HDRI WP3:
Status of GISAS software development

Céline Durniak, Walter Van Herck, Gennady Pospelov, Joachim Wuttke

Scientific Computing Group at FRM-II
Jülich Center for Neutron Science

Computing in Photon and Neutron Science Applications
4-5 March, 2013, DESY, Hamburg
Outline

Introduction
  o Work Package 3
  o Scientific Computing Group at FRM-II

Development of software for GISAS
  o Strategy
  o Current status, available geometries, usage example
  o Infrastructure for code development and software management

Conclusion
Work package 3

Data Analysis, Modeling and Simulation
- Coordination: JCNS Th. Brückel
- Partners: DESY, HZG, HZB

Project goals
- Visualization and standard analysis for SANS/SAXS
- Simulation and analysis software for off-specular scattering and GISAS

Strategy
- Development of example application for simulation and analysis of Grazing Incidence Small Angle Scattering
- Extend the supplied analysis and visualization tool box to other experiments
Scientific Computing group at FRM II

Organization
- A joint group of HGF and TUM

Staff
- Joachim Wuttke, nominated July 2011
- Christian Felder, since August 2011 (also admin duties)
- Walter Van Herck, since Januar 2012
- Gennady Pospelov, since Januar 2012
- Céline Durniak, since Januar 2013

Scope of responsibility
- Supporting all scattering instruments at FRM II
- With focus on simulation and data analysis
- Leaving data acquisition in firm hands of instrument responsible

Current main project
- Development of reference software for x-ray, neutron experiments at grazing incidence
Grazing Incidence Small Angle Scattering

Experimental setup
- monochromatic beam is directed on a surface with a very small incident angle
- any kind of deviation from perfectly smooth layered system leads to beam scattered in an out-specular direction
- 2D detector records the intensity of scattered wave vector giving access to lateral and vertical sample structure information

Distorted Wave Born Approximation (DWBA)
- accounts for reflection-refraction effects close to the critical angle
- interference between scattering from direct and reflected beam via

\[
\mathcal{F}^{i}_{\text{DWBA}}(\mathbf{k}_i, \mathbf{k}_f, R_z) = T_i T_f F^i (\mathbf{k}_i - \mathbf{k}_f) e^{i(k_{iz}-k_{fz})R_z} + R_i T_f F^i (\mathbf{k}_i - \mathbf{k}_f) e^{i(-k_{iz}-k_{fz})R_z} \\
+ T_i R_f F^i (\mathbf{k}_i - \mathbf{k}_f) e^{i(k_{iz}+k_{fz})R_z} + R_i R_f F^i (\mathbf{k}_i - \mathbf{k}_f) e^{i(-k_{iz}+k_{fz})R_z},
\]
SOFTWARE

Available Software

Software dedicated to the simulation and analysis of GISAS from nanostructures

- X-rays: IsGISAXS, FitGisaxs, Logos, GenX, Imd/Windt, HipGISAXS (new and yet unstudied)
- Neutrons: Effi/FitSuite

Summing up typical problems of existing software

- have insufficient functionality
- rely on proprietary environment
- platform dependent
- no longer maintained
- often developed by one person
- not extendable
- small or absent user community
Strategy

- Reproduce functionality of IsGISAXS using modern programming approaches
  - most known in the field, largest user community
  - no longer maintained

- Extend it with most demanded features
  - multilayer support, roughness correlation
  - different types of nanoparticles in one sample
  - polarized neutrons and magnetic domains

- Build user community and follow their demands

What is IsGISAXS?
- X-ray scattering at grazing incidence from nanostructures
- Author: Remi Lazzari
- FORTRAN90, ~13k lines of code
- Windows (with GUI)
- Linux (without GUI)

Rules to follow
- decouple physical modeling / data fitting / GUI
- dependence on open source and actively maintained libraries
- produce open source code
- platform independence (Linux, Mac, Windows)
- write mathematics before coding
The name of the software, BornAgain, indicates the central role of the Distorted Wave Born Approximation

