xraydb

Release 4.5.7

Matthew Newville

2025-July-18

CONTENTS

XrayDB provides atomic data, characteristic X-ray energies, and X-ray cross sections for the elements in an SQLite3 database, `xraydb.sqlite`. This file can be used directly with SQLite [Hipp (2012)] or from the many programming language that have interfaces to SQLite. A Python module providing an interface to this database is also provided. Some of the components of the database hold arrays of numbers, which are stored as JSON-encoded strings, and will need to be decoded from JSON to be used.

The current version of the XrayDB database is **9.2**, and the Python module is version 4.5.7, which can be installed with:

```
pip install xraydb
```

The XrayDB Github Page has data sources, code, development discussions and issues.

Values in XrayDB use the most common SI units for X-ray work: Cross sections are in cm^2/gr, and energies are in eV. Energy-dependent data for cross-sections are typically most reliable between about 250 eV to about 250,000 eV. Elements from Z=1 to 92 are supported, with some data are included for elements between Z=93 and Z=98.

Some useful resources using this library include:

XrayDB Web App (SEES)	Web Application for calculations with XrayDB
XrayDB Web App (xrayabsorption.org)	Web Application for calculations with XrayDB
X-ray Periodic Table of the Elements	Printable Poster-sized Periodic Tables

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Installation

The X-ray database is held in the SQLite3 file `xraydb.sqlite`. If you are looking for direct use with SQLite, you can download this from here: xraydb.sqlite.

To install the XrayDB Python module (which includes the sqlite database), use:

```
pip install xraydb
```

Depending on your system and Python installation, you may need administrative privileges or to use *sudo* to install to a system-installed Python environment.

Note

The Python module supports Python 3.9 and above.

1.1.1 Development Version

To work with the data sources or to add or modify data in the XrayDB, you will want to clone or download the full source code kit xrayDB on github.com which contains the current database, original source data, python module, and files for generating the Periodic Table posters. To get the latest development version, use:

```
git clone https://github.com/xraypy/XrayDB.git
```

1.1.2 Testing

There are a set of tests scripts for the Python interface that can be run with the pytest testing framework. These are located in the `python/tests` folder. These tests are automatically run as part of the development process. For any release or any master branch from the git repository, running `pytest` should run all of these tests to completion without errors or failures.

1.1.3 Acknowledgements

The project began with the data from the compilation of basic atomic properties and X-ray absorption edge energies, emission energies, and absorption cross sections from [Elam, Ravel, and Sieber (2002)], who assembled data from a several sources. The original code to store this with SQLite was written by Darren S. Dale (see https://github.com/praxes/elam_physical_reference) from CHESS. More data has been added from other sources. Energy widths of core holes for excited electronic levels from [Keski-Rahkonen and Krause (1974)] and [Krause and Oliver (1979)]. Elastic X-ray scattering data, $f_0(q)$ is taken from [Waasmaier and Kirfel (1995)]. Resonant scattering cross sections $f'(E)$ and $f''(E)$ and absorption cross sections from [Chantler (2000)] as from https://www.nist.gov/pml/data/ffast/index.cfm,

but on a finer energy grid been provided directly by Christopher T. Chantler [Chantler (2016)]. Nathan Whittington from U Tennessee and Roman Chernikov from BNL provided the code to calculate reflectivity of multilayer and coated mirrors. Code for better calculation of sample thicknesses for XAFS transmission samples was provided by easyXAFS. Most of the remaining code was written and is maintained by Matt Newville, though with contributions from many others.

1.1.4 Citing this work

To cite this work, please use https://zenodo.org/badge/latestdoi/205441660

1.1.5 Copyright, Licensing, and Re-distribution

The original sources of the data included here are mostly based on published works with the clear intent of providing data to the general public. Some of the datasets here do not have clear statements of copyright or license, but have been freely available for many years. The work here is a compilation and reformatting of those datasets.

To the extent possible, and unless otherwise stated, the database files, data sources, and documentation files here are placed in the public domain, using the Creative Commons 1.0 Universal (CC0 1.0) Public Domain Dedication below.

As an important note, the Python code in the *xraydb* package is copyrighted and available under the terms of the MIT License.

```
Public Domain

The original sources of the data included here are mostly based on published
works with the clear intent of providing data to the general public.  Some of
the datasets here do not have clear statements of copyright or license, but
have been freely available for many years. The work here is a compilation and
reformatting of those datasets.

To the extent possible, and unless otherwise stated, the database files, data
sources, and documentation files here are placed in the public domain, using
the Creative Commons 1.0 Universal (CC0 1.0) Public Domain Dedication below.

In particular, the files named "xraydb.sqlite", "xraydb.schema", and all files
in the following folders and subfolders:
  data_sources/
  poster/
  doc/

are all placed in the "Public Domain" using the CC0 1.0 dedication.

The files in the folder 'python' and 'python/xraydb' are copyrighted
by the lead authors and copyrighted using an MIT License, which allows
for distribution and re-use of the source code with the only
restriction being to not remove the notice of copyright.  Each of
these files will have an explicit notice of copyright and license for
use.

For files that do not explicitly carry a notice copyrighted, no claim
of copyright is made and the CC0 1.0 dedication applies.

The Creative Commons 1.0 Universal (CC0 1.0) Public Domain Dedication
(https://creativecommons.org/publicdomain/zero/1.0/):

```

(continues on next page)

(continued from previous page)

```
The person who associated a work with this deed has dedicated the work to
the public domain by waiving all of his or her rights to the work worldwide
under copyright law, including all related and neighboring rights, to the
extent allowed by law.

You can copy, modify, distribute and perform the work, even for commercial
purposes, all without asking permission. See Other Information below.

In no way are the patent or trademark rights of any person affected by CC0,
nor are the rights that other persons may have in the work or in how the
work is used, such as publicity or privacy rights.

Unless expressly stated otherwise, the person who associated a work with
this deed makes no warranties about the work, and disclaims liability for
all uses of the work, to the fullest extent permitted by applicable law.

When using or citing the work, you should not imply endorsement by the
author or the affirmer.
```

1.2 X-ray Periodic Table of the Elements

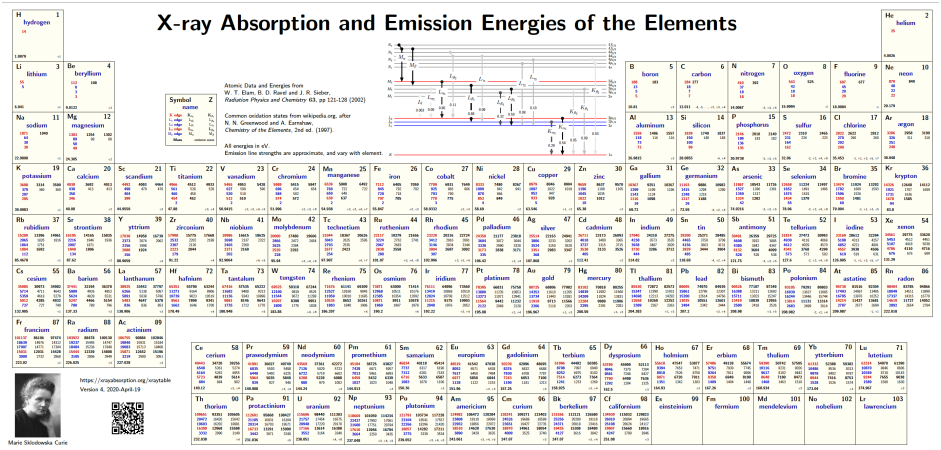
XrayDB has been used to generate X-ray Periodic Tables of the Elements. These are built using latex and the Python code and Makefile in poster folder.

There is a choice of two sizes, and a choice of an image of one of four prominent scientists associated with the Periodic Table and X-ray spectroscopies, or with SEES, the group supporting this work.

Table of X-ray Periodic Tables of the Elements The poster comes in two sizes: Large is 127.5x61 cm (about 50x24 inches) and Medium is 91x45.5 cm (about 36x17 inches).

Scientist	Large Periodic Table	Medium Periodic Table
Charles Barkla	Barkla (Large)	Barkla (Medium)
Marie Curie	Curie (Large)	Curie (Medium)
Dmitri Mendeleev	Mendeleev (Large)	Mendeleev (Medium)
Henry Moseley	Moseley (Large)	Moseley (Medium)
SEES	SEES (Large)	SEES (Medium)

These periodic tables will look like this:



but you should definitely download the PDFs linked for high-quality results.

1.3 Calculations of X-ray properties of materials

This section presents several detailed examples of using *xraydb.sqlite* to calculate X-ray properties of materials. All of the examples here us the functions in the python *xraydb* module, which is describe in more detail in the next chapter, *Using XrayDB from Python*. The examples will explore some aspects of X-ray physics, but will not give a complete tutorial on the concepts here. For a good reference on X-ray physics, see [Als-Nielsen and McMorrow (2011)].<

Many of these calculations are also available at XrayDB Web App (xrayabsorption.org).

1.3.1 X-ray attenuation by elements

The XrayDB database tabulates values of the X-ray mass attenuation coefficient, μ/ρ, for each element. In most of the X-ray regime used in materials characterization (say, up to 150 keV), the photo-electric effect is the main process that causes X-ray attenuation. When the photo-electric process is dominant, the values for μ/ρ depends strongly on Z of the atom and on X-ray energy E. In addition to these strong dependencies, sharp increases – so-called absorption edges – with be see at energies of bound core electron levels of atoms. To illustrate these characteristics, the following script will plot μ/ρ for selected elements:

```
#!/usr/bin/env python
# XrayDB example script python/examples/mu_elements.py
#
# plot X-ray mass attenuation for selected elements
#
import numpy as np
import matplotlib.pyplot as plt
import wxmplot.interactive as wi
from xraydb import mu_elam , atomic_symbol

energy = np.arange(500, 120000, 10)  # energy in eV

for elem in ('C', 'Cu', 'Au'):
    mu = mu_elam(elem, energy)
    plt.plot(energy, mu, label=elem, linewidth=2)

plt.title('X-ray mass attenuation')
plt.xlabel('Energy (eV)')
plt.ylabel(r'$\mu/\rho \rm\, (cm^2/gr)$')
```

(continues on next page)

(continued from previous page)

```
plt.legend()
plt.yscale('log')
plt.xscale('log')
plt.show()
```

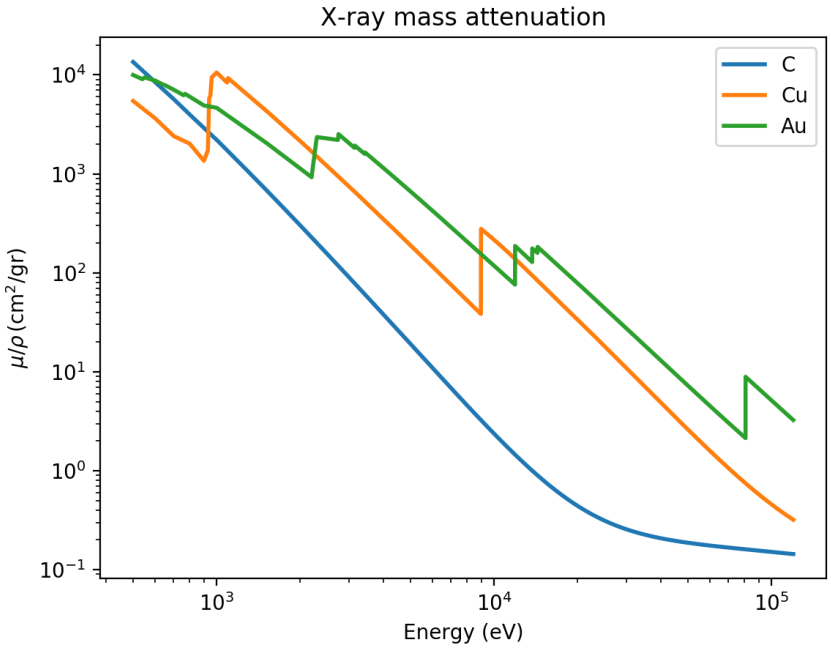


Fig. 1: X-ray mass attenuation coefficient for C, Cu, and Au.

As you can see in Figure from this figure, the attenuation drops very strongly with E – approximately as E^3. μ also depends strongly with Z, though the sharp absorption edges make this more complicated.

