nuclear_spin* and *abundance* properties for elements and isotopes.

Reference: *Rauch. H. and Waschkowski. W., ILL Neutron Data Booklet.*

number_density

Estimate the number density from atomic weight and density. The density for isotopes is assumed to match that of between atoms in natural abundance.

Parameters

element [element] Name of the element whose number density needs to be calculated.

Returns

Nb [float | unitless] Number density of a element.

xray

X-ray scattering properties for the elements.

Reference: *Center for X-Ray optics. Henke. L., Gullikson. E. M., and Davis. J. C.*

class `periodictable.core.PeriodicTable` (*table*)

Bases: `object`

Defines the periodic table of the elements with isotopes. Individual elements are accessed by name, symbol or atomic number. Individual isotopes are addressable by `element[mass_number]` or `elements.isotope(element name)`, `elements.isotope(element symbol)`.

For example, the following all retrieve iron:

```
>>> from periodictable import *
>>> print elements[26]
Fe
>>> print elements.Fe
Fe
>>> print elements.symbol('Fe')
Fe
```

```
>>> print elements.name('iron')
Fe
>>> print elements.isotope('Fe')
Fe
```

To get iron-56, use:

```
>>> print elements[26][56]
56-Fe
>>> print elements.Fe[56]
56-Fe
>>> print elements.isotope('56-Fe')
56-Fe
```

Deuterium and tritium are defined as 'D' and 'T'. Some neutron properties are available in `elements[0]`.

To show all the elements in the table, use the iterator:

```
>>> from periodictable import *
>>> for el in elements: # lists the element symbols
...     print el.symbol,el.name
n neutron
H hydrogen
He helium
...
Uuh ununhexium
```

Note: Properties can be added to the elements as needed, including *mass*, *nuclear* and *X-ray* scattering cross sections. See section [Adding properties](#) for details.

isotope (*input*)

Lookup the element or isotope in the periodic table. Elements are assumed to be given by the standard element symbols. Isotopes are given by number-symbol, or 'D' and 'T' for 2-H and 3-H.

Parameters

input [string] Element name or isotope to be looked up in periodictable.

Returns Element

Raises *ValueError* if element or isotope is not defined.

For example, print the element corresponding to '58-Ni'.

```
>>> import periodictable
>>> print periodictable.elements.isotope('58-Ni')
58-Ni
```

list (**props*, ***kw*)

Print a list of elements with the given set of properties.

Parameters

prop1, prop2, ... [string] Name of the properties to print

format: string Template for displaying the element properties, with one % for each property.

Returns None

For example, print a table of mass and density.

```
>>> from periodictable import elements
>>> elements.list('symbol', 'mass', 'density',
...             format="%-2s: %6.2f u %5.2f g/cm^3")
```

```
H : 1.01 u 0.07 g/cm^3
He: 4.00 u 0.12 g/cm^3
Li: 6.94 u 0.53 g/cm^3
...
Bk: 247.00 u 14.00 g/cm^3
```

name (*input*)

Lookup an element given its name.

Parameters

input [string] Element name to be looked up in periodictable.

Returns Element

Raises *ValueError* if element does not exist.

For example, print the element corresponding to ‘iron’:

```
>>> import periodictable
>>> print periodictable.elements.name('iron')
Fe
```

symbol (*input*)

Lookup the an element in the periodic table using its symbol. Symbols are included for ‘D’ and ‘T’, deuterium and tritium.

Parameters

input [string] Element symbol to be looked up in periodictable.

Returns Element

Raises *ValueError* if the element symbol is not defined.

For example, print the element corresponding to ‘Fe’:

```
>>> import periodictable
>>> print periodictable.elements.symbol('Fe')
Fe
```

`periodictable.core.delayed_load` (*all_props, loader, element=True, isotope=False, ion=False*)

Delayed loading of an element property table. When any of property is first accessed the loader will be called to load the associated data. The help string starts out as the help string for the loader function. The attribute may be associated with any of `Isotope`, `Ion`, or `Element`. Some properties, such as `mass`, have both an isotope property for the mass of specific isotopes, as well as an element property for the mass of the collection of isotopes at natural abundance. Set the keyword flags *element*, *isotope* and/or *ion* to specify which of these classes will be assigned specific information on load.

`periodictable.core.define_elements` (*table, namespace*)

Define external variables for each element in namespace. Elements are defined both by name and by symbol.

This is called from `__init__` as:

```
elements = core.default_table()
__all__ += core.define_elements(elements, globals())
```

Parameters

table [PeriodicTable] Set of elements

namespace [dict] Namespace in which to add the symbols.

Returns [string, ...] A sequence listing the names defined.

Note: This will only work for *namespace* globals(), not locals()!

`periodictable.core.get_data_path(data)`

Locate the directory for the tables for the named extension.

Parameters

data [string] Name of the extension data directory. For example, the xsf extension has data in the 'xsf' data directory.

Returns string Path to the data.

`periodictable.core.default_table(table=None)`

Return the default table unless a specific table has been requested.

This is to be used in a context like:

```
def summary(table=None):
    table = core.default_table(table)
    ...
```

`periodictable.core.change_table(atom, table)`

Search for the same element, isotope or ion from a different table

`periodictable.core.isatom(val)`

Return true if value is an element, isotope or ion

`periodictable.core.iselement(val)`

Return true if value is an element or ion in natural abundance

`periodictable.core.isisotope(val)`

Return true if value is an isotope or isotope ion.

`periodictable.core.ision(val)`

Return true if value is a specific ion of an element or isotope

2.2 Chemical formula

2.2.1 `periodictable.formulas`

Chemical formula parser.

`class periodictable.formulas.Formula` (*structure=()*, *density=None*, *natural_density=None*, *name=None*)

Bases: object

Simple chemical formula representation.

`change_table` (*table*)

Replace the table used for the components of the formula.

`natural_mass_ratio` ()

Natural mass to isotope mass ratio.

Returns *ratio* : float

The ratio is computed from the sum of the masses of the individual elements using natural abundance divided by the sum of the masses of the isotopes used in the formula. If the cell volume is preserved with isotope substitution, then the ratio of the masses will be the ratio of the densities.

neutron_sld (*args, **kw)

Neutron scattering information for the molecule.

Parameters

wavelength [float | Å] Wavelength of the neutron beam.

Returns

sld [(float, float, float) | 10^{-6}Å^{-2}] Neutron scattering length density is returned as the tuple (*real*, *imaginary*, *incoherent*), or as (None, None, None) if the mass density is not known.

Deprecated since version 0.95: Use `periodictable.neutron_sld(formula)` instead.

volume (packing_factor='hcp', *args, **kw)

Estimate unit cell volume.

The crystal volume can be estimated from the element covalent radius and the atomic packing factor using:

$$\text{packing_factor} = \text{N_atoms} \text{ V_atom} / \text{V_crystal}$$

Packing factors for a number of crystal lattice structures are defined.

Table 2.1: Crystal lattice names and packing factors

Code	Description	Formula	Packing factor
cubic	simple cubic	$\pi/6$	0.52360
bcc	body-centered cubic	$\pi\sqrt{3}/8$	0.68017
hcp	hexagonal close-packed	$\pi/\sqrt{18}$	0.74048
fcc	face-centered cubic	$\pi/\sqrt{18}$	0.74048
diamond	diamond cubic	$\pi\sqrt{3}/16$	0.34009

Parameters

packing_factor = 'hcp' [float or string] Atomic packing factor. If *packing_factor* is the name of a crystal lattice, use the *lattice* packing factor.

a, b, c [float | Å] Lattice spacings. *b* and *c* default to *a*.

alpha, beta, gamma [float | °] Lattice angles. These default to 90°

Returns

volume [float | cm³] Molecular volume.

Raises *KeyError*: unknown lattice type

TypeError: missing or bad lattice parameters

Using the cell volume, mass density can be set with:

```
formula.density = formula.molecular_mass/formula.volume()
```

xray_sld (*args, **kw)

X-ray scattering length density for the molecule.

Parameters

energy [float | keV] Energy of atom.

wavelength [float | Å] Wavelength of atom.

