BornAgain

Software for simulating and fitting
X-ray and neutron small-angle scattering
at grazing incidence

User Manual

Version 1.0.0 (February 26, 2015)

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

Scientific Computing Group
Jülich Centre for Neutron Science
at Heinz Maier-Leibnitz Zentrum Garching
Forschungszentrum Jülich GmbH

Authors: Scientific Computing Group
at Heinz Maier-Leibnitz Zentrum (MLZ) Garching:
Céline Durniak, Marina Ganeva, Gennady Pospelov,
Walter Van Herck, Joachim Wuttke

Contents

Introduction

BornAgain is a free and open-source software package to simulate and fit small-angle scattering at grazing incidence (GISAS). It supports analysis of both X-ray (GISAXS) and neutron (GISANS) data. Its name, 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.

BornAgain almost completely reproduces the functionality of the widely used program IsGISAXS by Rémi Lazzari [**?**]. BornAgain goes beyond IsGISAXS by supporting an unrestricted number of layers and particles, diffuse reflection from rough layer interfaces, particles with inner structures, neutron polarization and magnetic scattering. Adhering to a strict object-oriented design, BornAgain provides a solid base for future extensions in response to specific user needs.

BornAgain is a platform-independent software, with active support for Linux, MacOS and Microsoft Windows. It is a free and open source software provided under the terms of the GNU General Public License (GPL, version 3 or higher). This documentation is released under the Creative Commons license CC-BY-SA. When BornAgain is used in preparing scientific papers, please cite software and manual as follows:

> C. Durniak, M. Ganeva, G. Pospelov, W. Van Herck, J. Wuttke (2015),
> BornAgain — Software for simulating and fitting X-ray and neutron small-angle scattering at grazing incidence, version 1.0.0,
> http://www.bornagainproject.org

This User Manual is complementary to the online documentation at http://www.bornagainproject.org. It does not duplicate information that is more conveniently read online. Therefore, Section 1 just contains a few pointers to the web site. The remainder of this User Manual mostly contains background on the sample models and on the scattering theory implemented in BornAgain, and some documentation of the corresponding Python functions.

Software and documentation are work in progress. We cannot guarantee that they are accurate and correct. Anyway, it is entirely in the responsibility of users to ensure that their data interpretation is physically meaningful. If in doubt, please contact us.

We are grateful for all kind of feedback: criticism, praise, bug reports, feature requests or contributed modules. If questions go beyond normal user support, we will be glad to discuss a scientific collaboration.

Chapter 1

Online documentation

This User Manual is complementary to the online documentation at the project web site `http://www.bornagainproject.org`. It does not duplicate information that is more conveniently read online. This dummy chapter contains no more than a few pointers to the web site.

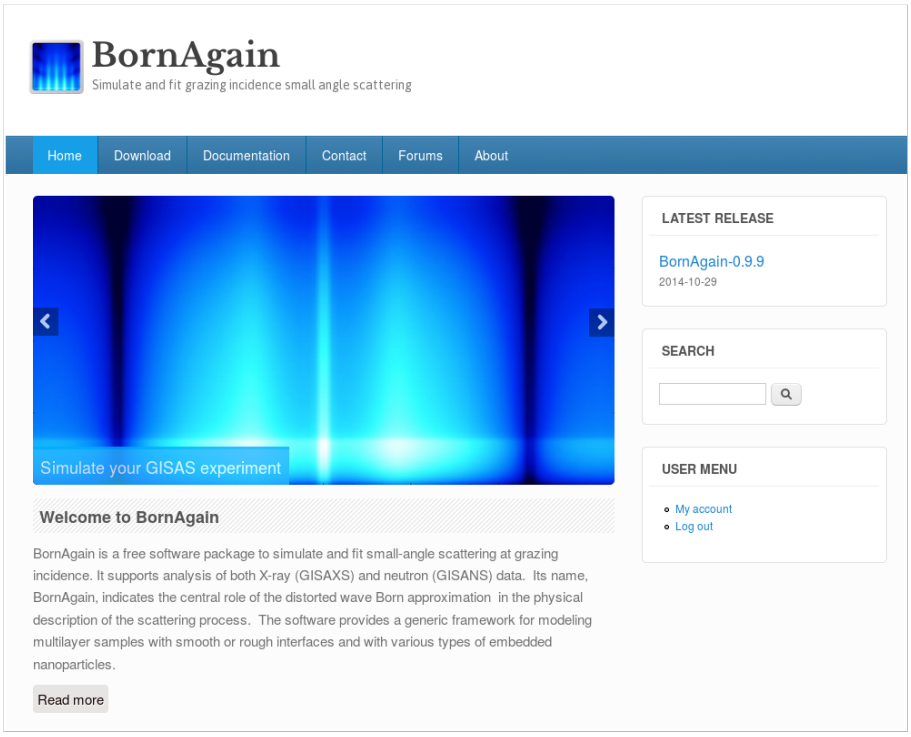


Figure 1.1: A screenshot of the home page `http://www.bornagainproject.org`.

1.1 Download and installation

The DOWNLOAD section on the `BornAgain` web site points to the download location for binary and source packages. It also provides a link to our `git` server where the unstable development trunk is available for contributors or for users who want to live on the edge.

The DOCUMENTATION section contains pages with *Installation instructions*.

1.2 Further online information

The DOCUMENTATION section of the project web site contains in particular

- an overview of the software architecture,
- a list of implemented functionality,
- tutorials for "Working with `BornAgain`", using either the Graphical User Interface or `Python` scripts,
- a comprehensice collection of examples that demonstrate how to use `BornAgain` for modelling various sample structures and different experimental conditions,
- a link to the API reference for using `BornAgain` through `Python` scripts or C++ programs.

1.3 Registration, contact, discussion forum

To stay informed about the ongoing development of `BornAgain`, register on the project homepage `http://www.bornagainproject.org` ("Create new account"). You will then receive our occasional newsletters, and be authorized to post to the discussion forum

To contact the `BornAgain` development and maintenance team in the Scientific Computing Group of Heinz Maier-Leibnitz Zentrum (MLZ) Garching, write a mail to `contact@bornagainproject.org`, or fill the form in the CONTACT section of the project web site.

For questions that might be of wider interest, please consider posting to the discussion forum, accessible through the FORUMS tab of the project web site.

Chapter 2

Foundations of GISAS

2.1 Wave propagation

2.1.1 X-ray propagation

...to come

2.1.2 Neutron propagation

In a GISANS experiment, we are only interested in small scattering vectors $\mathbf{q}$, and therefore do not register ordinary Bragg scattering by crystal lattices or other correlations at atomic level; at most, Bragg scattering must be accounted for as a loss channels. We therefore describe the neutron-sample interaction in continuum approximation by a refractive index n with

$$n^2 = 1 - \frac{4\pi}{K^2}\rho_s \tag{2.1}$$

where ρ_s is an effective scattering length density [**?**, **?**].

2.1.3 Helmholtz equation

X-ray and neutron propagation are both governed by the same stationary wave equation, the *Helmholtz equation*

$$\left(\boldsymbol{\nabla}^2 + K^2 n(\mathbf{r})^2\right)\Psi(\mathbf{r}), \tag{2.2}$$

where $K = 2\pi/\lambda_0$ is the vacuum wave number, and n is the refractive index. In this equation, the time dependence $\mathrm{e}^{i\omega t}$ of the wave field does not appear explicitly. In GISAS, we are only interested in elastic scattering. Therefore, the wave frequency ω is considered constant.

Implicitly, K depends on ω through a *dispersion relation*. This relation is fundamentally different for X-rays (where it is linear, $\omega \propto K$) and for thermal neutrons (where it is quadratic, $\omega \propto K^2$). However, for stationary problems this frequency dependence of K does not matter. Therefore we can use K instead of ω as a constant that

characterizes the incoming radiation. In this way, scalar X-ray and neutron scattering can be described by exactly the same wave equation (2.2).

2.2 Scalar scattering

For both X-rays and neutrons the refractive index n is close to 1. In the following, we shall use this fact to solve the Helmholtz equation in a perturbation expansion that is known as the Born approximation though it goes back to Lord Rayleigh who devised it for the scattering of sound, and later also applied it to electromagnetic waves.

2.2.1 Born approximation

We rewrite the scalar Helmholtz equation (2.2) as

$$\left(\nabla^2 + K^2\right)\Psi(\mathbf{r}) = \frac{\chi(\mathbf{r})}{4\pi}\Psi(\mathbf{r}) \tag{2.3}$$

with

$$\chi(\mathbf{r}) := \frac{4\pi}{K^2}\left(1 - n^2(\mathbf{r})\right). \tag{2.4}$$

For neutrons, this definition just compensates (2.1) so that $\chi = \rho_s$. Equation (2.2) looks like an inhomogenous differential equation — provided we neglect for a moment that unknown function Ψ reappears on the right side. The homogeneous equation

$$\left(\nabla^2 + K^2\right)\Psi_0(\mathbf{r}) = 0 \tag{2.5}$$

is solved by plane waves and superpositions thereof. For an isolated inhomogeneity,

$$\left(\nabla^2 + K^2\right)G(\mathbf{r}) = \delta(\mathbf{r}) \tag{2.6}$$

is solved by the Green's function[1]

$$G(\mathbf{r}) = \frac{\mathrm{e}^{iKr}}{4\pi r}, \tag{2.7}$$

which is an outgoing spherical wave originating from $\mathbf{r} = 0$. Convoluting this function with the given inhomogeneity $(\chi/4\pi)\Psi$, we obtain the formal solution of the full inhomogeneous equation (2.3)

$$\Psi(\mathbf{r}) = \Psi_0(\mathbf{r}) + \int \mathrm{d}^3 r' G(\mathbf{r} - \mathbf{r}')\frac{\chi(\mathbf{r}')}{4\pi}\Psi(\mathbf{r}'). \tag{2.8}$$

[1] Verification under the condition $\mathbf{r} \neq 0$ is a straightforward exercice in vector analysis. For the special case $\mathbf{r} = 0$, one encloses the origin in a small sphere and integrates by means of the Gauß-Ostrogadsky divergence theorem. This explains the appearance of the factor 4π.

However, the integral kernel still contains the full solution Ψ. If we replace this occurence of Ψ by the full right-hand side of (2.8) we obtain

$$\Psi(\mathbf{r}) = \Psi_0(\mathbf{r}) + \int \mathrm{d}^3 r' G(\mathbf{r}-\mathbf{r}') \frac{\chi(\mathbf{r}')}{4\pi} \left(\Psi_0(\mathbf{r}') + \int \mathrm{d}^3 r'' G(\mathbf{r}'-\mathbf{r}'') \frac{\chi(\mathbf{r}'')}{4\pi} \Psi(\mathbf{r}'') \right). \tag{2.9}$$

This can be iterated to obtain an infinite series representation of Ψ. Successive terms in this series contain rising powers of χ. Since χ is assumed to be small, the series is likely to converge. In first Born approximation, only the linear order in χ is retained,

$$\Psi(\mathbf{r}) \simeq \Psi_0(\mathbf{r}) + \int \mathrm{d}^3 r' G(\mathbf{r}-\mathbf{r}') \frac{\chi(\mathbf{r}')}{4\pi} \Psi_0(\mathbf{r}'). \tag{2.10}$$

This is practically always adequate for material investigations with X-rays or neutrons, where the aim is to deduce $\chi(\mathbf{r}')$ from the scattered intensity $|\Psi(\mathbf{r})|^2$.

2.2.2 Far-field expansion

In a scattering experiment, the intensity $|\Psi(\mathbf{r})|^2$ is measured at a detector location $\mathbf{r}$ far away from the sample volume. We choose the coordinate origin at the nominal center of the sample so that the integral in (2.10) runs over $\mathbf{r}'$ with $r' \ll r$. This allows us to expand

$$\left|\mathbf{r}-\mathbf{r}'\right| \simeq \sqrt{r^2 - 2\mathbf{r}\mathbf{r}'} \simeq r - \frac{\mathbf{r}\mathbf{r}'}{r} \equiv r - \frac{\mathbf{k}_f \mathbf{r}'}{K}, \tag{2.11}$$

where we have introduced the outgoing wave vector

$$\mathbf{k}_f := K\frac{\mathbf{r}}{r}. \tag{2.12}$$

We apply this to the Green's function (2.7), retaining the first order in $\mathbf{r}'$ in the exponent, but not in the denominator:

$$G(\mathbf{r}-\mathbf{r}') \simeq \frac{\mathrm{e}^{iKr - i\mathbf{k}_f\mathbf{r}'}}{4\pi r}. \tag{2.13}$$

This is an outgoing spherical wave with respect to $\mathbf{r}$, but a plane wave with respect to $\mathbf{r}'$. If the incident radiation is also a plane wave,

$$\Psi_0(\mathbf{r}) = \mathrm{e}^{i\mathbf{k}_i\mathbf{r}}, \tag{2.14}$$

then the Born approximation (2.10) becomes essentially a Fourier transform of χ,

$$\Psi(\mathbf{r}) \simeq \Psi_0(\mathbf{r}) + \frac{\mathrm{e}^{iKr}}{r} \int \mathrm{d}^3 r' \mathrm{e}^{i\mathbf{q}\mathbf{r}'} \chi(\mathbf{r}') \tag{2.15}$$

with the *scattering vector*

$$\mathbf{q} := \mathbf{k}_i - \mathbf{k}_f. \tag{2.16}$$

2.2.3 Differential cross section

Above, we said somewhat sloppily that a scattering experiment measures intensities $|\Psi(\mathbf{r}|^2$. We shall now make this more rigorous. In the case of neutron scattering, one actually measures the probability flux. We define it in arbitrary relative units as

$$\mathbf{J} := \Psi^* \frac{\boldsymbol{\nabla}}{2i}\Psi - \Psi\frac{\boldsymbol{\nabla}}{2i}\Psi^*. \tag{2.17}$$

With (2.14), the incident flux is

$$\mathbf{J}_0 = \mathbf{k}_i. \tag{2.18}$$

The scattered wavefield (2.15) is observed at a detector position $\mathbf{r}$ that is not illuminated by the incident beam, hence $\Psi_0(\mathbf{r}) = 0$. This leaves us with

$$\mathbf{J}(\mathbf{r}) = \left|\frac{1}{r}\int \mathrm{d}^3 r' \mathrm{e}^{i\mathbf{q}\mathbf{r}'}\chi(\mathbf{r}')\right|^2 K\frac{\mathbf{r}}{r}. \tag{2.19}$$

The ratio of the scattered current hitting an infinitesimal detector area $r^2\mathrm{d}\Omega$ to the incident flux is expressed as a *differential cross section*

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega} := \frac{r^2 J(\mathbf{r})}{J_0}. \tag{2.20}$$

Using $k_i = K$ and changing notation $\mathbf{r}' \to \mathbf{r}$, we find

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega} = \left|\int \mathrm{d}^3 r \mathrm{e}^{i\mathbf{q}\mathbf{r}}\chi(\mathbf{r})\right|^2 =: \left|\chi(\mathbf{q})\right|^2 \tag{2.21}$$

The differential cross section is just the squared modulus of the Fourier transform $\chi(\mathbf{q})$ of the scattering-length distribution (2.4).

2.3 Distorted Wave Born Approximation

As we have seen the Born approximation relies on plane waves, which basically means that incoming and scattered radiation are assumed to propagate along straight lines, with directions expressed by $\mathbf{k}_i$ and $\mathbf{k}_f$. This is adequate for most experiments with weakly interacting radiation, and especially for most X-ray and neutron scattering techniques, but not for grazing-incidence small-angle scattering where radiation is refracted and reflected by interfaces and therefore does not propagate along straight lines. The Born approximation also fails for collisions of charged particles. For them, Massey and Mott devised in the early 1930s the distorted wave Born approximation (DWBA). A particularly simple variant of this approximation is routinely used to account for refraction and reflection in grazing-incidence scattering.

Refraction and reflection are caused by vertical variations of the average refractive index $n_0(z)$. We rewrite the Helmholtz equation (2.3 as

$$\left(\boldsymbol{\nabla}^2 + K^2 n_0^2(z)\right)\Psi(\mathbf{r}) = \frac{\chi(\mathbf{r})}{4\pi}\Psi(\mathbf{r}) \tag{2.22}$$

with (2.4) replaced by

$$\chi(\mathbf{r}) := \frac{4\pi}{K^2}\left(n_0^2(z) - n^2(\mathbf{r})\right). \tag{2.23}$$

2.3.1 Multilayer systems

In multilayer systems, the first term of (**??**) denotes the specular part of the reflection, while the second term is responsible for the off-specular scattering. This off-specular part is caused by deviations from the perfectly smooth layered system, as e.g. interface roughnesses or included nanoparticles. In here only the case of nanoparticles will be treated.

In the conventions where $H = -\Delta + V$, the potential splits into two parts V_1 and V_2, where only the second part is treated as a perturbation:

$$\begin{aligned} V_1 &= K^2\left(1 - n_0^2(\mathbf{r})\right) \\ V_2 &= \sum_i K^2\left(n_0^2(\mathbf{R}^i) - n_i^2\right) S^i(\mathbf{r}) \otimes \delta(\mathbf{r} - \mathbf{R}^i), \end{aligned}$$

where $n_0(\mathbf{r})$ denotes the refractive index of the unperturbed system (which, in case of a multilayer system, will only depend on its z-coordinate) and n_i is the refractive index of the nanoparticle with shape function S^i and position $\mathbf{R}^i$.

For nanoparticles in a specific layer j, i.e. $V_2 \neq 0$ only in layer j, one only needs the unperturbed solutions in layer j:

$$\begin{aligned} \left\langle \mathbf{r} | \Psi^+_{1k_i} \right\rangle &= (2\pi)^{-3/2}\left[R_j(\mathbf{k}_i)e^{i\mathbf{k}_{j,R}(\mathbf{k}_i)\cdot\mathbf{r}} + T_j(\mathbf{k}_i)e^{i\mathbf{k}_{j,T}(\mathbf{k}_i)\cdot\mathbf{r}}\right] \\ \left\langle \Psi^-_{1k_f} | \mathbf{r} \right\rangle &= (2\pi)^{-3/2}\left[R_j(-\mathbf{k}_f)e^{i\mathbf{k}_{j,R}(-\mathbf{k}_f)\cdot\mathbf{r}} + T_j(-\mathbf{k}_f)e^{i\mathbf{k}_{j,T}(-\mathbf{k}_f)\cdot\mathbf{r}}\right]. \end{aligned}$$

The off-specular contribution to the scattering amplitude then becomes:

$$\begin{aligned} f(\theta,\phi) = -\int d^3\mathbf{r}\frac{V_2(\mathbf{r})}{4\pi}\Big[& T_i T_f e^{i(\mathbf{k}_{j,i} - \mathbf{k}_{j,f})\cdot\mathbf{r}} + R_i T_f e^{i(\widetilde{\mathbf{k}}_{j,i} - \mathbf{k}_{j,f})\cdot\mathbf{r}} \\ & + T_i R_f e^{i(\mathbf{k}_{j,i} - \widetilde{\mathbf{k}}_{j,f})\cdot\mathbf{r}} + R_i R_f e^{i(\widetilde{\mathbf{k}}_{j,i} - \widetilde{\mathbf{k}}_{j,f})\cdot\mathbf{r}}\Big], \end{aligned}$$

where the following shorthand notations were used:

$$\begin{aligned} T_i &\equiv T_j(\mathbf{k}_i) & R_i &\equiv R_j(\mathbf{k}_i) \\ T_f &\equiv T_j(-\mathbf{k}_f) & R_f &\equiv R_j(-\mathbf{k}_f) \\ \mathbf{k}_{j,i} &\equiv \mathbf{k}_{j,T}(\mathbf{k}_i) & \widetilde{\mathbf{k}}_{j,i} &\equiv \mathbf{k}_{j,R}(\mathbf{k}_i) \\ \mathbf{k}_{j,f} &\equiv -\mathbf{k}_{j,T}(-\mathbf{k}_f) & \widetilde{\mathbf{k}}_{j,f} &\equiv -\mathbf{k}_{j,R}(-\mathbf{k}_f). \end{aligned}$$

From this expression, one sees that the scattering amplitude consists of a weighted sum of Fourier transforms of the potential V_2. Using

$$V_2(\mathbf{r}) = \sum_i 4\pi\rho_{s,rel,i} S^i(\mathbf{r}) \otimes \delta(\mathbf{r} - \mathbf{R}^i),$$

with $\rho_{s,rel,i} \equiv K^2\left(n_0^2(\mathbf{R}^i) - n_i^2\right)/4\pi$, the scattering amplitude becomes

$$f(\theta,\phi) = -\sum_i \rho_{s,rel,i}\mathscr{F}^i_{\mathrm{DWBA}}(\mathbf{k}_{j,i},\mathbf{k}_{j,f},\mathbf{R}^i_z)e^{i(\mathbf{k}_{j,i\parallel}-\mathbf{k}_{j,f\parallel})\cdot\mathbf{R}^{i\parallel}},$$

with

$$\begin{aligned}\mathscr{F}^i_{\mathrm{DWBA}}(\mathbf{k}_i,\mathbf{k}_f,R_z) \equiv\ & T_iT_fF^i(\mathbf{k}_i-\mathbf{k}_f)e^{i(k_{iz}-k_{fz})R_z} + R_iT_fF^i(\widetilde{\mathbf{k}}_i-\mathbf{k}_f)e^{i(-k_{iz}-k_{fz})R_z}\\ &+ T_iR_fF^i(\mathbf{k}_i-\widetilde{\mathbf{k}}_f)e^{i(k_{iz}+k_{fz})R_z} + R_iR_fF^i(\widetilde{\mathbf{k}}_i-\widetilde{\mathbf{k}}_f)e^{i(-k_{iz}+k_{fz})R_z},\end{aligned}$$

TO MERGE IN: In the DWBA, the form factor of a particle in a multilayer system is given by

$$\begin{aligned}F_{\mathrm{DWBA}}(\mathbf{k}_i,\mathbf{k}_f,r_z) =\ & T_iT_fF_{\mathrm{BA}}(\mathbf{k}_i-\mathbf{k}_f)e^{i(k_{i,z}-k_{f,z})r_z} + R_iT_fF_{\mathrm{BA}}(\widetilde{\mathbf{k}}_i-\mathbf{k}_f)e^{i(-k_{i,z}-k_{f,z})r_z}\\ &+ T_iR_fF_{\mathrm{BA}}(\mathbf{k}_i-\widetilde{\mathbf{k}}_f)e^{i(k_{i,z}+k_{f,z})r_z} + R_iR_fF_{\mathrm{BA}}(\widetilde{\mathbf{k}}_i-\widetilde{\mathbf{k}}_f)e^{i(-k_{i,z}+k_{f,z})r_z},\end{aligned} \tag{2.24}$$

where F_{BA} is the expression of the form factor in the Born approximation, r_z is the z-coordinate of the particle's position (measured from the bottom of the particle), $\mathbf{k}_i = (k_{i,x},k_{i,y},k_{i,z})$ $\mathbf{k}_f = (k_{f,x},k_{f,y},k_{f,z})$ are the incident and scattered wave vectors in air, respectively [**?**]. With a tilde ($\widetilde{}$), these wavevectors components are evaluated in the multilayer system (the refractive indices of the different constituting materials have to be taken into account). T_i, T_f, R_i, R_f are the transmission and reflection coefficients for the incident wave (index i) or the scattered one (index f). These coefficients can be calculated using the Parratt formalism [**?**] or the matrix method [**?**]. $\mathbf{k}_i - \mathbf{k}_f$ is equal to the scattering vector $\mathbf{q}$ and the z-axis is pointing upwards.

With this last expression, the same techniques as demonstrated in section **??** can be applied, leading to the following expression for the expectation value of the scattering cross-section:

$$\begin{aligned}&\left\langle\frac{d\sigma}{d\Omega}(\mathbf{k}_i,\mathbf{k}_f)\right\rangle_{\text{Off-specular}}\\ &= \sum_\alpha p_\alpha\left|\mathscr{F}_\alpha(\mathbf{k}_{j,i},\mathbf{k}_{j,f},R_{\alpha,z})\right|^2 + \frac{\rho_S}{S}\sum_{\alpha,\beta}p_\alpha p_\beta\mathscr{F}_\alpha(\mathbf{k}_{j,i},\mathbf{k}_{j,f},R_{\alpha,z})\mathscr{F}^*_\beta(\mathbf{k}_{j,i},\mathbf{k}_{j,f},R_{\beta,z})\\ &\times\iint_S d^2\mathbf{R}^\parallel_\alpha d^2\mathbf{R}^\parallel_\beta\mathscr{G}_{\alpha,\beta}\left(\mathbf{R}^\parallel{}_\alpha,\mathbf{R}^\parallel{}_\beta\right)\exp\left[i\mathbf{q}_{j\parallel}\cdot(\mathbf{R}^\parallel_\alpha-\mathbf{R}^\parallel_\beta)\right].\end{aligned}$$

The main differences with respect to the cross-section in the Born approximation are:

1. The particle form factor now consists of a more complex expression and now depends on both incoming and outgoing wavevectors and also on the z-coordinate of the particle;
2. Since the z-coordinate of the particles is implicitly included in its formfactor, the position integrals only run over x- and y-coordinates and the volume and density gets replaced with the surface area and surface density respectively.

