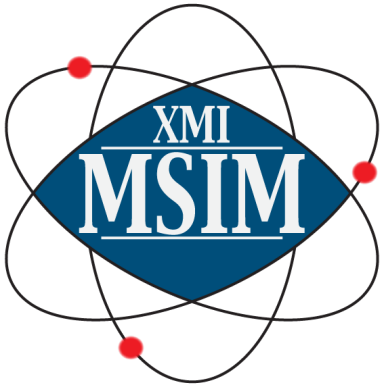

Ghent University, Belgium

University of Sassari, Italy

European Synchrotron Radiation Facility,
Grenoble, France

Diamond Light Source,
Didcot, United Kingdom

XMI-MSIM

The definitive manual

Version 8.0

Tom Schoonjans

February 7, 2020

Contents

1 Introduction **3**

2 Installation instructions **3**
2.1 Compiling from source . . . 3
2.1.1 Compilation stages . . . 4
2.1.2 Preparing the precompiled dataset . . . 5
2.1.3 Note on the random number generators . . . 5
2.2 Linux . . . 5
2.2.1 Fedora, Centos and Scientific Linux . . . 6
2.2.2 Debian and Ubuntu . . . 6
2.3 Windows . . . 7
2.4 macOS . . . 8

3 User guide **8**
3.1 Launching XMI-MSIM . . . 8
3.2 Creating an input-file . . . 10
3.2.1 General . . . 10
3.2.2 Composition . . . 10
3.2.3 Geometry . . . 16
3.2.4 Excitation . . . 20
3.2.5 Beam and detection absorbers . . . 21
3.2.6 Detector settings . . . 22
3.3 Saving an input-file . . . 24
3.4 Starting a simulation . . . 24
3.4.1 Control panel . . . 24
3.4.2 Executable . . . 25
3.4.3 Options . . . 26
3.4.4 Export results . . . 27
3.4.5 During a simulation . . . 27
3.5 Visualizing the results . . . 29
3.5.1 Plot canvas . . . 29
3.5.2 Net-line intensities . . . 29
3.5.3 Exporting the plot canvas . . . 30
3.6 Global preferences . . . 31
3.6.1 Simulation defaults . . . 31
3.6.2 Updates . . . 32
3.6.3 User-defined layers . . . 32
3.6.4 Advanced . . . 32
3.7 Checking for updates . . . 32
3.8 Command line interface . . . 32
3.9 Example files . . . 33

4 Advanced usage **34**
4.1 X-ray sources . . . 34
4.1.1 X-ray tube spectrum generator . . . 34
4.1.2 Radionuclides . . . 36
4.1.3 Custom sources . . . 37
4.2 Batch simulations . . . 39
4.2.1 Batch simulations: simulate a number of unrelated input-files . . . 39
4.2.2 Batch simulations: vary one or two parameters in a single input-file . . . 40
4.3 Generate XRMC input-files . . . 43
4.4 Using the XMI-MSIM API from Python . . . 44
4.5 Custom detector response functions . . . 47
4.5.1 Building modules on macOS . . . 49
4.5.2 Building modules on Linux . . . 50
4.5.3 Building modules on Windows . . . 50

5 The XMI-MSIM API: list of functions **51**

6 References and addtional resources **52**
6.1 Papers by Laszlo Vincze et al. . . . 52
6.2 Papers by Tom Schoonjans et al. . . . 52
6.3 Posters by Tom Schoonjans et al. . . . 53
6.4 Oral presentations by Tom Schoonjans et al. . . . 53

1 Introduction

XMI-MSIM is an open source tool designed for predicting the spectral response of energy-dispersive X-ray fluorescence spectrometers using Monte Carlo simulations. It comes with a fully functional graphical user interface in order to make it as user friendly as possible. Considerable effort has been taken to ensure easy installation on all major platforms.
Development of this package was part of my PhD thesis. The algorithms were inspired by the work of my promotor Prof. Laszlo Vincze of Ghent University. Links to his and my own publications can be found in this manual.
A manuscript has been published in Spectrochimica Acta Part B that covers the algorithms that power XMI-MSIM. Please include a reference to this publication in your own work if you decide to use XMI-MSIM for academic purposes.
A second manuscript was published that covers our XMI-MSIM based quantification plug-in for PyMca.
XMI-MSIM is released under the terms of the GPLv3.

2 Installation instructions

2.1 Compiling from source

XMI-MSIM has been successfully built on Linux (Debian/Ubuntu and RHEL/CentOS/Fedora), macOS (High Sierra and up) and Windows 7 (with the 64-bit MinGW-w64 compilers installed using MSYS2). Obtain the source code from **our download repository**, kindly hosted by the X-ray Microspectroscopy and Imaging research group of Ghent University.
The following dependencies are required to build XMI-MSIM:

- fortran 2003 compiler (gfortran >= 4.4, Intel Fortran are known to work)
- C compiler with OpenMP support (gcc and clang). The native Mac OS X version requires that the compiler supports Objective-C as well. When compiling the GUI, a C++ compiler becomes an additional requirement.
- HDF5
- libxml2
- libxslt
- Fortran GSL bindings (FGSL) or easyRNG
- xraylib 4.0.0+ (including Fortran bindings)
- glib2
- GTKMM3, Gtkmm-PLplot and libpeas (1.22.0+) for the graphical user interface (optional though highly recommended)
- optional for the GUI: json-glib, libsoup (Linux only)

- MPI (OpenMPI or Intel MPI): optional. Recommended for those that want to perform brute-force simulations with a very high number of simulated photons

All dependencies should be easy to obtain, with the exception of those projects I manage personally:
Windows users will have to compile most of these dependencies themselves, which will require them to install a bash shell with all basic UNIX utilities. Our 64-bit version was built with MSYS2 and its GCC packages.
It is absolutely critical that all Fortran packages are compiled with the exact same compiler, and this same compiler also needs to be used when building XMI-MSIM.

2.1.1 Compilation stages

Unpack the tarball:

```
tar xvfz xmimsim-x.y.tar.gz
cd xmimsim-x.y
```

Configure the source tree by examining the capabilities of the host system:

```
./configure
```

The configure command has a long list of options. You can have a look at them by executing:

```
./configure --help
```

A commonly used option is to change the installation destination: this can be accomplished by using the `--prefix` option. If your Fortran compiler does not have a standard name, you may have to specify it as an option to configure such as `FC=gfortran-9`. Packages that are not installed in default locations, may not be detected by the configure script and could result in the configure script aborting prematurely. This is particularly likely for packages like xraylib and/or fgsl that are installed in /usr/local. Such a problem can be avoided by setting the PKG_CONFIG_PATH environment variable manually:

```
export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
```

If the configure script terminates without error, try building the code by running:

```
make
```

It is not recommended to invoke make with the -j option, as it may confuse the fortran compiler.
After compilation, install the program using:

```
make install
```

This may have to be executed with root privileges.

2.1.2 Preparing the precompiled dataset

XMI-MSIM loads, before running a simulation, datasets of precomputed physical data into memory, depending on the exact parameters in the input file. These datasets are mostly inverse cumulative distribution functions of scattering cross-sections, which would be very computationally expensive to calculate during the simulation itself based on the corresponding probability density functions.
This file is generated using the `xmimsim-db` executable, which produces a file called `xmimsimdata.h5` in the current working directory. After this file is created, which can take up to half an hour on slower machines, copy it to the data folder of your XMI-MSIM installation. Assuming the default installation destination was not altered, this would be /usr/local/share/xmimsim.

2.1.3 Note on the random number generators

XMI-MSIMs random number generators are seeded on Mac OS X and Linux using high quality noise produced by /dev/urandom. The seeds can be collected in two ways:

1. The user launches the `xmimsim-harvester` daemon, which will collect seeds at frequent intervals and pass them along to XMI-MSIM when requested. The daemon is ideally started at boottime (using some initd script), or on Mac OS X, by copying the `be.ugent.xmi.harvester.plist` file from its installation location (*prefix*/Library/LaunchDaemons) to /Library/LaunchDaemons and subsequently invoking `sudo launchctl load /Library/LaunchDaemons/be.ugent.xmi.harvester.plist`. It should be noted that the daemon is buggy, and it is generally not recommended to use this solution.
2. If the daemon is not running, then a separate thread is launched in XMI-MSIM at runtime which takes care of harvesting the seeds (a bit slower, but reliable).

2.2 Linux

Keep in mind that we only provide packages for distributions that are currently officially supported!

2.2.1 Fedora, Centos and Scientific Linux

To facilitate the installation on RPM based Linux distributions, the package includes a spec file which can be used to produce RPM packages for linux distributions that support them (Fedora, Red Hat etc). The developers have built 64-bit RPM packages of XMI-MSIM for the officially supported Fedora and Redhat EL/CentOS/Scientific Linux 7/8 distributions. These can be downloaded from the RPM repository that is hosted by the X-ray Microspectroscopy and Imaging research group of Ghent University. Access to this repository can be obtained as follows for Fedora distros:

```
su -c 'rpm -Uvh http://lvserver.ugent.be/yum/xmi-repo-key-fedora.noarch.rpm'
```

for Red Hat EL 7 based distributions:

```
su -c 'rpm -Uvh http://lvserver.ugent.be/yum/xmi-repo-key-7.0-1.el7.noarch.rpm'
```

and for Red Hat EL 8 based distributions:

```
su -c 'rpm -Uvh http://lvserver.ugent.be/yum/xmi-repo-key-8.0-1.el8.noarch.rpm'
```

The XMI-MSIM packages themselves can then be downloaded using yum:

```
su -c 'yum install xmimsim'
```

Updates can be installed in a similar way:

```
su -c 'yum update xmimsim'
```

The OpenCL plugin can be installed using:

```
su -c 'yum install xmimsim-opencl'
```

For this to work you will need to install OpenCL drivers that match your videocard.

2.2.2 Debian and Ubuntu

Packages were created for Debian and Ubuntu. Currently the following flavors are supported: Debian Stretch and several Ubuntu versions.
In order to access these packages using your favorite package manager, execute the following command to import our public key:

```
curl http://lvserver.ugent.be/apt/xmi.packages.key | sudo
apt-key add -
```

Next, add the package download location corresponding to your distribution to the /etc/apt/sources.list file (as root):
Debian Buster:

```
deb http://lvserver.ugent.be/apt/debian buster stable
deb-src http://lvserver.ugent.be/apt/debian buster stable
```

Ubuntu Bionic 18.04:

```
deb [arch=amd64] http://xmi-apt.tomschoonjans.eu/ubuntu bionic stable
deb-src http://xmi-apt.tomschoonjans.eu/ubuntu bionic stable
```

Ubuntu Eoan 19.10:

```
deb [arch=amd64] http://xmi-apt.tomschoonjans.eu/ubuntu eoan stable
deb-src http://xmi-apt.tomschoonjans.eu/ubuntu eoan stable
```

When the sources.list file contains the correct download locations, update the apt cache by running:

```
sudo apt-get update
```

After this, one can install XMI-MSIM by executing the following command:

```
sudo apt-get install xmimsim
```

The OpenCL plugin can be installed using:

```
sudo apt-get install libxmimsim-ocl
```

For this to work you will need to install OpenCL drivers that match your videocard.

2.3 Windows

Installers containing the 64-bit binaries of XMI-MSIM for the Windows platform can be found in the **Downloads** section. It will download and install *xraylib* if necessary.
The Windows releases ship with the OpenCL plug-in for XMI-MSIM. In order for this plug-in to function, you need 1) to have a videocard that supports OpenCL 1.1 and 2) have the OpenCL drivers installed as provided by your videocard's manufacturer. To confirm that this is indeed the case, try running XMI-MSIM with GPU support enabled: if it fails, an error message will be shown in the log box in red, followed by a fallback to the default Fortran implementation.
Nightly builds for Windows can be downloaded here. These installers will download the main HDF5 data file, which may take quite some time, depending on the connection speed. Keep in mind that these are development snapshots, and that they should not be considered 'stable'. If you run into bugs or unexpected behavior while using these nightly builds, I would be grateful if you could let me know.

2.4 macOS

A **dmg** file has been created containing an application bundle which integrates nicely within Mac OS X, through the use of some dedicated API's. The provided app will run on macOS Yosemite and newer (all 64-bit Intel only). After downloading, mount the dmg file and drag the XMI-MSIM app to the Applications folder.
Homebrew can also be used to install a more UNIX-like version of XMI-MSIM, allowing you to run the GUI and other executables from the command-line, and is required when installing XRMC with its XMI-MSIM plug-in.
In order to install XMI-MSIM using Homebrew type in a terminal:

```
brew install tschoonj/tap/xmi-msim
```

To install XRMC with XMI-MSIM support (in this case the previous command does not have to be executed since XMI-MSIM will be installed first as a dependency):

```
brew install tschoonj/tap/xrmc --with-xmi-msim
```

3 User guide

In this section a short manual is presented that should allow users to get started with XMI-MSIM. Although the screenshots were obtained on a Mac, they should be representative for Windows and Linux as well. Significant differences will be indicated. The following guide assumes that the user has already installed XMI-MSIM, according to the Installation instructions.

3.1 Launching XMI-MSIM

For macOS users: assuming you dragged the app into the Applications folder, use Finder or Spotlight to launch XMI-MSIM. If you get a dialog complaining about XMI-MSIM being untrusted, you will need to right-click the app icon and click 'Open' where there will be an option to open the app anyway. Afterwards, you will not have to do this again. The warning is due to me not signing the app with an Apple issued certificate, for which I would need to pay $100 US a year...
For Windows users: an entry should have been added to the Start menu. Navigate towards it in *Programs* and click on XMI-MSIM.
For Linux users: an entry should have been added to the Education section of your Start menu. Since this may very considerably depending on the Linux flavor that is being used, this may not be obvious at first. Alternatively, fire up a terminal and type:

```
xmimsim-gui
```

Your desktop should now be embellished with a window resembling the one in the following screenshot.

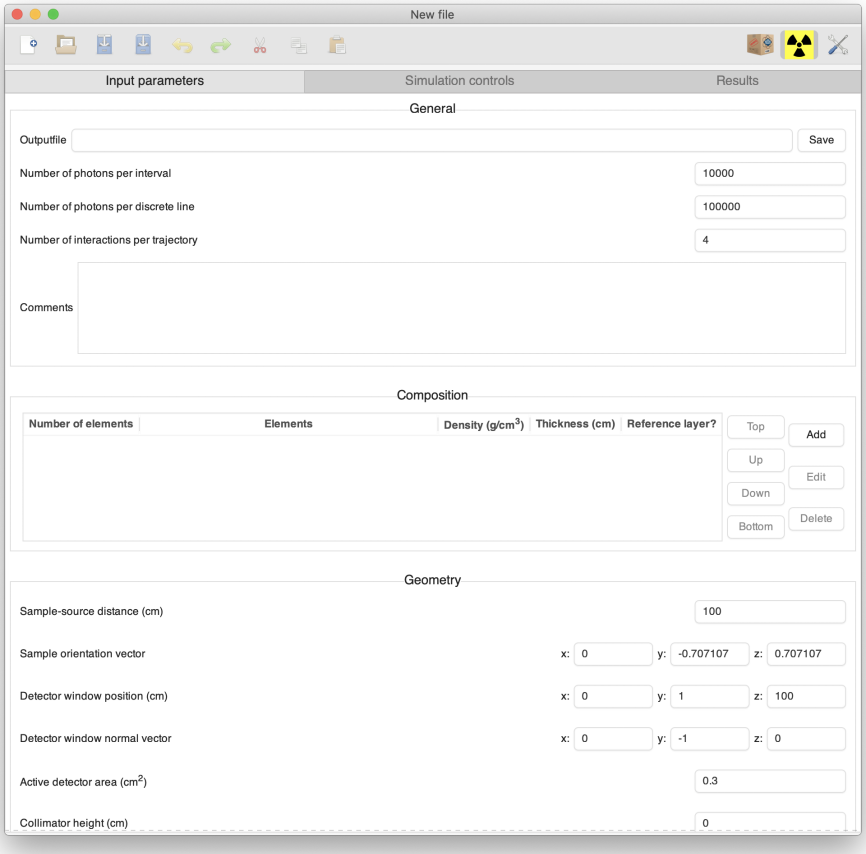

Figure 1: XMI-MSIM on startup

XMI-MSIM may also be started on most platforms by double clicking XMI-MSIM input-files (.xmsi extension) and output-files (.xmso extension) in your platform's file manager, thereby loading the file's contents.
The main window of the XMI-MSIM GUI consists of three pages that each serve a well-defined purpose. The first page is used to generate inputfiles, based on a number of parameters that are defined by the user. The second page allows for the execution of these files, while the third and last page is designed to visualise the results and help in their interpretation. The purpose of the following sections is to provide an in-depth guide on how to operate these pages.
When starting XMI-MSIM without providing a file to open, a new file will

be started with default settings. The same situation can be obtained at any moment by clicking on *New* in the toolbar.

