

# BornAgain Python API

Gennady Pospelov  
Jülich Centre for Neutron Science at MLZ

BornAgain school and user meeting  
Garching, December 2018

day\_2

# Outline

~/BornAgain-tutorial/talks/day\_2/python\_api\_G

Introduction to Python API	talking	pyapi01_introduction.pdf
Minimal simulation example	demo	pyapi02_minimal_example.py pyapi02_jupyter_example.ipynb
Simulating lamellar structure	demo	pyapi03_lamellar.py
Modifying lamellar example	task	pyapi04_lamellar_vertical_solution1.py pyapi04_lamellar_vertical_solution2.py
Introduction to fitting	talking	pyapi05_fitting.pdf

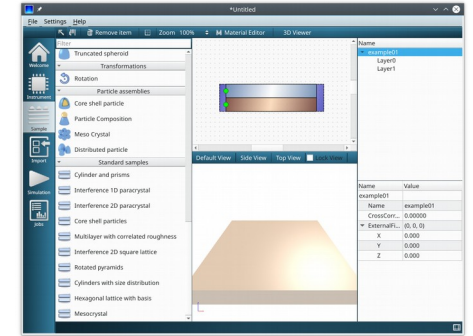
# BornAgain Python API

Application Programming Interface (API) can be used to build and run simulations.

## Advantages

- Complex sample construction
- Access to features non existing in GUI
- Possibility to reuse code in other projects
- Less back-compatibility problems
- Git based workflow

# Minimal simulation example



GUI equivalent

```
from bornagain import *

# --- sample ---

air = Layer(HomogeneousMaterial("Air", 0.0, 0.0))
substrate = Layer(HomogeneousMaterial("Substrate", 6e-6, 2e-8))

multi_layer = MultiLayer()
multi_layer.addLayer(air)
multi_layer.addLayer(substrate)

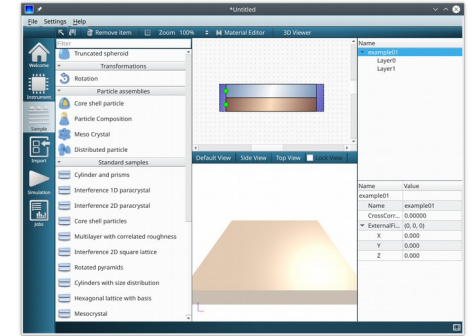
# --- simulation ---

simulation = GISASSimulation()
simulation.setDetectorParameters(100, -2.0*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)

# --- run and plot ---

simulation.runSimulation()
plot_simulation_result(simulation.result())
```

# Minimal simulation example



GUI equivalent

```
from bornagain import *

# --- sample ---

air = Layer(HomogeneousMaterial("Air", 0.0, 0.0))
substrate = Layer(HomogeneousMaterial("Substrate", 6e-6, 2e-8))

multi_layer = MultiLayer()
multi_layer.addLayer(air)
multi_layer.addLayer(substrate)

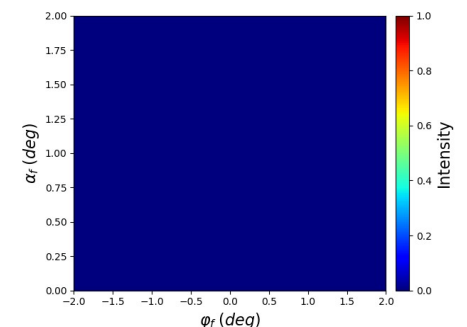
# --- simulation ---

simulation = GISASSimulation()
simulation.setDetectorParameters(100, -2.0*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)

# --- run and plot ---

simulation.runSimulation()
plot_simulation_result(simulation.result())
```

```
$ python pyapi02_minimal_example.py
```



# Demo

Running trivial simulations

`pyapi02_minimal_example.py`

`pyapi02_jupyter_example.py`

# Structuring the code

```
import bornagain as ba

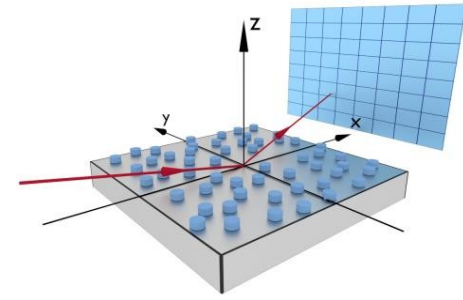
def get_sample():
    multi_layer = ba.MultiLayer()
    ...
    return multi_layer

def get_simulation():
    simulation = ba.GISASSimulation()
    ...
    sample = get_sample()
    simulation.setSample(sample)
    return simulation

def run_simulation():
    simulation = get_simulation()
    simulation.runSimulation()
    return simulation.result()

if __name__ == '__main__':
    result = run_simulation()
    ba.plot_simulation_result(result)
```

# Example on sample construction



```
import bornagain as ba
```

```
def get_sample():
```

```
    # defining materials
```

```
    air = ba.HomogeneousMaterial("Air", 0.0, 0.0)
```

```
    substrate = ba.HomogeneousMaterial("Substrate", 6e-6, 2e-8)
```

```
    gold = ba.HomogeneousMaterial("Gold", 6e-4, 2e-8)
```

```
    # creating particles
```

```
    cylinder_ff = ba.FormFactorCylinder(5*nm, 5*nm)
```

```
    cylinder = ba.Particle(gold, cylinder_ff)
```

```
    layout = ba.ParticleLayout()
```

```
    layout.addParticle(cylinder, 1.0)
```

```
    air_layer = ba.Layer(air)
```

```
    air_layer.addLayout(layout)
```

```
    substrate_layer = ba.Layer(substrate)
```

```
    multi_layer = ba.MultiLayer()
```

```
    multi_layer.addLayer(air_layer)
```

```
    multi_layer.addLayer(substrate_layer)
```

```
    return multi_layer
```

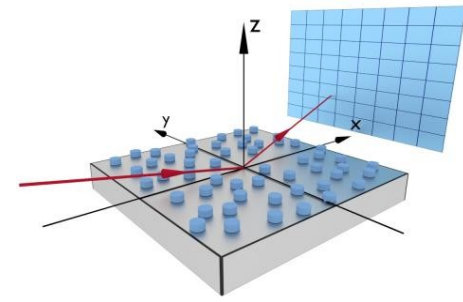
Material definition

Particle collection

MultiLayer construction



# Example on sample construction



```
import bornagain as ba
```

```
def get_materials():  
    # defining materials  
    air = ba.HomogeneousMaterial("Air", 0.0, 0.0)  
    substrate = ba.HomogeneousMaterial("Substrate", 6e-6, 2e-8)  
    gold = ba.HomogeneousMaterial("Gold", 6e-4, 2e-8)  
    return air, substrate, gold
```

