Periodic Table Documentation

Release 1.6.1

Paul Kienzle

May 19, 2022

Contents

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.

CHAPTER 1

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, desnity 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.

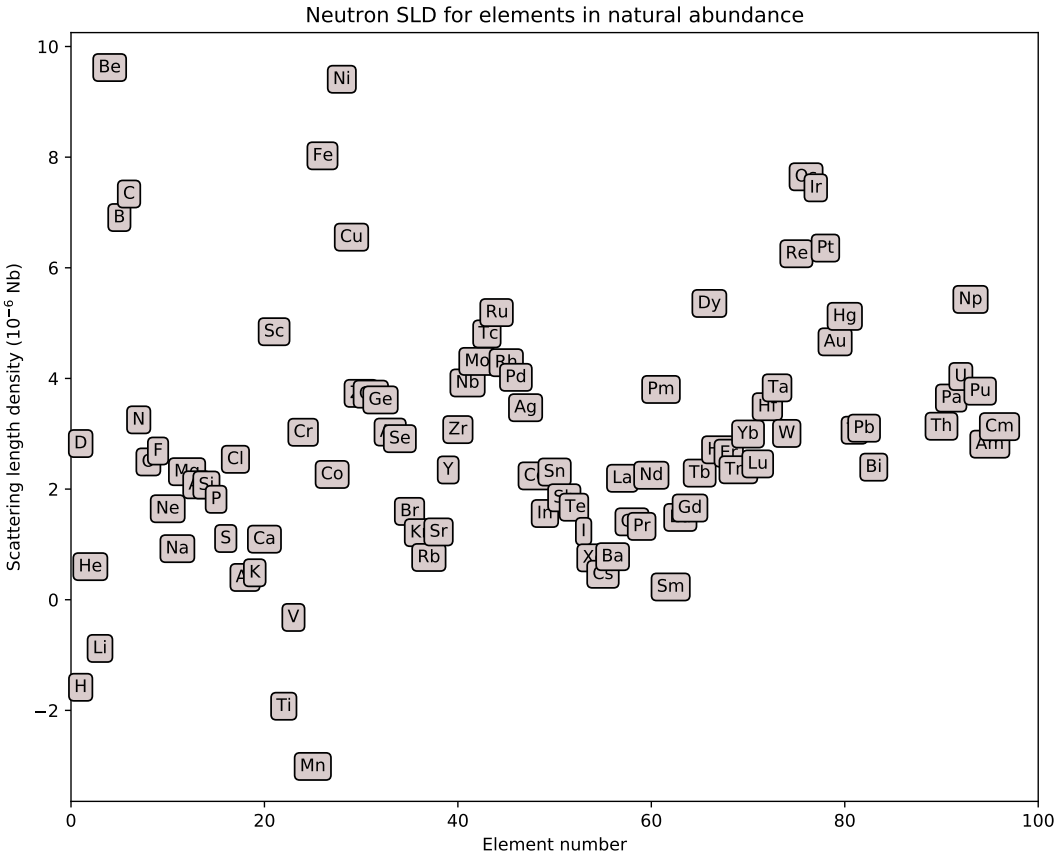
`Isotopes` Mass and relative abundance of isotopes are included for known isotopes. Formulas can include isotope composition.

`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.


Neutron SLD for elements in natural abundance
Scattering length density (10^{-6} Nb)
Element number
10
8
6
4
2
0
−2
0
20
40
60
80
100
Be
Ni
Fe
C
B
Cu
Ir
Re
Pt
Dy
Np
Ru
Sc
Hg
Au
Mo
Pb
Nb
Pd
Pm
Ta
Ge
Ag
Pa
Pu
N
Cr
Zr
W
Th
Cm
D
F
Se
Yb
Pb
Am
Cl
Co
Y
Sn
Lu
Bi
Si
P
Ne
Br
Te
Gd
Pr
Sr
S
Ca
Na
Rb
He
K
Ba
Sm
V
Li
H
Ti
Mn

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 %s %s"%(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 %s"%B.neutron.absorption)
B absorption 767.0
>>> print("Ni f1/f2 for Cu K-alpha X-rays f'=%.5f f''=%.5f"
...       % Ni.xray.scattering_factors(wavelength=Cu.K_alpha))
Ni f1/f2 for Cu K-alpha X-rays f'=25.02293 f''=0.52493
```

Access isotopes using mass number subscripts:

```
>>> print("58-Ni vs 62-Ni scattering %s:%s"%(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 %.2f %s"%(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 %s"%D.mass)
D mass 2.014101778
>>> print("neutron mass %s"%n.mass)
neutron mass 1.00866491597
```

Process all the elements:

```
>>> import periodictable
>>> for el in periodictable.elements:
...     print("%s %s"%(el.symbol,el.name))
n neutron
```

(continues on next page)

(continued from previous page)

```
H hydrogen
He helium
   ...
Og oganesson
```

Another example for processing all elements:

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

Process all the *isotopes* for an element:

```
>>> for iso in periodictable.H:
...     print("%s %s"%(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+ is %d"%Ni_2.charge)
charge for Ni2+ is 2
>>> print("mass for Ni[58] and for natural abundance: %.4f %.4f"%(Ni58_2.mass, Ni_2.
↪mass))
mass for Ni[58] and for natural abundance: 57.9343 58.6923
```

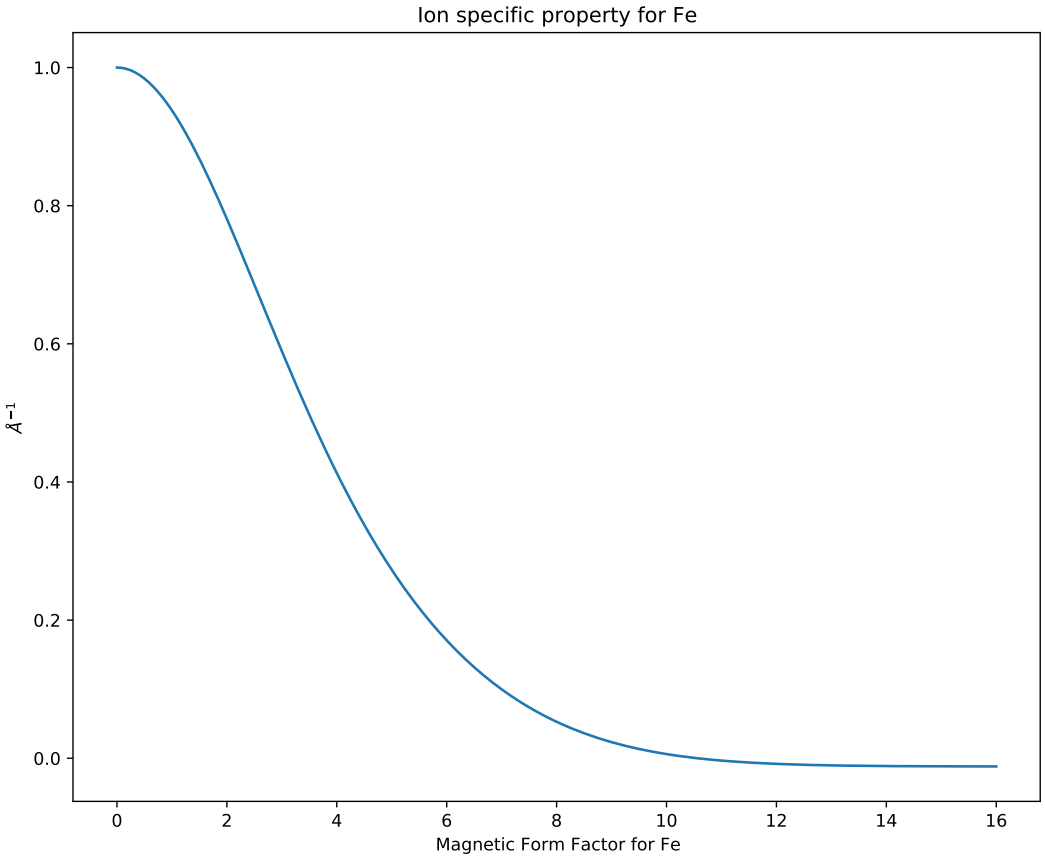
The ion specific properties can be accessed from the ion using ion.charge for the ion index:

```
>>> import periodictable
>>> Fe_2 = periodictable.Fe.ion[2]
>>> print("[%.5f, %.5f, %.5f]"
...       % tuple(Fe_2.magnetic_ff[Fe_2.charge].M_Q([0,0.1,0.2])))
[1.00000, 0.99935, 0.99741]
```

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

```
>>> import pylab
>>> 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*:


Ion specific property for Fe
1.0
0.8
0.6
0.4
0.2
0.0
Å⁻¹
0
2
4
6
8
10
12
14
16
Magnetic Form Factor for Fe

```
>>> print("Radon density %s"%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")
Ne K-alpha = 14.6102
Na K-alpha = 11.9103
Mg K-alpha = 9.8902
Al K-alpha = 8.3402
   ...
Cf K-alpha = 0.1094
Es K-alpha = 0.1067
Fm K-alpha = 0.104
```

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] = ^{18}O:

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

- Ions are represented by charge, such as O{2-} = O^{2-}:

```
>>> print(formula("P{5+}O{2-}4"))
P{5+}O{2-}4
```

  If charge is +/- 1 then the number is optional:

```
>>> print(formula("Na{+}Cl{1-}"))
Na{+}Cl{-}
```

When specifying both charge and isotope, isotope comes first:

```
>>> print(formula("Fe[56]{2+}"))
Fe[56]{2+}
```

Even though the charge is on the individual atoms, the entire formula has a charge:

```
>>> print(formula("P{5+}O{2-}4").charge)
-3
```

- Counts can be integer or decimal:

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

- Formula density can be specified using the special '@' tag:

```
>>> print(formula("NaCl@2.16").density)
2.16
```

Density gives the isotopic density of the compound, so for example, D2O could be specified using:

```
>>> print("%.3f"%formula("D2O@1.112").density)
1.112
```

It can also be specified using the natural density of the compound, assuming the isotopes substitution does not change the unit cell volume:

```
>>> print("%.3f"%formula("D2O@1n").density)
1.112
```

Density applies to the entire formula, so for example a D2O-H2O 2:1 mixture (not by mass or by volume) would be:

```
>>> print("%.3f"%formula("2D2O + H2O@1n").density)
1.074
```

- Mass fractions use %wt, with the final portion adding to 100%:

```
>>> print(formula("10%wt Fe // 15% Co // Ni"))
FeCo1.4214Ni7.13602
```

Only the first item needs to specify that it is a mass fraction, and the remainder can use a bare %.

- Volume fractions use %vol, with the final portion adding to 100%:

```
>>> print(formula("10%vol Fe // Ni"))
FeNi9.68121
```

Only the first item needs to specify that it is a volume fraction, and the remainder can use a bare %.

Volume fraction mixing is only possible if the densities are known for the individual components, which will require the formula density tag if the component is not an element. A density estimate is given for the mixture but in general it will not be correct, and should be set explicitly for the resulting compound.

- Specific mass can be giving with count follwed by mass units:

```
>>> print(formula("5g NaCl // 50mL H2O@1"))
NaCl(H2O)32.4407
```

  Density will be required for materials given by volume. Mass will be stored in the *total_mass* attribute of the resulting formula.

- Multilayers can be specified by thickness:

```
>>> print(formula("1 um Si // 5 nm Cr // 10 nm Au"))
Si119.99CrAu1.41722
```

  Density will be required for each layer. Thickness will be stored in the *total_thickness* attribute of the resulting formula. Thickness can be converted to *total_volume* by multiplying by cross section, and to *total_mass* by multiplying that by *density*.

- Mixtures can nest. The following is a 10% salt solution by weight mixed 20:80 by volume with D2O:

```
>>> print(formula("20%vol (10%wt NaCl@2.16 // H2O@1) // D2O@1n"))
NaCl(H2O)29.1966(D2O)122.794
```

- 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:

```
formula     :: compound | mixture | nothing
mixture     :: quantity | percentage
quantity    :: count unit part ('//' count unit part)*
percentage  :: count '%wt|%vol' part ('//' count '%' part)* '//' part
part        :: compound | '(' mixture ')'
compound    :: group (separator group)* density?
group       :: count element+ | '(' formula ')' count
element     :: symbol isotope? ion? count?
symbol      :: [A-Z][a-z]*
isotope     :: '[' number ']'
ion         :: '{' number? [+-] '}'
density     :: '@' count
count       :: number | fraction
number      :: [1-9][0-9]*
fraction    :: ([1-9][0-9]* | 0)? '.' [0-9]*
separator   :: space? '+'? space?
unit        :: mass | volume | length
mass        :: 'kg' | 'g' | 'mg' | 'ug' | 'ng'
volume      :: 'L' | 'mL' | 'uL' | 'nL'
length      :: 'cm' | 'mm' | 'um' | 'nm'
```

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("%s %.4g"%(D2O,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("%s %.4g"%(mix,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("%s %.4g"%(mix,mix.density))
(H2O)2.2234D2O 1.035
```

Except in the simplest of cases, the density of the mixture cannot be computed from the densities of the components, and the resulting density should be set explicitly.

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

1.3.4 Biomolecules

The *periodictable.fasta* module can be used to load and manage bio molecules. These can be used to compute molecular weights, approximate volumes and scattering for various lipids and proteins. In addition it supports labile hydrogen calculations, allowing you to compute the neutron scattering length density of the molecule in the presence of D2O as a solvent, assuming all labile hydrogens are substituted.

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.items():
        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",
```

(continues on next page)

(continued from previous page)

```
 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
```

(continues on next page)

(continued from previous page)

```
    # Load the data
    for symbol,eldata in data.items():
        el = table.symbol(symbol)
        for iso,isodata in eldata.items():
            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 neutron scattering factors as `Ni[58]` (the neutron is only sensitive to the nucleus), but different scattering factors from `Ni.ion[3]`. X-rays are sensitive to the electronic structure of the atom and not the nucleus, 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]`. Support for ion mass, which is isotope mass adjusted for the number of electrons is built into the Ion class. There are not yet any examples of extension tables that depend on both isotope and ion.

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 is not None:
...         el._density /= scale
...     for iso in el:
...         iso._mass /= scale

>>> print("%.10f %.10f"%(mytable.H[1].mass, mytable.C[12].mass))
1.0000000000 11.9068286833
>>> print("%.10f %.10f"%(periodictable.H[1].mass, periodictable.C[12].mass))
1.0078250321 12.0000000000
>>> print("%.10f"%formula('2H[1]', table=mytable).mass)
2.0000000000
```

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

The table name (*H=1* 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

Neutron activation IAEA (1987) Handbook on Nuclear Activation Data. TR 273 (International Atomic Energy Agency, Vienna, Austria, 1987).

Shleien, B., Slaback, L.A., Birky, B.K., (1998). *Handbook of health physics and radiological health*. Williams & Wilkins, Baltimore.

Crystal structure Ashcroft and Mermin

Oxidation states wikipedia: List of oxidation states of the elements

Greenwood, Norman N.; Earnshaw, Alan (1997). *Chemistry of the Elements*, 2nd ed. Butterworth-Heinemann. pp. 27–28. ISBN 0-08-037941-9.

FASTA and biomolecule formulas Perkins (1988), *Modern Physical Methods in Biochemistry Part B*, 143-265.

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 at https://github.com/pkienzle/periodictable. To make changes, create a fork of the project on github, then do following, with your github user name substituted for *GITNAME*:

```
git clone https://github.com/GITNAME/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 remote add upstream https://github.com/pkienzle/periodictable.git
git remote -v   # check that it is set
git fetch upstream
git merge upstream/master
```

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:

```
pip install pytest pytest-cov
pytest
```

When all the tests run, create a pull request (PR) on the github page.

