xraylib: a library for interactions of X-rays with matter

Tom Schoonjans

16 May 2019

Outline

1. Introduction and history
2. Current status
    - Available datasets
    - Supported platforms
    - Language bindings
3. Typical usage examples
4. Additional interfaces and applications
5. Under development

A brief introduction to *xraylib*

The quest for an X-ray parameter library...

1. <u>Ease of use</u>
   Preferably software that can be linked with our own software

2. <u>Reliable data</u>
   Most compilations are based on a selection of values considered to be reliable by the authors

XCOM

- NIST: Berger and Hubbell
- Cross sections for scattering and photo ionization
- Compiles into an executable, not a library
- Fortran 77

mucal

- Illinois Institute of Technology: written by Pathikrit Bandyopadhyay
- McMaster and Krause
- Edge energies, fluorescence yields, scattering and photoionization cross sections
- Exports only one function for all types of data
- Fortran 77 and C

Origins of *xraylib*: 2002-2005

- Created by Bruno Golosio to support his work on Monte Carlo simulations and XRF tomography reconstructions
- Contributions from Manuel Sanchez del Rio, Alexandre Simionovici
- Databases taken from Krause, Cullen, Larskin, Deslattes, Hubbell
- Bindings for IDL and Python

A library for X-ray–matter interaction cross sections for X-ray fluorescence applications

A. Brunetti, M. Sanchez del Rio, B. Golosio,
A. Simionovici and A. Somogyi.
Spectrochim. Acta Part B, 59, 1725-1731 (2004).

xraylib: current status

xraylib: What is it?

- Core library written in ANSI-C
- Bindings for many languages
- Very simple and light interface
- Extensive use of macros (`KL3_LINE`, `K_SHELL`, `K_L2M5_AUGER` etc...)
- Thread-safe!
- BSD license: very liberal...
- Hosted on www.github.com/tschoonj/xraylib
- Comprehensive documentation

List of available datasets: part1

- Atomic weight
- (Partial) Photoionization cross sections
- Rayleigh cross sections
- Compton cross sections
- XRF cross sections: cascade effect corrected!
- Klein-Nishina CS
- Differential cross sections: polarized and unpolarized
- Absorption edges
- XRF line energies
- Jump factors
- Radiative rates
- Coster-Kronig transition probabilities

List of available datasets: part 2

- Auger non-radiative rates
- Atomic level widths
- Electronic configurations
- Anomalous scattering
- Atomic form factors
- Incoherent scattering function
- Compton broadening profiles
- Refractive indices
- Crystal diffraction: structure factors, Bragg angles, unit cell volumes, d-spacing, Q scattering amplitude
- Elemental density
- Mass-energy absorption cross sections

Utility functions

- Compound parser for chemical formulas: support for brackets and real indices
- Builtin composition and densities of the NIST compound list
- List of X-ray producing radionuclides
- Many functions support passing chemical formulas and NIST compound names!

xraylib: supported platforms

Core library requires only C standard library

- Source tarball
- Linux
  - DEB packages for Debian/Ubuntu
  - RPM packages for RHEL/Centos/Fedora
- macOS
  - Homebrew, MacPorts and Fink
- Windows
  - Windows 32- and 64-bit SDKs available with DLLs
  - MSYS2: compile from source
- Anaconda: packages available from conda-forge

xraylib: language bindings

- C++ and Objective-C
- Fortran 2003
- Perl
- Python (2.7/3.x)
- Lua
- Java
- Ruby
- PHP
- .NET/C#
- Pascal/Delphi
- IDL (soon unsupported)
- Matlab (unsupported)
- Mathematica (unsupported)
- Labview (unsupported)

xraylib: language bindings vs. platforms

	C C++	Fortran 2003	Python	Perl	Lua	Ruby	Java	.NET C#	PHP	Pascal & Delphi
Linux										
macOS										
Windows (MSYS2)										
Windows (SDKs)										
Anaconda										