Material definition

```
def get_sample():  
    air, substrate, gold = get_materials()  
  
    # creating particles  
    cylinder_ff = ba.FormFactorCylinder(5*nm, 5*nm)  
    cylinder = ba.Particle(gold, cylinder_ff)
```

Particle collection

```
    layout = ba.ParticleLayout()  
    layout.addParticle(cylinder, 1.0)
```

```
    air_layer = ba.Layer(air)  
    air_layer.addLayout(layout)  
    substrate_layer = ba.Layer(substrate)
```

MultiLayer construction

```
    multi_layer = ba.MultiLayer()  
    multi_layer.addLayer(air_layer)  
    multi_layer.addLayer(substrate_layer)  
    return multi_layer
```

# Where to start

## Python viewer embedded in GUI

The screenshot shows a software interface for particle simulation, titled '\*Untitled'. The interface is divided into several panels:


- Left Sidebar:** Contains icons and labels for 'Welcome', 'Instrument', 'Sample', 'Import', 'Simulation', and 'Jobs'. Below these are expandable sections for 'Transformations' (Rotation), 'Particle assemblies' (Core shell particle, Particle Composition, Meso Crystal, Distributed particle), and 'Standard samples' (Cylinder and prisms, Interference 1D paracrystal, Interference 2D paracrystal, Core shell particles).
- Main Workspace:** Displays a 3D visualization of a particle assembly. It includes a 'Particle0' and 'Particle1' box, a 'Particle Layout' box, and a 'transformation' box. The 'Particle Layout' box is connected to the 'Particle0' and 'Particle1' boxes. The 'transformation' box is connected to the 'Particle0' box.
- Bottom Panel:** Contains a Python code editor with the following code:


```
def get_sample():  
    # Defining Materials  
    material_1 = ba.HomogeneousMaterial("example01_Air", 0.0, 0.0)  
    material_2 = ba.HomogeneousMaterial("example01_Particle", 0.0006, 2e-08)  
    material_3 = ba.HomogeneousMaterial("example01_Substrate", 6e-06, 2e-08)  
  
    # Defining Layers  
    layer_1 = ba.Layer(material_1)  
    layer_2 = ba.Layer(material_3)  
  
    # Defining Form Factors  
    formFactor_1 = ba.FormFactorCylinder(5.0*nm, 5.0*nm)  
    formFactor_2 = ba.FormFactorPrism3(10.0*nm, 5.0*nm)  
  
    # Defining Particles  
    particle_1 = ba.Particle(material_2, formFactor_1)  
    particle_2 = ba.Particle(material_2, formFactor_2)  
  
    # Defining Particle Layouts and adding Particles  
    layout_1 = ba.ParticleLayout()
```
- Right Sidebar:** Contains a hierarchical tree of the simulation components and a table of their properties.
  - Tree:** example01
    - Layer0
      - ParticleLayout
        - Particle0
        - Particle1
    - Layer1
  - Table:**

Name	Value
Particle0	
Form Fact...	Cylinder
Radius	5.000
Height	5.000
Material	example01_Par...
Abundance	0.500
Position ...	(0, 0, 0)
X	0.000
Y	0.000
Z	0.000

# Where to start

## Sample model reference on website

 BornAgain

Home Gallery Download News Documentation Contact About 

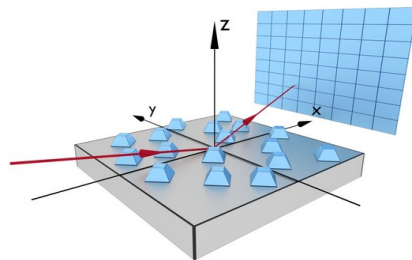
Home > Documentation > Sample model reference > Embedded particles > Rotated Pyramids

### Rotated Pyramids

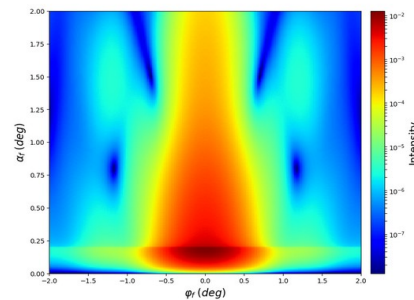
Scattering from a monodisperse distribution of rotated pyramids.

This example illustrates how the in-plane rotation of non-radially symmetric particles influences the scattering pattern.

- The sample is made of pyramids deposited on a substrate.
- Each pyramid is characterized by a squared-base side length of 10 nm, a height of 5 nm, and a base angle  $\alpha$  equal to  $54.73^\circ$ .
- These particles are rotated in the  $(x, y)$  plane by  $45^\circ$ .
- There is no interference between the scattered waves.
- The wavelength is equal to 1 Å.
- The incident angles are  $\alpha_i = 0.2^\circ$  and  $\phi_i = 0^\circ$ .



Real-space model



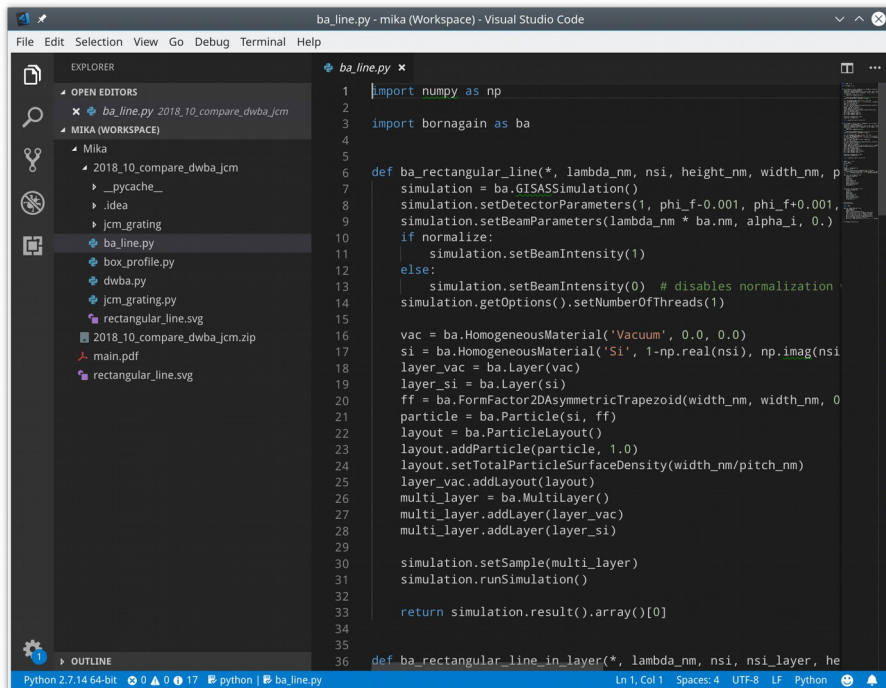
Intensity image

```
1  """
2  Rotated pyramids on top of substrate
3  """
4  import bornagain as ba
5  from bornagain import deg, angstrom, nm
6
7
8  def get_sample():
    ....
```