- generic frame for modeling multi layer samples with smooth or rough interfaces
- with various types of embedded nano particles
- grazing incidence geometry
- fit of experimental data using variety of external minimization engines

- **Core library**
  - programming language C++
  - 25k lines of code
  - Python interface
  - externals: boost, GNU Scientific Library
  - Mac/Linux, Windows soon

- **Graphical user interface**
  - at early prototyping stage
  - Qt, ROOT libraries
How to use

**BornAgain** core library can be imported in Python or linked to C++ application

- python script demonstrates simulation of cylindrical nano particles on top of substrate

```python
from libBornAgainCore import *

# defining materials
ambience = HomogeneousMaterial("Air", 1.0, 0.0)
substrate = HomogeneousMaterial("Substrate", 1.0-6e-6, 2e-8)

# two empty layers
air_layer = Layer(ambience)
substrate_layer = Layer(substrate)

# creating particles
cylinder_ff = FormFactorPyramid(5*nm, 5*nm, 54.73*deg)
nParticle = complex(1.0-5e-5, 2e-8)
cylinder = Particle(nParticle, cylinder_ff)

# additional properties: transformation
angle_around_z = 45.*degree
transform = RotateZ3D(angle_around_z)

# additional properties: interference function
interference = InterferenceFunction1DParaCrystal(20*nm, 7*nm, 1e7*nm)

# decorating air layer with particles
particle_decoration = ParticleDecoration()
particle_decoration.addParticle(cylinder, transform)
particle_decoration.addInterferenceFunction(interference)
decorated_layer = LayerDecorator(air_layer, particle_decoration)

# creating multilayer
multi_layer = Multilayer()
multi_layer.addLayer(decorated_layer)
multi_layer.addLayer(substrate_layer)

# simulating
simulation = Simulation()
simulation.setDetector(100, 0.0*deg, 2.0*deg, 100, 0.0*deg, 2.0*deg)
simulation.setBeamParameters(1.0*angstrom, -0.2*deg, 0.0*deg)
simulation.setSample(multi_layer)
simulation.runSimulation()

# retrieving data
arr = GetOutputData(simulation)
```
How to use

```python
from libBornAgainCore import *

# defining materials
ambience = HomogeneousMaterial("Air", 1.0, 0.0)
substrate = HomogeneousMaterial("Substrate", 1.0-6e-6, 2e-8)

# two empty layers
air_layer = Layer(ambience)
substrate_layer = Layer(substrate)

# creating particles
cylinder_ff = FormFactorPyramid(5*nm, 5*nm, 54.73*deg)
n_particle = complex(1.0-5e-5, 2e-8)
cylinder = Particle(n_particle, cylinder_ff)

# additional properties: transformation
angle_around_z = 45.*degree
transform = RotateZ3D(angle_around_z)

# additional properties: interference function
interference = InterferenceFunction1DParaCrystal(20*nm, 7*nm, 1e7*nm)

# decorating air layer with particles
particle_decoration = ParticleDecoration()
particle_decoration.addParticle(cylinder, transform)
particle_decoration.addInterferenceFunction(interference)
decorated_layer = LayerDecorator(air_layer, particle_decoration)

# creating multilayer
multi_layer = MultiLayer()
multi_layer.addLayer(decorated_layer)
multi_layer.addLayer(substrate_layer)

# simulating
simulation = Simulation()
simulation.setDetector(100, 0.0*deg, 2.0*deg, 100, 0.0*deg, 2.0*deg)
simulation.setBeamParameters(1.0*angstrom, -0.2*deg, 0.0*deg)
simulation.setSample(multi_layer)
simulation.runSimulation()

# retrieving data
arr = GetOutputData(simulation)
```
How to use

```
from libBornAgainCore import *

# defining materials
ambience = HomogeneousMaterial("Air", 1.0, 0.0)
substrate = HomogeneousMaterial("Substrate", 1.0-6e-6, 2e-8)

# two empty layers
air_layer = Layer(ambience)
substrate_layer = Layer(substrate)

# creating particles
cylinder_ff = FormFactorPyramid(5*nm, 5*nm, 54.73*deg)
n_particle = complex(1.0-5e-5, 2e-8)
cylinder = Particle(n_particle, cylinder_ff)

# additional properties: transformation
angle_around_z = 45.*degree
transform = RotateZ3D(angle_around_z)

# additional properties: interference function
interference = InterferenceFunction1DParaCrystal(20*nm, 7*nm, 1e7*nm)

# decorating air layer with particles
particle_decoration = ParticleDecoration()
particle_decoration.addParticle(cylinder, transform)
particle_decoration.addInterferenceFunction(interference)
decorated_layer = LayerDecorator(air_layer, particle_decoration)

# creating multilayer
multi_layer = MultiLayer()
multi_layer.addLayer(decorated_layer)
multi_layer.addLayer(substrate_layer)

# simulating
simulation = Simulation()
simulation.setDetector(100, 0.0*deg, 2.0*deg, 100, 0.0*deg, 2.0*deg)
simulation.setBeamParameters(1.0*angstrom, -0.2*deg, 0.0*deg)
simulation.setSample(multi_layer)
simulation.runSimulation()

# retrieving data
arr = GetOutputData(simulation)
```
How to use