Returns

sld [(float, float) | 10^{-6}Å^{-2}]

X-ray scattering length density is returned as the tuple (*real, imaginary*), or as (None, None) if the mass density is not known.

Deprecated since version 0.95: Use `periodictable.xray_sld(formula)` instead.

atoms

{ *atom: count, ...* }

Composition of the molecule. Referencing this attribute computes the *count* as the total number of each element or isotope in the chemical formula, summed across all subgroups.

hill

Formula

Convert the formula to a formula in Hill notation. Carbon appears first followed by hydrogen then the remaining elements in alphabetical order.

mass

atomic mass units u (C[12] = 12 u)

Molar mass of the molecule. Use `molecular_mass` to get the mass in grams.

molecular_mass

g

Mass of the molecule in grams.

natural_density

$\text{g}\cdot\text{cm}^{-3}$

Density of the formula with specific isotopes of each element replaced by the naturally occurring abundance of the element without changing the cell volume.

`periodictable.formulas.formula` (*value=None, density=None, natural_density=None, name=None, table=None*)

Construct a chemical formula representation from a string, a dictionary of atoms or another formula.

Parameters

formula [see below] Chemical formula.

density [float | $\text{g}\cdot\text{cm}^{-3}$] Material density. Not needed for single element formulas.

natural_density [float | $\text{g}\cdot\text{cm}^{-3}$] Material density assuming naturally occurring isotopes and no change in cell volume.

name [string] Common name for the molecule.

table [PeriodicTable] Private table to use when parsing string formulas.

Exceptions *ValueError* : invalid formula initializer

After creating a formula, a rough estimate of the density can be computed using:

```
formula.density = formula.molecular_mass/formula.volume(packing_factor=...)
```

The `volume()` calculation uses the covalent radii of the components and the known packing factor or crystal structure name. If the lattice constants for the crystal are known, then they can be used instead:

```
formula.density = formula.molecular_mass/formula.volume(a,b,c,alpha,beta,gamma)
```

Formulas are designed for calculating quantities such as molar mass and scattering length density, not for representing bonds or atom positions. The representations are simple, but preserve some of the structure for display purposes.

`periodictable.formulas.formula_grammar` (*table*)

Construct a parser for molecular formulas.

Parameters

table = **None** [PeriodicTable] If table is specified, then elements and their associated fields will be chosen from that periodic table rather than the default.

Returns

parser [pyparsing.ParserElement.] The `parser.parseString()` method returns a list of pairs (*count,fragment*), where fragment is an *isotope*, an *element* or a list of pairs (*count,fragment*).

`periodictable.formulas.mix_by_volume(*args, **kw)`
 Generate a mixture which apportions each formula by volume.

Parameters

formula1 [Formula OR string] Material
quantity1 [float] Relative quantity of that material
formula2 [Formula OR string] Material
quantity2 [float] Relative quantity of that material
 ...
density [float] Density of the mixture, if known
natural_density [float] Density of the mixture with natural abundances, if known.
name [string] Name of the mixture
table [PeriodicTable] Private table to use when parsing string formulas.

Returns *formula* : Formula

If density is not given, then it will be computed from the density of the components, assuming the components take up no more nor less space because they are in the mixture. If component densities are not available, then the resulting density will not be computed.

`periodictable.formulas.mix_by_weight(*args, **kw)`
 Generate a mixture which apportions each formula by weight.

Parameters

formula1 [Formula OR string] Material
quantity1 [float] Relative quantity of that material
formula2 [Formula OR string] Material
quantity2 [float] Relative quantity of that material
 ...
density [float] Density of the mixture, if known
natural_density [float] Density of the mixture with natural abundances, if known.
name [string] Name of the mixture
table [PeriodicTable] Private table to use when parsing string formulas.

Returns *formula* : Formula

If density is not given, then it will be computed from the density of the components, assuming the components take up no more nor less space because they are in the mixture. If component densities are not available, then the resulting density will not be computed.

`periodictable.formulas.parse_formula` (*str*, *table=None*)

Parse a chemical formula, returning a structure with elements from the given periodic table.

2.3 Covalent radius

2.3.1 `periodictable.covalent_radius`

This module adds the following fields to the periodic table

- `covalent_radius`
- `covalent_radius_uncertainty`
- `covalent_radius_units = 'angstrom'`

Use `init()` to initialize a private table.

Data is taken from Cordero et al., 2008¹. Bond specific values (single, double, or triple) are available from Pyykkö et al., 2009², but they are generally smaller. The CRC Handbook uses the average of Cordero and Pyykkö. Note that the combined Cordero/Pyykkö tables are included herein as `periodictable.covalent_radius.CorderoPyykko`, but are not yet parsed.

The abstract of Cordero reads as follows:

A new set of covalent atomic radii has been deduced from crystallographic data for most of the elements with atomic numbers up to 96. The proposed radii show a well behaved periodic dependence that allows us to interpolate a few radii for elements for which structural data is lacking, notably the noble gases. The proposed set of radii therefore fills most of the gaps and solves some inconsistencies in currently used covalent radii. The transition metal and lanthanide contractions as well as the differences in covalent atomic radii between low spin and high spin configurations in transition metals are illustrated by the proposed radii set.

Note:

1. Values are averages only. The particular radius can be highly dependent on oxidation state and chemical compound.
2. The paper lists values for multiple spin states on select elements. We are using sp³ for carbon and low spin for manganese, iron and cobalt.
3. Elements with zero or one measurements of covalent radius are assigned an uncertainty of 0.00. These are He, Ne, Pm, At, Rn, Fr, Ac, Pa.
4. Elements above 96 are assigned a covalent radius and uncertainty of None.
5. Radii are measured from bonds to C, N or O. The choice of which compound was used is element dependent. Details are available in the references.

`periodictable.covalent_radius.init` (*table*, *reload=False*)

Add the covalent radius property to a private table. Use `reload = True` to replace the covalent radius property on an existing table.

¹ Beatriz Cordero, Verónica Gómez, Ana E. Platero-Prats, Marc Revés, Jorge Echeverría, Eduard Cremades, Flavia Barragán and Santiago Alvarez. Covalent radii revisited. Dalton Trans., 2008, 2832-2838. doi:10.1039/b801115j

² P. Pyykkö and M. Atsumi. Molecular Double-Bond Covalent Radii for Elements Li-E112. Chemistry, A European Journal, 15, 2009, 12770-12779. doi:10.1002/chem.200901472

2.4 Crystal structure

2.4.1 `periodictable.crystal_structure`

Crystal structure data.

Adds *crystal_structure* to the periodic table. Each crystal structure is a dictionary which contains the key 'symmetry'. Depending on the value of `crystal_structure['symmetry']`, one or more parameters 'a', 'c/a', 'b/a', 'd', and 'alpha' may be present according to the following table:

Table 2.2: Crystal lattice parameters

Symmetry	Parameters
atom	
diatom	d
BCC	a
fcc	a
hcp	c/a, a
Tetragonal	c/a, a
Cubic	a
Diamond	a
Orthorhombic	c/a, a, b/a
Rhombohedral	a, alpha
SC	a
Monoclinic	

Example:

```
>>> import periodictable as elements
>>> print elements.C.crystal_structure['symmetry']
Diamond
>>> print elements.C.crystal_structure['a']
3.57
```

This data is from Ashcroft and Mermin.

```
periodictable.crystal_structure.init(table, reload=False)
```