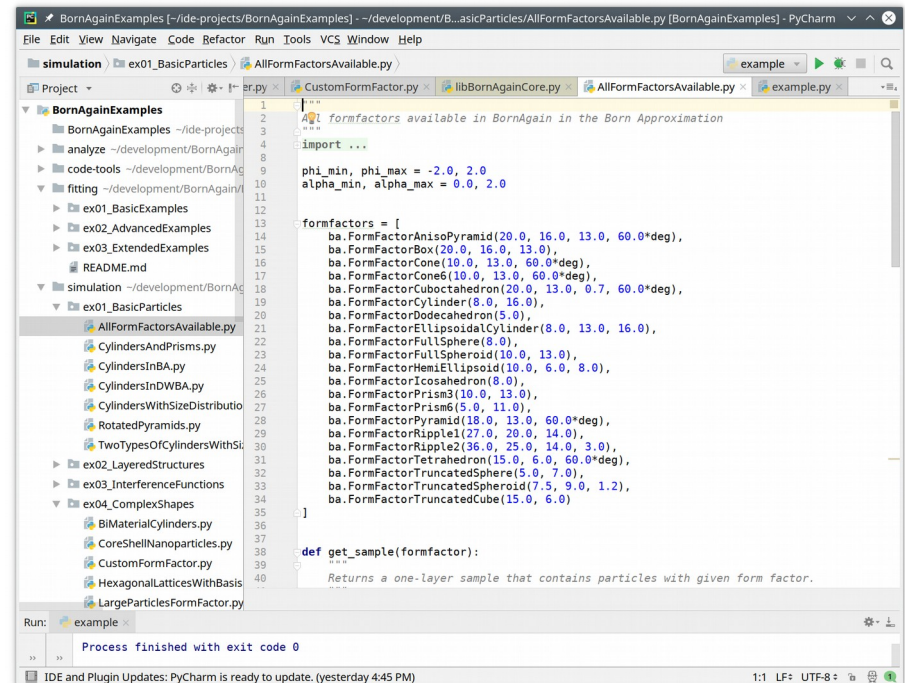
- [-] Documentation
  - [+] Introduction
  - [+] Getting started
  - [+] Using Graphical User Interface
  - [+] Working with Python scripts
  - [-] Sample model reference
    - [-] Embedded particles
      - Cylinders in Born Approximation
      - Cylinders in Distorted Wave Born Approximation
      - Cylinders with size distribution
      - Two types of cylinders with size distribution
      - Rotated Pyramids
      - Cylinders and Prisms
      - All available form factors
    - [+] Layered structures
    - [+] Interference functions
    - [+] Complex shapes
    - [+] Beam and detector
    - [+] Reflectometry
    - [+] Fitting
    - [+] Miscellaneous
  - [+] Getting help
  - [+] Developer's corner

# Where to code

- Editor of your choice, Python interpreter
- Integrated development environment (IDE)
  - MS Visual Studio Code, PyCharm
  - Syntax and error highlighting, debug, code navigation
- Jupyter notebooks
  - Quick prototyping



