XaNSoNS

Xray and Neutron Scattering on Nanoscale Structures

Author: Vladislav Neverov, vs-never@hotmail.com, neverov_vs@nrcki.ru

User manual

XaNSoNS is a free open source software with GPU support that simulates 2D and 1D x-ray or neutron diffraction patterns and PDF functions for the ordered or disordered nanoscale structures (up to 10^7 atoms).

Table of Contents

I. Introduction

XaNSoNS is a free open source software with GPU support that simulates x-ray and neutron 1D (or 2D) diffraction patterns and pair-distribution functions (PDF) (partial and full) for amorphous or crystalline atomic ensembles of heterogeneous structural content.

XaNSoNS uses general equations (non-specific to crystalline structures) to calculate the scattering intensity. These equations do not consider multiple scattering, inelastic scattering and absorption. The equations used depend on the simulation scenario (see Eq. (**1**)-(**11**) in **Sec. III**). XaNSoNS does not use the Debye-Waller factor to take into account the thermal motion of the atoms. However, this effect may be considered if the user specifies the atomic displacement parameters (see the example in **Sec. IV.3**).

XaNSoNS simulates the diffraction on a single nanoparticle (up to 10^7 atoms) and does not support the periodic boundary conditions (PBC) for now. This nanoparticle may be of heterogeneous structural content, namely, it may contain multiple blocks with variable structural parameters, to be defined by the user for each type of the structural block. Among these parameters are atomic displacements, site occupancies, molecular displacements and random molecular rotations (the latter was introduced specially for the spherical molecules, e.g. fullerenes). Note, that lack of PBC support is not necessary a shortcoming in the diagnostics of amorphous materials. The single nanoparticle may represent a volume element of a sample, however, it is important to keep in mind that certain values may be distorted, namely, (i) the scattering intensity for the low values of scattering vector magnitude and (ii) the PDF for the high values of interatomic distance.

The code supports parallel computing and does calculations on both the CPU and the GPU. The CPU version uses double-precision floating point math, while GPU version uses single-precision floating point math and intrinsic functions. There are six versions of the executable that may be built by the user, all have identical features but use different types of parallelization and are optimized for different computational devices and architectures. Namely, MPI or MPI+OpenMP versions may be used on the CPU-only high performance clusters (HPCs); OpenMP version may be used on ordinary PCs; CUDA and OpenCL versions may be used on both the workstations, equipped with GPU-based co-processors, and the consumer-level PCs, equipped with modern GPU (discrete or integrated). For the instructions on how to build different versions of XaNSoNS for different platforms see **Sec. II**. The ability to simulate diffraction patterns and PDFs on the GPU is the most powerful feature of the application. The user is advised to use the GPU version of XaNSoNS if the proper GPU is available in the system. E.g., the simulation of the powder (1D) diffraction pattern for the ensemble of 10^6 atoms with the resolution of 1024 point by scattering vector magnitude takes the time of about 1 minute on a single Nvidia GTX Titan GPU. On a modern top-level CPU, it takes 20 – 40 times longer.

All the simulation parameters including the parameters of the model sample are described in the XML file. The detailed description of the format of this XML file is given in **Sec. III.1**. For the instructions on how to run XaNSoNS from the console see **Sec. III.2**.

To simplify the processing of the experimental data, XaNSoNS can be called in the python script with the help of the API for Python. This API is described in **Sec. III.3.** The example of the experimental data processing which combines XaNSoNS features with the powerful tools of SciPy python package (http://www.scipy.org/) is given in **Sec. IV.5**).

While the source code is original, many ideas and methods are taken from the other works. The works are listed below are the main ones.

1. T. Egami, S.J.L. Billinge. Underneath the Bragg peaks: structural analysis of complex

materials. Elsevier, 2003.

This book details the problems that arise in the structural analysis of amorphous materials. It provides the methods to solve these problems and describes the general theory that underlies the methods. Anyone who is new to the problem is advised to read this book.

2. Debyer software (https://github.com/wojdyr/debyer) by Marcin Wojdyr.

   Debyer calculates diffraction patterns (using the Debye equation), PDFs and structure functions. The idea to use the histogram of interatomic distances to calculate the scattering intensity was taken from this software. This code is parallel (versions with MPI or OpenMP support may be built) but does not support GPU computing. Also it supports the PBC, the feature not supported by XaNSoNS for now.

3. A. Lønning Reiten. X-ray scattering simulations using GPU-enabled algorithms. Norwegian University of Science and Technology, preprint (2010), http://nanowiki.no/images/0/0f/Gpu_scattering_v1.pdf.

   This preprint contains the essential algorithms on how to efficiently calculate the diffraction pattern on the GPU using the Debye equation. Although, the algorithms used in XaNSoNS are slightly different, some of the ideas, e.g. the idea to sort the atoms on their chemical elements before starting the calculation to reduce the number of GPU memory operations, were taken from this preprint. Also this preprint contains the comprehensive research on how the use of single-precision math and intrinsic functions affects the accuracy of calculations.

In addition to the dependencies listed in the **Sec. II.1** XaNSoNS integrates TinyXML-2 parser (https://github.com/leethomason/tinyxml2), which is used to parse the configuration of parameters from the XML file.

XaNSoNS was used to interpret the experimental data in a few works. These two are the major ones:

1. A.B. Kukushkin, V.S. Neverov, N.L. Marusov, I.B. Semenov, B.N. Kolbasov, V.V. Voloshinov, A.P. Afanasiev, A.S. Tarasov, V.G. Stankevich, N.Yu. Svechnikov, A.A. Veligzhanin, Ya.V. Zubavichus, L.A. Chernozatonskii, “Few-nanometer-wide carbon toroids in the hydrocarbon films deposited in tokamak T-10”, Chem. Phys. Lett. 506 (2011) 265.
2. V.S. Neverov, P.A. Borisova, A.B. Kukushkin, V.V Voloshinov, “A method for diffraction-based identification of amorphous sp^2 carbon materials”, J. Non-Cryst. Solids, 2015, 427, 166-174.

XaNSoNS is used in the BOINC project XANSONS for COD (http://xansons4cod.com) aimed to create an open access database of simulated x-ray and neutron powder diffraction patterns for the nanocrystalline phase of the materials contained in the Crystallography Open Database (http://www.crystallography.net/cod/index.php).

The author is grateful to A.B. Kukushkin and V.Yu. Murzin, for helpful discussion and advices on all stages of the work, A.A. Kulichenko, for testing the software, P.A. Borisova, for providing the experimental data for **Sec. IV.5** and all volunteers participating in the BOINC project XANSONS for COD (http://xansons4cod.com).

The development of XaNSoNS was made possible thanks to the support of the Russian Foundation for Basic Research (projects RFBR #12-07-00529-a and #15-07-07901-a).

II. Installation

To better understand the process described in this section one should understand the structure of the application. This structure is shown in Fig. **1**. The software consists of the wrapper, written in Python, and the main module, written in C/C++.

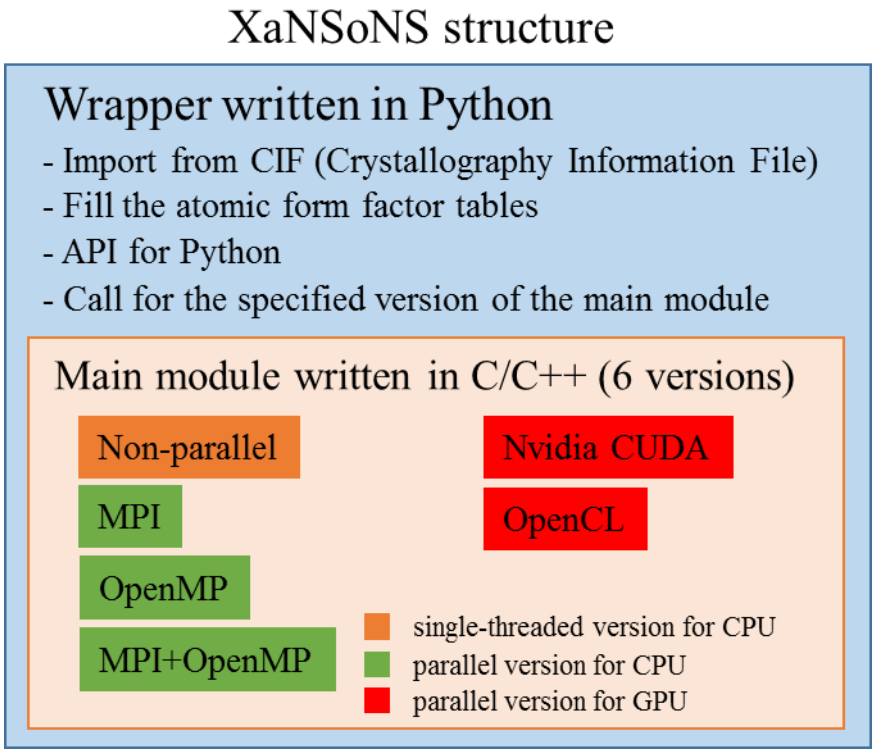

Fig. 1. The structure of XaNSoNS

The main module does all the heavy computation. The wrapper itself has a modular structure. It has the modules: (**i**) to import the data from the crystallography information file (CIF) format with the help of the PyCifRW python library; (**ii**) to fill the tables of x-ray atomic form factors or neutron scattering lengths for the specific model sample with the help of the Periodictable python library; (**iii**) to read, write and edit the configuration of parameters of the simulation. The wrapper calls for the specified version (out of six) of the main module. Also, the wrapper implements the application programming interface (API) for Python, enabling one to use XaNSoNS in a python application.

The main module of XaNSoNS has several versions for various computing architectures (see Table **1**).

Table 1. Six versions of XaNSoNS main module.

Executable name	Comp. device	Precision	Parallelism	Purpose
XaNSoNS	CPU	Double	No	Accuracy control and debugging
XaNSoNS_OMP	CPU	Double	Yes (OpenMP)	Simulation on single PCs and workstations.
XaNSoNS_MPI	CPU	Double	Yes (MPI)	Simulation on CPU-based HPC.
XaNSoNS_MPI_OMP	CPU	Double	Yes (MPI + OpenMP) Each MPI process	Simulation on CPU-based HPC with proper resource management (separate

			parallelized with multiple OpenMP threads.	node should be allocated for each MPI process).
XaNSoNS_CUDA	GPU	Single	Yes (Nvidia CUDA)	Simulation on Nvidia GPU with CUDA compute capability of 1.3 or higher.
XaNSoNS_OCL	GPU	Single	Yes (OpenCL)	Simulation on AMD, Intel or Nvidia GPU with OpenCL support (version 1.2 or higher).

XaNSoNS depends on a few libraries and packages that must be installed to run the application.

II.1 Dependencies

The following dependencies are required to run any version of XaNSoNS.

- Python 2.7.x (Python 3.x is not supported yet due to PyCifRW package),
- Numpy python package,
- Periodictable python package (is required to fill the tables of x-ray atomic form factors and neutron scattering lengths),
- PyCifRW python package (is required to import unit cell configuration from CIF files).

XaNSoNS distribution includes pre-built binaries for Windows, Linux and Mac OS. *Note, MPI and MPI+OpenMP binaries are not included for Linux and Mac OS.* The requirements to run these binaries are platform specific:

Windows

- Windows 7 SP1 64-bit and above.
- Nvidia GPU with CUDA Compute Capability 1.3 and above and Nvidia driver 340.21 or newer for **CUDA** version.
- AMD, Nvidia or Intel GPU with OpenCL 1.2 (or above) support for **OpenCL** version.
- Visual C++ Redistributable Packages for Visual Studio 2013 (64-bit version) for **OpenMP** version.
- Microsoft MPI v7 SDK for **MPI** and **MPI+OpenMP** versions.

Linux

- 64-bit Linux with kernel 3.16 or newer.
- Nvidia GPU with CUDA Compute Capability 1.3 and above and Nvidia driver 340.21 or newer for **CUDA** version.
- AMD, Nvidia or Intel GPU with OpenCL 1.2 (or above) support for **OpenCL** version.

Mac OS

- OS X 10.9.5 or newer (including macOS).
- Nvidia GPU with CUDA Compute Capability 2.0 and CUDA Driver for MAC 7.0.29 or newer for **CUDA** version.
- AMD, Nvidia or Intel GPU with OpenCL 1.2 (or above) support for **OpenCL** version.

The installation process is described below for the specific platform.

II.1.a Windows

The easiest way to install python is to use the scientific python distributions like Anaconda (https://www.continuum.io/downloads) or Python(x,y) (https://python-xy.github.io/downloads.html). These distributions also include Numpy python package.

The Periodictable and PyCifRW may be installed via pip or Setuptools. Install Anaconda (*make sure that Python 2 version is selected*) or Python(x,y) using the instructions provided on the sites. *Note, avoid non-ascii characters in the install path or user directory path.* Regardless of the choice of the python distribution (Anaconda or Python(x,y)) at least one of the package managers (pip or Setuptools) will be installed by default.

Microsoft Visual C++ Compiler for Python 2.7 is required to install PyCifRW. Install it using this link: http://aka.ms/vcpython27.

After the python distribution and the compiler are installed, the Periodictable and PyCifRW can be installed via Setuptools:

```
easy_install periodictable
easy_install PyCifRW
```

or via pip:

```
pip install periodictable
pip install PyCifRW
```

To run **CUDA** version, make sure that your Nvidia GPU has CUDA Compute Capability 1.3 and above (check it here, https://en.wikipedia.org/wiki/CUDA#GPUs_supported) and the Nvidia driver 340.21 or newer is installed (if not, download it from the Nvidia site: http://www.nvidia.com/download/index.aspx)

To run **OpenCL** version, make sure that your GPU (and the driver) supports OpenCL 1.2 or above. To check the supported version of OpenCL use this link for Nvidia GPUs: https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units, this link for AMD GPUs: https://en.wikipedia.org/wiki/List_of_AMD_graphics_processing_units, and this one for Intel GPUs: https://en.wikipedia.org/wiki/List_of_Intel_graphics_processing_units.

To run **OpenMP** version, install 64-bit version of Visual C++ Redistributable Packages for Visual Studio 2013 using this link https://www.microsoft.com/en-US/download/details.aspx?id=40784.

To run **MPI** and **MPI+OpenMP** versions, install Microsoft MPI v7 SDK using this link https://www.microsoft.com/en-us/download/details.aspx?id=49926.

When all the requirements are installed, just unpack the archive **Win64_executables.7z** into the XaNSoNS root directory with the source code. Make sure that the extracted executable files are in the root directory and **not** in the **Win64_executables** directory**.**

II.1.b Linux

Python 2.7.x and Numpy can be easily installed via the default package manager. Install pip or Setuptools the same way. Then, Periodictable and PyCifRW can be installed via Setuptools:

```
sudo easy_install periodictable
sudo easy_install PyCifRW
```

or via pip:

```
sudo pip install periodictable
sudo pip install PyCifRW
```

If `sudo` command is not available, use `su` command to get the super user privileges first and then do the commands above without `sudo`.

To run **CUDA** version, make sure that your Nvidia GPU has CUDA Compute Capability 1.3 and above (check it here, https://en.wikipedia.org/wiki/CUDA#GPUs_supported) and the proprietary Nvidia driver 340.21 or newer is installed. **CUDA programs do not work with open source nouveau driver.** If proprietary Nvidia driver is not available in your distro's repositories, download it from the Nvidia site: http://www.nvidia.com/download/index.aspx.

Note, in some Linux distros, running CUDA or OpenCL applications requires root access. To solve this issue, install nvidia-modprobe via the package manager or follow the solution from this thread: http://stackoverflow.com/questions/11104320/root-access-required-for-cuda.

To run **OpenCL** version, make sure that your GPU supports OpenCL 1.2 or above. To check the supported version of OpenCL use this link for Nvidia GPUs: https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units, this link for AMD GPUs: https://en.wikipedia.org/wiki/List_of_AMD_graphics_processing_units, and this one for Intel GPUs: https://en.wikipedia.org/wiki/List_of_Intel_graphics_processing_units. The proprietary driver for Nvidia or AMD GPUs should be installed to run OpenCL programs on these devices.