Nano particle is defined by the refractive index and form factor

\[
F_{yy}(q, R, H, \alpha) = \int_{-H}^{H} 4R^2 \sin(q_y R) \sin(q_z R) \exp(iq_z z) \, dz
\]
\[
R_y = R - z / \tan(\alpha)
\]
\[
H / R < \tan(\alpha)
\]
\[
V_{yy} = \frac{4}{3} \tan(\alpha) \left[ R^3 - \left( R - \frac{H}{\tan(\alpha)} \right)^3 \right], \quad S_{yy} = 4R^2, \quad R_{yy} = \sqrt{2}R
\]
\[
F_{yy}(q, R, H, \alpha) = \frac{H}{q_y q_z} \times
\]
\[
\{ \cos(q_z - q_y)R \lVert \ket{K_1} + \sin(q_z - q_y)R \lVert \ket{K_2} - \cos(q_z + q_y)R \lVert \ket{K_3} - \sin(q_z + q_y)R \lVert \ket{K_4} \}
\]
\[
K_1 = \sin(q_y R) \exp(iq_y H) + \sin(q_z R) \exp(-iq_y H)
\]
\[
K_2 = \sin(q_y R) \exp(iq_y H) - \sin(q_z R) \exp(-iq_y H)
\]
\[
K_3 = \sin(q_z R) \exp(iq_z H) + \sin(q_y R) \exp(-iq_z H)
\]
\[
K_4 = \sin(q_z R) \exp(iq_z H) - \sin(q_y R) \exp(-iq_z H)
\]
\[
q_1 = \frac{1}{2} \frac{q_z - q_y}{\tan(\alpha)} + q_y, \quad q_2 = \frac{1}{2} \frac{q_z - q_y}{\tan(\alpha)} - q_y
\]
\[
q_3 = \frac{1}{2} \frac{q_z + q_y}{\tan(\alpha)} + q_y, \quad q_4 = \frac{1}{2} \frac{q_z + q_y}{\tan(\alpha)} - q_y
\]

Additional transformation can be defined for orientation and position of the particle

from libBornAgainCore import *

# defining materials
ambience = HomogeneousMaterial("Air", 1.0, 0.0 )
substrate = HomogeneousMaterial("Substrate", 1.0-6e-6, 2e-8 )