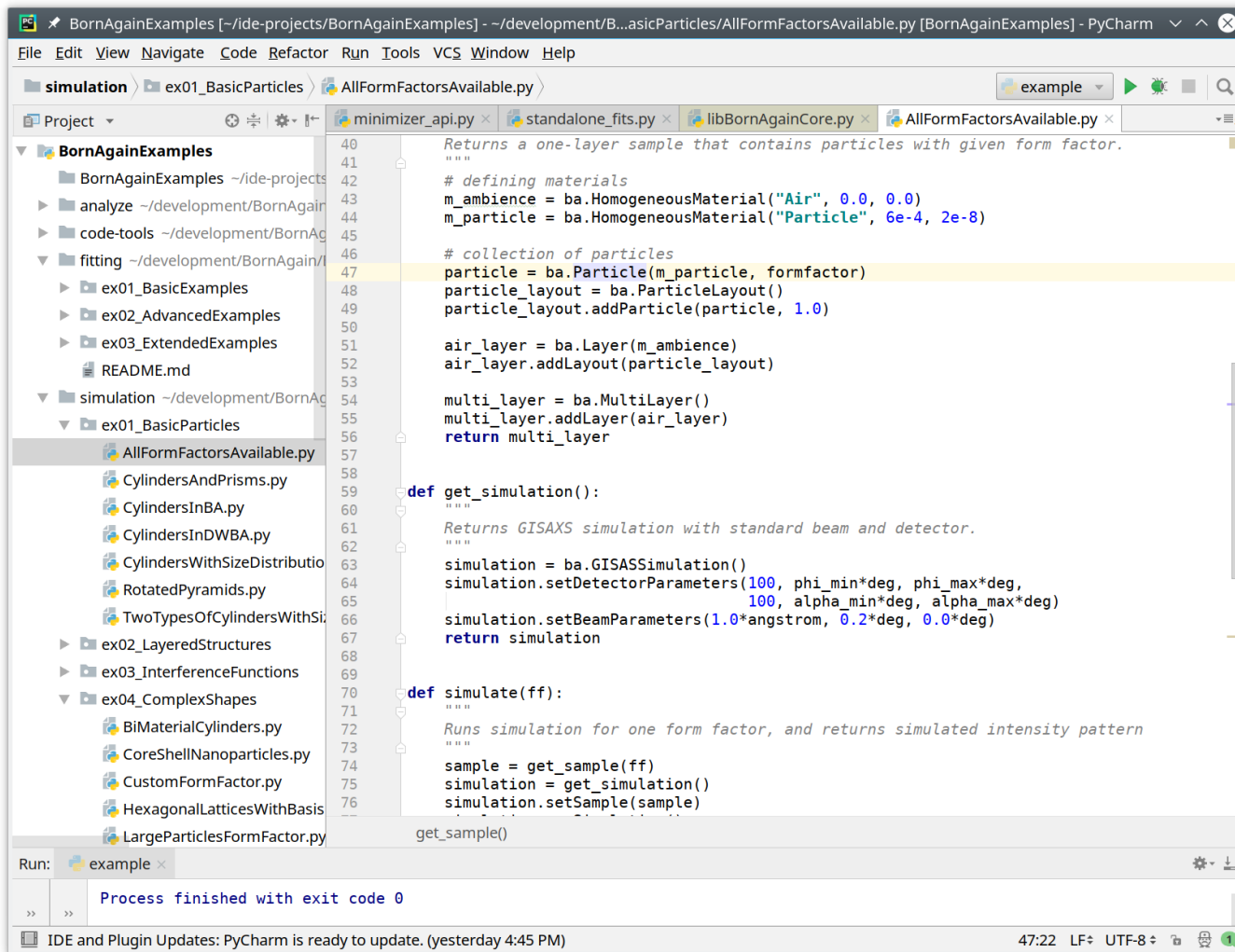
VS Code



PyCharm

# Python API technicalities

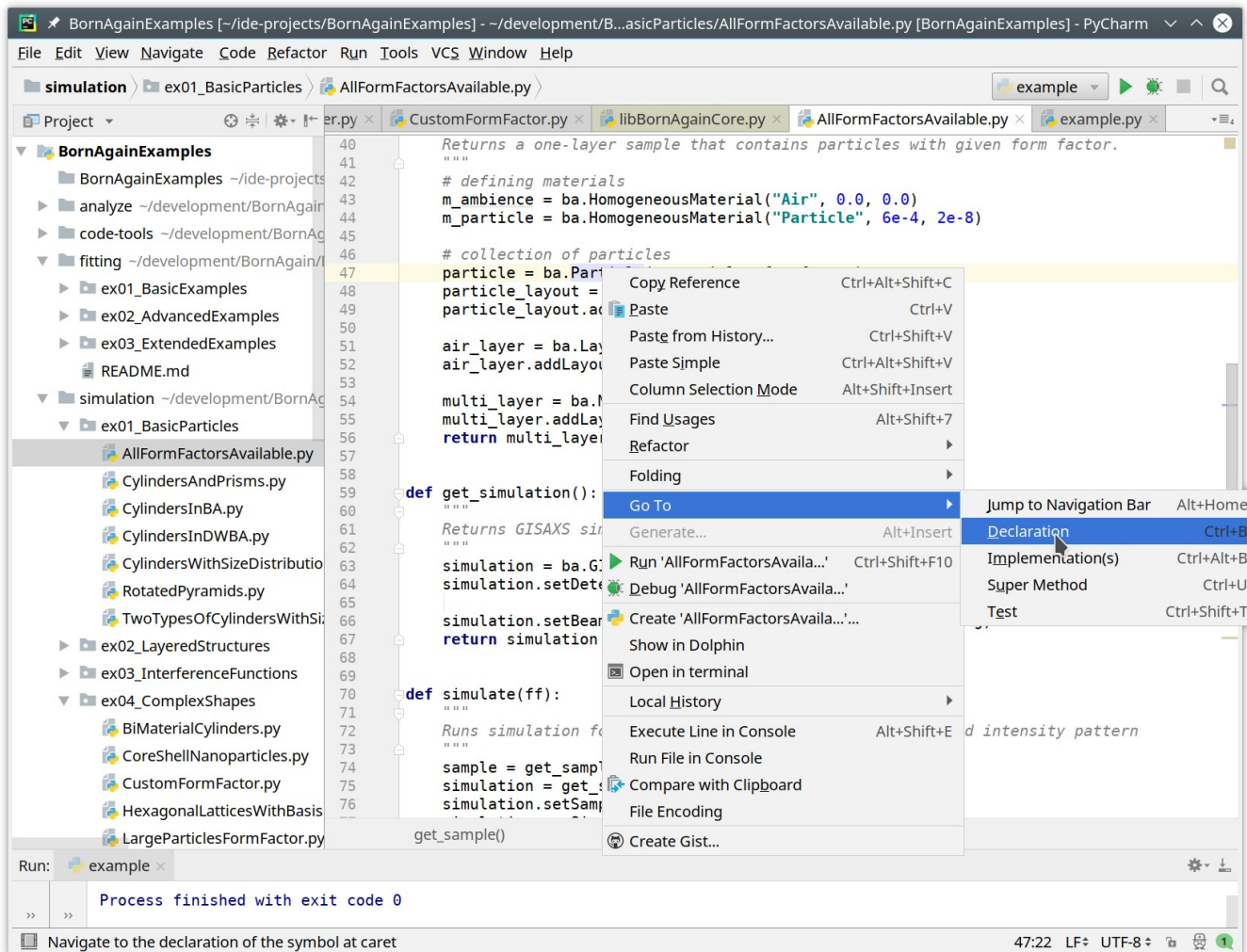
BornAgain Python API is automatically generated from C++ code





# Python API technicalities

BornAgain Python API is automatically generated from C++ code



# Python API technicalities


BornAgain Python API is automatically generated from C++ code


*Python API internals  
looks unhuman*

```
26223 class Particle(IParticle):
26224     """
26225
26226     A particle with a form factor and refractive index.
26227
26228     C++ includes: Particle.h
26229
26230     """
26231
26232     __swig_setmethods__ = {}
26233     for _s in [IParticle]:
26234         __swig_setmethods__.update(getattr(_s, '__swig_setmethods__', {}))
26235     __setattr__ = lambda self, name, value: __swig_setattr(self, Particle, name, value)
26236     __swig_getmethods__ = {}
26237     for _s in [IParticle]:
26238         __swig_getmethods__.update(getattr(_s, '__swig_getmethods__', {}))
26239     __getattr__ = lambda self, name: __swig_getattr(self, Particle, name)
26240     __repr__ = __swig_repr
26241
26242
26243 def __init__(self, *args):
26244     """
26245     __init__(Particle self) -> Particle
26246     __init__(Particle self, Material material) -> Particle
26247     __init__(Particle self, Material material, IFormFactor form_factor) -> Particle
26248     __init__(Particle self, Material material, IFormFactor form_factor, IRotation rota
26249
26250     Particle::Particle(Material material, const IFormFactor &form_factor, const IRotat
26251
26252     """
26253     this = _libBornAgainCore.new_Particle(*args)
26254     try:
26255         self.this.append(this)
26256     except __builtin__.Exception:
26257         self.this = this
26258
```

# Documentation on Python API

C++ API documentation sometimes might help ...