When all the requirements are installed, just unpack the archive **Linux64_executables.tgz** into the XaNSoNS root directory with the source code. Make sure that the extracted executable files are in the root directory and **not** in the **Linux64_executables** directory**.**

Note, the pre-built OpenCL Linux binary has some issues with the portability (only Nvidia GPUs are affected). If it's not working for you, use the CUDA version or build the OpenCL binary from the source by yourself.

II.1.c Mac OS

It is not recommended to use the default versions of Python 2.7.x and Numpy installed in the system. The latest fully functional versions of Python 2 and Numpy can be installed via Anaconda python distribution (https://www.continuum.io/downloads) or via MacPorts (https://www.macports.org/). The Xcode Developer Tools with the command line tools are required to install PyCifRW package. Install the Xcode from the Mac App Store. Accept the Xcode EULA either by running Xcode or by doing in the Terminal:

```
xcodebuild --license
```

Then install the Xcode Command Line Tools by doing in the Terminal:

```
xcode-select --install
```

Anaconda python distribution. Install Anaconda (*make sure that Python 2 version is selected*) using the instructions provided on the site. Then, do the following commands in the Terminal to install the Periodictable and PyCifRW packages:

```
~/anaconda/bin/pip install periodictable
~/anaconda/bin/pip install PyCifRW
```

MacPorts. Install MacPorts base using the instruction provided on the site: https://www.macports.org/install.php. Then, run the following commands in the Terminal:

```
sudo port install python27
sudo port select --set python python27
sudo port install py27-numpy
sudo port select py27-setuptools
sudo easy_install-2.7 periodictable
sudo easy_install-2.7 PyCifRW
```

To run **CUDA** version, make sure that your Nvidia GPU has CUDA Compute Capability 1.3 and above (check it here, https://en.wikipedia.org/wiki/CUDA#GPUs_supported). Install CUDA Driver for MAC 7.0.29 or newer using this link, http://www.nvidia.com/object/mac-driver-archive.html.

To run **OpenCL** version, make sure that your GPU supports OpenCL 1.2 or above. To check the supported version of OpenCL use this link for Nvidia GPUs: https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units, this link for AMD GPUs: https://en.wikipedia.org/wiki/List_of_AMD_graphics_processing_units, and this one for Intel GPUs: https://en.wikipedia.org/wiki/List_of_Intel_graphics_processing_units.

When all the requirements are installed, just unpack the archive **MacOS_executables.zip** into the XaNSoNS root directory with the source code. Make sure that the extracted executable files are in the root directory and **not** in the **MacOS_executables** directory**.**

II.2 Building from source

Note, if the provided executables work for you, there is no need to build them by yourself.

The six versions of the main module of XaNSoNS can be built from the source (see Table 1).

The information on how to build all six versions of the main module of XaNSoNS for the specific platform is given below. *Note, there is no need to build them all, build only those versions that are required.*

II.2.a Windows

The project file for the Microsoft Visual Studio is included in the software distribution. The project is configured for the Microsoft Visual Studio 2013 Community Edition (https://www.visualstudio.com/en-us/news/vs2013-community-vs.aspx), the Nvidia CUDA Toolkit 6.5 (https://developer.nvidia.com/cuda-toolkit-65) (**CUDA** and **OpenCL** versions) and the Microsoft MPI v7 SDK (https://www.microsoft.com/en-us/download/details.aspx?id=49926) (**MPI** and **MPI+OpenMP** versions). *Note, to install the Nsight plugin, the Microsoft Visual Studio must be installed prior to the Nvidia CUDA Toolkit.*

The project has six solution configurations for x64 platform with self-explanatory names: Release, Release_CUDA, Release_MPI, Release_MPI_OMP, Release_OCL, Release_OMP. To build any configuration, select it and press F7.

To change the platform toolset if the newer (older) version of the Visual Studio is used, do the following.

a. Open project properties with Alt+F7.
b. Select All Configurations in the Configuration list.
c. Go to Configuration Properties → General.
d. Change the value in the Platform Toolset from Visual Studio 2013 (v120) to the toolset used.

To change the CUDA toolkit configuration for the **CUDA** version of the application, do the following.

a. Select Release_CUDA configuration.
b. Select XaNSoNS in Solution Explorer.
c. Go to Project → Build Customizations.
d. Select the CUDA build customizations of the proper version.

The CUDA Toolkit version 7.0 or higher does not support the GPUs with CUDA Compute Capability (CC) number below 2.0. To remove the compilation rules for devices with CC < 2.0, do the following.

e. Open project properties with Alt+F7.
f. Go to Configuration Properties → CUDA C/C++ → device.
g. Remove compute_13 and sm_13 compilation options in the Code Generation field.

Note, the file CalcFunctionsCUDA.cu is visible in the solution explorer only if the project is opened in the Visual Studio with the "Release_CUDA" as a starting configuration. To make CalcFunctionsCUDA.cu visible (if it is not) select the "Release_CUDA" configuration and restart the Visual Studio.

To change the OpenCL SDK configuration for the **OpenCL** version of the application, do the following.

a. Open project properties with Alt+F7.
b. Select Release_OCL configuration in the Configuration list.
c. Go to Configuration properties → C/C++ → General.
d. Change "C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v6.5\include" path in "Additional Include Directories" field to the proper path for include directory of the OpenCL SDK.
e. Go to Configuration properties → Linker → General.
f. Change "C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v6.5\lib\x64" path in "Additional Library Directories" field to the proper path for library directory of the OpenCL SDK.

Three major GPU vendors provide the OpenCL SDKs: Nvidia CUDA Toolkit (https://developer.nvidia.com/cuda-toolkit), AMD APP SDK (http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/), Intel SDK for OpenCL applications (https://software.intel.com/en-us/intel-opencl/download).

Note, the OpenCL implements runtime compilation of the kernels. So, the executable built with OpenCL is vendor independent (e.g. the executable built with the Nvidia CUDA Toolkit will run on the AMD GPU).

To change the MPI SDK configuration for the **MPI** and **MPI+OpenMP** versions of the application, do the following.

a. Open project properties with Alt+F7.
b. Change configuration to "Multiple Configurations…" then select "Release_MPI" and "Release_MPI_OMP" configurations.
c. Go to Configuration properties → C/C++ → General.
d. Change "C:\Program Files %28x86%29\Microsoft SDKs\MPI\Include" path in "Additional Include Directories" field to the proper path for include directory of the selected MPI distribution.

e. Go to Configuration properties → Linker → General.
f. Change "C:\Program Files %28x86%29\Microsoft SDKs\MPI\Lib" path in "Additional Library Directories" field to the proper path for library directory of the selected MPI distribution.

II.2.b Linux

The Makefile is configured for the GCC C++ compiler. The code can be compiled with any modern version of GCC. The GCC and Make can be easily installed via the default package manager.

To build the **single-threaded** version, in the XaNSoNS root directory do:

```
make
```

To build the **OpenMP** version, do:

```
make OpenMP=1
```

The Nvidia CUDA Toolkit should be installed to build the **CUDA** version. Many Linux distros include CUDA Toolkit in their repos. For example, to install CUDA Toolkit in Ubuntu, do:

```
sudo apt-get install nvidia-cuda-toolkit nvidia-cuda-dev
```

If the CUDA Toolkit is not available in the repos, it must be installed using one of the installers from the Nvidia site: https://developer.nvidia.com/cuda-downloads. If the CUDA Toolkit was installed this way, make sure that your .profile or .bashrc file contains the following lines:

```
export PATH="/path/to/cuda/bin:$PATH"
export LD_LIBRARY_PATH="/path/to/cuda/lib64:$LD_LIBRARY_PATH"
export LIBRARY_PATH="/path/to/cuda/lib64:$LIBRARY_PATH"
```

where `/path/to/cuda` is the path where CUDA Toolkit was installed.

When the CUDA Toolkit is installed, do:

```
make CUDA=1
```

to build the CUDA version.

The CUDA version built that way supports the GPUs with the CUDA Compute Capability (CC) 2.0 or higher. To build the version for the GPUs with the CC 1.3, install the Nvidia CUDA Toolkit 6.5 (this will also require to install the compatible version of the GCC) or older and do:

```
make CUDA=1 TESLA=1
```

Note, the version of the Nvidia CUDA Toolkit should be compatible with the version of the GCC (*see the Nvidia CUDA Toolkit documentation,* http://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html#system-requirements).

To build the **OpenCL** version, install the OpenCL headers via the package manager first (search for opencl-headers). Then install the OpenCL SDK depending on your GPU vendor.

Three major GPU vendors provide the OpenCL SDKs: Nvidia CUDA Toolkit (https://developer.nvidia.com/cuda-toolkit), AMD APP SDK (http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/), Intel SDK for OpenCL applications (https://software.intel.com/en-us/intel-opencl/download).

On how to install the Nvidia CUDA Toolkit see the instruction for the CUDA version above.

To install and configure the AMD APP SDK, see the Installation Notes: http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/AMD_APP_SDK_InstallationNotes.pdf

To install the Intel SDK for OpenCL, see the Section "Installation on Linux OS" in the Release Notes for the specific version of the SDK, https://software.intel.com/en-us/articles/opencl-code-builder-release-notes.

Whichever OpenCL SDK was installed, make sure that the `LD_LIBRARY_PATH` and `LIBRARY_PATH` environmental variables are updated. Then do:

```
make OpenCL=1
```

to build the OpenCL version.

To build the **MPI** and **MPI+OpenMP** versions, the MPI distribution should be installed. Install any MPI distribution (e.g. MPICH or OpenMPI) you want via the package manager. Then do:

```
make MPI=1
```

to build the MPI version and do:

```
make MPI=1 OpenMP=1
```

to build the MPI+OpenMP version.

II.2.c Mac OS

The Makefile is configured to use the Apple Clang C++ compiler (included in the Xcode Developer Tools) to build **single-threaded**, **CUDA** and **OpenCL** versions and the GCC C++ compiler to build **OpenMP**, **MPI** and **MPI+OpenMP** versions. Don't forget to install the Xcode Command Line Tools by doing in the Terminal:

```
xcode-select --install
```

The easiest way to obtain the GCC compiler is by installing the MacPorts (https://www.macports.org/). Install MacPorts base using the instruction provided on the site: https://www.macports.org/install.php. Then, in the Terminal do:

```
sudo port install gcc6
```

To build the **single-threaded** version, in the XaNSoNS root directory do:

```
make
```

To build the **OpenMP** version, do:

```
make OpenMP=1
```

To build the **OpenCL** version, do:

```
make OpenCL=1
```

To build the **OpenCL** version, do:

Note, unlike Linux, all necessary components to build and run OpenCL applications are installed in Mac OS by default.

The Nvidia CUDA Toolkit is required to build the **CUDA** version. Download the .dmg package from the official website: https://developer.nvidia.com/cuda-downloads. Install the Toolkit and the CUDA driver for Mac following the instructions. Once the CUDA Toolkit is installed, do:

```
make CUDA=1
```

The CUDA version built that way supports the GPUs with the CUDA Compute Capability (CC) 2.0 or higher. To build the version for the GPUs with the CC 1.3, install the Nvidia CUDA Toolkit 6.5 or older (this will also require to install the compatible version of the Xcode Developer Tools) and do:

```
make CUDA=1 TESLA=1
```

To build the **MPI** and **MPI+OpenMP** versions, the MPI distribution should be installed. The following commands will install and configure MPICH MPI distribution for the previously installed gcc6:

```
sudo port install mpich-gcc6
sudo port select --set mpi mpich-gcc6-fortran
```

Once the MPI distribution is installed, do:

```
make MPI=1
```

to build the MPI version and do:

```
make MPI=1 OpenMP=1
```

to build the MPI+OpenMP version.

III. Usage

XaNSoNS can be used from the console or in a python application with the help of the API for Python. The configuration of parameters is provided via XML interface (file with XML markup, see https://en.wikipedia.org/wiki/XML).

III.1 XML interface

The XML file must contain one element with the tag **<Calculation>** and at least one element with the tag **<Block>**. The **<Calculation>** element defines the parameters of the specific simulation, such as the simulation scenario, the type and the parameters of the source, the size and the step for the scattering intensity array, etc. The **<Block>** element defines the parameters of the single structural block of the model sample. XaNSoNS allows for heterogeneous samples containing multiple structural blocks, therefore the XML file may contain multiple **<Block>** elements (see the example in **Sec. IV.2**).

III.1.a Calculation parameters

```
<Calculation name='CalcName' source='xray' scenario='debye'
hist_bin='0.001' PolarFactor='no' PrintAtoms='no'
PartialIntensity='no' FFfilename='$(name)_FFtable.txt'
wavelength='1.54'>
   <q min='0.08' max='7.0' N='1024' Nfi='1024'/>
   <Euler min='0 0 0' max='0 0 0' N='1 1 1' convention='ZXZ'/>
   <PDF type='PDF' q='0' density='0' PrintPartial='no'/>
</Calculation>
```

Here, the **mandatory** elements and attributes are colored with **red**, **conditionally mandatory** ones are colored in **light blue**, **optional** ones are colored with **dark blue** and the default values of the attributes are colored in green. *Note, all tags and names of the attributes are case sensitive while the values of the attributes are not.*

Click on any attribute name or tag in the gray frame above to jump to its definition.

Attribute **name** defines the name of the specific simulation. It determines the prefix of the output file names. Each simulation in the series must have the unique name. See the full list of output files in **Appendix A**.

Attribute **source** defines the type of the radiation source. It can take values `xray` or `neutron`. The default value is `xray`.

Attribute **scenario** defines the simulation scenario, namely, what will be calculated and how. There are 5 different simulation scenarios: `Debye`, `Debye_hist`, `PDFonly`, `DebyePDF`, `2D`. These scenarios are described below.

1. **scenario**='Debye'

   The x-ray (Eq. (**1**)) or neutron (Eq. (**2**)) powder diffraction pattern will be calculated in this case using the exact Debye equation:

$$S_{xray}(q) = \frac{1}{N}\sum_{el_i=1}^{N_{el}}\sum_{el_j=1}^{N_{el}} f_{el_i}(q) f_{el_j}(q) \sum_{i\in el_i}\sum_{j\in el_j}\frac{\sin(qr_{ij})}{qr_{ij}}, \quad (1)$$

$$S_{neut}(q) = \frac{1}{N}\sum_{el_i=1}^{N_{el}}\sum_{el_j=1}^{N_{el}} b_{el_i} b_{el_j} \sum_{i\in el_i}\sum_{j\in el_j}\frac{\sin(qr_{ij})}{qr_{ij}}, \quad (2)$$

where el_i and el_j are the indexes of chemical elements (ions, isotopes, etc.), N_{el} is the total number of different chemical elements in the sample, N is the total number of atoms of all elements in the sample, q is the scattering vector magnitude, $f_{el_i}(q)$ is the x-ray atomic form factor for the chemical element (or ion) with index el_i, b_{el_i} is the neutron scattering length for the chemical element (or isotope) with index el_i, i and j are the indexes of the atoms of the elements el_i and el_j respectively, r_{ij} is the distance between i-th and j-th atoms.

Of course, only one half of the summand in the Debye sum is computed in real simulation, because $r_{ij} = r_{ji}$. While using the exact Debye sum is the easiest way to calculate the powder diffraction pattern, it is not the fastest one, which is `'Debye_hist'` scenario described below.

Although `'Debye_hist'` simulation scenario is preferable for general usage, some special features of XaNSoNS are available only in `'Debye'` simulation scenario.

See **Appendix A** for the specifications of the output file formats.