2.5 Density

2.5.1 `periodictable.density`

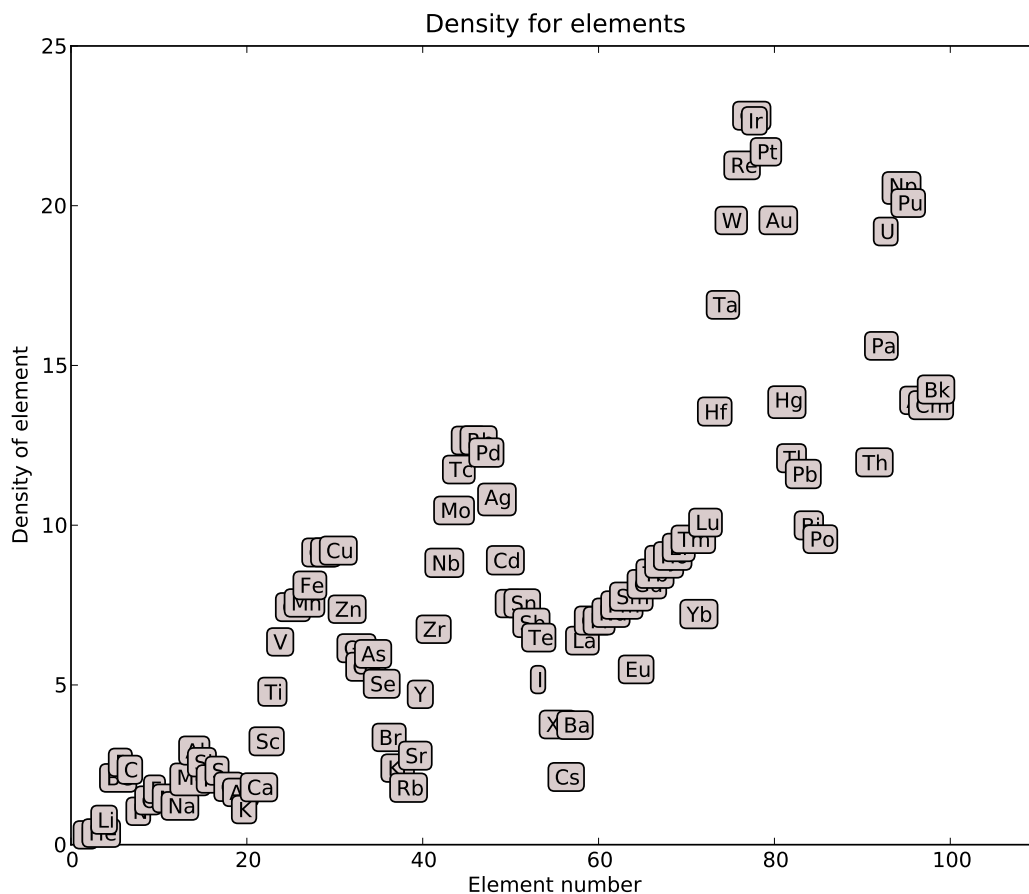
The following properties are added:

- **density, density_units** ($\text{g}\cdot\text{cm}^{-3}$) Densities for solids and liquids are given as specific gravities at 20° C unless other wise indicated by *density_caveat*. Densities for gaseous elements are given for the liquids at their boiling points. Missing data are represented by *None*.
- **density_caveat** Comments on the density, if not taken in standard conditions.
- **interatomic_distance, interatomic_distance_units** (Å) Interatomic distance estimated from element density.
- **number_density, number_density_units** (unitless) Number density estimated from mass and density.

Density for the isotope is computed assuming that the atomic spacing is the same as that for the element in the natural abundance.

```
>>> from periodictable import D, H
>>> print "H :",H.density," , D :",D.density
H : 0.0708 , D : 0.141475093639
>>> print (D.density/H.density) / (D.mass/H.mass)
1.0
```

The following plot shows density for all elements:



From the X-ray data book: http://xdb.lbl.gov/Section5/Sec_5-2.html

Data were taken mostly from ³. These values are reproduced in ⁴.

`periodictable.density.density(iso_el)`

Element density for natural abundance. For isotopes, return the equivalent density assuming identical interatomic spacing as the naturally occurring material.

Parameters

iso_el [isotope or element] Name of the element or isotope.

Returns *density* : float | g·cm⁻³

Reference: *ILL Neutron Data Booklet, original values from CRC Handbook of Chemistry and Physics, 80th*

³ Lide, D. R., Ed., CRC Handbook of Chemistry and Physics, 80th ed. (CRC Press, Boca Raton, Florida, 1999)

⁴ The ILL Neutron Data Booklet, Second Edition.

ed. (1999).

`periodictable.density.init` (*table*, *reload=False*)

`periodictable.density.interatomic_distance` (*element*)

Estimated interatomic distance from atomic weight and density. The distance between isotopes is assumed to match that between atoms in the natural abundance.

Parameters

element [Element] The element whose interatomic distance is to be calculated.

Returns

distance [float | Å] Estimated interatomic distance.

Interatomic distance is computed using:

$$d = (m/(\rho_m N_A 10^{-24}))^{1/3}$$

with units:

$$((g \cdot mol^{-1})/((g \cdot cm^{-3})(atoms \cdot mol^{-1})(10^{-8}cm \cdot \text{Å}^{-1})^3))^{1/3} = \text{Å}$$

`periodictable.density.number_density` (*element*)

Estimate the number density from atomic weight and density. The density for isotopes is assumed to match that of between atoms in natural abundance.

Parameters

element [element] Name of the element whose number density needs to be calculated.

Returns

Nb [float | unitless] Number density of a element.

2.6 Mass

2.6.1 `periodictable.mass`

Adds average mass for the elements:

- **mass, mass_units (u)** The molar mass averaged over natural isotope abundance.

Adds mass and abundance information for isotopes:

- **mass, mass_units (u)** The molar mass of the individual isotope.
- **abundance, abundance_units (%)** Natural abundance for the isotope.

Atomic Weights and Isotopic Composition ⁵.

The atomic weights are available for elements 1 through 112, 114, & 116 and isotopic compositions or abundances are given when appropriate. The atomic weights data were published by Coplen ⁶ in Atomic Weights of the Elements 1999, (and include changes reported from the 2001 review in Chem. Int., 23, 179 (2001)) and the isotopic compositions data

⁵ Coursey, J. S., Schwab, D. J., and Dragoset, R. A., NIST, Physics Laboratory, Office of Electronic Commerce in Scientific and Engineering Data.

⁶ Coplen, T. B. : U.S. Geological Survey, Reston, Virginia, USA.

were published by Rosman ⁷ and Taylor ⁸ in *Isotopic Compositions of the Elements 1997*. The relative atomic masses of the isotopes data were published by Audi ⁹ and Wapstra ¹⁰ in the 1995 Update To The Atomic Mass Evaluation.

This data has been compiled from the above sources for the user's convenience and does not represent a critical evaluation by the NIST Physics Laboratory. <http://physics.nist.gov/PhysRefData/Compositions/>

Neutron mass from NIST Reference on Constants, Units, and Uncertainty <http://physics.nist.gov/cuu/index.html>

`periodictable.mass.abundance(isotope)`

Natural abundance.

Parameters `isotope` : Isotope

Returns `abundance` : float | %

Reference: *Coursey. J. S., Schwab. D. J, and Dragoset. R. A., NIST Atomic Weights and Isotopic Composition Database.*

`periodictable.mass.getval(str)`

`periodictable.mass.init(table, reload=False)`

`periodictable.mass.mass(isotope)`

Atomic weight.

Parameters `isotope` : Isotope

Returns

`mass` [float | u] Atomic weight of the element.

Reference: *Coursey. J. S., Schwab. D. J, and Dragoset. R. A., NIST Atomic Weights and Isotopic Composition Database.*

2.7 Neutron scattering potentials

2.7.1 `periodictable.nsf`

Neutron scattering factors for the elements and isotopes.

For details of neutron scattering factor values, see `Neutron`. The property is set to `None` if there is no neutron scattering information for the element. Individual isotopes may have their own scattering information.

Example

Print a table of coherent scattering length densities for isotopes of a particular element:

```
>>> import periodictable
>>> for iso in periodictable.Ni:
...     if iso.neutron.has_sld():
...         print iso, iso.neutron.sld()[0]
58-Ni 13.152605395
60-Ni 2.55745104902
```

⁷ Rosman. K. J. R. : Department of Applied Physics, Curtin University of Technology, Australia.

⁸ Taylor. P. D. P. : Institute for Reference Materials and Measurements, European Commission, Belgium.

⁹ Audi. G. : Centre de Spectrométrie Nucléaire et de Spectrométrie de Masse, Orsay Campus, France.