Typical usage examples

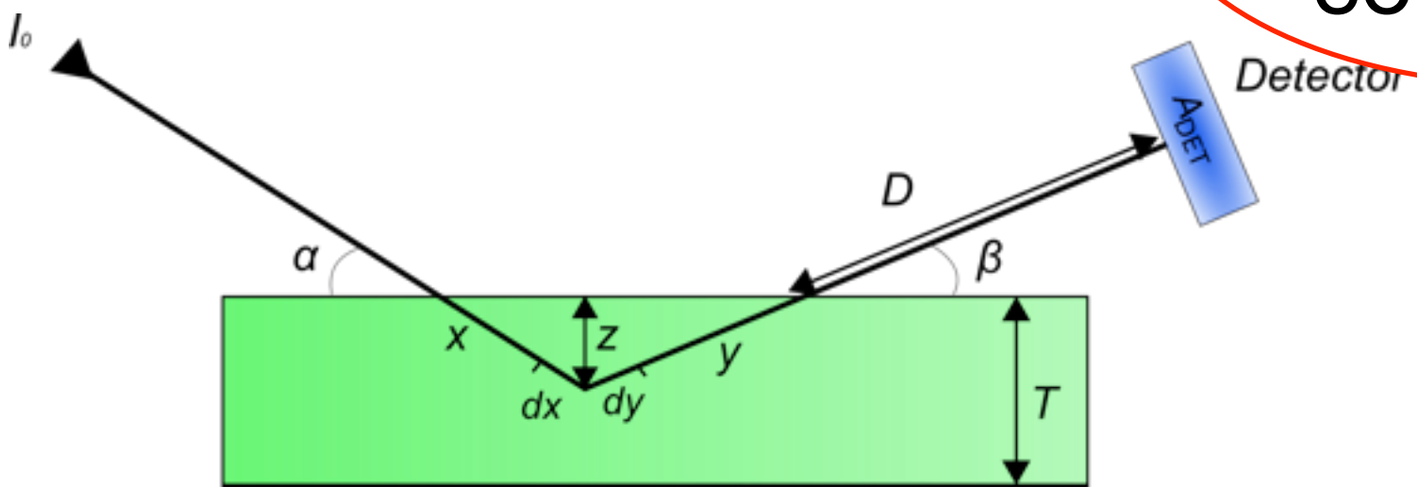
Fundamental pa ethod

XRF cross section

$$I_{i,K\alpha} = I_0 G w_i Q_{i,K\alpha} \rho T \left(\frac{1 - \exp(-\chi \rho T)}{\chi \rho T} \right)$$

$$\chi = \frac{\mu_0}{\sin \alpha} + \frac{\mu_1}{\sin \beta}$$

Mass attenuation coefficients



Fundamental parameter method

Parse chemical formula

```
struct compoundData* CompoundParser(
  const char compoundString[])
```

XRF production cross section $Q_{i,K\alpha}$

```
double CS_FluorLine_Kissel(int Z, int line, double energy)
```

Mass attenuation coefficients $\mu_0 \mu_1$

```
double CS_Total_CP(const char compound[], double energy)
```

```
double LineEnergy(int Z, int line)
```

```
PROGRAM fpm

USE :: xraylib
USE, INTRINSIC :: ISO_C_BINDING

IMPLICIT NONE

REAL (C_DOUBLE) :: flux = 1E9 !photons/s
REAL (C_DOUBLE) :: G = 1E-5
REAL (C_DOUBLE) :: density = 3.19 !g/cm3
REAL (C_DOUBLE) :: thickness = 0.1 !cm
REAL (C_DOUBLE) :: xrf_intensity, chi
REAL (C_DOUBLE) :: mu_0, mu_1, w_Ca, A_corr, Q
REAL (C_DOUBLE) :: alpha = 45.0, beta = 45.0 !degrees
REAL (C_DOUBLE) :: beam_energy = 20.0 !keV
TYPE (compoundData), POINTER :: cd
CHARACTER (len=50) :: apatite = 'Ca5(PO4)3(OH)0.33F0.33Cl0.33'
REAL (C_DOUBLE), PARAMETER :: deg2rad = 3.14159265359/180.0

cd => compoundParser(apatite)
w_Ca = cd%massFractions(6)

mu_0 = CS_Total_CP(apatite, beam_energy)
mu_1 = CS_Total_CP(apatite, LineEnergy(20, KL3_LINE))
chi = mu_0/SIN(deg2rad*alpha) + mu_1/SIN(deg2rad*beta)
A_corr = (1.0-EXP(-chi*density*thickness))/chi*density*thickness
Q = CS_FluorLine_Kissel(20, KL3_LINE, beam_energy)

xrf_intensity = flux*G*Q*w_Ca*density*thickness*A_corr

CALL FreeCompoundData(cd)

WRITE (*, '(A, ES12.4)') 'xrf_intensity: ', xrf_intensity
END PROGRAM fpm
```