You can also observe that at relatively high energies for relatively low-Z elements (such as C above about 20 keV) that the attenuation levels off. This is because the coherent (Rayleigh) and incoherent (Compton) scattering processes dominate, so that the photo-electric absorption is no longer the dominant X-ray scattering process. This can be illustrated by plotting the different components of μ/ρ for C, as with the following script:

```
#!/usr/bin/env python
# XrayDB example script python/examples/mu_components_C.py
#
# plot components of X-ray mass attenuation for C
#
import numpy as np
import matplotlib.pyplot as plt
from xraydb import mu_elam

energy = np.arange(500, 120000, 10)  # energy in eV

elem = 'C'
mu_total = mu_elam(elem, energy, kind='total')
mu_photo = mu_elam(elem, energy, kind='photo')
```

(continues on next page)

(continued from previous page)

```
mu_incoh = mu_elam(elem, energy, kind='incoh')
mu_coher = mu_elam(elem, energy, kind='coh')

plt.title('X-ray mass attenuation for %s' % elem)
plt.plot(energy, mu_total, linewidth=2, label='Total')
plt.plot(energy, mu_photo, linewidth=2, label='Photo-electric')
plt.plot(energy, mu_incoh, linewidth=2, label='Incoherent')
plt.plot(energy, mu_coher, linewidth=2, label='Coherent')

plt.xlabel('Energy (eV)')
plt.ylabel(r'$\mu/\rho \rm\, (cm^2/gr)$')
plt.legend()
plt.yscale('log')
plt.show()
```

which will generate the following plot:

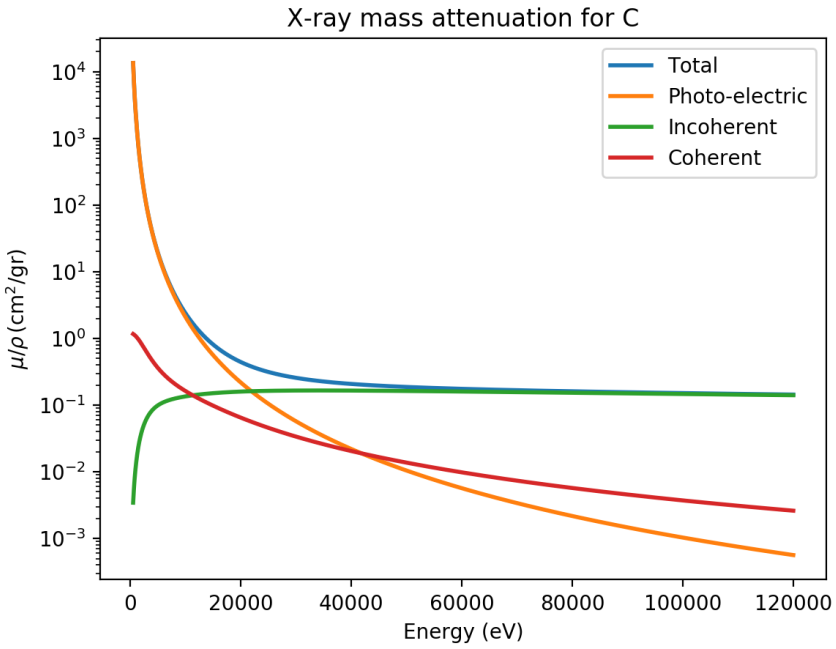


Fig. 2: X-ray scattering and attenuation factors for C.

Note that above 20 keV, the photo-electric absorption and incoherent Compton contributions are about equal, and that the Compton scattering dominates above 50 keV. As shown above, the photo-electric scattering will be much higher for heavier elements. The Rayleigh and Compton scattering have a much weaker dependence on Z, so that the photo-electric process dominates to higher energies. Replacing 'C' with 'Fe' in the script above will generate the following plot:

which shows that the Compton scattering reaching about 0.1 to 0.25 $\mathrm{cm}^2/\mathrm{gr}$ for Fe, about the same value as it was for C, while the photo-electric cross-section dominates past 100 keV.

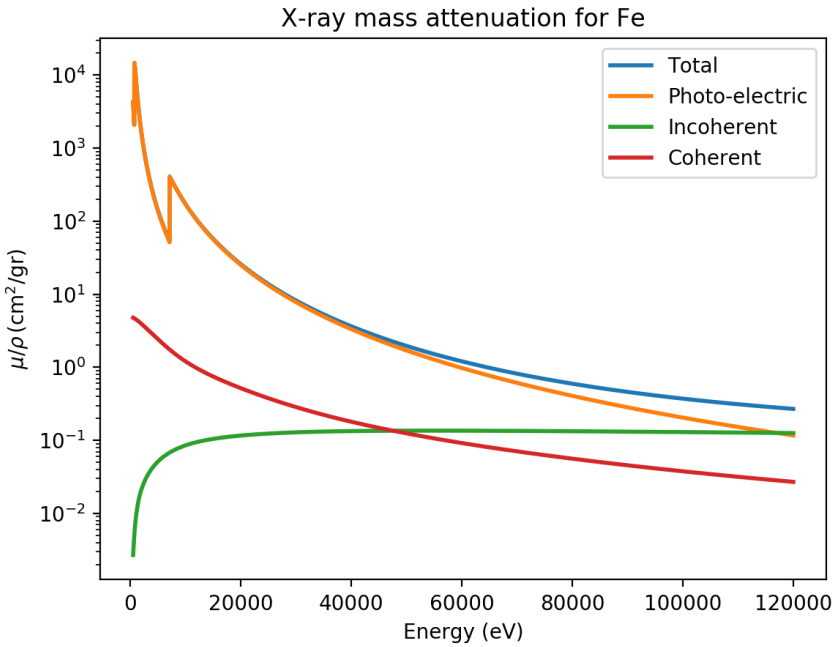


Fig. 3: X-ray scattering and attenuation factors for Fe.

1.3.2 μ calculations for materials

While one can use the above values for μ/ρ to calculate the attenuation of X-rays by multi-element materials, the *material_mu()* function is available to do the more convenient calculation of the X-ray absorption coefficient μ in units of 1/cm for a material and energy value and density (which are known for several common materials). This gives the length for which X-ray intensity is reduced by a factor of e, and so can be used to calculate the fraction of the X-rays transmitted through a material of known thickness, as $\exp(-t\mu)$ for a material of thickness t. As a first example, we calculate the the fraction of X-ray transmitted through 1 mm of the water as a function of X-ray energy:

```
#!/usr/bin/env python
# XrayDB example script    python/examples/mu_water.py
#
# calculate the fraction of X-rays transmitted through 1 mm of water
#
import numpy as np
import matplotlib.pyplot as plt

from xraydb import material_mu

energy = np.linspace(1000, 41000, 201)

mu = material_mu('H2O', energy)

# mu is returned in 1/cm
trans = np.exp(-0.1 * mu)

plt.plot(energy, trans, label='transmitted')
plt.plot(energy, 1-trans, label='attenuated')
```

(continues on next page)

(continued from previous page)

```
plt.title('X-ray absorption by 1 mm of water')
plt.xlabel('Energy (eV)')
plt.ylabel('Transmitted / Attenuated fraction')
plt.legend()
plt.show()
```

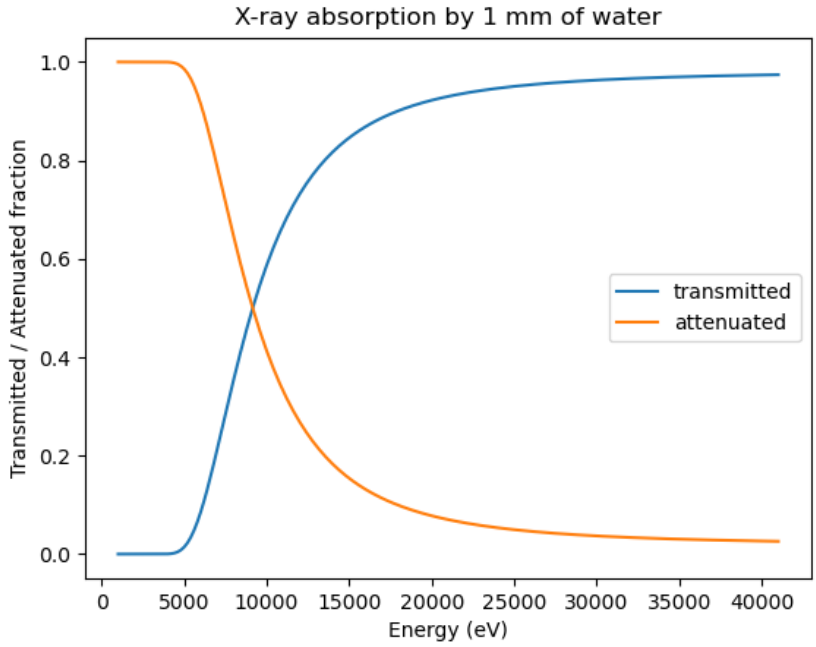


Fig. 4: Fraction of X-rays absorbed and transmitted by water

replacing:

```
mu = material_mu('H2O', energy)
```

with:

```
mu = material_mu('CaCO3', energy, density=2.71)
```

would generate the following plot

For many X-ray experiments, selecting the size of a material size so that its thickness is approximately 1 to 2 absorption length is convenient so that X-ray scattering and emission can be observed strongly, with neither all primary and scattered X-rays being absorbed by the material itself, but also not simply passing through the material without any interaction. For example, one can simply do:

```
>>> from xraydb import material_mu
>>> mu_20kev = xraydb.material_mu('CaCO3', 20000, density=2.71)
>>> print("CaCO3 1/e depth at 20keV = {:.3f} mm".format(10/mu_20kev))
CaCO3 1/e depth at 20keV = 0.648 mm
```

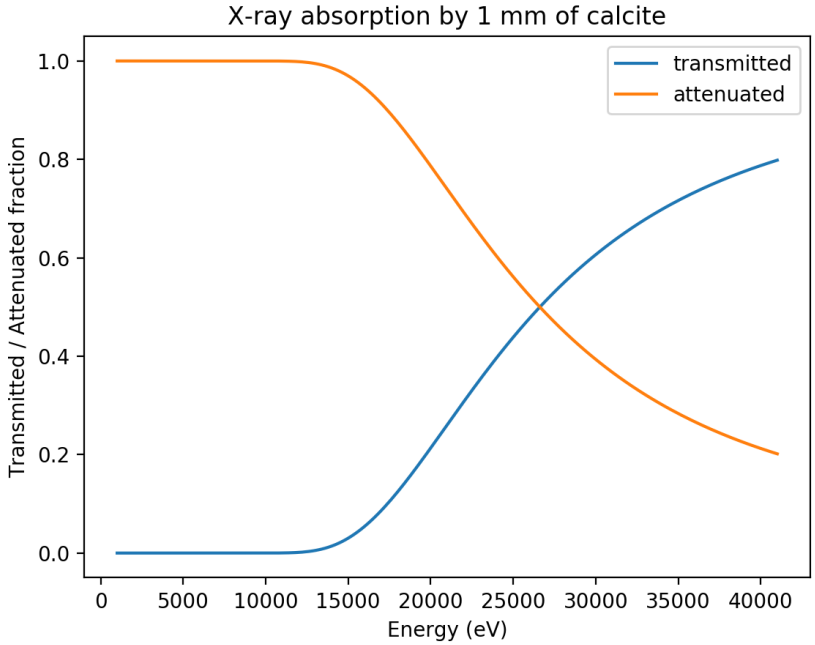


Fig. 5: Fraction of X-rays absorbed and transmitted by calcite

1.3.3 X-ray flux calculations for ionization chambers and photodiodes

Gas-filled ionization chambers are widely used as X-ray detectors. They are simple to use, inexpensive, and can give highly linear measures of photon flux over many orders of magnitude. X-rays entering a chamber filled with an inert gas (typically He, N2, or one of the noble gases, or a mixture of these) will be partially absorbed by the gas, with the strong energy dependence shown above. By adjusting the composition of the gas, nearly any fraction of the incident X-ray beam can be absorbed at a particular X-ray energy, making these ideal detectors to sample the intensity of an X-ray beam incident on a sample, while attenuating only a fraction of the beam.

Some of the X-rays in the gas will be absorbed by the photo-electric effect which will *ionize* the gas, generating free electrons and energetic ions. The first ionization event will generate an electron-ion pair with the energy of the X-ray minus the binding energy of the core electron. The high-energy electron and ion pair will further ionize other gas molecules. With an electric potential (typically on the order of 1 kV /cm) across the plates of the chamber, a current is generated that is proportional to the X-ray energy and fluence of the X-rays.

Effective Ionization Potentials of gases and semiconductors

The process of converting the X-ray generated current into X-ray fluence involves several steps. The energy from a single X-ray-generated electron is converted into a number of electron-ion pairs given by the *effective ionization potential* of the gas. These values are available from a few sources and range between 20 and 40 eV, given in the *Table of Effective Ionization Potentials*.

> Table of Effective Ionization Potentials. Many of these are taken from [Knoll (2010)], while others appear to come from International Commission on Radiation Units & Measurement, Report 31, 1979. The names given are those supported by the functions `ionization_potential()` and `ionchamber_fluxes()`.

gas/material name(s)	potential (eV)
hydrogen, H	36.5
helium, He	41.3
nitrogen, N, N2	34.8
oxygen, O, O2	30.8
neon, Ne	35.4
argon, Ar	26.4
krypton, Kr	24.4
xenon, Xe	22.1
air	33.8
methane, CH4	27.3
carbondioxide, CO2	33.0
silicon, Si	3.68
germanium, Ge	2.97

From this table, we can see that the absorption (by photo-electric effect) of 1 X-ray with energy 10 keV will generate about 300 electron-ion pairs. That is not much current, but if 10^8 Hz X-rays are absorbed per second, then the current generated will be around 5 nA. Of course, the length of the gas or more precisely the length of gas under ionizing potential will have an impact on how much current is generated. The photo-current generated can be amplified and converted to a voltage using a current amplifier, and that voltage will then recorded by a number of possible mean: a voltage-to-frequency generator and a digital counter is a common method for integrated current for a specific amount of time, but other sampling methods can also be used.

An ion chamber can be linear over many orders of magnitude of X-ray flux, provided the potential between the plates is high enough - typically in the 1 kV/cm range to efficiently collect all the charged particles before the recombine. As an important practical note, a typical current amplifier at a particular setting of sensitivity will be linear only over a limited range (often over an output voltage of 0.02 to 5 V). Because of this, the sensitivity of the current amplifier used with an ion chamber needs careful attention to avoid saturation and maintain sensitivity.