2. **`scenario`**`='Debye_hist'`

In this scenario the scattering intensity is calculated using the pre-calculated histogram of interatomic distances. The histogram approximation used here is exactly the same that is used in the Debyer code (https://github.com/wojdyr/debyer). This approximation significantly reduces the number of sine functions to compute which reduces the computational time. Once the interatomic distance r_{ij} is calculated, the respective histogram bin, $\left[H_{el_i,el_j}\right]_k$, for which the condition (3) is valid, is updated, $\left[H_{el_i,el_j}\right]_k = \left[H_{el_i,el_j}\right]_k + 1$.

$$k\Delta_{bin} < r_{ij} \leq (k+1)\Delta_{bin}, \qquad k = 0 : N_{bin} - 1 \tag{3}$$

The number of histogram bins, N_{bin}, is determined by the linear size of the sample and the user provided bin width, Δ_{bin} (default value is 0.001 Å). The scattering intensity is calculated then using the following equations:

$$S_{xray}(q) = \frac{1}{N}\sum_{el_i=1}^{N_{el}}\left(\left(f_{el_i}(q)\right)^2 N_{el_i}^{"at"} + 2\sum_{el_j=el_i}^{N_{el}} f_{el_i}(q) f_{el_j}(q) \sum_{k=0}^{N_{bin}-1}\left[H_{el_i,el_j}\right]_k \frac{\sin(qr_k)}{qr_k}\right), \tag{4}$$

$$S_{neut}(q) = \frac{1}{N}\sum_{el_i=1}^{N_{el}}\left(\left(b_{el_i}\right)^2 N_{el_i}^{"at"} + 2\sum_{el_j=el_i}^{N_{el}} b_{el_i} b_{el_j} \sum_{k=0}^{N_{bin}-1}\left[H_{el_i,el_j}\right]_k \frac{\sin(qr_k)}{qr_k}\right), \tag{5}$$

where $N_{el_i}^{"at"}$ is the total number of atoms of the element el_i in the model sample, and $r_k = (k + 0.5)\,\Delta_{bin}$.

3. **`scenario`**`='PDFonly'`

In this scenario, the pair distribution function (PDF) is calculated. PDF values coincide with the values of the histogram up to a certain function of r_k determined by the type of the PDF. For the samples containing more than one chemical element, the partial PDFs are calculated first. XaNSoNS supports the same types of PDFs that Debyer code does. This types are the following:

1. radial distribution function (RDF), $R_{el_i,el_j}(r_k)$,
2. pair distribution function (PDF), $P_{el_i,el_j}(r_k)$,
3. reduced pair distribution function (rPDF), $G_{el_i,el_j}(r_k)$.

Here are the equations for partial PDFs:

$$R_{el_i,el_j}(r_k) = \frac{1}{N\Delta_{bin}}\left[H_{el_i,el_j}\right]_k, \tag{6}$$

$$P_{el_i,el_j}(r_k) = \frac{1}{4\pi\rho_0 {r_k}^2 N\Delta_{bin}}\left[H_{el_i,el_j}\right]_k, \tag{7}$$

$$G_{el_i,el_j}(r_k) = \frac{1}{r_k N\Delta_{bin}}\left[H_{el_i,el_j}\right]_k - 2^{1-\delta_{el_i,el_j}} 4\pi\rho_0 r_k \frac{N_{el_i}^{"at"} N_{el_j}^{"at"}}{N^2}, \delta_{el_i,el_j} = \begin{cases} 1, el_i = el_j \\ 0, el_i \neq el_j \end{cases} \tag{8}$$

Here ρ_0 is the average atomic density provided by the user or calculated approximately.

The total PDFs are calculated using the following equation:

$$F(r_k) = \sum_{el_i=1}^{N_{el}} \sum_{el_j=el_i}^{N_{el}} \frac{f_{el_i} f_{el_j}}{f_{av}^2} F_{el_i,el_j}(r_k), \qquad f_{av} = \sum_{el_i=1}^{N_{el}} f_{el_i} \frac{N_{el_i}^{"at"}}{N}, \tag{9}$$

where $F(r_k)$ is the RDF, PDF or rPDF and f_{el_i} is either $f_{el_i}(q_p)$ for x-ray scattering (where q_p is the user defined value of scattering vector magnitude for which the total PDF is calculated), or b_{el_i} for neutron scattering.

Note, PBC are currently unsupported in XaNSoNS, so the PDF/RDF/rPDF tails are distorted and therefore should be excluded from the analysis.

See **Appendix A** for the specification of the output file formats.

4. **scenario**='DebyePDF'

   This scenario combines previous two scenarios. Both the PDF and the diffraction pattern are calculated here using the histogram of interatomic distances.

5. **scenario**='2D'

   The two-dimensional diffraction pattern is calculated in this scenario. Generally, this type of patterns is used in single-crystal analysis, but it is also very helpful in the analysis of advanced carbon nanostructures such as carbon nanotubes and fullerenes. In this case the spatial orientation of the sample is important. The source incident vector, k_0, is always directed opposite to the Z-axis direction and the orientation of the sample is defined by the user with the Euler angle representation in one of the 12 possible conventions (see Fig. **2**). It is also possible to average the 2D diffraction pattern over the Euler angles of the sample, which corresponds to the rotation of the sample during the exposure.

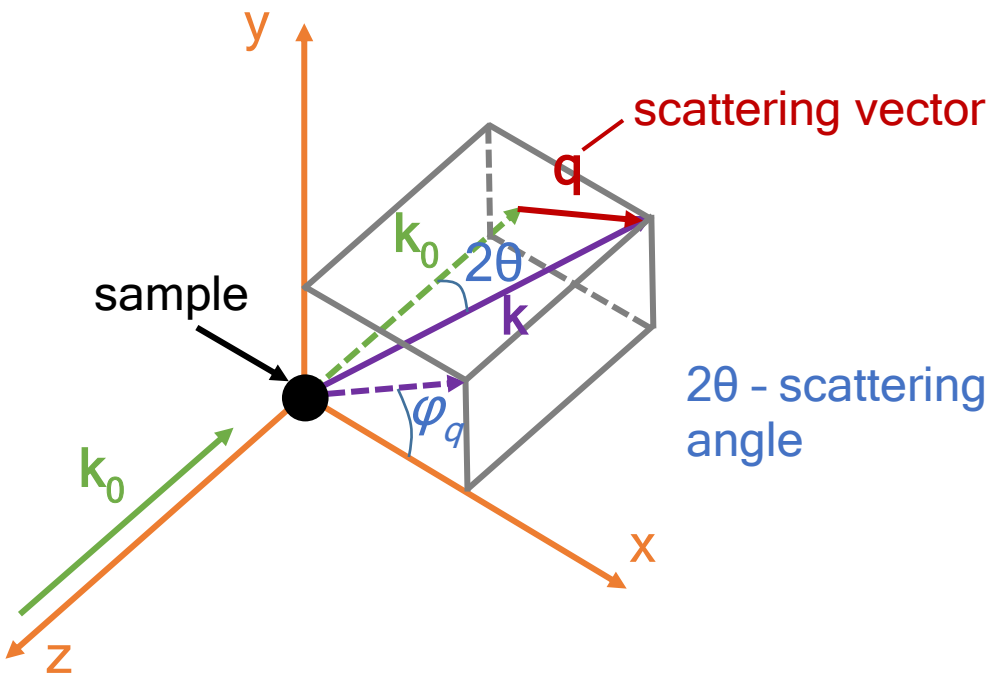

Fig. 2. Elastic scattering scheme.

The equations used in this case are more general than Eq. (**1**) and (**2**). The patterns are calculated in polar coordinates (q,ϕ_q).

$$S_{xray}(q,\phi_q) = \frac{1}{N}\left|\sum_{el_i=1}^{N_{el}} f_{el_i}(q)\sum_{i\in el_i} e^{i\mathbf{qr_i}}\right|^2, \tag{10}$$

$$S_{neut}(q,\phi_q) = \frac{1}{N}\left|\sum_{el_i=1}^{N_{el}} b_{el_i}\sum_{i\in el_i} e^{i\mathbf{qr_i}}\right|^2, \tag{11}$$

*Note, **qr**$_i$ is a dot product.* The one-dimensional diffraction pattern that averages the 2D diffraction pattern over ϕ_q is also calculated in this scenario.

See **Appendix A** for the specification of the output file formats.

Attribute **`hist_bin`** defines the width of histogram bins in Å. The spatial resolution of PDF is equal to this value. The default value is `'0.001'` Å. This attribute will be processed only if **`scenario`**`='Debye_hist'`, **`scenario`**`='PDFonly'`, or **`scenario`**`='DebyePDF'`.

Attribute **`PolarFactor`** defines whether the resulting scattering intensity is multiplied by the polarization factor or not in the case of x-ray scattering. Only unpolarised x-ray source is supported in XaNSoNS, so the polarization factor is equal to $(1+\cos(2\theta)^2)/2$. The default value is `'no'`. This attribute will be processed only if **`source`**`='xray'`. The **`wavelength`** attribute should be defined if **`PolarFactor`** attribute is set to `'yes'`.

Attribute **`PrintAtoms`** defines whether the final atomic ensemble of the model sample will be printed to the file or not. Printing the atomic ensemble is very useful for testing. The resulting file has .xyz format, so it can be visualized in various software, e.g. in Pymol. The default value is `'no'`.

Attribute **PartialIntensity** allows one to examine the interference effects caused by the presence of multiple structural blocks in the sample. If this attribute is set to `'yes'` the partial contribution to the intensity from the scattering on the atoms of all possible pairs of structural blocks (see Eq. **13** below) will be printed to the files.

One can separate the atoms in Eq. (**1**) by their belonging to a particular structural block.

$$S_{xray}(q) = \sum_{B_i=1}^{N_{block}} \sum_{B_j=1}^{N_{block}} [S_{xray}(q)]_{B_i,B_j}, \tag{12}$$

$$[S_{xray}(q)]_{B_i,B_j} = \frac{1}{N} \sum_{el_i=1}^{N_{el}} \sum_{el_j=1}^{N_{el}} f_{el_i}(q) f_{el_j}(q) \sum_{i \in \{el_i,B_i\}} \sum_{j \in \{el_j,B_j\}} \frac{\sin(qr_{ij})}{qr_{ij}}. \tag{13}$$

Here N_{block} is the total number of structural blocks, B_i is the index of the structural block. The expression $i \in \{el_i, B_i\}$ means that the i-th atom belongs to the chemical element with index el_i and to the structural block with index B_i. $[S_{xray}(q)]_{B_i,B_j}$ are the partial contributions to the intensities.
The default value of this attribute is `'no'`. This attribute will be processed only if **scenario**=`'Debye'`.

Attribute **FFfilename** defines the path to the file containing the x-ray atomic form factors (or neutron scattering lengths). XaNSoNS allows the user-defined form factors. This attribute should be provided only if the specific file with the atomic form factors should be used or if this file should have the specific name. The specified file will be created if it does not exist (the path to the folder, however, should exist, otherwise, the error will occur). If the file exist it will be checked for the consistency first, and will be complemented with the missing data if necessary. So, if the user needs to provide the non-standard atomic form factors for some chemical elements or ions, the respective file should contain only the form factors for these elements. The standard form factors for all other elements will be added to this file automatically. XaNSoNS uses the Periodictable python module for the data of x-ray atomic form factors and neutron scattering lengths. **Important:** the Periodictable module **does not contain** the x-ray from-factors for certain ions. These form-factors must be provided by the user.
The default name of the file with the atomic form factors is `'$(name)_FFtable.txt'`, where `$(name)` is the value of the **name** attribute. See **Appendix B** for the description of the file format for atomic form-factors. Also, please, note that **you should manually remove the file with the atomic form-factors** created on the previous run, if you change one of the following parameters: computational scenario, type of the source (x-ray, neutron), range of the scattering vector magnitude.

Attribute **wavelength** defines the wavelength of the source in Å (either the wavelength of x-ray radiation or that of thermal neutrons). Since the scattering intensity is calculated as a function of q, this attribute is mandatory only if **PolarFactor** attribute is set to `'yes'` (because the polarization factor is a function of scattering angle). Also, this parameter will be processed in the case of **scenario**=`'2D'`. If the **wavelength** is not set it will be set to $4\pi/q_{max}$.

Element <**q**> defines the numerical mesh for the scattering vector. This element is mandatory for all scenarios except **scenario**=`'PDFonly'`.

> Attributes **min** and **max** define the minimum and maximum values of scattering vector magnitude in $Å^{-1}$. They are mandatory and do not have the default values.

Attribute **N** defines the number of points in the mesh of scattering vector magnitude. The default value is '1024'.

Attribute **Nfi** defines the number of points in the mesh of the polar angle of the scattering vector in the case of **scenario**='2D'. This attribute will not be processed in other cases. The default value is '1024'.

Element <**Euler**> defines the spatial orientation of the sample in the case of **scenario**='2D'. This element will not be processed in other cases. It is possible to average the 2D diffraction pattern over the Euler angles of the sample: this corresponds to the rotation of the sample during the exposure. Source incident vector, k_0, is always directed opposite to the Z-axis direction (see Fig. **2**).

Attributes **min** and **max** define the minimum and maximum values of Euler angles (α, β, γ) in degrees. Values should be separated with spaces. If the sample does not rotate during the exposure, the values of **min** and **max** attributes should be equal. The default value is '0 0 0' for both attributes.

Attribute **N** defines the resolution of 3D mesh, namely, the number of points for α, β and γ between the minimum and the maximum values. If the sample does not rotate during the exposure the value should be equal to the default: '1 1 1'.

Attribute **convention** defines the convention, namely, the sequence of the axes about which the rotations are carried out. These rotations are intrinsic (the rotations are performed about the axes of the rotating coordinate system). See https://en.wikipedia.org/wiki/Euler_angles#Relationship_to_other_representations for the details. The following values are supported: 'XZX', 'XYX', 'YXY', 'YZY', 'ZYZ', 'ZXZ', 'XZY', 'XYZ', 'YXZ', 'YZX', 'ZYX', 'ZXY'. The default value is 'ZXZ'.

Element <**PDF**> defines the parameters of the pair-distribution function. This element will be processed only in the cases of **scenario**='PDFonly' and **scenario**='DebyePDF'.

Attribute **type** defines the type of the PDF function to calculate. The possible values are: 'PDF', 'RDF' and 'rPDF'. The default value is 'PDF'.

Attribute **q** defines the value of the scattering vector magnitude, q_p, for which in the case of x-ray scattering the total PDF is calculated (see Eq. (**9**) and the subparagraph after). This attribute will be processed only in the case of **source**='xray'. The default value is '0'.

Attribute **density** defines the average atomic density of the sample, ρ_0 (see Eq. (**7**) and (**8**)). The default value is '0', which means that the density will be calculated approximately for the given atomic layout of the sample.

Attribute **PrintPartial** defines whether the partial PDFs (Eq. (**6**), (**7**) and (**8**)) will be printed or not. The default value is 'yes'.

III.1.b Parameters of the structural block

```
<Block fractional='no' centeredAtoms='no' centered='no'
mol_rotation='no' mol_Uiso='0'>
     <Atoms filename='AtomsFile.cif' disp='yes' MaxAtoms2XML='50'>
          <Atom name='C' Z='6' r='0 0 0' occ='1.0' Uiso='0'/>
          <Atom name='C' Z='6' r='1 1 1' occ='1.0' Uiso='0'/>
     </Atoms>
```

```
      <SymmEqPos r='0 0 0' R1='1 0 0' R2='0 1 0' R3='0 0 1'/>
      <SymmEqPos r='2 2 0' R1='1 0 0' R2='0 1 0' R3='0 0 1'/>
      <CellVectors a='1 0 0' b='0 1 0' c='0 0 1' N='1 1 1'/>
      <Copies filename='CopiesFile.txt' convention='ZYX'>
            <Copy r='0 0 0' Euler='0 0 0'/>
            <Copy r='0 10 0' Euler='0 0 30'/>
      </Copies>
      <CutOff Rcut='0' RcutCopies='0'/>
</Block>
```

Here the same color scheme as for the **<Calculation>** element in **Sec. III.1.a** is used. The pairs of mutually exclusive parameters are the only addition, which are shown with underlined text of different colors. E.g., the atomic parameters can be defined in the separate file or in the XML file itself, so the attribute **filename** of <**Atoms**> element is mutually exclusive with the elements with the tag <**Atom**>.

The single structural block is defined in XaNSoNS as follows. The smallest structural unit is the atomic cluster defined in <**Atoms**> element. This atomic cluster may describe the atoms of a symmetry element in the unit cell, or the atoms of the entire unit cell, or the atoms of the single molecule, or even the atoms of the entire structural block. If the atomic cluster describes the atoms of the symmetry element, the symmetry equivalent positions should be provided via the elements with the tag <**SymmEqPos**> to define the unit cell. In the case of the crystalline structure the lattice vectors, the lengths of the cell edges and the number of the translations along the lattice vectors should be provided via the element with the tag <**CellVectors**>. The model sample may contain multiple copies of the structural block. The positions and orientations in 3D space of these copies are defined in the element with the tag <**Copies**>.