2.4 Polarization

ome GISAS experiments are performed with polarized radiation or/and involve polarization-dependent interactions. Therefore we need also to consider the propagation of polarized X-rays and neutrons. Electromagnetic field vectors and neutron spinors are fundamentally different mathematical objects Nevertheless, for our purpose they can be mapped onto a uniform .... <to elaborate> .... so that one and the same formalism can be applied to polarized GISAXS and GISANS. This shall be derived in the following.

2.4.1 Polarized X-rays

2.4.2 Polarized neutrons

2.5 TO RECYCLE

Consider a scattering volume V, containing N scattering centers with shape functions $S^i(\mathbf{r})$, positions $\mathbf{R}^i$ and scattering length density ρ_s (relative to the ambient material).

The differential cross-section (per scattering center) is then given by:

$$\frac{d\sigma}{d\Omega}(\mathbf{q}) = \frac{1}{N}\left|\int_V \rho_s(\mathbf{r}) e^{i\mathbf{q}\cdot\mathbf{r}} d^3\mathbf{r}\right|^2. \tag{2.25}$$

Chapter 3

Wave propagation through multilayers

This chapter describes how to calculate the coherent wave solution for neutrons incident on a multilayered sample. In the first section, only scalar interactions will be considered. The second part uses a spinor and matrix formalism to account for neutron polarization.

⚠ Remark: In `BornAgain` there is no limitation to the number of layers composing the sample.

3.1 Geometry of the sample

The geometry used to describe the sample is shown in figure 3.2. The z-axis is perpendicular to the sample's surface and pointing upwards. The x-axis is perpendicular to the detector plane. The input and the scattered output beams are each characterized by two angles α_i, ϕ_i and α_f, ϕ_f, respectively. Our choice of orientation for the angles α_i and α_f is so that they are positive as shown in figure 3.2.

The layers are defined by their thicknesses (parallel to the z-direction), their possible roughnesses (equal to 0 by default) and the materials they are made of. They have an infinite extension in the x and y directions. And, except for roughness, their interfaces are plane and perpendicular to the z-axis. There is also no limitation to the number of layers that could be defined in `BornAgain`. Note that the thickness of the top and bottom layer are not defined.

The nanoparticles are characterized by their form factors (*i.e.* the Fourier transform of the shape function - see Appendix **??** for a list of form factors implemented in `BornAgain`) and the composing material. 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).

By placing the particles inside or on top of a layer, we impose their vertical positions, whose values correspond to the bottoms of the particles. The in-plane distri-

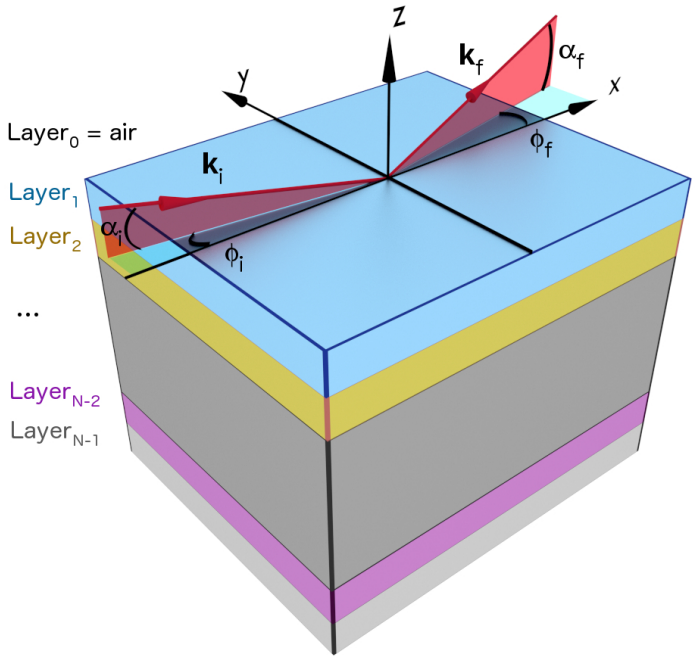


Figure 3.1: Representation of the scattering geometry. n_j is the refractive index of layer j and α_i and ϕ_i are the incident angles of the wave propagating. α_f is the exit angle with respect to the sample's surface and ϕ_f is the scattering angle with respect to the scattering plane.

bution of particles is linked with the way the particles interfere with each other. It is therefore implemented when dealing with the interference function.

The complex refractive index associated with a layer or a particle is written as $n = 1 - \delta + i\beta$, with $\delta, \beta \in \mathbb{R}$. In our program, we input δ and β directly.
The input beam is assumed to be monochromatic without any spatial divergence.

3.2 Basic formalism for scalar index of refraction

In this section, we investigate refraction by a multilayer system with flat interfaces and with scalar refraction indices. We first develop the basic formalism without absorption, and far from total reflection conditions. Absorption, total reflection and evanescent waves will be discussed in subsection 3.2.5. In section 3.3, we generalize the formalism to include polarization-dependent interaction.

Polarized neutron propagation at glancing incidence in multilayer systems has been studied by Spiering and Deak [...], using Abeles' matrix formalism and matrix exponentials. In Abeles' formalism [], plane waves are parametrized by ψ and $\partial_z \psi$. We prefer a parametrization (3.4) by the amplitudes of forward and backward propagat-

ing plane waves, which is mathematically simpler, and allows a more direct physical interpretation.

3.2.1 Wave equation for a basic multilayer model

The geometry for reflection by a multilayer sample is shown in figure 3.2. Here we only consider specular reflection, hence the azimuthal angle $\phi_f = 0$.

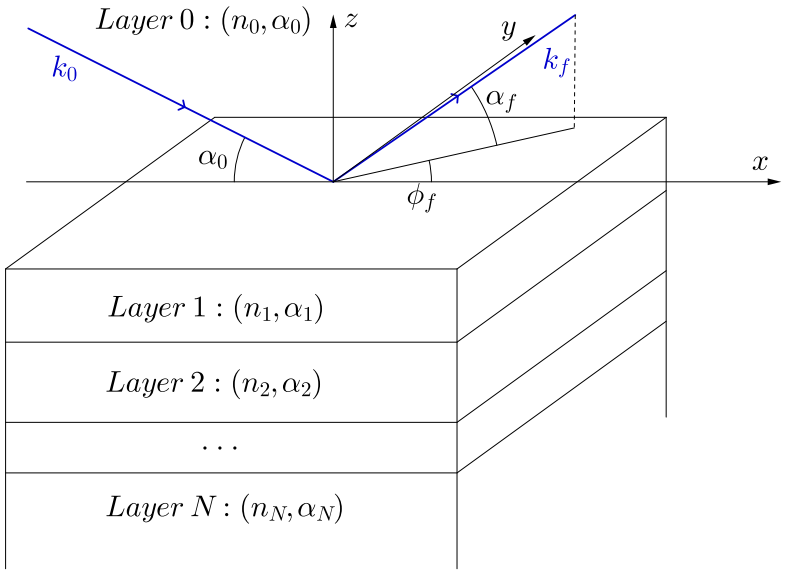


Figure 3.2: Representation of the scattering geometry for multilayer specular reflectivity. n_i is the refractive index of layer i and α_i is the incident angle of the wave propagating in layer i and incident on layer $i+1$. α_f is the exit angle with respect to the sample's surface and ϕ_f is the scattering angle with respect to the scattering plane.

All layer interfaces are assumed to be perfectly smooth. All layers are homogenous; layer i has the complex refractive index n_i. Layer 0 is the air or vacuum on top of the sample; layer N is the substrate; these two layers are assumed semi-inifinite. We first consider scalar interactions. Polarization-dependent reflection will be discussed in section 3.3.

Within one layer, stationary wave propagation with given frequency, hence with fixed vacuum wavenumber K, obeys the differential equation

$$\left(\Delta + K^2 n_i^2\right)\psi(\mathbf{r}) = 0. \tag{3.1}$$

At interfaces between layers, the wave function $\psi(\mathbf{r})$ and its first derivative $\nabla\psi(\mathbf{r})$ must evolve continuously.

3.2.2 Wave propagation within one layer

Since our multilayer model has infinite extension in x and y–directions, we can factorize $\psi(\mathbf{r})$, and assume plane-wave propagation for the in-plane component $\mathbf{r}_\parallel$:

$$\psi(\mathbf{r}) = \psi(z)\mathrm{e}^{i\mathbf{k}_\parallel \mathbf{r}_\parallel}. \tag{3.2}$$

From the continuity conditions we infer that $\mathbf{k}_\parallel$ is constant across layers. The wave equation (3.1) reduces to a one-dimensional equation in z,

$$\left(\partial_z^2 + K^2 n_i^2 - k_\parallel^2\right)\psi(z) = 0, \tag{3.3}$$

which is solved by

$$\psi_i(z) = a_i \mathrm{e}^{ik_{\perp i}(z-z_i)} + b_i \mathrm{e}^{-ik_{\perp i}(z-z_i)}, \tag{3.4}$$

where z_i is the coordinate of the *bottom* interface of layer i, introduced here as a constant offset for later convenience. In the case of the semi-infinite bottom layer N, z_N can be chosen arbitrarily.

Inserting (3.4) into (3.3), we obtain the dispersion relation

$$k_\parallel^2 + k_{\perp i}^2 = K^2 n_i^2. \tag{3.5}$$

If n_i is real and $k_\parallel < K n_i$, this is just the Pythagorean equation for the perpendicular components of either of the two wavevectors: vector $\mathbf{k}_\pm = \mathbf{k}_\parallel \pm k_{\perp i}\hat{\mathbf{z}}$. They correspond to the two summands in (3.4), and describe plane waves propagating upwards or downwards. These waves have glancing angles

$$\alpha_i := \arctan(k_{\perp i}/k_\parallel). \tag{3.6}$$

Equivalently,

$$k_\parallel = K n_i \cos\alpha_i. \tag{3.7}$$

Since $k_\parallel$ is constant across layers, we have

$$n_i \cos\alpha_i = \text{the same for all } i, \tag{3.8}$$

which is Snell's refraction law.

For later convenience we abbreviate

$$f_i := \sqrt{n_i^2 - n_0^2 \cos^2\alpha_0}, \tag{3.9}$$

which is a combination of material parameters (n_0, n_i) and a geometric parameter (α_0) that describes the incident beam. If n_i is real, and the argument of the square root nonnegative, we simply have $f_i = n_i \sin\alpha_i$. However, using f_i allows for later generalizations to cope with absorbing media or with total reflection and evanescent waves. The wavenumber in (3.4) can now be written as

$$k_{\perp i} = K f_i. \tag{3.10}$$

3.2.3 Wave propagation across layers

In this section, the continuity of $\psi(z)$ and $\partial_z\psi(z)$ at layer interfaces is used to derive a recursion rule for the coefficients a_i and b_i of (3.4). At the *bottom* interface of layer $i = 0,\ldots,N-1$, continuity requires

$$\begin{aligned} \psi_i(z_i) &= \psi_{i+1}(z_{i+1}+d_{i+1}), \\ \partial_z\psi_i(z_i) &= \partial_z\psi_{i+1}(z_{i+1}+d_{i+1}), \end{aligned} \tag{3.11}$$

where

$$d_i := z_{i-1} - z_i \tag{3.12}$$

is the thickness of layer i. With the solution (3.4), conditions (3.11) become

$$\begin{aligned} a_i + b_i, &= a_{i+1}\mathrm{e}^{iKf_{i+1}d_{i+1}} + b_{i+1}\mathrm{e}^{-iKf_{i+1}d_{i+1}}, \\ a_i f_i - b_i f_i, &= a_{i+1}\mathrm{e}^{iKf_{i+1}d_{i+1}} f_{i+1} - b_{i+1}\mathrm{e}^{-iKf_{i+1}d_{i+1}} f_{i+1}. \end{aligned} \tag{3.13}$$

We introduce the vector notation

$$c_i := \begin{pmatrix} a_i \\ b_i \end{pmatrix}, \tag{3.14}$$

to write (3.13) as

$$F_i c_i = F_{i+1} D_{i+1} c_{i+1} \tag{3.15}$$

with the matrices

$$F_i := \begin{pmatrix} 1 & 1 \\ f_i & -f_i \end{pmatrix}, \text{ and } D_i := \begin{pmatrix} \delta_i & 0 \\ 0 & \delta_i^* \end{pmatrix}, \tag{3.16}$$

and the phase factor

$$\delta_i := \mathrm{e}^{iKf_i d_i}. \tag{3.17}$$

We define the transfer matrix

$$M_i := F_{i-1}^{-1} F_i D_i, \tag{3.18}$$

to obtain the recursion

$$c_i = M_{i+1} c_{i+1}. \tag{3.19}$$

Straightforward computation yields

$$M_i = \frac{1}{2f_{i-1}} \begin{pmatrix} \delta_i(f_{i-1}+f_i) & \delta_i^*(f_{i-1}-f_i) \\ \delta_i(f_{i-1}-f_i) & \delta_i^*(f_{i-1}+f_i) \end{pmatrix} = \frac{1}{2} \begin{pmatrix} \delta_i(1+\rho_i) & \delta_i^*(1-\rho_i) \\ \delta_i(1-\rho_i) & \delta_i^*(1+\rho_i) \end{pmatrix} \tag{3.20}$$

with the ratio

$$\rho_i := \frac{f_i}{f_{i-1}}. \tag{3.21}$$

The recursion starts from the semi-infinite substrate layer N where there is no upwards propagating wave, hence $a_N = 0$. For layer i, the solution is

$$c_i = M_{i+1} \cdots M_N c_N. \tag{3.22}$$

3.2.4 Currents

Currents shall be computed in relative units as

$$\mathbf{J} := \psi^*(\mathbf{r}) \frac{\boldsymbol{\nabla}}{2i} \psi(\mathbf{r}) + \text{c.c.} \tag{3.23}$$

We are interested in the component J_z. With (3.4), we obtain in layer i

$$\begin{aligned} J_{zi} &= \tfrac{1}{2}\left(a_i^* \mathrm{e}^{-ik_{\perp i}(z-z_i)} + b_i^* \mathrm{e}^{ik_{\perp i}(z-z_i)}\right) k_{\perp i} \left(a_i \mathrm{e}^{ik_{\perp i}(z-z_i)} - b_i \mathrm{e}^{-ik_{\perp i}(z-z_i)}\right) + \text{c.c.} \\ &= \left(|a_i|^2 - |b_i|^2\right) K f_i, \end{aligned} \tag{3.24}$$

provided $k_{\perp i}$ is real. Obviously, the $|a_i|^2$ term is due to the upward beam and the $|b_i|^2$ term to the downward beam.

In the absence of absorption we expect that the net current is the same in all layers. We verify this as follows. Using the vector notation (3.14), we write (3.24) as a sesquilinear form

$$J_{zi} = c_i^\dagger \begin{pmatrix} K f_i & 0 \\ 0 & -K f_i \end{pmatrix} c_i. \tag{3.25}$$

Using the recursion (3.19) for c_i, we obtain a recursion for J_{zi}:

$$\begin{aligned} J_{zi} &= c_i^\dagger \begin{pmatrix} K f_i & 0 \\ 0 & -K f_i \end{pmatrix} c_i \\ &= c_{i+1}^\dagger M_{i+1}^\dagger \begin{pmatrix} K f_i & 0 \\ 0 & -K f_i \end{pmatrix} M_{i+1} c_{i+1} \\ &= c_{i+1}^\dagger \begin{pmatrix} K f_{i+1} & 0 \\ 0 & -K f_{i+1} \end{pmatrix} c_{i+1} \\ &= J_{z,i+1}, \end{aligned} \tag{3.26}$$

which means that the net current is constant.

For a single interface between two semi-infinite media, wave amplitudes are related by

$$\begin{pmatrix} a_0 \\ b_0 \end{pmatrix} = M_1 \begin{pmatrix} 0 \\ b_1 \end{pmatrix} = \frac{\delta_1^* b_1}{2f_0} \begin{pmatrix} f_0 - f_1 \\ f_0 + f_1 \end{pmatrix}. \tag{3.27}$$

The reflectivity of the interface is

$$R = \left| \frac{a_0}{b_0} \right|^2 = \left| \frac{f_0 - f_1}{f_0 + f_1} \right|^2, \tag{3.28}$$

which agrees with Fresnel's result for s-polarized light.

3.2.5 Damped waves in absorbing media or under total reflection conditions

If layer i absorbs radiation, its index of refraction n_i has a positive imaginary component. By (3.5), $k_\parallel^2 + k_{\perp i}^2$ then also has a positive imaginary component. From the continuity of ψ and $\boldsymbol{\nabla}\psi$ across layer interfaces it still follows that $\mathbf{k}_\parallel$ is constant. We assume that the top layer 0 is not absorbing. Hence $\mathbf{k}_\parallel$ is real. To fulfill (3.5), it is necessary that $k_{\perp i}$ has a positive imaginary component. In consequence, the intensities of the forward and backward travelling beams (3.4) decrease exponentially in $\pm z$.

Snell's law of refraction (3.8) cannot be fulfilled if

$$n_i < n_0 \cos \alpha_0. \tag{3.29}$$

In this case, total reflection occurs at the top interface of layer i, accompanied by an evanescent wave within layer i. Strictly speaking, total reflection is only possible if layer i is not absorbing. Otherwise, some intensity would be dissipated by the evanescent wave, and the reflection would not be total. Also, the unequality (3.29) is undefined if n_i has an imaginary component.

With definition (3.9), total reflection occurs if f_i is a pure imaginary number. For an absorbing medium, f_i has a positive imaginary part and a non-zero real part. Therefore it is appropriate to treat total reflection as a special case of refraction by a medium with complex f_i.

For complex f_i, the theory developed above remains applicable, with the following exceptions: The geometric interpretation of the wavevectors $\mathbf{k}_\pm$ in Eqs. (3.6–3.8) is untenable, and the computation of currents in Eqs. (3.24–3.26) is invalid because it relies on $\operatorname{Im} k_{\perp i} = 0$.

3.2.6 Numerical considerations

...

3.3 Reflection with polarization-dependent interactions

3.3.1 Wave equation and propagation within one layer

To allow for polarization-dependent interactions, we replace the squared index of refraction n^2 by $1+\underline{\underline{\chi}}$, where $\underline{\underline{\chi}}$ is a 2×2 susceptibility matrix. The wave equation (3.1) for layer i becomes

$$(\Delta + K^2 + K^2 \underline{\underline{\chi}}_i)\underline{\psi}(\mathbf{r}) = 0, \tag{3.30}$$

where $\underline{\psi}(\mathbf{r})$ is a two-component spinor wave function, with components $\psi_\uparrow(\mathbf{r})$ and $\psi_\downarrow(\mathbf{r})$. At interfaces between layers, both spinor components of $\underline{\psi}(\mathbf{r})$ and $\boldsymbol{\nabla}\underline{\psi}(\mathbf{r})$ must evolve continuously.

The reasons for the factorization (3.2) still apply, and so we can write

$$\underline{\psi}(\mathbf{r}) = \underline{\psi}(z)\mathrm{e}^{i\mathbf{k}_\parallel \mathbf{r}_\parallel}. \tag{3.31}$$

As before, $\mathbf{k}_\parallel$ is constant across layers. The wave equation (3.30) reduces to

$$\left(\partial_z^2 + K^2 + K^2\underline{\underline{\chi}}_i - k_\parallel^2\right)\underline{\psi}(z) = 0. \tag{3.32}$$

We abbreviate

$$\underline{\underline{H}}_i := K^2(1+\underline{\underline{\chi}}_i) - k_\parallel^2 \tag{3.33}$$

so that the wave equation becomes simply

$$\left(\partial_z^2 + \underline{\underline{H}}_i\right)\underline{\psi}(z) = 0. \tag{3.34}$$

The solution is

$$\underline{\psi}_i(z) = \sum_{j=1}^{2} \underline{x}_{ij}\left(\alpha_{ij}\mathrm{e}^{ip_{ij}(z-z_i)} + \beta_{ij}\mathrm{e}^{-ip_{ij}(z-z_i)}\right), \tag{3.35}$$

where the $\underline{x}_{ij}$ are eigenvectors of $\underline{\underline{H}}_i$ with eigenvalues p_{ij}^2:

$$\left(-p_{ij}^2 + \underline{\underline{H}}_i\right)\underline{x}_{ij} = 0 \quad \text{for} \quad j = 1,2. \tag{3.36}$$

In a reproducible algorithm, the eigenvectors $\underline{x}_{ij}$ must be chosen according to some arbitrary normalization rule, for instance

$$|\underline{x}_{ij}| = 1, \quad x_{ij\uparrow} \text{ real and nonnegative.} \tag{3.37}$$

Similarly, a rule is needed how to handle the case of one degenerate eigenvalue, which includes in particular the case of scalar interactions.

3.3.2 Wave propagation across layers

Generalizing (3.14), we introduce the coefficient vector

$$c_i := (\alpha_{i1}, \alpha_{i2}, \beta_{i1}, \beta_{i2})^{\mathrm{T}}. \tag{3.38}$$

To match solutions for neighboring layers, continuity is requested for both spinorial components of $\underline{\psi}$ and $\nabla\underline{\psi}$. As before (3.15), we have at the bottom of layer i

$$F_i c_i = F_{i+1} D_{i+1} c_{i+1}, \tag{3.39}$$

where the matrices are now

$$F_i := \begin{pmatrix} x_{i1\uparrow} & x_{i2\uparrow} & x_{i1\uparrow} & x_{i2\uparrow} \\ x_{i1\downarrow} & x_{i2\downarrow} & x_{i1\downarrow} & x_{i2\downarrow} \\ x_{i1\uparrow} p_{i1} & x_{i2\uparrow} p_{i2} & -x_{i1\uparrow} p_{i1} & -x_{i2\uparrow} p_{i2} \\ x_{i1\downarrow} p_{i1} & x_{i2\downarrow} p_{i2} & -x_{i1\downarrow} p_{i1} & -x_{i2\downarrow} p_{i2} \end{pmatrix} \tag{3.40}$$

and

$$D_i := \mathrm{diag}(\delta_{i1}, \delta_{i2}, \delta_{i1}^*, \delta_{i2}^*) \tag{3.41}$$

with the phase factor

$$\delta_{ij} := \mathrm{e}^{i p_{ij} d_i}. \tag{3.42}$$

Note that matrix F_i has the block form

$$F_i = \begin{pmatrix} \underline{\underline{x}}_i & \underline{\underline{x}}_i \\ \underline{\underline{x}}_i \underline{\underline{P}}_i & -\underline{\underline{x}}_i \underline{\underline{P}}_i \end{pmatrix} = \underline{\underline{x}}_i \cdot \begin{pmatrix} \underline{\underline{1}} & \underline{\underline{1}} \\ \underline{\underline{P}}_i & -\underline{\underline{P}}_i \end{pmatrix}, \tag{3.43}$$

with

$$\underline{\underline{x}}_i := \left(\underline{x}_{i1}, \underline{x}_{i2}\right), \quad \underline{\underline{P}}_i := \mathrm{diag}\left(p_{i1}, p_{i2}\right). \tag{3.44}$$

This facilitates the computation of the inverse

$$F_i^{-1} = \frac{1}{2} \begin{pmatrix} \underline{\underline{1}} & \underline{\underline{P}}_i^{-1} \\ \underline{\underline{1}} & -\underline{\underline{P}}_i^{-1} \end{pmatrix} \cdot \underline{\underline{x}}_i^{-1}, \tag{3.45}$$

which is needed for the transfer matrix M_i, defined as in (3.18). With the new meaning of c_i and M_i, the recursion (3.19) and the explicit solution (3.22) hold as derived above. To resolve (3.22) for the reflected amplitudes α_{0j} as function of the incident amplitudes β_{0j}, we choose the notations

$$\underline{\alpha}_i := \begin{pmatrix} \alpha_{i1} \\ \alpha_{i2} \end{pmatrix}, \quad \underline{\beta}_i := \begin{pmatrix} \beta_{i1} \\ \beta_{i2} \end{pmatrix}, \quad M := M_1 \cdots M_N =: \begin{pmatrix} \underline{\underline{m}}_{11} & \underline{\underline{m}}_{12} \\ \underline{\underline{m}}_{21} & \underline{\underline{m}}_{22} \end{pmatrix}, \tag{3.46}$$

where the $\underline{\underline{m}}_{jk}$ are 2×2 matrices. Eq. (3.22) then takes the form

$$\begin{pmatrix} \underline{\alpha}_0 \\ \underline{\beta}_0 \end{pmatrix} = \begin{pmatrix} \underline{\underline{m}}_{11} & \underline{\underline{m}}_{12} \\ \underline{\underline{m}}_{21} & \underline{\underline{m}}_{22} \end{pmatrix} \begin{pmatrix} \underline{0} \\ \underline{\beta}_N \end{pmatrix}, \tag{3.47}$$

which immediately yields

$$\underline{\alpha}_0 = \underline{\underline{m}}_{12} \underline{\underline{m}}_{22}^{-1} \underline{\beta}_0. \tag{3.48}$$

Chapter 4

Scattering by embedded particles

In this and the next chapter we consider scattering by particles that are attached to a substrate or embedded in a layer. These particles are assumed to be of a "nano" or "meso" size, so that their internal atomistic structure does not matter. We first consider "hard-shell" particles that are described by a binary shape function $S(\mathbf{r})$ and a constant coherent scattering-length density ρ_s that must differ from the environment (air or embedding layer) to result in a scattering signal. In Section 4.3 we briefly discuss "core-shell" particles that have a two valued scattering-length distribution $\rho_s(\mathbf{r})$.