Windows user can use TortoiseGit package which provides similar operations.

You can build the documentation as follows:

```
pip install sphinx
(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
```

ReStructured text format does not have a nice syntax for superscripts and subscripts. Units such as g·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 with contributions from the DANSE project.

The DANSE/Reflectometry team is supported by the University of Maryland Department of Materials Science and Engineering, the NIST Center for Neutron Research and the US National Science Foundation grant DMR-0520547. Any opinions, findings and recommendations expressed in this material are those of the author(s) and do not necessarily reflect the view of the supporting institutions.

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.

CHAPTER 2

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]`. Note that `element[isotope].ion[charge].mass` will depend on the particular charge since we subtract the charge times the rest mass of the electron from the overall mass.

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.

mass

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 occuring 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 Nuetron Data Booklet.*

neutron_activation
Neutron activation calculations for isotopes and formulas.

Reference: *IAEA 273: Handbook on Nuclear Activation Data. NBSIR 85-3151: Compendium of Benchmark Neutron Field.*

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.

abundance_units = '%'

charge = 0

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 occuring 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).*

density_units = 'g/cm^3'

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:

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

interatomic_distance_units = 'angstrom'

isotopes
List of all isotopes

magnetic_ff
Magnetic Form Fators. 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.*

mass_units = 'u'

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

Reference: *Rauch. H. and Waschkowski. W., ILL Nuetron 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 | cm^{-3}] Number density of a element.

Number density is computed using:

$$d = N_A \frac{\rho}{m}$$

with units:

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

number_density_units = '1/cm^3'

table = 'public'

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. Individidual 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("%s %s"%(el.symbol, el.name))
n neutron
H hydrogen
He helium
...
Og oganesson
```

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 operations

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`(*wavelength=None*, *energy=None*)
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.

replace (*source*, *target*, *portion=1*)

Create a new formula with one atom/isotope substituted for another.

formula is the formula being updated.

source is the isotope/element to be substituted.

target is the replacement isotope/element.

portion is the proportion of source which is substituted for target.

volume (**args*, ***kw*)

Estimate unit cell volume.

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

```
packing_factor = N_atoms V_atom / V_crystal
```

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

Table 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 | cm^3] 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 = n*formula.molecular_mass/formula.volume()
```

where n is the number of molecules per unit cell.

Note: a single non-keyword argument is interpreted as a packing factor rather than a lattice spacing of 'a'.

xray_sld (*energy=None*, *wavelength=None*)

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.

charge

Net charge of the molecule.

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.

mass_fraction

Fractional mass representation of each element/isotope/ion.

molecular_mass

g

Mass of the molecule in grams.

natural_density

g·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**(*compound=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

compound [Formula initializer] Chemical formula.

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

natural_density [float | g·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 a ValueError is raised. The density calculation assumes the cell volume remains constant for the original materials, which is not in general the case.

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. The density calculation assumes the cell volume remains constant for the original materials, which is not in general the case.

`periodictable.formulas.`**`parse_formula`**(*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., 2008[1]. Bond specific values (single, double, or triple) are available from PyykkÃ¶ et al., 2009[2], but they are generally smaller. The CRC Handbook uses the average of Cordero and PyykkÃ¶. Note that the combined Cordero/PyykkÃ¶ 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] Beatriz Cordero, VerÃ³nica GÃ³mez, Ana E. Platero-Prats, Marc RevÃ©s, Jorge EcheverrÃa, Eduard Cremades, Flavia BarragÃ¡n and Santiago Alvarez. Covalent radii revisited. Dalton Trans., 2008, 2832-2838. doi:10.1039/b801115j

[2] P. PyykkÃ¶ 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

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.

2.4 Fundamental constants

2.4.1 periodictable.constants

Various fundamental constants.

periodictable.constants.**atomic_mass_constant = 1.660538782e-27**
atomic mass constant (kg / u)

periodictable.constants.**avogadro_number = 6.02214179e+23**
Avogadro's number (mol^-1)

periodictable.constants.**electron_mass = 0.000548577990946**
electron mass (u)

periodictable.constants.**electron_radius = 2.8179402894e-15**
electron radius r_e (m)

periodictable.constants.**electron_volt = 1.602176487e-19**
Electron volt (J/eV)

periodictable.constants.**neutron_mass = 1.00866491597**
neutron mass (u)

periodictable.constants.**plancks_constant = 4.13566733e-15**
Planck's constant (eV s)

periodictable.constants.**speed_of_light = 299792458**
speed of light c (m/s)

2.5 Crystal structure

2.5.1 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: 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*)
Add crystal_structure field to the element properties.

2.6 Density

2.6.1 `periodictable.density`

The following properties are added:

- **density, density_units (g·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 (cm^{-3})** 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.

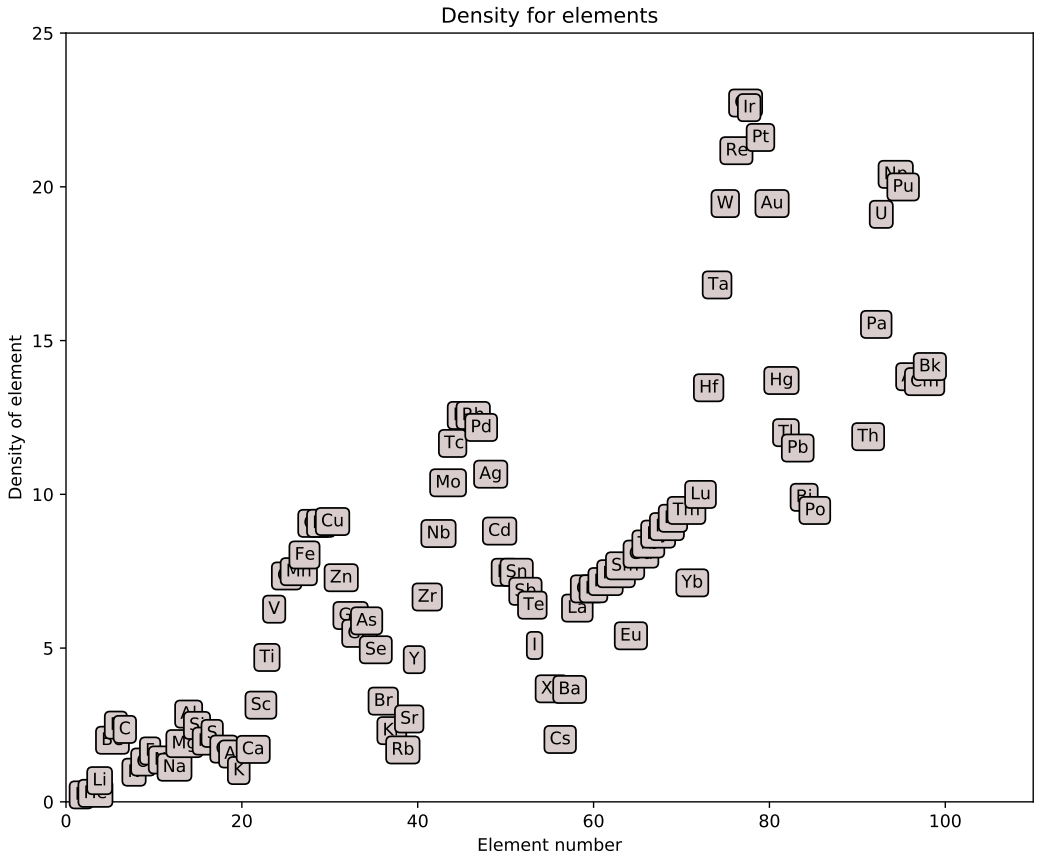
```
>>> from periodictable import D, H
>>> print("H: %.4f, D: %.4f"%(H.density, D.density))
H: 0.0708, D: 0.1415
```

(continues on next page)