¹⁰ Wapstra. A. H. : National Institute of Nuclear Physics and High-Energy Physics, Amsterdam, The Netherlands.

```
61-Ni 6.94165284735
62-Ni -7.94636575947
64-Ni -0.337948888621
```

Details

There are a number of functions available in `periodictable.nsf`

- `neutron_energy()` Return neutron energy given wavelength.
- `neutron_wavelength()` Return wavelength given neutron energy.
- `neutron_wavelength_from_velocity()` Return wavelength given neutron velocity.
- `neutron_scattering()` Computes scattering length density, cross sections and penetration depth for a compound.
- `neutron_sld()` Computes scattering length density for a compound.
- `neutron_composite_sld()` Returns a scattering length density for a compound whose composition is variable.
- `energy_dependent_table()` Lists isotopes with energy dependence.
- `sld_table()` Lists scattering length densities for all elements in natural abundance.
- `absorption_comparison_table()` Compares the imaginary bound coherent scattering length to the absorption cross section.
- `coherent_comparison_table()` Compares the bound coherent scattering length to the coherent scattering cross section.
- `total_comparison_table()` Compares the total scattering cross section to the sum of the coherent and incoherent scattering cross sections.

For private tables use `init()` to set the data.

The neutron scattering information table is reproduced from the Atomic Institute for Austrian Universities (2007 version):

<http://www.ati.ac.at/~neutropt/scattering/table.html>

The above site has references to the published values for every entry in the table. We have included these in the documentation directory associated with the `periodictable` package.

```
class periodictable.nsf.Neutron
    Bases: object
```

Neutron scattering factors are attached to each element in the periodic table for which values are available. If no information is available, then the neutron field of the element will be *None*. Even when neutron information is available, it may not be complete, so individual fields may be *None*.

The following fields are used:

- **b_c (fm)** Bounds coherent scattering length.
- **b_c_i (fm)** Imaginary bound coherent scattering length. This is related to absorption cross section by $\sigma_a = 4\pi b_i/k$ where $k = 2\pi/\lambda$ and an additional factor of 1000 for converting between Å·fm and barns. `b_c_i` is not available for all isotopes for which absorption cross sections have been measured.

- bp,bm (fm)** Spin-dependent scattering for I+1/2 and I-1/2 (not always available). Incoherent scattering arises from the spin-dependent scattering b+ and b-. The Neutron Data Booklet¹¹ gives formulas for calculating coherent and incoherent scattering from b+ and b- alone.
- bp_i,bm_i (fm)** Imaginary portion of bp and bm.
- is_energy_dependent (boolean)** Do not use this data if scattering is energy dependent.
- coherent (barn)** Coherent scattering cross section. In theory coherent scattering is related to bound coherent scattering by $4\pi b_c^2/100$. In practice, these values are different, with the following table showing the largest relative difference:

Sc 3%	Ti 4%	V 34%	Mn 1%	Cd 4%
Te 4%	Xe 9%	Sm 100%	Eu 46%	Gd 61%
Tb 1%	Ho 11%	W 4%	Au 7%	Hg 2%

- incoherent (barn)** Incoherent scattering cross section.
- total (barn)** Total scattering cross section. This is just coherent+incoherent.
- absorption (barn)** Absorption cross section at 1.798 Å. Scale to your beam by dividing by `periodictable.nsf.ABSORPTION_WAVELENGTH` and multiplying by your wavelength.

For elements, the scattering cross-sections are based on the natural abundance of the individual isotopes. Individual isotopes may have the following additional fields

- abundance (%)** Isotope abundance used to compute the properties of the element in natural abundance.
- nuclear_spin (string)** Spin on the nucleus: '0', '1/2', '3/2', etc.

Each field above has a corresponding `*_units` attribute with the name of the units. For scattering calculations, the scattering length density is the value of interest. This is computed from the `number_density` of the individual elements, as derived from the element density and atomic mass.

Note: 1 barn = 100 fm²

has_sld()

Returns *True* if sld is defined for this element/isotope.

scattering (*args, **kw)

Returns neutron scattering information for the element at natural abundance and density.

Warning: Incoherent SLD values have not been verified.

Parameters *wavelength*: float | Å

Returns

sld [(float, float, float) | 10⁻⁶ Å⁻²] (*real*, *imaginary*, *incoherent*) scattering length density

xs [(float, float, float) | cm⁻¹] (*coherent*, *absorption*, *incoherent*) cross sections.

penetration [float | cm] 1/e penetration length.

See `neutron_scattering()` for details.

sld (*args, **kw)

Returns scattering length density for the element at natural abundance and density.

Warning: Incoherent SLD values have not been verified.

¹¹ Rauch, H. and Waschkowski, W. (2003) Neutron Scattering Lengths in ILL Neutron Data Booklet (second edition), A.-J. Dianoux, G. Lander, Eds. Old City Publishing, Philadelphia, PA. pp 1.1-1 to 1.1-17.

Parameters *wavelength* : float | Å

Returns

sld [(float, float, float) | 10⁻⁶Å⁻²] (*real, imaginary, incoherent*) scattering length density.

See `neutron_scattering()` for details.

`periodictable.nsf.init` (*table, reload=False*)

Loads the Rauch table from the neutron data book.

`periodictable.nsf.neutron_energy` (*wavelength*)

Convert neutron wavelength to energy.

Parameters *wavelength* : float or vector | Å

Returns *energy* : float or vector | meV

Wavelength is converted to energy using

$$E = 1/2m_nv^2 = h^2/(2m_n\lambda^2)$$

where:

h = planck's constant in J·s

m_n = neutron mass in kg

`periodictable.nsf.neutron_wavelength` (*energy*)

Convert neutron energy to wavelength.

Parameters *energy* : float or vector | meV

Returns *wavelength* : float or vector | Å

Energy is converted to wavelength using

$$E = 1/2m_nv^2 = h^2/(2m_n\lambda^2) \Rightarrow \lambda = \sqrt{h^2/(2m_nE)}$$

where

h = planck's constant in J·s

m_n = neutron mass in kg

`periodictable.nsf.neutron_wavelength_from_velocity` (*velocity*)

Convert neutron velocity to wavelength.

Parameters *velocity* : float or vector | m/s

Returns *wavelength* : float or vector | Å

Velocity is converted to wavelength using

$$\lambda = h/p = h/(m_nv)$$

where

h = planck's constant in J·s

m_n = neutron mass in kg

`periodictable.nsf.neutron_scattering` (**args, **kw*)

Computes neutron scattering cross sections for molecules.

Warning: Incoherent SLD values have not been verified.

Parameters

compound [Formula initializer] Chemical formula

density [float | g·cm⁻³] Mass density

natural_density [float | g·cm⁻³] Mass density of formula with naturally occurring abundances

wavelength 1.798 [float | Å] Neutron wavelength.

energy [float | meV] Neutron energy. If energy is specified then wavelength is ignored.

Returns

sld [(float, float, float) | 10⁻⁶Å⁻²] (*real, imaginary, incoherent*) scattering length density.

xs [(float, float, float) | cm⁻¹] (*coherent, absorption, incoherent*) cross sections.

penetration [float | cm] 1/e penetration length of the beam

Raises *AssertionError* : density is missing.

The coherent and incoherent cross sections are calculated from the bound scattering lengths for nuclei. The actual cross sections depend on the incoming neutron energy and sample temperature, especially for light elements. For low energy neutrons (cold neutrons), the tabulated cross sections are generally a lower limit. The measured incoherent scattering from hydrogen, for example, can be considerably larger (by more than 20%) than its bound value. For example, the incoherent scattering cross section of H₂O is 5.621/cm as computed from these tables compared to ~7.0/cm as measured with 5 meV neutrons at 290K.¹²

The scattering factor tables are not self consistent. The following functions show discrepancies between the various measurements of the scattering potential:

```
absorption_comparison_table()
```

```
coherent_comparison_table()
```

```
total_comparison_table()
```

To compute the neutron cross sections we first need to average quantities for the unit cell of the molecule.

