BornAgain - simulating and fitting X-ray and neutron scattering at grazing incidence.

User Guide

version 0.1

Scientific Computing Group at FRM-II

July 15, 2013

Contents

0.1 Introduction

`BornAgain` is a software to simulate and fit neutron and X-ray scattering at grazing incidence. It is a multi–platform open–source project that aims at supporting scientists in the analysis and fitting of their GISAS data, both for synchrotron (GISAXS) and neutron (GISANS) facilities. The name of the software, `BornAgain` indicates the central role of the distorted-wave Born approximation (DWBA) in the physical description of the scattering process. The software provides a generic framework for modeling multilayer samples with smooth or rough interfaces and with various types of embedded nanoparticles. In this way, it reproduces and enhances the functionality of the present reference software, `IsGISAXS` by R. Lazzari [1], and lays a solid base for future extensions in response to specific user needs.

To meet the growing demand for GISAS simulation of more complex structured materials, BornAgain has extended the IsGISAXS program's functionality by removing the restrictions on the number of layers and particles, by providing diffuse reflection from rough layer interfaces and by adding particles with inner structure.

The user guide starts with a brief description of steps necessary for compiling and running the simulation, Section 0.2. More detailed overview of software architecture and installation procedure is given in Section 0.4. General methodology of simulation with `BornAgain` and detailed usage examples are given in Section **??**. Fitting tools provided by the frame work are presented in Section **??**.

Icons used in this manual:

 : this sign highlights further references.

 : this sign highlights essential points.

0.2 Quick start

This section shortly describes how to build `BornAgain` from source and run first simulation. More details about software architecture and installation procedure are given in Section 0.3 and Section **??**.

Step I: installing third party libraries

- boost library (≥ 1.48)
- GNU scientific library (≥ 1.15)
- fftw3 library (≥ 3.3.1)
- Eigen3 library (≥ 3.1.0), optional
- ROOT framework (≥ 5.34.00), optional

Step II: getting the source

```
git clone git://apps.jcns.fz-juelich.de/BornAgain.git
```

Step III: building the source

```
mkdir <build_dir>; cd <build_dir>;
cmake <source_dir> -DCMAKE_INSTALL_PREFFIX=<install_dir>
make
make check
make install
```

Step IV: running example

```
cd <install_dir>/Examples/python/ex001_CylindersAndPrisms
python CylindersAndPrisms.py
```

0.3 Software architecture

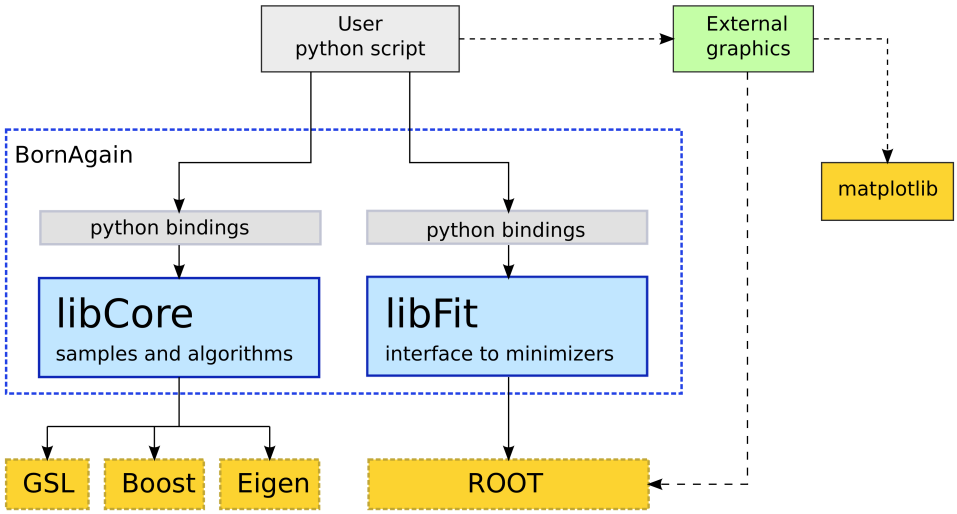

Figure 1: Left:

0.4 Installation

This section shortly describes how to build `BornAgain` from source and run first simulation.

0.5 Examples

0.6 General methodology

A simulation of GISAXS using BornAgain platform can be decomposed into the following points:

- Definition of the materials, specifying their names and their refractive indices,
- Definition of particles: shapes, sizes, refractive indices of the constituting material, interference functions,
- Definition of the layers: thicknesses, links with the previously defined materials,
- Inclusion of the particles in layers: density, positions, orientations,
- Assembling the sample: generation of a mulitlayered system,
- Specifying the input beam and the output detector's characteristics,
- Running the simulation,
- Saving the data.