(continued from previous page)

```
>>> 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[1]. These values are reproduced in[2].

`periodictable.density.`**`density`**(*iso_el*)

Element density for natural abundance. For isotopes, return the equivalent density assuming identical interatomic spacing as the naturally occuring 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).*

[1] Lide. D. R., Ed., CRC Handbook of Chemistry and Physics, 80th ed. (CRC Press, Boca Raton, Florida, 1999)
[2] The ILL Neutron Data Booklet, Second Edition.

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:

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

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 | cm^{-3}] Number density of a element.

Number density is computed using:

$$d = N_A \frac{\rho}{m}$$

with units:

$$(\mathrm{atoms}\cdot\mathrm{mol}^{-1})(\mathrm{g}\cdot\mathrm{cm}^{-3})/(\mathrm{g}\cdot\mathrm{mol}^{-1}) = \mathrm{atoms}\cdot\mathrm{cm}^{-3}$$

2.7 FASTA format for DNA/RNA and amino acid sequences

2.7.1 periodictable.fasta

Biomolecule support.

Molecule lets you define biomolecules with labile hydrogen atoms specified using H[1] in the chemical formula. The biomolecule object creates forms with natural isotope ratio, all hydrogen and all deuterium. Density can be provided as natural density or cell volume. A %D2O contrast match value is computed for matching the molecule SLD in the presence of labile hydrogens. *Molecule.D2Osld()* computes the neutron SLD for the solvated molecule in a %D2O solvent.

Sequence lets you read amino acid and DNA/RNA sequences from FASTA files.

Tables for common molecules are provided[1]:

AMINO_ACID_CODES : amino acids indexed by FASTA code

RNA_CODES, DNA_CODES* : nucleic bases indexed by FASTA code

RNA_BASES, DNA_BASES* : individual nucleic acid bases

NUCLEIC_ACID_COMPONENTS, *LIPIDS*, *CARBOHYDRATE_RESIDUES*

Neutron SLD for water at 20C is also provided as *H2O_SLD* and *D2O_SLD*.

For unmodified protein need to add 2*H[1] and O for terminations.

Assumes that proteins were created in an environment with the usual H/D isotope ratio on the non-swappable hydrogens.

[1] Perkins, S.J., 1985. Chapter 6 X-Ray and Neutron Solution Scattering, in: New Comprehensive Biochemistry. Elsevier, pp. 143-265. https://doi.org/10.1016/S0167-7306(08)60575-X

AMINO_ACID_CODES:

```
-: gap
A: alanine
B: aspartic acid/asparagine
C: cysteine
D: aspartic acid
E: glutamic acid
F: phenylalanine
G: glycine
H: histidine
I: isoleucine
J: leucine/isoleucine
K: lysine
L: leucine
M: methionine
N: asparagine
P: proline
Q: glutamine
R: arginine
S: serine
T: threonine
V: valine
W: tryptophan
X: any
Y: tyrosine
Z: glutamic acid/glutamine
```

NUCLEIC_ACID_COMPONENTS:

```
adenine: C5H2H[1]2N5
cytosine: C4H2H[1]2N3O
deoxyribose: C5H7O2
guanine: C5HH[1]3N5O
phosphate: NaPO3
ribose: C5H6H[1]O3
thymine: C5H4H[1]N2O2
uracil: C4H2H[1]N2O2
```

CARBOHYDRATE_RESIDUES:

```
Fuc (terminal): C6H7H[1]3O4
Gal: C6H7H[1]3O5
```

(continues on next page)

(continued from previous page)

```
GalNAc: C8H10H[1]3NO5
Glc: C6H7H[1]3O5
GlcNAc: C8H10H[1]3NO5
Man: C6H7H[1]3O5
Man (terminal): C6H7H[1]4O5
NeuNac (terminal): C11H11H[1]5NO8
chondroitin sulphate: C14H15H[1]4NO14SNa
hyaluronate: C14H15H[1]5NO11Na
keratan sulphate: C14H17H[1]5NO13SNa
```

LIPIDS:

```
DLPE: C29H55H[1]3NO8P
DMPC: C36H72NO8P
DMPC-D52: C36H20D52NO8P
cholesteral: C27H45H[1]O
methylene: CH2
methylene-D: CD2
oleate: C45H78O2
palmitate ester: C39H77H[1]2N2O2P
phospholipid headgroup: C10H18NO8P
triglyceride headgroup: C6H5O6
trioleate form: C57H104O6
```

RNA_BASES:

```
A:adenosine
C:cytidine
G:guanosine
T:uridine
```

DNA_BASES:

```
A:adenosine
C:cytidine
G:guanosine
T:thymidine
```

class periodictable.fasta.**Molecule**(*name*, *formula*, *cell_volume=None*, *density=None*, *charge=0*)

Bases: object

Specify a biomolecule by name, chemical formula, cell volume and charge.

Labile hydrogen positions should be coded using H[1] rather than H. H[1] will be substituded with H for solutions with natural water or D for solutions with heavy water. Any deuterated non-labile hydrogen can be marked with D, and they will stay as D regardless of the solvent.

name is the molecule name.

formula is the chemical formula as string or atom dictionary, with H[1] for labile hydrogen.

cell_volume is the volume of the molecule. If None, cell volume will be inferred from the natural density of the molecule. Cell volume is assumed to be independent of isotope.

density is the natural density of the molecule. If None, density will be inferred from cell volume.

charge is the overall charge on the molecule.

Attributes

labile_formula is the original formula, with H[1] for the labile H. You can retrieve the deuterated from using:

```
molecule.labile_formula.replace(elements.H[1], elements.D)
```

natural_formula has H substituted for H[1] in *labile_formula*.

D2Omatch is percentage of D2O by volume in H2O required to match the SLD of the molecule, including substitution of labile hydrogen in proportion to the D/H ratio in the solvent. Values will be outside the range [0, 100] if the contrast match is impossible.

sld/*Dsld* are the the SLDs of the molecule with H[1] replaced by naturally occurring H/D ratios and pure D respectively.

mass/*Dmass* are the masses for natural H/D and pure D respectively.

charge is the charge on the molecule

cell_volume is the estimated cell volume for the molecule

density is the estimated molecule density

Change 1.5.3: drop *Hmass* and *Hsld*. Move *formula* to *labile_formula*. Move *Hnatural* to *formula*.

D2Osld(*volume_fraction=1.0*, *D2O_fraction=0.0*)
Neutron SLD of the molecule in a deuterated solvent.

Changed 1.5.3: fix errors in SLD calculations.

class periodictable.fasta.**Sequence**(*name*, *sequence*, *type='aa'*)
Bases: *periodictable.fasta.Molecule*

Convert FASTA sequence into chemical formula.

name sequence name

sequence code string

type is one of:

```
aa: amino acid sequence
dna: dna sequence
rna: rna sequence
```

Note: rna sequence files treat T as U and dna sequence files treat U as T.

D2Osld(*volume_fraction=1.0*, *D2O_fraction=0.0*)
Neutron SLD of the molecule in a deuterated solvent.

Changed 1.5.3: fix errors in SLD calculations.

static load(*filename*, *type=None*)
Load the first FASTA sequence from a file.

static loadall(*filename*, *type=None*)
Iterate over sequences in FASTA file, loading each in turn.

Yields one FASTA sequence each cycle.

periodictable.fasta.**D2Omatch**(*Hsld*, *Dsld*)
Find the D2O% concentration of solvent such that neutron SLD of the material matches the neutron SLD of the solvent.

Hsld, *Dsld* are the SLDs for the hydrogenated and deuterated forms of the material respectively, where *D* includes all the labile protons swapped for deuterons. Water SLD is calculated at 20 C.

Note that the resulting percentage is only meaningful between 0% to 100%. Beyond 100% you will need an additional constrast agent in the 100% D2O solvent to increase the SLD enough to match.

Deprecated since version 1.5.3: Use periodictable.nsf.D2O_match(formula) instead.

Change 1.5.3: corrected D2O sld, which will change the computed match point.

periodictable.fasta.**fasta_table**()

periodictable.fasta.**isotope_substitution**(*formula*, *source*, *target*, *portion=1*)
Substitute one atom/isotope in a formula with another in some proportion.

formula is the formula being updated.

source is the isotope/element to be substituted.

target is the replacement isotope/element.

portion is the proportion of source which is substituted for target.

Deprecated since version 1.5.3: Use formula.replace(source, target, portion) instead.

periodictable.fasta.**read_fasta**(*fp*)
Iterate over the sequences in a FASTA file.

Each iteration is a pair (sequence name, sequence codes).

Change 1.5.3: Now uses H[1] rather than T for labile hydrogen.

periodictable.fasta.**test**()

periodictable.fasta.**D2O_SLD = 6.390934026937301**
real portion of D2O sld at 20 C Change 1.5.2: Use correct density in SLD calculation

periodictable.fasta.**H2O_SLD = -0.5595112084983276**
real portion of H2O sld at 20 C

2.8 Magnetic Form Factor

2.8.1 periodictable.magnetic_ff

Adds magnetic_ff[charge].t for t in j0, j2, j4, j6, and J. J should be the dipole approximation <j0> + (1 - 2/g) <j2>, according to the documentation for CrysFML[1] , 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 coeffients
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:

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