In this chapter, we present models isolated particles, and compute their form factor. In the next chapter, we discuss inter-particle correlations.

4.1 Particle shapes

The form factor of a hard-shell particle is its *shape transform*

$$F(\mathbf{q}) = \int_V \exp(i\mathbf{q}.\mathbf{r}) d^3\mathbf{r}, \tag{4.1}$$

where V is the volume of the particle, $\mathbf{q} = \mathbf{k}_i - \mathbf{k}_f$ is the scattering vector with $\mathbf{k}_f$ and $\mathbf{k}_i$ the scattered and incident wave vector, respectively.

The particle's shape is parametrized in a Cartesian frame, with its z-axis pointing upwards and its origin at the center of the bottom of the particle: $\mathbf{r} = (x, y, z)$.

All form factors have been implemented with complex scattering vectors in order to take any material absorption into account.

In `BornAgain` the expression of the form factor has been implemented in the Born approximation. Each of them is defined as

$$F(\mathbf{q}) = \int_V \exp(i\mathbf{q}.\mathbf{r}) d^3\mathbf{r},$$

where V is the volume of the particle's shape, $\mathbf{q} = \mathbf{k}_i - \mathbf{k}_f$ is the scattering vector with $\mathbf{k}_f$ and $\mathbf{k}_i$ the scattered and incident wave vector, respectively. The Distorted Wave Born Approximation can be taken into account as it has been explained in Section **??**.

The particle's shape is parametrized in a Cartesian frame, with its z-axis pointing upwards and its origin at the center of the bottom of the particle: $\mathbf{r} = (x, y, z)$. In the followings, a schematic view will depict this layout for each form factor.

All form factors have been implemented with complex scattering vectors in order to take any material absorption into account.

4.2 Implemented form factors

Table 4.1 lists the shapes whose form factors have been implemented in `BornAgain`. The following pages document each implemented form factor in detail.

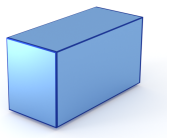
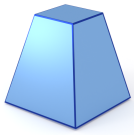
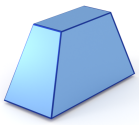
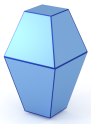
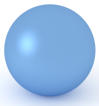
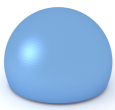
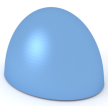
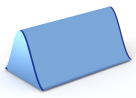
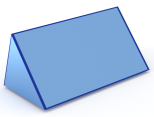
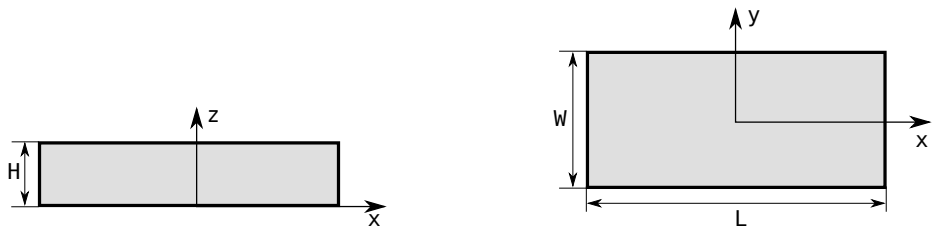
Box, Section 4.2.1 	Prism3, Section 4.2.2 	Tetrahedron, Section 4.2.3 	Prism6, Section 4.2.4 
Cone6, Section 4.2.5 	Pyramid, Section 4.2.6 	Anisotropic pyramid, Section 4.2.7 	Cuboctahedron, Section 4.2.8 
Cylinder, Section 4.2.9 	Ellipsoidal cylinder, Section 4.2.10 	Cone, Section 4.2.11 	Full Sphere, Section 4.2.12 
Truncated Sphere, Section 4.2.13 	Full Spheroid, Section 4.2.14 	Truncated Spheroid, Section 4.2.15 	Hemi Ellipsoid, Section 4.2.16 
Ripple1, Section 4.2.17 	Ripple2, Section 4.2.18 		

Table 4.1: Table of form factors implemented in `BornAgain`.

Page intentionally left blank

4.2.1 Box (cuboid)

Real-space geometry
This shape is a rectangular cuboid as shown in fig. 4.1.



(a) Side view (b) Top view

Figure 4.1: Sketch of a Box.

Parameters

- length of the base L,
- width of the base W,
- height H.

Properties

- volume $V = LWH$,
- particle surface seen from above $S = LW$.

Expression of the form factor

$$F(\mathbf{q}, L, W, H) = LWH \exp\left(i q_z \frac{H}{2}\right) \mathrm{sinc}\left(q_x \frac{L}{2}\right) \mathrm{sinc}\left(q_y \frac{W}{2}\right) \mathrm{sinc}\left(q_z \frac{H}{2}\right),$$

where $\mathrm{sinc}(x) = \sin(x)/x$ is the cardinal sine.

Syntax `FormFactorBox(length, width, height)`

Example
Figure 4.2 shows the normalized intensity $|F|^2/V^2$, computed with $L = 20$ nm, $W = 16$ nm, and $H = 13$ nm:

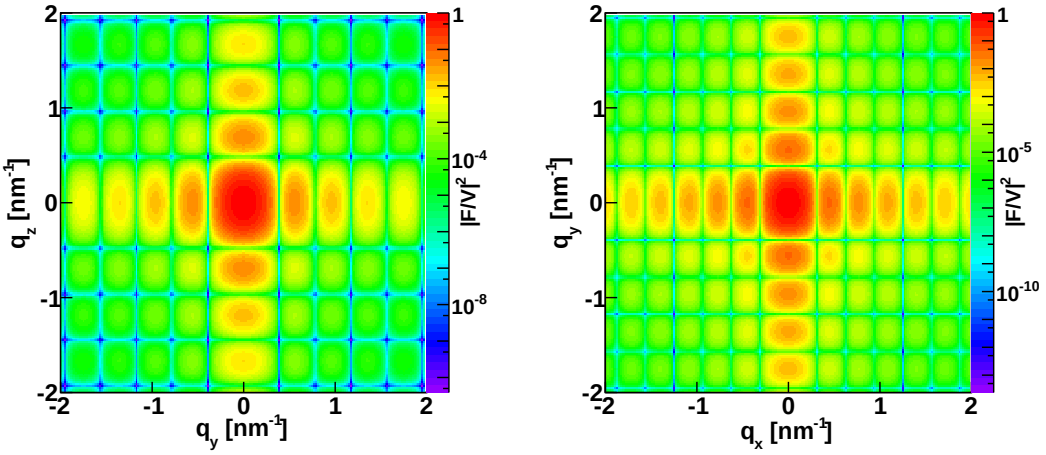
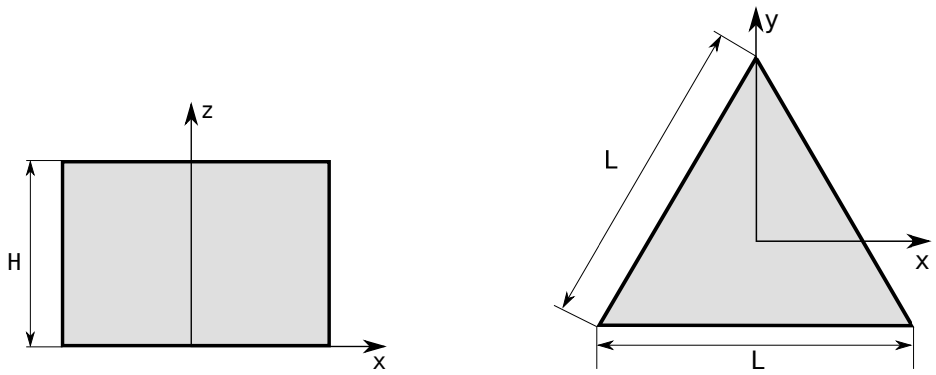


Figure 4.2: Normalized intensity for the form factor of a Box plotted against (q_y, q_z) and (q_x, q_y) and computed with `FormFactorBox(20.*nanometer, 16.*nanometer, 13.*nanometer)`.

References Agrees with "Box" form factor of `IsGISAXS` [?], except for factors 1/2 in the definitions of parameters L, W, H.

4.2.2 Prism3 (triangular)

Real-space geometry
This shape is a triangular prism, whose base is an equilateral triangle as shown in fig. 4.3.



(a) Side view (b) Top view

Figure 4.3: Sketch of a Prism3.

Parameters

- length L of one side of the base,
- height H.

Properties

- volume $V = \frac{\sqrt{3}}{4} HL^2$,
- particle surface seen from above $S = \frac{\sqrt{3}}{4} L^2$.

Expression of the form factor

$$F(\mathbf{q}, L, H) = \frac{2\sqrt{3}}{q_x^2 - 3q_y^2} \exp\left(-iq_y \frac{L}{2\sqrt{3}}\right) \left[\exp\left(i\sqrt{3}q_y \frac{L}{2}\right) - \cos\left(q_x \frac{L}{2}\right) - i\sqrt{3}q_y \frac{L}{2} \operatorname{sinc}\left(q_x \frac{L}{2}\right)\right] \times H \operatorname{sinc}\left(q_z \frac{H}{2}\right) \exp\left(iq_z \frac{H}{2}\right),$$

where $\operatorname{sinc}(x) = \sin(x)/x$ is the cardinal sine.

Syntax `FormFactorPrism3(length, height)`

Example
Figure 4.4 shows the normalized intensity $|F|^2/V^2$, computed with $L = 10$ nm and $H = 13$ nm.

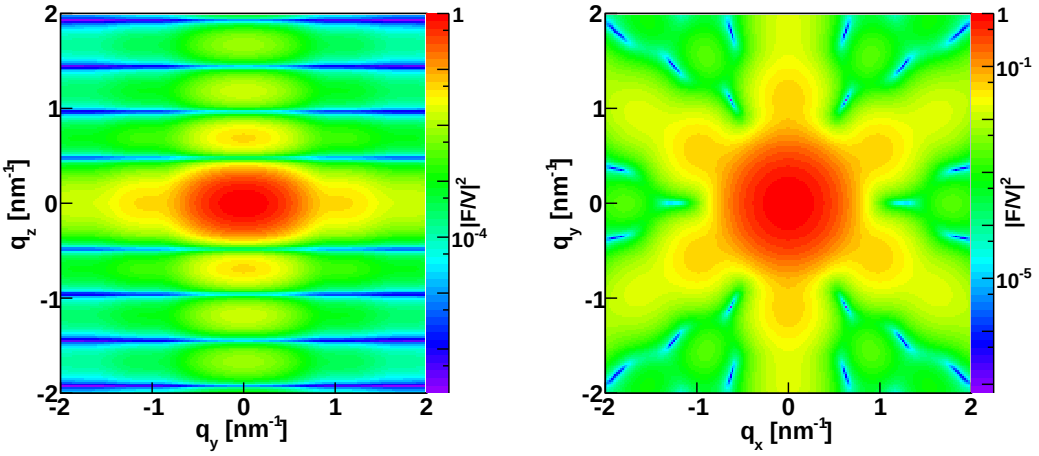
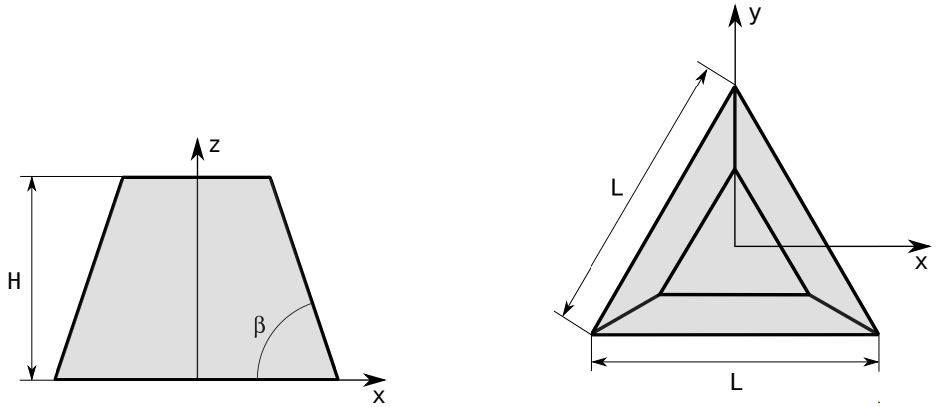


Figure 4.4: Normalized intensity for the form factor of a Prism3 plotted against (q_y, q_z) and (q_x, q_y) and computed with `FormFactorPrism3(10.*nanometer, 13.*nanometer)`.

References Agrees with "Prism3" form factor of `IsGISAXS` [**?**], except for the definition of parameter $L = 2R_{\texttt{IsGISAXS}}$.

4.2.3 Tetrahedron

Real-space geometry
This shape is a truncated tetrahedron as shown in fig. 4.5.



(a) Side view (b) Top view

Figure 4.5: Sketch of a Tetrahedron. The implementation of this shape uses angle α, which is linked to β via $\tan\alpha = 2\tan\beta$. α is the angle between the base and a side face and β the angle between the base and a side edge.

Parameters

- length of one side of the equilateral triangular base L,
- height H,
- angle α is the angle between the base and the side faces.

Restrictions on the parameters: $\dfrac{H}{L} < \dfrac{\tan\alpha}{2\sqrt{3}}$.

Properties

- volume $V = \dfrac{\tan(\alpha)L^3}{24}\left[1-\left(1-\dfrac{2\sqrt{3}H}{L\tan(\alpha)}\right)^3\right]$,
- particle surface seen from above $S = \dfrac{\sqrt{3}}{4}L^2$.

Expression of the form factor

$$F(\mathbf{q},L,H,\alpha) = \frac{\sqrt{3}H}{q_x(q_x^2-3q_y^2)}\exp\left(i\frac{q_z L\tan(\alpha)}{2\sqrt{3}}\right)\times$$
$$\left\{2q_x\exp(iq_3D)\,\mathrm{sinc}(q_3H) - (q_x+\sqrt{3}q_y)\exp(iq_1D)\,\mathrm{sinc}(q_1H) - (q_x-\sqrt{3}q_y)\exp(-iq_2D)\,\mathrm{sinc}(q_2H)\right\},$$

with $\mathrm{sinc}(x) = \sin(x)/x$,

$$q_1 = \frac{1}{2}\left[\frac{q_x\sqrt{3} - q_y}{\tan\alpha} - q_z\right], \quad q_2 = \frac{1}{2}\left[\frac{q_x\sqrt{3} + q_y}{\tan\alpha} + q_z\right], \quad q_3 = \frac{q_y}{\tan\alpha} - \frac{q_z}{2}, \quad D = \frac{L\tan\alpha}{\sqrt{3}} - H.$$

Syntax `FormFactorTetrahedron(length, height, alpha)`

Example
Figure 4.6 shows the normalized intensity $|F|^2/V^2$, computed with $L = 15$ nm, $H = 6$ nm and $\alpha = 60°$.

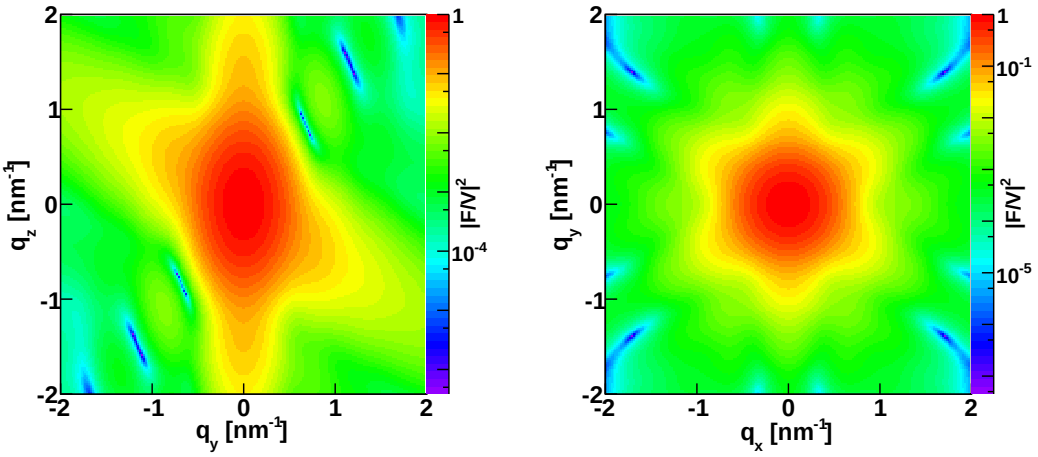
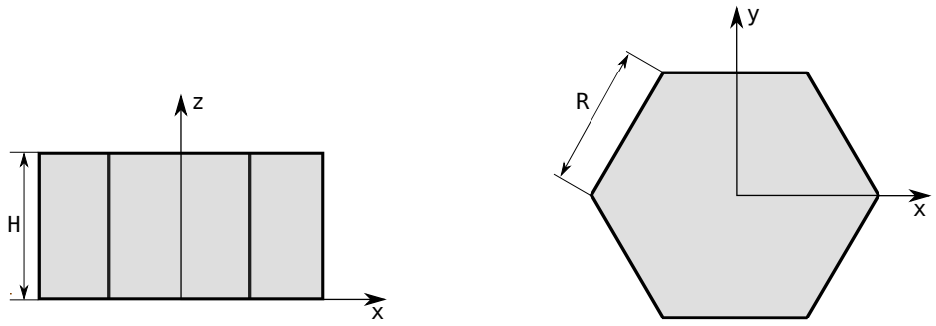


Figure 4.6: Normalized intensity for the form factor of a Tetrahedron plotted against (q_y, q_z) and (q_x, q_y) and computed with `FormFactorTetrahedron(15.*nanometer, 6.*nanometer, 60.*degree)`.

References ??

4.2.4 Prism6 (hexagonal)

Real-space geometry
This shape is an hexagonal prism (see fig. 4.7).



(a) Side view (b) Top view

Figure 4.7: Sketch of a Prism6.

Parameters

- radius of the hexagonal base R,
- height H.

Properties

- volume $V = \frac{3\sqrt{3}}{2} HR^2$,
- particle surface seen from above $S = \frac{3\sqrt{3}R^2}{2}$.

Expression of the form factor

$$F(\mathbf{q}, R, H) = \frac{4H\sqrt{3}}{3q_y^2 - q_x^2} \operatorname{sinc}\left(q_z \frac{H}{2}\right) \exp\left(-i q_z \frac{H}{2}\right) \times \left\{ \frac{3q_y^2 R^2}{4} \operatorname{sinc}\left(\frac{q_x R}{2}\right) \operatorname{sinc}\left(\frac{\sqrt{3} q_y R}{2}\right) + \cos(q_x R) - \cos\left(q_y \frac{\sqrt{3}R}{2}\right) \cos\left(\frac{q_x R}{2}\right) \right\},$$

with $\operatorname{sinc}(x) = \sin(x)/x$.

Syntax `FormFactorPrism6(radius, height)`

Example
Figure 4.8 shows the normalized intensity $|F|^2/V^2$, computed with $R = 5$ nm and $H = 11$ nm.

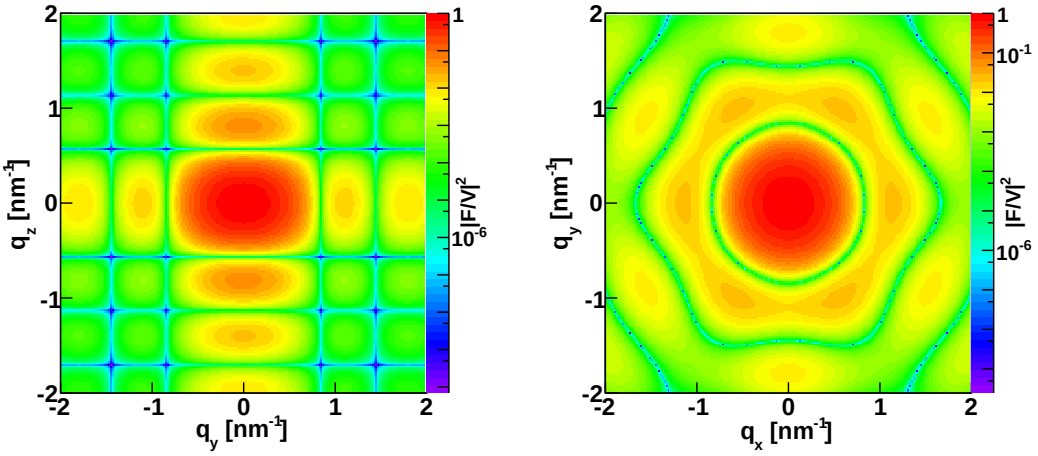
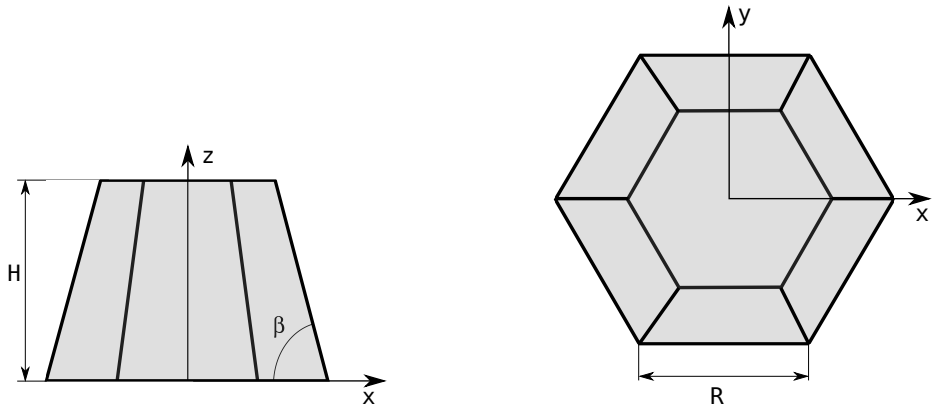


Figure 4.8: Normalized intensity for the form factor of a Prism6 plotted against (q_y, q_z) and (q_x, q_y) and computed with `FormFactorPrism6(5.*nanometer, 11.*nanometer)`.

References Agrees with "Prism6" form factor of `IsGISAXS` [**?**], except for different parametrization, and for a factor H missing in the `IsGISAXS` manual.

4.2.5 Cone6 (hexagonal)

Real-space geometry
It is a truncated hexagonal pyramid (see fig. 4.9).



(a) Side view (b) Top view

Figure 4.9: Sketch of a Cone6. The implementation of this shape uses angle α, which is linked to β via $\tan\alpha = \frac{2}{\sqrt{3}}\tan\beta$. α is measured along one of the base lines and β at one of the base vertices.

Parameters

- radius of the regular hexagonal base R,
- height H,
- angle α is considered between one of the side faces and the middle of a base length.

Restrictions on the parameters: $\frac{2H}{\sqrt{3}R} < \tan\alpha$.

Properties

- volume $V = \frac{3}{4}\tan(\alpha)R^3\left[1-\left(1-\frac{2H}{\tan(\alpha)R\sqrt{3}}\right)^3\right]$,
- particle surface seen from above $S = \frac{3\sqrt{3}R^2}{2}$.