Molar mass m (g/mol) is the sum of the masses of each component:

$$m = \sum n_i m_i \text{ for each atom } i = 1, 2, \dots$$

Cell volume V (Å³/molecule) is molar mass m over density ρ , with a correction based on Avogadro's number N_A (atoms/mol) and the length conversion 10⁸ Å/cm:

$$V = m/\rho \cdot 1/N_A \cdot (10^8)^3$$

Number density N is the number of scatterers per unit volume:

$$N = \sum n_i / V$$

Coherent scattering cross section σ_c of the molecule is computed from the average scattering length of its constituent atoms, weighted by their frequency.

$$b_c = \sum n_i \text{Im}(b_c) / \sum n_i$$

This is converted to a scattering cross section and scaled by 1 barn = 100 fm²:

$$\sigma_c = 4\pi b_c^2 / 100$$

¹² May, R.P., Ibel, K. and Haas, J. (1982) The forward scattering of cold neutrons by mixtures of light and heavy water. J. Appl. Cryst. 15, 15-19.

Similarly, the absorption cross section σ_a and the total scattering cross section σ_s can be computed from the corresponding cross sections of the constituent elements, already expressed in barns:

$$\sigma_a = \sum n_i \sigma_{ai} / \sum n_i$$

$$\sigma_s = \sum n_i \sigma_{si} / \sum n_i$$

The incoherent cross section is computed from the total scattering cross section and the coherent scattering cross section:

$$\sigma_i = \sigma_s - \sigma_c$$

Because the cross section tables are somewhat inconsistent, the total scattering cross section for the constituent elements σ_{si} is forced to be at least as large as the computed coherent scattering cross section σ_{ci} for that element. This guarantees that the computed incoherent scattering cross section σ_i is never negative.

The absorption cross sections are tabulated at wavelength 1.798 Å. In the thermal neutron energy range the absorption cross section is assumed to scale linearly with wavelength,¹³ and can be adjusted with a simple multiplication:

$$\sigma_a = \sigma_a \lambda / \lambda_o = \sigma_a \lambda / 1.798$$

For the scattering equations, the primary quantity of interest is the scattering potential $b = b' + ib''$. For most elements, the scattering potential at cold neutron and thermal neutron energies is simply related to the neutron energy, with no change in the real portion and a linear scaling of the imaginary portion with energy. The value of b is dominated by the bound coherent potential, with a small contribution from the incoherent scattering cross sections.

The potentials are related to the scattering cross sections as follows:¹⁴

$$\sigma_c = 4\pi |b_c|^2$$

$$\sigma_a = 4\pi b'' / k \text{ for } k = 2\pi / \lambda$$

$$\sigma_i = 4\pi |b_i|^2$$

Transforming these we get:

$$b' = b_c$$

$$b'' = \sigma_a / (2\lambda)$$

$$b_{inc} = \sqrt{\sigma_i / (4\pi)}$$

The incoherent potential b_{inc} can be treated primarily as an absorption potential in large scale structure calculations, with the complex potential b approximated by $b' + i(b'' + b_{inc})$.

The scattering potential is usually expressed as a scattering length density for calculation purposes. This is just the number density of the scatterers times their scattering potential:

$$\rho_{re} = N b_c$$

$$\rho_{im} = N \sigma_a / (2\lambda)$$

$$\rho_{inc} = N \sqrt{\sigma_i / 4\pi}$$

Scattering cross section:

$$\Sigma_{coh} = N \sigma_c$$

$$\Sigma_{inc} = N \sigma_i$$

$$\Sigma_{abs} = N \sigma_a$$

¹³ Lynn, J.E. and Seeger, P.A. (1990) Resonance effects in neutron scattering lengths of rare-earth nuclides. Atomic Data and Nuclear Data Tables 44, 191-207.

¹⁴ Sears, V. F. (1999) 4.4.4 Scattering lengths for neutrons. In Wilson & Prince eds. Intl. Tables for Crystallography C Kluwer Academic Publishers. pp 448-449.

1/e penetration depth d :

$$d = 1/(\Sigma_{\text{coh}} + \Sigma_{\text{inc}} + \Sigma_{\text{abs}})$$

Including unit conversion with $\mu = 10^{-6}$ the full equations are:

$$\begin{aligned} \rho_{\text{re}} (\mu/\text{\AA}^2) &= (N/\text{\AA}^3) (b_c \text{ fm}) (10^{-5} \text{\AA}/\text{fm}) (10^6 \mu) \\ \rho_{\text{im}} (\mu/\text{\AA}^2) &= (N/\text{\AA}^3) (\sigma_a \text{ barn}) (10^{-8} \text{\AA}^2/\text{barn}) / (2\lambda \text{\AA}) (10^6 \mu) \\ \rho_{\text{inc}} (\mu/\text{\AA}^2) &= (N/\text{\AA}^3) \sqrt{(\sigma_i \text{ barn}) / (4\pi) (100 \text{ fm}^2/\text{barn})} (10^{-5} \text{\AA}/\text{fm}) (10^6 \mu) \\ \Sigma_{\text{coh}} (1/\text{cm}) &= (N/\text{\AA}^3) (\sigma_c \text{ barn}) (10^{-8} \text{\AA}^2/\text{barn}) (10^8 \text{\AA}/\text{cm}) \\ \Sigma_{\text{inc}} (1/\text{cm}) &= (N/\text{\AA}^3) (\sigma_i \text{ barn}) (10^{-8} \text{\AA}^2/\text{barn}) (10^8 \text{\AA}/\text{cm}) \\ \Sigma_{\text{abs}} (1/\text{cm}) &= (N/\text{\AA}^3) (\sigma_a \text{ barn}) (10^{-8} \text{\AA}^2/\text{barn}) (10^8 \text{\AA}/\text{cm}) \\ d (\text{cm}) &= 1/(\Sigma_{\text{coh}} 1/\text{cm} + \Sigma_{\text{inc}} 1/\text{cm} + \Sigma_{\text{abs}} 1/\text{cm}) \end{aligned}$$

`periodictable.nsf.neutron_sld(*args, **kw)`

Computes neutron scattering length densities for molecules.

Warning: Incoherent SLD values have not been verified.

Parameters

compound [Formula initializer] Chemical formula

density [float | g·cm⁻³] Mass density

natural_density [float | g·cm⁻³] Mass density of formula with naturally occurring abundances

wavelength [float | \AA] Neutron wavelength.

energy [float | meV] Neutron energy. If energy is specified then wavelength is ignored.

Returns

sld [(float, float, float) | 10⁻⁶\AA⁻²] (*real, imaginary, incoherent*) scattering length density.

Raises *AssertionError* : density is missing.

Returns the scattering length density of the compound. See `neutron_scattering()` for details.

`periodictable.nsf.neutron_composite_sld(materials, wavelength=1.798)`

Create a composite SLD calculator.

Parameters

materials [[Formula]] List of materials

wavelength = 1.798: float OR [float] | \AA Probe wavelength(s).

Returns *calculator* : f(w) -> (sld_re, sld_im, sld_inc)

The composite calculator takes a vector of weights and returns the scattering length density of the composite. This is useful for operations on large molecules, such as calculating a set of contrasts or fitting a material composition.

Table lookups and partial sums and constants are precomputed so that the calculation consists of a few simple array operations regardless of the size of the material fragments.

`periodictable.nsf.sld_plot(table=None)`

Plots SLD as a function of element number.

Parameters

table [PeriodicTable] The default periodictable unless a specific table has been requested.

Returns None

periodictable.nsf.**absorption_comparison_table** (*table=None, tol=None*)

Prints a table comparing absorption to the imaginary bound coherent scattering length *b_c_i*. This is used to checking the integrity of the data and formula.

The relationship between absorption and *b_c_i* is:

$$\sigma_a = -2\lambda b_i \cdot 1000$$

The wavelength $\lambda = 1.798\text{\AA}$ is the neutron wavelength at which the absorption is tallied. The factor of 1000 transforms from $\text{\AA}\cdot\text{fm}$ to barn.

Parameters

table [PeriodicTable] The default periodictable unless a specific table has been requested.

tol = 0.01 [float | barn] Show differences greater than this amount.