3.2 Creating an input-file

The first page consists of a number of frames, each designed to manipulate a particular part of the parameters that govern a simulation.

3.2.1 General

The *General* section contains 4 parameters:

- Outputfile: clicking the *Save* button will pop up a file chooser dialog, allowing you to select the name of the outputfile that will contain the results of the simulation
- Number of photons per discrete line: the excitation spectrum as it is used by the simulation may consist of a number of discrete components with each a given energy and intensity (see Excitation for more information). This parameter will determine how many photons are to be simulated per discrete line. The calculation time is directly proportional to this value
- Number of photons per interval: the excitation spectrum as it is used by the simulation may consist of a number of continuous interval components defined by the given energies and intensity densities at the beginning and the end of the intervals (see Excitation for more information). This parameter will determine how many photons are to be simulated per interval. The calculation time is directly proportional to this value
- Number of interactions per trajectory: this parameter will determine the maximum number of interactions a photon can experience during its trajectory. It is not recommended to set this value to higher than 4, since the contribution of increasingly higher order interactions to the spectrum decreases fast. The calculation time is directly proportional to this value
- Comments: use this textbox to write down some notes you think are useful.

3.2.2 Composition

This interface allows you to define the system that will make up your sample and possibly its environment. XMI-MSIM assumes that the system is defined as a stack of parallel layers, each defined by its composition, thickness (measured along the Sample orientation vector) and density. Adding layers can be accomplished by simply clicking the *Add* button. A dialog will pop up as seen in the following screenshot:

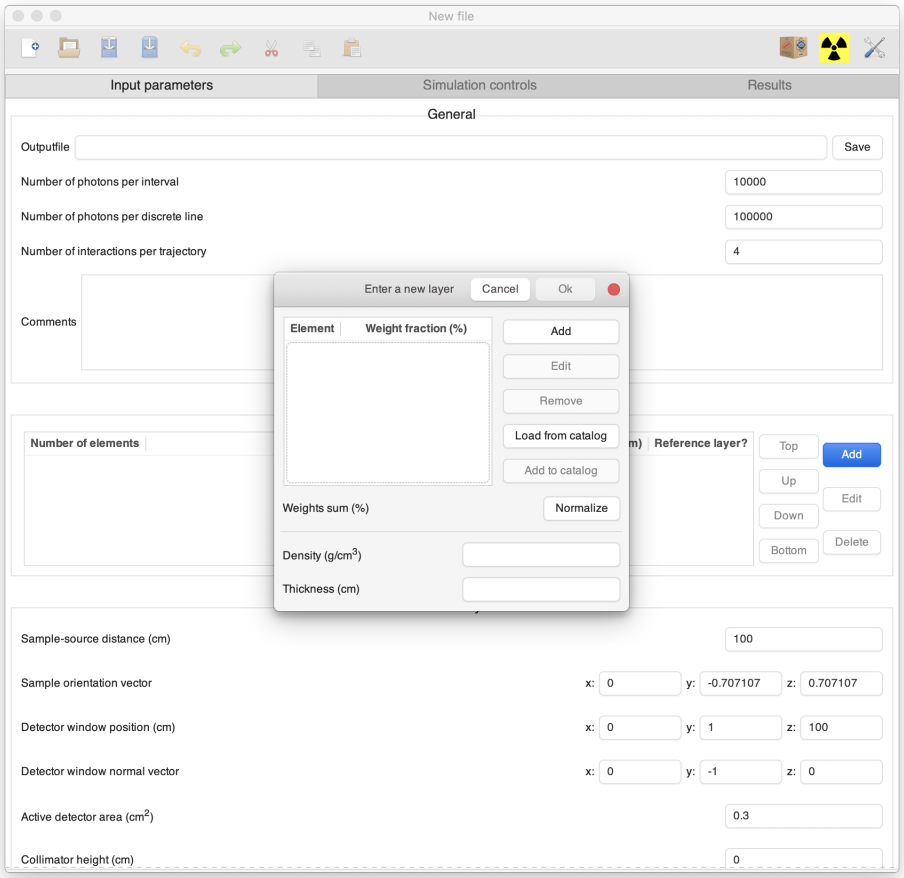

Figure 2: Defining a new layer

The different elements that make up the layer are added by clicking on the *Add* button. A small dialog will emerge, enabling you to define a compound or a single element, with its corresponding weight fraction. In the following screenshot, I used `CuSO4` with a weight fraction of 50 % to start with.
You may wonder at exactly which chemical formulas are accepted by the interface. Well the answer is: anything that is accepted by *xraylib*'s CompoundParser function. This includes formulas with (nested) brackets such as: `Ca10(PO4)3OH` (apatite). Invalid formulas will lead to the *Ok* button being greyed out and the *Compound* text box gaining a red background.
After clicking ok, you should see something resembling the following screenshot:
You will notice that the compound has been parsed and separated into its constituent elements, with weight fractions according to the mass fractions of the elements. In this example I added an additional 50 % of `U3O8` to the

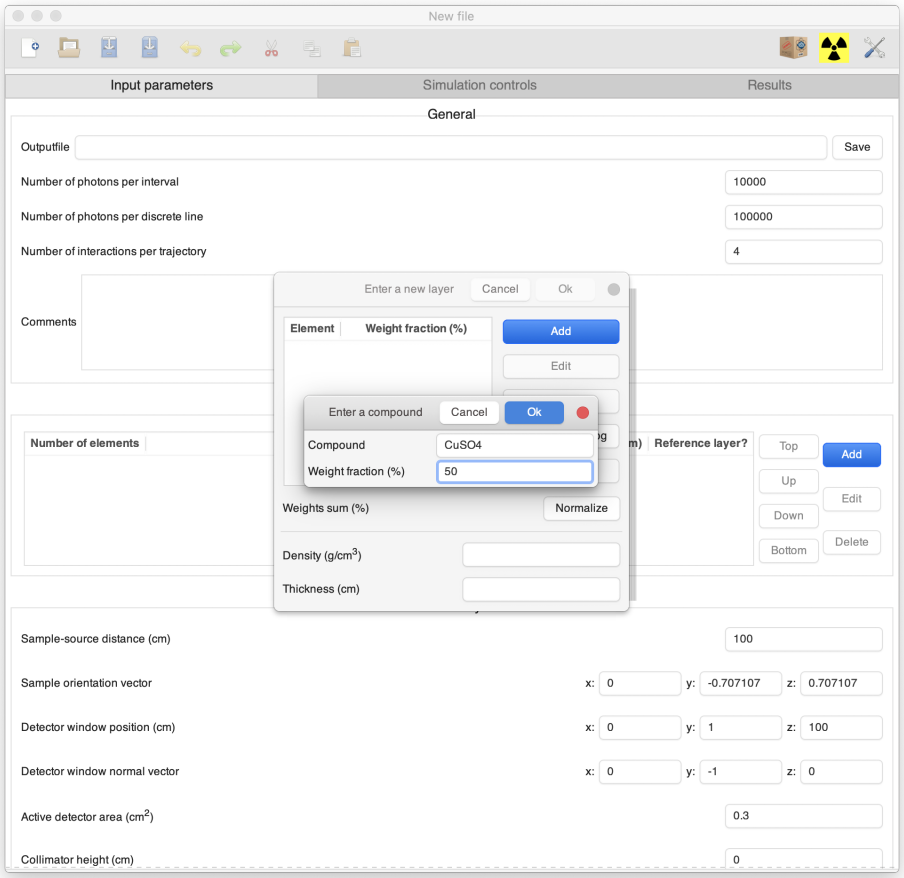

Figure 3: Adding a compound

composition and picked the values 2.5 g/cm3 and 1 cm for density and thickness, respectively, leading to a weights sum of 100 %. It is considered good practice to have the weights sum equal to 100 %. This can be accomplished by either adding/editing/removing compounds and elements from the list, or by clicking the *Normalize* button, which will scale **all** weight fractions in order to have their sum equal to 100 %. Your dialog should match with this screenshot:
Alternatively you may consider looking into the builtin catalog (press *Load from catalog*): a pop-up window will allow you to select a compound from one of two lists. The first list is generated using xraylib's GetCompoundDataNISTList function, which provides access to NISTs compound database of compositions and densities. The second one however, provides access to layers that you defined yourself: when a valid layer (i.e. composition, density and thickness) is showing in the layer dialog, click *Add to catalog* and choose

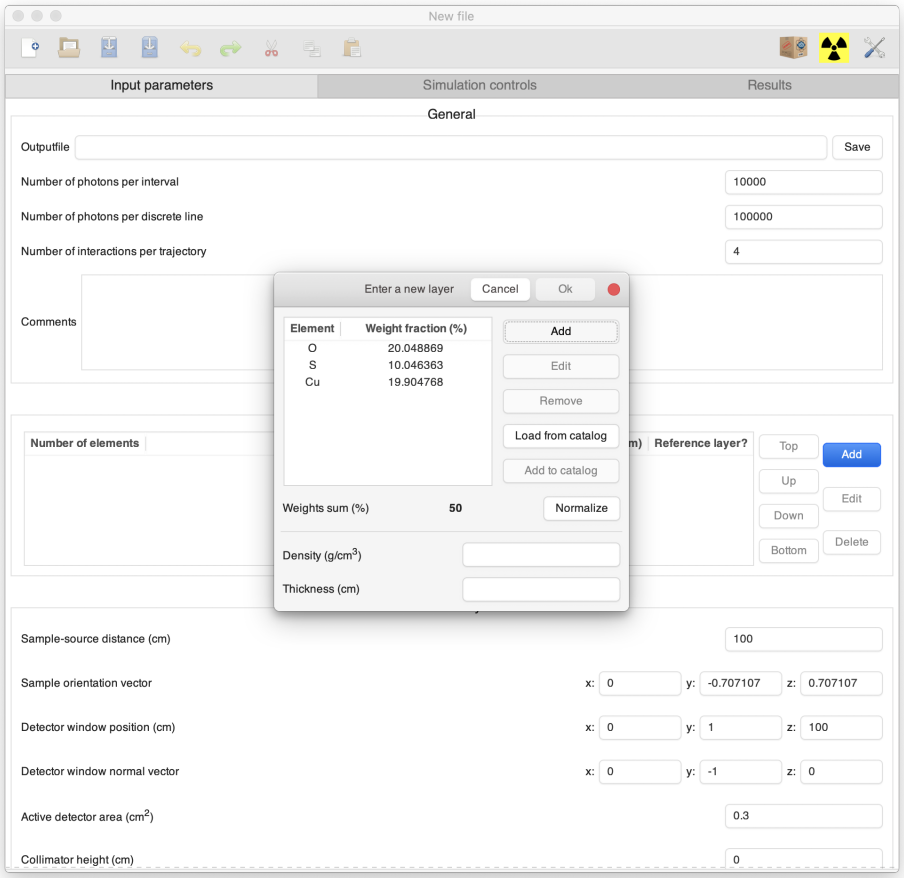

Figure 4: Adding a compound

a name for the layer: this layer will show up in the catalog list next time it is opened. Keep in mind that existing layers in the list will be overwritten without warning! If you would like to delete previously defined layers from the list, use the *Preferences* interface.
When satisfied with the layer characteristics, press *Ok*.
X-ray fluorescence experiments are quite often performed under atmospheric conditions. If so, it is of crucial importance to add the atmosphere to the system for several reasons:

1. The atmosphere attenuates the beam and the X-ray fluorescence
2. The intensity of the Rayleigh and Compton scatter peaks is greatly influenced by the atmosphere
3. The photons from the beam as well as the fluorescence and the scattered photons will lead to the production of Ar-K fluorescence, a common

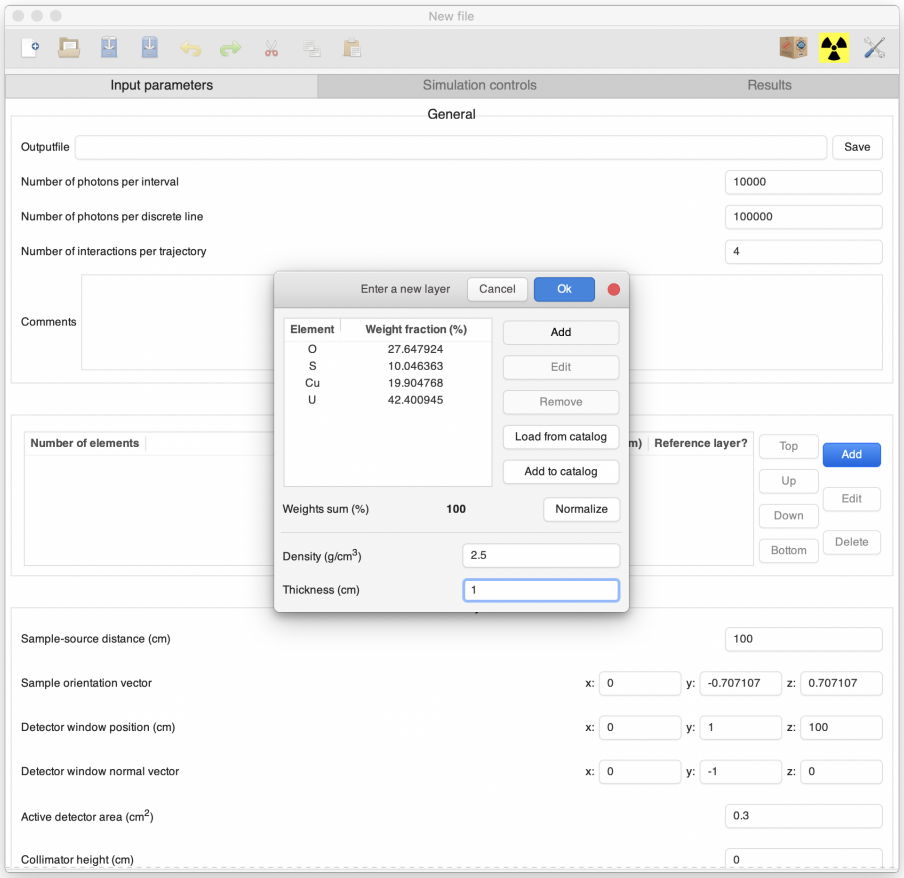

Figure 5: Adding another compound

artefact in X-ray fluorescence spectra. In some rare cases, one may even detect Xe fluorescence.