Below is the detailed description of <**Block**> element.

Attribute **fractional** defines whether the atomic coordinates are defined in the fractional coordinate system or not. In the case of `'yes'` the atomic coordinates in the Cartesian system will be calculated as follows:

$$\mathbf{r}_{Cart} = x_{frac}\mathbf{a} + y_{frac}\mathbf{b} + z_{frac}\mathbf{c} \tag{13}$$

where **a**, **b** and **c** are the lattice vectors with the magnitude of the lattice vector is equal to the length of the respective cell edge. The default value is `'no'`.

Attribute **centeredAtoms** specifies whether the point of geometric center (sum over the atomic coordinates) should be subtracted from the atomic coordinates defined in <**Atoms**> element before calculating the symmetry equivalent positions or not. In the case of `'yes'` the geometric center will be subtracted. The default value is `'no'`. This attribute is handy for describing the molecular crystals.

Attribute **centered** specifies whether the atomic coordinates of entire structural block should be corrected by subtracting the point of geometric center of the structural block before calculating the position and the orientation of a copy of the structural block in 3D space or not. The default value is `'no'`. If the atomic coordinates defined in <**Atoms**> element describes the entire structural block, **centered** and **centeredAtoms** attributes specify the same thing.

Attribute **mol_rotation** and **mol_Uiso** are introduced for the special case of spherical or near-spherical molecules. If these attributes are specified, it is implied that the atoms defined in <**Atoms**> element describe the spherical or the near-spherical molecule. In the case of

mol_rotation='yes' the spatial orientation of the molecule will be random. Attribute **mol_Uiso** defines the isotropic displacement of the molecule in terms of normal distribution. The value (in Å^2) of **mol_Uiso** is interpreted as σ^2 of this distribution. The position of each molecule in each copy of the structural block will be shifted to a random direction on the randomized, normally distributed distance. The default values are **mol_rotation**='no' and **mol_Uiso**='0'.

Element <**Atoms**> defines the parameters of the atomic cluster, which describes the smallest structural element of the structural block.

Attribute **filename** defines the path to the file containing the data of atomic cluster. XaNSoNS has a native file format similar to .xyz but extended with the site occupancies and atomic displacement parameters (this extension is optional). XaNSoNS also can import the data from the CIF (crystallography information file) file format.
Important: the CIF files which do not contain the '_symmetry_equiv_pos_as_xyz' or '_space_group_symop_operation_xyz' key **are unsupported**.
When importing from the CIF file the data of atomic cluster can be stored either in the external native file or in the XML file itself. The unit cell data (e.g., lattice vectors) will be stored in the XML file itself. See **Appendix B** for the description of the input files formats.

Attribute **disp** defines whether the atomic displacement (uncertainty) parameters ('_atom_site_U_iso_or_equiv', '_atom_site_U_equiv_geom_mean', '_atom_site_B_iso_or_equiv' or '_atom_site_B_equiv_geom_mean' keys) will be imported from the CIF file or not. In the case of 'yes' these parameters will be imported, converted (if required) to **Uiso** parameters and stored in the XML file itself or in the external file. The default value is 'yes'.

Attribute **MaxAtoms2XML** defines the maximum number of atoms that can be stored in the XML file itself when importing the data from the CIF file. If the number of atoms in the CIF file is greater than the value of this parameter, the atoms will be stored in the external file, otherwise they will be stored in the XML file itself. The default value is '50'.

The user may specify the parameters of atomic cluster directly in the XML file. In the XML each atom is described in the respective <**Atom**> element. These elements will be processed only if **filename** attribute is not specified or empty. Either **filename** attribute or at least one <**Atom**> element should be specified and valid.

Attribute **name** specifies the symbol of chemical element or ion. Technically, this attribute may take any value until the form factor for the specified name is provided either by the user or by the "Periodictable" python module, http://www.reflectometry.org/danse/docs/elements/index.html.

Alternative way to specify the chemical element is by providing its number in the periodic table. This can be done by setting the value of the attribute **Z**. Note, the ions cannot be specified this way. Either **name** or **Z** attribute should be specified in each <**Atom**> element.

Attribute **r** defines the atom's position in 3D space. This attribute is mandatory. The position should be specified either in the fractional coordinate system (if **fractional**='yes') or in Cartesian coordinate system in Å. XaNSoNS supports

the values outside the interval [0, 1] in the case of fractional coordinates. The Cartesian coordinates will be calculated according to Eq. (**13**).

Attribute **occ** defines the site occupancy. The value should be in the range of [0, 1]. Site occupancy is the probability of the specified atom to be located in the position with given **r**. The defects in the lattice structure such as replacements and vacancies can be modeled using this parameter. XaNSoNS calculates the atomic ensemble with respect to the values of **occ**. E.g. if two different atoms share the same site with **occ**='0.5' for both, the final ensemble will have the 50% of the atoms of the first type and 50% of the atoms of the second type in this site, while the atoms will be distributed randomly from cell to cell. Note, to allow for the possible displacement in the case when one atom is replaced by another, any pair of atomic positions with the interatomic distance lower than 0.5 Å will be considered as a single atomic position shared between these two atoms. The default value is '1.0'.

Attribute **Uiso** defines the isotropic displacement of the atom in the terms of normal distribution. The value (in $Å^2$) of **Uiso** is interpreted as σ^2 of this distribution. The position of each atom in each cell in each copy of the structural block will be shifted to a random direction on the randomized but normally distributed distance. The default value is '0'.

Elements <**SymmEqPos**> define the symmetry equivalent positions of the symmetry element in the unit cell. The attribute **r** defines the center of the symmetry equivalent position while the attributes **R1**, **R2** and **R3** define the rows of the rotational matrix. The symmetry equivalent position of the atom is calculated by the following equation:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{symm} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}_0 + \begin{pmatrix} R1_x & R1_y & R1_z \\ R2_x & R2_y & R2_z \\ R3_x & R3_y & R3_z \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \tag{14}$$

where $\begin{pmatrix} x \\ y \\ z \end{pmatrix}_0$ is specified by the attribute **r** of <**SymmEqPos**> element and $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ is the initial position. The initial symmetry equivalent position (**r**='0 0 0', **R1**='1 0 0', **R2**='0 1 0', **R3**='0 0 1') exist in any symmetry group, so it is not necessary to specify it by <**SymmEqPos**> element. Some symmetry equivalent positions may describe the identical transforms, XaNSoNS will handle this case and exclude excess positions when calculating of the atomic ensemble. Note, the general representation (**14**) of the symmetry equivalent positions allows for the positions that are out of any symmetry group. The default values are **r**='0 0 0', **R1**='1 0 0', **R2**='0 1 0', **R3**='0 0 1'.

Element <**CellVectors**> defines the lattice vectors, the lengths of the cell edges and the number of translation along each direction (the size of the nanocrystal).

Attributes **a**, **b** and **c** define the respective lattice vectors. The magnitude of each vector should be equal to the length of the respective cell edge. The default values are **a**='1 0 0', **b**='0 1 0', **c**='0 0 1'.

Attribute **N** define the size of the nanocrystal as the number of translation along each lattice vector. The value should be the 3D integer vector. The first, second and third values are the number of translation along the vector **a**, **b** and **c** respectively. The default value is **N**='1 1 1'.

XaNSoNS allows the structural block to have multiple copies. Element <**Copies**> defines the positions and spatial orientations of the copies of the structural block in 3D space. The positions and spatial orientation of the different structural blocks may be correlated by specifiing this elements for each structural block. Element <**Copies**> may be very handy when describing the amorphous or polycrystalline samples.

Attribute **filename** defines the path to the file containing the positions and spatial orientations of the copies of the structural block. See **Appendix B** for the specifications of the input file formats.

The spatial orientation of the copy of the structural blocks is described with the Euler angles, therefore, the convention should be specified. Attribute **convention** defines the convention, namely, the sequence of the axes about which the rotations are carried out. These rotations are intrinsic (the rotations are performed about the axes of the rotating coordinate system). See https://en.wikipedia.org/wiki/Euler_angles#Relationship_to_other_representations for the details. The following values are supported: `'XZX'`, `'XYX'`, `'YXY'`, `'YZY'`, `'ZYZ'`, `'ZXZ'`, `'XZY'`, `'XYZ'`, `'YXZ'`, `'YZX'`, `'ZYX'`, `'ZXY'`. The default value is `'ZYX'`.

As it is for the atoms, the user may specify the parameters of the copies in the XML file itself. In this case, each copy is described with the respective <**Copy**> element. These elements will be processed only if **filename** attribute is not specified or empty. If the parent elements <**Copies**> is present in the XML file, either **filename** attribute or at least one <**Copy**> element should be specified and valid.

Attribute **r** specifies the position in Å of a copy of the structural block in 3D space. Depending on the value of **centered** attribute, vector **r** may point to the geometrical center of the structural block (`'yes'`) or to the point (0,0,0) of the internal coordinate system of the structural block (`'no'`). This attribute is mandatory.

Attribute **Euler** specifies the Euler angles (α, β, γ) in degrees, which describe the orientation of the copy in 3D space. This attribute is mandatory.

Element <**CutOff**> enable the user to cut the cluster of a spherical shape either from the structural block or from the entire ensemble of the copies of the structural blocks. This sphere should not be mixed up with the cut-off sphere used in simulations with PBC, which are currently unsupported in XaNSoNS. Note, to cut the perfect sphere from the structural block (or from the model sample) the linear sizes of this block along any direction should exceed the diameter of the specified sphere.

Attribute **Rcut** define the radius (in Å) of the sphere, which will be cut from the structural block. The default value is `'0'`, which means that the sphere will not be cut-off.

Attribute **RcutCopies** define the radius (in Å) of the sphere, which will be cut from the entire ensemble of copies of the structural blocks. If the sphere should be cut-off from the entire model sample, the value of **RcutCopies** should be the same for all structural blocks. The default value is `'0'`, which means that the sphere will not be cut-off.

III.2 Running from the console

To run XaNSoNS from the console do:

```
python path_to/XaNSoNS.py options
```

where *path_to* is a path to XaNSoNS folder and *options* is the list of options described below.

Do:
```
python path_to/XaNSoNS.py -h
```
or:
```
python path_to/XaNSoNS.py --help
```
to show the list of options. The options are given below.

`-i` or `--XML` specifies the path to XML file containing the configuration of parameters described in **Sec. III.1**.

`-v` or `--version` specifies the version of XaNSoNS executable to run. The descriptions of XaNSoNS executables is presented in Table **1** in **Sec II.2**. Default value is `OMP`. All possible values of this option are given in Table **2**.

Table 2. Possible values of option `-v` and versions of XaNSoNS executable.

Value	Executable version
`single`	XaNSoNS
`OMP`	XaNSoNS_OMP
`MPI`	XaNSoNS_MPI
`MPI_OMP`	XaNSoNS_MPI_OMP
`CUDA`	XaNSoNS_CUDA
`OpenCL`	XaNSoNS_OCL

`-N` or `--Nproc` specifies the number of MPI processes if `MPI` or `MPI_OMP` version is selected. The default value is `4`.

`-n` or `--Nthread` specifies the number of OpenMP threads if `OMP` or `MPI_OMP` version is selected (for `MPI_OMP` it specifies the number of OpenMP threads per MPI process). The default value is `0`, which means system defaults.

`-d` or `--deviceID` specifies the number, starting with 0, of CUDA or OpenCL device to use if CUDA or OpenCL version is selected. For the OpenCL version, it will specify the number of GPU device for the selected OpenCL platform. The default value is `-1`, which means that the fastest device (of the selected platform in the case of OpenCL version) will be used.

`-p` or `--platformID` specifies the number, starting with 0, of OpenCL platform to use if OpenCL version is selected. The default value is `-1`, which means that the optimal OpenCL platform will be used.

Notes for the OpenCL version:

1. *If both `-d` and `-p` are set to -1, the fastest OpenCL device among all devices in all platforms will be used.*
2. *If option `-d` is set to -1 and option `-p` is not, the fastest device in the specified OpenCL platform will be used.*
3. *If option `-p` is set to -1 and option `-d` is not, the program will use the fastest device with the specified number among all OpenCL platforms.*
4. *The program will show all OpenCL platforms and devices available in the system if `-d` and `-p` options are set to -1.*

`--nobackup` option prevents the backup of XML files in the case they are changed by the application (e.g. in the case of importing the atomic data from the CIF file).

`--noFFcheck` option prevent XaNSoNS from checking the file with atomic form factors (or neutron scattering lengths) for existence and validity (completeness). If the user is sure that the file is exist and valid, this option may be set to speed up the application launch.

`--noCIFcheck` option prevent XaNSoNS from checking the XML file for the possible links to the CIF files. If there is no need in importing the atomic data from the CIF files, the user may set this option to slightly speed up the application launch.

Suppose that the XaNSoNS is located in the folder /home/user/xansons/ and the current working directory contains all necessary files to run the simulation including XML file with the name "calc0.xml". Suppose that there are two OpenCL platforms that are present in the system: Intel (with ID = 1) and AMD (with ID = 2), and the user want to run the simulation on the fastest AMD device. The user also knows that importing the data from the CIF files is not required. In that case the user should start the simulation with the following command:

```
python /home/user/xansons/XaNSoNS.py -i calc0.xml -v OpenCL -p 2
--noCIFcheck
```

III.3 API for Python

XaNSoNS can be integrated into complex computational scenarios using API for Python. With the help of this API the configuration of parameters can be easily specified in a python script.

If the application is located at /home/user/xansons/, one should include the following three lines to import the XaNSoNS module in a python script.

```
import sys
sys.path.append('/home/user/xansons/')
import XaNSoNS
```

To work with the configuration of parameters one should also import the numpy module.

```
import numpy as np
```

The configuration of parameters can be either imported from the existing XML file:

```
config=XaNSoNS.config('config.xml')
```

or created:

```
config=XaNSoNS.config()
```

The new structural block can be created with a following line:

```
config.AddBlock()
```

The same way a new atom can be added to the atomic cluster of the first structural block:

```
config.Block[0].Atoms.AddAtom()
```

or a new symmetry equivalent position to the configuration of the unit cell:

```
config.Block[0].AddSymmEqPos()
```

or a new copy of the first structural block to the configuration of the model sample:

```
config.Block[0].Copies.AddCopy()
```

Any parameter can be accessed with respect to the structure of XML file. E.g. one can set a name of the current simulation by the line:

```
config.Calculation.name='calc1'
```

or specify the position of the 3rd atom in the 2nd structural block by the line:

```
config.Block[1].Atoms.Atom[2].r=np.array([0., 0.5, 0])
```

The current configuration of parameters can be displayed by the line:

```
config.Print()
```

or stored to the XML file by the line:

```
config.ToXML('new_config.xml')
```

The user can choose whether to print parameters to the XML file, if their values coincide with the defaults, or not. By default, these parameters will not be printed. To print them call:

```
config.ToXML('new_config.xml',PrintDefaults=True)
```

The following line will run the simulation for the example from the end of the **Sec. III.2** in the python script.

```
calc=XaNSoNS.run('calc0.xml', version='OpenCL',platID=1,
CIFcheck=False)
```

`XaNSoNS.run()` returns a `Popen` object of `subprocess` module (https://docs.python.org/2/library/subprocess.html), therefore it returns immediately. The user can interact with `calc` as with any other `Popen` object. To wait until the end of the simulation call:

```
calc.wait()
```

Function `XaNSoNS.run()` has following arguments and defaults:

```
XaNSoNS.run(XML,version='OMP',Nproc=4,Nthread=0,devID=0,platID=0
,backup=True,CIFcheck=True,FFcheck=True)
```

The capabilities of the XaNSoNS API for Python are shown in the examples in **Sec. IV.4** and **IV.5**.

IV. Examples

Unpack “examples.7z” archive provided with the XaNSoNS distribution.

Note: all the commands below are path relative. The current working directory must be the directory with the current example.