Returns None

Example

```
>>> absorption_comparison_table (tol=0.5)
Comparison of absorption and (-2000 lambda b_c_i)
 3-He  5333.00  5322.08  0.2%
      Li   70.50   ----
 6-Li  940.00   934.96  0.5%
      B  767.00   755.16  1.6%
10-B  3835.00   ----
      N   1.90   ----
...

```

periodictable.nsf.**coherent_comparison_table** (*table=None, tol=None*)

Prints a table of $4\pi b_c^2/100$ and coherent for each isotope. This is useful for checking the integrity of the data and formula.

The table only prints where *b_c* exists.

Parameters

table [PeriodicTable] The default periodictable unless a specific table has been requested.

tol = 0.01 [float | barn] Amount of difference to show

Returns None

Example

```
>>> coherent_comparison_table (tol=0.5)
Comparison of (4 pi b_c^2/100) and coherent
  n   172.03  43.01  300.0%
 1-n  172.03  43.01  300.0%
  Sc  18.40   19.00  -3.2%
45-Sc 18.40   19.00  -3.2%
 65-Cu 13.08  14.10  -7.2%
 70-Zn  5.98   4.50  33.0%
 84-Sr  3.14   6.00 -47.6%
...

```

`periodictable.nsf.incoherent_comparison_table (table=None, tol=None)`

Prints a table of incoherent computed from total and b_c with incoherent.

Parameters

table [PeriodicTable] The default periodictable unless a specific table has been requested.

tol = 0.01 [float | barn] Amount of difference to show

Returns None

Example

```
>>> incoherent_comparison_table (tol=0.5)
Comparison of incoherent and (total - 4 pi b_c^2/100)
   Sc      4.50      5.10 -11.8%
 45-Sc     4.50      5.10 -11.8%
 65-Cu     0.40      1.42 -71.7%
 70-Zn     0.00     -1.48 -100.0%
 84-Sr     0.00      2.86 -100.0%
113-Cd     0.30      4.36 -93.1%
...

```

`periodictable.nsf.total_comparison_table (table=None, tol=None)`

Prints a table of neutron.total and sum coh,inc for each isotope where these exist. This is used to checking the integrity of the data and formula.

Parameters

table [PeriodicTable] The default periodictable unless a specific table has been requested.

tol = 0.01 [float | barn] Amount of difference to show

Returns None

Example

```
>>> total_comparison_table (tol=0.1)
Comparison of total cross section to (coherent + incoherent)
   n      43.01      ----
  1-n     43.01      ----
 84-Kr     6.60      ----
149-Sm    200.00    200.50 -0.2%
   Eu     9.20     9.07  1.4%
   Gd    180.00    180.30 -0.2%
155-Gd     66.00     65.80  0.3%
161-Dy     16.00     16.30 -1.8%
180-Ta     7.00     6.70  4.5%
187-Os    13.00    13.30 -2.3%

```

`periodictable.nsf.energy_dependent_table (table=None)`

Prints a table of energy dependent isotopes.

Parameters

table [PeriodicTable] If *table* is not specified, use the common periodic table.

Returns None

Example

```
>>> energy_dependent_table ()
Elements and isotopes with energy dependent absorption:
He-3

```

```
Cd Cd-113
Sm Sm-149
Eu Eu-151
Gd Gd-155 Gd-157
Yb-168
Hg-196 Hg-199
```

`periodictable.nsf.sld_table(wavelength=1, table=None, isotopes=True)`
 Scattering length density table for wavelength 4.75 Å.

Parameters

table [PeriodicTable] If *table* is not specified, use the common periodic table.
isotopes = True [boolean] Whether to consider isotopes or not.

Returns None

Example

```
>>> sld_table(wavelength=4.75)
Neutron scattering length density table
atom      mass density      sld      imag      incoh
H          1.008      0.071    -1.582    0.000    10.691
1-H        1.008      0.071    -1.583    0.000    10.691
D          2.014      0.141    2.823     0.000    1.705
T          3.016      0.212    2.027     0.000    0.453
He         4.003      0.122    0.598     0.000    0.035
3-He       3.016      0.092    1.054     0.272    0.706 *
4-He       4.003      0.122    0.598     0.000    0.035
...
248-Cm     248.072    13.569    2.536     0.000    0.207
* Energy dependent cross sections
```

`periodictable.nsf.neutron_sld_from_atoms(*args, **kw)`
 Deprecated since version 0.91: `neutron_sld()` now accepts dictionaries of {atom: count} directly.

2.8 X-ray scattering potentials

2.8.1 periodictable.xsf

This module has one class and nine functions.

Xray X-ray scattering properties for the elements.

The following attributes are added to each element:

Xray.sftable() Three column table of energy vs. scattering factors f1, f2.

Xray.scattering_factors() Returns f1, f2, the X-ray scattering factors for the given wavelengths interpolated from sftable.

Xray.f0() Returns f0 for the given vector Q, with Q[i] in $[0, 24\pi]$ Å⁻¹.

Xray.sld() Returns scattering length density (*real, imaginary*) for the given wavelengths or energies.

The following functions are available for X-ray scattering information processing:

xray_wavelength() Finds X-ray wavelength in angstroms given energy in keV.

xray_energy() Finds X-ray energy in keV given wavelength in angstroms.

`init()` Initializes a periodic table with the Lawrence Berkeley Laboratory Center for X-Ray Optics xray scattering factors.

`init_spectral_lines()` Sets the K_alpha and K_beta1 wavelengths for select elements.

`sld_table()` Prints the xray SLD table for the given wavelength.

`xray_sld()` Computes xray scattering length densities for molecules.

`xray_sld_from_atoms()` The underlying scattering length density calculator. This works with a dictionary of atoms and quantities directly.

`emission_table()` Prints a table of emission lines.

K_alpha, K_beta1 (Å): X-ray emission lines for various elements, including Ag, Pd, Rh, Mo, Zn, Cu, Ni, Co, Fe, Mn, Cr and Ti. K_alpha is the average of K_alpha1 and K_alpha2 lines.

X-ray scattering factors: Low-Energy X-ray Interaction Coefficients: Photoabsorption, scattering and reflection for E in 30 to 30,000 eV, and Z in 1 to 92.

Note: For *custom tables*, use `init()` and `init_spectral_lines()` to set the data.

X-ray f1 and f2 tables

The data for the tables is stored in the `periodictable/xsf/` directory. The following information is from `periodictable/xsf/read.me`, with minor formatting changes. These `[* .nff]` files were used to generate the tables published in reference ¹⁵. The files contain three columns of data:

Energy(eV), f_1 , f_2 ,

where f_1 and f_2 are the atomic (forward) scattering factors. There are 500+ points on a uniform logarithmic mesh with points added 0.1 eV above and below “sharp” absorption edges. The tabulated values of f_1 contain a relativistic, energy independent, correction given by:

$$Z^* = Z - (Z/82.5)^{2.37}$$

Note: Below 29 eV f_1 is set equal to -9999.

The atomic photoabsorption cross section, μ_a , may be readily obtained from the values of f_2 using the relation:

$$\mu_a = 2r_e\lambda f_2$$

where r_e is the classical electron radius, and λ is the wavelength. The index of refraction for a material with N atoms per unit volume is calculated by:

$$n = 1 - Nr_e\lambda^2(f_1 + if_2)/(2\pi).$$

These (semi-empirical) atomic scattering factors are based upon photoabsorption measurements of elements in their elemental state. The basic assumption is that condensed matter may be modeled as a collection of non-interacting atoms. This assumption is in general a good one for energies sufficiently far from absorption thresholds. In the threshold regions, the specific chemical state is important and direct experimental measurements must be made.

These tables are based on a compilation of the available experimental measurements and theoretical calculations. For many elements there is little or no published data and in such cases it was necessary to rely on theoretical calculations and interpolations across Z. In order to improve the accuracy in the future considerably more experimental measurements are needed.

¹⁵ B. L. Henke, E. M. Gullikson, and J. C. Davis. “X-ray interactions: photoabsorption, scattering, transmission, and reflection at E=50-30000 eV, Z=1-92”, *Atomic Data and Nuclear Data Tables* 54 no.2, 181-342 (July 1993).