To add such a layer, click again on *Add* button. In the *Modify layer* dialog, add the composition, density and thickness of the air layer. This is shown in the next screenshot:
Alternatively, click on *Load from catalog* and select *Air, Dry (near sea level)* from the NIST compositions list. Clicking the *Ok* button should produce the following situation in the *Composition* section:
However, the ordering of the layers in the table is wrong: **XMI-MSIM assumes that the layers are ordered according to distance from the X-ray source**. This means that the first layer is closest to the source and all subsequent layers are positioned at increasingly greater distances from the source. This can be easily remedied by selecting a layer and then moving

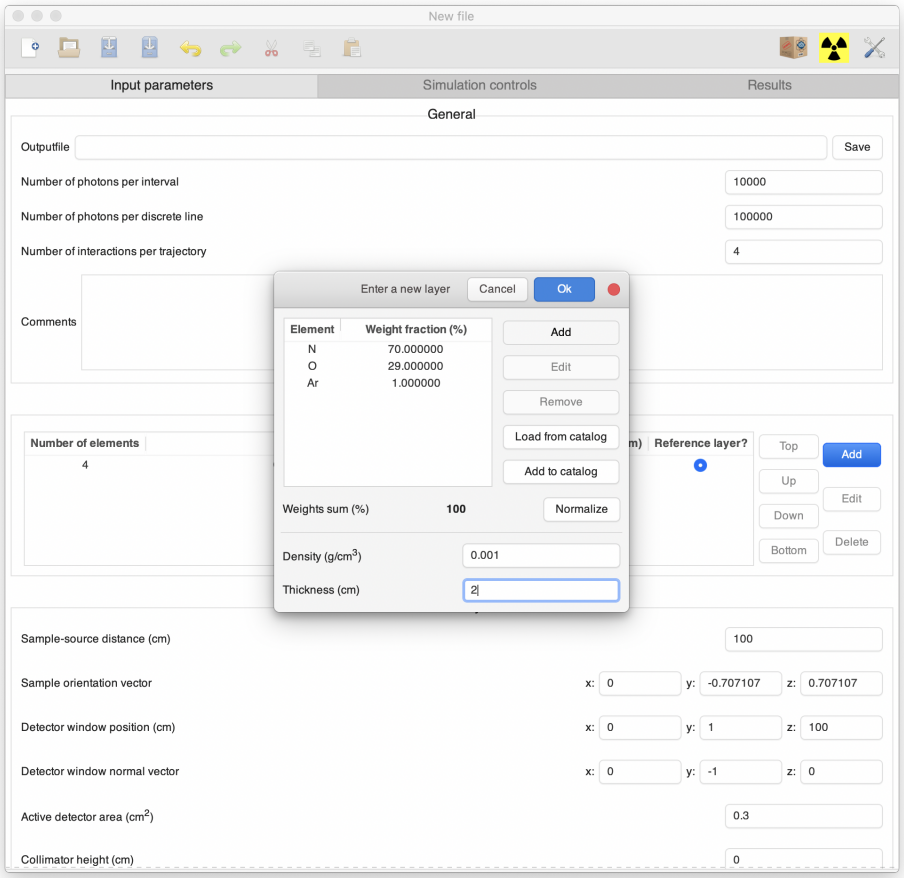

Figure 6: Adding air layer

it around using the *Top*, *Up*, *Down* and *Bottom* buttons. The following screenshot shows the corrected order of the layers:
An important parameter in this table is the *Reference layer*. Using the toggle button, you select which layer corresponds to the one that is considered to be the first layer of the actual *sample*. In most cases, this will indicate the first non-atmospheric layer. The *Reference layer* is also the layer that is used to calculate the *Sample-source distance* in the *Geometry* section.
Layers can be removed by selecting them and then clicking the *Remove* button. Existing layers may be modified by either double-clicking the layer of interest or by selecting the layer, followed by clicking the *Edit* button.
Keep in mind that the number of elements influences the computational time greatly, especially when dealing with high Z-elements that may produce L- and M-lines.

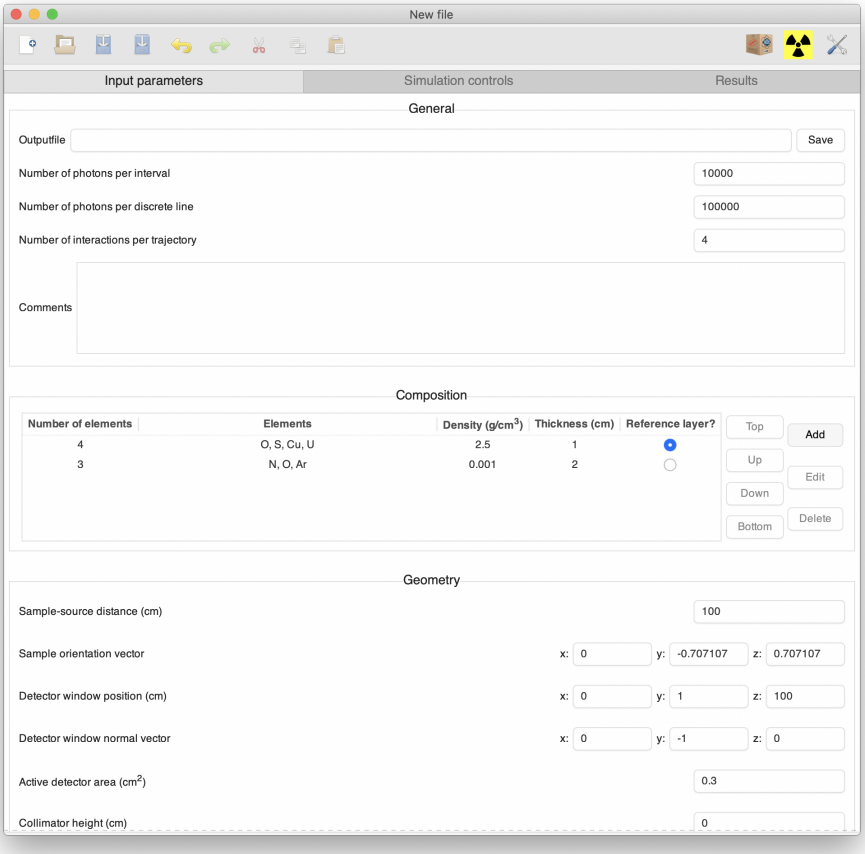

Figure 7: Wrong layer order

3.2.3 Geometry

Scrolling down a little on the *Input parameters* page reveals the *Geometry* section as shown in the next screenshot:
This sections covers the position and orientation of the system of layers, detector and slits. In order to fully appreciate the geometry parameters, it is important that I first describe the coordinate system that these position coordinates and directions are connected to:

- The coordinate system is right-handed Cartesian
- The origin of the coordinate system corresponds to the position of the source
- The z-axis is aligned with the beam direction and points from the source towards the sample
- The y-axis defines, along with the z-axis, the horizontal plane

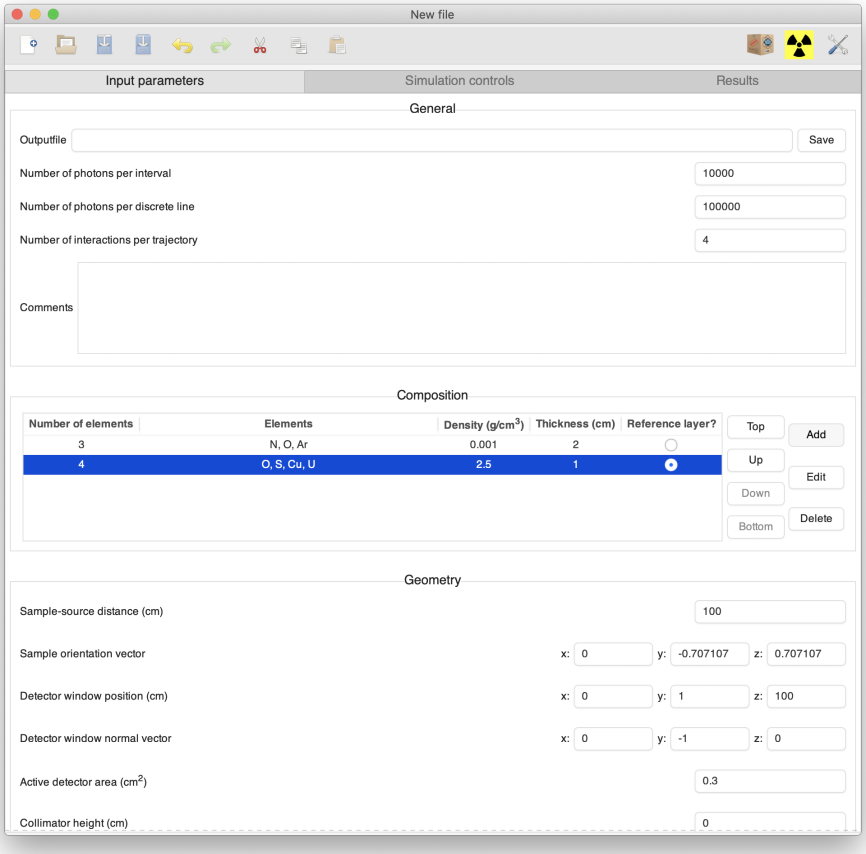

Figure 8: Correct layer order

- The x-axis emerges out from the plane formed by the y- and z-axes

This is demonstrated in the following figure:
Now with this covered, let us have a look at the different *Geometry* parameters:

- Sample-source distance: the distance between the source and the *Reference layer* in the system of layers as defined in the *Composition section*
- Sample orientation vector: the normal vector that determines the orientation of the stack of layers that define the sample and its environment. The z component must be strictly positive
- Detector window position: the position of the detector window. This is seen as the point where the photons are actually detected and terminated by the detector. Keep this in mind when defining a collimator

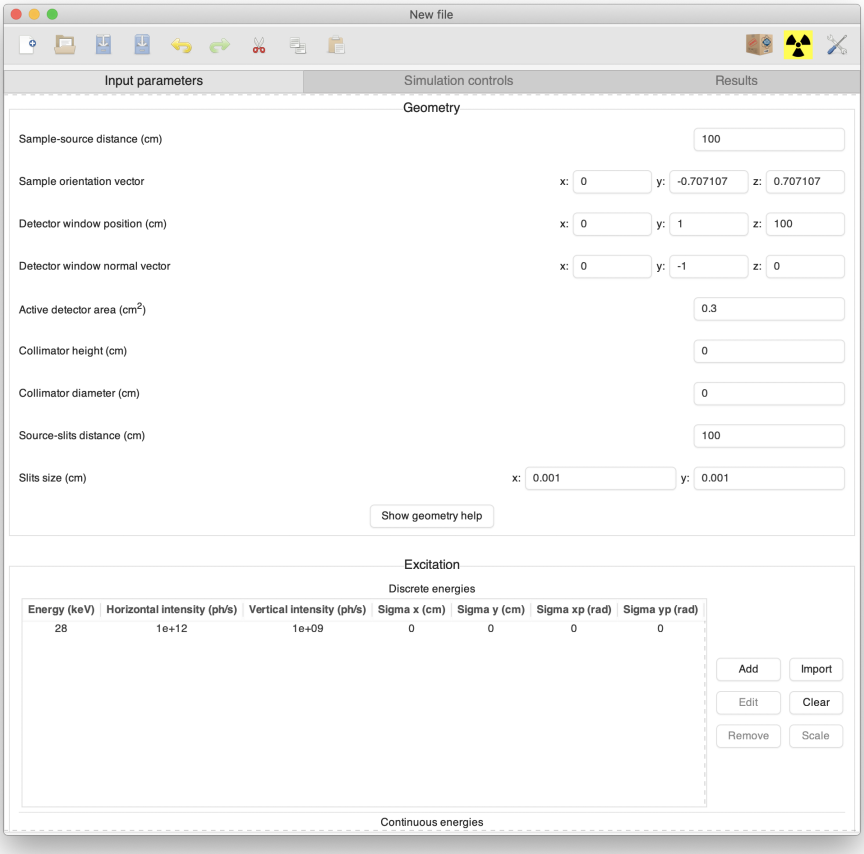

Figure 9: Geometry, excitation and beam absorbers

- Detector window normal vector: the normal vector of the detector window. Should be directed towards the sample (unless you have a very good reason not to do so)
- Active detector area: this corresponds to the area of the detector window that is capable of letting through *detectable* photons. Should be provided by the manufacturer of your detector
- Collimator height: XMI-MSIM allows for the definition of a conical detector-collimator whose properties are determined by this parameter and the *Collimator diameter*. Setting either to zero corresponds to a situation without collimator. This height parameter is seen as the height of the cone, measured from the detector window to the opening of the collimator, along the detector window normal vector
- Collimator diameter: diameter of the opening of the conical detector

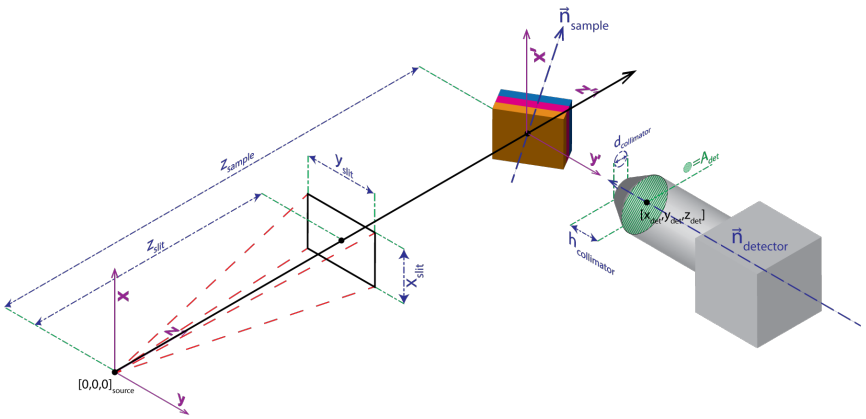

Figure 10: Schematic representation of the geometry

collimator. The base of the collimator corresponds to the *Active detector area*

- Source-slits distance: XMI-MSIM defines a set of virtual slits, whose purpose is to define the size of the beam at a given point, based on the distance between these slits and the X-ray source, as well as the *Slits size*, defined by the next parameter. I recommend to have the *Source-slits distance* correspond to the *Sample-source distance*, since this way the beam, upon hitting the *Reference layer*, will have exactly the dimensions specified by *Slits size* (if using a point source!). These slits should not be thought of as physical slits, as they are often encountered in many experimental setups: XMI-MSIM's virtual slits do not reduce the beam intensity (flux) at all! They do impact the flux density which increases when decreasing the *Slits size* parameters, and vice versa.
- Slits size: see previous parameter. Refers to the dimensions of the beam at the *Source-slits* distance. This parameter will be ignored when dealing with a Gaussian source (see Excitation section)

In order to visualize these different parameters, click the *Show geometry help* button: a new window will pop up showing the aforementioned coordinate system. Hovering the mouse over the different components in the new window will have the corresponding widgets light up in green in the main window.

This works both ways: hover the mouse over the geometry widgets in the main window and little boxes will appear in the coordinate system window.

3.2.4 Excitation

Next, there is the *Excitation* section, which is used to define the X-ray beam that irradiates the sample. The corresponding excitation spectrum may consist of a number of discrete components, each with a horizontally and vertically polarized intensity, as well as a number of parameters that define the type and the aperture of the source. Furthermore, one can also insert a number of continuous interval components, defined through a list of intensity densities, each with their horizontally and vertically polarized components. In this case, one has two insert at least two intensity densities in order to have at least one interval.
At runtime, the code will use the *Number of photons per discrete line* and *Number of photons per interval* parameters to determine how many photons will be simulated per discrete component and continuous energy interval component. Adding, editing and removing components is handled through the buttons in the *Excitation* section. For example, we can change the settings of the default value by clicking the *Edit* button. The dialog contains the fields necessary to define a particular component:

- Energy: the energy of this particular part of the excitation spectrum, expressed in keV
- Horizontally and vertically polarized intensities: the number of photons that are polarized in the horizontal and vertical planes, respectively. A completely unpolarized beam has identical horizontal and vertical intensities (such as those produced by X-ray tubes), while synchrotron beams will have very, very low vertically polarized intensities. For information on how to convert the total number of photons given the degree of polarization to the horizontal and vertical polarized intenties, consult Part 5 in our series of papers on Monte-Carlo simulations
- Source size x and y: if both these values are equal to zero, then the source is assumed to be a point source, and the divergence of the beam is completely determined by the *Source-slits distance* and *Slits size* parameters of the *Geometry* section. Otherwise the source is considered a Gaussian source, in which case the photon starting position is chosen according to Gaussian distributions in the x and y planes, determined by the *Source size x* and *Source size y* parameters
- Source divergence x and y: if these values are non-zero, AND the source is Gaussian, then the *Source-slits distance* takes on a new role as it becomes the distance between the actual focus and the source position. In this way a convergent beam can be defined, emitted by a Gaussian source at the origin. For the specific case of focusing on

the sample the *Sample-source distance* should be set to the *Source-slits distance*.