```
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 scalled 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 Fe^2+ computed at 0, 0.1 and 0.2:

```
>>> import periodictable
>>> ion = periodictable.Fe.ion[2]
>>> print("[%.5f, %.5f, %.5f]"
...       % tuple(ion.magnetic_ff[ion.charge].M_Q([0, 0.1, 0.2])))
[1.00000, 0.99935, 0.99741]
```

J_Q(*Q*)
: Returns J scattering potential at *Q* Å^{-1}

M_Q(*Q*)
: Returns *j0* scattering potential at *Q* Å^{-1}

j0_Q(*Q*)
: Returns *j0* scattering potential at *Q* Å^{-1}

j2_Q(*Q*)
: Returns *j2* scattering potential at *Q* Å^{-1}

j4_Q(*Q*)
: Returns *j4* scattering potential at *Q* Å^{-1}

j6_Q(*Q*)
: Returns j6 scattering potential at *Q* Å^{-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*)
: Add magnetic form factor properties to the periodic table

2.9 Mass

2.9.1 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[1].

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[2] 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 were published by Rosman[3] and Taylor[4] in Isotopic Compositions of the Elements 1997. The relative atomic masses of the isotopes data were published by Audi[5] and Wapstra[6] 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.**init**(*table*, *reload=False*)
Add mass attribute to period table elements and isotopes

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.*

2.10 Neutron activation

2.10.1 periodictable.activation

Calculate expected neutron activation from time spent in beam line.

Notation information for activation product:

m, m1, m2: indicate metastable states. Decay may be to the ground state or to another nuclide.

+: indicates radioactive daughter production already included in daughter listing several parent t1/2's required to acheive calculated daughter activity. All activity assigned at end of irradiation. In most cases the added activity to the daughter is small.

*: indicates radioactive daughter production NOT calculated, approximately secular equilibrium

[1] Coursey. J. S., Schwab. D. J., and Dragoset. R. A., NIST, Physics Laboratory, Office of Electronic Commerce in Scientific and Engineering Data.
[2] Coplen. T. B. : U.S. Geological Survey, Reston, Virginia, USA.
[3] Rosman. K. J. R. : Department of Applied Physics, Curtin University of Technology, Australia.
[4] Taylor. P. D. P. : Institute for Reference Materials and Measurements, European Commission, Belgium.
[5] Audi. G. : Centre de SpectromÃ©trie NuclÃ©aire et de SpectromÃ©trie de Masse, Orsay Campus, France.
[6] Wapstra. A. H. : National Institute of Nuclear Physics and High-Energy Physics, Amsterdam, The Netherlands.

s: indicates radioactive daughter of this nuclide in secular equilibrium after several daughter t1/2's

t: indicates transient equilibrium via beta decay. Accumulation of that nuclide during irradiation is separately calculated.

Reaction = b indicates production via decay from an activation produced parent

Accounts for burnup and 2n, g production.

This is only a gross estimate. Many effects are not taken into account, such as self-shielding in the sample and secondary activation from the decay products.

Example:

```
>>> from periodictable import activation
>>> env = activation.ActivationEnvironment(fluence=1e5, Cd_ratio=70, fast_ratio=50,␣
↪location="BT-2")
>>> sample = activation.Sample("Co30Fe70", 10)
>>> sample.calculate_activation(env, exposure=10, rest_times=[0, 1, 24, 360])
>>> sample.show_table()
                                    ----------------- activity (uCi) ---------------
↪---
isotope  product  reaction  half-life        0 hrs        1 hrs       24 hrs      360␣
↪hrs
-------- -------- -------- ---------- ------------ ------------ ------------ ---------
↪---
Co-59    Co-60         act    5.272 y     0.000496     0.000496    0.0004958    0.
↪0004933
Co-59    Co-60m+       act     10.5 m        1.664       0.0317          ---          ␣
↪---
-------- -------- -------- ---------- ------------ ------------ ------------ ---------
↪---
                              total        1.665      0.03221    0.0005084     0.
↪000505
-------- -------- -------- ---------- ------------ ------------ ------------ ---------
↪---

>>> print("%.3f"%sample.decay_time(0.001)) # number of hours to reach 1 nCi
2.053
```

The default rest times used above show the sample activity at the end of neutron activation and after 1 hour, 1 day, and 15 days.

The neutron activation table, *activation.dat*,[1] contains details about the individual isotopes, with interaction cross sections taken from from IAEA-273[2].

Activation can be run from the command line using:

```
$ python -m periodictable.activation FORMULA
```

where FORMULA is the chemical formula for the material.

class periodictable.activation.**ActivationEnvironment**(*fluence=100000.0*, *Cd_ratio=0.0*, *fast_ratio=0.0*, *location=''*)

Bases: `object`

Neutron activation environment.

[1] Shleien, B., Slaback, L.A., Birky, B.K., 1998. Handbook of health physics and radiological health. Williams & Wilkins, Baltimore.

[2] IAEA (1987) Handbook on Nuclear Activation Data. TR 273 (International Atomic Energy Agency, Vienna, Austria, 1987). http://cds.cern.ch/record/111089/files/IAEA-TR-273.pdf

The activation environment provides details of the neutron flux at the sample position.

fluence : float | n/cm^2/s

Thermal neutron fluence on sample. For COLD neutrons enter equivalent thermal neutron fluence.

Cd_ratio : float

Neutron cadmium ratio. Use 0 to suppress epithermal contribution.

fast_ratio : float

Thermal/fast ratio needed for fast reactions. Use 0 to suppress fast contribution.

epithermal_reduction_factor
Used as a multiplier times the resonance cross section to add to the thermal cross section for all thermal induced reactions.

class periodictable.activation.**ActivationResult**(***kw*)
Bases: object

class periodictable.activation.**Sample**(*formula*, *mass*, *name=None*)
Bases: object

Sample properties.

formula : chemical formula

Chemical formula. Any format accepted by *formulas.formula()* can be used, including formula string.

mass : float | g

Sample mass.

name : string

Name of the sample (defaults to formula).

calculate_activation(*environment*, *exposure=1*, *rest_times=(0, 1, 24, 360)*, *abundance=<function NIST2001_isotopic_abundance>*)
Calculate sample activation after exposure to a neutron flux.

environment is the exposure environment.

exposure is the exposure time in hours (default is 1 h).

rest_times is the list of deactivation times in hours (default is [0, 1, 24, 360]).

abundance is a function that returns the relative abundance of an isotope. By default it uses *NIST2001_isotopic_abundance()*, and there is the alternative *IAEA1987_isotopic_abundance()*.

decay_time(*target*)
After determining the activation, compute the number of hours required to achieve a total activation level after decay.

show_table(*cutoff=0.0001*, *format='%.4g'*)
Tabulate the daughter products.

cutoff=1 : float | uCi

The minimum activation value to show.

format="%.1f" : string

The number format to use for the activation.

periodictable.activation.**IAEA1987_isotopic_abundance**(*iso*)
Isotopic abundance in % from the IAEA, as provided in the activation.dat table.