IV.1 NaCl

This is the simplest example. Navigate to “NaCl” folder in “examples” folder. “NaCl.xml” file has the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<PMML>
     <Calculation PrintAtoms="Yes" name="NaCl_10x10x10">
          <q max="7.0" min="0.5"/>
     </Calculation>
     <Block>
          <Atoms filename="NaCl.cif"/>
          <CellVectors N="10 10 10"/>
     </Block>
</PMML>
```

This XML file contains the parameters of the simulation of the 1D x-ray diffraction pattern for the single NaCl nanocrystal with the size of 10x10x10 unit cells. Here the general Debye Eq. (**1**) is used without the histogram approximation (**4**). The unit cell parameters will be imported from the “NaCl.cif” file. The numerical mesh of the scattering vector magnitude q starts with the value 0.5 $Å^{-1}$ ends with the value 7.0 $Å^{-1}$ and consist of 1024 points (default). Run the parallel simulation on the CPU with the following command:

```
python ../../XaNSoNS.py -i NaCl.xml
```

Depending on the model of the CPU the simulation takes the time ~ 50 – 300 s. Four new files (including the updated XML) should be created if the simulation finished successfully. “NaCl_10x10x10_FFtable.txt” file contains the atomic x-ray form factors for the Na+ and Cl- ions. “NaCl_10x10x10_atoms.xyz” file contains the atomic ensemble of 8000 atoms. This ensemble may be visualized in various software including the Pymol. “NaCl_10x10x10_xray_1D.txt” file contains the simulated diffraction pattern (see Fig. **3**). The user may compare the calculated diffraction pattern with the reference one located in “NaCl_10x10x10_xray_1D_ref.txt” file.

The original XML file is backed up under the name “NaCl.xml.back”. The new “NaCl.xml” file has the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<PMML>
     <Calculation FFfilename="NaCl_10x10x10_FFtable.txt"
PrintAtoms="yes" name="NaCl_10x10x10">
          <q max="7" min="0.5"/>
     </Calculation>
     <Block fractional="yes">
          <Atoms>
               <Atom name="Na1+" r="0 0 0"/>
               <Atom name="Cl1-" r="0.5 0.5 0.5"/>
          </Atoms>
          <SymmEqPos R1="0 1 0" R2="0 0 1" R3="1 0 0" r="0 0 0"/>
          <SymmEqPos R1="0 0 1" R2="1 0 0" R3="0 1 0" r="0 0 0"/>
          <SymmEqPos R1="1 0 0" R2="0 0 1" R3="0 1 0" r="0 0 0"/>
          <SymmEqPos R1="0 1 0" R2="1 0 0" R3="0 0 1" r="0 0 0"/>
          …
```

```
          <SymmEqPos R1="0 0 1" R2="0 1 0" R3="-1 0 0" r="0.5 0.5 0"/>
          <CellVectors N="10 10 10" a="5.62 0 0" b="3.44126e-16 5.62
0" c="3.44126e-16 6.32155e-16 5.62"/>
     </Block>
</PMML>
```

Here the unit cell parameters are imported from "NaCl.cif" file. All the 192 symmetry equivalent positions are imported despite the fact that for the case of NaCl most of them describe the identical transforms. XaNSoNS eliminates excess symmetry equivalent positions while calculating the atomic ensemble. Once the file with the x-ray atomic form factors is created and the unit cell parameters are imported from the CIF file, the flags `--noFFcheck` and `--noCIFcheck` may be specified for the future launches.

Now change the value of the **`name`** attribute to "`NaCl_10x10x10_GPU`", save the XML file and run the new simulation on the OpenCL-compatible GPU with the following command:

```
python ../../XaNSoNS.py -i NaCl.xml -v OpenCL --noFFcheck --noCIFcheck
```

or on the CUDA-compatible Nvidia GPU with the command:

```
python ../../XaNSoNS.py -i NaCl.xml -v CUDA --noFFcheck --noCIFcheck
```

The simulation takes the time of about 0.5 – 4 s on the GPU depending on the GPU model. Direct comparison of "NaCl_10x10x10_xray_1D.txt" and "NaCl_10x10x10_GPU_xray_1D.txt" files shows that the calculated values of the diffraction patterns slightly differ. This is the result of using the single precision floating point math and the intrinsic sine function. However, the visual comparison of the calculated diffraction patterns (see Fig. **3**) shows good accuracy.

Now let's see how the use of the histogram approximation increases the computational speed. Add the attribute **`scenario`**`='Debye_hist'` to the `<Calculation>` element and change the value of the **`name`** attribute to "`NaCl_10x10x10_hist`":

```
     <Calculation FFfilename="NaCl_10x10x10_FFtable.txt"
PrintAtoms="yes" name="NaCl_10x10x10_hist" scenario="Debye_hist">
          <q max="7" min="0.5"/>
     </Calculation>
```

Save the XML file and run the new simulation on the CPU. Now the computational time is less than a second! Of course, such a high performance is achieved at the expense of reduced precision. As in the case of the simulation on the GPU, the accuracy is acceptable. Fig. **3** shows the comparison of all three simulated diffraction patterns. No difference is visible between the plots.

The use of the GPU in the examined case of **`scenario`**`='Debye_hist'` is useless for such a small number of atoms, but starting from 10^5 atoms, the GPU speeds up the simulation from 3 to 50 times depending on the CPU and the GPU models.

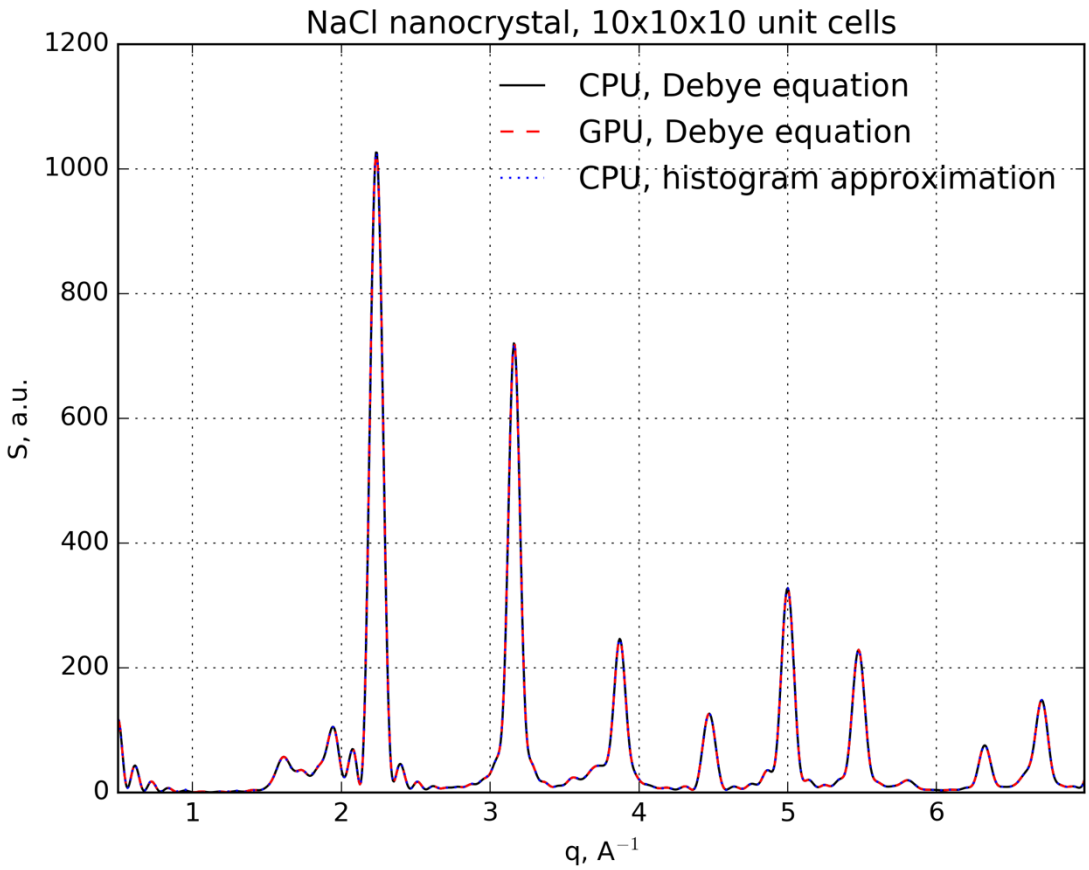

Fig. 3. Comparison of the simulated diffraction patterns for the single NaCl nanocrystal (10x10x10x unit cells, 8000 atoms).

IV.2 TWCNT

Navigate to "TWCNT" folder in the "examples" folder. "TWCNT.xml" file has the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<PMML>
     <Calculation PrintAtoms="Yes" scenario='2D' wavelength="0.5"
name="TWCNT-bundle">
          <q max="7.0" min="0.5"/>
          <Euler min="90 0 0" max="90 0 0" N="1 1 1"
convention="XZY"/>
     </Calculation>
     <Block centered="yes">
          <Atoms filename="CNT_(8,1)_140.00A.txt"/>
          <Copies filename="bundle.txt"/>
     </Block>
     <Block centered="yes">
          <Atoms filename="CNT_(11,9)_140.00A.txt"/>
          <Copies filename="bundle.txt"/>
     </Block>
     <Block centered="yes">
          <Atoms filename="CNT_(26,4)_140.00A.txt"/>
          <Copies filename="bundle.txt"/>
     </Block>
</PMML>
```

This XML file contains the parameters for the simulation of the 2D x-ray diffraction pattern (**scenario**='2D') for the bundle of identical triple-walled carbon nanotubes (TWCNT). This

TWCNT has the Russian Doll model (it consists of three separate single-walled carbon nanotube (SWCNT) of different radius and chirality). Each layer of the TWCNT is described by a separate structural block in the XML. The length of the TWCNT is 14 nm. The files "CNT_(8,1)_140.00A.txt", "CNT_(11,9)_140.00A.txt" and "CNT_(26,4)_140.00A.txt" contain the positions of the atoms in the SWCNTs. The file "bundle.txt" contain the positions of the TWCNTs in the bundle:

```
0 0 0 0 0 0
26. 0 0 0 0 0
13. 22.517 0 0 0 0
-13. 22.517 0 0 0 0
-26. 0 0 0 0 0
-13. -22.517 0 0 0 0
13. -22.517 0 0 0 0
```

Note, the parameters of this particular TWCNT bundle were not studied for the physical realness. This example just shows the application features.

Fig. **4** shows the TWCNT bundle visualized in Pymol.

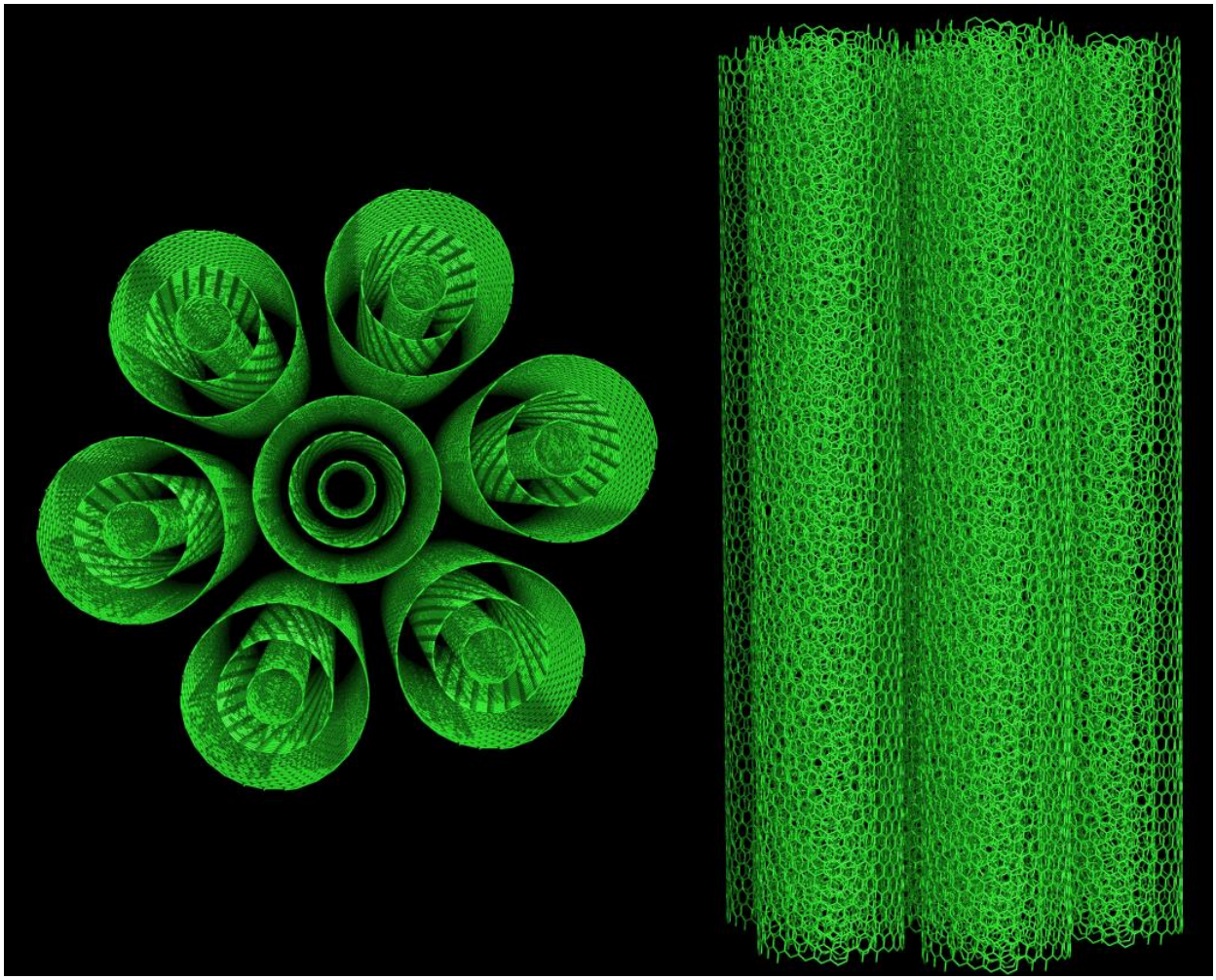

Fig. 4. Triple-walled carbon nanotube (TWCNT) bundle. Left: top view, right: side view.

The bundle is oriented along the Z-axis. To obtain the informative 2D diffraction pattern the CNT should be oriented perpendicular to the incident wave. Since the incident wave is directed opposite to the Z-axis the bundle needs to be rotated. The line:

```
<Euler min="90 0 0" max="90 0 0" N="1 1 1" convention="XZY"/>
```

in the XML file does the rotation. The TWCNT bundle contains about 50 000 atoms. The simulation of the 2D diffraction pattern for such a large atomic ensemble on the CPU may take dozens of minutes, so it is better to use the GPU. To run the simulation on the OpenCL-compatible GPU, do:

```
python ../../XaNSoNS.py -i TWCNT.xml -v OpenCL
```

or on the CUDA-compatible Nvidia GPU, do:

```
python ../../XaNSoNS.py -i TWCNT.xml -v CUDA
```

The simulation takes just a few seconds on the GPU. For the case of `scenario='2D'`, the use of the GPU gives the speed up of about few hundreds of times. The simulated 2D diffraction pattern is shown in Fig. **5**. The pattern is plotted in polar coordinates. The color map is adjusted to show the values lower than 2% of the maximum scattering intensity.

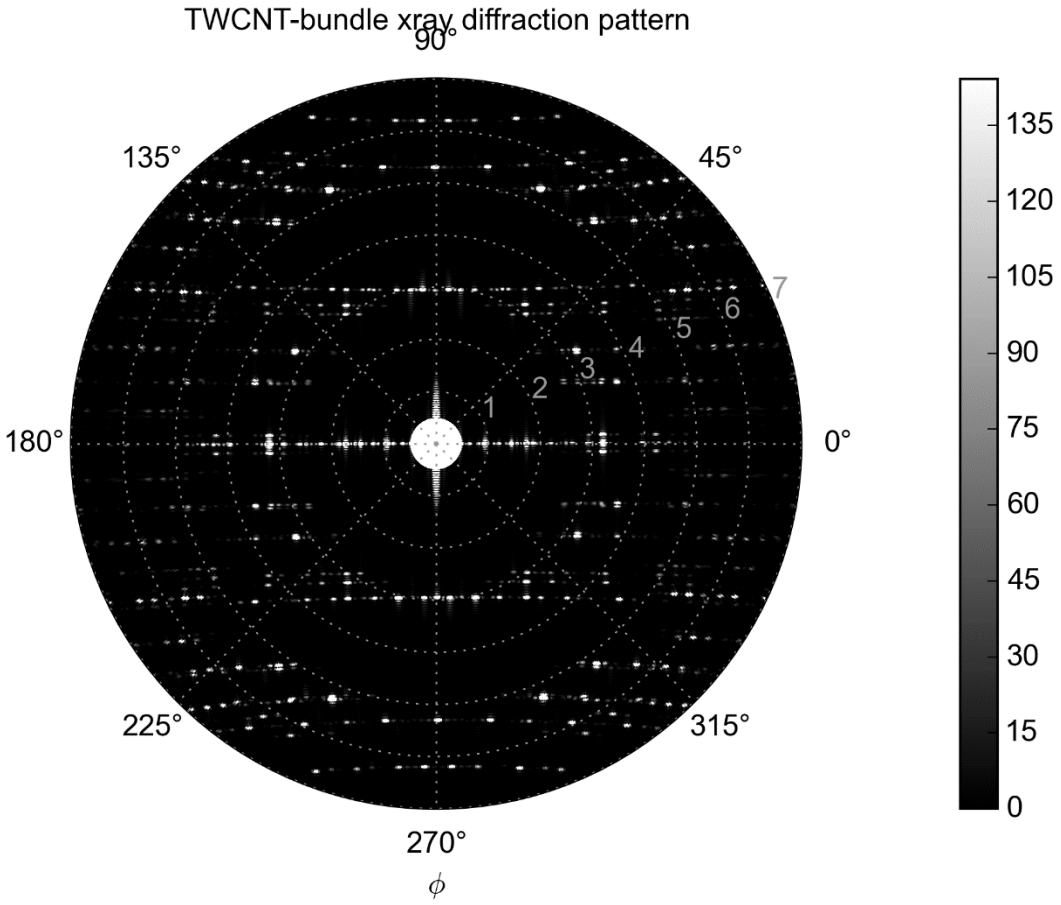

Fig. 5. Simulated 2D diffraction pattern for the bundle of TWCNTs show in Fig. **4**.

IV.3 Cheralite

This example shows how XaNSoNS handle nanocrystals of complex minerals. Navigate to "Cheralite" folder in "examples" folder. "Cheralite.xml" file has the following content:

```
<?xml version='1.0' encoding='UTF-8'?>
<PMML>
<Calculation name='Cheralite' PrintAtoms="Yes" scenario="Debye_hist">
     <q min='0.5' max='7.0'/>
</Calculation>
```

```
<Block>
     <Atoms filename='Cheralite.cif'/>
     <CellVectors N='20 20 20'/>
</Block>
</PMML>
```

The file “Cheralite.cif” was obtained from the Crystallography Open Database (http://www.crystallography.net/) and the original data were obtained from the paper: J.J. Finney, N.N. Rao, American Mineralogist 52 (1967) 13-19 (http://rruff.info/doclib/am/vol52/AM52_13.pdf). The mineral studied in this paper has the complex chemical formula: $(Ce_{0.15}La_{0.14}Th_{0.33}Ca_{0.32}Pb_{0.01}U_{0.05})(P_{0.96}Si_{0.04})O_4$. This formula means that the Ce, La, Th, Ca, Pb and U atoms share the same site and the P and Si atoms share another site. XaNSoNS can distribute atoms in unit cells randomly with the respect to the site occupancy numbers. So, the two different runs of the simulation create two different atomic ensembles. The substitutional defects are not the only source of the randomness in this example, because the atoms have nonzero atomic displacement parameters, **`Uiso`**.

The displacements of the atoms are usually caused by the thermal motion. The characteristic time of the thermal motion is much lesser than the time of the exposition. The methods used in XaNSoNS to account for atomic displacement use the opposite limit, as if the time of the exposition would be much lesser than the characteristic time of the thermal motion. Although at first glance, this method looks inappropriate, it leads to the correct result when the number of atoms in the model nanoparticle is large enough (~10^4 or greater). This may be explained as follows. The difference between the real experiment and the simulation in XaNSoNS is that in the real experiment each photon or neutron interact with the slightly different (due to the thermal motion) atomic ensemble, but in the simulation the atomic ensemble is motionless. To correct this, one may perform multiple simulations and then calculate the averaged diffraction pattern. It appears, however, that for the large nanoparticles, different simulation produce almost identical diffraction pattern despite the atomic ensemble is randomized. So, the averaging is unnecessary.

To run the simulation on the OpenCL-compatible GPU, do:

```
python ../../XaNSoNS.py -i Cheralite.xml -v OpenCL
```

or on the CUDA-compatible Nvidia GPU, do:

```
python ../../XaNSoNS.py -i Cheralite.xml -v CUDA
```

The simulation may take several seconds on low-end GPUs with DDR3 memory. Fast memory access is required for the atomic operations (https://en.wikipedia.org/wiki/Linearizability) used in the case of `'Debye_hist'`, `'DebyePDF'` and `'PDFonly'` scenarios.

Fig. **6** shows the results of three different simulations. Despite the randomness in the atomic ensembles the diffraction patterns are almost identical.

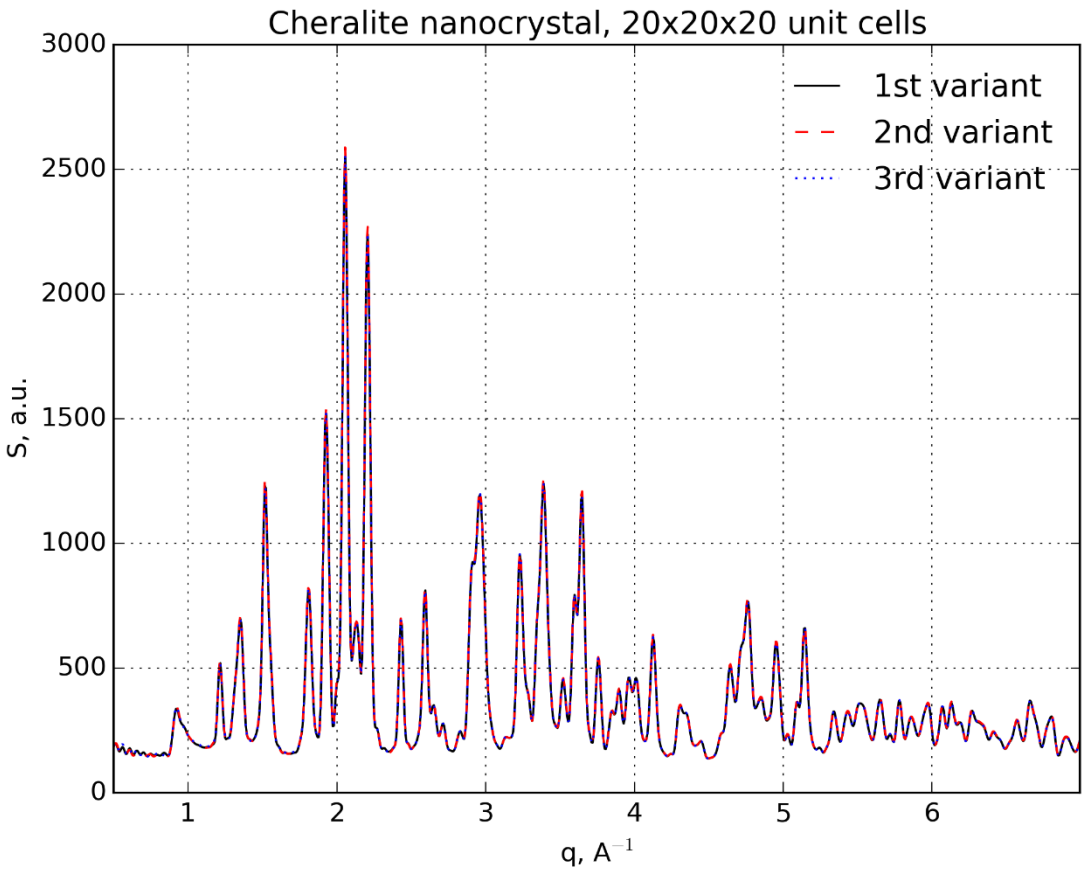

Fig. 6. Diffraction patterns for the cheralite nanocrystal of 20x20x20 unit cells obtained by the three different runs of XaNSoNS. Despite the randomness in the atomic ensembles the diffraction patterns are almost identical.

IV.4 Crystallography Open Database (API for Python)

This example shows how the XaNSoNS API for Python may help to write a simple program which simulates the powder diffraction pattern for the finite-size crystal of a given structure from the Crystallography Open Database (COD), http://www.crystallography.net/cod/index.php.

Navigate to “COD” folder in “examples” folder. The file “COD.py” is a simple python program that takes the COD ID number of the structure and the crystallite size in unit cells as the arguments and simulates and plots the x-ray or neutron powder diffraction pattern.

This program uses the Matplotlib python module (http://matplotlib.org/). This module is installed by default if the Anaconda python distribution is used. In other cases, it can be installed via the default package manager on Linux or via MacPorts on Mac OS (`sudo port install py27-matplotlib`).

The list of COD.py options is given below.

`-i` or `--ID` specifies COD ID number.

`-s` or `--size` specifies the size of the crystallite in unit cells (use coma as separator, default is '10,10,10').

`-n` or `--neutron` switches the source to 'neutron' (default is 'xray').

`-c` or `--cpu` tells the program to use the CPU instead of GPU.

`--CUDA` tells the program to use the CUDA version instead of the OpenCL version when calculating on the GPU.

This program has two major limitations.

1. XaNSoNS does not support the CIF files which do not contain the `'_symmetry_equiv_pos_as_xyz'` or `'_space_group_symop_operation_xyz'` key. The application knows nothing about the symmetry groups and is unable to perform the symmetry operations without this key. About 4% of COD entries cannot be processed for this reason.
2. The Periodictable module does not contain the x-ray from-factors for certain ions. Therefore, the x-ray diffraction patterns cannot be calculated for another 4% of COD entries.

The COD IDs of the structures for which XaNSoNS cannot calculate x-ray or neutron diffraction patterns may be found in the files "COD_blacklist_xray.txt" and "COD_blacklist_neutron.txt". These files are created on 8th December 2016 and do not include the IDs of the structures added to the COD after this date.

For example, to simulate the x-ray diffraction pattern for the crystallite of the structure with COD ID – 1000105 and the size of 15x15x15 unit cells, do:

```
python COD.py -i 1000105 -s 15,15,15
```

The Matplotlib window with the simulated diffraction pattern will pop-up (see Fig. 7). The program saves the text file with the diffraction pattern in the working directory. The CIF file, the from-factor table and the XML file are saved in "cif", "ff" and "xml" directories respectively.

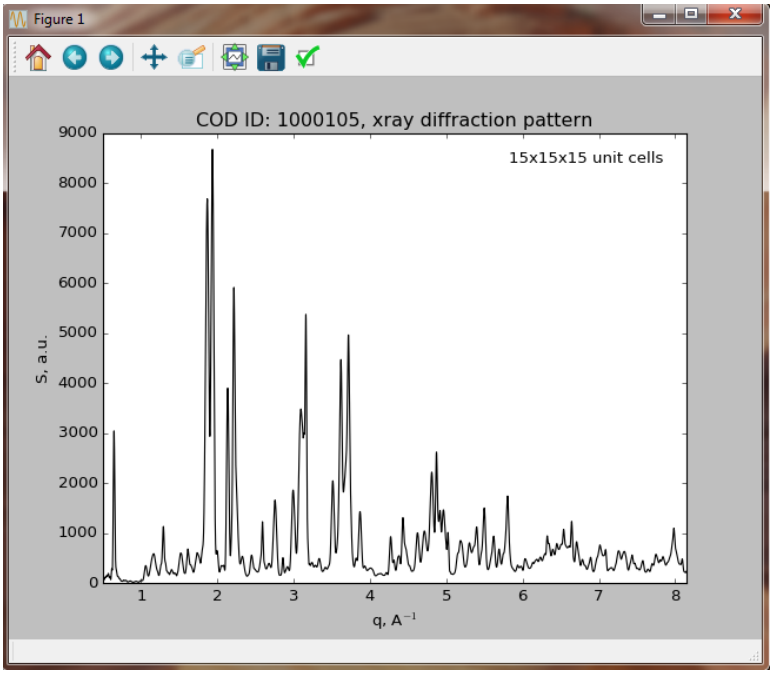

Fig. 7. Matplotlib window with the simulated diffraction pattern for the crystallite (15x15x15 unit cells) of the structure with COD ID – 1000105.

The only class ProcessCOD in COD.py has only 68 lines of code excluding the docstrings and comments. This class:

1) downloads the CIF file,
2) creates the XML configuration,

3) imports the data from the CIF file to the XML configuration,
4) fills in the form-factor tables,
5) runs the simulation and
6) plots the results.

The code contains many comments, so it should be easy to understand how it works.

IV.5 Experimental data processing for C_{60} fullerene (API for Python)

This example shows how to organize the series of simulations with the help of the XaNSoNS API for Python and how to fit the experimental data with the optimization tools available in the SciPy python module.

Navigate to "C60" folder in "examples" folder. File "experiment.txt" contains the neutron diffraction pattern of the crystalline C_{60} fullerene. The data are provided by P.A. Borisova and are published in [P.A. Borisova, M.S. Blanter, V.V. Brazhkin, V.A. Somenkov, V.P. Filonenko, "Phase transformations in amorphous fullerite C_{60} under high pressure and high temperature", J. Physics Chem. of Solids, 83 (2015), 104-108, http://www.sciencedirect.com/science/article/pii/S0022369715000888] (see curve "1" in Fig. 2). The samples of C_{60} fullerenes of 99.5% purity produced by "NeoTechProduct" were obtained by the high-temperature treatment of graphite, followed by isolation with organic solvents and subsequent chromatographic separation. The sample has FCC lattice with a period 14.16 Å. The diffraction pattern shown in Fig. **8** was obtained with the neutron multidetector (224 scintillation detectors) diffractometer DISC [V.P. Glazkov, I.V. Naumov, V.A. Somenkov, S.Sh. Shil'shtein, Nucl. Instr. Methods A264 (1988) 367, doi:10.1007/BF02471308] in Kurchatov Institute. The wavelength λ of monochromatic heat neutron waves of 1.67 Å. The resolution is $\Delta d/d = \Delta q/q \sim 2$ - 3%.

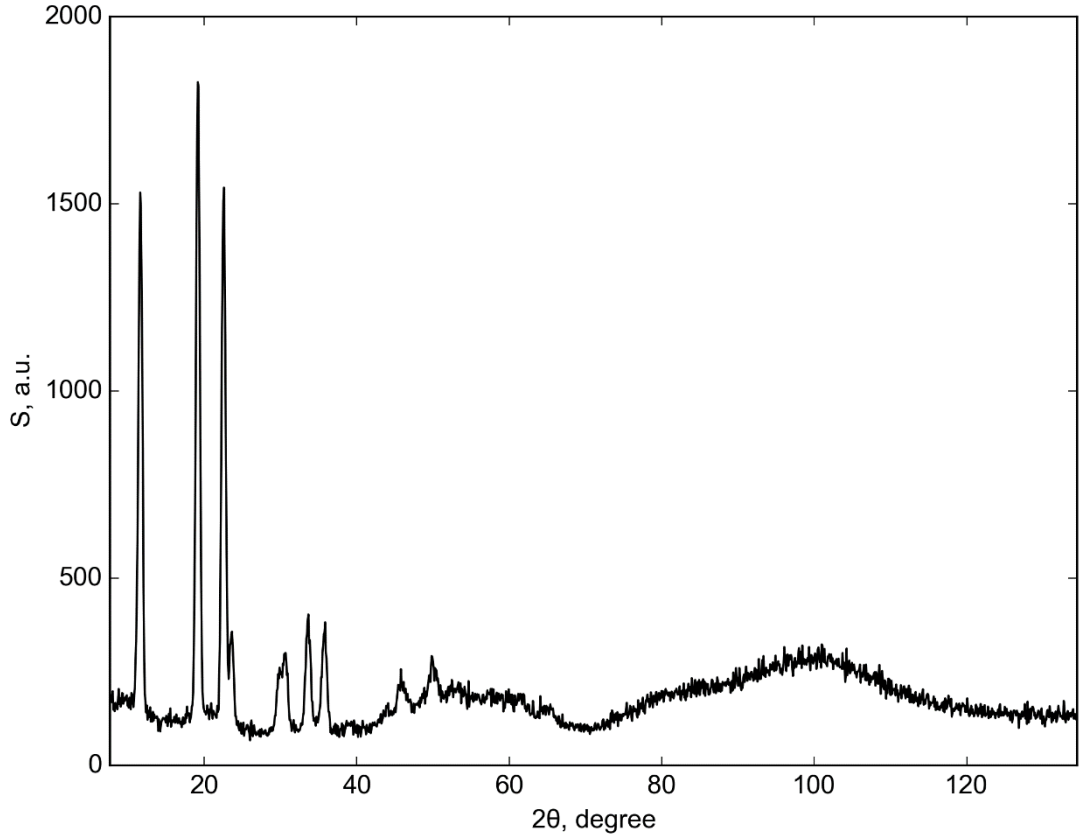

Fig. 8. Experimental diffraction pattern of the crystalline C_{60} sample.

In this example, we'll try to estimate the lower limit for the value of the average size of the crystallite in the polycrystalline sample as well as the possible value of the atomic displacement parameter, **Uiso**.

The file "C60.xml" has the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<PMML>
   <Calculation FFfilename="FFtable.txt" name="C60_16x16x16"
scenario="debye_hist" source="neutron">      <q max="7.2"
min="0.42" N="8096"/>
   </Calculation>
   <Block mol_rotation="yes" centeredAtoms="yes">
      <Atoms>
         <Atom name="C" r="2.22673 0.59492 2.6842"/>
         <Atom name="C" r="3.13408 0.16181 1.63544"/>
          …57 lines skipped…
         <Atom name="C" r="0.023 -0.07494 -3.55242"/>
      </Atoms>
      <SymmEqPos R1="1 0 0" R2="0 1 0" R3="0 0 1" r="0.5 0.5 0"/>
      <SymmEqPos R1="1 0 0" R2="0 1 0" R3="0 0 1" r="0.5 0 0.5"/>
      <SymmEqPos R1="1 0 0" R2="0 1 0" R3="0 0 1" r="0 0.5 0.5"/>
      <CellVectors N="16 16 16" a="14.16 0 0" b="0 14.16 0" c="0 0
14.16"/>
   </Block>
</PMML>
```