- Energy distribution type: additionally for the discrete components, it is possible to set the *Energy distribution type*, which may assume the values *Monochromatic*, *Gaussian* and *Lorentzian*. The first case assumes that the discrete energy is purely monochromatic and that only the selected energy will be used in the simulation. The two other cases corrspond to a scenario in which the simulation will sample from a Gaussian or Lorentzian distribution respectively. If either of these two cases is selected, the user is expected to provide respectively the standard deviation and the scale parameter.

In this particular case, I have changed the energy to 20.0 keV, and made the beam unpolarized by equalizing both intensities, as shown in the following screen shot. The source remains a point source.
The discrete energies and continuous energies widgets each contain six buttons:

- *Add*: add a new component.
- *Edit*: edit a previously defined component.
- *Remove*: delete previously components.
- *Import*: import a list of discrete lines or continuous energy intensity densities from an ASCII file. These files must consist of either two, three or seven columns, with the first column containing the energies, the second the total intensity (if only two columns are found), or the second and third the resp. horizontally and vertically polarized intensities or intensity densities. If seven columns are encountered, the last four columns are assumed to contain source sizes and divergencies. It is possible through the interface to start reading only at a certain linenumber and also to read only a set number of lines.
- *Clear*: delete all previously defined components.
- *Scale*: multiply the intensities or intensity densities with a positive real number.

The excitation spectrum can also be defined using the X-ray sources dialog.

3.2.5 Beam and detection absorbers

The two following sections deal with absorbers, first absorbers that are optionally placed in the excitation path (for example a sheet of Al or Cu), and next the absorbers that are optionally placed in the detector path. This means that the former will reduce the intensity of the incoming beam, while the latter will reduce the intensity of the photons that hits the detector. It is important to realize that these absorbers are only used here for their attenuating properties, they are *not* considered as objects in the simulations

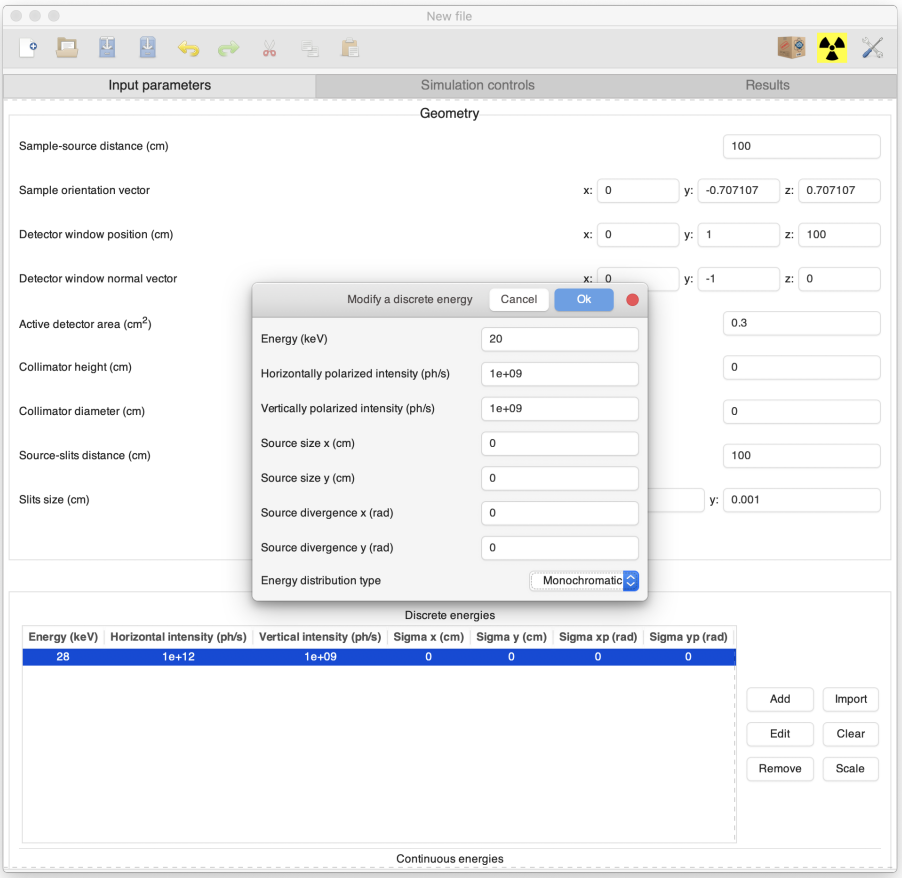

Figure 11: Modifying the energy

so they cannot contribute fluorescence lines to the eventual spectrum! Adding, editing and removing absorbers is performed through an interface identical to the one seen in the Composition section, but without the *Reference layer* toggle button. New inputfiles will always have a Be detector absorber added, corresponding to the detector window commonly found in ED-XRF detectors.

3.2.6 Detector settings

The last section deals with the settings of the detector and its associated electronics, as can be seein in the following screenshot:

- Detector type: every detector comes with its own detector response function, which can be influenced by several detector and electronics parameters. XMI-MSIM offers some predefined detector response

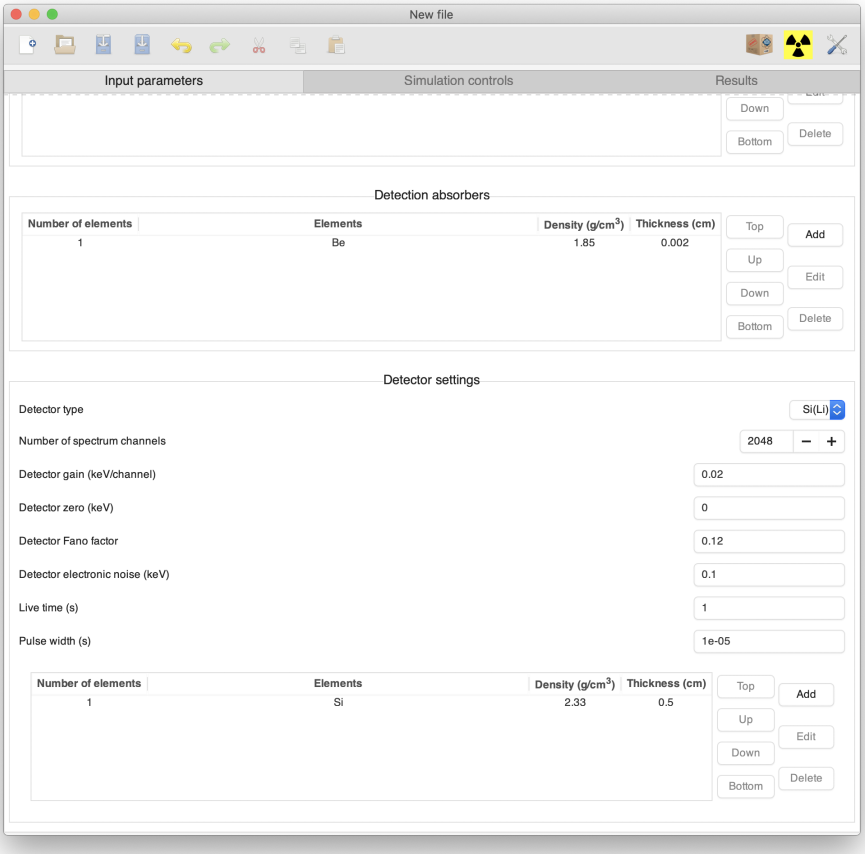

Figure 12: Detector settings

functions that its authors have found to be reasonably well for two detector types: Si(Li) and Si Drift Detectors. Generally speaking, our policy is to encourage users to implement their own detector response functions in the form as a plug-in and load it in the Simulation options

- Number of spectrum channels: the number of channels in the produced spectrum
- Live time: the actual measurement time of the simulated experiment, taking into account dead time
- Detector gain: the width of one channel of the spectrum, expressed in keV/channel
- Detector zero: the energy of the first channel in the spectrum (channel number zero)
- Detector Fano factor: measure of the dispersion of a probability dis-

tribution of the fluctuation of an electric charge in the detector. Very much detector type dependent
- Detector electronic noise: the result of random fluctuations in thermally generated leakage currents within the detector itself and in the early stages of the amplifier components. Contributes to the Gaussian broadening
- Pulse width: the time that is necessary for the electronics to process one incoming photon. This value will be used only if the user enables the pulse pile-up simulation in the Simulation controls. Although this parameter is connected to several detector and electronics parameters, typically the value is obtained after trial and error
- Crystal composition: the composition of the detector crystal. Adding, editing and removing absorbers is performed through an interface identical to the one seen in the Composition section, but without the *Reference layer* toggle button. Will be used to calculate the detector transmission and the escape peak ratios

3.3 Saving an input-file

Once an acceptable inputfile is detected by the application, the *Save* and *Save As* buttons will become activated. If the file has not been saved before, clicking either of these buttons will launch a dialog allowing you to choose a filename for the input-file.
If the file was saved before, then clicking *Save* will result in the file contents will be overwritten with the new file contents.
Keep in mind that XMI-MSIM input-files have the xmsi extension (blue logo), while the output-files the xmso extension (red logo).

3.4 Starting a simulation

In order to start a simulation, the *Input parameters* page must contain a valid input-file description. This can be obtained by either preparing a new input-file based on the instructions in a previous section (and saving it!), or by opening an existing input-file by double clicking an XMI-MSIM input-file in your file manager or opening an input-file through the *Open* interface of XMI-MSIM.
Either way, the *Simulation controls* page should look as shown in the following screenshot:

3.4.1 Control panel

The top of the page contains the actual control panel that is used to start, stop and pause the simulation, as well as a slider that allows the user to select the number of threads that will be used by the simulation. To the right of the slider, there are three progress bars that indicate different stages

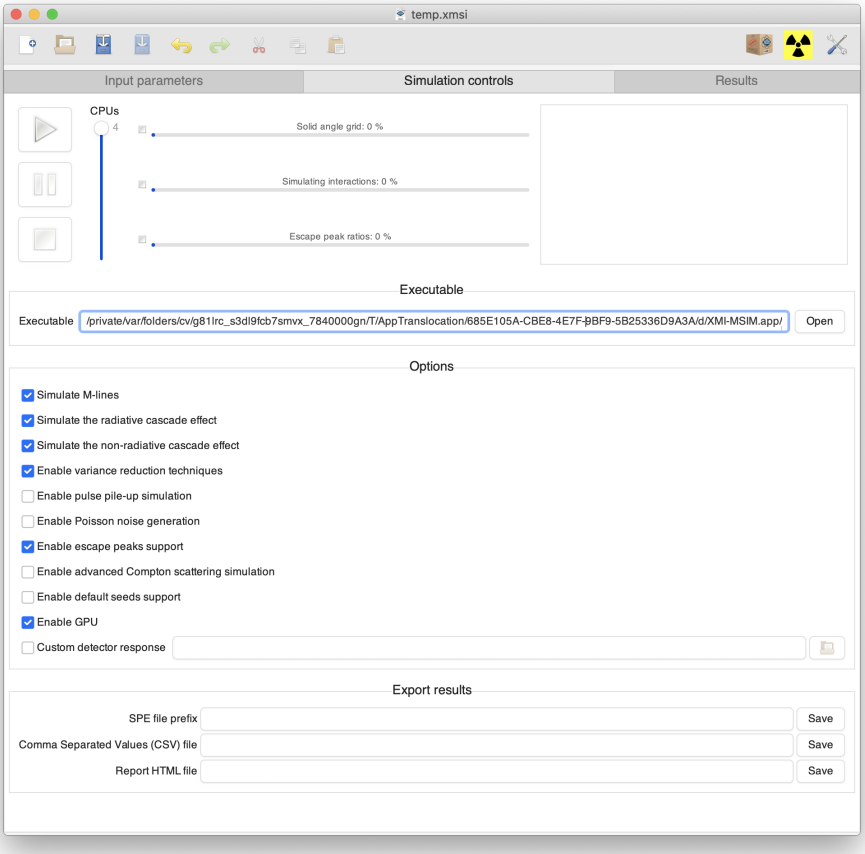

Figure 13: Simulation controls

of the Monte Carlo program: the calculation of the solid angle grid for the variance reduction, the simulation of the photon--matter interactions and the calculation of the escape peak ratios. More information about the status of the Monte Carlo program is presented in the adjacent log window.

3.4.2 Executable

Underneath these controls is a section that contains the name of the executable that will be used to launch the simulation. Most likely, you will never have to change this value, but it could be interesting to power users, who have customized versions of the simulation program.

3.4.3 Options

This section is followed by a number of options that change the behaviour of the Monte Carlo program:

- *Simulate M-lines*: If disabled, then the code will ignore M-lines that may be produced based on the elemental composition of the sample. In such a case, the code will probably run faster. I strongly recommend to simulate M-lines
- *Simulate the radiative and non-radiative cascade effect*: the cascade effect is composed of two components, a radiative and a non-radiative one. Although these will always occur simultaneously in reality, the code allows to deactivate one or both of them. This could be interesting to those that want to investigate the contribution of both components. Otherwise, it is recommended to keep both enabled
- *Enable variance reduction techniques*: disabling this option will trigger the brute-force mode, disabling all variance reduction techniques, thereby greatly reducing the precision of the estimated spectrum and net-line intensities for a given *Number of photons per discrete line*. This reduced precision may be improved upon by greatly increasing the *Number of photons per discrete line*, but this will result in a much longer runtime of the Monte-Carlo program. Expert use only. Consider building XMI-MSIM with MPI support and running it on a cluster
- *Enable pulse pile-up simulation*: this option activates the simulation of the so-called sum peaks in a spectrum due to the pulse pile-up effect which occurs when more photons are entering the detector than it can process. The magnitude of this effect can controlled through the *Pulse width* parameter
- *Enable Poisson noise generation*: enabling this option will result in every channel of the detector convoluted spectrum being subjected to Poisson noise, controlled by Poisson distributions with lambda equal to the number of counts in a channel
- *Enable escape peaks support*: enable this option to activate the support for escape peaks in the detector response function. Typically you will want to leave this on
- *Enable advanced Compton scattering simulation*: this option activates an alternative algorithm for the simulation of the Compton profiles based on the work of Fernandez and Scot, which takes into account the fact that not all orbitals are completely populated, leading to a more accurate reproduction of the profiles. The downside of this approach is that it's slower than the default implementation. Recommended only for advanced users. In order to fully understand the importance of an improved Compton profile simulation, the reader is advised to read the aforementioned manuscript of Fernandez and Scot

- *Enable default seeds support*: instead of using randomly chosen seeds for the random number generators, use default values, which should result in reproducible results, at least when using a single thread. This option should normally not be used.
- *Enable GPU*: this option invokes XMI-MSIMs GPU plug-in which, if the platform comes with a videocard chipset that supports it, will use the GPU to perform the solid angle calculation, which could lead to a tremendous speed increase. Keep in mind that when this option is used during the solid angle calculation stage, the screen may have a noticeably lower refresh rate and may lose its responsiveness briefly. This option is only available when an OpenCL (all platforms) or Metal (macOS only) framework was found at compile-time
- *Custom detector response*: through this option, one can load a plug-in that exports a routine that will override the default detector response functions of XMI-MSIM. Click here more information on how to write and build such plug-ins

3.4.4 Export results

The page ends with a section that allows the user to export the output of the Monte-Carlo program at run-time to several fileformats in addition to the default XMSO fileformat.