The sample is built from object oriented building blocks **instead of the more common implementation of loading data files.**

0.7 Conventions

polarization term

0.7.1 Geometry of the sample

- Definitions of the angles:
  - Definitions of the layers:
  - Definitions of the particles:

vertical position in each layer.

horizontal distribution of particles.

The complex refractive index associated with a layer or a particle is written as $n = 1 - \delta - i\beta$, with $\delta, \beta \in \mathbb{R}_+$ and $\delta, \beta \ll 1$.

Remark - Order of the different steps:
When assembling the sample, the layers are defined from top to bottom. So in most cases the first layer will be the air layer.

The input beam is assumed to be monochromatic without any spatial divergence.

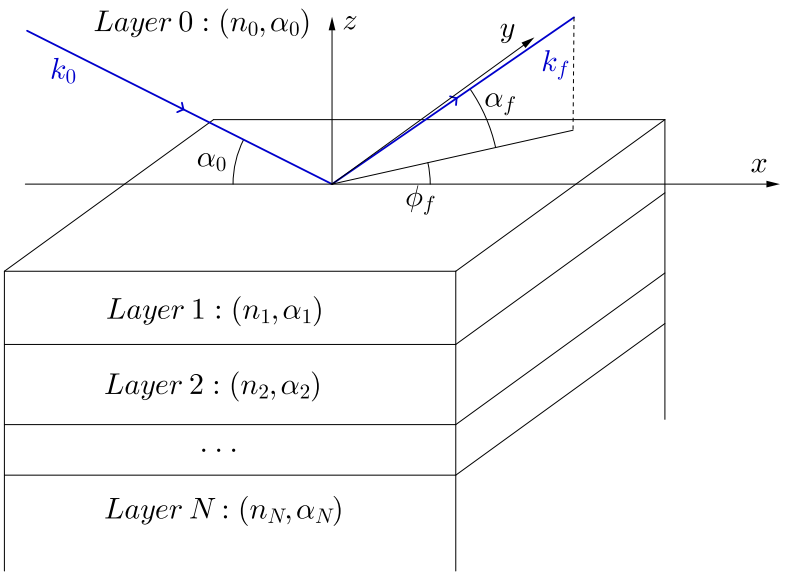

Figure 2: Representation of the scattering geometry for multilayer specular reflectivity. n_i is the refractive index of layer i and α_i is the incident angle of the wave propagating in layer i and incident on layer $i+1$. α_f is the exit angle with respect to the sample's surface and ϕ_f is the scattering angle with respect to the scattering plane.

0.7.2 Units

By default angles are expressed in radians and lengths are given in nanometers. But it is possible to use other units by specifying them right after the value of the corresponding parameter.

0.7.3 Programs

Programming: The examples presented in the next paragraphs are written in C++ or Python. For tutorials about these programming languages, the users are referred to [3] and [4] respectively.

Note about the version of C++ and Python to run the examples.

Where can the following examples be found?

List of examples:
What is the command to run the examples?

ex-1	Two types of islands (cylinder, prism3) on top of substrate. No interference function
ex-2	Bimodal cylinders on top of substrate. Two gaussian size cylinder distribution
ex-3	Cylinder formfactor in BA and DWBA
ex-4	1D and 2D paracrystal
ex-5	1D paracrystal fit example
ex-6	2D lattice with different disorder
ex-7	Mixture of different particles defined in morphology file
ex-8	2DDL paracrystal
ex-9	Pyramids on top of substrate Rotated pyramids on top of substrate
ex-10	Cylinders with interference on top of substrate
ex-11	Core shell nano particles
ex-12	Constrained fit example Mixture of two cylinder types
ex-13	Simulated annealing fit example
ex-14	Layered spheres on graded interface
ex-15	Size spacing correlation approximation

Table 1: List of form factors implemented in BornAgain.

0.8 Example 1: Two types of islands on top of substrate. No interference function

In this example, using Python language, we simulate the scattering from a mixture of cylindrical and prismatic nanoparticles without any interference between them. These particles are placed on top of a substrate. We are going to go through each step of the simulation. The Python script specific to one stage will be given at the beginning. But for the sake of completeness the full code is given at the end of this section (Listing 1).