Expression of the form factor
The calculation can be derived from "Prism6" (Section 4.2.4) by considering a side length varying with the vertical position:

$$F(\mathbf{q},R,H,\alpha)=\frac{4\sqrt{3}}{3q_y^2-q_x^2}\int_0^H \exp(iq_z z)\Big[\frac{3}{4}R_z^2 q_y^2\,\mathrm{sinc}\left(\frac{q_x R_z}{2}\right)\mathrm{sinc}\left(\frac{\sqrt{3}q_y R_z}{2}\right) + \cos(q_x R_z)-\cos\left(\frac{\sqrt{3}q_y R_z}{2}\right)\cos\left(\frac{q_x R_z}{2}\right)\Big]dz$$

with $R_z = R - \dfrac{2z}{\sqrt{3}\tan(\alpha)}$ and $\mathrm{sinc}(x)=\sin(x)/x$.

Syntax `FormFactorCone6(radius,height, alpha)`

Example
Figure 4.10 shows the normalized intensity $|F|^2/V^2$, computed with $R = 10$ nm, $H = 13$ nm, and $\alpha = 60°$.

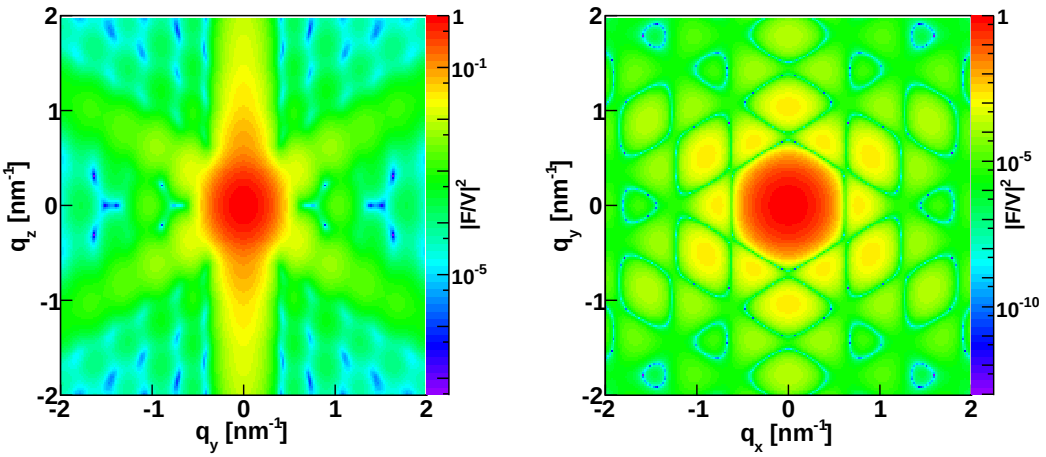
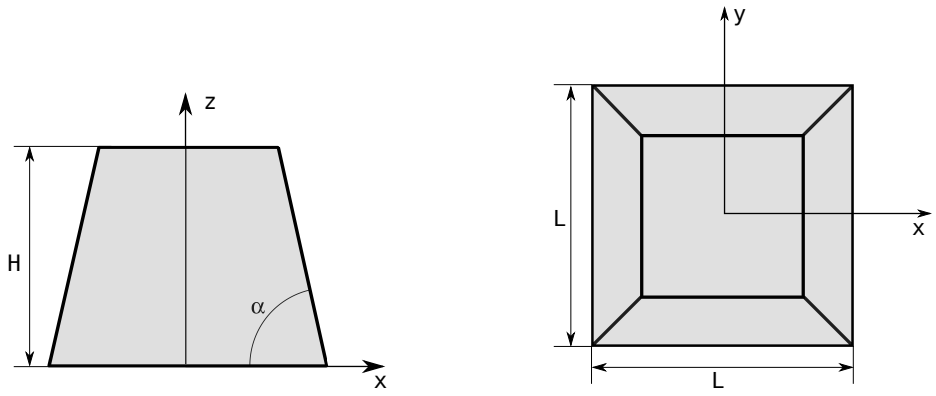


Figure 4.10: Normalized intensity for the form factor of a Cone6 plotted against (q_y, q_z) and (q_x, q_y) and computed with `FormFactorCone6(10.*nanometer,13.*nanometer, 60.*degree)`.

References Agrees with "Cone6" form factor of `IsGISAXS` [**?**] ???

4.2.6 Pyramid (square-based)

Real-space geometry
This shape is a truncated pyramid with a square base as shown in fig. 4.11.



(a) Side view (b) Top view

Figure 4.11: Sketch of a Pyramid

Parameters

- length of one side of the square base L,
- height H,
- α is the angle between the base and the side faces, taken in the middle of the base lines.

Restrictions on the parameters: $\dfrac{2H}{L} < \tan(\alpha)$.

Properties

- volume $V = \dfrac{1}{6}\tan(\alpha)L^3\left[1-\left(1-\dfrac{2H}{\tan(\alpha)L}\right)^3\right]$,
- particle surface seen from above $S = L^2$.

Expression of the form factor

$$F(\mathbf{q}, L, H, \alpha) = \frac{H}{q_x q_y} \times$$
$$\left\{K_1 \cos\left[(q_x - q_y)\frac{L}{2}\right] + K_2 \sin\left[(q_x - q_y)\frac{L}{2}\right] - K_3 \cos\left[(q_x + q_y)\frac{L}{2}\right] - K_4 \sin\left[(q_x + q_y)\frac{L}{2}\right]\right\},$$

with $\mathrm{sinc}(x) = \sin(x)/x$,

$$q_1 = \frac{1}{2}\left[\frac{q_x - q_y}{\tan(\alpha)} + q_z\right], \quad q_2 = \frac{1}{2}\left[\frac{q_x - q_y}{\tan(\alpha)} - q_z\right]$$
$$q_3 = \frac{1}{2}\left[\frac{q_x + q_y}{\tan(\alpha)} + q_z\right], \quad q_4 = \frac{1}{2}\left[\frac{q_x + q_y}{\tan(\alpha)} - q_z\right]$$
$$K_1 = \mathrm{sinc}(q_1 H)\exp(i q_1 H) + \mathrm{sinc}(q_2 H)\exp(-i q_2 H)$$
$$K_2 = -i\,\mathrm{sinc}(q_1 H)\exp(i q_1 H) + i\,\mathrm{sinc}(q_2 H)\exp(-i q_2 H)$$
$$K_3 = \mathrm{sinc}(q_3 H)\exp(i q_3 H) + \mathrm{sinc}(q_4 H)\exp(-i q_4 H)$$
$$K_4 = -i\,\mathrm{sinc}(q_3 H)\exp(i q_3 H) + i\,\mathrm{sinc}(q_4 H)\exp(-i q_4 H)$$

Syntax `FormFactorPyramid(length, height, alpha)`

Examples Figure 4.12 shows the normalized intensity $|F|^2/V^2$, computed with $L =$ 18 nm, $H = 13$ nm and $\alpha = 60°$.

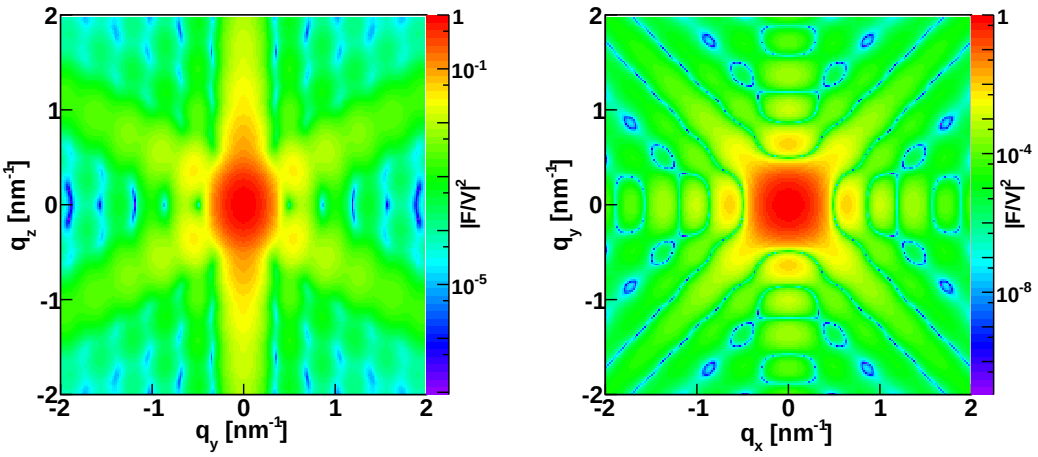
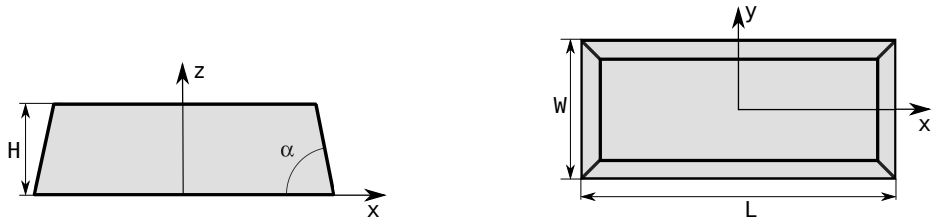


Figure 4.12: Normalized intensity for the form factor of a pyramid plotted against $(q_y,\ q_z)$ and $(q_x,\ q_y)$ and computed with `FormFactorPyramid(18.*nanometer, 13.*nanometer, 60.*degree)`.

References Agrees with "Pyramid" form factor of `IsGISAXS` [**?**], except for different parametrization $L = 2R_{\texttt{IsGISXAXS}}$. [and sign problem??]

4.2.7 AnisoPyramid (rectangle-based)

Real-space geometry
This shape is a truncated right pyramid with a rectangular base as shown in fig. 4.13.



(a) Side view (b) Top view

Figure 4.13: Sketch of an Anisotropic Pyramid.

Parameters

- full length of the base L,
- full width of the base W,
- height H,
- α is the angle between the base and the side faces, taken in the middle of the base lines.

Restrictions on the parameters: $\frac{2H}{L} < \tan(\alpha)$ and $\frac{2H}{W} < \tan(\alpha)$.

Properties

- volume $V = H\Big[LW - \frac{(L+W)H}{\tan(\alpha)} + \frac{4}{3}\frac{H^2}{\tan^2(\alpha)}\Big]$,
- particle surface seen from above $S = LW$.

Expression of the form factor

$$F(\mathbf{q}, L, W, H, \alpha) = \frac{H}{q_x q_y} \times$$
$$\Big\{K_1 \cos\Big(q_x\frac{L}{2} - q_y\frac{W}{2}\Big) + K_2 \sin\Big(q_x\frac{L}{2} - q_y\frac{W}{2}\Big) - K_3 \cos\Big(q_x\frac{L}{2} + q_y\frac{W}{2}\Big) - K_4 \sin\Big(q_x\frac{L}{2} + q_y\frac{W}{2}\Big)\Big\},$$

with $\mathrm{sinc}(x) = \sin(x)/x$,

$$K_1 = \exp(-i q_2 H)\,\mathrm{sinc}(q_2 H) + \exp(i q_1 H)\,\mathrm{sinc}(q_1 H)$$
$$K_2 = i\exp(-i q_2 H)\,\mathrm{sinc}(q_2 H) - i\exp(i q_1 H)\,\mathrm{sinc}(q_1 H)$$
$$K_3 = \exp(-i q_4 H)\,\mathrm{sinc}(q_4 H) + \exp(i q_3 H)\,\mathrm{sinc}(q_3 H)$$
$$K_4 = i\exp(i q_4 H)\,\mathrm{sinc}(q_4 H) - i\exp(i q_3 H)\,\mathrm{sinc}(q_3 H)$$
$$q_1 = \frac{1}{2}\left[\frac{q_x - q_y}{\tan\alpha} + q_z\right], \quad q_2 = \frac{1}{2}\left[\frac{q_x - q_y}{\tan\alpha} - q_z\right]$$
$$q_3 = \frac{1}{2}\left[\frac{q_x + q_y}{\tan\alpha} + q_z\right], \quad q_4 = \frac{1}{2}\left[\frac{q_x + q_y}{\tan\alpha} - q_z\right]$$

Syntax `FormFactorAnisoPyramid(length, width, height, alpha)`

Example
Figure 4.14 shows the normalized intensity $|F|^2/V^2$, computed with $L = 20$ nm, $W = 16$ nm, $H = 13$ nm, and $\alpha = 60°$.

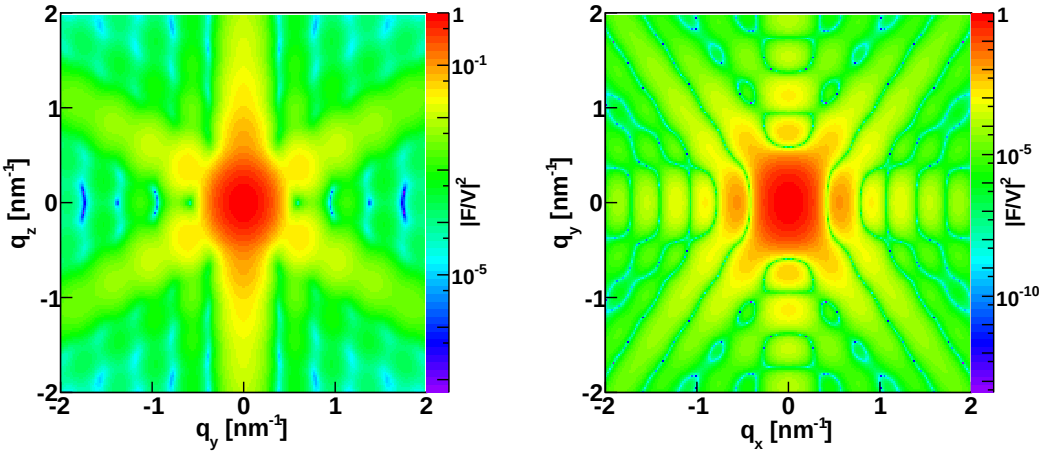


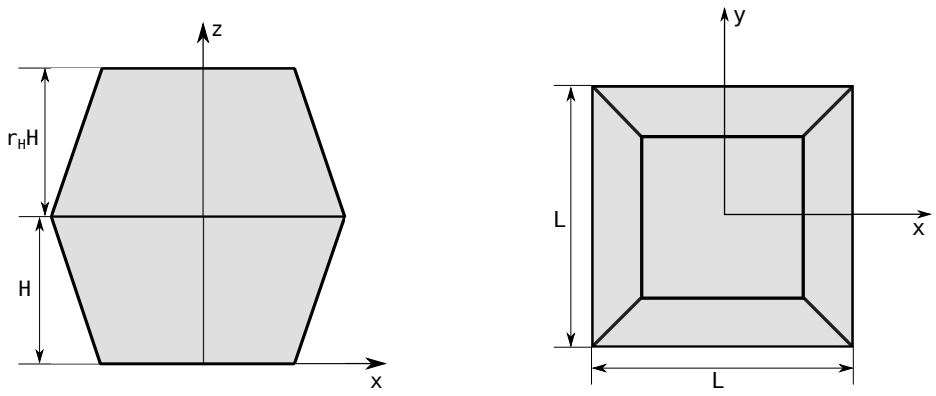
Figure 4.14: Normalized intensity for the form factor of an anisotropic pyramid $|F|^2/V^2$, plotted against (q_y, q_z) and (q_x, q_y) and computed with `FormFactorAnisoPyramid(20.*nanometer, 16.*nanometer, 60.*degree)`.

References Agrees with "AnisoPyramid" form factor of `IsGISAXS` [?], except for different parametrization. ???

4.2.8 Cuboctahedron

Real-space geometry

It is a combination of two pyramids with square bases, as shown in fig. 4.15: the bottom one is upside down with an height H and the top one has the opposite orientation (the standard one) and an height $r_H \times H$.



(a) Side view (b) Top view

Figure 4.15: Sketch of a Cuboctahedron.

Parameters

- length of the shared square base L,
- height H,
- height_ratio r_H,
- α is the angle between the base and the side faces, taken in the middle of the base lines (see fig. 4.11 in Section 4.2.6).

Restrictions on the parameters: $\frac{2H}{L} < \tan(\alpha)$ and $\frac{2r_H H}{L} < \tan(\alpha)$.

Properties

- volume $V = \frac{1}{6}\tan(\alpha)L^3\left[2 - \left(1 - \frac{2H}{L\tan(\alpha)}\right)^3 - \left(1 - \frac{2r_H H}{L\tan(\alpha)}\right)^3\right]$,
- particle surface seen from above $S = L^2$.

Expression of the form factor

$$F(\mathbf{q}, L, H, r_H, \alpha) = \exp(iq_z H)\left[F_{\text{Pyramid}}(q_x, q_y, q_z, L, r_H H, \alpha) + F_{\text{Pyramid}}(q_x, q_y, -q_z, L, H, \alpha))\right]$$

Syntax `FormFactorCuboctahedron(length, height, height_ratio, alpha)`

Example
Figure 4.16 shows the normalized intensity $|F|^2/V^2$, computed with $L = 20$ nm, $H = 13$ nm, $r_H = 0.7$, and $\alpha = 60°$.

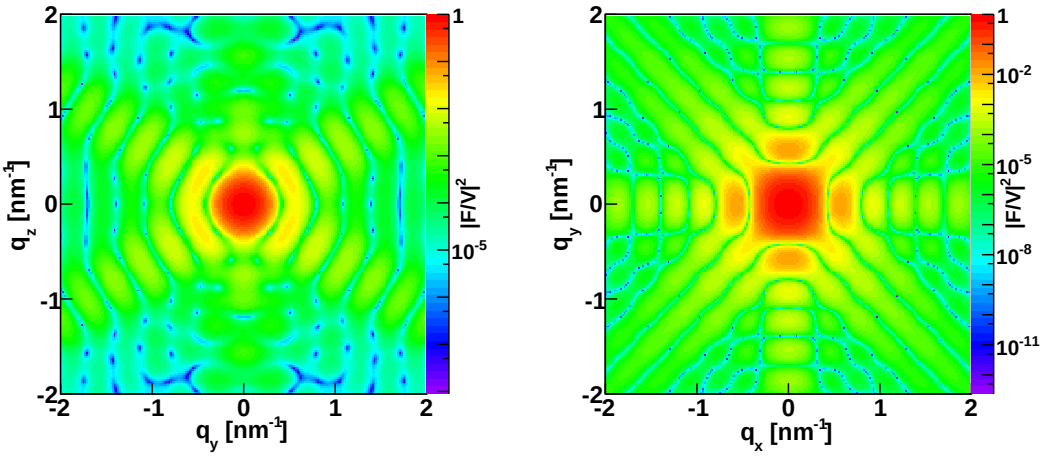
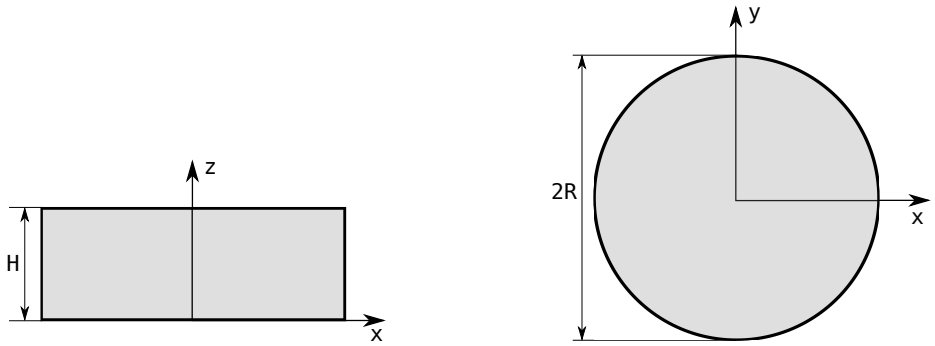


Figure 4.16: Normalized intensity for the form factor of a cuboctahedron plotted against (q_y, q_z) and (q_x, q_y) and computed with `FormFactorCuboctahedron(20.*nanometer, 13.*nanometer, 0.7, 60.*degree)`.

References Agrees with "Cuboctahedron" form factor of `IsGISAXS` [?], except for different parametrization $L = 2R_{\texttt{IsGISAXS}}$.

4.2.9 Cylinder

Real-space geometry
This shape is a right circular cylinder (see fig. 4.17).



(a) Side view (b) Top view

Figure 4.17: Sketch of a Cylinder.

Parameters

- radius of the circular base R,
- height H.

Properties

- volume $V = \pi R^2 H$,
- particle surface seen from above $S = \pi R^2$.

Expression of the form factor

$$F(\mathbf{q}, R, H) = 2\pi R^2 H \,\mathrm{sinc}\left(q_z \frac{H}{2}\right) \exp\left(i q_z \frac{H}{2}\right) \frac{J_1(q_\parallel R)}{q_\parallel R},$$

with $q_\parallel = \sqrt{q_x^2 + q_y^2}$ and $J_1(x)$ is the first order Bessel function of the first kind [**?**].

Syntax `FormFactorCylinder(radius, height)`

Example
Figure 4.18 shows the normalized intensity $|F|^2/V^2$, computed with $R = 8$ nm and $H = 16$ nm.

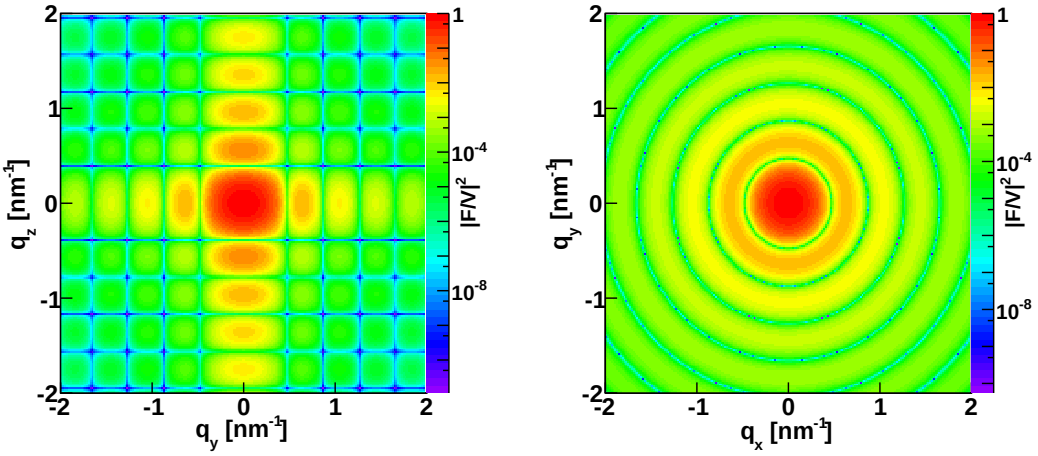
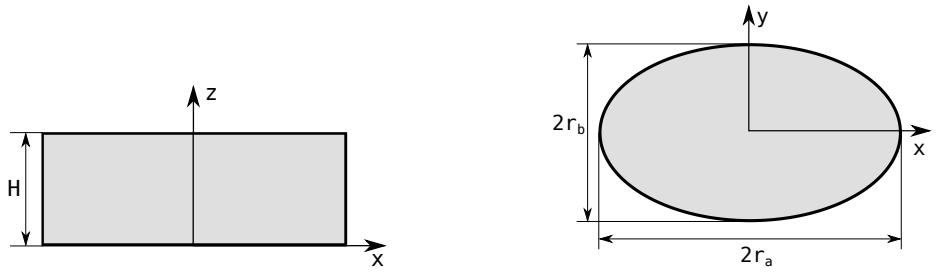


Figure 4.18: Normalized intensity for the form factor of a cylinder plotted against (q_y, q_z) and (q_x, q_y.) It has been computed with `FormFactorCylinder(8.*nanometer, 16.*nanometer)`.

References Agrees with "Cylinder" form factor of `IsGISAXS` [?].

4.2.10 EllipsoidalCylinder

Real-space geometry
This is a cylinder whose cross section is an ellipse.



(a) Side view (b) Top view

Figure 4.19: Sketch of an Ellipsoidal Cylinder.

Parameters

- r_a = half length of the ellipse main axis parallel to x,
- r_b = half length of the ellipse main axis parallel to y,
- height H.

Properties

- volume $V = \pi r_a r_b H$,
- particle surface seen from above $S = r_a r_b$.

Expression of the form factor The total form factor is given by

$$F(\mathbf{q}, R, W, H) = 2\pi r_a r_b H \exp\left(i\frac{q_z H}{2}\right) \mathrm{sinc}\left(\frac{q_z H}{2}\right) \frac{J_1(\gamma)}{\gamma},$$

with $\gamma = \sqrt{(q_x r_a)^2 + (q_y r_b)^2}$ and $J_1(x)$ is the first order Bessel function of the first kind [**?**].