A photo-diode works in much the same way as an ionization chamber. X-rays incident on the diode (typically Si or Ge) will be absorbed and generate a photo-current that can be collected. Typically PIN diodes are used, and a small reverse bias voltage is often applied. Because the electrons do not need to escape the material but generate a current transported in the semiconductor, the effective ionization potential is much lower - a few times the semiconductor band gap instead of a few time the lowest core-level ionization potential. The current generated per X-ray will therefore be larger than for an ion chamber, and will also generally have a much faster response time. The generated current will still measured in the same manner as a gas-filled ionization typically using a current amplifier and integrating counter. Of course, the thickness of the diode is difficult to adjust. The active length of diodes are typically a few hundred microns, and so are generally much more absorbing than an ion chamber.

Compton scattering and Ion Chamber Current

In addition to photo-electric absorption, X-rays can be attenuated by gas molecules in an ion chamber by incoherent (Compton) or coherent (Rayleigh) scattering processes. The coherent scattering will not generate any electrons in the gas, but will elastically scatter X-rays out of the main beam. On the other hand, incoherent scattering will generate some current, though typically only a small portion of the incident X-ray energy is given to a scattered electron. In fact, Compton scattering has a distribution of energies given to the scattered electron depending on the angle of scattering, so that the energy of the scattered electron is

$$E_e = E_\gamma - E_\gamma/[1 + E_\gamma/(m_e c^2)(1 - \cos\theta)]$$

where E_γ is the incident X-ray energy, and θ is the scattering angle. From this, it easy to estimate the median energy of electrons generated by Compton scattering X-rays of energy E at 90 degrees will be

$$E_{\rm median} = E_\gamma/(1 + m_e c^2/E_\gamma)$$

(recall that $1 - 1/(1+x) = 1/(1+1/x)$). For X-rays of 10 keV, $E_{\rm median}$ is about 192 eV. For 20 keV X-rays, it will be 750 eV, and for 50 keV X-rays, it will be 4.5 keV. Because of the angular distribution of Compton scattering is not uniform, these median values over-estimate the amount of energy transferred to the scattered electron by a small amount that increases with energy. The mean energy of the Compton-scattered electron can be found by integrating the Klein-Nishina distribution. Since these values depend only on the incident X-ray energy, these calculations have been done and the values tabulated in the *Compton_energies* table in the XrayDB sqlite database.

Although the energy transferred to the electron by Compton scattering is much less than by the photo-electric process the contribution can be important. This is especially true for low-Z gas molecules such as He and N2 at relatively high energies (10 keV and above) for which incoherent scattering becomes much more important than photo-electric absorption, as shown above for C. That is, for accurate estimates of fluxes from ion chamber currents at energies about 20 keV or so, the contribution from Compton scattering should be included. For photo-diodes (typically made of Si), the Compton scattering cross-section exceeds the photo-electric cross-section about 56 keV, and so should also be included for high-energy X-ray measurements.

Ion Chamber Flux calculations

The conversion of incident flux at a particular energy to generated current is not too difficult if considering only the photo-electric effect of a single gas, but can be somewhat subtle in the more general case. For the discussion here, we assume that the potential across the plates of the ion chamber is high enough to prevent any recombination of charged particles.

For a given gas at an incident X-ray energy E, we calculate the total, photo-electric, incoherent (Compton), and coherent (Rayleigh) values of μ. If more than one gas is used, the weighted sum is calculated, so that we have $\mu_{\rm total}$, $\mu_{\rm photo}$, $\mu_{\rm incoh}$, and $\mu_{\rm coh}$ for the gas in the chamber or diode material.

The flux transmitted out of the chamber is

$$I_{\rm trans} = I_0 e^{-t\mu_{\rm total}}$$

where t is the length of the chamber and I_0 is the incident flux. These two intensities are the quantity we are most interested in. The attenuated flux (in number per second, or Hz) is

$$I_{\rm atten} = I_0(1 - e^{-t\mu_{\rm total}})$$

can be separated into the various source of attenuation as

$$\begin{aligned} I_{\rm photo} &= I_{\rm atten}\, \mu_{\rm photo}/\mu_{\rm total} \\ I_{\rm incoh} &= I_{\rm atten}\, \mu_{\rm incoh}/\mu_{\rm total} \\ I_{\rm coh} &= I_{\rm atten}\, \mu_{\rm coh}/\mu_{\rm total} \end{aligned}$$

The photo-electric effect converts all of the X-ray energy into a current of both electrons and ions using the effective ionization potential above:

$$C_{\rm photo} = 2 q_e E I_{\rm photo}/V_{\rm eff}$$

where q_e is the electron charge (1.6×10^{-19}C), E is the incident X-ray energy (in eV), $I_{\rm photo}$ is the flux (in Hz), and $V_{\rm eff}$ is the effective ionization potential for the gas. The leading 2 comes because both electrons and ions are typically counted for the current from an ion chamber. It is sometimes useful to add a Frisch mesh grid to collect the slower ions and shunt them so as to not count that portion of the current, and thereby give the ion chamber a faster time response. In that case, the current will be half of the value given above.

As discussed above, the coherent (Rayleigh) scattering produces no electrons, but the incoherent (Compton) scattering does, and the energy of the the Compton-scattered electron varies with both X-ray energy and scattering angle, as does

the probability of scattering. Integrating over all angles (and assuming the ion chamber is large enough to stop the scattered electrons) gives the mean electron energy, which we use to obtain the current from the incoherent scattering:

$$C_{\rm incoh} = 2q_e E_{\rm mean} I_{\rm incoh}/V_{\rm eff}$$

where $E_{\rm mean}$ is the mean energy of Compton-scattered electron (approximately, but slightly less than the $E_{\rm median}$ value above.

The current from an ion_chamber is typically measured as a voltage generated by a current-to-voltage amplifier. The measured voltage will have a gain or sensitivity in units of *A/V*. The goal is typically to calculate the flux I_0 from the measured voltage and knowledge of the sensitivity as well as the gas(es), ion chamber length t, and X-ray energy E. The measured voltage is given by

$$V = I_0(C_{\rm photo} + C_{\rm incoh})/S$$

where S is the amplifier sensitivity in A/V. From this, I_0 and $I_{\rm trans}$ can be calculated.

ionchamber_fluxes()

The function *ionchamber_fluxes()* will calculate X-ray fluxes for an ion chamber as described above the following inputs:

- *gas*: the gas, or mixture of gases used or 'Si' or 'Ge' for diodes.
- *length*: the length of the ion chamber, in cm.
- *energy*: the X-ray energy, in eV.
- *volts*: the output voltage of the current amplifier
- *sensitivity* and *sensitivity_units*: the sensitivity or gain of the amplifier used to convert the photo-current to the recorded voltage.
- *with_compton*: whether to include the current generated by Compton-scattered electrons [True]
- *both_carriers*: whether to include the current generated by both positive and negative charged particles [True]

The default *sensitivity_units* is 'A/V' but can be set to any of the common SI prefixes such as 'p', 'pico', 'n', 'nano', μ, (unicode '03bc'), 'u', 'micro', 'm', or 'milli', so that:

```
>>> fluxes = ionchamber_fluxes('N2', volts=1, energy=10000, length=10,
                               sensitivity=1.e-9)
>>> fluxes = ionchamber_fluxes('N2', volts=1, energy=10000, length=10,
                               sensitivity=1, sensitivity_units='nA/V')
```

will give the same results.

The output from *ionchamber_fluxes()* is a named tuple with 4 fields:

- *photo* - the flux absorbed by the photo-electric effect, in Hz.
- *incoherent* - the flux scattered by the Compton effects, in Hz.
- *incident* - the flux incident on the ion chamber, in Hz.
- *transmitted* - the flux beam leaving the ion chamber, in Hz.

As described above, the current in the ion chamber or photo-diode is generated by electrons and ions produced by both the photo-electric and incoherent or Compton scattering. The photo-electric cross-section will dominate for heavy elements and relatively low X-ray energies, but does not necessarily dominate at high X-ray energies. The photo-electric cross-section with the incident X-ray energy and the incoherent cross-section with the ***mean*** Compton-scattering energy, using the calculated and tabulated mean energies of the Compton-scattered electrons are used to estimate the

incident flux from the photo-current. The total attenuation cross-section, including the coherent cross-sections, is used to calculate the transmitted flu from the incident flux.

As an example calculation of ion chamber currents:

```
>>> fl = ionchamber_fluxes(gas='nitrogen', volts=1.25, energy=18000,
                           length=10.0, sensitivity=1.e-6)
>>> print(f"Incident= {fl.incident:g} Hz, Transmitted flux= {fl.transmitted:g} Hz")
Incident= 2.2358e+12 Hz, Transmitted flux= 2.214e+12 Hz
```

It is not uncommon for an ion chamber to be filled with a mixture of 2 or more gases so as to better control the fraction of X-rays absorbed in a chamber of fixed length. This can be specified by passing in a dictionary of gas name and fractional density, as with:

```
>>> fl = ionchamber_fluxes(gas={'Kr':0.5, 'Ar': 0.5}, volts=1.25,
                           energy=18000, length=10,
                           sensitivity=1, sensitivity_units='microA/V')
>>> print(f"Incident= {fl.incident:g} Hz, Transmitted flux= {fl.transmitted:g} Hz")
Incident= 1.43737e+10 Hz, Transmitted flux= 3.28986e+09 Hz
```

Finally, the pressure of the gas is sometimes adjusted to alter the fraction of the beam absorbed. The calculations here all use the densities at STP, but changes in gas density will be exactly linear to changing the length of the ion chamber.

1.3.4 X-ray mirror reflectivities

At very shallow angles of incidence X-rays can be reflected by total external reflection from a material. The reflectivity can be very high at relatively low energies and shallow angles, but drops off dramatically with increasing energy, increasing angle, and decreasing electron density. Still, this reflectivity is one of the few ways to steer X-ray beams and so is widely used in synchrotron radiation sources.

Single Layer mirrors

The reflectivity can be calculated with the *mirror_reflectivity()* function which takes X-ray energy, incident angle, and mirror material as arguments.

An example script, comparing the energy-dependence of the reflectivity for a few common mirror materials is given as

```
import numpy as np
from xraydb import mirror_reflectivity
import matplotlib.pyplot as plt

energy = np.linspace(1000, 51000, 501)

r_si = mirror_reflectivity('Si', 0.002, energy)
r_ni = mirror_reflectivity('Ni', 0.002, energy)
r_rh = mirror_reflectivity('Rh', 0.002, energy)
r_pt = mirror_reflectivity('Pt', 0.002, energy)

plt.plot(energy, r_si, label='Si')
plt.plot(energy, r_ni, label='Ni')
plt.plot(energy, r_rh, label='Rh')
plt.plot(energy, r_pt, label='Pt')

plt.title('X-ray reflectivity at $\\theta=2 \\mathrm{mrad}$')
plt.xlabel('Energy (eV)')
```

(continues on next page)

(continued from previous page)

```
plt.ylabel('Reflectivity')
plt.legend()
plt.show()
```

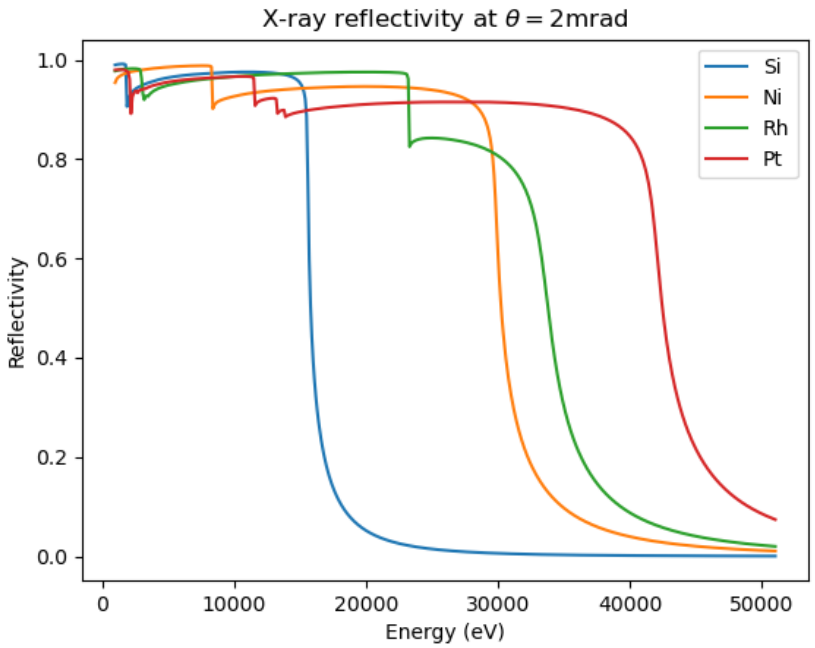


Fig. 6: X-ray mirror reflectivity at $\theta = 2\mathrm{mrad}$ for selected mirror surfaces and coatings used for mirrors.

Multilayer Mirrors

For more complex mirrors, such as those made of multiple materials, the reflectivity can be calculated with Parratt's Recursive Method using the *multilayer_reflectivity()* function. This function takes a list of materials from top to bottom, a matching list of thicknesses of each layer, substrate material, incident angle, energy, and the number of periods.