- SPE file: the well known ASCII format, readable by PyMca and AXIL. Produces one file per additional interaction. When using the file dialog to choose the filename, make sure not to add a file extension: the Monte-Carlo program will append an underscore, the number of interactions and the .spe extension automatically
- Comma Separated Values (CSV) file: produces a CSV file containing several columns. The first column contains the channel number, the second one contains the corresponding channel energy and the following columns contain the intensities for increasing number of interactions
- Report HTML file: produces an html file that can be opened with most Internet Browsers (Internet Explorer being a notable exception), featuring an interactive overview of the results of the Monte-Carlo simulation, simular to the ones shown on the Results page

It is possible to generate these files afterwards based on the XMSO file, by clicking in the menubar on *Tools -> Convert XMSO file to.*

3.4.5 During a simulation

When all required options are set up correctly, the simulation can be started by clicking the *Play* button. After this, you will notice a lot of output being

generated in the log window, as well as some activity in the progress bars, as shown in the next screenshot:

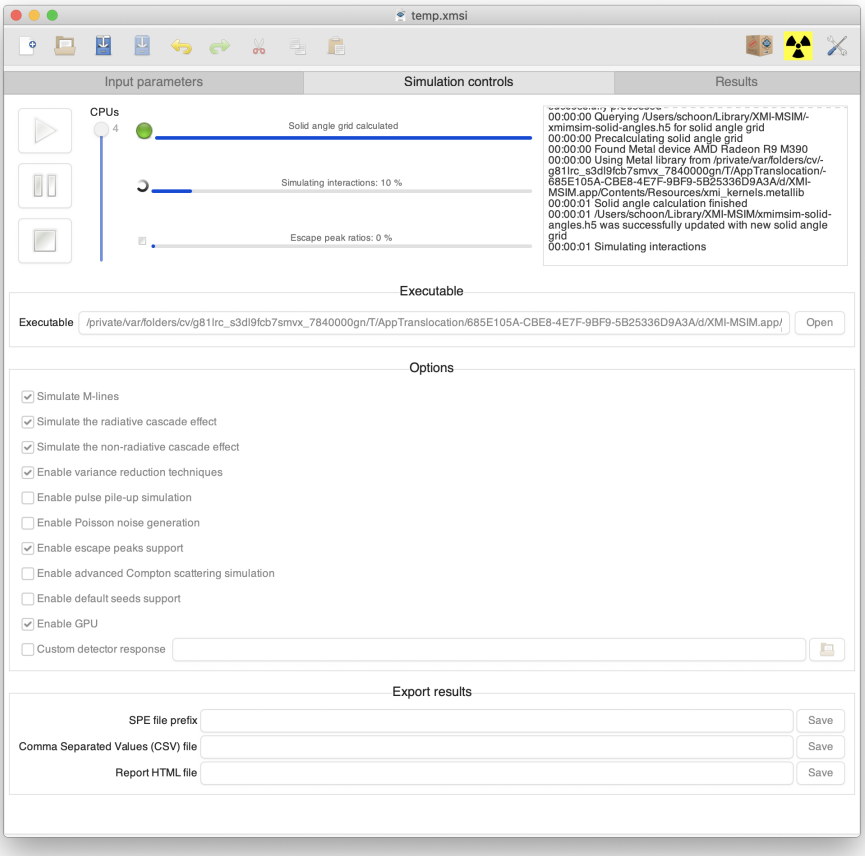

Figure 14: Running the simulation

The first and the third progress bars will in many cases display a message that the Solid angle grid and the Escape peak ratios were loaded from file: this indicates that a simulation with similar parameters was performed before and that the relevant data was written to a file, leading to a huge increase in speed. It is possible that some red text appears during a run, particularly with reference to Solid angle and Escape ratio HDF5 files being outdated. This only happens when you are running a new version of XMI-MSIM that introduced a new format for these files: the old files are deleted and new ones will be created and used from then onwards.
After the simulation, assuming everything went fine, the XMSO outputfile as defined in the General section will be loaded and its contents displayed

on the Results page.

3.5 Visualizing the results

The results of a simulation are stored in an XMSO file (red logo): you should be able to open these files directly by double clicking them from your file manager. Alternatively, you can also load these files from within XMI-MSIM by clicking the *Open* button, and subsequently setting the filetype filter to *XMI-MSIM outputfiles*. On Linux and Windows, you can also open these files from the command-line:

```
xmimsim-gui file.xmso
```

XMSO files created after a successful simulation are automatically loaded in the Results page, where the spectra and net-line intensities are represented.

3.5.1 Plot canvas

If a simulation was performed according to the inputfile that was defined earlier, you should get a result similar to the one in the following screenshot: The plot canvas shows by default the different spectra obtained after an increasing number of interactions. Individual spectra may be hidden and shown by toggling the boxes to the right of the plotting window. Their properties of a spectrum may be modified by clicking on the *Properties* button connected to it, which launches a dialog allowing the user to change the line width, line type and line color of the spectrum. The *Properties* button above the coordinate entries opens a dialog with the option to switch between linear and logarithmic display of the spectra, as well as the opportunity to change the axes titles. More options will be added in future releases allowing the user to customize this further.
Zooming in on the plot canvas by dragging a rectangle with the mouse while keeping the left button clicked in. Zooming out can be accomplished by double-clicking anywhere in the canvas. While moving the mouse cursor in the plot canvas, one can track the current Energy, Channel number and Intensity in the textboxes to the right. The size of the canvas can be changed by grabbing and moving the handle that separates the upper part from the lower part of the page.

3.5.2 Net-line intensities

The lower part of the page contains a list of all the intensities of all the X-ray fluorescence lines of all elements, as shown in the following screenshot:
By clicking the arrows on the left side of the list, it is possible to expand the sections belonging to a particular element, line, and for different number of interactions, thereby revealing the individual contributions to a particular

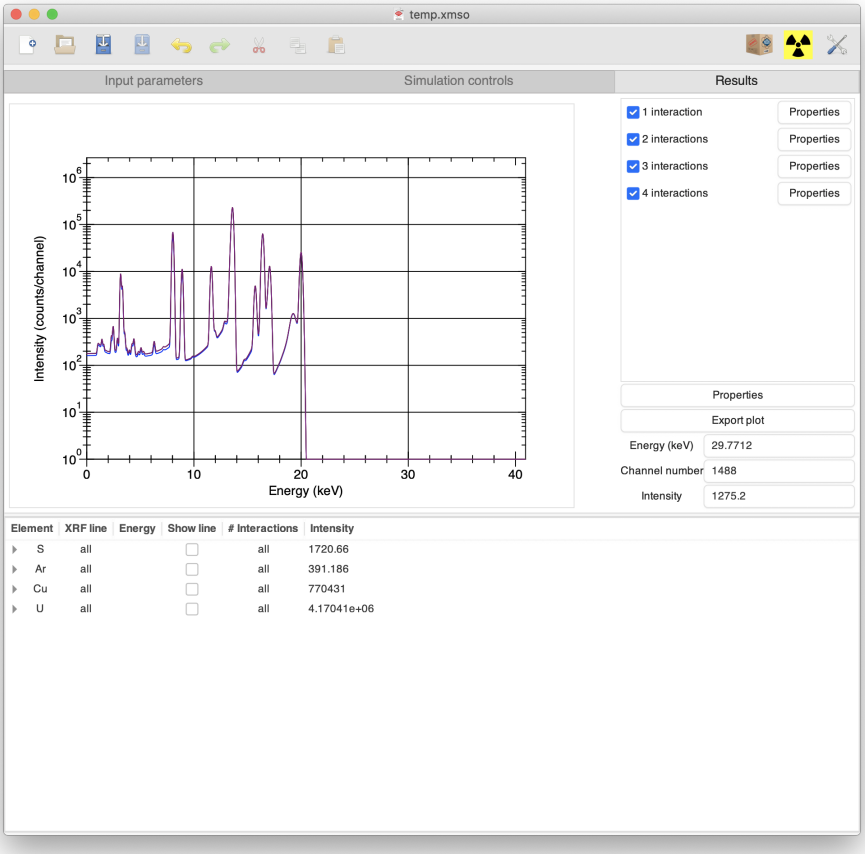

Figure 15: Visualizing the results

intensity. The lines can be shown on the plot canvas by activating the *Show line* flag for the appropriate line or element.

3.5.3 Exporting the plot canvas

The plot canvas can be exported to several filetypes using the *Export plot* button to the right of the plot canvas. This will result in an exact copy of the current state of the canvas being stored to file: it will take into account all the changes that were made to the spectra properties, as well as any lines that were activated using the *Show line* togglebuttons of the *Net-line intensities* section. Supported filetypes are PNG, EPS, PDF and SVG.

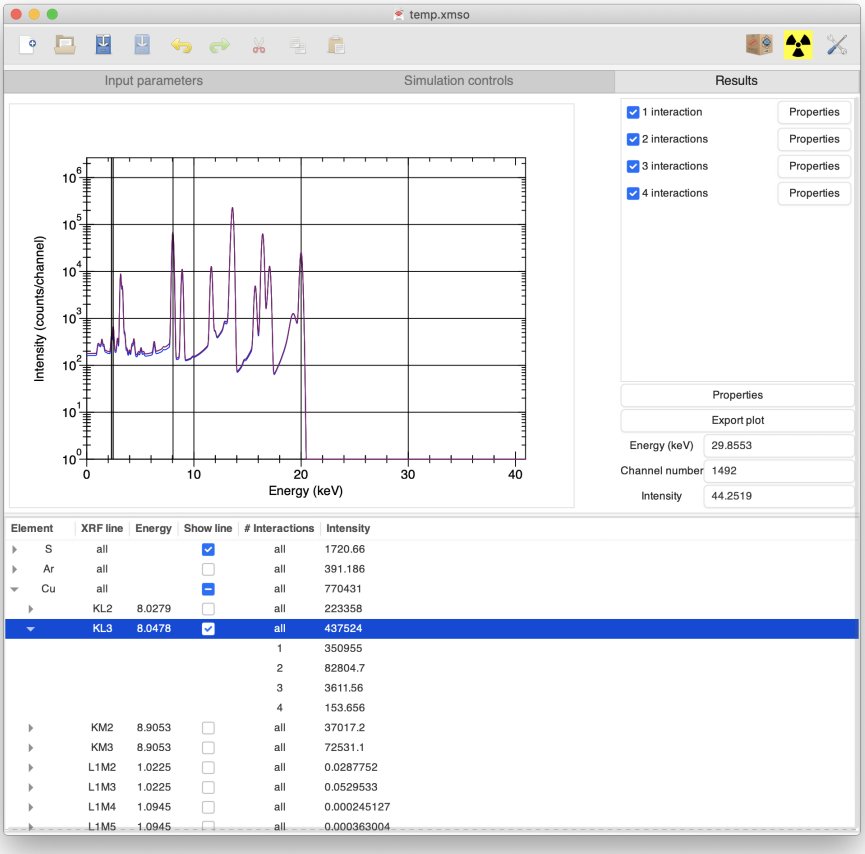

Figure 16: Selecting XRF lines

3.6 Global preferences

Clicking the *Preferences* button in the toolbar will launch a dialog allowing the user to set some preferences that will be preserved across sessions off XMI-MSIM. Make sure to press apply after making any changes.

3.6.1 Simulation defaults

The first page of the preferences window contains the same settings that are available on the *Simulation controls* page. The values that are selected here will be activated in the *Simulation controls* page the next time that XMI-MSIM is started.

3.6.2 Updates

If XMI-MSIM was compiled with support for automatic updates then this page will contain two widgets: firstly a checkbox that will determine if the program will check for updates at startup, and secondly a list of locations that will be used to download updates from.

3.6.3 User-defined layers

Select layers and hit backspace to remove them from the list of user-defined layers (which are defined in the *layer* dialogs). The layers are deleted when the Apply button is clicked.

3.6.4 Advanced

The first two options revolve around the deleting of the XMI-MSIM HDF5 files that contain the solid angle grids and the escape peak ratios, respectively. It is recommended to remove these files manually when a complete uninstall of XMI-MSIM is considered necessary (before running the uninstaller or removing the application manually), or if these files somehow got corrupted. The following two options allow the user to import solid angle grids and escape peak ratios from external files into his own. This may be interesting if another user already has a huge collection of these and it may save a lot of time using someone elses. In the file dialog only those files will be shown that are valid HDF5 files of the required kind and minimum version.
The last option is *Enable notifications*, which when supported at compile-time and a suitable notifications server is found, will generate messages whenever a calculation finishes. On a Mac OS X native version of XMI-MSIM this will only work on Mountain Lion and newer.

3.7 Checking for updates

For packages of XMI-MSIM that were compiled with support for automatic updates, checking for new versions will occur by default when launching the program. This can be disabled in the Preferences window. If you would like to check explicity, then click on *Check for Updates* in the toolbar menu. When updates are available, a dialog will pop up, inviting the user to download the package through the interface. When the download is completed, quit XMI-MSIM and install the new version. It is highly recommended to always use the latest version of XMI-MSIM.

3.8 Command line interface

XMI-MSIM ships with a number of command line utilities that may be useful for some users. An overview:

- `xmimsim-gui`: This executable corresponds to the graphical user interface of XMI-MSIM, as described in this user guide. This will mostly be useful for Linux users or Mac OS X users that compiled from source without gtk-mac-integration support.
- `xmimsim`: The executable that actually does the hard simulation work. It is usually launched from within the graphical user interface with the *Play*, *Pause* and *Stop* buttons from the control panel, but in some circumstances it may be useful from the command-line as well. It has a lot of options: consult them by running `xmimsim --help`. Windows users will have to use `xmimsim-cli.exe` since `xmimsim.exe` is compiled as a graphical user interface executable in order to avoid a console window from popping up during the simulation in XMI-MSIM.
- `xmimsim-db`: Used to generate the `xmimsimdata.h5` file that contains the tables of physical data (mostly inverse cumulative distribution functions) that will be used during the simulation to speed things up drastically. This executable is intended for those that compile XMI-MSIM from source.
- `xmimsim-conv`: A recently added executable that allows to extract the unconvoluted spectra from an XMSO file and apply the detector response function to it with different settings that were used initially to generate the XMSO file.
- `xmimsim-harvester`: a daemon that collects seeds for the random number generators. Read the note on the random number generators in the installation instructions for more information.
- `xmso2xmsi`, `xmso2spe`, `xmso2csv`, `xmso2htm`: utilities that allow for the conversion of XMSO files to the corresponding XMSI, SPE, CSV and HTML counterparts, providing the same functionality as obtained through *Tools -> Convert -> XMSO file -> ...*
- `xmsa2xmso`: utility that allows for the extraction of XMSO files from an XMSA file, providing the same functionality as obtained through *Tools -> Convert -> XMSA file -> to XMSO*.
- `xmsi2xrmc`: Utility to convert an XMSI file to the corresponding XRMC input-files. Read here for more information. Also available through *Tools -> Convert -> XMSI file -> to XRMC input-files*.
- `xmimsim-pymca`: The quantification plug-in that is used by PyMca.

Most of these executables have quite a few options. Consult them by passing the `--help` option to the executable.

3.9 Example files

The example input-file that was created throughout the *Creating an input-file section* can be downloaded at *test.xmsi*. The corresponding output-file can be found at *test.xmso*.

4 Advanced usage

In this section, we will describe some more advanced features of XMI-MSIM, which may useful for some users with specific needs.

4.1 X-ray sources

In the *Excitation* section, we have shown how one can introduce the necessary components of the X-ray excitation spectrum, through a number of discrete energies and intervals of continuous energies. XMI-MSIM tries to facilitate the process of defining the excitation spectrum through its *X-ray sources* dialog, which can be invoked by clicking the corresponding button (with the radiation warning symbol) in the toolbar. Currently two types of sources can be defined this way: X-ray tubes and radionuclides. Switching between both types can be accomplished easily by clicking on the desired tab in the dialog. It is possible to add user-defined sources to this dialog by writing custom plug-ins. Documentation on how to do this will be added soon.
After adjusting the required parameters of the selected source, click *Update spectrum* to obtain a new excitation spectrum in the plot window. Using *Export spectrum*, it is possible to save the generated spectrum to an ASCII file, while *Save image* will allow the user to save the plot window to an image file. Clicking *About* will display some information regarding the origins of the model or dataset. Using the *Ok* button one can close the window while replacing the contents of the *Excitation* section with the newly generated spectrum.