Monte Carlo simulation of an X-ray attenuation experiment

Lambert-Beer probability density function:

$$f(x) = \mu\rho \exp(-\mu\rho x)$$

Lambert-Beer cumulative distribution function:

$$F(x) = \int_0^x \mu\rho \exp(-\mu\rho t)dt = 1 - \exp(-\mu\rho x) \equiv R \Longrightarrow x = -\frac{1}{\mu\rho}\ln(1-R) \Leftrightarrow x = -\frac{1}{\mu\rho}\ln(R)$$

1. With a random number generator, calculate *n* attenuation depths *x*.
2. Whenever *x* greater then thickness, add 1 to *transmitted*

Monte Carlo simulation of an X-ray absorption experiment

Density of a NIST catalog compound

```
struct compoundDataNIST* GetCompoundDataNISTByName(
  const char compoundString[])
```

Mass attenuation coefficients μ

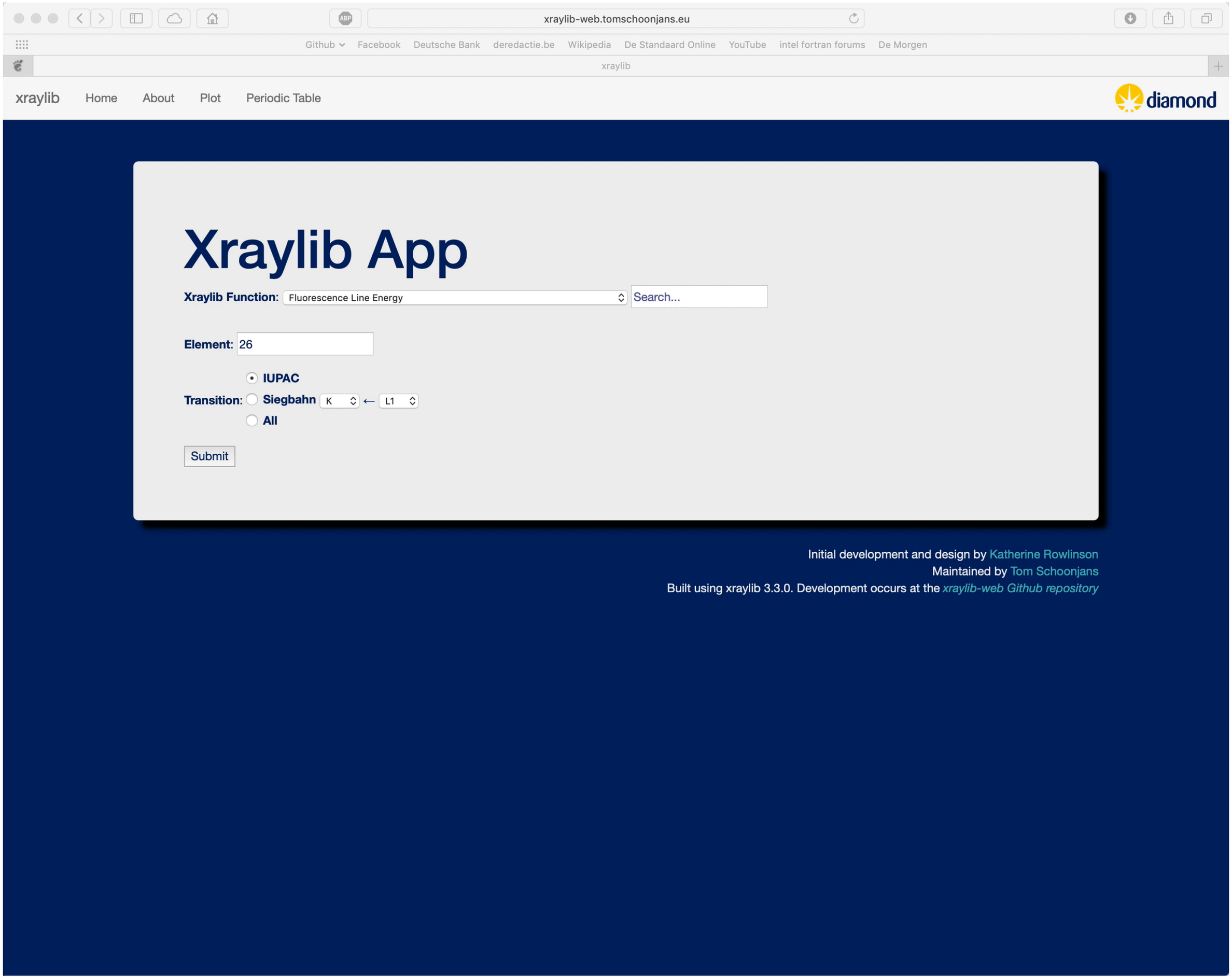
```
double CS_Total_CP(char compound[], double energy)
```

```ruby
require 'xraylib'

compound = "Uranium Monocarbide"

cdn = Xraylib.GetCompoundDataNISTByName(compound)
density = cdn['density'] #g/cm3
thickness = 0.01 #cm
energy = 50.0 #keV

mu_rho = Xraylib.CS_Total_CP(compound, energy)*density

transmitted = 0
total = 100000

total.times {|i|
  x = -Math.log(rand())/mu_rho
  transmitted += 1 if x > thickness
}

printf("transmitted: %i\n", transmitted)
printf("MC fraction: %f\n", Float(transmitted)/total)
printf("True fraction: %f\n", Math.exp(-mu_rho*thickness))
```