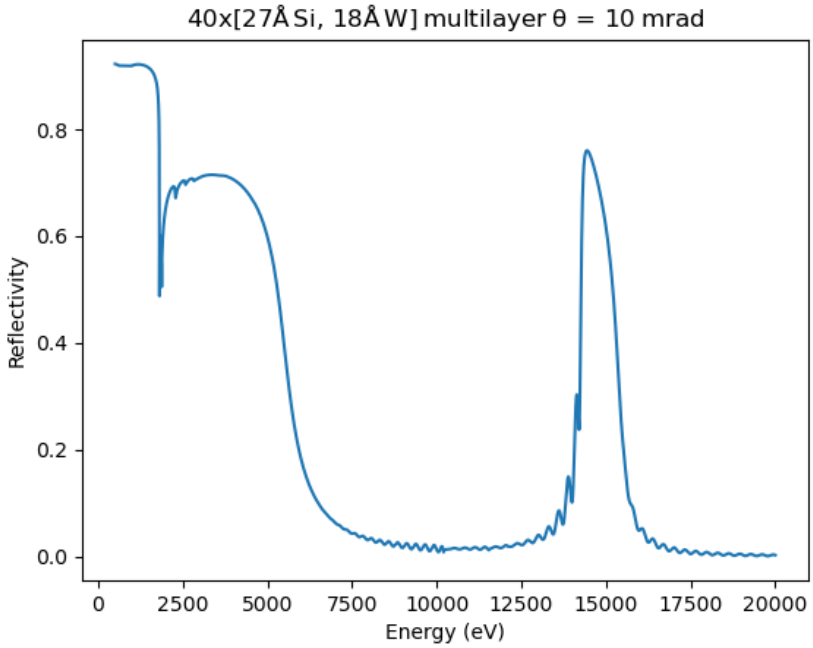
An example usage, calculating the reflectivity at 1000 eV vs. grazing angle for a 40*[Si, W] layered mirror on SiO2 substrate is given as

```
import numpy as np
from xraydb import multilayer_reflectivity
import matplotlib.pyplot as plt

# 40 layers of [27Ang Si, 18Ang W]
materials = ['Si', 'W']
thicknesses = [27, 18] # angstroms
n_periods = 40
substrate = 'SiO2'
theta = 0.01
energy = np.linspace(500, 20000, 5000)

reflect = multilayer_reflectivity(materials, thicknesses, substrate,
```

(continues on next page)

(continued from previous page)

```
                                    theta, energy, n_periods=n_periods)

plt.plot(energy, reflect)
plt.title(f'$\\rm 40 x [27\\AA\\, Si,\\, 18\\AA\\, W] \\> multilayer \\> \\theta\\, = \\,
↪ {theta*1000:.0f}\\> mrad$')
plt.xlabel('Energy (eV)')
plt.ylabel('Reflectivity')
plt.show()
```



Coated mirrors

In principle, reflectivity calculation for coated mirrors is the same as for multilayer mirrors; however, `coated_reflectivity`() is provided to simplify the recursion and have more clear input parameters. The function takes top layer material and thickness, substrate material and thickness, incident angle, energy, and optional binding layer parameters.

An example script calculating the reflectivity at $\theta = 4\mathrm{mrad}$ for a rhodium slab and rhodium coated mirror is given as

```
import numpy as np
from xraydb import (coated_reflectivity, mirror_reflectivity)
import matplotlib.pyplot as plt

coating = 'Rh'
coating_thick = 300
substrate = 'Si'
theta = 0.004
energy = np.linspace(10000, 35000, 500)
binder = 'Cr'
binder_thick = 30
```

(continues on next page)

(continued from previous page)

```
refl_coat = coated_reflectivity(coating, coating_thick, substrate, theta, energy,␣
↪binder=binder, binder_thick=binder_thick)
refl_rhod = mirror_reflectivity('Rh', theta, energy)

title = f'$\\rm {coating_thick} \\AA \\> {coating}/{binder_thick} \\AA \\> {binder} \\>␣
↪coated \\> {substrate} \\> mirror \\> \\theta\\, =\\, {theta*1000:.0f} mrad $'
plt.plot(energy, refl_coat, label='Rh coated')
plt.plot(energy, refl_rhod, label='Rh bulk')
plt.title(title)
plt.xlabel('Energy (eV)')
plt.ylabel('Reflectivity')
plt.yscale('log')
plt.legend()
plt.show()
```

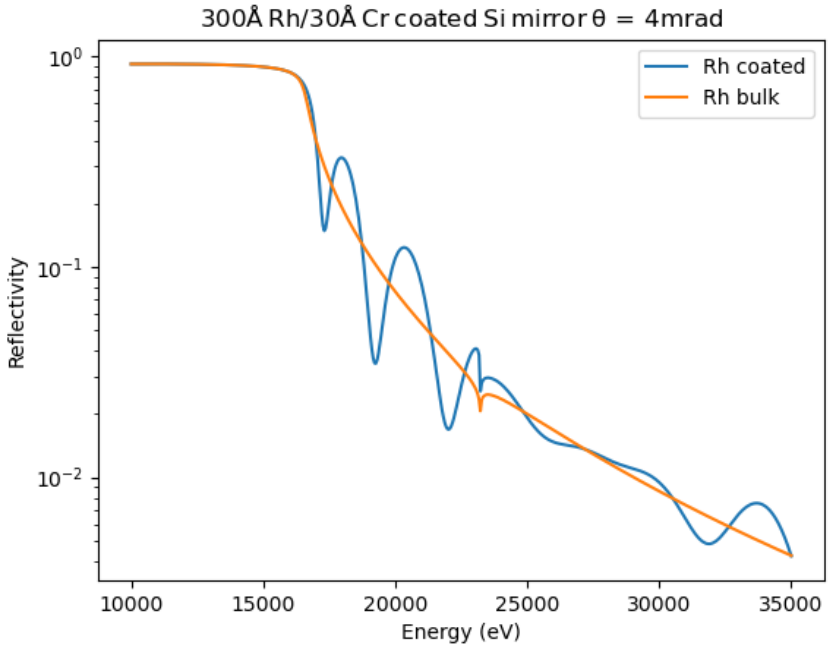


Fig. 7: X-ray reflectivity of a rhodium on silicon mirror with a 3 chromium binding layer compared to a pure slab of rhodium.

1.3.5 Darwin widths of monochromator crystals

Bragg's law describes X-ray diffraction from crystals as

$$m\lambda = 2d\sin(\theta)$$

where λ is the X-ray wavelength, d the d-spacing of the crystal lattice plane, θ the incident angle, and m the order of the reflection. For imperfect crystals, in which the lattice planes are not stacked perfectly over extended distances, the angular width of any particular reflection is dominated by the spread in d-spacing and the mosaicity inherent in the crystal. For perfect crystals, however, the angular width of a reflection is dominated by the fact that effectively all of the X-rays will scatter from the lattice well before any attenuation of the X-ray beam occurs. This *dynamical*

diffraction gives a small but finite offset from the Bragg angle, and gives a broadened angular width to reflection. This is usually called the Darwin width (named for Charles G. Darwin, grandson of the more famous Charles R. Darwin). In addition, the refraction and in particular the absorption effects that give anomalous scattering (as calculated with `xray_delta_beta()`) make the "rocking curve" of reflected intensity as a function of angle an asymmetric shape.

All of these effects are included in the `darwin_width()` function, which follows very closely the description from chapter 6.4 in [Als-Nielsen and McMorrow (2011)]. The function takes inputs of

- *energy*: the X-ray energy, in eV.
- *crystal*: the atomic symbol for the crystal: 'Si', 'Ge', or 'C'. ['Si']
- *hkl*: a tuple with (h, k, l) of the reflection used. [(1, 1, 1)]
- *a*: lattice constant [*None* - use nominal value for crystal]
- *polarization*: *s*, *p*, or *u* to specify the X-ray polarization relative to the crystal [*s*]
- *m*: the order of the reflection. [1]
- *ignore_f1*: whether to ignore *f1*. [False]
- *ignore_f2*: whether to ignore *f2*. [False]

Polarization of *s* should be used for vertically deflecting monochromators at most synchrotron sources (which will normally be horizontally polarized), and *p* should be used for horizontally deflecting monochromators. For crystals used to analyzed unpolarized X-ray emission, use *u*, which will give the average of *s* and *p* polarization.

As with `ionchamber_fluxes()`, the output here is complicated enough that it is put into a named *DarwinWidth* tuple that will contain the following fields:

- *theta* - the nominal Bragg angle, in rad
- *theta_offset* - the offset from the nominal Bragg angle, in rad.
- *theta_width* - estimated angular Darwin width, in rad
- *theta_fwhm* - estimated FWHM of the angular reflectivity curve, in rad
- *rocking_theta_fwhm* - estimated FWHM of a rocking curve, in rad
- *energy_width* - estimated energy Darwin width, in eV
- *energy_fwhm* - estimated FWHM energy reflectivity curve, in eV
- *rocking_energy_fwhm* - estimated FWHM of a rocking curve, in eV
- *zeta* - nd-array of $\zeta = \Delta\lambda/\lambda$.
- *dtheta* - nd-array of angles around from Bragg angle, in rad
- *denergy* - nd-array of energies around from Bragg energy, in eV
- *intensity* - nd-array of reflected intensity at *zeta* values.
- *rocking_curve* - nd-array of rocking curve of 2 crystals

Here, *dtheta* will be given by $\Delta\theta = \zeta\tan(\theta)$, and *denergy* will be given by $\Delta E = \zeta E$. Note that the *rocking_* quantities are estimated as the convolution of the intensity with itself, to simulate an angular scan of one crystal with respect to the other, as is often done with double-crystal monochromators. *rocking_theta_fwhm* and *rocking_energy_fwhm* will be the FWHM of this curve in angle and energy, and will typically be ~1.5x the Darwin widths in *theta_width* and *energy_width*, respectively.

All of the nd-arrays will be the same size, so that plots of reflectivity can be readily made. An example usage, printing the predicted energy and angular widths is

```
import numpy as np
from xraydb import darwin_width
import matplotlib.pyplot as plt

dw_si111 = darwin_width(10000, 'Si', (1, 1, 1))
dw_si333 = darwin_width(30000, 'Si', (3, 3, 3))

print(dw_si111.theta_width*1e6)

fmt_string = "Darwin Width for {:s} at {:.0f} keV: {:5.2f} microrad, {:5.2f} eV"
print(fmt_string.format('Si(111)', 10,
                        dw_si111.theta_width*1e6,
                        dw_si111.energy_width))


print(fmt_string.format('Si(333)', 30,
                        dw_si333.theta_width*1e6,
                        dw_si333.energy_width))

dtheta  = dw_si111.dtheta*1e6
denergy = dw_si111.denergy[::-1]

#  slightly advanced matplotlib hackery:
fig, ax = plt.subplots(constrained_layout=True)


ax.plot(dtheta, dw_si111.intensity, label='$I$, Si(111)', linewidth=2)
ax.plot(dtheta, dw_si111.intensity**2, label='$I^2$, Si(111)', linewidth=2)
ax.plot(dw_si333.dtheta*1e6, dw_si333.intensity**2, label='$I^2$ Si(333) 30 keV',␣
↪linewidth=2)


ax.set_title('X-ray diffraction intensity at 10keV')
ax.set_xlabel('Angle - $\\theta_B$ ($ \\mu \\mathrm{rad}$)')
ax.set_ylabel('Reflectivity')
ax.legend()

plt.show()
```

which will print out values of:

```
Darwin Width for Si(111) at 10 keV: 26.96 microrad,  1.34 eV
Darwin Width for Si(333) at 30 keV:  1.81 microrad,  0.27 eV
```

and generates a plot of

Note that the values reported for *theta_fwhm* and *energy_fwhm* will be about 6% larger than the reported values for *theta_width* and *energy_width*. The *width* values closely follow the region of the curve where the reflectivity ignoring absorption would be 1 - the flat top of the curve. Since a double-crystal monochromator will suppress the tails of the reflectivity, this smaller value is the one typically reported as "the Darwin width", though some sources will report this smaller value as "FWHM".