4.1.1 X-ray tube spectrum generator

In many cases, one will perform X-ray experiments using an X-ray tube generator as source, which corresponds to a combination of discrete part (the anode element specific XRF lines) and a continuous part (the Bremsstrahlung generated through electron-nucleus interactions). Such excitation spectra are typically quite difficult to obtain experimentally and instead one relies quite often on theoretical calculations to obtain (an approximation) of the spectrum. One popular model is the one derived by Horst Ebel in his manuscripts X-ray Spectrometry 28 (1999), 255-266 and X-ray Spectrometry 32 (2003), 46-51. XMI-MSIM's implementation of this model is based on the similar dialog in PyMca. The X-ray tube dialog should be similar to the following screenshot (click the *X-ray tube* tab if the *Radionuclide* panel showing):
By changing the different parameters to values appropriate for the X-ray tube that the user would like to simulate, one obtains an **approximate** model for the corresponding X-ray tube excitation spectrum. The following parameters can be changed:

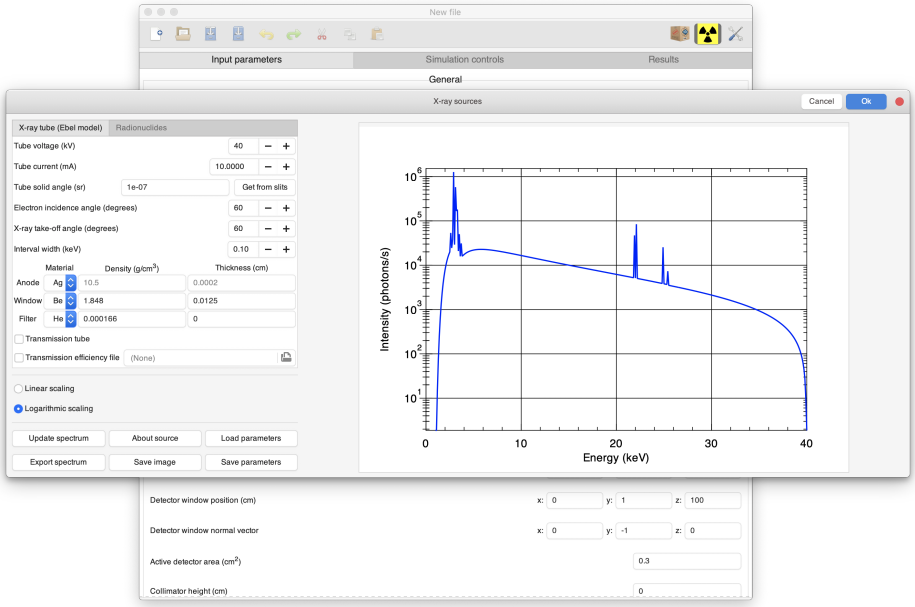

Figure 17: Introducing the X-ray tube parameters

- *Tube voltage*: the voltage in kV at which the X-ray tube is supposed to operate. This will determine the extent of the Bremsstrahlung contribution and through this which XRF lines (discrete energies) that will be present in the spectrum.
- *Tube current*: the current in mA at which the X-ray tube is supposed to operate. The this value is directly proportional to the intensity of the spectrum components.
- *Tube solid angle*: the solid angle in sr (steradian) under which the beam emerges from the X-ray tube. The default value here is determined by the *Source-slits distance* and the *Slits size*, taken from the [*Geometry* section].
- *Electron incidence angle* and *X-ray tube take-off angle*: X-ray tube geometry parameters
- *Interval width*: the width of the continuous energy intervals of Bremsstahlung part of the spectrum. Decreasing this value will lead to a better simulation, but will increase the computational time.
- *Anode*: the material that the tube anode is made of. The density and the thickness become sensitive when *Transmission tube* is activated.
- *Window* and *Filter*: tube filtration materials. Set the thickness and/or the density to zero to ignore.
- *Transmission tube*: activating this option effectively places the tube exit-window on the opposite side of the anode with respect to the

cathode, thereby operating in transmission mode.

- *Transmission efficiency file*: it is possible to load a two column ASCII file (first column energies and second column efficiencies between 0 and 1), with at least 10 lines that will be used to multiply the generated intensities and intensity densities with using interpolation of the supplied efficiencies.

4.1.2 Radionuclides

A second type of X-ray sources offered by XMI-MSIM concerns radionuclides. Through xraylib's radionuclide API, one can gain access to the excitation profiles (X- and gamma-rays) of several radionuclides that are commonly used as X-ray sources. After clicking on the *Radionuclide* tab, the following dialog should be visible:

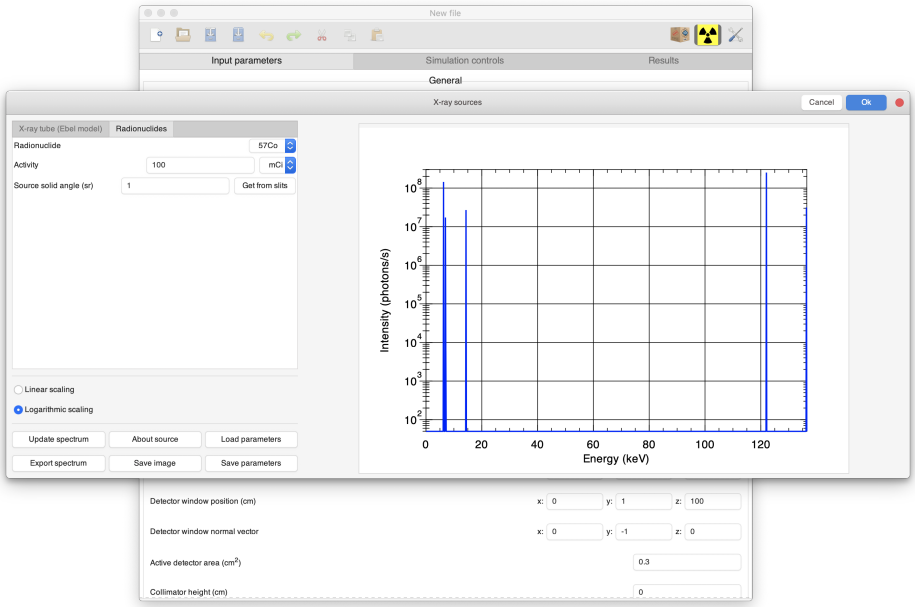

Figure 18: Introducing the radionuclide parameters

The following parameters will determine the excitation spectrum:

- *Radionuclide*: select the desired radionuclide
- *Activity*: set the activity (disintegrations per second) of the radionuclide, as well as its unit (mCi, Ci, Bq, GBq)

Clicking the *Ok* button will produce a dialog asking whether to add the radionuclide to the excitation spectrum or to replace it entirely. Clicking *Add*

allows the user ultimately to define a source composed of several radionuclides, which may be useful when the different sources are positioned on a ring, positioned perpendicular to the detector axis.

4.1.3 Custom sources

The two aforementioned sources are implemented as libpeas plugins, written in C, and are derived from the abstract base class `[XmiMsimGuiSourceAbstract](https://github.co`
While it is certainly possible to write new plug-ins in C or C++ that extend this class, this is certainly not trivial. Interested parties may want to have a look at the source code of the X-ray tube, radionuclide plug-ins, as well as the random spectrum example.
I recommend instead to write plugins in Python 3, which is made possible through GObject-Introspection and PyGobject. Large parts of the exposed `libxmimsim` and `libxmimsim-gui` API can be called directly from Python, in a manner that will feel familiar to Python coders.
An example (which is part of the examples folder of the installation) follows:

```
from gi.repository import XmiMsimGui, XmiMsim, Gtk, GLib
import sys
import numpy as np


class TestSource(XmiMsimGui.SourceAbstract):
    # Optional, will not affect GUI in any way if left out...
    __gtype_name__ = "TestSourcePython"

    # initialize parent first, and add GUI elements
    def __init__(self):
        XmiMsimGui.SourceAbstract.__init__(self)
        print("Calling __init__")
        button = Gtk.Button.new_with_label("Click me!")
        self.add(button)
        self.counter = 1000

    # source name: this will end up in the tab label
    def do_get_source_name(self):
        return "Python Source"

    # the about text: will be shown in the Source About Dialog
    def do_get_about_text(self):
        return "This could very well be some clever text about this source"

    # this method is executed whenever "Update Spectrum" is clicked
```

```
    def do_generate(self):
        print("Calling do_generate")
        # if something goes wrong: initialize a new GLib.Error instance and send it
        #     error = GLib.Error.new_literal(XmiMsimGui.SourceAbstractError.quark(),
        #     self.emit("after-generate", error)
        #     return
        x = np.linspace(1.0, 50.0, num=2000, dtype=np.double)
        y = self.counter * np.exp(-1.0 * (x - 20.0) * (x - 20.0) / 2.0) / np.sqrt(2
        y[y < 1.0] = 1.0
        self.counter += 100

        excitation = XmiMsim.Excitation.new(discrete=[XmiMsim.EnergyDiscrete.new(20

        # this updates the source internal data: raw, plot_x, and plot_y
        self.set_data(excitation, x.tolist(), y.tolist())

        # afterwards emit after-generate with None argument to update the plot windo
        self.emit("after-generate", None)
```

As shown in this example, the class you define (`TestSource` here...), must extend `XmiMsimGui.SourceAbstract`. The class must implement 4 methods:

1. `__init__`: the *constructor*, which should be used to define the GUI elements that will be added to the GUI, as well as initialize any member variables you may need. `XmiMsimGui.SourceAbstracts` extends [`Gtk.Box`], so you will need to use its API to add new widgets to it.
2. `do_get_source_name`: This method must return a string containing a short name of the source, and will be used to populate the tab label.
3. `do_get_about_text`: This method must return a string with a (long) description of the source. It will be displayed when in the dialog that is shown after clicking the *About* button.
4. `do_generate`: This method gets called whenever the *Update spectrum* button is clicked, and is meant to extract the currently introduced information from the widgets that were added by the constructor. Based on this data, if valid (write code to check!), create a new `XmiMsim.Excitation` object, as well as two Python lists of doubles that will contain the X- and Y-values that will be plotted. Ensure that the Y-values are greater than zero, as they may be plotted logarithmically. Afterwards, call the `set_data` method on the source object, followed by an emission of the `after-generate` signal to trigger an update the plot window. If the data is invalid, a `GLib.Error` object should be instantiated, followed by the error object being added to an `after-generate` signal emission.

I strongly recommend that people with no experience in coding Gtk based

apps in Python, to have a look at this tutorial.
When you have written a source plug-in, make sure to copy to it to `Documents\XMI-MSIM\sources` in your home directory, which will be searched at run-time for usable source plug-ins. It is not sufficient to merely copy the Python source file, you will also need to write a short `.plugin` file that will contain the following:

```
[Plugin]
Module=test-source
Name=Test source
Loader=python3
```

Important here is that the name of the Python source file, without its `.py` extension is assigned to `Module`.

4.2 Batch simulations

XMI-MSIM version 3.0 introduced the possibility to perform batch simulations. Activate this feature by clicking the *Batch mode* button in the toolbar. This will produce a wizard as shown here:
Clicking *Next* will move the wizard to a filechooser dialog:
At this point it becomes very important to distinguish between two different possible outcomes that depend on whether the user selects either one or multiple files.

4.2.1 Batch simulations: simulate a number of unrelated input-files

If the user has selected multiple files, then these files will be used as input-files for a round of successive unrelated simulations. After the file selection, the user will be presented with a question regarding whether the options should be set for each input-file separately. The options refer to the same options that can be seen in the *Control panel* of the main interface window. Either way, after setting the options, one will end up with the *Batch simulation controls* window:
Similar to the *Control panel* of the main interface window, this widget features *Play*, *Stop* and *Pause* to control the execution. The number of threads that will be used for the simulations may be set using the CPUs slider. During execution, all output will be shown in the central area. The verbosity level can be changed from the default *Verbose* to *Very verbose* for even more information about the runs. While running the simulations, it is possible to save all output that is placed on the screen to a file that will be continuously updated. Click the *Save As* button to choose a filename.

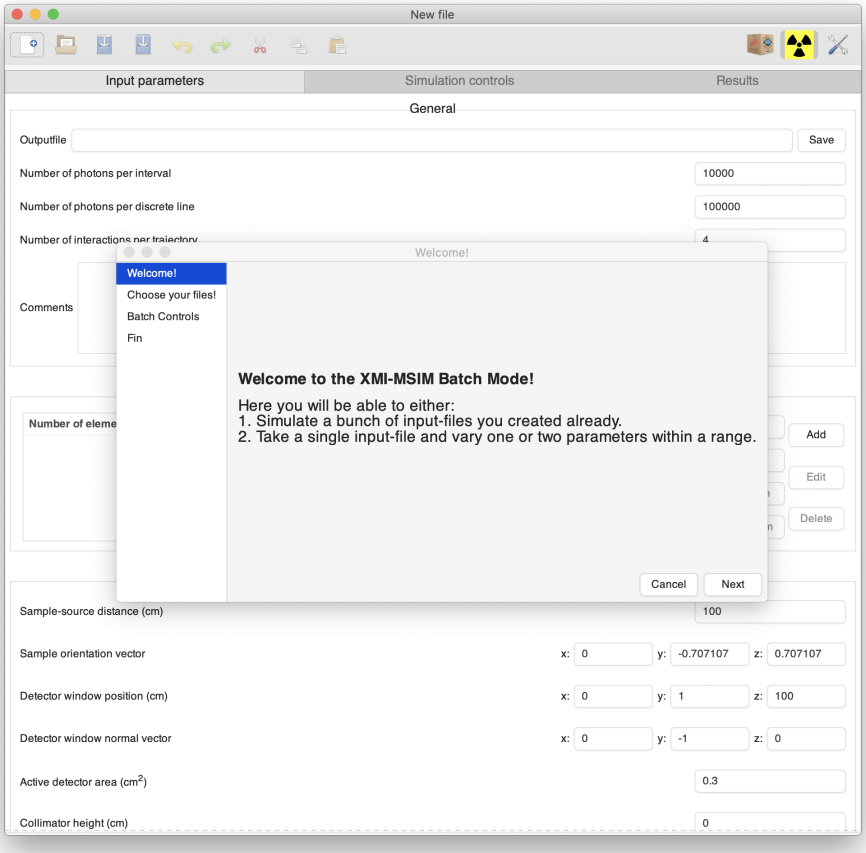

Figure 19: Wizard welcome page

4.2.2 Batch simulations: vary one or two parameters in a single input-file

A considerably more interesting feature of the batch simulation is its second operational mode: if the user selects a single file from the filechooser page, and selecting the desired options, he will be presented with a new page in which he is asked to select one or two parameters that will be varied during a series of simulations based on the initially selected input-file, as is seen in the following screenshot:
After expanding the different components of the tree structure representing the initial input-file contents, rows describing the input-file contents will emerge: only the clickable components are eligible as variable parameters! Furthermore, it should be noted that within a layer, one can only select an element's *weight_fraction* if there are at least two elements available: this