 BornAgain

Home Gallery Download News Documentation Contact About 

Home > Documentation > [Working with Python scripts](#) > Python API

## Python API

Simulation scripts interact with the BornAgain core library through an Application Programmer Interface (API). This API consists of numerous classes and their member functions. The primary API is written in the programming language C++. All important classes and their member functions are also available through a Python API.


The [BornAgain C++ User API Reference](#), and the [Comprehensive BornAgain C++ API Reference](#) are always up to date, since they are automatically extracted from the source code (which contains comment lines in the special **Doxygen** format in order to enable this self documentation).

For the moment, we do not dispose of a similarly efficient documentation generator for Python. Therefore, Python users need to refer to the C++ API. Even though Python and C++ have different syntax, it is usually straightforward to infer from the C++ API how the corresponding Python method call will look like.

[<](#) [>](#)

Documentation

- ⊕ Introduction
- ⊕ Getting started
- ⊕ Using Graphical User Interface
- ⊖ Working with Python scripts
  - Setup of a PyCharm project
  - ⊕ Basic simulation tutorial
  - Material types
  - ⊕ Detector types
  - Accessing simulation results
  - Particle positioning
  - Particle rotation
  - Particle composition
  - ⊕ Interference functions
    - Magnetic particles
    - [Python API](#)
  - ⊕ Fitting
- ⊕ Sample model reference
- ⊕ Getting help
- ⊕ Developer's corner





# BornAgain C++ API

Simulate and fit neutron and x-ray scattering at grazing incidence

Main Page

User API

Full C++ API

File List

Class List

Class Index

Class Hierarchy

BornAgain

User API

Full C++ API

Class List

Class Index

Class Hierarchy

File List

Search

Class Index

A | B | C | D | E | F | G | H | I | K | L | M | N | O | P | R | S | T | U | V | W | Z

A

AnisoPyramid

AsymRippleBuilder

AttLimits

Attributes

AxesUnitsWrap

UnitConverterSimple::AxisData

AxisInfo

B

BaseMaterialImpl

Basic2DLatticeBuilder

Basic2DParaCrystalBuilder

BasicLattice

BasicVector3D

Beam

Bin1D

Bin1DCVector

Bin1DKVector

BiPyramid

Box

BoxCompositionBuilder

BoxCompositionRotateXBuilder

BoxCompositionRotateYBuilder

BoxCompositionRotateZandYBuilder

BoxCompositionRotateZBuilder

BoxesSquareLatticeBuilder

BoxStackCompositionBuilder

C

IntegratorReal::CallbackHolder

IntegratorMCMiser::CallbackHolder

FitStatus

FixedBinAxis

FootprintFactorGaussian

FootprintFactorSquare

FormatErrorException (Exceptions)

FormFactorAnisoPyramid

FormFactorBox

FormFactorCoherentPart

FormFactorCoherentSum

FormFactorComponents

FormFactorCone

FormFactorCone6

FormFactorCoreShell

FormFactorCrystal

FormFactorCuboctahedron

FormFactorCylinder

FormFactorDebyeBueche

FormFactorDecoratorDebyeWaller

FormFactorDecoratorMaterial

FormFactorDecoratorPositionFactor

FormFactorDecoratorRotation

FormFactorDodecahedron

FormFactorDot

FormFactorDWBA

FormFactorDWBAPol

FormFactorEllipsoidalCylinder

FormFactorFullSphere

FormFactorFullSpheroid

FormFactorGauss

FormFactorHemiEllipsoid

FormFactorIcosahedron

FormFactorLongBox

FormFactorLongBoxGauss

I

IClusteredParticles

IComputation

Icosahedron

IdentityRotation

IDetector

IDetector2D

IDetectorResolution

IDistribution1D

IDistribution1DSampler

IDistribution2DSampler

IFactory

IFootprintFactor

IFormFactor

IFormFactorBorn

IFormFactorDecorator

IFresnelMap

IFTDecayFunction1D

IFTDecayFunction2D

IFTDistribution1D

IFTDistribution2D

IFunctionAdapter (Fit)

IHistogram

IIntensityFunction

IIntensityNormalizer

IInterferenceFunction

IInterferenceFunctionStrategy

ILatticeOrientation

ILayerRTCoefficients

ILayout

IMinimizer

IMultiLayerBuilder

INamed

InfinitePlane

M

MagneticCylindersBuilder

MagneticMaterialImpl

MagneticParticleZeroFieldBuilder

MagneticRotationBuilder

MagneticSpheresBuilder

MagneticSubstrateZeroFieldBuilder

Material

MaterialBySLDImpl

PlainMultiLayerBySLDBuilder::MaterialData

MatrixFresnelMap

MatrixRTCoefficients

MesoCrystal

MesoCrystalBuilder

MillerIndex

MillerIndexOrientation

Minimizer (Fit)

MinimizerCatalogue

MinimizerFactory

MinimizerInfo

MinimizerOptions

MinimizerResult (Fit)

Minuit2Minimizer

MPSimulation

MultiLayer

MultilayerInfo

MultiLayerWithRoughnessBuilder

MultiOption

MultipleLayoutBuilder

# BornAgain C++ API

## FormFactorBox Class Reference

Hard particles

### Description

A rectangular prism (parallelepiped).

Definition at line 23 of file [FormFactorBox.h](#).