It starts by importing different functions from external modules (lines 1-7). For example, line 3 imports NumPy, which is the fundamental package for scientific computing with Python (`http://www.numpy.org/`). In particular, line 7 imports the features of BornAgain software.

```
1 import sys
2 import os
3 import numpy
4
5 sys.path.append(os.path.abspath(os.path.join(os.path.split(__file__)[0],'..', '..', '..', 'lib
      ')))
6
7 from libBornAgainCore import *
```

First step: Defining materials

```
9  #  defining materials
10 mAmbience = MaterialManager.getHomogeneousMaterial("Air", 1.0, 0.0 )
11 mSubstrate = MaterialManager.getHomogeneousMaterial("Substrate", 1.0-6e-6, 2e-8)
```

Lines 10 and 11 define two different materials using function `getHomogeneousMaterial` from class `MaterialManager`. The general syntax is the following

```
Interface material name = MaterialManager.getHomogeneousMaterial("name", Re(n), Im(n))
```

where `name` is the name of the material associated with its complex refractive index `n` decomposed into its real and imaginary parts. `Interface material name` is later used when referring to this particular material. The two defined materials in this example are `Air` with a refractive index of 1 and a `Substrate` associated with a complex refractive index equal to $1-6\times10^{-6}-i2\times10^{-8}$.

Remark: there is no condition on the choice of `name`.

Second step: Defining the particles

```
12 # collection of particles
13 n_particle = complex(1.0-6e-4, 2e-8)
14 cylinder_ff = FormFactorCylinder(5*nanometer, 5*nanometer)
15 cylinder = Particle(n_particle, cylinder_ff)
16 prism_ff = FormFactorPrism3(5*nanometer, 5*nanometer)
17 prism = Particle(n_particle, prism_ff)
```

We implement two different shapes of particles: cylinders and prisms (elongated particle with a constant equilateral triangular cross section).
All particles implemented in BornAgain are defined by their form factors (*i.e.* The Fourier transform of the shape function - see the list of form factors implemented in BornAgain), their sizes and the refractive index of the material they are made of. The number of input parameters for the form factor depends on the particle symmetry; it ranges from one parameter for a sphere (its radius) to three for an ellipsoid (its three main axis lengths). Here, for the cylinders we can input its radius and its height. For the prism, the possible inputs are the length of one side of its equilateral triangular base and its height.

In line 13, we define the complex refractive index associated with both particle shapes: `n`$= 1-6\times10^{-4}-i2\times10^{-8}$.

In order to define a particle, we proceed in two steps. For example for the cylindrical particle, we first specify the form factor of a cylinder with its radius and height, both equal to 5 nanometers in this particular case (see line 14). Then we associate this shape with the refractive index of the constituting material in line 15.

The same procedure has been applied for the prism in line 16 and 17 respectively.

Third step: Characterizing the layers and assembling the sample

Particle decoration

```
18 particle_decoration = ParticleDecoration()
19 particle_decoration.addParticle(cylinder, 0.0, 0.5)
20 particle_decoration.addParticle(prism, 0.0, 0.5)
21 interference = InterferenceFunctionNone()
22 particle_decoration.addInterferenceFunction(interference)
```

The process of defining the positions and densities of particles in our sample is called "particle decoration". We use the functions `ParticleDecoration()` (line 18) and the associated `addParticle` (lines 19, 20). The general syntax is

```
particledecoration.addParticle(particle_name, depth, abundance)
```

where `particle_name` is the name used to define the particles (lines 15 and 17), `depth` (default value =0) is the vertical positions, expressed in nanometers, of the particles in a given layer (the association with a particular layer will be done during the next step) and `abundance` is the proportion of this type of particles, normalized to the total number of particles, *i.e.* here we have 50% of cylinders and 50% of prisms.

Remark - Depth of particles
The vertical positions of particles in a layer are given in relative coordinates. For the top layer, the bottom corresponds to `depth=0`. But for all the other layers, it is the top of the layer which corresponds to `depth=0`.

Finally lines 21 and 22 specify that there is **no coherent interference** between the waves scattered by these particles. The intensity is calculated by the incoherent sum of the scattered waves: $\langle |F_n|^2 \rangle$, where F_n is the form factor associated with the particle of type n. The way these waves interfere imposes the horizontal distribution of the particles as the interference reflects the long or short-range order of the particles distribution. On the opposite, the vertical position is imposed when we add the particles in a given layer by parameter `depth`, as shown in line 19 and 20.