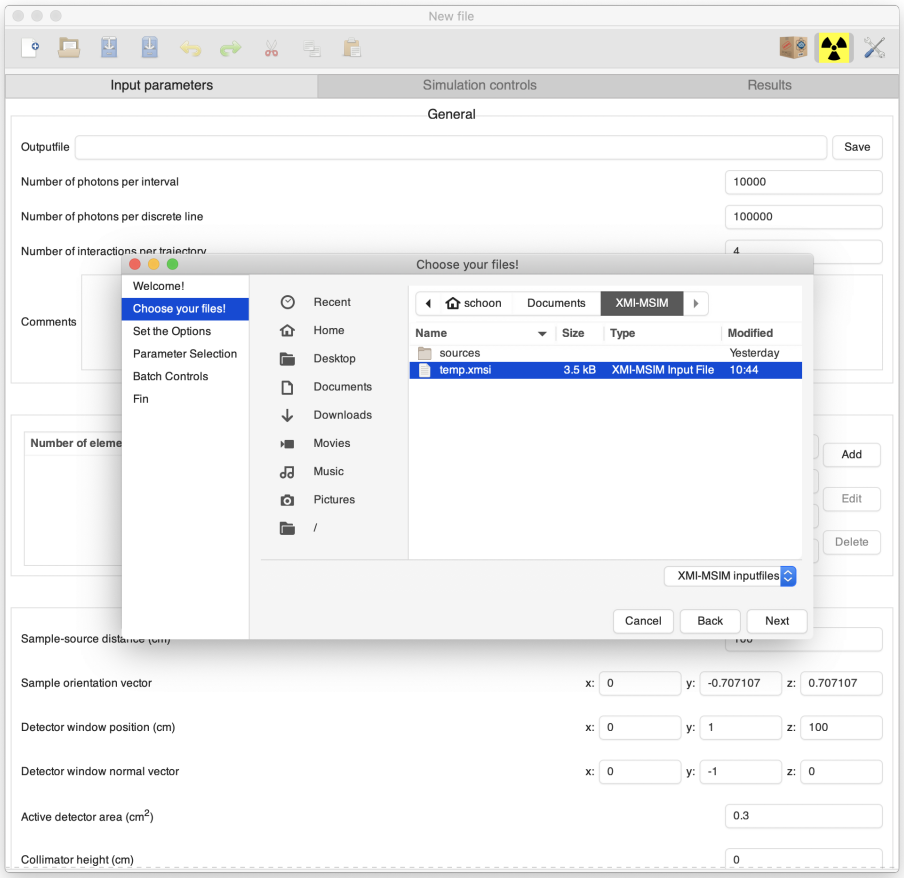

Figure 20: Select one or more files to enter the batch simulation mode

is necessary because at any given moment, the sum of the weight fractions needs to be equal to 100% after rescaling. If two weight fractions within the same layer need to te varied, then at least three elements need to be present in that layer for the same reason.
Clicking _Nextafter selecting the required parameter(s), will move the wizard to to a page that will allow users to define the range and the number of steps that will be used to determine the parameter(s) values in the different input-files that will be produced and later on, simulated. In bold, above the *Start*, *End* and *#Steps* entries, are the name(s) of the selected parameter expressed in its XPath notation, which corresponds to an internal description of the parameter's location in the XMI-MSIM input-file. This page also contains a *Save As* button that will launch a file chooser dialog, which will ask the user to determine the XMI-MSIM archive that will eventually be produced containing all results from the simulation. This is shown (for a

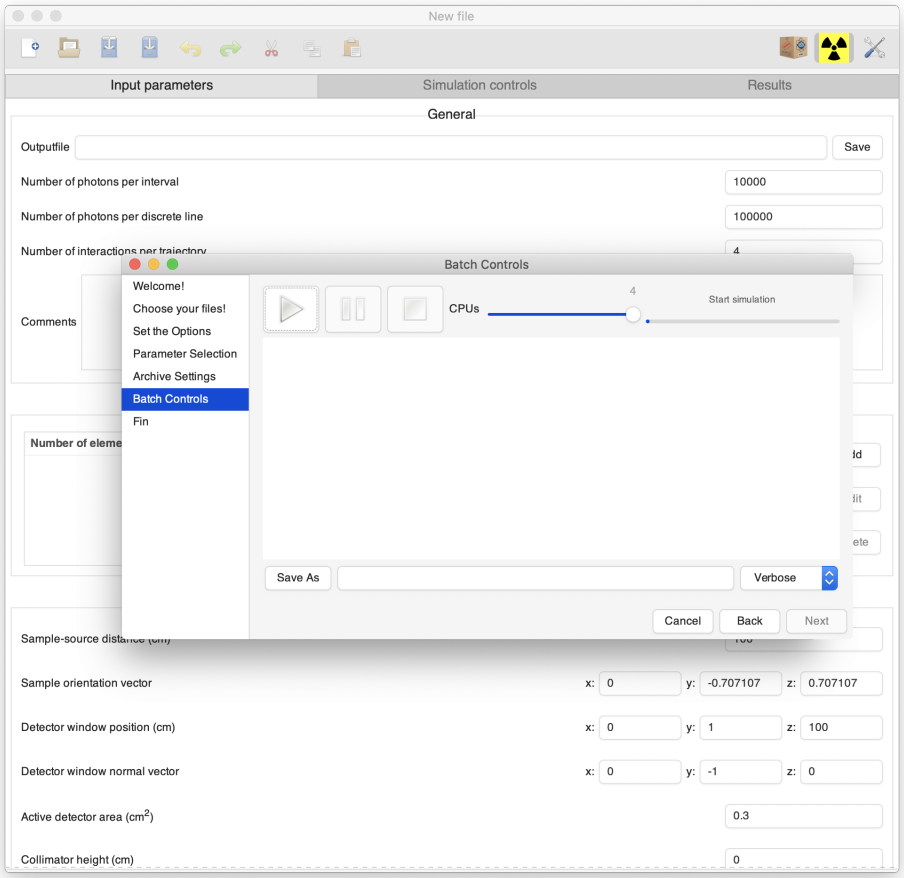

Figure 21: Batch simulation controls

case with one selected variable parameter) in the following screenshot:
After confirming the introduced values by clicking *Next*, a *Batch simulation controls* page will appear, as was already described and shown in the preceding section. Clicking the *Play* button will launch the simulations. After all simulations have been successfully performed, click the *Next* button to proceed to the final page, which can be closed. In the meantime, a window should have popped up that allows the user to analyze the results of the batch simulation.
In this window, one can analyze the results of the batch simulation by selecting specific elements, lines, regions of interest etc for individual or cumulative interaction contributions. It is possible to save the plot as an image file using *Save image*, while the data that makes up the currently shown plot can be exported in a CSV file. Change the axes titles to a more appropriate description if deemed necessary. The following screenshot shows

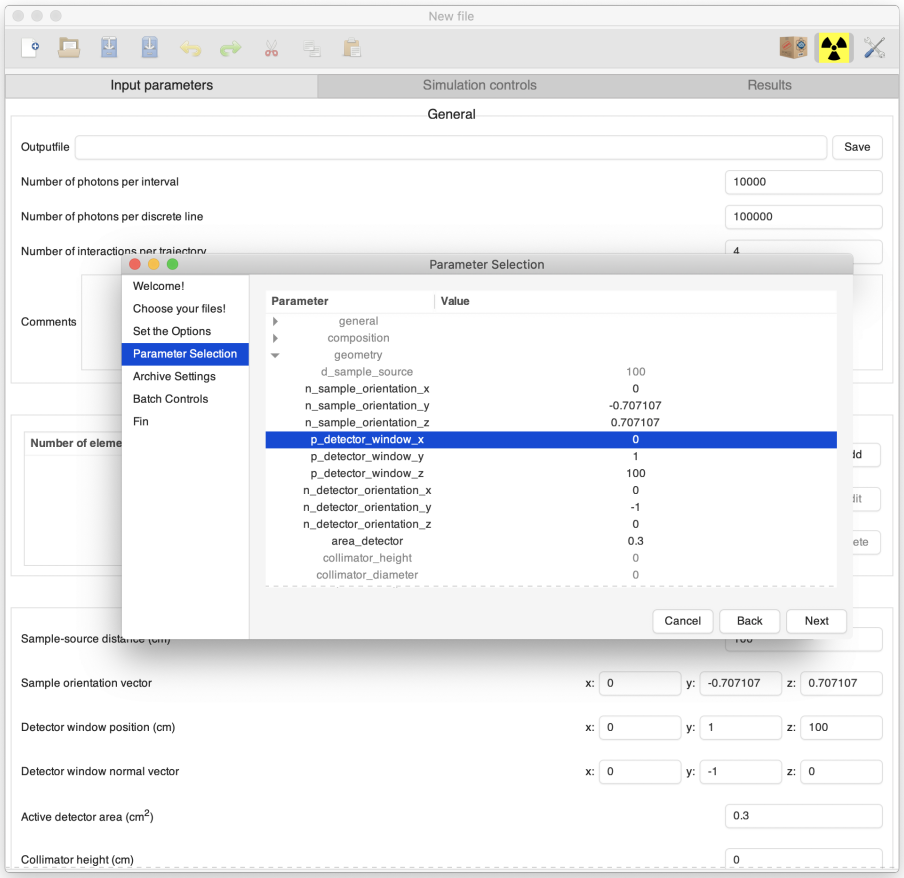

Figure 22: Select one or two parameters to be varied

a case where two variable parameters were chosen:
All information that was produced in the batch simulation has been stored in an XMI-MSIM archive file (.xmsa extension). If one would like to inspect its contents again with the *Batch mode plot* window, just double-click such a file from your favorite file manager, or open it from within XMI-MSIM by clicking *Open* in the toolbar or menubar and setting the filter to *XMI-MSIM archives*, and then selecting the desired file.

4.3 Generate XRMC input-files

Using the *Convert XMSI file to XRMC* option from the *Tools* menu, one can produce input-files for the XRMC software package, a Monte Carlo simulation tool for X-ray imaging and spectroscopy experiments. This should be of particular interest to those users that are interested in a simulation

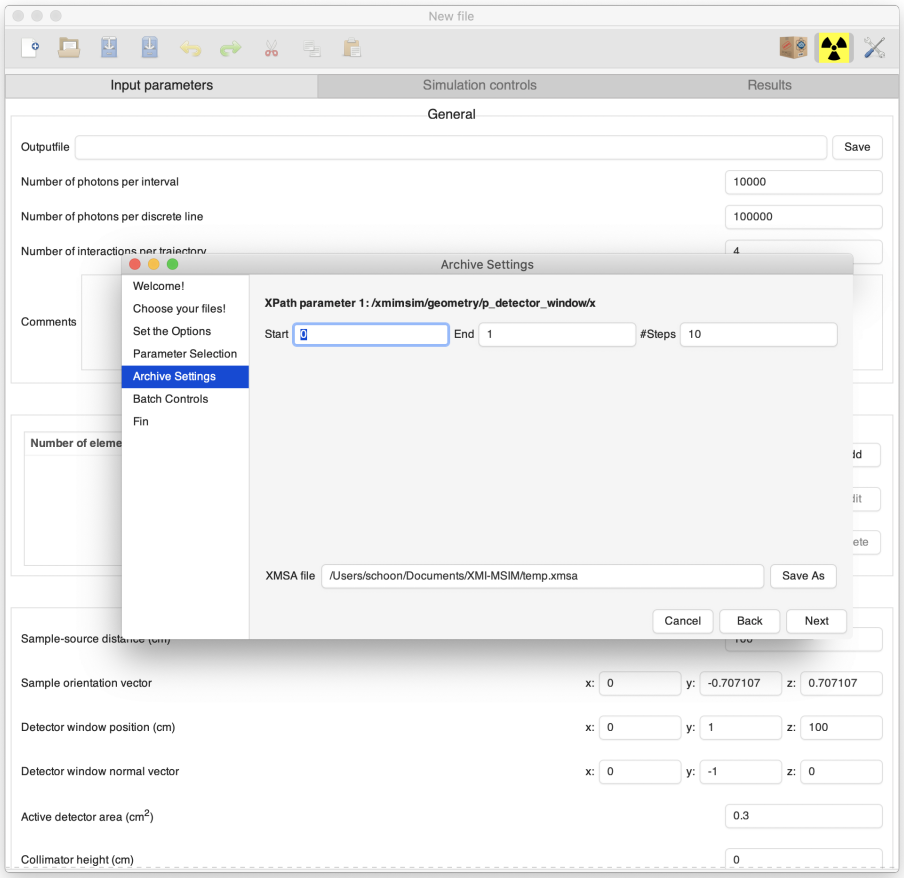

Figure 23: Set the range of the variable parameter(s) and the name of the XMI-MSIM archive file

that includes scattering and XRF that is generated by the collimator, which is being ignored by XMI-MSIM. Keep in mind though that simulations with XRMC typically will take considerably longer compared to XMI-MSIM for a result with equivalent statistical variance. In order to use the produced input-files, install XRMC including its XMI-MSIM plug-in, which will be used for the detector response function. One can also generate the XRMC input-files using the command-line utility `xmsi2xrmc`.

4.4 Using the XMI-MSIM API from Python

The section on custom X-ray sources mentioned that large parts of the `libxmimsim` and `libxmimsim-gui` libraries may be called from Python. Apart from creating these custom sources, it is also an interesting feature

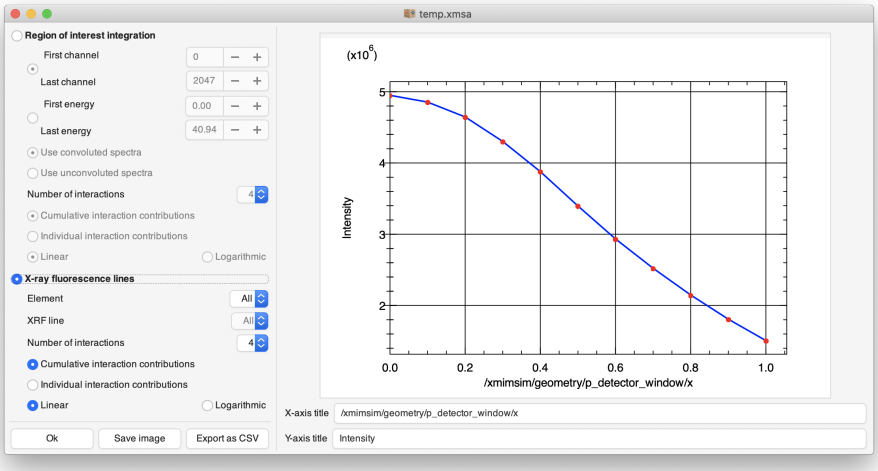

Figure 24: Batch mode plot window for one variable parameter

to exploit in regular Python scripts. Consider the following example, which sets up a batch simulation that varies the atomic number in a layer across he periodic table, saves the results to an XMSA archive, and opens it into an `XmiMsimGui.XmsaViewerWindow` instance:

```
import xraylib as xrl
import sys
import os
import math
import logging
logging.basicConfig(format='%(asctime)s %(message)s', level=logging.DEBUG)

import gi
gi.require_version('XmiMsim', '1.0')
gi.require_version('XmiMsimGui', '1.0')
from gi.repository import XmiMsim, XmiMsimGui, GLib, Gio, Gtk

OUTPUTFILE = 'mendeljev.xmsa'

XmiMsim.xmlLoadCatalog()

main_loop = GLib.MainLoop.new(None, False)
input = XmiMsim.Input.init_empty()
general = input.general.copy()
general.n_photons_line = 100000 # 1M photons
```