xraylib:
Additional interfaces and applications

xraylib: web interface

- Website built on top of *xraylib's* PHP bindings
- Provides syntax examples for all supported bindings
- Hosted at http://lvserver.ugent.be/xraylib-web
- Soon to be replaced!


xraylib-web.tomschoonjans.eu
Github
Facebook
Deutsche Bank
deredactie.be
Wikipedia
De Standaard Online
YouTube
intel fortran forums
De Morgen
xraylib
xraylib
Home
About
Plot
Periodic Table
diamond
Xraylib App
Xraylib Function:
Fluorescence Line Energy
Search...
Element:
26
Transition:
IUPAC
Siegbahn
K
←
L1
All
Submit
Initial development and design by Katherine Rowlinson
Maintained by Tom Schoonjans
Built using xraylib 3.3.0. Development occurs at the xraylib-web Github repository

xraylib: DAWN and GDA

- *xraylib's* Java implementation is now used in DAWN and GDA: Operations, Views, Tools, …
- x*raylib's* Python bindings support available in DAWN!

Xraylib Database

H He
Li Be B C N O F Ne
Na Mg Al Si P S Cl Ar
K Ca Sc Ti V Cr Mn Fe Co Ni Cu Zn Ga Ge As Se Br Kr
Rb Sr Y Zr Nb Mo Tc Ru Rh Pd Ag Cd In Sn Sb Te I Xe
Cs Ba Lu Hf Ta W Re Os Ir Pt Au Hg Tl Pb Bi Po At Rn
Fr Ra Lr Rf Db Sg Bh

La Ce Pr Nd Pm Sm Eu Gd Tb Dy Ho Er Tm Yb
Ac Th Pa U Np Pu Am Cm Bk Cf Es Fm Md No

Fe

Atomic number: 26

Atomic weight: 55.85 g/mol

Density: 7.874 g/cm³

▾ Absorption edge energies

Shell	Energy (keV)
K	7.112
L1	0.8461
L2	0.7211
L3	0.7081
M1	0.0929
M2	0.054
M3	0.054
M4	0.0036
M5	0.0036

▾ XRF line energies

Transition	Energy (keV)

About Close

XAS Sample Calculation

Composition

Compound Formula: CuSO4

Area density (g/cm²): 0.018000000000000002

Density (g/cm³): 8.96

Thickness (µm): 50.0

Estimate

Dilute?