Note: this will return an abundance of 0 if there is no neutron activation for the isotope even though for isotopes such as H[1], the natural abundance may in fact be rather large.

IAEA 273: Handbook on Nuclear Activation Data, 1987.

periodictable.activation.**NIST2001_isotopic_abundance**(*iso*)
Isotopic abundance in % from the periodic table package.

BÃ¶hlke, et al. Isotopic Compositions of the Elements, 2001. J. Phys. Chem. Ref. Data, Vol. 34, No. 1, 2005

periodictable.activation.**activity**(*isotope*, *mass*, *env*, *exposure*, *rest_times*)
Compute isotope specific daughter products after the given exposure time and rest period.

periodictable.activation.**demo**()

periodictable.activation.**find_root**(*x*, *f*, *df*, *max=20*, *tol=1e-10*)
Find zero of a function.

Returns when $|f(x)| < tol$ or when max iterations have been reached, so check that $|f(x)|$ is small enough for your purposes.

Returns x, f(x).

periodictable.activation.**init**(*table*, *reload=False*)
Add neutron activation levels to each isotope.

periodictable.activation.**sorted_activity**(*activity_pair*)
Interator over activity pairs sorted by isotope then daughter product.

2.11 Neutron scattering potentials

2.11.1 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("%s %7.4f"%(iso,iso.neutron.sld()[0]))
58-Ni 13.1526
60-Ni  2.5575
61-Ni  6.9417
62-Ni -7.9464
64-Ni -0.3379
```

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 densitys 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[1] (retrieve March 2008):

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 defined:

- **b_c (fm)** Bounds coherent scattering length.
- **total (barn)** Total scattering cross section σ_s. This does not include the absorption cross section. To compute the total collision cross section use $\sigma_t = \sigma_s + \sigma_a$
- **absorption (barn)** Absorption cross section σ_a at 1.798 Å. Scale to your beam by dividing by periodictable.nsf.ABSORPTION_WAVELENGTH and multiplying by your wavelength.
- **b_c_complex (fm)** Complex coherent scattering length derived from the tabulated values using $b_c - i\sigma_a/(1000 \cdot 2\lambda)$.

Additional columns not used for calculation include:

[1] 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, Philidelphia, PA. pp 1.1-1 to 1.1-17. (https://www.ill.eu/fileadmin/user_upload/ILL/1_About_ILL/Documentation/NeutronDataBooklet.pdf)

- **b_c_i (fm)** Imaginary bound coherent scattering length. This is related to absorption cross section by $\sigma_a = 4\pi \mathrm{Im}(b_c)/k$ 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 Booklet[1] 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. This is tabulated but not used. In theory coherent scattering is related to bound coherent scattering by $\sigma_c = 4\pi|\mathrm{Re}(b_c) + i\mathrm{Im}(b_c)|^2/100$. In practice, these values are different, with the following table showing the largest relative difference:

Sc 3%	Ti 4%	V 34%	Mn 1%	Cd 2%
Te 4%	Xe 9%	Sm 19%	Eu 44%	Tb 1%
Ho 11%	W 4%	Au 7%	Hg 2%	Ra 3%

- **incoherent (barn)** Incoherent scattering cross section σ_i. This is tabulated but not used. Instead, the incoherent cross section is computed from the total cross section minus the coherent cross section even for single atoms so that results from compounds are consistent with results from single atoms.

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 `T` above has a corresponding `T_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 fm^2

---

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

scattering(*wavelength=1.798*)
Returns neutron scattering information for the element at natural abundance and density.

Parameters *wavelength* : float(s) | Å

Returns

***sld* : float(s), float(s), float(s) | 10^{-6}Å^{-2}** (*real*, *-imaginary*, *incoherent*) scattering length density

***xs* : float(s), float(s), float(s) | cm^{-1}** (*coherent*, *absorption*, *incoherent*) cross sections. :w

***penetration* : float(s) | cm** 1/e penetration length.

Returns (None, None, None) if sld is not known for this element.

See *neutron_scattering()* for details.

scattering_by_wavelength(*wavelength*)
Return scattering length and total cross section for each wavelength.

For rare earth isotopes this returns the energy-dependent $\mathrm{Re}(b_c)$ and $\mathrm{Im}(b_c)$ interpolated into the scattering length tables. Values are extrapolated with constant values at the ends of the table. Total scattering is returned as $4\pi/100|b_c|^2$ with no contribution for bound incoherent scattering.

Parameters *wavelength* : float(s) | Å

Returns *b_c* : complex(s) | fm

sigma_s : float(s) | barn

sld(*wavelength=1.798*)

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

Parameters *wavelength* : float(s) | Å

Returns

***sld* : float(s), float(s), float(s) | 10^{-6}Å^{-2}** (*real*, *-imaginary*, *incoherent*) scattering length density.

Returns (None, None, None) if sld is not known for this element.

See *neutron_scattering()* for details.

absorption = None

absorption_units = 'barn'

abundance = 0.0

abundance_units = '%'

b_c = None

b_c_complex = None

b_c_complex_units = 'fm'

b_c_i = None

b_c_i_units = 'fm'

b_c_units = 'fm'

bm = None

bm_i = None

bm_units = 'fm'

bp = None

bp_i = None

bp_units = 'fm'

coherent = None

coherent_units = 'barn'

incoherent = None

incoherent_units = 'barn'

is_energy_dependent = False

nsf_table = None

total = None

total_units = 'barn'

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

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

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

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

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

$$\lambda = h/p = h/(m_n v)$$

where

h = planck's constant in J·s

m_n = neutron mass in kg

periodictable.nsf.**neutron_scattering**(*compound*, *density=None*, *wavelength=None*, *energy=None*, *natural_density=None*, *table=None*)
Computes neutron scattering cross sections for molecules.

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 occuring abundances

***wavelength* 1.798 : float(s) | Å** Neutron wavelength (default=1.798 Å).

***energy* : float(s) | meV** Neutron energy. If energy is specified then wavelength is ignored.

***table* : PeriodicTable** Alternate table to use when parsing *compound*.

Returns

***sld* : float(s), float(s), float(s) | 10^{-6}Å^{-2}** (*real*, *-imaginary*, *incoherent*) scattering length density.

***xs* : float(s), float(s), float(s) | cm^{-1}** (*coherent*, *absorption*, *incoherent*) cross sections.

***penetration* : float(s) | cm** 1/e penetration depth of the beam

Returns (None, None, None) if sld is unknown for any component.

Raises *AssertionError* : density is missing.

Neutron scattering cross sections for materials are calculated from tabulated values for the different nuclei. The result is only an approximation. Actual scattering depends on details of sample composition, as well as the incoming neutron energy and sample temperature, especially for light elements. For low energy 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, leading to an estimate of 5.621/cm for H2O as computed compared to ~7.0/cm as measured with 5 meV neutrons at 290K.[7] The alignment of the neutron spin with the nuclei spin also matters, as demonstrated by ^{3}He polarizers.

The tables themselves are not self-consistent. Because the different quantities are measured indirectly with a variety of techniques, there are discrepencies when converting values from one column to another. These differences can be seen with the following:

`absorption_comparison_table()`

`coherent_comparison_table()`

`total_comparison_table()`

For our calculations we use the real part of the bound coherent scattering length $\mathrm{Re}(b_c)$ (labelled b_c in the tables) and the absorption cross section σ_a from which we derive the imaginary scattering length, $\mathrm{Im}(b_c)$. See Sears (1992) for details.[6]

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:

$$m = \sum n_k m_k \text{ for each atom } k = 1, 2, \ldots$$

Cell volume V (Å^3/molecule) is molar mass m over density ρ, with a correction based on Avogadro's number N_A (atoms/mol) and the length conversion 10^8 Å/cm:

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

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