An example of the intensity of a single reflection and the "rocking curve" of two crystals is given in

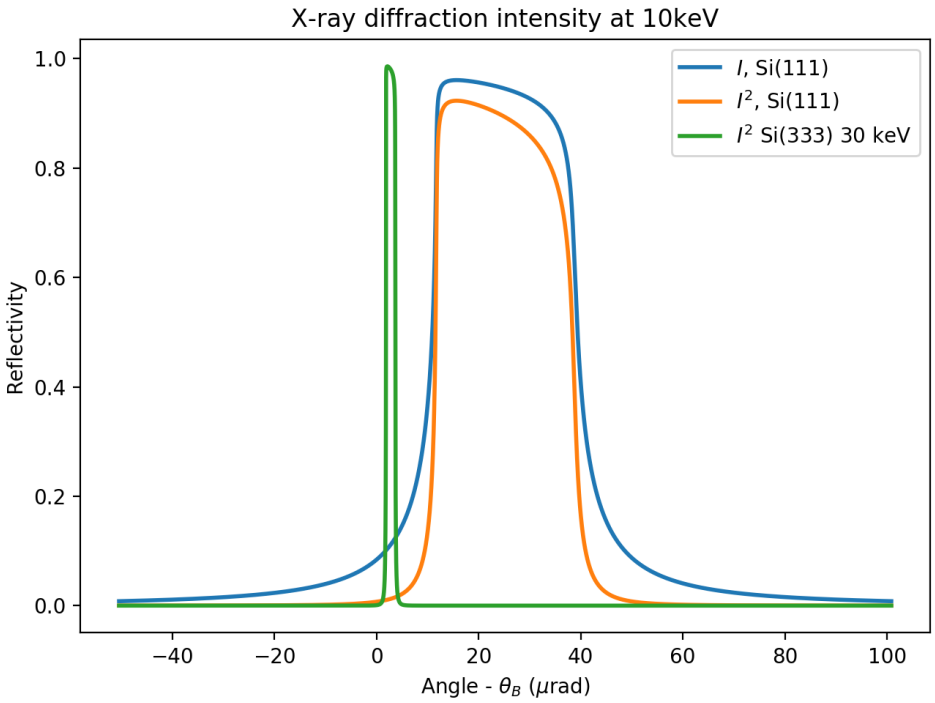


Fig. 8: X-ray monochromator diffracted intensities around the Si(111) reflection. Here, i represents the intensity of a single reflection, and i^2 the intensity from 2 bounces, as for a double-crystal monochromator. The intensity and angular offset of the third harmonic is also shown.

```
import numpy as np
from xraydb import darwin_width
import matplotlib.pyplot as plt

dw_si111 = darwin_width(10000, 'Si', (1, 1, 1))

dtheta  = dw_si111.dtheta*1e6

fig, ax = plt.subplots(constrained_layout=True)

ax.plot(dtheta, dw_si111.intensity, label='$I$, 1 crystal', linewidth=2)
ax.plot(dtheta, dw_si111.rocking_curve, label='Rocking Curve', linewidth=2)


ax.set_title('X-ray Rocking Curve at 10keV, Si(111)')
ax.set_xlabel('Angle - $\\theta_B$ ($ \\mu \mathrm{rad}$)')
ax.set_ylabel('Reflectivity')
ax.legend()

plt.show()
```

which generates a plot of

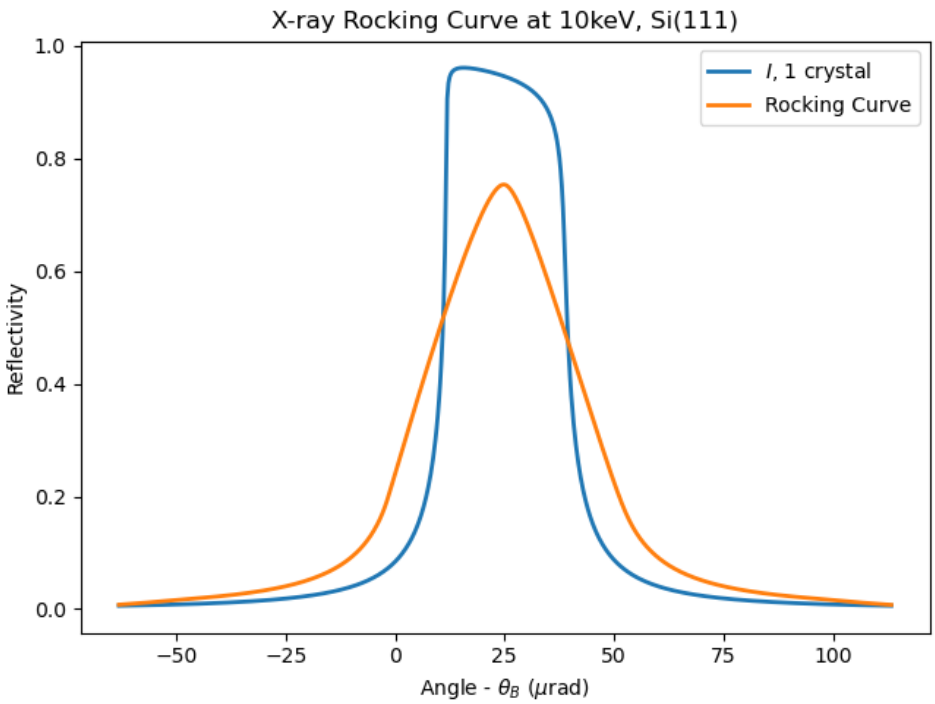


Fig. 9: X-ray monochromator rocking curve around the Si(111) reflection. Here, the blue curve shows the intensity of a single reflection, as above, while the orange curve shows the intensity from rocking one crystal through the reflection of the other.

1.4 Using XrayDB from Python

The *xraydb* Python module gives a user-friendly wrapping of the XrayDB, and automates the the conversion of data from sqlite database into Python and numpy arrays. The module requires the *numpy*, *scipy* and *sqlalchemy* modules, all of which are readily available and can be installed with:

```
pip install xraydb
```

The current version of the Python module is 4.5.7, corresponding to version 6 of *xraydb.sqlite*. The source code for this module is in the *python* folder from the GitHub repository.

1.4.1 The Python `xraydb` module

The *xraydb* module gives a low-level interface to the data in the *xraydb.sqlite* database. To use this import the *xraydb* module with:

```
>>> import xraydb
>>> xraydb.atomic_number('Ag')
47
#
# X-ray elastic (Thomson) scattering factors:
>>> import numpy as np
>>> q =np.linspace(0, 5, 11)
>>> xraydb.f0('Fe', q)
array([25.994603  , 11.50848469,  6.55945765,  4.71039413,  3.21048827,
       2.20939146,  1.65112769,  1.36705887,  1.21133507,  1.10155689,
```

(continues on next page)

(continued from previous page)

```
       1.0035555 ])
#
# X-ray emission lines:
>>> for name, line in xraydb.xray_lines('Zn', 'K').items():
...     print(name, ' = ', line)
...
Ka3  =  XrayLine(energy=8462.8, intensity=0.000316256, initial_level='K', final_level='L1
↪')
Ka2  =  XrayLine(energy=8614.1, intensity=0.294353, initial_level='K', final_level='L2')
Ka1  =  XrayLine(energy=8637.2, intensity=0.576058, initial_level='K', final_level='L3')
Kb3  =  XrayLine(energy=9567.6, intensity=0.0438347, initial_level='K', final_level='M2')
Kb1  =  XrayLine(energy=9570.4, intensity=0.0846229, initial_level='K', final_level='M3')
Kb5  =  XrayLine(energy=9648.8, intensity=0.000815698, initial_level='K', final_level=
↪'M4,5')
#
# X-ray absorption edges:
>>> xraydb.xray_edge('As', 'K')
XrayEdge(energy=11867.0, fyield=0.548989, jump_ratio=7.314)
#
# X-ray attenuation factors:
>>> as_kedge = xraydb.xray_edge('As', 'K').energy
>>> energies = np.linspace(-50, 50, 5) + as_kedge
>>> muvals = xraydb.mu_elam('As', energies)
>>> for en, mu in zip(energies, muvals):
...     print("{:.0f}   {:8.2f}".format(en, mu))
...
11817      26.07
11842      25.92
11867      25.77
11892     178.32
11917     177.38
```

Table of XrayDB function for Atomic and X-ray data for the elements

Most of these function return some element-specific property from the element symbol or atomic number. Some of the data extends to Z=98 (Cf), but some data may not be available for Z > 92 (U). Except where noted, the data comes from [Elam, Ravel, and Sieber (2002)].

xraydb functions	description
atomic_number()	atomic number from symbol
atomic_symbol()	atomic symbol from number
atomic_mass()	atomic mass
atomic_name()	atomic name (English)
atomic_density()	density of pure element
f0()	elastic scattering factor ([Waasmaier and Kirfel (1995)])
f0_ions()	list of valid "ions" for *f0()* ([Waasmaier and Kirfel (1995)])
xray_edge()	xray edge data for a particular element and edge
xray_edges()	dictionary of all X-ray edges data for an element
xray_lines()	dictionary of all X-ray emission line data for an element
fluor_yield()	fluorescent yield for an X-ray emission line
ck_probability()	Coster-Kronig transition probability between two atomic levels

continues on next page

Table 1 – continued from previous page

xraydb functions	description
mu_elam()	absorption cross-section, photo-electric or total for an element
coherent_cross_section_elam()	coherent scattering cross-section for an element
incoherent_cross_section_elam()	incoherent scattering cross-section for an element
chantler_energies()	energies of tabulation for Chantler data ([Chantler (2000)])
f1_chantler()	$f'(E)$ anomalous scattering factor ([Chantler (2000)])
f2_chantler()	$f"(E)$ anomalous scattering factor ([Chantler (2000)])
mu_chantler()	absorption cross-section ([Chantler (2000)])
guess_edge()	guess element and edge from energy of absorption edge
chemparse()	parse a chemical formula to atomic abundances
validate_formula()	test whether a chemical formula can be parsed.
get_materials()	get a dictionary of known materials {name:(formula, density)}
get_material()	get a (formula, density) tuple for a material in the materials database
find_material()	get a material instance for a material in the materials database
add_material()	add a material to local materials database
material_mu()	absorption cross-section for a material at X-ray energies
material_mu_components()	dictionary of elemental components of *mu* for material
xray_delta_beta()	anomalous index of refraction for material and energy
darwin_width()	Darwin widths for monochromator crystals
mirror_reflectivity()	X-ray reflectivities for mirror materials (thick slab limit)
multilayer_reflectivity()	X-ray reflectivities for multilayer mirrors
coated_reflectivity()	X-ray reflectivities for coated mirrors
ionization_potential()	effective ionization potential for a gas, as for ion chambers
ionchamber_fluxes()	calculate fluxes from ion chamber voltages, gases, and sensitivities

get_xraydb()

return instance of the XrayDB

Returns
XrayDB

Example

```
>>> import xraydb
>>> xdb = xraydb.get_xraydb()
```

1.4.2 Atomic Properties

atomic_number(*element*)

z for element name

Parameters
element (*str*) – atomic symbol

Returns
atomic number

atomic_symbol(*z*)

atomic symbol for atomic number

Parameters
z (*int*) – atomic number

Returns
atomic symbol

atomic_mass(*element*)

molar mass for an element

Parameters
element (*int, str*) – atomic number, atomic symbol for element

Returns
atomic mass, in AMU

atomic_name(*z*)

atomic name for atomic number

Parameters
z (*int*) – atomic number

Returns
atomic name (English)

atomic_density(*element*)

density (gr/cm^3) for common for of an element

Parameters
element (*int, str*) – atomic number, atomic symbol for element

Returns
density in gm/cm^3

1.4.3 Elastic Scattering Factors

f0(*ion*, *q*)

elastic X-ray scattering factor, f0(q), for an ion.

Parameters

- **ion** (*int or str*) – atomic number, atomic symbol or ionic symbol of scatterer
- **q** (*float, ndarray*) – Q value(s) for scattering

Returns
scattering factor for each Q value

Notes

1. from D. Waasmaier and A. Kirfel, Acta Cryst. A51 p416 (1995) and International Tables for Crystallography, Vol. C.
2. *ion* can be of the form: 26, *Fe*, *Fe2+*. For a full list of ions use *f0_ions()*
3. elements supported are from Z = 1 to 98 ('H' to 'Cf')
4. q = sin(theta) / lambda, where theta=incident angle, lambda=X-ray wavelength

f0_ions(*element=None*)

list ion names supported in the f0() calculation from Waasmaier and Kirfel.

Parameters
element (*None, int, str*) – scatterer

Returns
list of strings for matching ion names

Notes

if element is None, all 211 ions are returned.

1.4.4 X-ray Edges

`xray_edge`(*element*, *edge*, *energy_only=False*)

get x-ray absorption edge data for an element: (energy(in eV), fluorescence yield, jump ratio)

Parameters

- **element** (*int, str*) – atomic number, atomic symbol for element
- **edge** (*str*) – iupac symbol of X-ray edge
- **energy_only** (*bool*) – whether to return only the energy [False]

Returns
XrayEdge namedtuple containing (energy, fluorescence_yield, edge_jump) or float of energy

`xray_edges`(*element*)

get dictionary of x-ray absorption edges:
energy(in eV), fluorescence yield, and jump ratio for an element.

Parameters
element (*int, str*) – atomic number, atomic symbol for element

Returns
dictionary of XrayEdge named tuples.

Notes

1. The dictionary will have keys of edge (iupac symbol) and values containing an XrayEdge namedtuple containing (energy, fluorescence_yield, edge_jump)

`core_width`(*element*, *edge=None*)

returns core hole width for an element and edge

Parameters

- **element** (*int or str*) – element
- **edge** (*None or str*) – edge to consider

Returns
a dict of `{edge: core_hole_width}` if the edge is not specified or a float with the core hole width for a specified edge.