Matrix Formula: H2O

Matrix Density (g/cm³): 1.0

Concentration: 1.0E-4 millimolar

Atmosphere Filter?

Filter material: Air, Dry (near sea level)

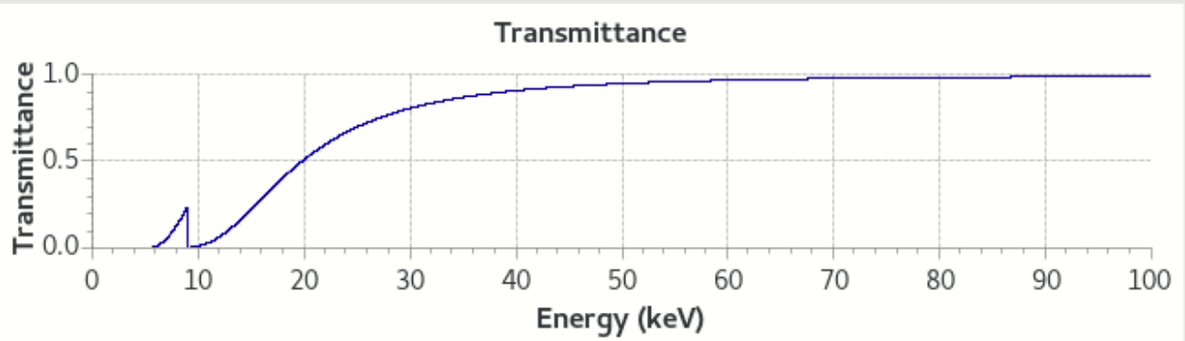
Filter Density (g/cm³): 0.001205

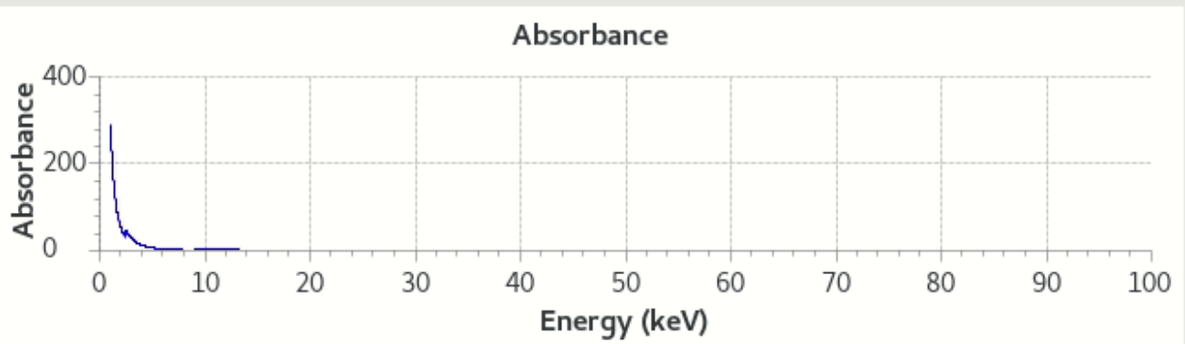
Attenuation path length is equal to sample-detector distance!