► Inheritance diagram for FormFactorBox:

### Public Member Functions

	<b>FormFactorBox</b> (double length, double width, double height) Constructor of a rectangular cuboid. <a href="#">More...</a>
<b>FormFactorBox</b> *	<b>clone</b> () const overridefinal Returns a clone of this <b>ISample</b> object.
void	<b>accept</b> ( <b>INodeVisitor</b> *visitor) const overridefinal Calls the <b>INodeVisitor</b> 's visit method.
double	<b>getLength</b> () const
double	<b>getHeight</b> () const
double	<b>getWidth</b> () const
double	<b>radialExtension</b> () const overridefinal Returns the (approximate in some cases) radial size of the particle of this form factor's shape. <a href="#">More...</a>
complex_t	<b>evaluate_for_q</b> ( <b>cvector_t</b> q) const overridefinal Returns scattering amplitude for complex scattering wavevector $q=k_i-k_f$ . <a href="#">More...</a>

# BornAgain C++ API

## FormFactorBox.cpp

[Go to the documentation of this file.](#)

```
1 // ***** //  
2 //  
3 // BornAgain: simulate and fit scattering at grazing incidence  
4 //  
5 ///! @file      Core/HardParticle/FormFactorBox.cpp  
6 ///! @brief      Implements class FormFactorBox.  
7 ///!  
8 ///! @homepage   http://www.bornagainproject.org  
9 ///! @license     GNU General Public License v3 or higher (see COPYING)  
10 ///! @copyright   Forschungszentrum Jülich GmbH 2018  
11 ///! @authors     Scientific Computing Group at MLZ (see CITATION, AUTHORS)  
12 //  
13 // ***** //  
14 //  
15 #include "FormFactorBox.h"  
16 #include "BornAgainNamespace.h"  
17 #include "Box.h"  
18 #include "MathFunctions.h"  
19 #include "RealParameter.h"  
20 //  
21 ///! Constructor of a rectangular cuboid.  
22 ///! @param length: length of the base in nanometers  
23 ///! @param width: width of the base in nanometers  
24 ///! @param height: height of the box in nanometers  
25 FormFactorBox::FormFactorBox(double length, double width, double height)  
26     : m_length(length), m_width(width), m_height(height)  
27 {  
28     setName(BornAgain::FFBoxType);  
29     registerParameter(BornAgain::Length, &m_length).setUnit(BornAgain::UnitsNm).setNonnegative();  
30     registerParameter(BornAgain::Width, &m_width).setUnit(BornAgain::UnitsNm).setNonnegative();  
31     registerParameter(BornAgain::Height, &m_height).setUnit(BornAgain::UnitsNm).setNonnegative();  
32     onChange();  
33 }  
34 //  
35 complex_t FormFactorBox::evaluate_for_q(cvector_t q) const  
36 {  
37     complex_t qzHdiv2 = m_height/2*q.z();  
38     return m_height*m_length*m_width *  
39         MathFunctions::sinc(m_length/2*q.x()) * MathFunctions::sinc(m_width/2*q.y()) *  
40         MathFunctions::sinc(qzHdiv2) * exp_I(qzHdiv2);  
41 }  
42
```

# Features available only in Python

## Simulation

- Simulation with distributed parameters
- Some exotic interference functions

## Fitting

- Fitting multiple data sets
- Fit along slices
- Third party minimizers
- Custom objective functions

## Complex workflows

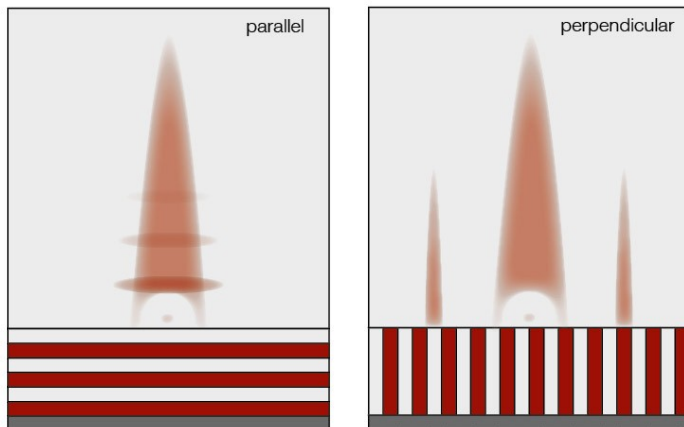
# Demo

Simulating lamellar structure

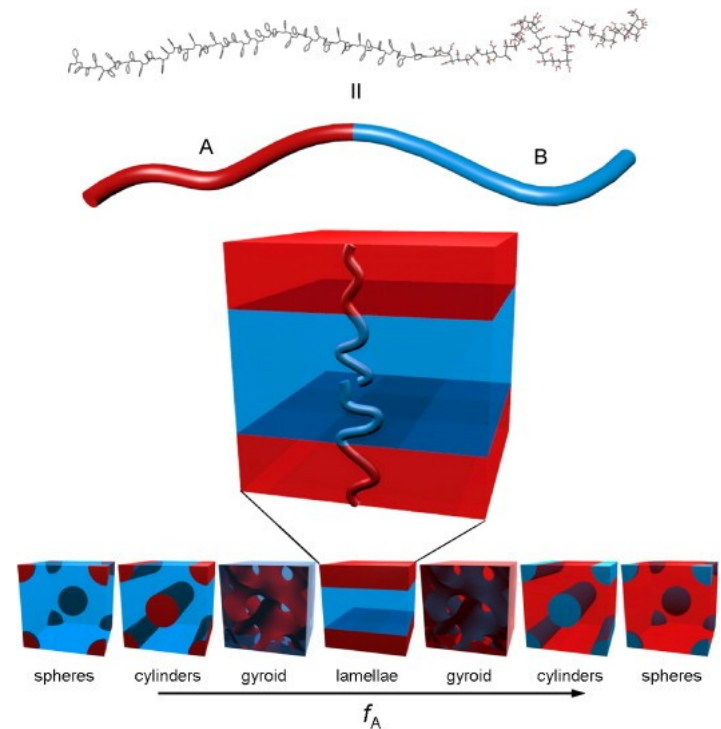
`pyapi04_lamellar.py`

# Simulating lamellar structures

- Represents one of ordered phase of block copolymers during self-assembly
- Alternating layers of different materials in the form of lamellae