Notes

1. if edge is None, a dict of core hole widths for all edges is returned
2. Data from Krause and Oliver (1979) and Keski-Rahkonen and Krause (1974)
3. Values are in eV

guess_edge(*energy*, *edges=('K', 'L3', 'L2', 'L1', 'M5')*)

guess an element and edge based on energy (in eV)

Parameters

- **energy** (*float*) – approximate edge energy (in eV)
- **edges** (*None or list of strings*) – edges to consider

Returns

a tuple of (atomic symbol, edge) for best guess

Notes

by default, the list of edges is ('K', 'L3', 'L2', 'L1', 'M5')

1.4.5 X-ray Emission Lines

xray_lines(*element*, *initial_level=None*, *excitation_energy=None*)

get dictionary of X-ray emission lines of an element

Parameters

- **element** (*int, str*) – atomic number, atomic symbol for element
- **initial_level** (*None or str*) – iupac symbol of initial level
- **excitation_energy** (*None or float*) – exciation energy

Returns

dict of X-ray lines with keys of siegbahn notation and values of XrayLine tuples of (energy, intensity, initial level, final level)

Notes

1. excitation energy will supercede initial_level, as it means 'all intial levels with below this energy

Exaample:

```
>>> for name, line in xraydb.xray_lines('Mn', 'K').items():
...     print(name, line)
...
Ka3 XrayLine(energy=5769.9, intensity=0.000265963, initial_level='K', final_
↪level='L1')
Ka2 XrayLine(energy=5889.1, intensity=0.293941, initial_level='K', final_level=
↪'L2')
Ka1 XrayLine(energy=5900.3, intensity=0.58134, initial_level='K', final_level=
↪'L3')
Kb3 XrayLine(energy=6491.8, intensity=0.042234, initial_level='K', final_level=
↪'M2')
Kb1 XrayLine(energy=6491.8, intensity=0.0815329, initial_level='K', final_level=
↪'M3')
Kb5 XrayLine(energy=6537.0, intensity=0.000685981, initial_level='K', final_
↪level='M4,5')
```

fluor_yield(*element*, *edge*, *line*, *energy*)

fluorescence yield for an X-ray emission line or family of lines.

Parameters

- **element** (*int, str*) – atomic number, atomic symbol for element
- **edge** (*str*) – iupac symbol of X-ray edge
- **line** (*str*) – siegbahn notation for emission line
- **energy** (*float*) – incident X-ray energy

Returns
fluorescence yield, weighted average fluorescence energy, net_probability

Examples

```
>>> xraydb.fluor_yield('Fe', 'K', 'Ka', 8000)
0.350985, 6400.752419799043, 0.874576096
```

```
>>> xraydb.fluor_yield('Fe', 'K', 'Ka', 6800)
0.0, 6400.752419799043, 0.874576096
```

```
>>> xraydb.fluor_yield('Ag', 'L3', 'La', 6000)
0.052, 2982.129655446868, 0.861899000000000
```

See also

xray_lines which gives the full set of emission lines ('Ka1', 'Kb3', etc) and probabilities for each of these.

ck_probability(*element*, *initial*, *final*, *total=True*)

transition probability for an element, initial, and final levels.

Parameters

- **element** (*int, str*) – atomic number, atomic symbol for element
- **initial** (*str*) – iupac symbol for initial level
- **final** (*str*) – iupac symbol for final level
- **total** (*bool*) – whether to include transitions via possible intermediate levels [True]

Returns
transition probability, or 0 if transition is not allowed.

1.4.6 Absorption and Scattering Cross-sections

mu_elam(*element*, *energy*, *kind='total'*)

X-ray mass attenuation coefficient, mu/rho, for an element and energy or array of energies. Data is from the Elam tables.

Parameters

- **element** (*int, str*) – atomic number, atomic symbol for element
- **energy** (*float or ndarray*) – energy or array of energies
- **kind** (*str*) – type of cross-section to use, one of ('total', 'photo', 'coh', 'incoh') ['total']

Returns
float value or ndarray

Notes

1. Values returned are in units of cm^2/gr
2. The default is to return total attenuation coefficient.

coherent_cross_section_elam(*element*, *energy*)

coherent scaattering cross-section for an element and energy or array of energies. Data is from the Elam tables.

Parameters

- **element** (*int, str*) – atomic number, atomic symbol for element
- **energy** (*float or ndarray*) – energy or array of energies

Returns
float value or ndarray

Notes

1. Values returned are in units of cm^2/gr

incoherent_cross_section_elam(*element*, *energy*)

incoherent scaattering cross-section for an element and energy or array of energies. Data is from the Elam tables.

Parameters

- **element** (*int, str*) – atomic number, atomic symbol for element
- **energy** (*float or ndarray*) – energy or array of energies

Returns
float value or ndarray

Notes

1. Values returned are in units of cm^2/gr

chantler_energies(*element*, *emin=0*, *emax=1000000000.0*)

energies at which Chantler data is tabulated for a particular element.

Parameters

- **element** (*int, str*) – atomic number, atomic symbol for element
- **emin** (*float*) – lower bound of energies (default=0)
- **emax** (*float*) – upper bound of energies (default=1.e9)

Returns
ndarray of energies

Notes

energies are in eV

f1_chantler(*element*, *energy*, ***kws*)

real part of anomalous x-ray scattering factor for an element and energy or array of energies. Data is from the Chantler tables.

Parameters

- **element** (*int, str*) – atomic number, atomic symbol for element

- **energy** (*float or ndarray*) – energy or array of energies

Returns
float value or ndarray

Notes

1. Values returned are in units of electrons

f2_chantler(*element*, *energy*)

imaginary part of anomalous x-ray scattering factor for an element and energy or array of energies. Data is from the Chantler tables.

Parameters

- **element** (*int, str*) – atomic number, atomic symbol for element
- **energy** (*float or ndarray*) – energy or array of energies

Returns
float value or ndarray

Notes

1. Values returned are in units of electrons

mu_chantler(*element*, *energy*, *incoh=False*, *photo=False*)

X-ray mass attenuation coeficient, mu/rho, for an element and energy or array of energies. Data is from the Chantler tables.

Parameters

- **element** (*int, str*) – atomic number, atomic symbol for element
- **energy** (*float or ndarray*) – energy or array of energies
- **incoh** (*bool*) – whether to return only the incoherent contribution [False]
- **photo** (*bool*) – whether to return only the photo-electric contribution [False]

Returns
float value or ndarray

Notes

1. Values returned are in units of cm^2/gr
2. The default is to return total attenuation coefficient.

1.4.7 Chemical and Materials database

chemparse(*formula*)

parse a chemical formula to a dictionary of elemental abundances

Parameters
formula (*str*) – chemical formula

Returns
dict of element symbol and abundance.

Examples

```
>>> from xraydb import chemparse
>>> chemparse('Mn(SO4)2(H2O)7)')
{'H': 14.0, 'S': 2.0, 'Mn': 1, 'O': 15.0}
```

```
>>> chemparse('Zn1.e-5Fe3O4')
{'Zn': 1e-05, 'Fe': 3.0, 'O': 4.0}
```

```
>>> chemparse('CO')
{'C': 1, 'O': 1}
>>> chemparse('Co')
{'Co': 1}
```

```
>>> chemparse('co')
ValueError: unrecognized element or number:
co
```

validate_formula(*formula*)

return whether a chemical formula is valid and can be parsed to a dictionary with chemparse()

Parameters
formula (*str*) – chemical formula

Returns
bool (True or False) for whether chemparse() will succeed

Examples

```
>>> from xraydb import validate_formula
>>> validate_formula('Mn(SO4)2(H2O)7)')
True
```

```
>>> validate_formula('Mn(SO4)2(H2O7')
False
```

```
>>> validate_formula('Z')
False
```

get_materials(*force_read=False*, *categories=None*)

get dictionary of all available materials

Parameters

- **force_read** (*bool*) – whether to force a re-reading of the materials database [False]
- **categories** (*list of strings or None*) – restrict results to those that match category names

Returns
dict with keys of material name and values of Materials instances

Examples

```
>>> for name, m in xraydb.get_materials().items():
...      print(name, m)
...
water H2O 1.0
lead Pb 11.34
aluminum Al 2.7
kapton C22H10N2O5 1.42
polyimide C22H10N2O5 1.42
nitrogen N 0.00125
argon Ar 0.001784
...
```

find_material(*name*)

look up material name, return material instance

Parameters
name (*str*) – name of material or chemical formula

Returns
material instance

Examples

```
>>> xraydb.find_material('kapton')
Material(formula='C22H10N2O5', density=1.42, name='kapton', categories=['polymer'])
```

See also

get_material()

get_material(*name*)

look up material name, return formula and density

Parameters
name (*str*) – name of material or chemical formula

Returns
chemical formula, density of material

Examples

```
>>> xraydb.get_material('kapton')
('C22H10N2O5', 1.43)
```

See also

find_material()

add_material(*name*, *formula*, *density*, *categories=None*)

add a material to the users local material database

Parameters

- **name** (*str*) – name of material
- **formula** (*str*) – chemical formula
- **density** (*float*) – density
- **categories** (*list of strings or None*) – list of category names

Returns
None

Notes

the data will be saved to the file 'xraydb/materials.dat' in the users configuration folder, and will be useful in subsequent sessions.

Examples

```
>>> xraydb.add_material('becopper', 'Cu0.98e0.02', 8.3, categories=['metal'])
```

1.4.8 X-ray properties of materials

For some further examples, see *Calculations of X-ray properties of materials*.

material_mu(*name*, *energy*, *density=None*, *kind='total'*)

X-ray attenuation length (in 1/cm) for a material by name or formula

Parameters

- **name** (*str*) – chemical formul or name of material from materials list.
- **energy** (*float or ndarray*) – energy or array of energies in eV
- **density** (*None or float*) – material density (gr/cm^3).
- **kind** (*str*) – 'photo' or 'total' for whether to return the photo-absorption or total cross-section ['total']

Returns
absorption length in 1/cm

Notes

1. material names are not case sensitive, chemical compounds are case sensitive.
2. mu_elam() is used for mu calculation.
3. if density is None and material is known, that density will be used.

Examples

```
>>> material_mu('H2O', 10000.0)
5.32986401658495
```

material_mu_components(*name*, *energy*, *density=None*, *kind='total'*)

material_mu_components: absorption coefficient (in 1/cm) for a compound

Parameters

- **name** (*str*) – chemical formul or name of material from materials list.

- **energy** (*float or ndarray*) – energy or array of energies in eV
- **density** (*None or float*) – material density (gr/cm^3).
- **kind** (*str*) – 'photo' or 'total'for whether to return photo-absorption or total cross-section ['total']

Returns

dict for constructing mu per element,
with elements 'mass' (total mass), 'density', and

'elements' (list of atomic symbols for elements in material).
For each element, there will be an item (atomic symbol as key) with tuple of (stoichiometric fraction, atomic mass, mu)

Examples

```
>>> xraydb.material_mu('quartz', 10000)
50.36774553547068
>>> xraydb.material_mu_components('quartz', 10000)
{'mass': 60.0843, 'density': 2.65, 'elements': ['Si', 'O'],
'Si': (1, 28.0855, 33.87943243018506), 'O': (2.0, 15.9994, 5.952824815297084)}
```

xray_delta_beta(*material*, *density*, *energy*)

anomalous components of the index of refraction for a material, using the tabulated scattering components from Chantler.

Parameters

- **material** – chemical formula ('Fe2O3', 'CaMg(CO3)2', 'La1.9Sr0.1CuO4')
- **density** – material density in g/cm^3
- **energy** – x-ray energy in eV

Returns
(delta, beta, atlen)

where
delta : real part of index of refraction beta : imag part of index of refraction atlen : attenuation length in cm

These are the anomalous scattering components of the index of refraction:

n = 1 - delta - i*beta = 1 - lambda**2 * r0/(2*pi) Sum_j (n_j * fj)

Adapted from code by Yong Choi

darwin_width(*energy*, *crystal='Si'*, *hkl=(1, 1, 1)*, *a=None*, *polarization='s'*, *ignore_f2=False*, *ignore_f1=False*, *m=1*)

darwin width for a crystal reflection and energy

Args: energy (float): X-ray energy in eV crystal (string): name of crystal (one of 'Si', 'Ge', or 'C') ['Si'] hkl (tuple): h, k, l for reflection [(1, 1, 1)] a (float or None): lattice constant [None - use built-in value] polarization ('s','p', 'u'): mono orientation relative to X-ray polarization ['s'] ignore_f1 (bool): ignore contribution from f1 - dispersion (False) ignore_f2 (bool): ignore contribution from f2 - absorption (False) m (int): order of reflection [1]

Returns:

A named tuple 'DarwinWidth' with the following fields

theta: float, nominal Bragg angle, in rad,

theta_offset: float, angular offset from Bragg angle, in rad,

theta_width: float, estimated angular Darwin width, in rad,

theta_fwhm: float, estimated FWHM of angular intensity, in rad,

rocking_theta_fwhm: float, estimated FWHM of a rocking curve, in rad,

energy_width: float, estimated angular Darwin width, in rad,

energy_fwhm: float, estimated FWHM of energy intensity, in eV,

rocking_energy_fwhm: float, estimated FWHM of a rocking curve, in eV,

zeta: nd-array of Zeta parameter (delta_Lambda / Lambda),

dtheta: nd-array of angles away from Bragg angle, theta in rad,

denergy: nd-array of energies away from Bragg energy, in eV,

intensity: nd-array of reflected intensity,

rocking_curve: nd-array of rocking curve of 2 crystals,

Notes:

1. This follows the calculation from section 6.4 of Elements of Modern X-ray Physics, 2nd Edition J Als-Nielsen, and D. McMorrow.

2. Only diamond structures (Si, Ge, diamond) are currently supported. Default values of lattice constant *a* are in Angstroms: 5.4309 for Si, 5.6578, for 'Ge', and 3.567 for 'C'.