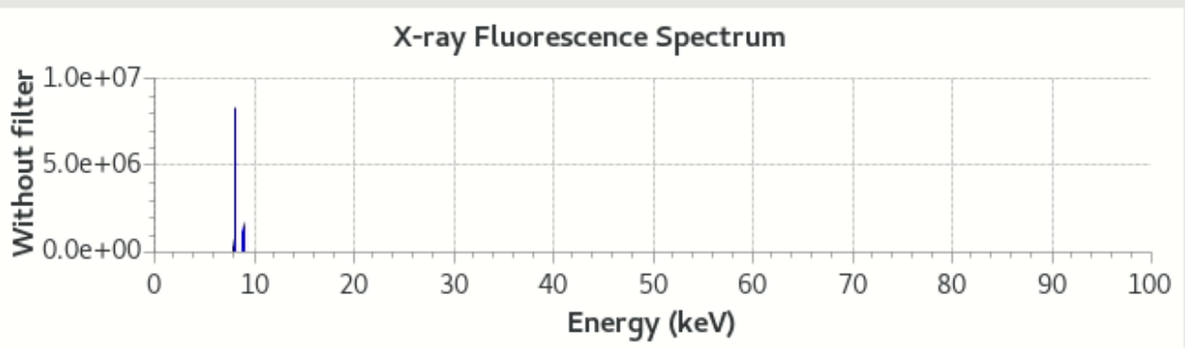
Detector Filter?

Filter material: Polyethylene Terephthalate (Mylar)

Filter Density (g/cm³): 1.4







Total intensity without filters: 4.82900e+07 counts/s

Element	Line	Energy (keV)	Counts	Counts with filter
S	KL2 (Kα2)	2.30660	16607.2	
S	KL3 (Kα1)	2.30780	32965.8	
S	KM2 (Kβ3)	2.46400	1467.39	
S	KM3 (Kβ1)	2.46400	2917.43	
Cu	KL2 (Kα2)	8.02790	1.42664e+07	

Export to XMI-MSIM

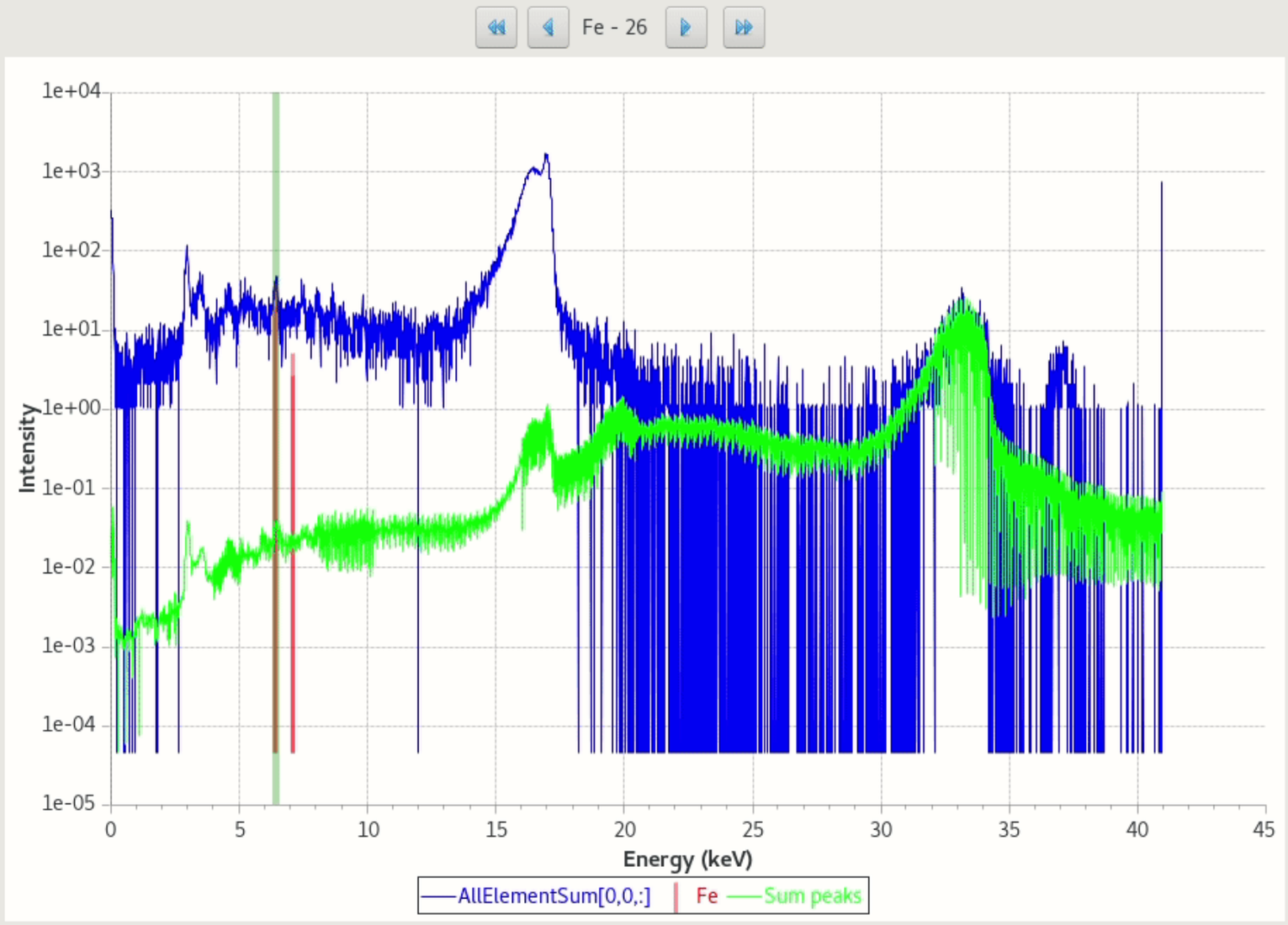
Status: Ok!