$$N = \sum n_k \Big/ V$$

The coherent scattering length of the molecule is computed from the average scattering length $b_c = \mathrm{Re}(b_c) + i\mathrm{Im}(b_c)$ weighted by frequency:

$$\mathrm{Re}(b_c) = \sum n_k \mathrm{Re}(b_{ck}) \Big/ \sum n_k$$
$$\mathrm{Im}(b_c) = \sum n_k \mathrm{Im}(b_{ck}) \Big/ \sum n_k$$

[7] 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. doi:10.1107/S0021889882011285

[6] Sears, V.F. (1992) Neutron scattering lengths and cross sections. Neutron News 3, No. 3, 26-37.

The individual $\mathrm{Im}(b_{ck})$ values are derived from the absorption cross sections σ_a, tabulated at wavelength $\lambda = 1.798$ Å and scaled to fm (with 1000 fm = 1 barn/Å):

$$\mathrm{Im}(b_{ck}) = -\sigma_{ak}/(1000 \cdot 2\lambda)$$

Note the sign change relative to b'' in Sears (1992), with Eq 2 defining $b = b' - ib''$. Since we are not considering polarized nuclei, the imaginary incoherent contribution is zero and $b'' = -\mathrm{Im}(b_c)$.

Some rare earth isotopes are energy-dependent with complex bound coherent scattering length b_c tabulated by energy.[4] For the given input wavelength λ, b_c is interpolated from the table values, with the end points used for values outside the tabulated range.

The average scattering length is converted to scattering cross sections, with σ_c scaled to barn (with 1 barn= 100 fm^2) and σ_a scaled to barn (with 1000 barn = 1 fm Å):

$$\sigma_c = 4\pi|\mathrm{Re}(b_c) + i\mathrm{Im}(b_c)|^2/100$$
$$\sigma_a = -1000 \cdot 4\pi \langle \mathrm{Im}(b_c) \rangle / k \text{ for } k = 2\pi/\lambda$$

For most elements the scattering length is independent of energy in the thermal neutron energy range so the coherent scattering length σ_c is unchanged. The absorption cross section σ_a for these elements scales linearly with wavelength and can be adjusted with a simple multiplication:

$$\sigma'_a = \sigma_a \lambda'/\lambda_o = \sigma_a \lambda'/1.798$$

The incoherent scattering length is more complicated, including contributions from spin incoherence for different possible spin states as well as isotope incoherence from diffuse coherent scattering.[10] Using the total cross section σ_s from the table, the incoherent scattering length is estimated as:

$$\sigma_s = \sum n_k \sigma_{sk} \Big/ \sum n_k$$
$$\sigma_i = \sigma_s - \sigma_c$$
$$b_i = \sqrt{100\sigma_i/(4\pi)}$$

For the energy dependent rare earth isotopes the total scattering is estimated from $b = \mathrm{Re}(b_c) + i\mathrm{Im}(b_c)$, ignoring any spin incoherence effects. As a result, incoherent scattering for materials with energy-dependent rare earth isotopes with non-zero nuclear spin will be underestimated.

The scattering potential can be expressed as a scattering length density (SLD). This is the number density of the scatterers (per Å^3) times their scattering lengths, scaled to 10^6/Å^2 (with $1/$Å^2 = 10^5fm/Å^3). Following the convention of Sears (1992), we define sld as $\rho = \rho_{\mathrm{re}} - i\rho_{\mathrm{im}}$.

$$\rho_{\mathrm{re}}(10^6/\text{Å}^2) = 10N\mathrm{Re}(b_c)$$
$$\rho_{\mathrm{im}}(10^6/\text{Å}^2) = -10N\mathrm{Im}(b_c)$$
$$\rho_{\mathrm{inc}}(10^6/\text{Å}^2) = 10Nb_i$$

Similarly, the macroscopic scattering cross section for the sample includes number density:

$$\Sigma_{\mathrm{coh}}(1/\mathrm{cm}) = N\sigma_c$$
$$\Sigma_{\mathrm{inc}}(1/\mathrm{cm}) = N\sigma_i$$
$$\Sigma_{\mathrm{abs}}(1/\mathrm{cm}) = N\sigma_a$$
$$\Sigma_{\mathrm{s}}(1/\mathrm{cm}) = N\sigma_s$$

[4] 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. doi:10.1016/0092-640X(90)90013-A

[10] Glinka, C.J. (2011) Incoherent Neutron Scattering from Multi-element Materials. J. Appl. Cryst. 44, 618-624. doi: 10.1107/S0021889811008223

The 1/e penetration depth t_u represents the the depth into the sample at which the unscattered intensity is reduced by a factor of e:

$$t_u(cm) = 1/\left(\Sigma_\mathrm{s} + \Sigma_\mathrm{abs}\right)$$

The calculated penetration depth includes the effects of both absorption and incoherent scattering (which spreads the beam over the full 4π spherical surface, and so it looks like absorption with respect to the beam), as well as the coherent scattering from the sample. If you instead want to calculate the effective shielding of the sample, you should recalculate penetration depth with absorption only.

Transmission rate can be computed from e^{-d/t_u} for penetration depth t_u and sample thickness d. This does not include many real world effects, such as single phonon scattering[8] and forward scattering[7], which result in measured transmission significantly different from the values predicted from nuclear properties alone.

There is also a wavelength dependence for single phonon interactions which gives rise to significant inelastic scattering for lighter isotopes (H, D) and/or longer wavelengths (above 5 Å). This factor is both temperature and material dependent and will not be included in the scattering calculations. In particular, penetration length and transmitted flux are going to be significantly overestimated.

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

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

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

Computes neutron scattering length densities for molecules.

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 occuring abundances

wavelength **: float | Å** Neutron wavelength (default=1.798 Å).

energy **: float | meV** Neutron energy. If energy is specified then wavelength is ignored.

table **: PeriodicTable** Alternate table to use when parsing *compound*.

Returns

sld **: (float, float, float) | 10^{-6}Å^{-2}** (*real*, -*imaginary*, *incoherent*) scattering length density.

Raises *AssertionError* : density is missing.

[8] Mildner, D.F.R., Lamaze, G.P. (1998) Neutron Transmission of Single-Crystal Sapphire. J Appl Crystallogr 31, 835–840. doi:10.1107/S0021889898005846

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] | Å** Probe wavelength(s).

Returns *calculator* : f(w, density=1) -> (*real*, *-imaginary*, *incoherent*)

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. Note that density must be provided for each set of material weights. The density on the individual materials is ignored.

The returned slds will be vectors if the input wavelength is a vector and if any of the elements are energy dependent.

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:

$$\sigma_a = -2\lambda \mathrm{Im}(b_c) \cdot 1000$$

The wavelength $\lambda = 1.798$Å is the neutron wavelength at which the absorption is tallied. The factor of 1000 transforms from Å·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) # doctest: +ELLIPSIS, +NORMALIZE_
↪WHITESPACE
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. Use -tol to show elements within tolerance rather than those outside tolerance.

Returns None

Example

```
>>> coherent_comparison_table (tol=0.5) # doctest: +ELLIPSIS, +NORMALIZE_
↪WHITESPACE
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. Use -tol to show elements within tolerance rather than those outside tolerance.

Returns None

Example

```
>>> incoherent_comparison_table (tol=0.5) # doctest: +ELLIPSIS, +NORMALIZE_
↪WHITESPACE
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%
   ...
```

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. Use -tol to show elements within tolerance rather than those outside tolerance.

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)  # doctest: +ELLIPSIS, +NORMALIZE_WHITESPACE
 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   0.652 *
4-He      4.003   0.122   0.598   0.000   0.035
```

(continues on next page)

(continued from previous page)

```
   ...
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()* accepts dictionaries of {atom: count}.

2.12 X-ray scattering potentials

2.12.1 periodictable.xsf

This module has one class and nine fuctions.

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 scatting 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.

index_of_refraction() Express xray scattering length density as an index of refraction

mirror_reflectivity() X-ray reflectivity from a mirror made of a single compound.

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 elements beyond neon, with $K_\alpha = (2K_{\alpha 1} + K_{\alpha 2})/3$.

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.