3. The *theta_width* and *energy_width* values will closely match the width of the intensity profile that would = 1 when ignoring the effect of absorption. These are the values commonly reported as 'Darwin Width'. The value reported for *theta_fwhm' and `energy_fwhm* are larger than this by sqrt(9/8) ~= 1.06.

4. Polarization can be 's', 'p', 'u', or None. 's' means vertically deflecting crystal and a horizontally-polarized source, as for most synchrotron beamlines. 'p' is for a horizontally-deflecting crystal. 'u' or None is for unpolarized light, as for most fluorescence/emission.

5. The *rocking_curve* will be the convolution of the intensity with itself, to simulate an angular scan of one crystal with respect to the other, as is often done with double-crystal monochromators. *rocking_theta_fwhm* and *rocking_energy_fwhm* will be the FWHM of this curve in angle and energy, and will typiccally be ~1.5x the Darwin widths in *theta_width* and *energy_width*, respectively.

Examples: >>> dw = darwin_width(10000, crystal='Si', hkl=(1, 1, 1)) >>> print(dw.theta_width, dw.energy_width) 2.695922108316184e-05 1.336668903324966

mirror_reflectivity(*formula*, *theta*, *energy*, *density=None*, *roughness=0.0*, *polarization='s'*, *output='intensity'*)

mirror reflectivity for a thick, single-layer mirror.

Parameters

- **formula** (*string*) – material name or formula ('Si', 'Rh', 'silicon')
- **theta** (*float or nd-array*) – mirror angle in radians
- **energy** (*float or nd-array*) – X-ray energy in eV
- **density** (*float or None*) – material density in g/cm^3
- **roughness** (*float*) – mirror roughness in Angstroms
- **polarization** (*'s' or 'p'*) – mirror orientation relative to X-ray polarization

- **output** (*str*) – output intensity or or complex amplitude

Returns
mirror reflectivity values

Notes

1. If both theta and energy are nd-arrays, it returns a 2D array of theta vs energy. Ex: r[i,j] = r(theta[i], energy[j])
2. density can be *None* for known materials
3. polarization of 's' puts the X-ray polarization along the mirror surface, 'p' puts it normal to the mirror surface. For horizontally polarized X-ray beams from storage rings, 's' will usually mean 'vertically deflecting' and 'p' will usually mean 'horizontally deflecting'.

multilayer_reflectivity(*stackup*, *thickness*, *substrate*, *theta*, *energy*, *n_periods=1*, *density=None*, *substrate_density=None*, *substrate_rough=0.0*, *surface_rough=0.0*, *polarization='s'*, *output='intensity'*)

reflectivity for a multilayer mirror.

Parameters

- **stackup** (*list of formulas*) – material name or formula ('Si', 'Rh', 'silicon')
- **thickness** (*list*) – thickness of layers in Angstroms
- **substrate** (*string*) – substrate material name or formula
- **theta** (*float or nd-array*) – mirror angle in radians
- **energy** (*float or nd-array*) – X-ray energy in eV
- **n_periods** (*int*) – number of periods in multilayer
- **density** (*list or None*) – material densities in g/cm^3
- **substrate_density** (*float*) – density of substrate in g/cm^3
- **substrate_rough** (*float*) – mirror roughness in Angstroms
- **surface_rough** (*float*) – mirror roughness in Angstroms
- **polarization** (*'s' or 'p'*) – mirror orientation relative to X-ray polarization
- **output** (*str*) – output intensity or or complex amplitude

Returns
mirror reflectivity values

Notes

1. only one of theta or energy can be an nd-array
2. thickness should be the same length as stackup
3. density can be *None* for known materials or a list of 1 period (['Mo', 'Si']) for a multilayer stackup.
4. polarization of 's' puts the X-ray polarization along the mirror surface, 'p' puts it normal to the mirror surface. For horizontally polarized X-ray beams from storage rings, 's' will usually mean 'vertically deflecting' and 'p' will usually mean 'horizontally deflecting'.

coated_reflectivity(*coating*, *coating_thick*, *substrate*, *theta*, *energy*, *coating_dens=None*, *surface_roughness=0.0*, *substrate_dens=None*, *substrate_roughness=0.0*, *binder=None*, *binder_thick=None*, *binder_dens=None*, *polarization='s'*, *output='intensity'*)

reflectivity for a coated mirror.

Parameters

- **coating** (*str*) – coating material name or formula ('Si', 'Rh', 'silicon')
- **coating_thick** (*float*) – thickness of coating in Angstroms
- **substrate** (*string*) – substrate material name or formula
- **theta** (*float or nd-array*) – mirror angle in radians
- **energy** (*float or nd-array*) – X-ray energy in eV
- **coating_dens** (*None*) – density of mirror coating in g/cm^3
- **surface_roughness** (*float*) – coating roughness in Angstroms
- **substrate_dens** (*float*) – density of substrate in g/cm^3
- **substrate_roughness** (*float*) – substrate roughness in Angstroms
- **binder** (*str*) – binder material name or formula
- **binder_thick** (*float*) – thickness of binder in Angstroms
- **binder_dens** (*float*) – density of binder in g/cm^3
- **polarization** (*'s' or 'p'*) – mirror orientation relative to X-ray polarization
- **output** (*str*) – output intensity or or complex amplitude

Returns

mirror reflectivity values

Notes

1. only one of theta or energy can be an nd-array
2. densities can be *None* for known materials or a list of 1 period (['Mo', 'Si']) for a multilayer stackup.
3. polarization of 's' puts the X-ray polarization along the mirror surface, 'p' puts it normal to the mirror surface. For horizontally polarized X-ray beams from storage rings, 's' will usually mean 'vertically deflecting' and 'p' will usually mean 'horizontally deflecting'.

ionization_potential(*gas*)

return effective ionization potential for a gas or diode semiconductor, as appropriate for ionization chambers in the linear regime (not in the 'proportional counter' regime) or for PIN photodiodes (not in 'avalanche' mode).

Parameters

gas (*string*) – name of gas or 'Si' or 'Ge'

Returns

ionization potential in eV

Notes

Data from G. F. Knoll, Radiation Detection and Measurement, Table 5-1, and from ICRU Report 31, 1979. Supported gas names and effective potentials:

gas names	potential (eV)
hydrogen, H	36.5
helium, He	41.3
nitrogen, N, N2	34.8
oxygen, O, O2	30.8
neon, Ne	35.4
argon, Ar	26.4
krypton, Kr	24.4
xenon, Xe	22.1
air	33.8
methane, CH4	27.3
carbondioxide, CO2	33.0
silicon, Si	3.68
germanium, Ge	2.97

If the gas is not recognized the default value of 32.0 eV will be returned.

ionchamber_fluxes(*gas='nitrogen'*, *volts=1.0*, *length=100.0*, *energy=10000.0*, *sensitivity=1e-06*, *sensitivity_units='A/V'*, *with_compton=True*, *both_carriers=True*)

return ion chamber and PIN diode fluxes for a gas, mixture of gases, or semiconductor material, ion chamber length (or diode thickness), X-ray energy, recorded voltage and current amplifier sensitivity. See note for details.

Parameters

- **gas** (*string or dict*) – name or formula of fill gas (see note 1) ['nitrogen']
- **volts** (*float*) – measured voltage output of current amplifier [1.0]
- **length** (*float*) – active length of ion chamber in cm [100]
- **energy** (*float*) – X-ray energy in eV [10000]
- **sensitivity** (*float*) – current amplifier sensitivity [1.e-6]
- **sensitivity_units** (*string*) – units of current amplifier sensitivity (see note 2 for options) ['A/V']
- **with_compton** (*bool*) – switch to control the contribution of Compton scattering (see note 3) [True]
- **both_carriers** (*bool*) – switch to control whether to count both electron and ion current (see note 4) [True]

Returns

named tuple IonchamberFluxes with fields

incident flux of beam incident on ion chamber in Hz

transmitted flux of beam output of ion chamber in Hz

photo flux absorbed by photo-electric effect in Hz

incoherent flux attenuated by incoherent scattering in Hz

Examples

```
>>> from xraydb import ionchamber_fluxes
>>> fl = ionchamber_fluxes(gas='nitrogen', volts=1.25,
                           length=20.0, energy=10e3, sensitivity=1.e-6)
```

```
>>> print(f"Fluxes: In={fl.incident:g}, Out={fl.transmitted:g}, Transmitted={100*fl.
↪transmitted/fl.incident:.2f}%")
Fluxes: In=3.20045e+11, Out=2.90464e+11, Transmitted=90.76%
```

```
>>> fl = ionchamber_fluxes(gas={'nitrogen':0.5, 'helium': 0.5},
                           volts=1.25, length=20.0, energy=10000.0,
                           sensitivity=1.e-6)
```

```
>>> print(f"Fluxes: In={fl.incident:g}, Out={fl.transmitted:g}, Transmitted={100*fl.
↪transmitted/fl.incident:.2f}%")
Fluxes: In=6.83845e+11, Out=6.51188e+11, Transmitted=95.22%
```

Notes

1. The gas value can either be a string for the name of chemical formula for the gas or diode material, or dictionary with keys that are gas names or formulas and values that are the relative fraction for mixes gases. For diode materials, mixtures are not supported.

   The gas formula is used both the contributions for mu and to get the weighted effective ionization potential for the material.

   The effective ionization potentials are known for a handful of gases and diodes (see *ionization_potential* function), and range between 20 and 45 eV for gases, and around 3 eV for semiconductors. For unknown gases the value of 32.0 eV will be used.

2. The *sensitivity* and *sensitivity_units* arguments have some overlap to specify the sensitivity of the current amplifier. Generally, the units are in *A/V*, but you can add a common SI prefix of 'p', 'pico', 'n', 'nano', (unicode 'u03bc'), 'u', 'micro', 'm', 'milli' so that, *ionchamber_fluxes(..., sensitivity=1.e-6)* and *ionchamber_fluxes(..., sensitivity=1, sensitivity_units='uA/V')* will both give a sensitivity of 1 microAmp / Volt.

3. The effect of Compton scattering on the ion chamber current can be approximated using the mean energy of the Compton-scattered electron. See the documentation for more details. Set *with_compton=False* to turn off this correction.

4. The effective ionization potential generates an electron and ions pair, and normally both carriers will contribute to the current. Thus, the number of carries below, *N_carriers* is 2. To consider the current from 1 carrier, for example if using a Frisch grid, use *both_carries=False*, which will set *N_carriers* to 1.

1.5 Overview of Atomic and X-ray Data

The data provided in XrayDB includes Atomic data and characteristic energies and cross sections for the interaction of X-rays with elements. A few definitions and conventions necessary for using this data are discussed here.

1.5.1 Elements

Most of the data resources are accessed by an elements *Atomic Symbol*. For the Python module, most methods will take *element* as the first argument, and this can either be the integer atomic number or the string for the atomic symbol.

1.5.2 Physical Units

Elemental densities are given in gr/cm^3, and molar masses are given in AMU. Unless otherwise stated, all energies are in units of eV.

1.5.3 X-ray Edges

Several resources (database tables, python methods) take either an `edge` or a `level` argument to signify a core electronic level. These are strings and must be one of the levels listed in the *Table of X-ray edge names*.

Table of X-ray Edges and Core electronic levels. The Names are the IUPAC symbols for the core electronic levels.

Name	electronic level	Name	electronic level
K	$1s$	N5	$4d_{5/2}$
L3	$2p_{3/2}$	N4	$4d_{3/2}$
L2	$2p_{1/2}$	N3	$4p_{3/2}$
L1	$2s$	N2	$4p_{1/2}$
M5	$3d_{5/2}$	N1	$4s$
M4	$3d_{3/2}$	O3	$5p_{3/2}$
M3	$3p_{3/2}$	O2	$5p_{1/2}$
M2	$3p_{1/2}$	O1	$5s$
M1	$3s$	P3	$6p_{3/2}$
N7	$4f_{7/2}$	P2	$6p_{1/2}$
N6	$4f_{5/2}$	P1	$6s$

1.5.4 X-ray Lines

Many resources (database tables or methods) take emission line arguments. These are all strings and follow the latinized version of the Siegbahn notation as indicated in the *Table of X-ray emission line names*.

Table of X-ray emission line names and the corresponding Siegbahn and IUPAC notations

Name	IUPAC	Siegbahn	Name	IUPAC	Siegbahn
Ka1	K-L3	$K\alpha_1$	Lb4	L1-M2	$L\beta_4$
Ka2	K-L2	$K\alpha_2$	Lb5	L3-O4,5	$L\beta_5$
Ka3	K-L1	$K\alpha_3$	Lb6	L3-N1	$L\beta_6$
Kb1	K-M3	$K\beta_1$	Lg1	L2-N4	$L\gamma_1$
Kb2	K-N2,3	$K\beta_2$	Lg2	L1-N2	$L\gamma_2$
Kb3	K-M2	$K\beta_3$	Lg3	L1-N3	$L\gamma_3$
Kb4	K-N4,5	$K\beta_2$	Lg6	L2-O4	$L\gamma_6$
Kb5	K-M4,5	$K\beta_3$	Ll	L3-M1	Ll
La1	L3-M5	$L\alpha_1$	Ln	L2-M1	$L\nu$
La2	L3-M4	$L\alpha_1$	Ma	M5-N6,7	$M\alpha$
Lb1	L2-M4	$L\beta_1$	Mb	M4-N6	$M\beta$
Lb2,15	L3-N4,5	$L\beta_2, L\beta_{15}$	Mg	M3-N5	$M\gamma$
Lb3	L1-M3	$L\beta_3$	Mz	M4,5-N6,7	$M\zeta$

1.5.5 Cross Sections

The photo-absorption and scattering cross sections from [Elam, Ravel, and Sieber (2002)] and [Chantler (2000)] are in cm^2/gr.