Syntax `FormFactorEllipsoidalCylinder(`r_a`, `r_b`, height)`

Example
Figure 4.20 shows the normalized intensity $|F|^2/V^2$, computed with $r_a = 13$ nm, $r_b = 8$ nm, and $H = 16$ nm.

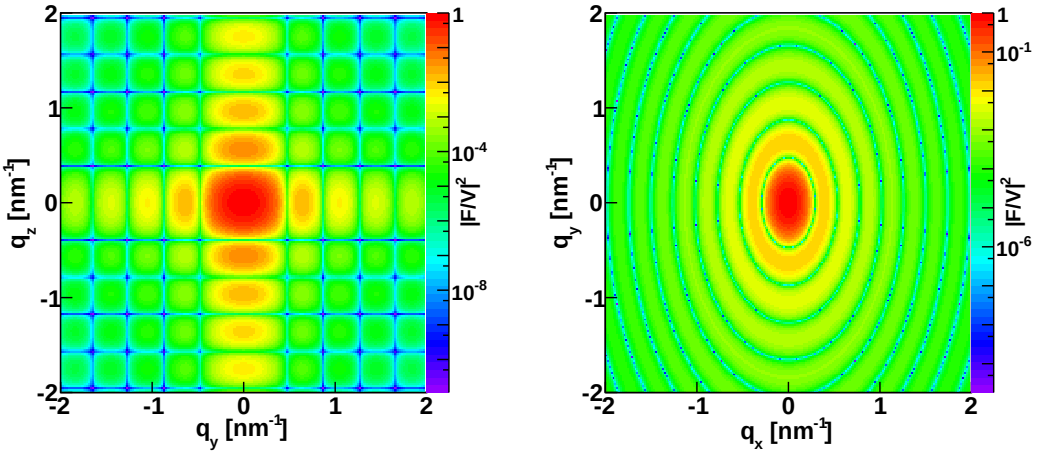
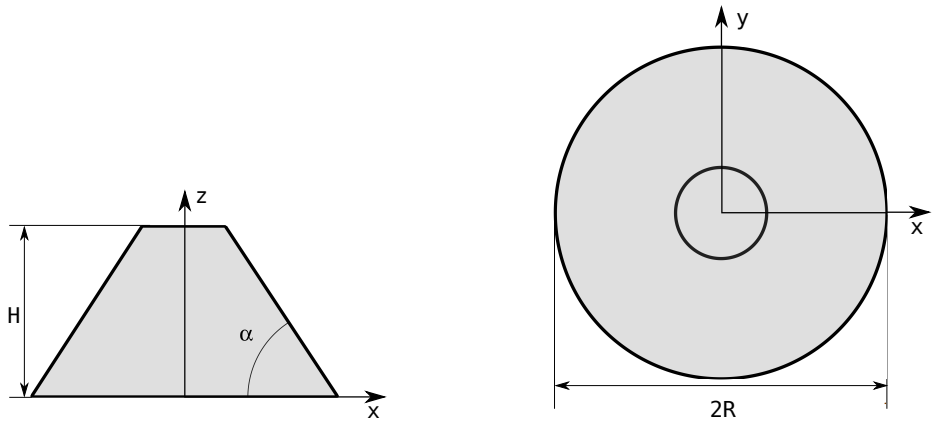


Figure 4.20: Normalized intensity for the form factor of an ellipsoidal cylinder plotted against (q_y, q_z) and (q_x, q_y) and computed with `FormFactorEllipsoidalCylinder(8.*nanometer, 13.*nanometer, 16*nanometer)`.

References Agrees with "Ellipsoid" form factor of `IsGISAXS` [?].

4.2.11 Cone (circular)

Real-space geometry This shape is a truncated cone as shown in fig. 4.21.



(a) Side view (b) Top view

Figure 4.21: Sketch of a Cone.

Parameters

- radius R,
- height H,
- α is the angle between the side and the circular base.

Restrictions on the parameters: $\dfrac{H}{R} < \tan(\alpha)$.

Properties

- volume $V = \dfrac{\pi}{3}\tan(\alpha)R^3\left[1-\left(1-\dfrac{H}{\tan(\alpha)R}\right)^3\right]$,
- particle surface seen from above $S = \pi R^2$.

Expression of the form factor

$$F(\mathbf{q}, R, H, \alpha) = \int_0^H 2\pi R_z^2 \frac{J_1(q_\parallel R_z)}{q_\parallel R_z}\exp(i q_z z)dz,$$

with $R_z = R - \dfrac{z}{\tan\alpha}$, $\mathbf{q}_\parallel = \sqrt{q_x^2 + q_y^2}$ and $J_1(x)$ is the first order Bessel function of the first kind [**?**].

Syntax `FormFactorCone(radius, height, alpha)`.

Example
Figure 4.22 shows the normalized intensity $|F|^2/V^2$, computed with $R = 10$ nm, $H = 13$ nm, and $\alpha = 60°$.

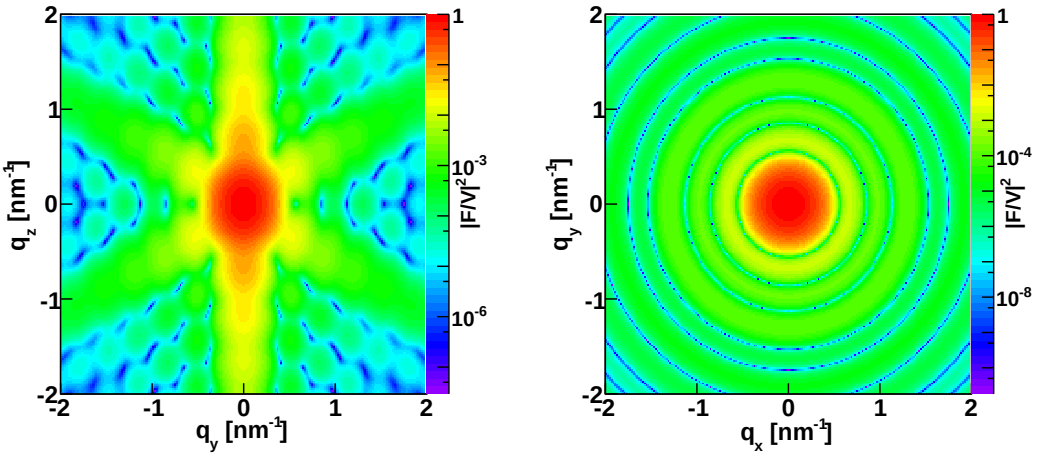
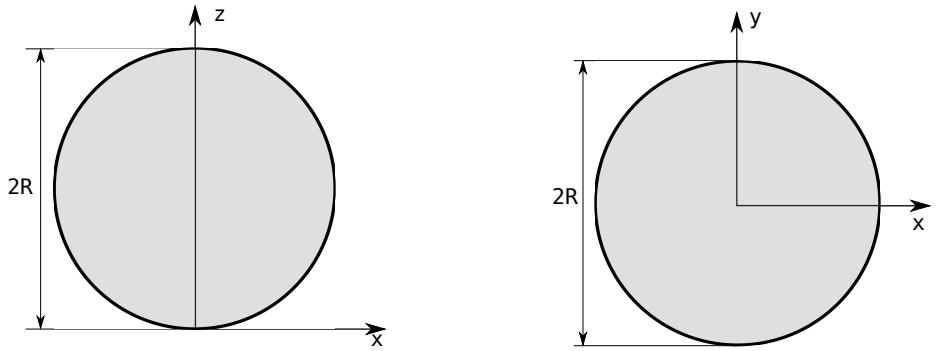


Figure 4.22: Normalized intensity for the form factor of a Cone plotted against (q_y, q_z) and $(q_x, q_y.)$ It has been computed with `FormFactorCone(10.*nanometer,13.*nanometer, 60.*degree)`.

References Agrees with "Cone" form factor of `IsGISAXS` [?].

4.2.12 FullSphere

Real-space geometry
The full sphere is parametrized by its radius R.



(a) Side view (b) Top view

Figure 4.23: Sketch of a Full Sphere.

Parameters radius R.

Properties

- volume $V = \frac{4\pi}{3}R^3$,
- particle surface seen from above $S = \pi R^2$.

Expression of the form factor

$$F(\mathbf{q}, R) = 4\pi R^3 \exp(i q_z R) \frac{\sin(qR) - qR\cos(qR)}{(qR)^3},$$

where $q = \sqrt{q_x^2 + q_y^2 + q_z^2}$.

Syntax `FormFactorFullSphere(radius)`

Example
Figure 4.24 shows the normalized intensity $|F|^2/V^2$, computed with $R = 8$ nm.

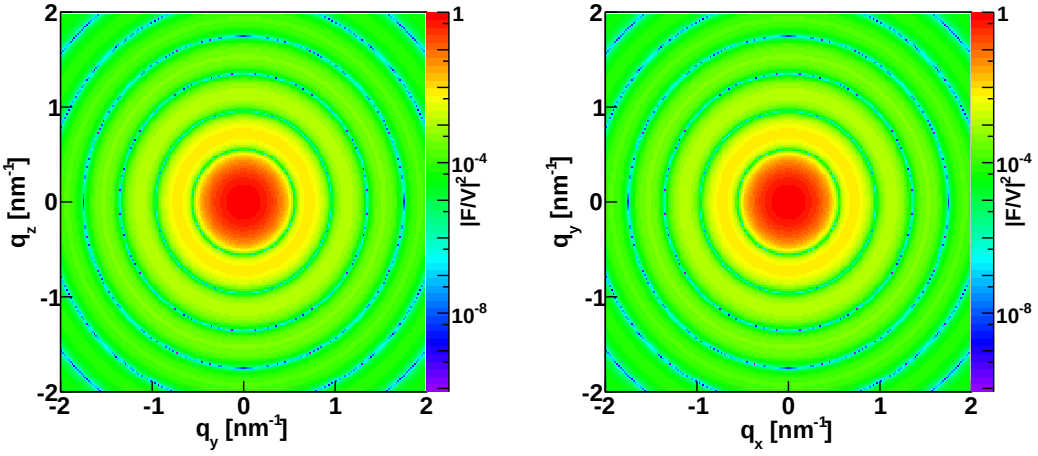
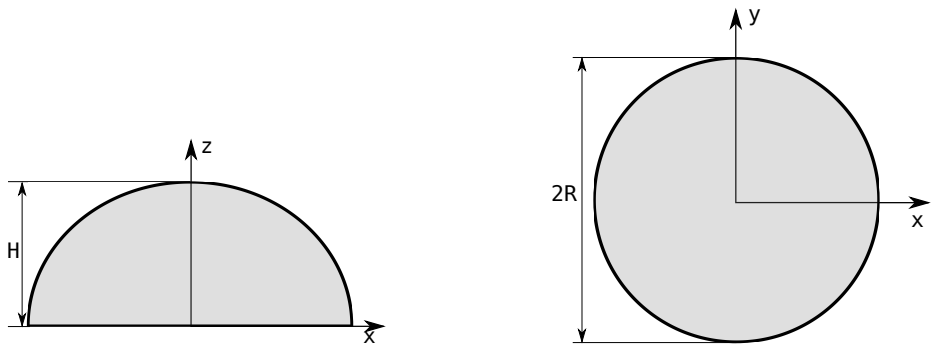


Figure 4.24: Normalized intensity for the form factor of a Full Sphere plotted against (q_y, q_z) and (q_x, q_y) and computed with `FormFactorFullSphere(8.*nanometer)`.

References Agrees with "?? " form factor of `IsGISAXS` [?].

4.2.13 TruncatedSphere

Real-space geometry
This shape is a spherical dome, *i.e.* a portion of a sphere cut off by a plane (perpendicular to z-axis) as shown in fig. 4.25.



(a) Side view (b) Top view

Figure 4.25: Sketch of a Truncated Sphere.

Parameters

- radius R,
- height H.

Restrictions on the parameters: $0 \leq H \leq 2R$.

Properties

- volume $V = \pi R^3 \left[\frac{2}{3} + \frac{H-R}{R} - \frac{1}{3}\left(\frac{H-R}{R}\right)^3\right]$,
- particle surface seen from above $S = \begin{cases} \pi R^2, & H \geq R \\ \pi\left(2RH - H^2\right), & H < R \end{cases}$.

Expression of the form factor

$$F(\mathbf{q}, R, H) = 2\pi \exp[iq_z(H-R)] \int_{R-H}^{R} R_z^2 \frac{J_1(q_\parallel R_z)}{q_\parallel R_z} \exp(iq_z z) dz,$$

with $J_1(x)$ the first order Bessel function of the first kind [?], $q_\parallel = \sqrt{q_x^2 + q_y^2}$, and $R_z = \sqrt{R^2 - z^2}$

Syntax `FormFactorTruncatedSphere(radius, height)`

Example
Figure 4.26 shows the normalized intensity $|F|^2/V^2$, computed with $R = 5$ nm and $H = 7$ nm:

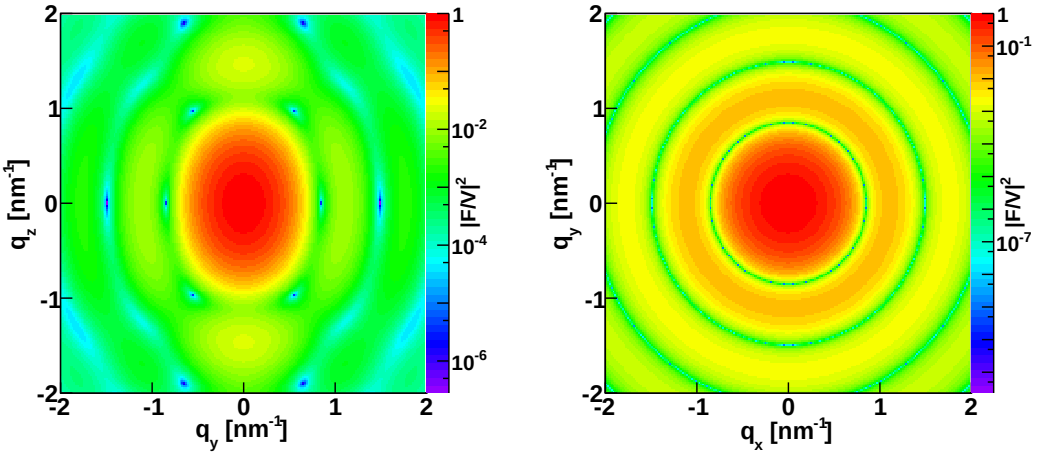
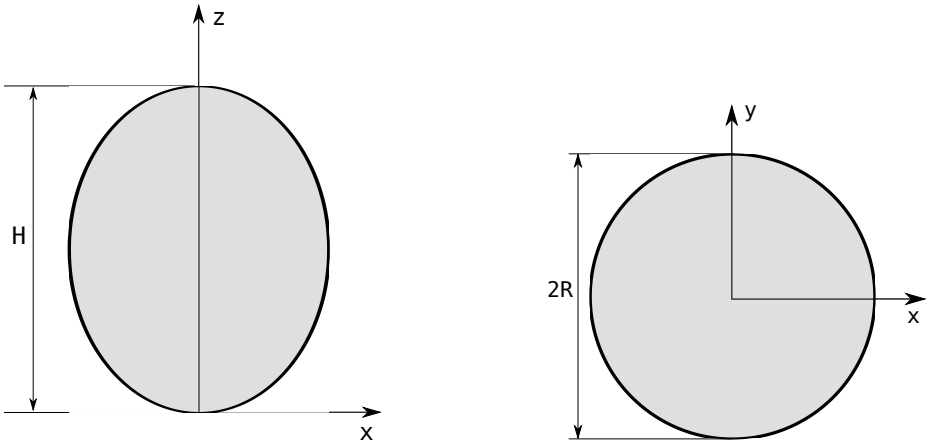


Figure 4.26: Normalized intensity for the form factor of a Truncated Sphere plotted against (q_y, q_z) and (q_x, q_y) and computed with `FormFactorTruncatedSphere(5.*nanometer, 7.*nanometer)`.

References Agrees with "Sphere" form factor of `IsGISAXS` [**?**].

4.2.14 FullSpheroid

Real-space geometry
A full spheroid is generated by rotating an ellipse around the vertical axis (see fig. 4.27).



(a) Side view (b) Top view

Figure 4.27: Sketch of a Full Spheroid.

Parameters

- radius R,
- height H.

Properties

- volume $V = \frac{2}{3}R^2H$,
- particle surface seen from above $S = \pi R^2$.

Expression of the form factor

$$F(\mathbf{q}, R, H) = 4\pi \exp(iq_zH/2) \int_0^{H/2} R_z^2 \frac{J_1(q_\parallel R_z)}{q_\parallel R_z} \cos(q_z z) dz,$$

with $J_1(x)$ the first order Bessel function of the first kind [?], $R_z = R\sqrt{1 - \frac{4z^2}{H^2}}$, $\gamma_z = \sqrt{(q_xR_z)^2 + (q_yR_z)^2}$.

Syntax `FormFactorFullSpheroid(radius,height)`

Example
Figure 4.28 shows the normalized intensity $|F|^2/V^2$, computed with $R = 10$ nm, and $H = 13$ nm.

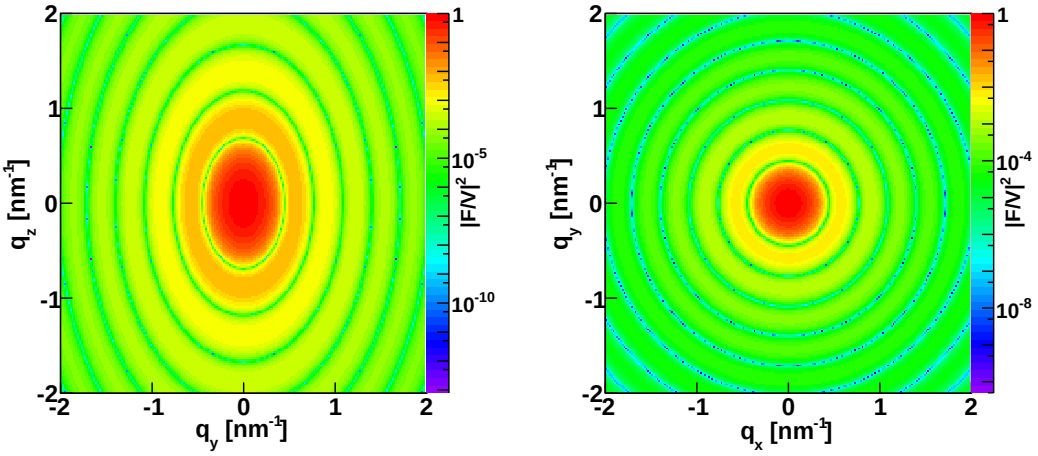
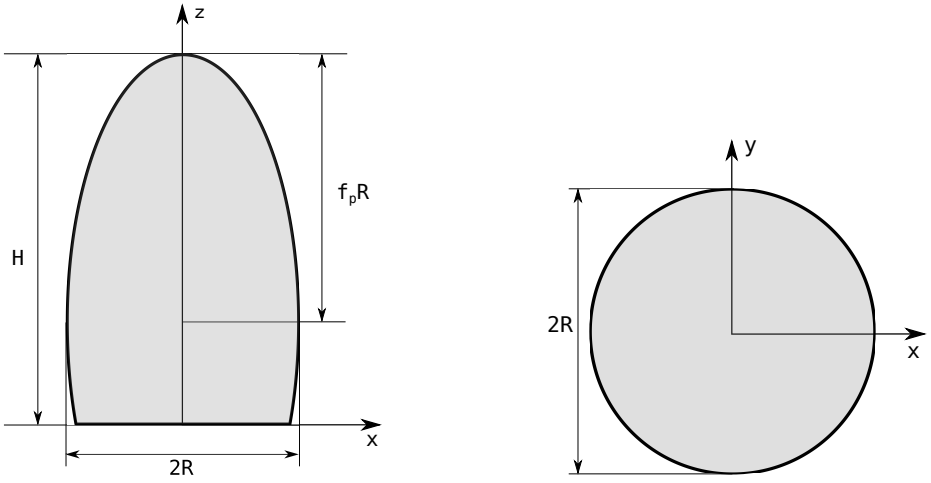


Figure 4.28: Normalized intensity for the form factor of a full spheroid plotted against (q_y, q_z) and (q_x, q_y) and computed with `FormFactorFullSpheroid(10.*nanometer, 13.*nanometer)`.

References Corrected version of the "FullSpheroid" form factor of `IsGISAXS` [**?**].

4.2.15 TruncatedSpheroid

Real-space geometry
This shape is a spheroidal dome: a portion of a full spheroid cut off by a plane perpendicular to the z-axis.



(a) Side view (b) Top view

Figure 4.29: Sketch of a Truncated Spheroid.

Parameters

- radius R,
- height H,
- height_flattening coefficient in the perpendicular direction f_p.

Restrictions on the parameters: $0 < \dfrac{H}{R} < 2f_p$.

Properties

- volume $V = \dfrac{\pi R H^2}{f_p}\left(1 - \dfrac{H}{3f_p R}\right)$,
- particle surface seen from above $S = \begin{cases} \pi R^2, & H \geq f_p R \\ \pi\left(\dfrac{2RH}{f_p} - \dfrac{H^2}{f_p^2}\right), & H < R \end{cases}$.

Expression of the form factor

$$F(\mathbf{q}, R, H, f_p) = 2\pi \exp[iq_z(H - f_p R)] \int_{f_p R - H}^{f_p R} R_z^2 \frac{J_1(q_\parallel R_z)}{q_\parallel R_z} \exp(iq_z z) dz$$

with $J_1(x)$ the first order Bessel function of the first kind [?], $q_\parallel = \sqrt{q_x^2 + q_y^2}$ and $R_z = \sqrt{R^2 - z^2/f_p^2}$.

Syntax `FormFactorTruncatedSpheroid(radius, height, height_flattening)`

Example
Figure 4.30 shows the normalized intensity $|F|^2/V^2$, computed with $R = 7.5$ nm, $H = 9$ nm and $f_p = 1.2$.

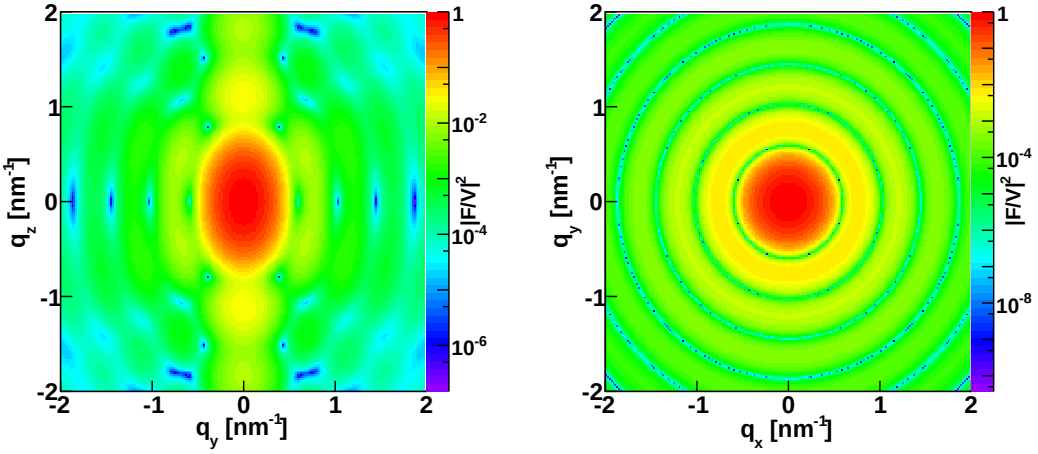
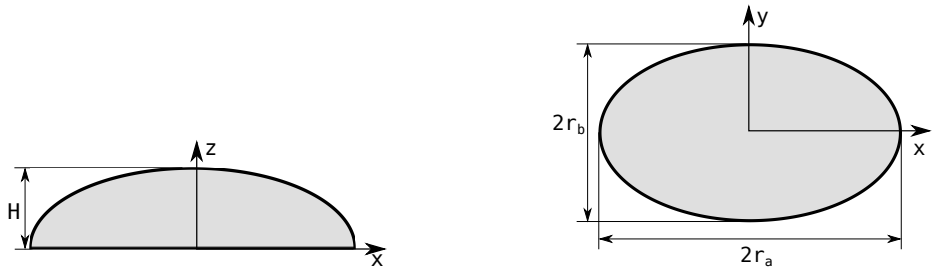


Figure 4.30: Normalized intensity for the form factor of a Truncated Spheroid plotted against (q_z, q_y) and (q_x, q_y) and computed with `FormFactorTruncatedSpheroid(7.5*nanometer, 9.*nanometer, 1.2)`.