# two empty layers
air_layer = Layer(ambience)
substrate_layer = Layer(substrate)

# creating particles
cylinder_ff = FormFactorPyramid(5*nm, 5*nm, 54.73*deg)
n_particle = complex(1.0-5e-5, 2e-8)
cylinder = Particle(n_particle, cylinder_ff)

# additional properties: transformation
angle_around_z = 45.*degree
transform = RotateZ3D(angle_around_z)

# additional properties: interference function
interference = InterferenceFunction1DParaCrystal(20*nm, 7*nm, 1e7*nm)

# decorating air layer with particles
particle_dec = ParticleDecoration()
particle_dec.addParticle(cylinder, transform)
particle_dec.addInterferenceFunction(interference)
decorated_layer = LayerDecorator(air_layer, particle_dec)

# creating multilayer
multi_layer = Multilayer()
multi_layer.addLayer(decorated_layer)
multi_layer.addLayer(substrate_layer)

# simulating
simulation = Simulation()
simulation.setDetector(100, 0.0*deg, 2.0*deg, 100, 0.0*deg, 2.0*deg)
simulation.setBeamParameters(1.0*angstrom, -0.2*deg, 0.0*deg)
simulation.setSample(multi_layer)
simulation.runSimulation()

# retrieving data
arr = GetOutputData(simulation)
How to use

**InterferenceFunction1DParaCrystal**
defines interference function of regular lattice with long range order gradually destroyed

```
from libBornAgainCore import *
# defining materials
ambience = HomogeneousMaterial("Air", 1.0, 0.0)
substrate = HomogeneousMaterial("Substrate", 1.0-6e-6, 2e-8)
# two empty layers
air_layer = Layer(ambience)
substrate_layer = Layer(substrate)
# creating particles
cylinder_ff = FormFactorPyramid(5*nm, 5*nm, 54.73*deg)
n_particle = complex(1.0-5e-5, 2e-8)
cylinder = Particle(n_particle, cylinder_ff)
# additional properties: transformation
angle_around_z = 45.*degree
transform = RotateZ3D(angle_around_z)
# additional properties: interference function
interference = InterferenceFunction1DParaCrystal(20*nm, 7*nm, 1e7*nm)
# decorating air layer with particles
particle_decoration = ParticleDecoration()
particle_decoration.addParticle(cylinder, transform)
particle_decoration.addInterferenceFunction(interference)
decorated_layer = LayerDecorator(air_layer, particle_decoration)
# creating multilayer
multi_layer = MultiLayer()
multi_layer.addLayer(decorated_layer)
multi_layer.addLayer(substrate_layer)
# simulating
simulation = Simulation()
simulation.setDetector(100, 0.0*deg, 2.0*deg, 100, 0.0*deg, 2.0*deg)
simulation.setBeamParameters(1.0*angstrom, -0.2*deg, 0.0*deg)
simulation.setSample(multi_layer)
simulation.runSimulation()
# retrieving data
arr = GetOutputData(simulation)
```