<https://wiki.anton-paar.com/>

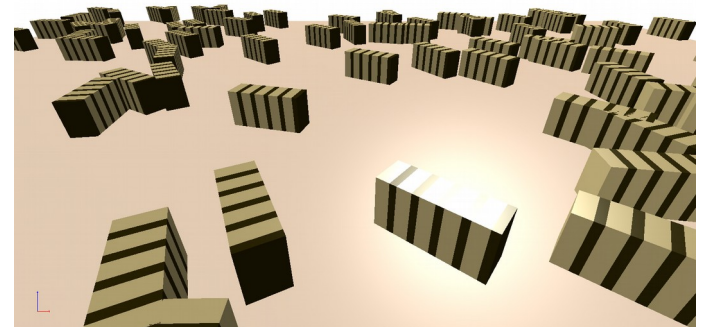
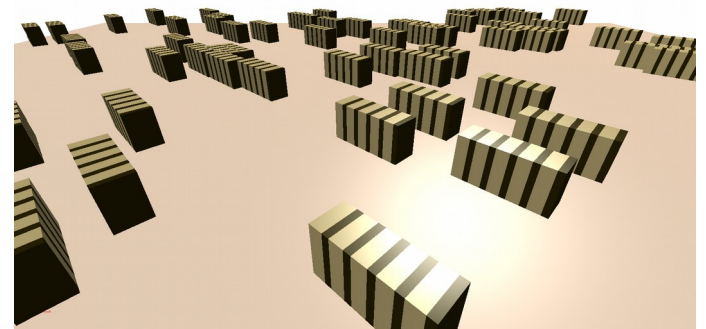
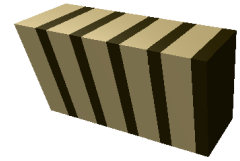


The resulting GISAS pattern depends on the size and arrangement of lamellar structure

# Simulating lamellar structures

Steps to simulate vertically oriented lamellae in BornAgain using PythonAPI

- Define two materials
  - `HomogeneousMaterial`
- Define two boxes
  - `FormFactorBox`
- Define lamellar structure as stack of boxes
  - `ParticleComposition`
- Add rotation around Z
  - `RotationZ`
- Apply rotation angle distribution 0-180
  - `ParticleDistribution`



# Simulating lamellar structures

## 11.1.2 Box (cuboid)

Real-space geometry

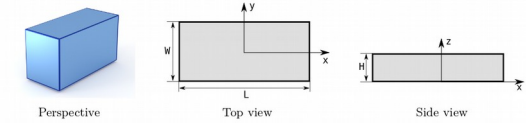
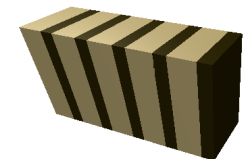
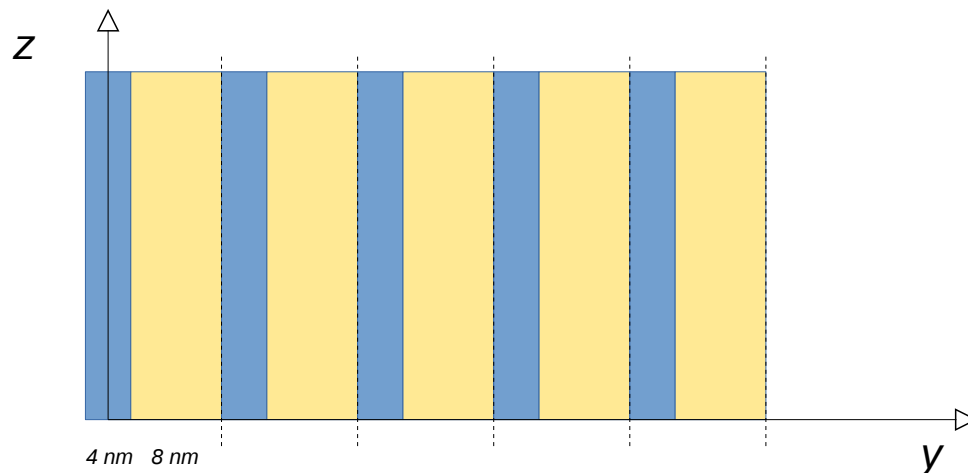


Figure 11.5: A rectangular cuboid.

Syntax and parameters

```
FormFactorBox(double length, double width, double height)
```

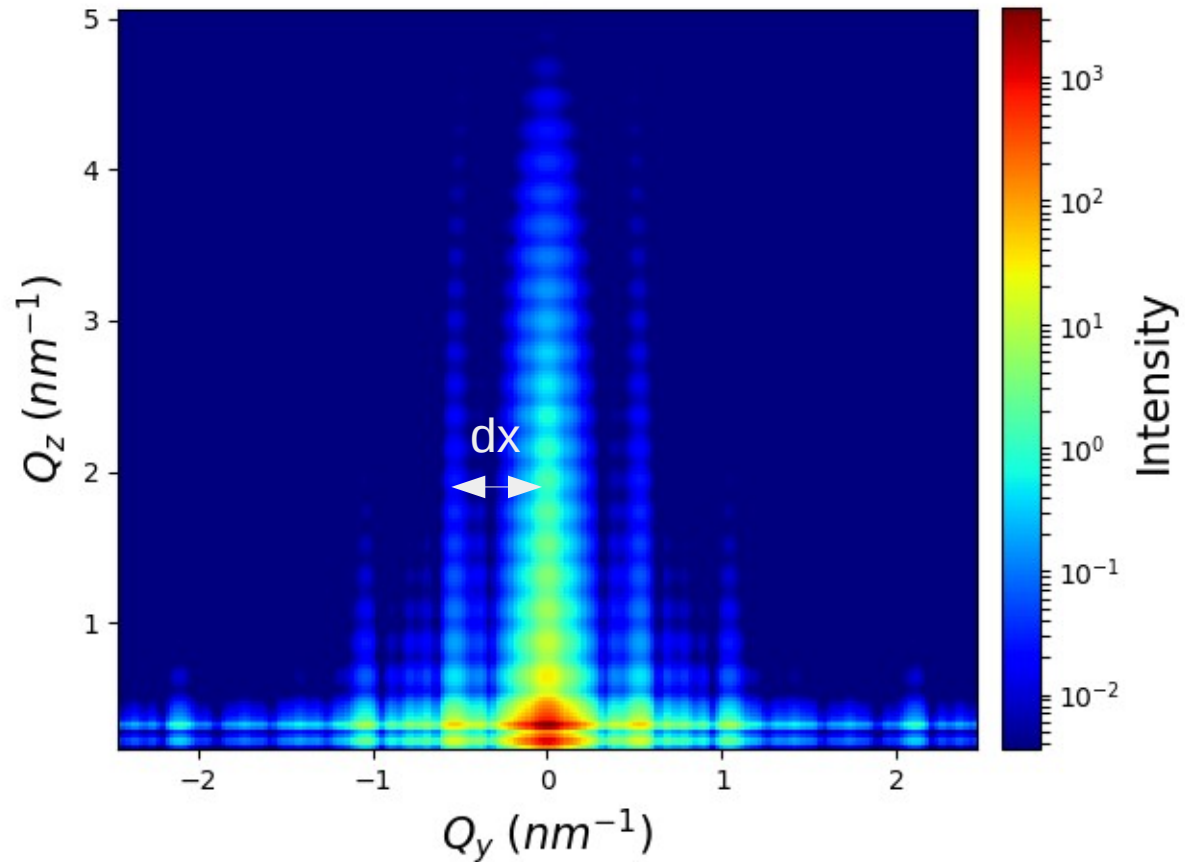
```
def get_horizontal_lamellar():
    mat_a = ba.HomogeneousMaterial("PTFE", 5.20508729E-6, 1.96944292E-8)
    mat_b = ba.HomogeneousMaterial("HMDSO", 2.0888308E-6, 1.32605651E-8)
    length = 30*nm
    width_a = 4*nm
    width_b = 8*nm
    height = 30*nm
    nstack = 5
    stack = ba.ParticleComposition()
    for i in range(0, nstack):
        box_a = ba.Particle(mat_a, ba.FormFactorBox(length, width_a, height))
        box_b = ba.Particle(mat_b, ba.FormFactorBox(length, width_b, height))
        stack.addParticle(box_a, ba.kvector_t(0.0, i*(width_a+width_b), 0.0))
        stack.addParticle(box_b, ba.kvector_t(0.0, (width_a + width_b)/2. + i*(width_a+width_b), 0.0))
```