References Agrees with "TruncatedSpheroid" form factor of `IsGISAXS` [?].

4.2.16 HemiEllipsoid

Real-space geometry
This shape is a truncated ellipsoid as shown in fig. 4.31.



(a) Side view (b) Top view

Figure 4.31: Sketch of an Hemi-ellipsoid.

Parameters

- r_a = half length of the ellipse main axis parallel to x,
- r_b = half length of the ellipse main axis parallel to y,
- H = height (half length of the vertical main axis of a full ellipsoid).

Properties

- volume $V = \frac{2}{3}\pi r_a r_b H$,
- particle surface seen from above $S = \pi r_a r_b$.

Expression of the form factor

$$F(\mathbf{q}, r_a, r_b, H) = 2\pi \int_0^H r_{a,z} r_{b,z} \frac{J_1(\gamma_z)}{\gamma_z} \exp(i q_z z) dz,$$

with $J_1(x)$ the first order Bessel function of the first kind [**?**], $r_{a,z} = r_a\sqrt{1-\left(\frac{z}{H}\right)^2}$, $r_{b,z} = r_b\sqrt{1-\left(\frac{z}{H}\right)^2}$ and $\gamma_z = \sqrt{(q_x r_{a,z})^2 + (q_y r_{b,z})^2}$.

Syntax `FormFactorHemiEllipsoid(`r_a`, `r_b`, height)`

Example
Figure 4.32 shows the normalized intensity $|F|^2/V^2$, computed with $r_a = 10$ nm, $r_b = 6$ nm and $H = 8$ nm.

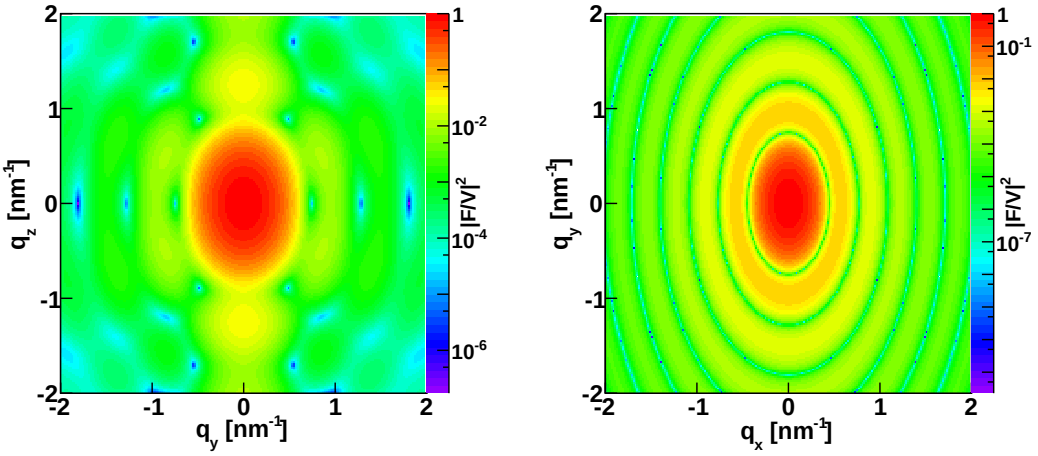
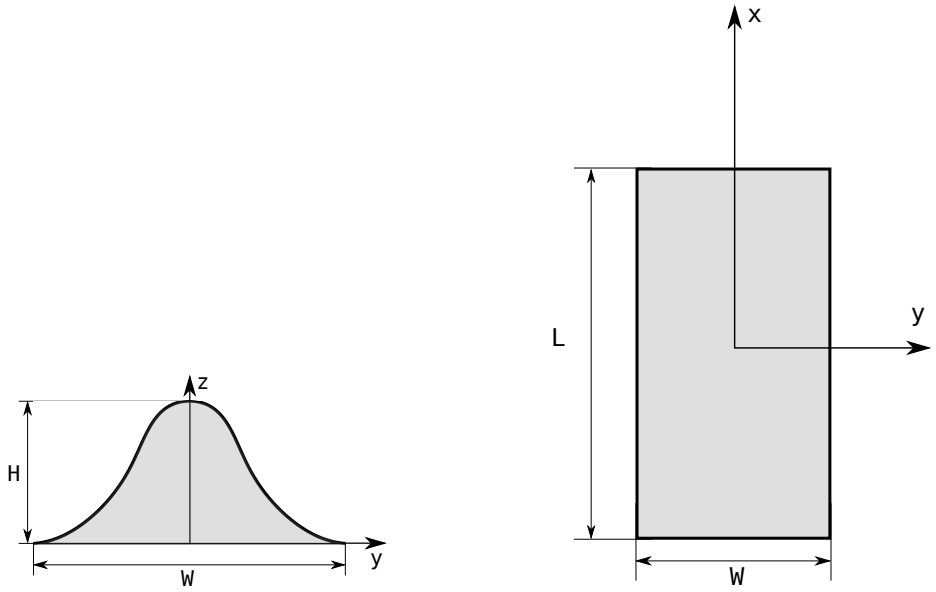


Figure 4.32: Normalized intensity for the form factor of an Hemi-Ellipsoid plotted against (q_y, q_z) and (q_x, q_y) computed with `FormFactorHemiEllipsoid(10.*nanometer, 6.*nanometer, 8.*nanometer)`.

References Called "Anisotropic hemi ellipsoid"in `ISGISAXS`, which seems to have a sign error in the z dependence.

4.2.17 Ripple1 (sinusoidal)

Real-space geometry
This shape has a sinusoidal profile (see fig. 4.33).



(a) Side view (b) Top view

Figure 4.33: Sketch of a Ripple1.

Parameters

- length L,
- width W,
- height H.

Properties

- volume $V = \dfrac{LWH}{2}$,
- particle surface seen from above $S = LW$.

Expression of the form factor

$$F(\mathbf{q}, L, W, H) = L \cdot \frac{W}{\pi} \cdot \operatorname{sinc}\left(\frac{q_x L}{2}\right) \times \int_0^H dz \arccos\left(\frac{2z}{H} - 1\right) \operatorname{sinc}\left[\frac{q_y W}{2\pi} \arccos\left(\frac{2z}{H} - 1\right)\right] \exp\left(i q_z z\right),$$

where arccos is the arc cosine (*i.e.* the inverse operation of cosine).

Syntax `FormFactorRipple1(length, width, height)`

Example
Figure 4.34 shows the normalized intensity $|F|^2/V^2$, computed with $L = 27$ nm, $W = 20$ nm and $H = 14$ nm.

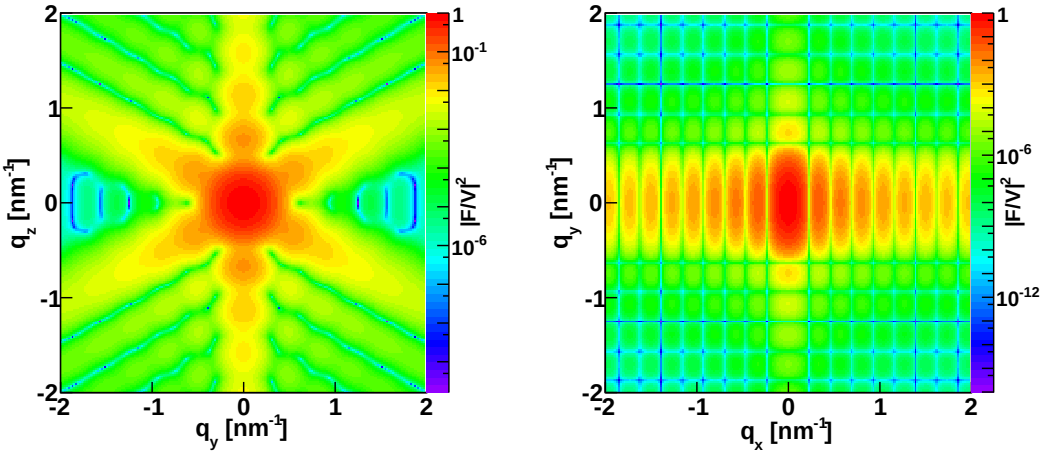
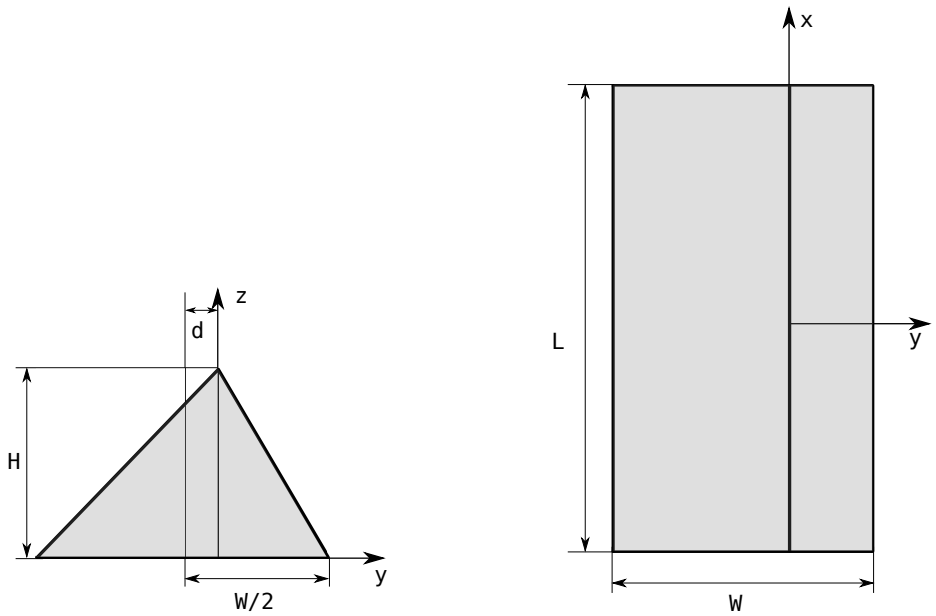


Figure 4.34: Normalized intensity for the form factor of a ripple1 $|F|^2/V^2$, plotted against (q_y, q_z) and (q_x, q_y) computed with `FormFactorRipple1(27.*nanometer, 20.*nanometer, 14.*nanometer)`.

4.2.18 Ripple2 (saw-tooth)

Real-space geometry
This shape has an asymmetric saw-tooth profile.



(a) Side view (b) Top view

Figure 4.35: Sketch of a Ripple2.

Parameters

- length L,
- width W,
- height H,
- asymmetry d.

Restriction on the parameters: $|d| < \frac{W}{2}$.

Properties

- volume $V = \dfrac{LWH}{2}$,
- particle surface seen from above $S = LW$.

Expression of the form factor

$$F(\mathbf{q}, L, W, H, d) = LW\,\mathrm{sinc}\left(\frac{q_x L}{2}\right) \times \int_0^H \left(1-\frac{z}{H}\right)\mathrm{sinc}\left[\frac{q_y W}{2}\left(1-\frac{z}{H}\right)\right]\exp\left\{i\left[q_z z - q_y d\left(1-\frac{z}{H}\right)\right]\right\} dz$$

Syntax `FormFactorRipple2(length, width, height, asymmetry)`

Examples Figure 4.36 shows the normalized intensity $|F|^2/V^2$, computed with $L =$ 36 nm, $W = 25$ nm, $H = 14$ nm, and $d = 3$ nm.

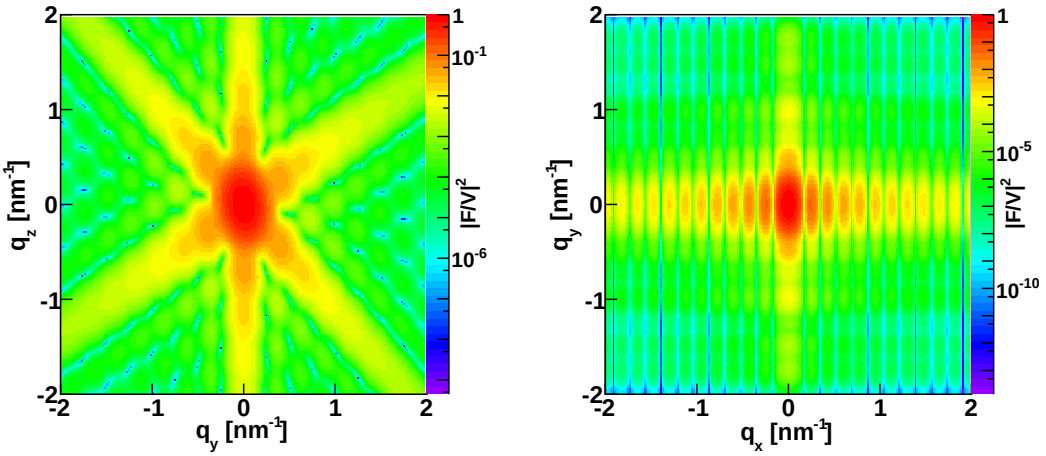


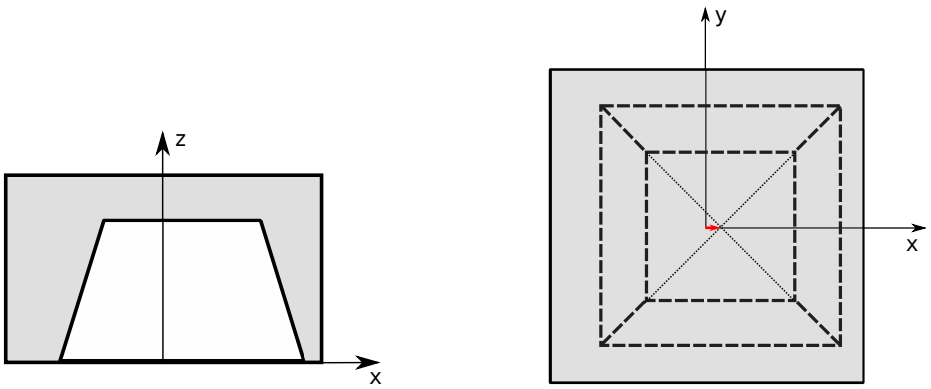
Figure 4.36: Normalized intensity for the form factor of a ripple2 plotted against (q_y, q_z) and (q_x, q_y) computed with `FormFactorRipple2(36.*nanometer, 25.*nanometer, 14.*nanometer, 3.*nanometer)`.

4.3 Core-shell particles

To generate a core-shell particle, the combination is performed using the following command:
`ParticleCoreShell(shell_particle, core_particle, relative_core_position)`,
where `shell_particle` and `core_particle` are the outer and inner parts of the core-shell particle, respectively. They refer to one of the form factors defined previously and to an associated material. For example, for the outer part,
`shell_particle=Particle(material_shell, outer_form_factor)`,
where `material_shell` is the material of the shell and `outer_form_factor` is the shape of the outer part (cf. listing 4.1).

`relative_core_position` defines the position of the inner shape with respect to the outer one; it is defined with respect to the center of the base of the particular form factor. An example in fig. 4.37 shows a core shell particle made of a box for the outer part and of a shifted pyramidal shape for the inner one.

Figure 4.38 displays the output intensity scattered in the Born Approximation using the code listed in 4.1 to generate the core-shell particle.



(a) Side view (b) Top view

Figure 4.37: Example of a core-shell particle composed of a box with a pyramidal inset. The relative core shell position is marked by the positions of the centers of the bases.

Listing 4.1: `Python` script to create a core-shell particle made of a box with a pyramidal shifted inset.

```
outer_ff = FormFactorBox(16.0*nanometer, 16.0*nanometer, 8.0*
    nanometer)
inner_ff = FormFactorPyramid(12.0*nanometer, 7.0*nanometer,
    60.0*degree)
shell_particle = Particle(m_shell, outer_ff)
core_particle = Particle(m_core, inner_ff)
core_position = kvector_t(1.5, 0.0, 0.0)

particle = ParticleCoreShell(shell_particle, core_particle,
    core_position)
```

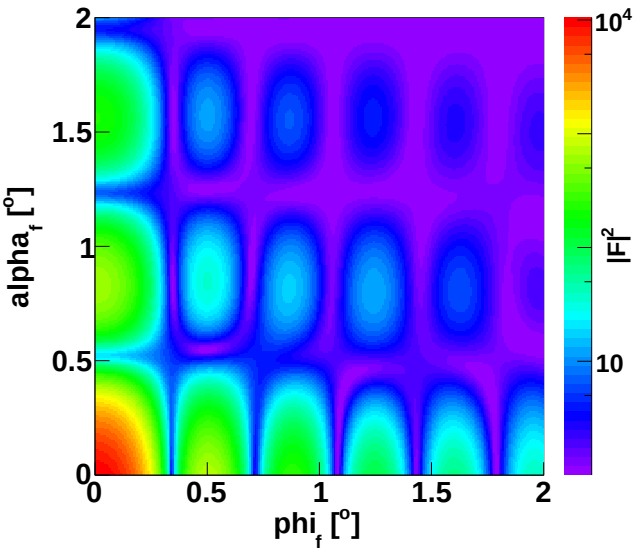


Figure 4.38: Intensity map of a core-shell form factor in Born Approximation using `FormFactorBox(16*nanometer, 16*nanometer, 8*nanometer)` and `FormFactorPyramid(12*nanometer, 7*nanometer, 60*degree)` for the outer and inner shells, respectively. The core particle is shifted by 1.5 nm in the x-direction with respect to the center of the outer shell. The sample used to generate the particle is listed in 4.1. There is no substrate and no interference between the particles.

4.4 Rotation of particles

The particles can be rotated in a different direction by using one of the following transformations: `CreateRotateX(`θ`)`, `CreateRotateY(`θ`)`, `CreateRotateZ(`θ`)`, where capital X, Y, Z mark rotations around the associated axis and θ is the angle of rotation

from this axis. For example, the following `Python` script shows how to rotate a pyramid by 45° around the z-axis:

```
pyramid_ff = FormFactorPyramid(10*nanometer, 5*nanometer,
    deg2rad(54.73 ) )
pyramid = Particle(m_particle, pyramid_ff)
angle_around_z = 45.*degree
transform = Transform3D.createRotateZ(angle_around_z)
particle_layout = ParticleLayout()
particle_layout.addParticle(pyramid, transform)
```

4.5 Polydispersity

Chapter 5

Particle Assemblies

5.1 Particle distributions

5.1.1 Position of the problem

This section describes how assemblies of particles and layers of materials contribute to the scattering cross-section *i.e.* the way their spatial distributions, the distribution of shapes and their correlations or the layers' roughness can influence the output intensity.

The samples generated with `BornAgain` are made of different layers of materials characterized by their thicknesses, refractive indices, and possible surface roughnesses. Except for the thickness, the other dimensions of the layers are infinite.
Particles can be embedded in or deposited on the top of any layers. Those particles are characterized by their shapes, refractive indices, their spatial distribution and concentration in the sample. The influence of the particles' shapes is described by the form factors. When the particles are densely packed, the distance relative to each other becomes of the same order as the particles' sizes. The radiation scattered from these various particles are going to interfere together.
We are first going to give a short overview of the theory involved, mostly in order to define the terminology. For a more complete theoretical description, the user is referred to, for example, reference [**?**]. hen we are going to describe how the interference features, the form factors and the characteristics of the material layers have been implemented in `BornAgain` and give some examples.

TO MERGE IN: The scattering length density can be written as:

$$\rho_s(\mathbf{r}) = \sum_i \rho_{s,i} S^i(\mathbf{r}) \otimes \delta(\mathbf{r} - \mathbf{R}^i),$$

with $\rho_{s,i}$ the scattering length density of particle i. The cross-section equation (2.23)

then becomes:

$$N\frac{d\sigma}{d\Omega}(\mathbf{q}) = \left|\sum_i F^i(\mathbf{q})\exp(i\mathbf{q}\cdot\mathbf{R}^i)\right|^2$$
$$= \left\{\sum_i \left|F^i(\mathbf{q})\right|^2 + \sum_{i\neq j} F^i(\mathbf{q})F^{j*}(\mathbf{q})\exp\left[i\mathbf{q}\cdot(\mathbf{R}^i-\mathbf{R}^j)\right]\right\}.$$

In the last expression, the formfactors $F^i(\mathbf{q})$ are the Fourier transforms of the shape functions, including their scattering length densities:

$$F^i(\mathbf{q}) \equiv \int d^3\mathbf{r}\rho_{s,i}S^i(\mathbf{r})\exp(i\mathbf{q}\cdot\mathbf{r}).$$

5.1.2 Collection of particles

Let us consider the general geometry of a scattering experiment. An incident neutron with a wave vector $\mathbf{k}_i$ is scattered in a new direction $\mathbf{k}_f$ after interacting with a particle. This scattering occurs in a cone of solid angle $d\Omega$ around the direction of the scattered wave vector $\mathbf{k}_f$. Considering a set of N particles labeled with index i, located at $\mathbf{R}_i$ and having shapes $S_i(\mathbf{r})$ ($S_i = 0$ outside the particle and 1 inside), occupying a total volume V, the differential cross-section per particle is given by:

$$\frac{d\sigma}{d\Omega}(\mathbf{q}) = \frac{1}{N}\left\{\sum_i \left|F_i(\mathbf{q})\right|^2 + \sum_{i\neq j} F_i(\mathbf{q})F_j^*(\mathbf{q})\exp\left[i\mathbf{q}\cdot(\mathbf{R}_i-\mathbf{R}_j)\right]\right\}.$$

where $\mathbf{q} = \mathbf{k}_i - \mathbf{k}_f$ is the wave vector transfer and F_i is the form factor of particle i (see Section **??** for a description).

Since in most experimental conditions only the statistical properties of the particles are known, one can consider the probabilistic value of this cross-section *i.e.* its expectation value. Assuming that the particles' shapes are determined by their class α, with the abundance ratio $p_\alpha \equiv N_\alpha/N$, and defining the particle density as $\rho_V \equiv N/V$, the expectation value becomes:

$$\left\langle\frac{d\sigma}{d\Omega}(\mathbf{q})\right\rangle = \sum_\alpha p_\alpha\left|F_\alpha(\mathbf{q})\right|^2 + \frac{\rho_V}{V}\sum_{\alpha,\beta} p_\alpha p_\beta F_\alpha(\mathbf{q})F_\beta^*(\mathbf{q})$$
$$\times\iint_V d^3\mathbf{R}_\alpha d^3\mathbf{R}_\beta \mathcal{G}_{\alpha,\beta}\left(\mathbf{R}_\alpha,\mathbf{R}_\beta\right)\exp\left[i\mathbf{q}\cdot(\mathbf{R}_\alpha-\mathbf{R}_\beta)\right],$$

where $\mathcal{G}_{\alpha,\beta}\left(\mathbf{R}_\alpha,\mathbf{R}_\beta\right)$ is called the *partial pair correlation function*. It represents the normalized probability of finding particles of type α and β in positions $\mathbf{R}_\alpha$ and $\mathbf{R}_\beta$ respectively.

TO MERGE IN: This expression can be split into a diffuse part, which by definition should be zero for the case of only one particle type, and a coherent part, resulting from the coherent superposition of scattering amplitudes for particles at different positions:

$$\left\langle\frac{d\sigma}{d\Omega}(\mathbf{q})\right\rangle = I_d(\mathbf{q}) + \left\langle F_\alpha(\mathbf{q})S_{\alpha\beta}(\mathbf{q})F_\beta^*(\mathbf{q})\right\rangle_{\alpha\beta},$$

where

$$I_d(\mathbf{q}) \equiv \left\langle \left|F_\alpha(\mathbf{q})\right|^2 \right\rangle_\alpha - \left|\left\langle F_\alpha(\mathbf{q})\right\rangle_\alpha\right|^2,$$
$$S_{\alpha\beta}(\mathbf{q}) \equiv 1 + \rho_V \int_V d^3\mathbf{r}\mathscr{G}_{\alpha\beta}(\mathbf{r}) \exp\left[i\mathbf{q}\cdot\mathbf{r}\right].$$

$S_{\alpha\beta}(\mathbf{q})$ is called the *interference function* and $\langle\ldots\rangle_\alpha$ is the expectation value over the classes $\{\alpha\}$.