Please send any comments about the tables to EMGullikson@lbl.gov.

Note that the following elements have been updated since the publication of Ref. ¹ in July 1993.

Element	Updated	Energy Range
Mg	1/15/94	30-50 eV
Al	1/15/94	30-73 eV
Si	1/15/94	30-100 eV
Au	11/7/94	2000-6500 eV
Li	11/15/94	2000-30000 eV
Si	6/95	30-500 eV
Fe	10/95	600-800 eV
Mo	11/97	10-930 eV
Be	8/04	40-250 eV
Mo	8/04	25-60 eV
W	8/04	35-250 eV
Ru	8/04	40-1300 eV
Ti	8/04	20-150 eV
Sc	4/06	50-1300 eV
Gd	6/07	12-450 eV
La	6/07	14-440 eV

Data available at:

1. http://henke.lbl.gov/optical_constants/asf.html
2. http://henke.lbl.gov/optical_constants/update.html

class `periodictable.xsf.Xray` (*element*)

Bases: `object`

X-ray scattering properties for the elements. Refer `help(periodictable.xsf)` from command prompt for details.

f0 (*Q*)

Isotropic X-ray scattering factors *f0* for the input *Q*.

Parameters

Q [float or vector in $[0, 24\pi]$ | \AA^{-1}] X-ray scattering properties for the elements.

Returns

f0 [float] Values outside the valid range return NaN.

Note: *f0* is often given as a function of $\sin(\theta)/\lambda$ whereas we are using $Q = 4\pi \sin(\theta)/\lambda$, or in terms of energy $Q = 4\pi \sin(\theta)E/(hc)$.

Reference: D. Wassmaier, A. Kerfel, Acta Crystallogr. A51 (1995) 416.
<http://dx.doi.org/10.1107/S0108767394013292>

scattering_factors (**args*, ***kw*)

X-ray scattering factors *f'*, *f''*.

Parameters

energy [float or vector | keV] X-ray energy.

Returns

scattering_factors [(float, float)] Values outside the range return NaN.

Values are found from linear interpolation within the Henke Xray scattering factors database at the Lawrence Berkeley Laboratory Center for X-ray Optics.

sld (*args, **kw)
 X ray scattering length density.

Parameters

wavelength [float or vector | Å] Wavelength of the X-ray.

energy [float or vector | keV] Energy of the X-ray (if *wavelength* not specified).

Returns

sld [(float, float) | Å⁻²] (*real, imaginary*) X-ray scattering length density.

Raises *TypeError* : neither *wavelength* nor *energy* was specified.

The scattering length density is $r_e N(f_1 + i f_2)$. where r_e is the electron radius and N is the number density. The number density is $N = \rho_m / m N_A$, with mass density ρ_m molar mass m and Avogadro's number N_A .

The constants are available directly:

$r_e = \text{periodictable.xsf.electron_radius}$

$N_A = \text{periodictable.constants.avogadro_number}$

Data comes from the Henke Xray scattering factors database at the Lawrence Berkeley Laboratory Center for X-ray Optics.

sftable

X-ray scattering factor table (E,f1,f2)

`periodictable.xsf.init` (*table, reload=False*)

`periodictable.xsf.init_spectral_lines` (*table*)

Sets the *K_alpha* and *K_beta1* wavelengths for select elements

`periodictable.xsf.xray_energy` (*wavelength*)

Convert X-ray wavelength to energy.

Parameters *wavelength* : float or vector | Å

Returns *energy* : float or vector | keV

Wavelength can be converted to energy using

$$E = hc/\lambda$$

where:

h = planck's constant in eV·s

c = speed of light in m/s

`periodictable.xsf.xray_wavelength` (*energy*)

Convert X-ray energy to wavelength.

Parameters *energy* : float or vector | keV

Returns *wavelength* : float | Å

Energy can be converted to wavelength using

$$\lambda = hc/E$$

where:

h = planck's constant in eV·s

c = speed of light in m/s

`periodictable.xsf.xray_sld(*args, **kw)`

Compute xray scattering length densities for molecules.

Parameters

compound [Formula initializer] Chemical formula initializer.

density [float | g·cm⁻³] Density of the compound.

wavelength [float | Å] Wavelength of the X-ray.

energy [float | keV] Energy of the X-ray, if *wavelength* is not specified.

Returns

sld [(float, float) | 10⁻⁶Å⁻²] (*real, imaginary*) scattering length density.

Raises *AssertionError*: *density* or *wavelength/energy* is missing.

`periodictable.xsf.xray_sld_from_atoms(*args, **kw)`

Deprecated since version 0.91: `xray_sld()` now accepts dictionaries of {atom: count} directly.

`periodictable.xsf.emission_table(table=None)`

Prints a table of emission lines.

Parameters

table [PeriodicTable.] The default periodictable unless a specific table has been requested.

Returns None

Example

```
>>> emission_table()
El  Kalpha  Kbeta1
Ti  2.7496  2.5138
Cr  2.2909  2.0848
Mn  2.1031  1.9102
Fe  1.9373  1.7565
Co  1.7905  1.6208
Ni  1.6591  1.5001
Cu  1.5418  1.3922
Zn  1.4364  1.2952
Mo  0.7107  0.6323
Rh  0.6147  0.5456
Pd  0.5869  0.5205
Ag  0.5608  0.4970
```

`periodictable.xsf.sld_table(wavelength=None, table=None)`

Prints the xray SLD table for the given wavelength.

Parameters

wavelength = Cu K-alpha [float | Å] X-ray wavelength.

table [PeriodicTable] The default periodictable unless a specific table has been requested.

Returns None

Example

```
>>> sld_table()
X-ray scattering length density for 1.5418 Ang
El   rho   irho
H    1.19   0.00
He   1.03   0.00
```

```

Li   3.92  0.00
Be  13.93  0.01
B   18.40  0.01
C   17.86  0.03
N    6.88  0.02
O    9.74  0.04
F   12.16  0.07
Ne  10.26  0.09
Na   7.98  0.09
Mg  14.78  0.22
...
    
```

`periodictable.xsf.plot_xsf(el)`

Plots the xray scattering factors for the given element.

Parameters *el*: Element

Returns None

2.9 Magnetic Form Factor

2.9.1 `periodictable.magnetic_ff`

Adds `magnetic_ff[charge].t` for *t* in *j0*, *j2*, *j4*, *j6*, and *J*. *J* should be the dipole approximation $\langle j0 \rangle + (1 - 2/g) \langle j2 \rangle$, according to the documentation for CrysFML¹⁶, but that does not seem to be the case in practice.

class `periodictable.magnetic_ff.MagneticFormFactor`

Bases: object

Magnetic form factor for the ion.

The available form factors are:

```

M = <j0> form factor coefficients
J = <j0> + C2 <j2> form factor coefficients
jn = <jn> form factor coefficients for n = 0, 2, 4, 6
    
```

Not all form factors are available for all ions. Use the expression `hasattr(ion.magnetic_ff, '<ff>')` to test for the particular form factor `<ff>`. The form factor coefficients are a tuple (A, a, B, b, C, c, D). The following expression computes the *M*/*j0* and *J* form factors from the corresponding coefficients:

```

s = q^2 / 16 pi^2
ff = A exp(-a s^2) + B exp(-b s^2) + C exp(-c s^2) + D
    
```

The remaining form factors *j2*, *j4* and *j6* are scaled by an additional s^2 . The form factor calculation is performed by the `<ff>_Q` method for `<ff>` in *M*, *J*, *j0*, *j2*, *j4*, *j6*. For example, here is the calculation for the *M* form factor for Fe²⁺ computed at 0, 0.1 and 0.2:

```

>>> import periodictable
>>> ion = periodictable.Fe.ion[2]
>>> print ion.magnetic_ff[ion.charge].M_Q([0, 0.1, 0.2])
[ 1.          0.99935255  0.99741366]
    
```

`J_Q(Q)`

Returns J scattering potential at $Q \text{ \AA}^{-1}$

¹⁶ Brown. P. J. (Section 4.4.5) International Tables for Crystallography Volume C, Wilson. A. J. C.(ed).