**ParticleDecoration**
Holds information about particle types, positions and interference

**LayerDecorator**
Decorates air layer with particle decoration
How to use

**MultiLayer**

- owns decorated air layer and substrate layer

**Simulation**

- Holds beam and detector parameters, accept multi layer as an input and perform calculations

```python
from libBornAgainCore import *

# defining materials
ambience = HomogeneousMaterial("Air", 1.0, 0.0)
substrate = HomogeneousMaterial("Substrate", 1.0-6e-6, 2e-8)

# two empty layers
air_layer = Layer(ambience)
substrate_layer = Layer(substrate)

# creating particles
cylinder_ff = FormFactorPyramid(5*nm, 5*nm, 54.73*deg)
n_particle = complex(1.0-5e-5, 2e-8)
cylinder = Particle(n_particle, cylinder_ff)

# additional properties: transformation
angle_around_z = 45.*degree
transform = RotateZ3D(angle_around_z)

# additional properties: interference function
interference = InterferenceFunction1DParaCrystal(20*nm, 7*nm, 1e7*nm)

# decorating air layer with particles
particle-decoration = ParticleDecoration()
particle-decoration.addParticle(cylinder, transform)
particle-decoration.addInterferenceFunction(interference)
decorated_layer = LayerDecorator(air_layer, particle-decoration)

# creating multilayer
multi_layer = MultiLayer()
multi_layer.addLayer(decorated_layer)
multi_layer.addLayer(substrate_layer)

# simulating
simulation = Simulation()
simulation.setDetector(100, 0.0*deg, 2.0*deg, 100, 0.0*deg, 2.0*deg)
simulation.setBeamParameters(1.0*angstrom, -0.2*deg, 0.0*deg)
simulation.setSample(multi_layer)
simulation.runSimulation()

# retrieving data
arr = GetOutputData(simulation)
```
How to use

OutputData
Simulated intensity as a function of outgoing $\alpha_f$, $\phi_f$ angles can be retrieved as NumPy array

```python
from libBornAgainCore import *

# defining materials
ambience = HomogeneousMaterial("Air", 1.0, 0.0)
substrate = HomogeneousMaterial("Substrate", 1.0-6e-6, 2e-8)

# two empty layers
air_layer = Layer(ambience)
substrate_layer = Layer(substrate)

# creating particles
cylinder_ff = FormFactorPyramid(5*nm, 5*nm, 54.73*deg)
n_particle = complex(1.0-5e-5, 2e-8)
cylinder = Particle(n_particle, cylinder_ff)

# additional properties: transformation
angle_around_z = 45.*degree
transform = RotateZ3D(angle_around_z)

# additional properties: interference function
interference = InterferenceFunction1DParaCrystal(20*nm, 7*nm, 1e7*nm)

# decorating air layer with particles
particle_decoration = ParticleDecoration()
particle_decoration.addParticle(cylinder, transform)
particle_decoration.addInterferenceFunction(interference)
decorated_layer = LayerDecorator(air_layer, particle_decoration)

# creating multilayer
multi_layer = MultiLayer()
multi_layer.addLayer(decorated_layer)
multi_layer.addLayer(substrate_layer)

# simulating
simulation = Simulation()
simulation.setDetector(100, 0.0*deg, 2.0*deg, 100, 0.0*deg, 2.0*deg)
simulation.setBeamParameters(1.0*angstrom, -0.2*deg, 0.0*deg)
simulation.setSample(multi_layer)
simulation.runSimulation()

# retrieving data
arr = GetOutputData(simulation)
```
Available scattering geometries

IsGISAXS functionality is mostly covered
- IsGISAXS has 15 examples in package bundle, we can reproduce 14 of them
  - islands on substrate, particles encapsulated in a layer in DWBA
  - account between positions and size of particles (LMA, SSCA approximations)
  - disordered systems of particles, bidimensional crystals and paracrystals

What is uncovered
- few form factors and pair correlation functions
- DWBA on the graded interface
- is planned within two months from now

Extra functionality available up to now
- diffuse scattering from rough layers and multi-layers with correlated roughness
- no restriction on number of layers and particle types
- particles with inner structure (mesocrystals)
Available scattering geometries

Grazing incidence small-angle neutron scattering from a mesocrystalline system compared to a BornAgain simulation
- Mesocrystals have cylindrical shape and size of ~1000nm
- Mesocrystal consist of an FCC lattice composed of 5 nm spherical particles

Experimental data courtesy Elisabeth Josten et al

BornAgain simulation in collaboration with Artur Glavic and Elisabeth Josten

Results of 11 parameter fit
The work on graphical user interface has just started.
- current prototypes rely on Qt and ROOT libraries
- BornAgainCore library used as a plugin

We have in plans
- drag and drop sample editor
- interactive plots
- access to large collection of minimizers
- Python and C++ scripting
Development process