TO MERGE IN: More precisely, the probability density for finding a particle α at position $\mathbf{R}_\alpha$ and another one of type β at $\mathbf{R}_\beta$ is given by:

$$\mathscr{P}(\alpha, \mathbf{R}_\alpha; \beta, \mathbf{R}_\beta) \equiv \rho_V^2 p_\alpha p_\beta \mathscr{G}_{\alpha,\beta}\left(\mathbf{R}_\alpha, \mathbf{R}_\beta\right).$$

5.2 Horizontal particle distributions

To proceed further, when the morphology and topology are not exactly known, some hypotheses need to be made since the correlation between the kinds of scatterers and their relative positions included in the pair correlation functions are difficult to estimate. Several options are available:

Decoupling approximation (DA) neglects all correlations. It supposes that the particles are positioned in a way that is completely independent on their kinds (shapes, sizes). An example is given in figure 5.1. Thus the kind of scattering objects and their positions are not correlated and the partial pair correlation function is independent of the particle class α. We can therefore replace $\mathscr{G}_{\alpha\beta}\left(\mathbf{R}_{\alpha\beta}\right)$ by $g(\mathbf{R}_{\alpha\beta})$.

This leads to the following expression of the scattering cross-section:

$$\left\langle \frac{d\sigma}{d\Omega}(\mathbf{q}) \right\rangle = I_d(\mathbf{q}) + \left|\left\langle F_\alpha(\mathbf{q})\right\rangle_\alpha\right|^2 \times S(\mathbf{q}),$$

where I_d is the diffuse part of the scattering. It is the signature of the fluctuations of shapes, sizes or orientations of the particles; its maximum is located in $q_\parallel = 0$. In the second term of the expression of the scattering cross-section, $S(\mathbf{q})$ is the interference function and is given by

$$S(\mathbf{q}) = 1 + \rho_V \int_V d^3\mathbf{r} g(\mathbf{r}) \exp\left[i\mathbf{q}\cdot\mathbf{r}\right].$$

In concentrated systems, DA breaks down because of correlations. One solution is to reintroduce some correlations between particles sizes and distributions using, for example, the size spacing correlation approximation described below.

Local monodisperse approximation (LMA) partially accounts for some coupling between the positions and the kinds of the particles [**?**]. It requires a subdivision of the layers of particles into monodisperse domains. The contributions of these subdomains are then incoherently summed up and weighted by the size-shape probabilities.

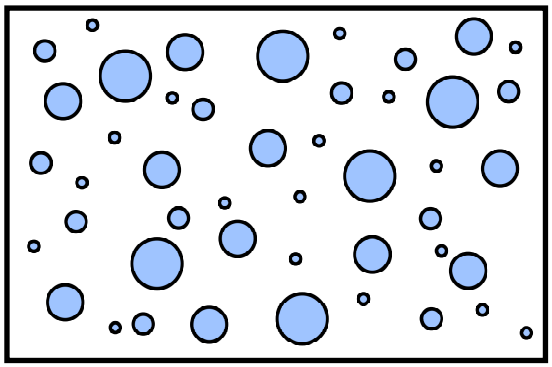


Figure 5.1: Sketch of a collection of particles deposited on a substrate whose scattering could be described by the decoupling approximation.

In this approximation, a particle is supposed to be surrounded by particles of the same size and shape, within the coherence length of the input beam (see fig. 5.2). The scattering cross-section is expressed as

$$\left\langle \frac{d\sigma}{d\Omega}(\mathbf{q}) \right\rangle \simeq \left\langle \left|F_\alpha(\mathbf{q})\right|^2 S_\alpha(\mathbf{q}) \right\rangle_\alpha .$$

Contrary to the Decoupling Approximation, the Local Monodisperse Approximation can account for particle class/size/shape-dependent pair correlation functions by having distinct interference functions $S_\alpha(\mathbf{q})$.

One has to remember that in most cases, this approximation corresponds to an unphysical description of the investigated systems.

DA and LMA separate the contributions of the form factors and of the interference function. For disordered systems DA and LMA give the same result as the scattering vector gets larger *i.e.* the scattered intensity is dominated by the contribution of the form factor.

Terminology

For collections of particles, the scattered intensity contains contributions from neighboring particles. This additional pattern can be called the structure factor, the interference function or even in crystallography, the lattice factor. In this manual, we use the term ”interference function” or interferences.

5.2.1 Layout of particles

5.2.1.1 The uncorrelated or disordered lattice

For very diluted distributions of particles, the particles are too far apart from each other to lead to any interference between the waves scattered by each of them. In this case the interference function is equal to 1. The scattered intensity is then entirely determined by the form factors of the particles distributed in the sample.

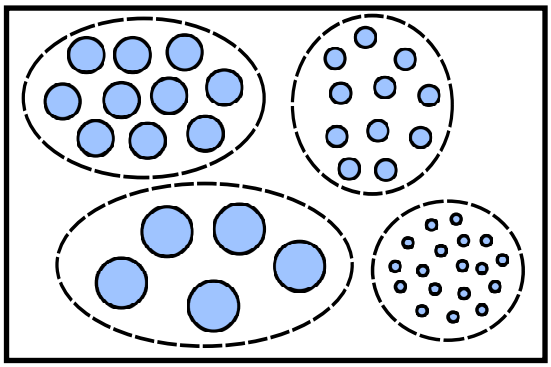


Figure 5.2: Sketch of a collection of particles deposited on a substrate whose scattering could be described by the local monodisperse correlation approximation. The dashed areas mark the coherent domains. In this case, the total scattering intensity is the incoherent sum from all these domains.

5.2.1.2 The regular lattice

The particles are positioned at regular intervals generating a layout characterized by its base vectors **a** and **b** (in direct space) and the angle between these two vectors. This lattice can be two or one-dimensional depending on the characteristics of the particles. For example when they are infinitely long, the implementation can be simplified and reduced to a "pseudo" 1D system.

The ideal paracrystal

A paracrystal, whose notion was developed by Hosemann[**?**], allows fluctuations of the lengths and orientations of lattice vectors. Paracrystals can be defined as distorted crystals in which the crystalline order has not disappeared and for which the behavior of the interference functions at small angles is coherent. It is a transition between the regular lattice and the disordered state.

For example, in one dimension, a paracrystal is generated using the following method. First we place a particle at the origin. The second particle is put at a distance x with a density probability $p(x)$ that is peaked at a mean value D: $\int_{-\infty}^{\infty} p(x)dx = 1$ and $\int_{-\infty}^{\infty} xp(x)dx = D$. The third one is added at a distance y from the second site using the same rule with a density probability $p_2(y) = \int_{-\infty}^{\infty} p(x)p(y-x)dx = p \otimes p(y)$.
With such a method, the pair correlation function $g(x)$ is built step by step. Its expression and the one of its Fourier transform, which is the interference function are

$$g(x) = \delta(x) + p(x) + p(x) \otimes p(x) + \ldots + p(-x) + \ldots \text{ and } S(q) = \mathrm{Re}\left(\frac{1+P(q)}{1-P(q)}\right),$$

where $P(q)$ is the Fourier transform of the density probability $p(x)$.

Example For a probability distribution function is Gaussian:

$$p(x) = \frac{1}{\omega\sqrt{2\pi}} \exp\left(-\frac{(x-D)^2}{\omega^2}\right), \qquad P(q) = \exp\left(-\frac{q^2\omega^2}{2}\right)\exp(iqD).$$

The interference function of a one-dimensional paracrystal is given by

$$S(q) = \mathrm{Re}\left(\frac{1+\Phi(q)}{1-\Phi(q)}\right), \quad \text{where} \quad \Phi(q) = P(q)\exp\left(-\frac{D}{\Lambda}\right),$$

where Λ is a damping length used in order to introduce some finite-size effects.

Figure 5.3 shows the evolution of $S(q)$ for different values of ω/D.

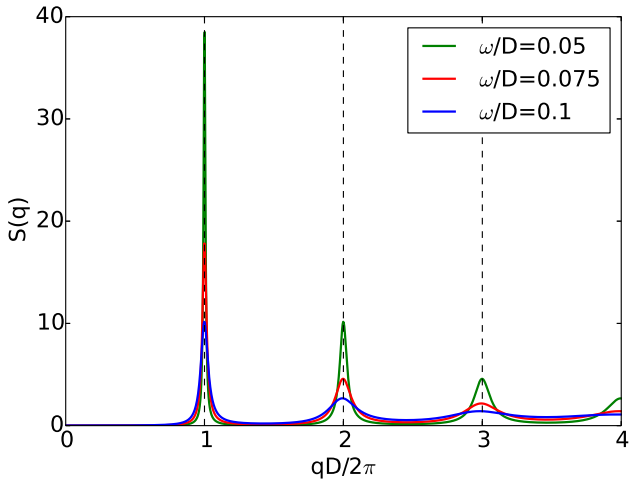


Figure 5.3: Interference function of a 1D Gaussian paracrystal plotted for different values of ω/D. The peaks broaden with a decreasing amplitude as ω/D increases. This shows the transition between an ordered and a disordered states.

In two dimensions, the paracrystal is constructed on a pseudo-regular lattice with base vectors **a** and **b** using the following conditions for the densities of probabilities:
$\int p_{\mathbf{a}}(\mathbf{r})d^2\mathbf{r} = \int p_{\mathbf{b}}(\mathbf{r})d^2\mathbf{r} = 1$, $\int \mathbf{r}p_{\mathbf{a}}(\mathbf{r})d^2\mathbf{r} = \mathbf{a}$, $\int \mathbf{r}p_{\mathbf{b}}(\mathbf{r})d^2\mathbf{r} = \mathbf{b}$.
In the ideal case the deformations along the two axes are decoupled and each unit cell should retain a parallelogram shape. The interference function is given by
$S(q_\parallel) = \prod_{k=a,b}\mathrm{Re}\left(\frac{1+P_k(q_\parallel)}{1-P_k(q_\parallel)}\right)$ with P_k the Fourier transform of p_k, $k = a, b$.

Probability distributions

The scattering by an ordered lattice gives rise to a series of Bragg peaks situated at the nodes of the reciprocal lattice. Any divergence from the ideal crystalline case modifies the output spectrum by, for example, widening or attenuating the Bragg peaks. The influence of these ”defects” can be accounted for in direct space by using correlation

functions or by truncating the lattice or, in reciprocal space with structure factors or interference functions by convoluting the scattered peaks with a function which could reproduce the experimental shapes.

5.2.1.3 Size-Spacing Correlation Approximation

TO MERGE IN: introduces correlations between polydisperse particles, more precisely between the size of the particles and their mutual spacing. A classical example would consist in particles whose closest-neighbor spacing depends linearly on the sum of their respective sizes [?], as illustrated in fig. 5.4.

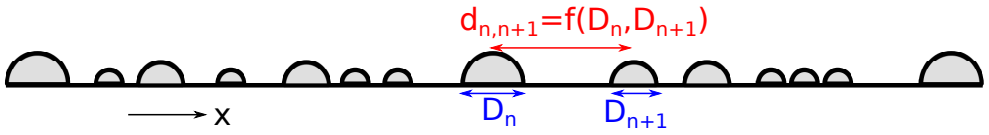


Figure 5.4: Sketch of a 1D distributed collection of particles, whose scattering could be described by the size-spacing correlation approximation: the distance between two particles depends on their sizes.

In the Size-Spacing Correlation Approximation, a correlation is assumed between the shape/size of the particles and their mutual spacing. A classical example would consist of particles whose closest-neighbor spacing depends linearly on the sum of their respective sizes. The following discussion of this type of correlation is inspired by [?]

The scattered intensity can also be calculated as the Fourier transform of the Patterson function, which is the autocorrelation of the scattering length density:

$$\mathscr{P}(\mathbf{r}) \equiv \sum_{ij} S_i(-\mathbf{r}) \otimes S_j(\mathbf{r}) \otimes \delta(\mathbf{r} + \mathbf{r}_i - \mathbf{r}_j).$$

For a sample where only the statistical properties of particle positions and shape/size are known, the scattered intensity per scattering particle becomes average over an ensemble of the Fourier transform of the Patterson function:

$$I(\mathbf{q}) = \frac{1}{N} \langle \mathscr{F}(\mathscr{P}(\mathbf{r})) \rangle ,$$

where $\mathscr{F}$ denotes the Fourier transform.

The ensemble averaged Patterson function will be denoted as:

$$Z(r) \equiv \frac{1}{N} \langle \mathscr{P}(\mathbf{r}) \rangle .$$

In the case of systems where the particles are aligned in one dimension, this autocorrelation function can be further split into nearest neighbor probabilities. First, it is split into terms for negative, zero or positive distance:

$$Z(\mathbf{r}) \equiv z_0(\mathbf{r}) + z_+(\mathbf{r}) + z_-(\mathbf{r}).$$

Taking x as the coordinate in the direction in which the particles are arranged and s as an orthogonal coordinate ($\mathbf{r} \equiv (x, s)$), one obtains:

$$
\begin{aligned}
z_0(\mathbf{r}) &= \sum_{\alpha_0} p(\alpha_0) S_{\alpha_0}(-x,-s) \otimes S_{\alpha_0}(x,s) \\
z_+(\mathbf{r}) &= \sum_{\alpha_0\alpha_1} p(\alpha_0,\alpha_1) S_{\alpha_0}(-x,-s) \otimes S_{\alpha_1}(x,s) \otimes P_1(x|\alpha_0\alpha_1) \\
&\quad + \sum_{\alpha_0\alpha_1\alpha_2} p(\alpha_0,\alpha_1,\alpha_2) S_{\alpha_0}(-x,-s) \otimes S_{\alpha_2}(x,s) \otimes P_1(x|\alpha_0\alpha_1) \otimes P_2(x|\alpha_0\alpha_1\alpha_2) \\
&\quad + \cdots \\
z_-(\mathbf{r}) &= z_+(-\mathbf{r}),
\end{aligned}
$$

where $p(\alpha_0,\ldots,\alpha_n)$ denotes the probability of having a sequence of particles of the indicated sizes/shapes and $P_n(x|\alpha_0\ldots\alpha_n)$ is the probability density of having a particle of type α_n at a (positive) distance x of its nearest neighbor of type α_{n-1} in a sequence of the given order.

In the Size-Spacing Correlation Approximation, one assumes that the particle sequence probabilities are just a product of their individual fractions:

$$p(\alpha_0,\ldots,\alpha_n) = \prod_i p(\alpha_i),$$

and the nearest neighbor distance distribution is dependent only on the two particles involved:

$$P_n(x|\alpha_0\ldots\alpha_n) = P_1(x|\alpha_{n-1}\alpha_n).$$

Furthermore, the distance distribution $P_1(x|\alpha_0\alpha_1)$ depends on the particle sizes/shapes only through its mean value D:

$$P_1(x|\alpha_0\alpha_1) = P_0(x - D(\alpha_0,\alpha_1)),$$

where $D(\alpha_0,\alpha_1) = D_0 + \kappa\,[\Delta R(\alpha_0) + \Delta R(\alpha_1)]$, with $\Delta R(\alpha_i)$ the deviation of a size parameter of particle i with respect to the mean over all particles sizes/shapes and κ the coupling parameter.

In momentum space, the sum of convolutions can be written as a geometric series, which can be exactly calculated to be:

$$I(\mathbf{q}) = \left\langle \left|F_\alpha(\mathbf{q})\right|^2 \right\rangle_\alpha + 2\,\mathrm{Re}\left\{ \widetilde{\mathscr{F}_\kappa}(\mathbf{q}) \widetilde{\mathscr{F}_\kappa^*}(\mathbf{q}) \cdot \frac{\Omega_\kappa(\mathbf{q})}{\tilde{p}_{2\kappa}(\mathbf{q})\left[1 - \Omega_\kappa(\mathbf{q})\right]} \right\}, \tag{5.1}$$

with

$$
\begin{aligned}
\tilde{p}_\kappa(\mathbf{q}) &= \int d\alpha\; p(\alpha) e^{i\kappa q_x \Delta R(\alpha)} \\
\Omega_\kappa(\mathbf{q}) &= \tilde{p}_{2\kappa}(\mathbf{q}) \phi(\mathbf{q}) e^{i q_x D_0} \\
\widetilde{\mathscr{F}_\kappa}(\mathbf{q}) &= \int d\alpha\; p(\alpha) F_\alpha(\mathbf{q}) e^{i\kappa q_x \Delta R(\alpha)},
\end{aligned}
$$

and the Fourier transform of $P_1(x|\alpha_0\alpha_1)$ is

$$\mathscr{P}(\mathbf{q}) = \phi(\mathbf{q}) e^{iq_x D_0} e^{i\kappa q_x[\Delta R(\alpha_0)+\Delta R(\alpha_1)]}.$$

Using the result from the one-dimensional analysis, one can apply this formula ad hoc for distributions of particles in a plane, where the coordinate x will now be replaced with (x, y), while the s coordinate encodes a position in the remaining orthogonal direction. One must be aware however that this constitutes a further approximation, since this type of correlation does not have a general solution in more than one dimension.

The intensity in equation (5.1) will contain a Dirac delta function contribution, caused by taking an infinite sum of terms that are perfectly correlated at $\mathbf{q} = 0$. One can leverage this behaviour by multiplying the nearest neighbor distribution by a constant factor $e^{-D/\Lambda}$, which removes the division by zero in equation (5.1). Another way of dealing with this infinity at $\mathbf{q} = 0$ consists of taking only a finite number of terms, in which case the geometric series still has an analytic solution, but becomes a bit more cumbersome:

$$I(\mathbf{q}) = \left\langle \left|F_\alpha(\mathbf{q})\right|^2 \right\rangle_\alpha + 2\,\mathrm{Re}\left\{ \frac{1}{\tilde{p}_{2\kappa}(\mathbf{q})} \widetilde{\mathscr{F}_\kappa}(\mathbf{q}) \widetilde{\mathscr{F}_\kappa^*}(\mathbf{q}) \right.$$
$$\left. \times \left[\left(1 - \frac{1}{N}\right) \frac{\Omega_\kappa(\mathbf{q})}{1-\Omega_\kappa(\mathbf{q})} - \frac{1}{N} \frac{\Omega_\kappa^2(\mathbf{q})\left(1-\Omega_\kappa^{N-1}(\mathbf{q})\right)}{\left(1-\Omega_\kappa(\mathbf{q})\right)^2} \right] \right\}.$$

This expression has a well-defined limit for $\Omega_\kappa(\mathbf{q}) \to 1$ (when $\mathbf{q} \to 0$), namely:

$$\lim_{\mathbf{q}\to 0} I(\mathbf{q}) = \left\langle |F_\alpha(0)|^2 \right\rangle_\alpha + (N-1)\,|\langle F_\alpha(0)\rangle_\alpha|^2.$$

5.3 Vertical location of particles

⚠ Remark: The particles cannot sit in between layers. At most they can be sitting on any inner interfaces.

5.3.1 Particles deposited on a substrate

In this configuration, the particles are sitting on top of a substrate layer, in the air as shown in fig. 5.5. In the DWBA the expression of a form factor becomes

$$F_{\mathrm{DWBA}}(q_\parallel, k_{i,z}, k_{f,z}) = F_{\mathrm{BA}}(q_\parallel, k_{i,z} - k_{f,z}) + R_i F_{\mathrm{BA}}(q_\parallel, -k_{i,z} - k_{f,z})$$
$$+ R_f F_{\mathrm{BA}}(q_\parallel, k_{i,z} + k_{f,z}) + R_i R_f F_{\mathrm{BA}}(q_\parallel, -k_{i,z} + k_{f,z}), \tag{5.2}$$

where $q_\parallel$ is the component of the scattering beam in the plane of the interface ($\mathbf{q} = \mathbf{k}_i - \mathbf{k}_f$), $k_{i,z}$ and $k_{f,z}$ are the z-component of the incident and scattered beam, respectively. R_i, R_f are the reflection coefficients in incidence and reflection. They are defined as

$R = \dfrac{k_z + \sqrt{n_s^2 k_0^2 - |k_\parallel|^2}}{k_z - \sqrt{n_s^2 k_0^2 - |k_\parallel|^2}}$, where $n_s = 1 - \delta_s + i\beta_s$ is the refractive index of the substrate, k_0

is the wavelength in vacuum ($2\pi/\lambda$), k_z and $k_\parallel$ are the z-component and the in-plane component of $\mathbf{k}_i$ or $\mathbf{k}_f$.

Remark: If the particles are sitting on a multilayered system, the expression of the form factor in the DWBA is obtained by replacing the Fresnel coefficient by the corresponding coefficients of the underlying layers [**?**, **?**].

Figure 5.5 illustrates the four scattering processes for a supported particle, taken into account in the DWBA. The first term of eq. 5.2 corresponds to the Born approximation. Each term of F_{DWBA} is weighted by a Fresnel coefficient.

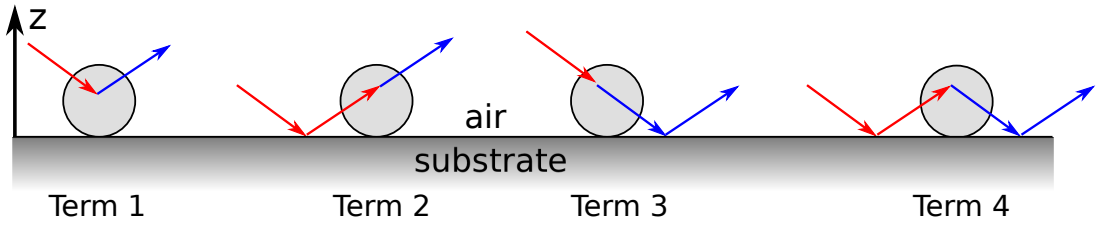


Figure 5.5: Schematic views of the different terms appearing in the expression of the form factor under DWBA for particles sitting on a substrate layer.

Script 5.1 illustrates the difference between BA and DWBA in `BornAgain` when generating the sample. We consider the simple case of:

- one kind of particles' shape,
- no interference between the particles,
- in the DWBA, a sample made of a layer of substrate on which are deposited the particles,
- in the BA, a sample composed of the particles in air.

Figure 5.6 shows the intensity contour plot generated using this script with truncated spheroids as particles.

Listing 5.1: `Python` script to generate a sample using Born (BA) or Distorted Wave Born Approximation (DWBA). The difference between BA and DWBA in this simple case is the absence or presence of a substrate layer in the sample.

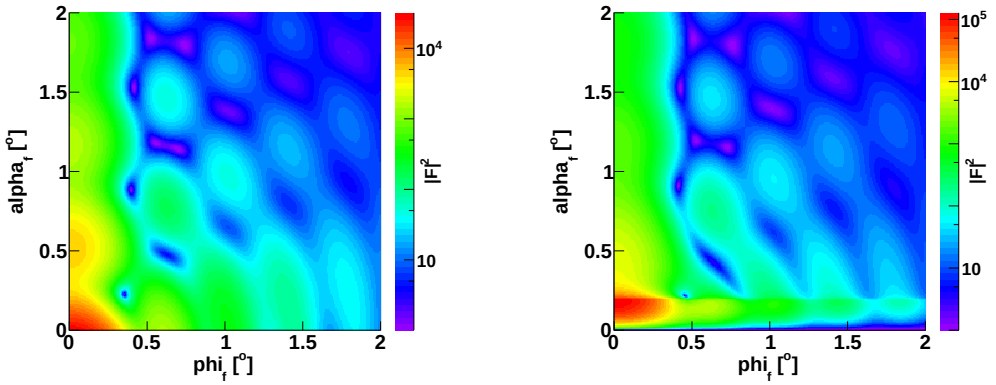
```
def get_sample():
    """
    Build and return the sample to calculate form factor of
    truncated spheroid in Born or Distorted Wave Born Approximation.
    """
    # defining materials
    m_ambience = HomogeneousMaterial("Air", 0.0, 0.0)
    m_substrate = HomogeneousMaterial("Substrate", 6e-6, 2e-8)
    m_particle = HomogeneousMaterial("Particle", 6e-4, 2e-8)

    # collection of particles
    ff= FormFactorTruncatedSpheroid(7.5*nanometer, 9.0*nanometer,
       1.2)
    particleshape = Particle(m_particle, ff)
    particle_layout = ParticleLayout()
    particle_layout.addParticle(particleshape, 0.0, 1.0)

    # interferences
    interference = InterferenceFunctionNone()
    particle_layout.addInterferenceFunction(interference)

    # assembling the sample
    air_layer = Layer(m_ambience)
    air_layer.addLayout(particle_layout)
    substrate_layer = Layer(m_substrate, 0)

    multi_layer = MultiLayer()
    multi_layer.addLayer(air_layer)
    # Comment the following line out for Born Approximation
    multi_layer.addLayer(substrate_layer)
    return multi_layer
```



(a) Born Approximation (b) DWB Approximation

Figure 5.6: Intensity map of TruncatedSpheroid form factor in BA and DWBA computing using script 5.1 for the sample.