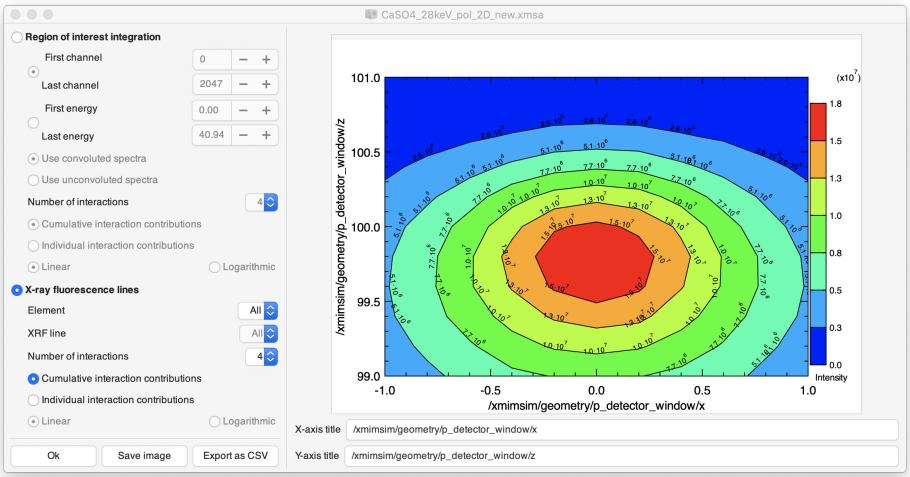

Figure 25: Batch mode plot window for two variable parameters

```
input.set_general(general)
options = XmiMsim.MainOptions.new()

single_data = [XmiMsim.BatchSingleData.new("/xmimsim/composition/layer[1]/element[1]
xmsi_data = list()

for Z in range(1, 93):
    input_copy = input.copy()
    layer = XmiMsim.Layer.new([Z], [1.0], 1.0, 1.0)
    composition = XmiMsim.Composition.new([layer], reference_layer=1)
    input_copy.set_composition(composition)
    xmsi_data.append(input_copy)

batch = XmiMsim.BatchSingle.new(xmsi_data, single_data, options)

assert batch.is_valid_object() == True

def _test_succeed_finished_cb(batch, result, buffer):
    logging.debug("message: {}".format(buffer))
    assert result == True
    main_loop.quit()

def _print_stdout(batch, string):
    logging.debug("stdout: {}".format(string))
```

```
def _print_stderr(batch, string):
    logging.debug("stderr: {}".format(string))

batch.connect('finished-event', _test_succeed_finished_cb)
batch.connect('stdout-event', _print_stdout)
batch.connect('stderr-event', _print_stderr)

batch.start()

main_loop.run()

assert batch.was_successful() == True

batch.write_archive(OUTPUTFILE)

XmiMsimGui.init()

archive = batch.props.archive

win = XmiMsimGui.XmsaViewerWindow.new(archive)
win.set_position(Gtk.WindowPosition.CENTER)
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

To use this functionality on Windows, launch an XMI-MSIM prompt from the Start Menu: the terminal will give you access to a Python3 interpreter (python3.exe) as well as launching the GUI (xmimsim-gui.exe). The Python installation contains xraylib, numpy, scipy and matplotlib, and should be enough to get you started. If not sufficient, use pip in this shell to install additional Python packages from PyPI (at your own risk!).
To do this on macOS, you will need to install XMI-MSIM using Homebrew (or from source!), and use it entirely from the command line. The app bundle and the Homebrew installation may be installed simultaneously, they won't conflict.
Linux users can do this using their terminals. You may need to install PyGobject first with your package manager though.

4.5 Custom detector response functions

As already mentioned in simulation options, it has become possible to use custom detector response functions in XMI-MSIM, thereby replacing its built-in routines with your own alternatives. Basically, you will have to write your own routine called `xmi_detector_convolute_all_custom` and

compile this function into a plug-in (dynamically loadable module). This routine should follow the prototype:

```
void xmi_detector_convolute_all_custom(xmi_inputFPtr inputFPtr, double **channels_no
```

or in Fortran:

```
INTERFACE
SUBROUTINE xmi_detector_convolute_all_custom(&
        inputFPtr,&
        channels_noconvPtr,&
        channels_convPtr,&
        brute_historyPtr,&
        var_red_historyPtr,&
        options,&
        escape_ratiosCPtr,&
        n_interactions_all,&
        zero_inter) &
        BIND(C,NAME='xmi_detector_convolute_all')
        IMPLICIT NONE
        TYPE (C_PTR), INTENT(IN), VALUE :: inputFPtr
        TYPE (C_PTR), INTENT(IN), VALUE :: channels_noconvPtr
        TYPE (C_PTR), INTENT(IN), VALUE :: channels_convPtr
        TYPE (C_PTR), INTENT(IN), VALUE :: var_red_historyPtr
        TYPE (C_PTR), INTENT(IN), VALUE :: brute_historyPtr
        TYPE (xmi_escape_ratiosC), INTENT(IN) :: escape_ratiosCPtr
        TYPE (xmi_main_options), VALUE, INTENT(IN) :: options
        INTEGER (C_INT), VALUE, INTENT(IN) :: n_interactions_all, zero_inter
ENDSUBROUTINE
ENDINTERFACE
```

Clearly for this to work you will have to make use of the datatypes exposed in the headers and the fortran module files. I recommend also to make as much as possible of the functions that are already contained within XMI-MSIM (a fully documented API will follow at some point...). For now, the best source of information would be the code itself, along with the two examples that are included in the distribution (Fortran and C).
Probably it will be easiest to write such a plug-in in Fortran: you will be able to easily reuse a lot of the existing code with minimal effort. However, if you're willing to write some simple wrappers in Fortran first, you can also use all Fortran functions from C. When writing a plug-in in C, you will probably want to use the following function:

```
void xmi_input_F2C(xmi_inputFPtr Ptr, struct xmi_input **xmi_inputC);
```

This function extracts from the Fortran datatype `TYPE (xmi_input)` (which is useless in C), the corresponding C datatype `struct xmi_input`, which contains all the easily accessable information from the simulation.
It should also be possible to write plug-ins in C++, just make sure the exported function gets the `extern "C"` attribute.
Here are some instructions on how to compile these modules on the three supported platforms. The resulting `module.so` (Mac OS X and Linux) or `module.dll` (Windows) should be loadable by XMI-MSIM. I recommend to make use of OpenMP as shown in the Fortran example (but possible in the C example too). If not desired, then remove the `-fopenmp` flags. You are of course free to make use of any other libraries that are necessary to implement your detector response functions. In this case however, do not forget to add the required header and linker flags.

4.5.1 Building modules on macOS

The following instructions apply to the XMI-MSIM app bundle only, and assume it is installed in `/Applications`. If compiled from source, follow the instructions for Linux instead.

Fortran The latest release of XMI-MSIM (8.0) has been compiled with gfortran 9, installed using Homebrew. Do not attempt to try a different fortran compiler, or even a different version of gfortran: it won't work! What should work (though I haven't tested this), is a gfortran 9 compiler installed with MacPorts or Fink.
Compile your source files (to be executed for each source file separately):

```
gfortran-9 -I/Applications/XMI-MSIM.app/Contents/Resources/include/xmimsim -fopenmp
```

Link the objects together:

```
gfortran-9 -fopenmp -Wl,-undefined -Wl,dynamic_lookup -o module.so -bundle object1.o
```

C Feel free to use any C compiler you want for this: the default clang, or any C compiler offered by Homebrew. Keep in mind though that the macOS system clang currently doesn't support OpenMP. clang as provided by Homebrew and MacPorts does support OpenMP.
Compile your source files (to be executed for each source file separately):

```
clang -I/Applications/XMI-MSIM.app/Contents/Resources/include/xmimsim -c -fno-common
```

Link the objects together:

```
clang -Wl,-undefined -Wl,dynamic_lookup -o module.so -bundle object1.o object2.o ...
```

4.5.2 Building modules on Linux

The following instructions assume that XMI-MSIM was installed using binary packages for selected Debian/Ubuntu and Redhat family distributions, include the development packages!. If compiled from source, you may have to set the `PKG_CONFIG_PATH` variable for the following instructions to work.

Fortran Compile your source files (to be executed for each source file separately):

```
gfortran `pkg-config --cflags libxmimsim` -fopenmp -ffree-line-length-none -c -fPIC
```

Link the objects together:

```
gfortran -fopenmp -shared -fPIC -Wl,-soname -Wl,module.so -o module.so object1.o obj
```

C Compile your source files (to be executed for each source file separately):

```
gcc `pkg-config --cflags libxmimsim` -c -fPIC -DPIC -o object1.o source1.c
```

Link the objects together:

```
gcc -fopenmp -shared -fPIC -Wl,-soname -Wl,module.so -o module.so object1.o object2
```

4.5.3 Building modules on Windows

For any of the following to work, ensure you selected the SDK when asked which components to install when running the installer!! Assuming you didn't change the default installation path during installation, XMI-MSIM should be installed in `C:\Program Files\XMI-MSIM 64-bit`
For the sake of convenience, let's call this installation folder *xmi_msim*. If for some reason you picked another installation directory, assume *xmi_msim* represents this one.
XMI-MSIM for Windows has been compiled with MinGW-w64 compilers offered through MSYS2. In order for this to work, you will have to install the exact same compilers on your system: download MSYS2 and install the gcc compiler suite
After installation, fire up a MinGW/MSYS2 shell from the Start Menu entries that were just created and cd to the directory that contains your source. Depending on what language was written, follow either the Fortran or C instructions, keeping in mind that *xmi_msim* refers to the full path to the XMI-MSIM installation folder!!!

Fortran Fortran does not have a standardized way of exporting a function from a dll. The following should work for both gfortran and Intel Fortran: include these lines just after the definition of the `xmi_detector_convolute_all_custom` function.

```
#ifdef __GFORTRAN__
!GCC$ ATTRIBUTES DLLEXPORT:: xmi_detector_convolute_all_custom
#elif defined(__INTEL_COMPILER)
!DEC$ ATTRIBUTES DLLEXPORT:: xmi_detector_convolute_all_custom
#endif?
```

Compile your source files (to be executed for each source file separately):

```
gfortran -I"xmi_msim\SDK\Include" -fopenmp -ffree-line-length-none -DDLL_EXPORT -c
```

Link the objects together:

```
gfortran -shared -fopenmp -Wl,--enable-auto-image-base -o module.dll object1.o objec
```

C Exporting the `xmi_detector_convolute_all_custom` requires that the definition is proceded by `__declspec(dllexport)`, as shown in the C example where this is accomplished using conditional compilation.
Compile your source files (to be executed for each source file separately):

```
gcc -mms-bitfields -I"xmi_msim\SDK\Include" -fopenmp  -DDLL_EXPORT -c -o object1.o s
```

Link the objects together:

```
gcc -mms-bitfields -shared -fopenmp -Wl,--enable-auto-image-base -o module.dll objec
```

In theory you should also be able to compile a module from C source code using Visual Studio, but I have not tried this yet. For this, you will need the import library `libxmimsim-0.lib`, which is located in the SDK\Lib subdirectory of your XMI-MSIM installation.

5 The XMI-MSIM API: list of functions

(under construction)
XMI-MSIM exposes a large number of its internal functions for use in other programs. This has been used for example in the XMI-MSIM plug-in of the **XRMC software package for X-ray imaging and spectroscopy**, giving it access to the detector convolution routines, thereby enabling the simulation of ED-XRF spectrometers.
In order to access the XMI-MSIM functionality, you will need to include its headers and link against its library. On Linux and Mac OS X (when compiled from source!), this can be most easily accomplished using our pkg-config file:

```
gcc $(pkg-config --cflags libxmimsim) myprogram.c $(pkg-config
--libs libxmimsim)
```

We intend to include a static library that includes all exported symbols in our next Windows release, allowing you to start development on this platform as well.
Make sure you include the following header in your code:
`#include <xmi_msim.h>`
The following sections show a list of all exported functions, per header. It is not recommended to include these headers directly.

6 References and addtional resources

XMI-MSIM is the successor to the msim program of Prof. Laszlo Vincze of Ghent University. Together we have published six papers over a course of 20 years that cover our work on Monte Carlo simulations of ED-XRF spectrometers.

6.1 Papers by Laszlo Vincze et al.

- A General Monte-Carlo Simulation of Energy-Dispersive X-ray-Fluorescence Spectrometers Part 1. Unpolarized Radiation, Homogeneous Samples. Laszlo Vincze, Koen Janssens and Freddy Adams. Spectrochimica Acta Part B, 48(4), 553-573, 1993. DOI
- A General Monte-Carlo Simulation of Energy-Dispersive X-ray-Fluorescence Spectrometers Part 2. Polarized monochromatic radiation, homegeneous samples. Laszlo Vincze, Koen Janssens, Fred Adams, M.L. Rivers and K.W. Jones. Spectrochimica Acta Part B, 50(2), 127-147, 1995. DOI
- A General Monte-Carlo Simulation of Energy-Dispersive X-ray-Fluorescence Spectrometers Part 3. Polarized polychromatic radiation, homogeneous samples. Laszlo Vincze, Koen Janssens, Fred Adams and K.W. Jones. Spectrochimica Acta Part B, 50(12), 1481-1500, 1995. DOI
- A General Monte-Carlo Simulation of Energy-Dispersive X-ray-Fluorescence Spectrometers Part 4. Photon scattering at high X-ray energies. Laszlo Vincze, Koen Janssens, Bart Vekemans and Fred Adams. Spectrochimica Acta Part B, 54(12), 1711-1722, 1999. DOI

6.2 Papers by Tom Schoonjans et al.

- A General Monte-Carlo Simulation of Energy-Dispersive X-ray-Fluorescence Spectrometers Part 5. Polarized radiation, stratified samples, cascade effects, M-lines. Tom Schoonjans, Laszlo Vincze, Vicente Armando Solé, Manuel Sanchez del Rio, Philip Brondeel, Geert Silversmit, Karen

Appel, Claudio Ferrero. Spectrochimica Acta Part B, 70, 10-23, 2012. DOI

- A General Monte-Carlo Simulation of Energy-Dispersive X-ray-Fluorescence Spectrometers Part 6. Quantification through iterative simulations. Tom Schoonjans, Laszlo Vincze, Vicente Armando Solé, Manuel Sanchez del Rio, Karen Appel, Claudio Ferrero. Spectrochimica Acta Part B, 82, 36-41, 2013. DOI

6.3 Posters by Tom Schoonjans et al.

- A general Monte Carlo simulation of ED-XRF spectrometers. New developments. Tom Schoonjans, V. Armando Solé, Manuel Sanchez del Rio, Claudio Ferrero and Laszlo Vincze. ICXOM 2011 conference, Campinas, Brazil. 5-8 September 2011.
- XMI-MSIM: A general Monte Carlo simulation of ED-XRF spectrometers. Tom Schoonjans, Laszlo Vincze, V. Armando Solé, Manuel Sanchez del Rio and Claudio Ferrero. European Conference on X-Ray Spectrometry, Alma Mater Studiorum Università di Bologna, Italy, 15-20 June 2014
- XMI-MSIM: A general Monte Carlo simulation of ED-XRF spectrometers. Tom Schoonjans, Laszlo Vincze, V. Armando Solé, Manuel Sanchez del Rio and Claudio Ferrero. European Conference on X-Ray Spectrometry, University of Gothenburg, Sweden, 19-24 June 2016 PDF

6.4 Oral presentations by Tom Schoonjans et al.

- Quantification of ED-XRF datasets through iterative Monte Carlo simulations: new developments. Tom Schoonjans, Claudio Ferrero, V.Armando Solé, Manuel Sanchez del Rio, Geert Silversmit, Karen Appel and Laszlo Vincze. EXRS 2012 conference, Vienna, Austria. 18-22 June 2012.
- A general Monte Carlo simulation of energy-dispersive X-ray fluorescence spectrometers. Tom Schoonjans. Monte Carlo simulation tools for X-ray imaging and fluorescence workshop, ESRF, Grenoble, France, 24-25 February 2014. PDF
- An introduction to Monte Carlo Methods in XRF analysis and a tutorial on Monte Carlo methods in XRF analysis. Tom Schoonjans. Joint ICTP-IAEA School on Novel Experimental Methodologies for Synchrotron Radiation Applications in Nano-science and Environmental Monitoring, Trieste, Italy, 17-28 November 2014. Slides talk and slides tutorial
- A general Monte Carlo simulation of energy-dispersive X-ray fluorescence spectrometers. Tom Schoonjans. EXSA Workshop on Quantita-

tive methods in X-Ray Spectrometry, Berlin, Germany, 11-12 October 2017.