This file contains the parameters of the simulation of the neutron powder diffraction pattern for the single nanocrystal of C_{60} fullerene with the size of 16x16x16 unit cells. The mesh of the *q* values contains N = 8096 points. Such a high resolution is required not only to calculate the sharp peaks with high accuracy but also to convolute the simulated diffraction pattern with the instrument function. The element <**Atoms**/> describes the single C_{60} molecule. To account for the thermal rotation of the C_{60} molecules, which takes place in the real sample, the parameter **mol_rotation** is set to `"yes"`. Let's see, how this affects the simulated diffraction pattern. Fig. **9** shows the comparison of the diffraction pattern simulated with enabled thermal rotation of the C_{60} molecules (`mol_rotation="yes"`) with that simulated with static C_{60} molecules (`mol_rotation="no"`). Visual comparison of these diffraction patterns with the experimental data of Fig. **8** shows that the thermal rotation of the spherical molecules must be taken into account when simulating the diffraction patterns.

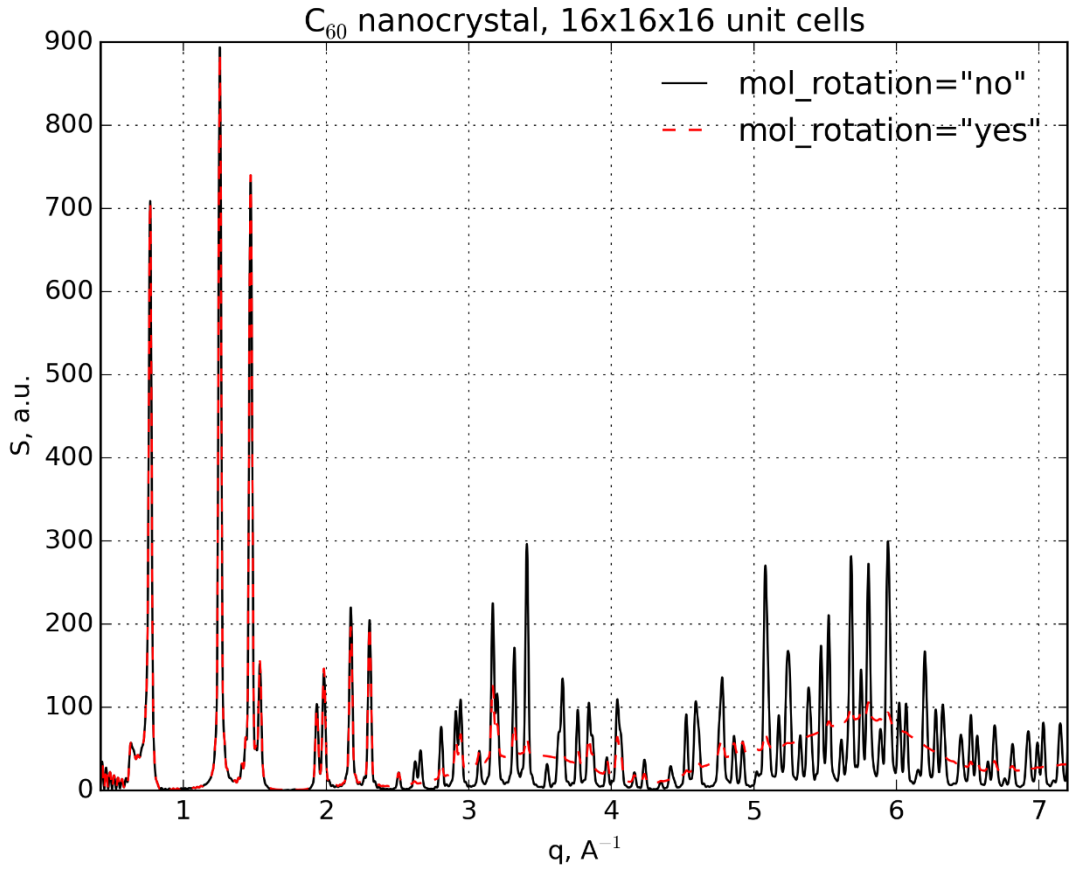

Fig. 9. Comparison of the diffraction pattern of the C_{60} nanocrystal with the size of 16x16x16 unit cells (16384 fullerene molecules) simulated with the rotating C_{60} molecules (**mol_rotation**="yes") with that simulated with the static C_{60} molecules (**mol_rotation**="no").

Before starting the analysis of the experimental data, let's see how the parameters of the structure, namely the size of the nanocrystal and the displacement of atoms, affect the simulated diffraction pattern. Fig. **10** shows the diffraction patterns for the nanocrystals with the size of: 10x10x10 (4000 molecules), 12x12x12 (6912 molecules), 14x14x14 (10976 molecules) and 16x16x16 (16384 molecules) unit cells. Fig. **11** shows the diffraction patterns for the nanocrystals with the size of 16x16x16 unit cells but with various values of the atomic displacement parameter, **Uiso**: 0, 0.033 Å^2 and 0.066 Å^2. While the size of the nanocrystal affects mostly the low-*q* side of the diffraction pattern, the atomic displacement parameter affects mostly the high-*q* side.

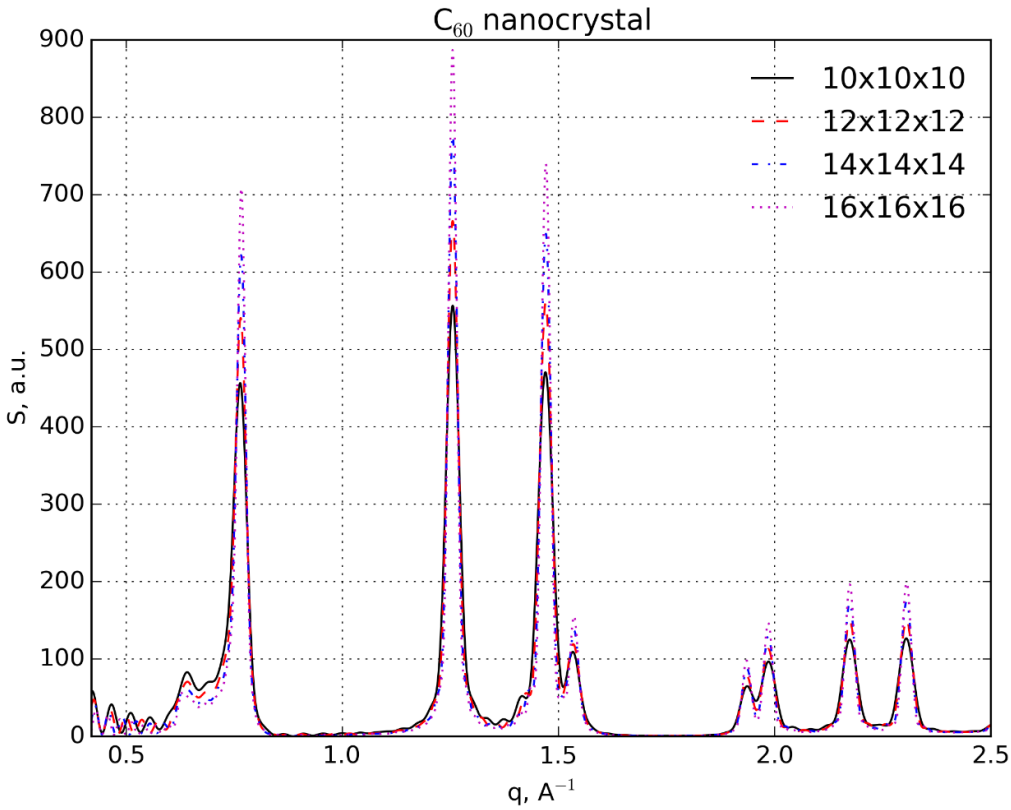

Fig. 10. Diffraction patterns for the C_{60} nanocrystals with the size of: 10x10x10, 12x12x12, 14x14x14 and 16x16x16 unit cells.

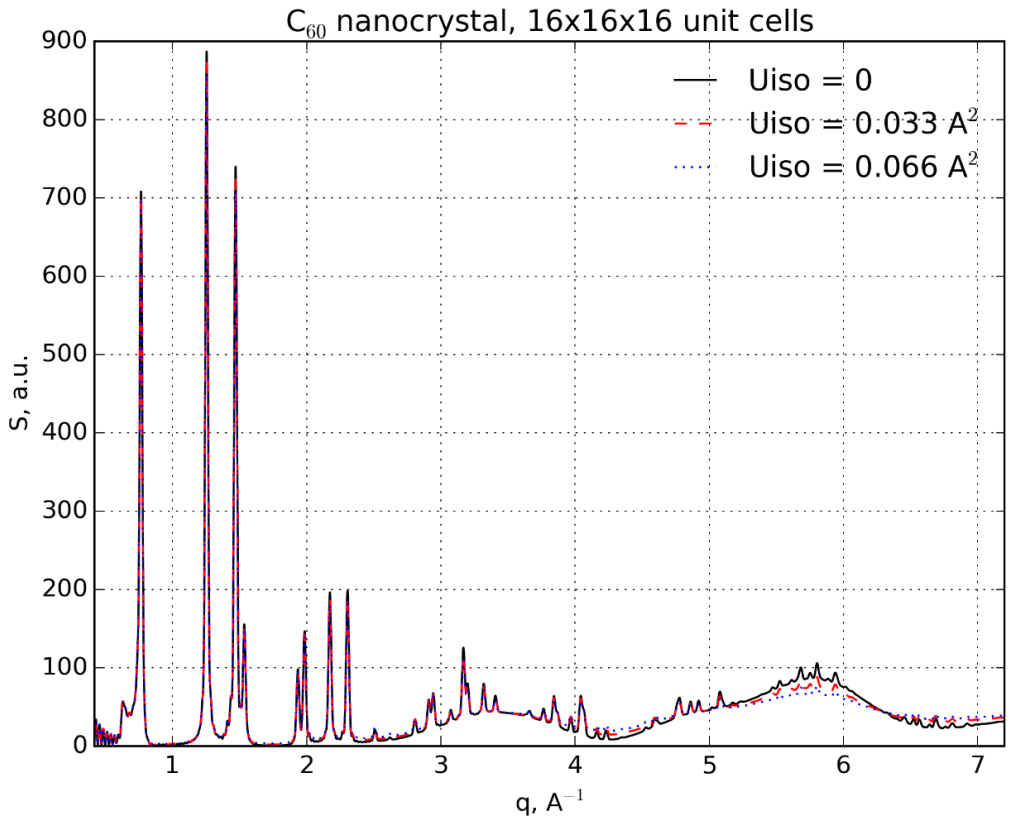

Fig. 11. Diffraction patterns for the C_{60} nanocrystals with the size of 16x16x16 unit cells but with various values of the atomic displacement parameter, **Uiso**: 0, 0.033 Å^2 and 0.066 Å^2.

When comparing the simulated diffraction patterns of the nanocrystals with the experiment it is important to account for the peak broadening due to the finite resolution of the experiment. In

our case the resolution is $\Delta q/q$ ~ 2 - 3%. Fig. **12** shows the simulated diffraction patterns of the C_{60} nanocrystal with the size of 16x16x16 unit cells before and after the instrument function correction for the case of $\Delta q/q$ = 2%. The instrument function correction takes the form of Eq. (**15**):

$$S_{corr}(q) = \int_{-\infty}^{+\infty} S(q')I(q-q')dq', \quad I(q-q') = \frac{\exp\left(-\frac{(q-q')^2}{2\sigma^2(q)}\right)}{\sigma(q)\sqrt{2\pi}}, \quad \sigma(q) = \frac{0.02q}{2.355} \qquad (15)$$

The convolution with the instrument function changes the diffraction pattern significantly even if the value of $\Delta q/q$ is equal to 2%. If the value of $\Delta q/q$ is equal to 3%, the FWHM of the instrument function $I(q - q')$ is approximately equal to the FWHM of the peaks in the experimental diffraction pattern making it impossible to estimate the size of the single crystal in the sample. Therefore, we can only estimate the **lower limit** of this size if the value of $\Delta q/q$ is equal to 2%.

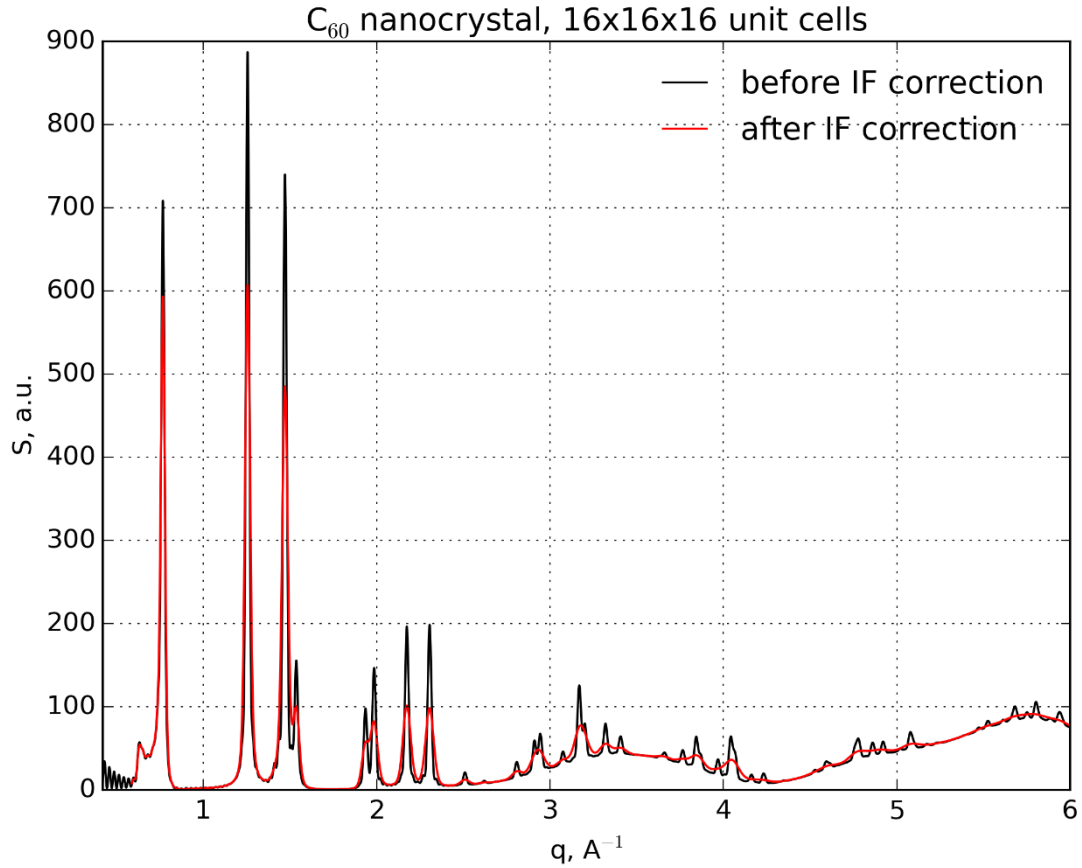

Fig. 12. Simulated diffraction patterns of the C_{60} nanocrystal with the size of 16x16x16 unit cells before and after the instrument function correction for the case of $\Delta q/q$ = 2%.