Emission line tables

Data for the K_α and K_β lines comes from [#Deslattes2003], with the full tables available at http://www.nist.gov/pml/data/xraytrans/index.cfm. Experimental Values are used, truncated to 4 digits of precision to correspond to the values for the subset of elements previously defined in the periodictable package.

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[1]. 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.
>
> Note that the following elements have been updated since the publication of Ref.[1] in July 1993. Check http://henke.lbl.gov/optical_constants/update.html for more recent updates.

[1] 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).

Element	Updated	Energy Range (eV)
Mg	Jan 2011	10-1300
Zr	Apr 2010	20-1000
La	Jun 2007	14-440
Gd	Jun 2007	12-450
Sc	Apr 2006	50-1300
Ti	Aug 2004	20-150
Ru	Aug 2004	40-1300
W	Aug 2004	35-250
Mo	Aug 2004	25-60
Be	Aug 2004	40-250
Mo	Nov 1997	10-930
Fe	Oct 1995	600-800
Si	Jun 1995	30-500
Au	Jul 1994	2000-6500
Mg,Al,Si	Jan 1994	30-200
Li	Nov 1994	2000-30000

Data available at:

1. http://henke.lbl.gov/optical_constants/asf.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 $[0, 24\pi]$ | Å^{-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(*energy=None*, *wavelength=None*)

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(*wavelength=None*, *energy=None*)
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 $r_e N(f_1 + if_2)$. where r_e is the electron radius and N is the number density. The number density is $N = \rho_m / mN_A$, with mass density ρ_m molar mass m and Avogadro's number N_A.

The constants are available directly:

r_e = periodictable.xsf.electron_radius

N_A = periodictable.constants.avogadro_number

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

scattering_factors_units = ['', '']

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

sftable_units = ['eV', '', '']

sld_units = ['1e-6/Ang^2', '1e-6/Ang^2']

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**(*compound*, *density=None*, *natural_density=None*, *wavelength=None*, *energy=None*)

Compute xray scattering length densities for molecules.

Parameters

compound [Formula initializer] Chemical formula.

density [float | g·cm^{-3}] Mass density of the compound, or None for default.

natural_density [float | g·cm^{-3}] Mass density of the compound at naturally occurring isotope abundance.

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

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

Returns

sld [(float, float) | 10^{-6}Å^{-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 a dictionary 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() # doctest: +ELLIPSIS, +NORMALIZE_WHITESPACE
 El  Kalpha  Kbeta1
 Ne 14.6102 14.4522
 Na 11.9103 11.5752
 Mg  9.8902  9.5211
 Al  8.3402  7.9601
 Si  7.1263  6.7531
 ...
```

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() # doctest: +ELLIPSIS, +NORMALIZE_WHITESPACE
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  18.71   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

periodictable.xsf.**index_of_refraction**(*compound*, *density=None*, *natural_density=None*, *energy=None*, *wavelength=None*)

Calculates the index of refraction for a given compound

Parameters

compound [Formula initializer] Chemical formula.

density [float | g·cm^{-3}] Mass density of the compound, or None for default.

natural_density [float | g·cm^{-3}] Mass density of the compound at naturally occurring isotope abundance.

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

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

Returns

n [float or vector | unitless] index of refraction of the material at the given energy

Notes

Formula taken from http://xdb.lbl.gov (section 1.7) and checked against http://henke.lbl.gov/optical_constants/getdb2.html

periodictable.xsf.**mirror_reflectivity**(*compound*, *density=None*, *natural_density=None*, *energy=None*, *wavelength=None*, *angle=None*, *roughness=0*)

Calculates reflectivity of a thick mirror as function of energy and angle

Parameters

compound [Formula initializer] Chemical formula.

density [float | g·cm^{-3}] Mass density of the compound, or None for default.

natural_density [float | g·cm^{-3}] Mass density of the compound at naturally occurring isotope abundance.

roughness [float | Å] High-spatial-frequency surface roughness.

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

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

angle [vector | °] Incident beam angles.

Returns

reflectivity [matrix] matrix of reflectivity as function of (angle, energy)

Notes

Formula taken from http://xdb.lbl.gov (section 4.2) and checked against http://henke.lbl.gov/optical_constants/mirror2.html

2.13 X-ray scattering factor f0 calculations

2.13.1 periodictable.cromermann

Cromer-Mann formula for calculating x-ray scattering factors.

class periodictable.cromermann.**CromerMannFormula**(*symbol*, *a*, *b*, *c*)

Bases: object

Cromer-Mann formula for x-ray scattering factors. Coefficient storage and evaluation.

Class data:

stollimit [float | $Å^{-1}$] maximum sin(theta)/lambda for which the formula works

Attributes:

symbol [string] symbol of an element

a [[float]] a-coefficients

b [[float]] b-coefficients

c [float] c-coefficient

Create a new instance of CromerMannFormula for specified element.

No return value

atstol(*stol*)

Calculate x-ray scattering factors at specified sin(theta)/lambda

stol [float or [float] | $Å^{-1}$] sin(theta)/lambda

Return float or numpy.array.

stollimit = 6

periodictable.cromermann.**fxrayatq**(*symbol*, *Q*, *charge=None*)

Return x-ray scattering factors of an element at a given Q.

symbol [string] symbol of an element or ion, e.g., "Ca", "Ca2+"

Q [float or [float] | $Å^{-1}$] Q value

charge [int] ion charge, overrides any valence suffixes such as "-", "+", "3+".

Return float or numpy array.

periodictable.cromermann.**fxrayatstol**(*symbol*, *stol*, *charge=None*)
Calculate x-ray scattering factors at specified sin(theta)/lambda

symbol [string] symbol of an element or ion, e.g., "Ca", "Ca2+"

stol [float or [float] | Å^{-1}] sin(theta)/lambda

charge [int] ion charge, overrides any valence suffixes such as "-", "+", "3+".

Return float or numpy.array.

periodictable.cromermann.**getCMformula**(*symbol*)
Obtain Cromer-Mann formula and coefficients for a specified element.

symbol [string] symbol of an element

Return instance of CromerMannFormula.

2.14 Element plotter

2.14.1 periodictable.plot

Table plotter

periodictable.plot.**table_plot**(*data*, *form='line'*, *label=None*, *title=None*)
Plot periodic table data using element symbol vs. value.

Parameters

data [{ Element: float }] Data values to plot

***form* = "line"** ["linelgrid"] Table layout to use

Returns None

2.15 Utility functions

2.15.1 periodictable.util

Helper functions

periodictable.util.**cell_volume**(*a=None*, *b=None*, *c=None*, *alpha=None*, *beta=None*, *gamma=None*)
Compute cell volume from lattice parameters.

Parameters

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

alpha*, *beta*, *gamma [float | °] Lattice angles. *alpha* defaults to 90°. *beta* and *gamma* default to *alpha*.

Returns

V [float | Å^3] Cell volume

Raises *TypeError* : missing or invalid parameters

The following formula works for all lattice types:

$$V = abc\sqrt{1 - \cos^2\alpha - \cos^2\beta - \cos^2\gamma + 2\cos\alpha\cos\beta\cos\gamma}$$

periodictable.util.**require_keywords** (*function*)

Decorator which forces all keyword arguments to the function to be explicitly named.

For example:

```
>>> @require_keywords
... def fn(a, b, c=3): pass
>>> fn(1, 2, 3)
Traceback (most recent call last):
...
TypeError: name=value required for c
>>> fn(1, 2, c=6)
>>> fn(b=1, a=2, c=6)
```

Variable arguments are not currently supported:

```
>>> @require_keywords
... def fn(a, b, c=6, *args, **kw): pass
Traceback (most recent call last):
...
NotImplementedError: only named arguments for now
```

Note: The call signature is not preserved.

We can't preserve the function signature for the call since the only way we can count the number of non-keyword arguments is to use the *args, **kw call style. Python 3+ provides the '*' call signature element which will force all keywords after '*' to be named.

CHAPTER 3

Indices and Tables

- genindex
- modindex

Python Module Index

p

Index

E

F

G

H

I

J

K

T

V

X