Multilayer

```
23 # air layer with particles and substrate form multi layer
24 air_layer = Layer(mAmbience)
25 air_layer_decorator = LayerDecorator(air_layer, particle_decoration)
26 substrate_layer = Layer(mSubstrate, 0)
27 multi_layer = MultiLayer()
28 multi_layer.addLayer(air_layer_decorator)
29 multi_layer.addLayer(substrate_layer)
```

We now have to configure our sample. For this first example, the particles, cylinders and prisms, are on top of a substrate in an air layer. **The order in which we define these layers is important: we start from the top layer up to the bottom one**.

Let us start with the air layer. It contains the particles. In line 24, we use the previously defined `mAmbience` (="air" material) (line 10). The command written in line 25 shows that this layer is decorated by adding the particles using the function `particledecoration` defined in lines 18-22. Note that the `depth` is referenced to the bottom of the top layer (negative alues would correspond to particles floating above the first layer as the vertical axis is pointing upwards) . The substrate layer only contains the substrate material (line 26).

There are different possible syntaxes to define a layer. As shown in lines 24 and 26, we can use `Layer(Interface material name,thickness)` or `Layer(Interface material name)`. The `thickness` is expressed in nanometers.

Our two layers are now fully characterized. The sample is assembled using `MultiLayer()` constructor (line 27): we start with the air layer decorated with the particles (line 29), which is the layer at the top and end with the bottom layer, which is the substrate (line 29).

Fourth step: Characterizing the input beam and output detector and running the simulation

```
30 # run simulation
31 simulation = Simulation()
32 simulation.setDetectorParameters(100,-1.0*degree, 1.0*degree,
33                                  100, 0.0*degree, 2.0*degree, True)
34 simulation.setBeamParameters(1.0*angstrom, -0.2*degree, 0.0*degree)
35 simulation.setSample(multi_layer)
36 simulation.runSimulation()
```

The first stage is to define the `Simulation()` object (line 31). Then we define the detector (line 33) and beam parameters (line 34) to finally run the simulation using the sample previously defined (line 35). Those functions are part of the Simulation class. The different incident and exit angles are shown in Fig. 2.

The detector parameters are set using ranges of angles via the function

```
setDetectorParameters(n_phi, phi_f_min, phi_f_max,
                      n_alpha, alpha_f_min, alpha_f_max, isgisaxs_style=false),
```

where `n_phi=100` is the number of points in the range of variations of angle ϕ_f,
`phi_f_min=-1.0*degree` and `phi_f_max=1.0*degree` are the minimum and maximum values respectively of ϕ_f, which is the in-plane direction of the scattering beam (measured with respect to the x-axis),
`n_alpha=100` is the number of points in the range of variations of the exit angle α_f measured from the x-axis in the z-direction,
`alpha_f_min=0.0*degree` and `alpha_f_max=2.0*degree` are the minimum and maximum values respectively of α_f,
`isgisaxs_style=True` (default value = False) is a boolean used to characterise the structure of the output data. If `isgisaxs_style=True`, **the output data is binned at constant values of the sine of**

α_f and ϕ_f otherwise it is binned at constant values of these two angles.

For the beam the function is `simulation.setBeamParameters(lambda, alpha_i, phi_i)`, where `lambda=1.0*angstrom` is the incident beam wavelength,
`alpha_i=-0.2*degree` is the incident grazing angle on the surface of the sample, `phi_i=0.0*degree` is the in-plane direction of the incident beam (measured with respect to the x-axis).

Remark: Note that, except for `isgisaxs_style`, there are no default values implemented for the parameters of the beam and detector.

Line 36 shows the command to run the simulation using the previously defined setup.

Fifth step: Saving the data

```
37 # retrieving intensity data
38 arr = GetOutputData(simulation)
```

In line 38 we record the simulated intensity as a function of outgoing angles α_f and ϕ_f for further uses (plots, fits,...) as a NumPy array containing `n_phi×n_alpha` datapoints. Some options are provided by BornAgain. For example, figure 3 shows the two-dimensional contourplot of the intensity as a function of α_f and ϕ_f.