M_Q (*Q*)
Returns *j0* scattering potential at $Q \text{ \AA}^{-1}$

j0_Q (*Q*)
Returns *j0* scattering potential at $Q \text{ \AA}^{-1}$

j2_Q (*Q*)
Returns *j2* scattering potential at $Q \text{ \AA}^{-1}$

j4_Q (*Q*)
Returns *j4* scattering potential at $Q \text{ \AA}^{-1}$

j6_Q (*Q*)
Returns *j6* scattering potential at $Q \text{ \AA}^{-1}$

M
j0

`periodictable.magnetic_ff.formfactor_0` (*j0*, *q*)
Returns the scattering potential for form factor *j0* at the given *q*.

`periodictable.magnetic_ff.formfactor_n` (*jn*, *q*)
Returns the scattering potential for form factor *jn* at the given *q*.

`periodictable.magnetic_ff.init` (*table*, *reload=False*)

INDICES AND TABLES

- *genindex*
- *modindex*

PYTHON MODULE INDEX

p

periodictable.core, 17
periodictable.covalent_radius, 27
periodictable.crystal_structure, 28
periodictable.density, 28
periodictable.formulas, 23
periodictable.magnetic_ff, 45
periodictable.mass, 30
periodictable.nsf, 31
periodictable.xsf, 40

INDEX

A

absorption_comparison_table() (in module periodictable.nsf), 38
abundance (periodictable.core.Isotope attribute), 18
abundance() (in module periodictable.mass), 31
add_isotope() (periodictable.core.Element method), 18
atoms (periodictable.formulas.Formula attribute), 25

C

change_table() (in module periodictable.core), 23
change_table() (periodictable.formulas.Formula method), 23
coherent_comparison_table() (in module periodictable.nsf), 38
covalent_radius (periodictable.core.Element attribute), 19
covalent_radius_uncertainty (periodictable.core.Element attribute), 19
covalent_radius_units (periodictable.core.Element attribute), 19
crystal_structure (periodictable.core.Element attribute), 19

D

default_table() (in module periodictable.core), 23
define_elements() (in module periodictable.core), 22
delayed_load() (in module periodictable.core), 22
density (periodictable.core.Element attribute), 19
density (periodictable.core.Isotope attribute), 18
density() (in module periodictable.density), 29

E

Element (class in periodictable.core), 18
emission_table() (in module periodictable.xsf), 44
energy_dependent_table() (in module periodictable.nsf), 39

F

f0() (periodictable.xsf.Xray method), 42
formfactor_0() (in module periodictable.magnetic_ff), 46
formfactor_n() (in module periodictable.magnetic_ff), 46
Formula (class in periodictable.formulas), 23

formula() (in module periodictable.formulas), 25
formula_grammar() (in module periodictable.formulas), 25

G

get_data_path() (in module periodictable.core), 23
getval() (in module periodictable.mass), 31

H

has_sld() (periodictable.nsf.Neutron method), 33
hill (periodictable.formulas.Formula attribute), 25

I

incoherent_comparison_table() (in module periodictable.nsf), 38
init() (in module periodictable.covalent_radius), 27
init() (in module periodictable.crystal_structure), 28
init() (in module periodictable.density), 30
init() (in module periodictable.magnetic_ff), 46
init() (in module periodictable.mass), 31
init() (in module periodictable.nsf), 34
init() (in module periodictable.xsf), 43
init_spectral_lines() (in module periodictable.xsf), 43
interatomic_distance (periodictable.core.Element attribute), 19
interatomic_distance() (in module periodictable.density), 30
Ion (class in periodictable.core), 17
isatom() (in module periodictable.core), 23
iselement() (in module periodictable.core), 23
ision() (in module periodictable.core), 23
isisotope() (in module periodictable.core), 23
Isotope (class in periodictable.core), 18
isotope() (periodictable.core.PeriodicTable method), 21
isotopes (periodictable.core.Element attribute), 20

J

j0_Q() (periodictable.magnetic_ff.MagneticFormFactor method), 46
j2_Q() (periodictable.magnetic_ff.MagneticFormFactor method), 46

- j4_Q() (periodictable.magnetic_ff.MagneticFormFactor method), 46
- j6_Q() (periodictable.magnetic_ff.MagneticFormFactor method), 46
- J_Q() (periodictable.magnetic_ff.MagneticFormFactor method), 45
- ## K
- K_alpha (periodictable.core.Element attribute), 19
- K_alpha_units (periodictable.core.Element attribute), 19
- K_beta1 (periodictable.core.Element attribute), 19
- K_beta1_units (periodictable.core.Element attribute), 19
- ## L
- list() (periodictable.core.PeriodicTable method), 21
- ## M
- M (periodictable.magnetic_ff.MagneticFormFactor attribute), 46
- M_Q() (periodictable.magnetic_ff.MagneticFormFactor method), 45
- magnetic_ff (periodictable.core.Element attribute), 20
- MagneticFormFactor (class in periodictable.magnetic_ff), 45
- mass (periodictable.core.Element attribute), 20
- mass (periodictable.core.Isotope attribute), 18
- mass (periodictable.formulas.Formula attribute), 25
- mass() (in module periodictable.mass), 31
- mix_by_volume() (in module periodictable.formulas), 26
- mix_by_weight() (in module periodictable.formulas), 26
- molecular_mass (periodictable.formulas.Formula attribute), 25
- ## N
- name() (periodictable.core.PeriodicTable method), 22
- natural_density (periodictable.formulas.Formula attribute), 25
- natural_mass_ratio() (periodictable.formulas.Formula method), 23
- Neutron (class in periodictable.nsf), 32
- neutron (periodictable.core.Element attribute), 20
- neutron (periodictable.core.Isotope attribute), 18
- neutron_composite_sld() (in module periodictable.nsf), 37
- neutron_energy() (in module periodictable.nsf), 34
- neutron_scattering() (in module periodictable.nsf), 34
- neutron_sld() (in module periodictable.nsf), 37
- neutron_sld() (periodictable.formulas.Formula method), 23
- neutron_sld_from_atoms() (in module periodictable.nsf), 40
- neutron_wavelength() (in module periodictable.nsf), 34
- neutron_wavelength_from_velocity() (in module periodictable.nsf), 34
- number_density (periodictable.core.Element attribute), 20
- number_density() (in module periodictable.density), 30
- ## P
- parse_formula() (in module periodictable.formulas), 26
- PeriodicTable (class in periodictable.core), 20
- periodictable.core (module), 17
- periodictable.covalent_radius (module), 27
- periodictable.crystal_structure (module), 28
- periodictable.density (module), 28
- periodictable.formulas (module), 23
- periodictable.magnetic_ff (module), 45
- periodictable.mass (module), 30
- periodictable.nsf (module), 31
- periodictable.xsf (module), 40
- plot_xsf() (in module periodictable.xsf), 45
- ## S
- scattering() (periodictable.nsf.Neutron method), 33
- scattering_factors() (periodictable.xsf.Xray method), 42
- sftable (periodictable.xsf.Xray attribute), 43
- sld() (periodictable.nsf.Neutron method), 33
- sld() (periodictable.xsf.Xray method), 42
- sld_plot() (in module periodictable.nsf), 37
- sld_table() (in module periodictable.nsf), 40
- sld_table() (in module periodictable.xsf), 44
- symbol() (periodictable.core.PeriodicTable method), 22
- ## T
- total_comparison_table() (in module periodictable.nsf), 39
- ## V
- volume() (periodictable.formulas.Formula method), 24
- ## X
- Xray (class in periodictable.xsf), 42
- xray (periodictable.core.Element attribute), 20
- xray (periodictable.core.Ion attribute), 17
- xray_energy() (in module periodictable.xsf), 43
- xray_sld() (in module periodictable.xsf), 43
- xray_sld() (periodictable.formulas.Formula method), 24
- xray_sld_from_atoms() (in module periodictable.xsf), 44
- xray_wavelength() (in module periodictable.xsf), 43