The data from [Elam, Ravel, and Sieber (2002)] is held as logarithms of energy, cross section, and logarithm of the 2nd derivative of cross section that allows for cubic spline interpolation in log-log space.

1.6 Using the XrayDB xraydb.sqlite

All the data for the X-ray database is held in the SQLite3 file `xraydb.sqlite`. To use with SQLite, this file is all you need. While many programs and languages can access SQLite files, basic usage with the `sqlite3` program (available from Windows, Mac OS X, and Linux) can be as simple as:

```
system~> sqlite3 xraydb.sqlite
sqlite> .headers on
sqlite> select * from elements where atomic_number=47;
atomic_number|element|name|molar_mass|density
47|Ag|silver|107.868|10.48
```

That is, you can retrieve the data using standard SQL queries built-in to SQLite. Of course, the expectation is that you'd want to use this database within a programming environment. Currently, wrappers exist only for Python.

1.6.1 Overall Database Schema

The schema for the SQLite3 database describes where data is held in the database, and how to access it. The schema for the current version (4) looks like this:

```
Table Version (id integer primary key,
               tag text,
               date text,
               notes text);
Table elements (atomic_number integer primary key,
                element text,
                name text,
                molar_mass real,
                density real);
Table xray_levels (id integer primary key,
                   element text,
                   iupac_symbol text,
                   absorption_edge real,
                   fluorescence_yield real,
                   jump_ratio real);
Table xray_transitions (id integer primary key,
                        element text,
                        iupac_symbol text,
                        siegbahn_symbol text,
                        initial_level text,
                        final_level text,
                        emission_energy real,
                        intensity real);
Table Coster_Kronig (id integer primary key,
                     element text,
                     initial_level text,
                     final_level text,
                     transition_probability real,
                     total_transition_probability real);
Table photoabsorption (id integer primary key,
                       element text,
                       log_energy text,
                       log_photoabsorption text,
                       log_photoabsorption_spline text);
```

(continues on next page)

(continued from previous page)

```
Table scattering (id integer primary key,
                  element text,
                  log_energy text,
                  log_coherent_scatter text,
                  log_coherent_scatter_spline text,
                  log_incoherent_scatter text,
                  log_incoherent_scatter_spline text);
Table Waasmaier (id integer primary key,
                 atomic_number integer,
                 element text,
                 ion text,
                 offset real,
                 scale text,
                 exponents text);
Table KeskiRahkonen_Krause (id integer primary key,
                            atomic_number integer,
                            element text,
                            edge text,
                            width float);
Table Krause_Oliver (id integer primary key,
                     atomic_number integer,
                     element text,
                     edge text,
                     width float);
Table corelevel_widths (id integer primary key,
                        atomic_number integer,
                        element text,
                        edge text,
                        width float);
Table Chantler (id integer primary key,
                element text,
                sigma_mu real,
                mue_f2 real,
                density real,
                corr_henke float,
                corr_cl35 float,
                corr_nucl float,
                energy text,
                f1 text,
                f2 text,
                mu_photo text,
                mu_incoh text,
                mu_total text);
```

More details for each table are given below.

Note

in the tables below the type of *json array* means that arrays of numerical data are stored in the database as text of JSON-encoded arrays.

1.6.2 Version Table

The *Version* table holds data about the revisions to the database file itself. Each row represents a single revision.

DB Table of Database Versions

Column	Type	Description
id	integer	counter (primary tag)
tag	text	version name
date	text	date string
notes	text	notes on changes for version

1.6.3 Elements Table

The *elements* table holds basic data about each element. Each row represents an element.

DB Table of Basic Properties of the Elements

Column	Type	Description
atomic_number	integer	Atomic Number, Z
element	text	Atomic symbol
name	text	English name of element
molar_mass	float	Atomic mass in AMU
density	float	Density of pure element (gr/cm^3)

1.6.4 Xray_Levels Table

The *xray_levels* table holds data for electronic levels of atoms. Each row represents a core electronic level.

DB Table of X-ray and core electronic levels. *fluorescence yield* gives the probability of an empty level refilling by X-ray fluorescence. The *jump ratio* is the ratio of values for photo-electric cross section (that is, from *Photoabsorption Table*) 1 eV above the absorption edge to that 1 eV below the absorption edge. See *Table of X-ray Edges*

Column	Type	Description
id	integer	Index (primary key)
element	text	Atomic symbol for element
iupac_symbol	text	IUPAC symbol for level ('K','L3',...)
absorption_edge	float	binding energy for level (eV)
fluorescence_yield	float	fluorescence yield (fraction)
jump_ratio	float	ratio of mu_photo across edge

1.6.5 Xray_Transitions Table

The *xray_transitions* table holds data for transitions between electronic levels of atoms. Each row represents a transition between two levels.

DB Table of X-ray Transitions. Both IUPAC and Siegbahn symbols are given (see *Table of X-ray emission lines*), as well as the initial and final levels. The *intensity* is the relative intensity of the transition for a given *initial level*.

Column	Type	Description
id	integer	Index (primary key)
element	text	Atomic symbol for element
iupac_symbol	text	IUPAC symbol for transition
siegbahn_symbol	text	Siegbahn symbol for transition
initial_level	text	IUPAC symbol for initial level
final_level	text	IUPAC symbol for final level
emission_energy	float	fluorescence energy (eV)
intensity	float	relative intensity for transition

1.6.6 Photoabsorption Table

The *photoabsorption* table holds data for the photo-electric absorption cross sections in cm^2/gr. Each row represents an element.

DB Table of Photoabsorption Cross Sections. JSON-encoded arrays are held for logs of energy, cross section, and cross section spline (second derivative useful for spline interpolation).

Column	Type	Description
id	integer	Index (primary key)
element	text	Atomic symbol for element
log_energy	json array	log of Energy values (eV)
log_photoabsorption	json array	log of cross section (cm^2/gr)
log_photoabsorption_spline	json array	log of cross section spline

1.6.7 Scattering Table

The *scattering* table holds data for the coherent and incoherent X-ray scattering cross sections, in cm^2/gr. Each row represents an element.

DB Table of Coherent and Incoherent Scattering Cross Sections. JSON-encoded arrays are held for logs of energy, cross section, and cross section spline (second derivative useful for spline interpolation).

Column	Type	Description
id	integer	Index (primary key)
element	text	Atomic symbol for element
log_energy	json array	log of Energy values (eV)
log_coherent_scatter	json array	log of cross section (cm^2/gr)
log_coherent_scatter_spline	json array	log of cross section spline
log_incoherent_scatter	json array	log of cross section (cm^2/gr)
log_incoherent_scatter_spline	json array	log of cross section spline

1.6.8 Coster_Kronig Table

The *Coster_Kronig* table holds data for energy levels, partial and total transition probabilities for the Coster-Kronig transitions (Auger processes in which the empty core level is filled from an electron in a higher level with the same principle quantum number). The partial probability describes direct transitions, while the total probability includes cascade effects. Each row represents a transition.

DB Table of Coster-Kronig Transitions.

Column	Type	Description
id	integer	Index (primary key)
element	text	Atomic symbol for element
initial_level	text	IUPAC symbol for initial level
final_level	text	IUPAC symbol for final level
transition_probability	float	direct transition probability
total_transition_probability	float	total transition probability

1.6.9 Waasmaier Table

The *Waasmaier* table holds data for calculating elastic X-ray scattering factors $f_0(k)$, from [Waasmaier and Kirfel (1995)]. The scattering factor is unitless, and $k = \sin(\theta)/\lambda$ where θ is the scattering angle and λ is the X-ray wavelength. available for many common ionic states for each element. Each row represents an ion.

DB Table of Elastic Scattering Cross Section Coefficients

Column	Type	Description
id	integer	Index (primary key)
atomic_number	integer	Atomic Number, Z
element	text	Atomic symbol for element
ion	text	symbol for element and ionization
offset	float	offset value
scale	json array	coefficients for calculation
exponents	json array	coefficients for calculation

1.6.10 KeskiRahkonen_Krause Table

The *KeskiRahkonen_Krause* table holds data for energy widths of the core electronic levels from [Keski-Rahkonen and Krause (1974)]. Values are in eV, and each row represents an energy level for an element.

DB Table of Core Hole Widths from Keski-Rahkonen and Krause

Column	Type	Description
id	integer	Index (primary key)
atomic_number	integer	Atomic Number, Z
element	text	Atomic symbol for element
edge	text	IUPAC symbol for energy level ('K')
width	float	width of level (eV)

1.6.11 Krause_Oliver Table

The *Krause_Oliver* table holds data for energy widths of the core electronic levels from [Krause and Oliver (1979)]. Values are in eV, and each row represents an energy level for an element.

DB Table of Core Hole Widths from Krause and Oliver

Column	Type	Description
id	integer	Index (primary key)
atomic_number	integer	Atomic Number, Z
element	text	Atomic symbol for element
edge	text	IUPAC symbol for energy level ('K')
width	float	width of level (eV)

1.6.12 Compton Energies Table

The *Compton_energies* table holds data for median (90 deg scattering) and mean values of the energies of Compton scattered X-rays, and the mean values of the Compton-scattered electrons as a function of incident X-ray energy. There is only 1 row in this table, with all columns being json-encoded arrays of floats. These values should be finely-spaced enough for linear interpolation

DB Table of Compton-scattered energies.

Column	Type	Description
incident	json_array	Incident X-ray energies (eV)
xray_90deg	json_array	Median scattered X-ray energies (eV)
xray_mean	json_array	Mean scattered X-ray energies (eV)
electron_mean	json_array	Mean scattered electron energies (eV)

1.6.13 Chantler Table

The *Chantler* table holds data for resonant X-ray scattering factors $f'(E)$ and $f''(E)$ as well as photo-electric absorption, coherent, and incoherent scattering factors from [Chantler (2000)]. As with other tables, scattering factors are unitless, and cross sections are in cm^2/gr. Each row represents an element.

DB Table of resonant scattering and mass attenuation coefficients from Chantler.

Column	Type	Description
id	integer	Index (primary key)
element	text	Atomic symbol for element
mue_f2	float	factor to convert mu(E) to f"(E)
density	float	atomic density (gr/cm^3)
corr_henke	float	Henke correction to f`(E)
corr_cl35	float	Cromer-Liberman correction to f`(E)
corr_nucl	float	nuclear correction to f`(E)
energy	json array	energies for interpolation
f1	json array	f'(E) (e)
f2	json array	f"(E) (e)
mu_photo	json array	photoabsorption mu(E) (cm^2/gr)
mu_incoh	json array	incoherent scattering (cm^2/gr)
mu_total	json array	total attenuation (cm^2/gr)

1.7 References

BIBLIOGRAPHY

[Als-Nielsen and McMorrow (2011)] J. Als-Nielsen and D. McMorrow. *Elements of Modern X-ray Physics, 2nd Edition*. John Wiley & Sons, 2011. URL: https://dx.doi.org/10.1002/9781119998365.

[Chantler (2000)] C. T. Chantler. Detailed tabulation of atomic form factors, photoelectric absorption and scattering cross section, and mass attenuation coefficients in the vicinity of absorption edges in the soft X-ray (Z=30-36, Z=60-89, E=0.1 keV-10 keV), addressing convergence issues of earlier work. *Journal of Physical and Chemical Reference Data*, 29(4):597–1048, JUL-AUG 2000. URL: https://dx.doi.org/10.1063/1.1321055.

[Chantler (2016)] C. T. Chantler. FFAST Data on a finer energy grid: Personal Communication. 2016.

[Elam, Ravel, and Sieber (2002)] W. T. Elam, B. D. Ravel, and J. R. Sieber. A new atomic database for x-ray spectroscopic calculations. *Radiation Physics and Chemistry*, 63(2):121–128, February 2002. URL: https://dx.doi.org/10.1016/S0969-806X(01)00227-4.

[Hipp (2012)] D. R. Hipp. Sqlite. 2012. URL: https://www.sqlite.org.

[Keski-Rahkonen and Krause (1974)] O. Keski-Rahkonen and M. O. Krause. Total and partial atomic-level widths. *Atomic Data and Nuclear Data Tables*, 14(2):139–146, 1974. URL: https://dx.doi.org/10.1016/S0092-640X(74)80020-3.

[Knoll (2010)] G. .F. Knoll. *Radiation Detection and Measurement, 4th Edition*. John Wiley & Sons, 2010. URL: https://dx.doi.org/10.1002/9780470131480.

[Krause and Oliver (1979)] M. O. Krause and J. H. Oliver. Natural widths of atomic k and l levels, ka x-ray lines and several kll auger lines. *Journal of Physical and Chemical Reference Data*, 8:329, 1979. URL: https://dx.doi.org/10.1063/1.555595.

[Waasmaier and Kirfel (1995)] D. Waasmaier and A. Kirfel. New analytical scattering factor functions for free atoms and ions. *Acta Crystallographica A*, 51:416–431, 1995. URL: https://dx.doi.org/10.1107/S0108767394013292.

PYTHON MODULE INDEX

INDEX