# Simulating lamellar structures

## Results

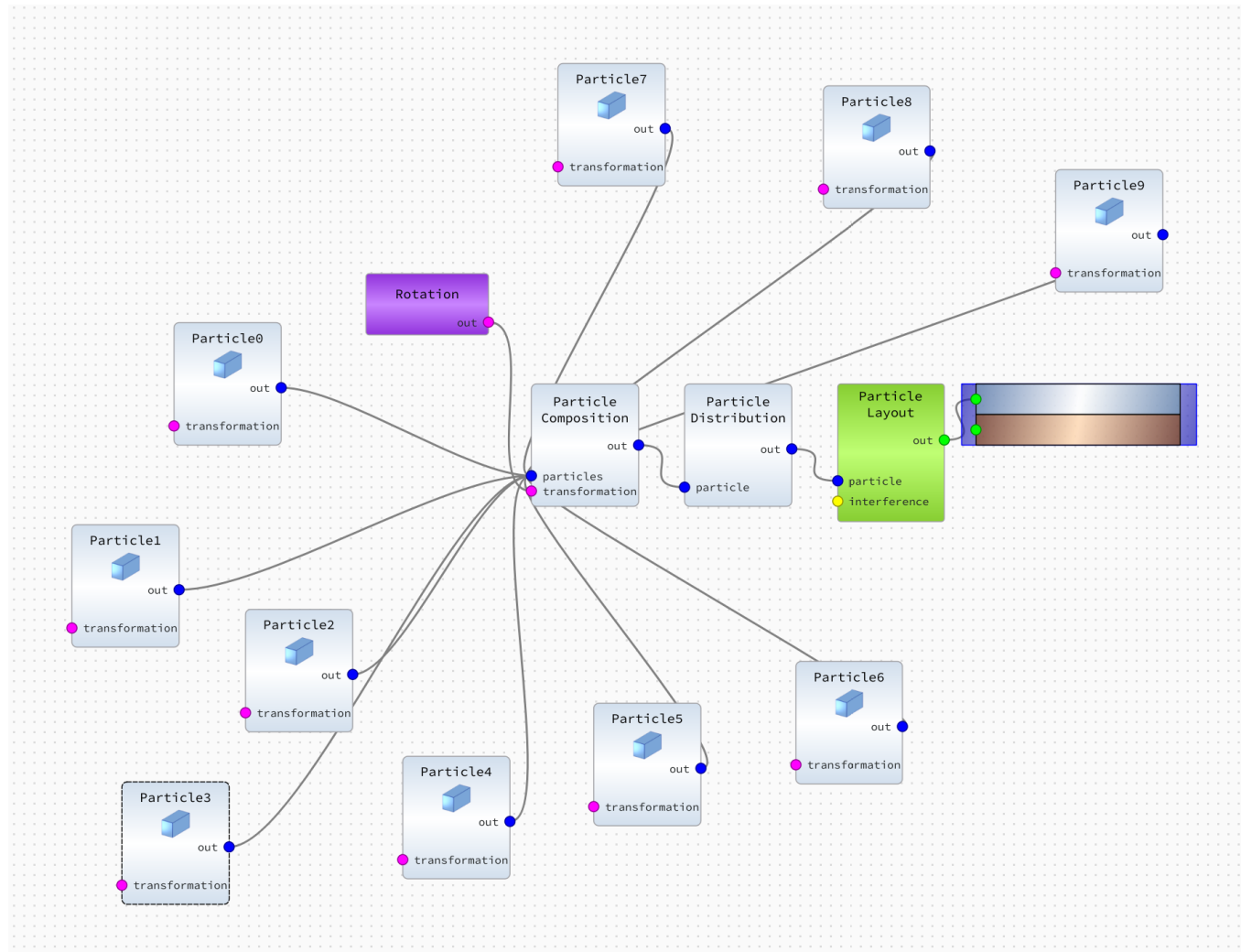


$$2 \cdot \pi / dx = 12.0 \text{ nm}^{-1}$$

the value coincide with lamellar period  
which was defined in simulation script

# Simulating lamellar structures

If we would have to do it in GUI ...



# Task

Modifying lamellar example

`pyapi04_lamellar_vertical_solution1.py`

`pyapi04_lamellar_vertical_solution2.py`

# Modifying lamellar example

Task: make lamellar parallel to surface