Close

Operation Model Configuration

Region of interest based elemental maps of 2D XRF datasets

Periodic Table | Analyze existing spectrum



- (•) Use current spectrum
- ( ) Use sum spectrum
- ( ) Use maximum pixel spectrum
- ( ) Use external spectrum

- [x] Kα
- [ ] Kβ
- [ ] Lα
- [ ] Lβ
- [ ] Mα

No escape peaks ▾ | [x] Show sum peaks | Energy unit is 1 keV ▾

Selected XRF linegroups

Fe-Kα

Remove linegroup

Min excitation energy (keV) 1.0

Max excitation energy (keV) 50.0

ROI width (# channels) 10

Cancel | Finish

Under development

Error handling

- Current implementation is outdated and useless
- Check return values: 0 indicates error or unavailable
- Not thread-safe!
- Consider calling `SetErrorMessages(0)`at the start of your program

Error handling

- Inspired by GLib's GError…
- Optional for C, C++, Fortran
- Translated into exceptions for all bindings!

```c
typedef enum {
    XRL_ERROR_MEMORY, /* set in case of a memory allocation problem */
    XRL_ERROR_INVALID_ARGUMENT, /* set in case an invalid argument gets passed to
a routine */
    XRL_ERROR_IO, /* set in case an error involving input/output occurred */
    XRL_ERROR_TYPE, /* set in case an error involving type conversion occurred */
    XRL_ERROR_UNSUPPORTED, /* set in case an unsupported feature has been
requested */
    XRL_ERROR_RUNTIME /* set in case an unexpected runtime error occurred */
} xrl_error_code;

typedef struct {
    xrl_error_code code;
    char *message;
} xrl_error;

double LineEnergy(int Z, int line, xrl_error **error);

xrl_error *error = NULL;
double energy = LineEnergy(26, M5N7_LINE, &error);
if (error != NULL) {
    fprintf(stderr, "Error: %s\n", error->message);
}
```

xraylib publications

A. Brunetti, M. Sanchez del Rio, B. Golosio,
A. Simionovici and A. Somogyi,
Spectrochim. Acta Part B, 59, 1725-1731, 2004

T. Schoonjans, A. Brunetti, B. Golosio,
M. Sanchez del Rio, V. A. Solé, C. Ferrero
and L. Vincze,
Spectrochim. Acta Part B, 66, 776–784, 2011

Acknowledgements

- Laszlo Vincze
- Manuel Sanchez del Rio
- Bruno G
- Antonio Brunetti
- Alexandre Simionovici
- V. Armando Solé
- Claudio Ferrero
- David Sagan
- ormington
- Teemu Ikonen
- Everyone that ever contributed code or reported bugs!

Thank you!