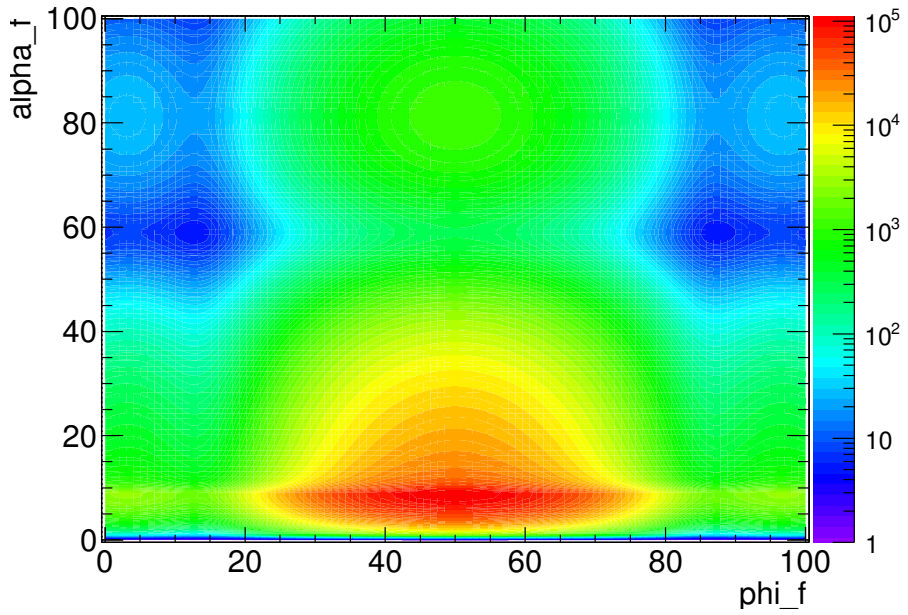

Figure 3: Figure of example 1: Simulated grazing-incidence small-angle X-ray scattering from a mixture of cylindrical and prismatic nanoparticles without any interference, deposited on top of a substrate. The input beam is characterized by a wavelength λ equal to 1 Å and incident angles $\alpha_i = -0.2°$, $\phi_i = 0°$. The cylinders have a radius and a height both equal to 5 nm, the prisms are characterized by a side length equal to 5 nm and they are also 5 nm high. The material of the particles has a refractive index of $1-6\times10^{-4}-i2\times10^{-8}$. For the substrate it is equal to $1-6\times10^{-6}-i2\times10^{-8}$. The colorscale is associated with the output intensity.

```
import sys
import os
import numpy

sys.path.append(os.path.abspath(os.path.join(os.path.split(__file__)[0],'..', '..', '..', 'lib
    ')))

from libBornAgainCore import *

    #  defining materials
    mAmbience = MaterialManager.getHomogeneousMaterial("Air", 1.0, 0.0 )
    mSubstrate = MaterialManager.getHomogeneousMaterial("Substrate", 1.0-6e-6, 2e-8)
    # collection of particles
    n_particle = complex(1.0-6e-4, 2e-8)
    cylinder_ff = FormFactorCylinder(5*nanometer, 5*nanometer)
    cylinder = Particle(n_particle, cylinder_ff)
```

```
prism_ff = FormFactorPrism3(5*nanometer, 5*nanometer)
prism = Particle(n_particle, prism_ff)
particle_decoration = ParticleDecoration()
particle_decoration.addParticle(cylinder, 0.0, 0.5)
particle_decoration.addParticle(prism, 0.0, 0.5)
interference = InterferenceFunctionNone()
particle_decoration.addInterferenceFunction(interference)
# air layer with particles and substrate form multi layer
air_layer = Layer(mAmbience)
air_layer_decorator = LayerDecorator(air_layer, particle_decoration)
substrate_layer = Layer(mSubstrate, 0)
multi_layer = MultiLayer()
multi_layer.addLayer(air_layer_decorator)
multi_layer.addLayer(substrate_layer)

# run simulation
simulation = Simulation()
simulation.setDetectorParameters(100,-1.0*degree, 1.0*degree,
                                 100, 0.0*degree, 2.0*degree, True)
simulation.setBeamParameters(1.0*angstrom, -0.2*degree, 0.0*degree)
simulation.setSample(multi_layer)
simulation.runSimulation()

# retrieving intensity data
 arr = GetOutputData(simulation)
```

Listing 1: Python script of example 1

0.9 Example 2

Bugs
License agreement
Directory layout
FAQ
Future development.

Bibliography

[1] Rémi Lazzari. *IsGISAXS*: a program for grazing-incidence small-angle X-ray scattering analysis of supported islands. *Journal of Applied Crystallography*, 35(4):406–421, Aug 2002.

[2] Rémi Lazzari. *IsGISAXS*: a program for grazing-incidence small-angle X-ray scattering analysis of supported islands.

[3] Kyle Loudon. *C++ pocket reference*. O'Reilly media, 2008.

[4] Mark Lutz. *Python pocket reference*. O'Reilly media, fourth edition edition, 2009.