Fitting the experimental data may be formulated as the minimization problem:

$$\left\|\mathbf{S_{calc}X} - \mathbf{S_{exp}}\right\|_2 \underset{\mathbf{X}}{\rightarrow} min, \quad x \geq 0, x \in \mathbf{X} \qquad (16)$$

where $\mathbf{S_{calc}}$ is the matrix which columns are the simulated diffraction patterns normalized to the number of atoms (*the patterns simulated by XaNSoNS are already normalized to the number of atoms*); $\mathbf{S_{exp}}$ is the column vector of the experimental diffraction pattern; **X** is the column vector of the unknown weight coefficients. If the experimental data contains the constant background

(as in our case) the additional column filled with “1” should be added to the $\mathbf{S}_{calc}$ and the additional unknown value should be added to **X**.

The problem (**16**) may be solved just in one line of code with the SciPy non-negative least squares solver (http://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.nnls.html).

The python script “C60_data_processing.py” does the experimental data processing. The first 42 lines of code are shown in Fig. **13**.

```
1  import sys
2  sys.path.append('../../')
3  import XaNSoNS
4  import numpy as np
5  from scipy import interpolate, optimize
6  from matplotlib import pyplot as plt
7  wavelength=1.67 #neutron source wavelength in Angstrom
8  exp_data=np.loadtxt('experiment.txt')#loading the experimental diffraction pattern
9  theta_exp=exp_data[:,0]
10 q_exp=4.*np.pi*np.sin(theta_exp/360.*np.pi)/wavelength
11 pattern_exp=exp_data[:,1]
12 XaNSoNS.formfactors('C60.xml','')#creating the file with form factors
13 config=XaNSoNS.config('C60.xml')#loading the initial configuration of parameters
14 Uiso_list=[0,0.033,0.066]#3 variants for the atomic displacement patameter
15 Ncell_list=range(10,18)#8 variants for the nanocrystal sizes in terms of the number of unit cells
16 sim_data=[np.ones(theta_exp.size)]#1st element in the list is for the constant background
17 labels=[]
18 for Uiso in Uiso_list:
19     for Atom in config.Block[0].Atoms.Atom:
20         Atom.Uiso=Uiso#setting the atomic displacement parameter for an atom
21     for Ncell in Ncell_list:
22         config.Block[0].CellVectors.N=Ncell*np.ones(3)#setting the number of unit cells
23         #setting the new name for the simulation
24         config.Calculation.name='C60_%dx%dx%d_Uiso_%.3f'%(Ncell,Ncell,Ncell,Uiso)
25         labels.append(config.Calculation.name)
26         XMLname='%s.xml'%config.Calculation.name
27         config.ToXML(XMLname)#saving the configuration to the XML file
28         #running the simulation
29         XaNSoNS.run(XMLname, version='OpenCL',CIFcheck=False,FFcheck=False).wait()
30         filename=config.Calculation.name+'_%s_1D.txt'%config.Calculation.source.lower()[:4]
31         data=np.loadtxt(filename)#loading the results
32         data_interp=interpolate.interp1d(data[:,0],data[:,1])#interpolating the data
33         #convoluting with the instrument function (delta(q)/q=0.02)
34         data_corrected=np.zeros(q_exp.size)
35         for i in range(q_exp.size):
36             sigma=0.02/2.355*q_exp[i]
37             q1,dq=np.linspace(q_exp[i]-3.5*sigma,q_exp[i]+3.5*sigma,100,retstep=True)
38             IF=np.exp(-(q_exp[i]-q1)**2/(2.*sigma**2))/(sigma*np.sqrt(2.*np.pi))
39             data_corrected[i]=(data_interp(q1)*IF).sum()*dq
40         sim_data.append(data_corrected)#storing the corrected data to the list
41 sim_data=np.array(sim_data).T#converting list to numpy array and trasposing
42 x,norm=optimize.nnls(sim_data[90:,:],pattern_exp[90:])#solving the problem
```

Fig. 13. First 42 lines of “C60_data_processing.py” python script.

This script does the following:

lines 7-11: reads the experimental data and converts the values of 2θ into the values of q;

lines 12-13: fills the neutron scattering lengths table and loads the initial configuration from “C60.xml” file;

lines 14-15: creates the lists with the variant for the atomic displacement parameter and the nanocrystal size (in term of the number of unit cells);

line 16: creates the list for the calculated diffraction patterns and fills the 1st element with “1” for the constant background;

lines 19,20,22-27: sets the parameters for the current simulation and saves them to the XML file;

lines 28: calls the OpenCL version of XaNSoNS to simulate the diffraction pattern on the GPU;

lines 30-32: reads the results and interpolates the data;

lines 33-39: convolutes the simulated diffraction pattern with the instrument function for $\Delta q/q$ = 2%;

line 40: appends the corrected data to the list;

line 41: converts the list into the two-dimensional numpy array and transpose it;

line 42: solves the problem (**16**).

Note, that the first 90 values of q are excluded when solving the problem (**16**) to exclude the first peak at 7.67 $Å^{-1}$ because its broadening appears to be stronger than the broadening of the other peaks (the values of $\Delta q/q$ may be a little higher than 2% for low q).

The last 20 lines of the script plot the results of the fitting (see Fig. **14)**.

Additional python modules SciPy and Matplotlib should be installed to run the script. These modules are installed by default if the Anaconda python distribution is used. In other case, they can be installed via the package manager on Linux or via MacPorts on Mac OS (port names: py27-scipy and py27-matplotlib).

To run the script, do:

```
python C60_data_processing.py
```

The calculation may take from several minutes to few tens of minutes depending on the performance of the GPU.

It appears that the size of the single crystal in the sample is definitely not less than 22 nm. The atomic displacement parameter is lower than 0.066 $Å^2$ and is about ~ 0.04 $Å^2$.

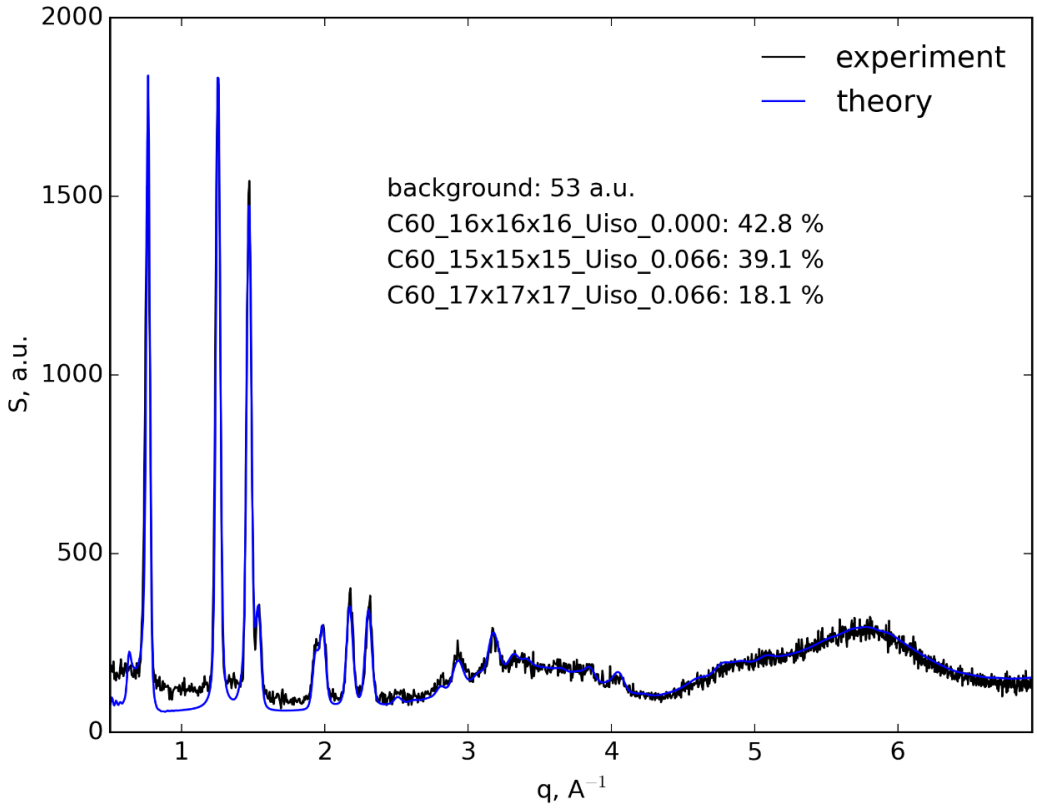

Fig. 14. Results of the fitting the experimental diffraction pattern.

Appendix A. Output file formats

The description of all possible output files is given in the table below. The names of the files start with the prefix defined in the **name** attribute of <**Calculation**/> element in XML file.

Data	Suffix	Extension	Condition for the creation	Format	Commentary
Atomic ensemble	_atoms	.xyz	**PrintAtoms**='yes'	n e_1 X_1 Y_1 Z_1 e_2 X_2 Y_2 Z_2 … e_n X_n Y_n Z_n	n is the total number of atoms in model sample, e is the chemical element (isotope, ion) symbol, X, Y, Z – coordinates of atom in Å.
1D diffraction pattern	_xray_1D _neut_1D	.txt	**scenario**='Debye' **scenario**='Debye_hist' **scenario**='DebyePDF' **scenario**='2D'	q_1 S_1 q_2 S_2 … q_n S_n	q is the scattering vector magnitude value in $Å^{-1}$, S is the scattering intensity value, The parameters of the mesh are defined in <**q**> element in the XML file.
2D diffraction pattern	_xray_2D _neut_2D	.txt	**scenario**='2D'	S_{11} S_{12} … S_{1m} S_{21} S_{22} … S_{2m} … S_{n1} S_{n2} … S_{nm}	S is the scattering intensity value, 1st index is a number of a point in the scattering vector magnitude mesh, 2nd index is a number of a point in the mesh of the polar angle of the scattering vector, ϕ_q (see Fig. **2**). The parameters of the meshes are defined in <**q**> element in the XML file.
Partial 1D diffraction patterns	_xray_1D_parts _neut_1D_parts	.txt	**scenario**='Debye' **PartialIntensity**='yes' The number of the <Block/> elements in the XML is greater than 1.	q_1 $S11_1$ $S12_1$ … $S22_1$ $S23_1$ … SNN_1 q_2 $S11_2$ $S12_2$ … $S22_2$ $S23_2$ … SNN_2 … q_n $S11_n$ $S12_n$ … $S22_n$ $S23_n$ … SNN_n	q is the scattering vector magnitude value in $Å^{-1}$, Smk is the partial contribution to the Debye sum from the structural blocks with indexes *m* and *k*, m ≤ k.

					If the number of blocks is *N*, the number of columns (including the first column with `q`) is $N(N+1)/2 + 1$. The sum over the all `Smk` in a row gives the total scattering intensity for the respective value of `q`.
PDF, RDF, rPDF.	_xray_PDF _neut_PDF _xray_RDF _neut_RDF _xray_rPDF _neut_rPDF	.txt	**scenario**='DebyePDF' **scenario**='PDFonly'	r_1 F_1 r_2 F_2 … r_n F_n	`r` is the interatomic distance in Å, `F` is the PDF, RDF or rPDF value (see Eq. (**9**)). The step between the r_i and r_{i+1} values defined by **hist_bin** attribute in the <**Calculation**> element, while the other parameters of PDF functions are described in the <**PDF**> element in the XML file.
Partial PDF, RDF or rPDF for the pair (e_1,e_2) of the chemical elements.	_xray_PDF_e_1-e_2 _neut_PDF_e_1-e_2 _xray_RDF_e_1-e_2 _neut_RDF_e_1-e_2 _xray_rPDF_e_1-e_2 _neut_rPDF_e_1-e_2	.txt	**scenario**='DebyePDF' **scenario**='PDFonly' **PrintPartial**='yes'	r_1 F_1 r_2 F_2 … r_n F_n	`r` is the interatomic distance in Å, `F` is the Partial PDF, RDF or rPDF value (see Eq. (**6**)-(**8**)). The partial PDF for the given pair of chemical elements is printed to the separate file. If the total number of chemical elements in the model sample is *N*, the number of files is $N(N+1)/2$.

Appendix B. Input file formats

The description of possible input files is given in the table below.

Data	**Attribute in the XML that specifies the path to the file**	**Format**	**Commentary**
User defined atomic x-ray form factors	<**Calculation**>→**FFfilename**	q q_1 q_2 … q_m e_1 F_{11} F_{12} … F_{1m} … e_n F_{n1} F_{n2} … F_{nm}	q is the 1st symbol in the row followed by the scattering vector magnitude values q_1 q_2 … q_m, e is the symbol of the chemical element (ion), F is the value of the atomic x-ray form factor for the respective chemical element (row) and the respective value of q (column). The values of q do not necessary have to coincide with the mesh values defined in the element <**q**> of the XML file. XaNSoNS will interpolate the form factors linearly between the values of q provided with the first row of this file. Therefore, the range of the values of q should include the values of the mesh defined in the element <**q**> of the XML file. The user does not need to provide all the form factors, but only the non-standard ones (namely, the form factors that are different from those provided by the Periodictable python package).
User defined neutron scattering lengths	<**Calculation**>→**FFfilename**	e_1 L_1 e_2 L_2 … e_n L_n	e is the symbol of the chemical element (isotope), L is the neutron coherent scattering length value. As it is for the x-ray form factors, the user does not need to provide all the scattering lengths, but only those that are different from the scattering lengths provided by the Periodictable python package).
Atomic configuration of the structural block.	<**Block**>→<**Atoms**>→**filename**	n e_1 x_1 y_1 z_1 u_1 o_1 e_2 x_2 y_2 z_2 u_2 o_2 … e_n x_n y_n z_n u_n o_n	n is the total number of atoms in the configuration, e is the symbol of the chemical element (ion, isotope) or its number in the periodic table, x, y, z are the Cartesian coordinates in Å or the fractional coordinates of the atom (**fractional** attribute in the XML file should be set to 'yes' in the latter case), u is the isotropic displacement parameter of the atom in $Å^2$, o is the site occupancy value.

			The values colored with blue are optional. The format is compatible with .xyz. If the o parameter needs to be specified, the u parameter should be specified as well even if it has the default value (1.0).
Spatial positions of the copies of the structural block.	<**Block**>→<**Copies**>→**filename**	x_1 y_1 z_1 α_1 β_1 γ_1 x_2 y_2 z_2 α_2 β_2 γ_2 … x_m y_m z_m α_m β_m γ_m	x, y, z are the Cartesian coordinates of the copy of the structural block in Å (depending on the value of the attribute **centered** the vector (x,y,z) may point to the geometrical center of the structural block ('yes') or to the point (0,0,0) of the internal coordinate system of the structural block ('no')), α β γ are the Euler angles of the copy of the structural block. The order of rotations is defined by the **convention** attribute of the <**Copies**> node in the XML file.