- More agile development
- Quality control
- Good documentation

BornAgain git repository: number of lines of code vs time
Development process

- More agile development
- Quality assurance
- Good documentation

Project are managed via Redmine management tool
- Work organization, user request handling, bug tracking tool, issue tracking
- Workflow consist of sprint cycles every 7-21 days during which the team create finished portions of product
Development process

- More agile development
- Quality assurance
- Good documentation

Unit test control
- small test cases are executed after each compilation to ensure code stability

Functional tests
- higher level tests produces scattering plots which are automatically analyzed
- special tests compare BornAgain results with IsGISAXS results
Development process

- More agile development
- Quality assurance
- Good documentation

Nightly build system
- Based on TeamCity, one build configuration so far

Code profiling
- Regular tracking of memory and CPU consumption
Development process

- More agile development
- Quality assurance
- Good documentation

Code documentation
- Doxygen generated, starting to write user manual

Theory and conventions
- described and stored in the repository parallel to the code development

Scattering on nanoparticles
Scientific Computing Group at FRM II
February 6, 2013

1 Formal treatment
1.1 Green operators and the \( T \)-matrix

For a particle, governed by the Schrödinger equation with Hamiltonian

\[ H = H_0 + V, \]

the time-independent scattering theory formally consists of solving the eigenvalue equations:

\[ H \psi_n = E \psi_n, \]

where \( E \) is the energy eigenvalue of the eigenstate \( \psi_n \). If the solutions of the free (or unperturbed) Hamiltonian \( H_0 \) are known:

\[ H_0 \psi_{n0} = E_{0n} \psi_{n0}, \]

one can write the solutions of the full Hamiltonian in terms of these asymptotic states and Green operators:

\[ \psi_n = \psi_{n0} + G^\dagger V \psi_{n0}, \]

where the Green operators are defined as:

\[ G^\dagger(E) = (E - H_0 \pm i \epsilon)^{-1}, \]

\[ G^\dagger(E) = (E - H \pm i \epsilon)^{-1}. \]

In these equations, the upper index or sign refers to the state corresponding with the free state \( \psi_{n0} \) at time \( t \to \infty \) (and vice versa for the lower sign). Since the solutions of the eigenvalue equations, both for the unperturbed as for the full Hamiltonian, are dependent on the energy eigenvalues \( E \), the index \( n \) is assumed to include this value (and possibly other quantum numbers).

The transition amplitude between two asymptotic states is given by the \( S \)-matrix elements, defined as:

\[ S_{ij} = \langle \psi_i^\dagger | \psi_j \rangle, \]
Conclusion and plans

- Born Again is a new software for simulating and fitting grazing incidence small angle scattering
- It almost reproduces and enhances the functionality of the present reference software, IsGISAXS
- The first major extension, planned for 2013/14, will provide coverage of polarized GISANS for magnetic domains studies
- Scientific Computing Group at FRMII organizes workshop on Grazing Incidence Small Angle Scattering software, 9-10 April
  
  [Link](http://apps.jcns.fz-juelich.de/doku/sc/giss_2013)

- There we will be happy to discuss with expert users challenges for the modeling and data analysis of GISAS experiment and make BornAgain open
Leptos Diffrac from Bruker

Disadvantages:
- Availability for users (price)
- Closed software
- Non extendable

<table>
<thead>
<tr>
<th>Analytical Tasks</th>
<th>XRR</th>
<th>HRXRD</th>
<th>GIXRD²</th>
<th>DS³</th>
<th>GISAXS</th>
<th>STRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer thickness (amorphous)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Layer thickness (crystalline)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Layer mass density</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chemical composition</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Roughness</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vertical correlations</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lateral correlations</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Porosity, Nano-objects size and distribution</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lattice Parameters</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strain, Relaxation, Mismatch</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Defects &amp; Crystal Structure</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mosaicity</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Residual stress (normal, shear, biaxial, triaxial)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>