Remark: In `BornAgain`, the DWBA is implemented automatically when assembling the sample with more layers than only the air layer (for example, for particles are sitting on a substrate).

5.3.2 Buried particles

The system considered in this section consists of particles encapsulated in a layer, which is sitting on a substrate (see fig. 5.7). In this case the form factor in the DWBA is given by

$$\begin{aligned} F_{\mathrm{DWBA}}(q_\parallel, k_{i,z}, k_{f,z}) = {} & T_i T_f F_{\mathrm{BA}}(q_\parallel, k_{i,z} - k_{f,z}) e^{i(k_{i,z}-k_{f,z})d} \\ & + R_i T_f F_{\mathrm{BA}}(q_\parallel, -k_{i,z} - k_{f,z}) e^{i(-k_{i,z}-k_{f,z})d} \\ & + R_f T_i F_{\mathrm{BA}}(q_\parallel, k_{i,z} + k_{f,z}) e^{i(k_{i,z}+k_{f,z})d} \\ & + R_f R_i F_{\mathrm{BA}}(q_\parallel, -k_{i,z} + k_{f,z}) e^{i(-k_{i,z}+k_{f,z})d}, \end{aligned} \tag{5.3}$$

$$R_j = \frac{t_{0,1}^j r_{1,2}^j \exp(2ik_{j,z}t)}{1 + r_{0,1}^j r_{1,2}^j \exp(2ik_{j,z}t)}, \quad T_j = \frac{t_{0,1}^j}{1 + r_{0,1}^j r_{1,2}^j \exp(2ik_{j,z}t)}, j = i, f$$

where $q_\parallel$ is the component of the scattering beam in the plane of the interface, $k_{i,z}$ and $k_{f,z}$ are the z-component of the incident and scattered beams, respectively. d is the depth at which the particles are sitting in the layer. Note that this value is given relative to the top of this layer and it is not the coordinate in the absolute referential (linked with the full sample) and it is measured up to the bottom of the particle. t

is the thickness of the intermediate layer containing the particles. $R_{i,f}$ and $T_{i,f}$ are the reflection and transmission coefficients in incidence and reflection (they can be calculated using Parratt or matrix formalism). $r^j_{0,1}$, $r^j_{1,2}$ $t^j_{0,1}$ are the reflection and transmission coefficients between layers; the indices are related to different boundaries with 0: air, 1: intermediate layer and 2: substrate layer and the superscript j is associated with the incident or scattered beams:

$$r^j_{n,n+1} = \frac{k_{j,z,n} - k_{j,z,n+1}}{k_{j,z,n} - k_{j,z,n+1}}, \qquad t^j_{n,n+1} = \frac{2k_{j,z,n}}{k_{j,z,n} - k_{j,z,n+1}}, \quad n = 0,1, \quad j = i,f,$$

where index n is related to the layers, z to the vertical component, and j to the beams (incident and outgoing).

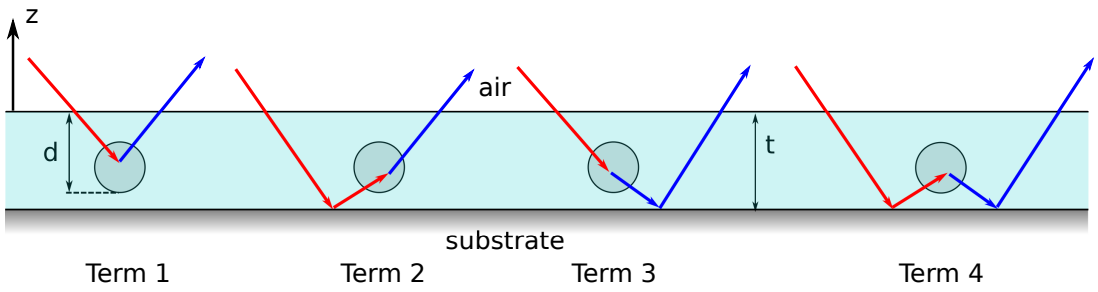


Figure 5.7: Schematic views of the different terms appearing in the expression of the form factor under the DWBA for buried particles.

Figure 5.8 shows a typical example of the output intensity scattered from a sample made of 3 layers: air, substrate, and in between, spherical particles embedded in the middle of a 30 nm-thick layer. This figure had been generated using listing 5.2.

Listing 5.2: `Python` script to generate a sample where spherical particles are embedded in the middle of a layer on a substrate.

```
def get_sample():
    """
    Build and return the sample with buried spheres in DWBA.
    """
    # defining materials
    m_ambience = HomogeneousMaterial("Air", 0.0, 0.0)
    m_interm_layer = HomogeneousMaterial("IntermLayer",3.45e-6, 5.24
        e-9)
    m_substrate = HomogeneousMaterial("Substrate", 7.43e-6, 1.72e-7)
    m_particle = HomogeneousMaterial("Particle", 0.0, 0.0)

    # collection of particles
    ff = FormFactorFullSphere(10.2*nanometer)
    particleshape = Particle(m_particle, ff)
    particle_layout = ParticleLayout()
```

```
    particle_layout.addParticle(particleshape,25.2,1.0)

    # interferences
    interference = InterferenceFunctionNone()
    particle_layout.addInterferenceFunction(interference)

    # assembling the sample
    air_layer = Layer(m_ambience)
    intermediate_layer = Layer(m_interm_layer, 30.*nanometer)
    intermediate_layer.addLayout(particle_layout)
    substrate_layer = Layer(m_substrate, 0)

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

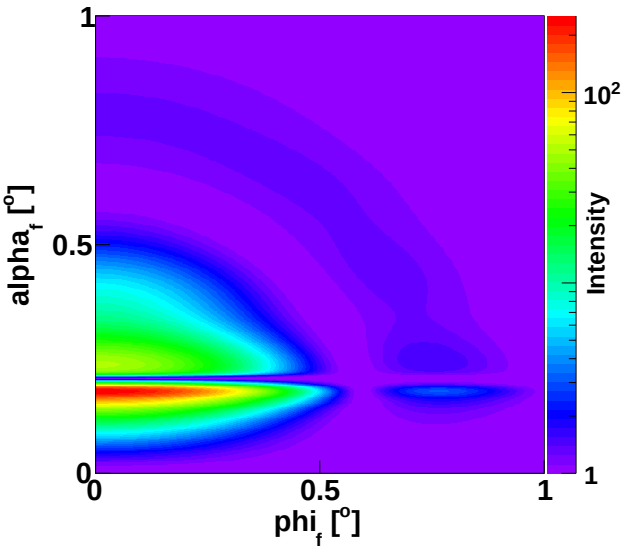
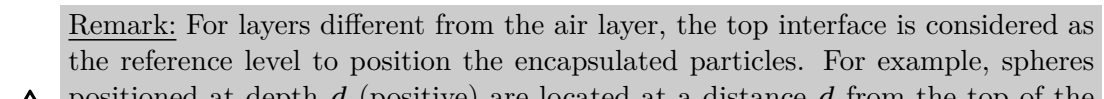


Figure 5.8: Map of intensity scattered from a sample made of spherical particles embedded in the middle of a 30 nm-thick layer on a substrate (see Script 5.2 for details about the sample).



Remark: For layers different from the air layer, the top interface is considered as the reference level to position the encapsulated particles. For example, spheres positioned at depth d (positive) are located at a distance d from the top of the layer up to the bottom of these particles. This convention is different for the top air layer, where particles sitting at the interface with an underlying layer (*i.e.* the bottom of the air layer) are located at depth 0 (see fig. 5.9).

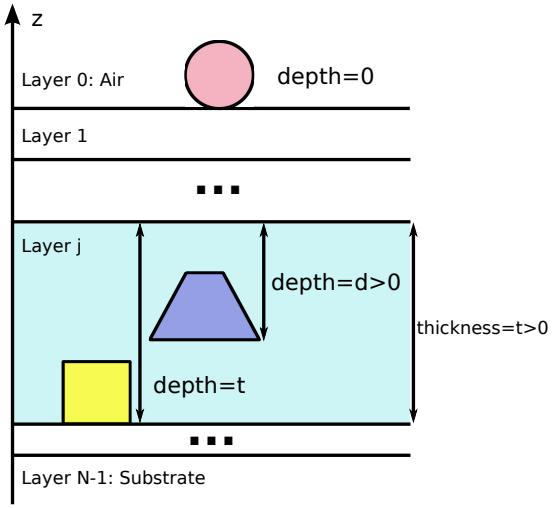


Figure 5.9: Illustration of the convention about `depth` used in `BornAgain` to encapsulate particles in layers.

5.3.3 Implementation in `BornAgain`

This section describes the implementation of the interference functions in `BornAgain`. For an implementation of all the components of a simulation, the use is referred, for example, to Section 8.1.2.

Remark: In `BornAgain` the particles are positioned in the same vertical layer.

5.3.3.1 Size-distribution models

The decoupling approximation (DA), local monodisperse approximation (LMA) and size spacing correlation approximation (SSCA) can be used in `BornAgain`. The selection between DA and SSCA is made using `ILayout.setApproximation(EInterferenceFunction approximation)` when defining the characteristics of the way particles and interference functions are embedded in a layer. For example,

```
    particle_layout = ParticleLayout()
   ....
# interference approx chosen between: DA (default) and SSCA
```

```
    particle_layout.setApproximation(ILayout.DA)
```

Note that with the SSCA, the users have to specify the coupling parameter κ (with the function `setKappa`), which should be a positive dimensionless value. κ characterizes the influence of the neighboring particles' sizes on their distance. If $\kappa = 0$, the SSCA reduces to the DA with a radial paracrystal for the interference function.

For the LMA, its implementation is automatically done when using more than one layout of particles:

```
    particle_layout0 = ParticleLayout()
    particle_layout1 = ParticleLayout()
   ....
# association of each particles' layout with materials, form factors
#... and with a material layer
    layer_a = Layer(m_material_a)
    layer_a.addLayout(particle_layout0)
    layer_a.addLayout(particle_layout1)
```

5.3.3.2 Probability distribution functions

The probability distribution functions have been implemented in the reciprocal space in `BornAgain`. Their expressions are given in Table 5.1.

Function	One dimension	Two dimensions
Cauchy	$(1+q^2\omega^2)^{-3/2}$	$(1+q_x^2 cl_x^2 + q_y^2 cl_y^2)^{-3/2}$
Gauss	$\frac{1}{2}\exp(-\frac{q^2\omega^2}{4})$	$\frac{1}{2}\exp\left(-\frac{q_x^2 cl_x^2 + q_y^2 cl_y^2}{4}\right)$
Voigt	$\frac{\eta}{2}\exp\left(-\frac{q^2\omega^2}{4}\right) + \frac{1-\eta}{(1+q^2\omega^2)^{3/2}}$	$\frac{\eta}{2}\exp\left(-\frac{q_x^2 cl_x^2 + q_y^2 cl_y^2}{4}\right) + \frac{1-\eta}{(1+q_x^2 cl_x^2 + q_y^2 cl_y^2)^{3/2}}$

Table 5.1: List of probability distribution functions in reciprocal space. ω, cl stand for coherence lengths (the index refers to the axis) and η is a weighting coefficient.

The Cauchy distribution corresponds to $\exp(-r)$ in real space and the Voigt one is a linear combination of the Gaussian and Cauchy probability distribution functions.

One dimension

- `FTDistribution1DCauchy(`ω`)`,
- `FTDistribution1DGauss(`ω`)`,
- `FTDistribution1DVoigt(`ω`,`η`)`.

where ω is the coherence length and η is a weighting factor.

Two dimensions

- `FTDistribution2DCauchy(`cl_x`,` cl_y`)`,
- `FTDistribution2DGauss(`cl_x`,` cl_y`)`,
- `FTDistribution2DVoigt(`cl_x`,` cl_y`)`

where $cl_{x,y}$ are the coherence lengths in the x or y direction, respectively.

These functions can be used with all interference functions, except the case without any interference.

5.3.3.3 Interferences

The interference function is specified when building the sample. It is linked with the particles (shape, material). Examples of implementation are given at the end of each description.

Syntax: `particle_layout.addInterferenceFunction(interference_function)`, where `particle_layout` holds the information about the different shapes and their proportions for a given layer of particles, and `interference_function` is one of the following expressions:

- `InterferenceFunctionNone()`
- `InterferenceFunction1DLattice(lattice_parameters)`
- `InterferenceFunctionRadialParaCrystal(peak_distance, damping_length)`
- `InterferenceFunction2DLattice(lattice_parameters)`
- `InterferenceFunction2DParaCrystal(length_1, length_2,` α`_lattice,` ξ`, damping_length)`

Remark: `InterferenceFunction1DLattice` can only be used for particles which are infinitely long in one direction of the sample's surface like for example a rectangular grating.

5.3.3.4 ➽ `InterferenceFunctionNone()`

The particles are placed randomly in the dilute limit and are considered as individual, non-interacting scatterers. The scattered intensity is function of the form factors only.

Example The sample is made of a substrate on which are deposited half-spheres. Script 5.3 details the commands necessary to generate such a sample. Figure 5.10 shows an example of output intensity: Script 5.3 + detector's + input beam's characterizations.

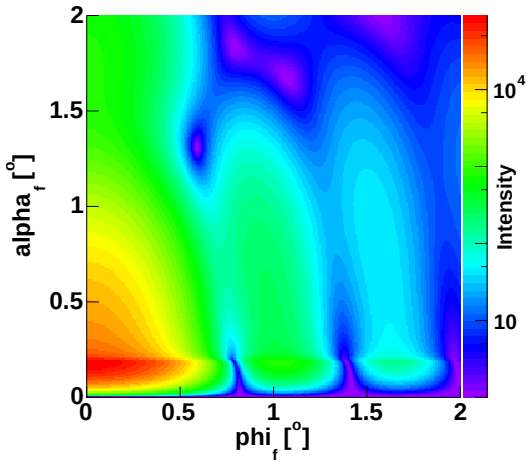


Figure 5.10: Output intensity scattered from a sample made of half-spheres with no interference between them.

Listing 5.3: `Python` script to simulate a sample made of half-spheres deposited on a substrate layer without any interference. The part specific to the interferences is marked in a red italic font.

```
def get_sample():
    """
    Build and return the sample representing particles with no
        interference
    """
    # defining materials
    m_ambience = HomogeneousMaterial("Air", 0.0, 0.0)
    m_substrate = HomogeneousMaterial("Substrate", 6e-6, 2e-8)
    m_particle = HomogeneousMaterial("Particle", 6e-4, 2e-8)
    # collection of particles
    sphere_ff = FormFactorTruncatedSphere(5*nanometer, 5*nanometer)
    sphere = Particle(m_particle, sphere_ff)
    particle_layout = ParticleLayout()
    particle_layout.addParticle(sphere, 0.0, 1.0)
    interference = InterferenceFunctionNone()
    particle_layout.addInterferenceFunction(interference)
    # assembling the sample
    air_layer = Layer(m_ambience)
    air_layer.addLayout(particle_layout)
    substrate_layer = Layer(m_substrate, 0)

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

5.3.3.5 ➽ `InterferenceFunction1DLattice(lattice_length, xi)`

where lattice_length is the lattice constant and ξ the angle in radian between the lattice unit vector and the **x**-axis of the reference Cartesian frame as shown in fig. 5.11.

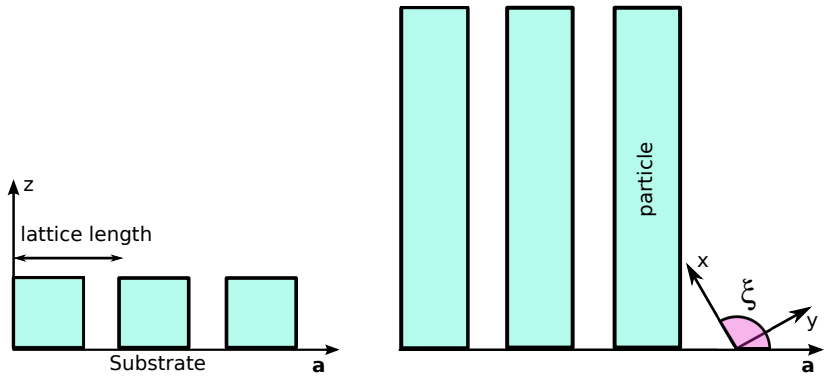


Figure 5.11: Schematic representation of a 1D lattice (side and top views). Such a lattice is characterized by a lattice length and the angle ξ.

⚠ Remark: By default the long axis of the particles in this 1D lattice is along the beam axis. That is the reason why in the example below the particles are rotated by 90° in the (x, y) plane: the main axis of the lattice is therefore parallel to the y-axis, perpendicular to the long axis of the particles.

A probability distribution function `pdf` has to be chosen from the list in section 5.3.3.2 in order to apply some modifications to the scattering peaks. This function is implemented using `setProbabilityDistribution(pdf)`.

Example: Script 5.4 details how to build in `BornAgain` a sample using `InterferenceFunction1DLattice` as the interference function. As mentioned previously, this interference function can only be used with infinitely wide or long particles. Here the sample is made of infinitely long boxes deposited on a substrate (these particles are characterized by their widths and heights). They are also rotated by 90° in the sample surface in order to have their long axis perpendicular to the input beam, which is along the x-axis.
The lattice parameters (the lattice length and angle between the lattice main axis and the x-axis) are passed into the constructor of the interference function.

Listing 5.4: `Python` script to generate a sample made of infinitely long boxes deposited on a substrate layer with the 1DLatticeInterference function. The part specific to the interferences is marked in a red italic font.

```
def get_sample():
    """
    Build and return the sample with 1DLatticeInterference function
    """
    # defining materials
    m_air = HomogeneousMaterial("Air", 0.0, 0.0)
    m_substrate = HomogeneousMaterial("Substrate", 6e-6, 2e-8)
    m_particle = HomogeneousMaterial("Particle", 6e-4, 2e-8)

    # collection of particles
    ff = FormFactorInfLongBox(10.*nanometer, 15.0*nanometer)
    box = Particle(m_particle, ff)
    particle_layout = ParticleLayout()
    transform = Transform3D.createRotateZ(90.0*degree)
    particle_layout.addParticle(box, transform)

    # interference function
    interference = InterferenceFunction1DLattice(30.0*nanometer,
        0.0*degree)
    pdf = FTDistribution1DCauchy(200./2./M_PI*nanometer)
    interference.setProbabilityDistribution(pdf)
    particle_layout.addInterferenceFunction(interference)

    # air layer with particles and substrate form multi layer
    air_layer = Layer(m_air)
    air_layer.addLayout(particle_layout)
    substrate_layer = Layer(m_substrate, 0)

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

5.3.3.6 ➽ `InterferenceFunctionRadialParaCrystal(peak_distance, damping_length)`

where `peak_distance` is the average distance to the first neighbor peak,

`width` is the width parameter of the probability distribution,

`damping_length` is used to introduce finite size effects by applying a multiplicative coefficient equal to `exp(-peak_distance/damping_length)` to the Fourier transform of the probability densities. `damping_length` is equal to 0 by default and, in this case, no correction is applied.

A probability distribution function `pdf` has to be chosen from the list in section 5.3.3.2 in order to apply some modifications to the scattering peaks. This function is implemented using `setProbabilityDistribution(pdf)`.

Remark This interference function is not one-dimensional. It takes into account the radial component of the scattering vector.

Example To illustrate the radial paracrystal interference function, we use the same sample as in the case without interference: half-spheres deposited on a substrate.

Listing 5.5: `Python` script to define the radial paracrystal interference function between half-spheres, where `trsphere` is of type `Particle`.

```
particle_layout = ParticleLayout()
particle_layout.addParticle(trsphere, 0.0, 1.0)
interference = InterferenceFunctionRadialParaCrystal(25.0*
   nanometer, 1e3*nanometer)
pdf = FTDistribution1DGauss(7 * nanometer)
interference.setProbabilityDistribution(pdf)
particle_layout.addInterferenceFunction(interference)
```

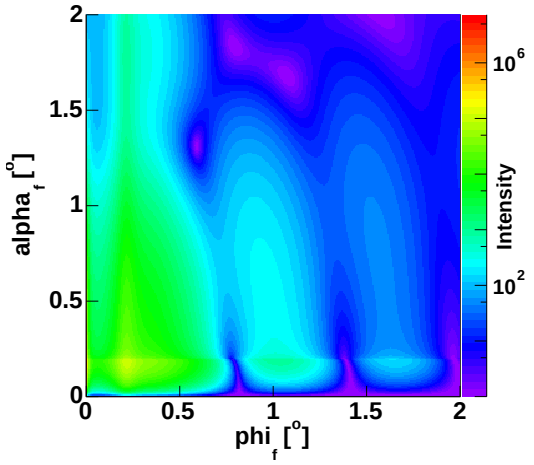


Figure 5.12: Output intensity scattered from a sample made of half-spheres with the radial paracrystal interference between them. This figure has been generated using Script 5.5 for the interference function.

5.3.3.7 ➽ `InterferenceFunction2DLattice(L_1, L_2, alpha, xi)`

where (L_1, L_2, α, ξ) are shown in figure 5.13 with

L_1, L_2 the lengths of the lattice cell,

α the angle between the lattice basis vectors **a**,**b** in direct space,

ξ is the angle defining the lattice orientation (set to 0 by default); it is taken as the angle between the **a** vector of the lattice basis and the **x** axis of the reference Cartesian frame (as shown in figure 3.2).

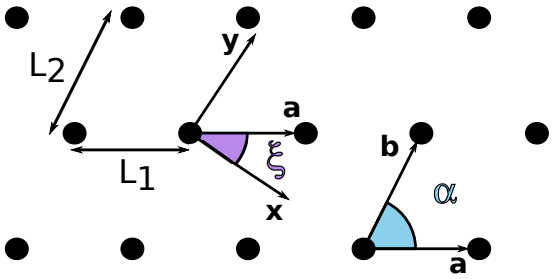


Figure 5.13: Schematic representation of a 2D lattice (top view). Such a lattice is characterized by lattice lengths L_1, L_2 and angles α and ξ.

Like for the one-dimensional case, a probability distribution function `pdf` has to be defined. One can choose between those listed in Section 5.3.3.2 and implements it using `setProbabilityDistribution(pdf)`.

Example The sample used to run the simulation is made of half-spheres deposited on a substrate. The interference function is "2Dlattice" and the particles are located at the nodes of a square lattice with $L_1 = L_2 = 20$ nm, $\mathbf{a} \equiv \mathbf{b}$ and the probability distribution function is Gaussian. We also use the Decoupling Approximation.

Listing 5.6: `Python` script to define a 2DLattice interference function between hemispherical particles as well as the Decoupling Approximation in `getSimulation()`. The part specific to the interferences is marked in a red italic font.

```
#collection of particles
sphere_ff = FormFactorTruncatedSphere(5*nanometer, 5*nanometer)
sphere = Particle(m_particle, sphere_ff)
interference = InterferenceFunction2DLattice(20.0*nanometer,
    20.0*nanometer, 90.0*degree, 0.0*degree)
pdf = FTDistribution2DGauss(200.0*nanometer/2.0/M_PI, 75.0*
    nanometer/2.0/M_PI)
interference.setProbabilityDistribution(pdf)
particle_layout = ParticleLayout()
particle_layout.addParticle(sphere, 0.0, 1.0)
particle_layout.addInterferenceFunction(interference)

# interference approx chosen between: DA (default) and SSCA
particle_layout.setApproximation(ILayout.DA)
```

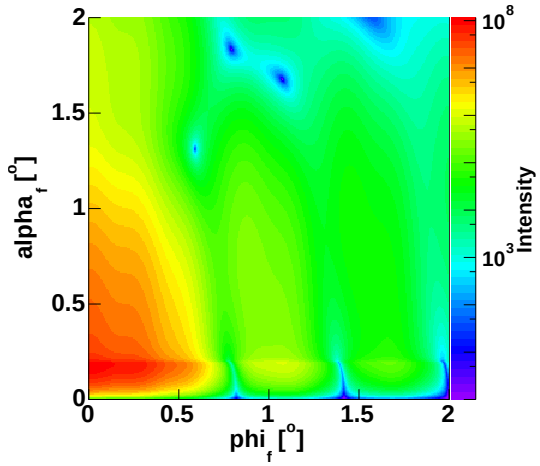


Figure 5.14: Output intensity scattered from a sample made of half-spheres with 2DLattice interference function in the Decoupling Approximation.

5.3.3.8 ➽ `InterferenceFunction2DParaCrystal(L_1, L_2, lattice_angle, ξ, damping_length)`

where L_1, L_2 are the lengths of the lattice cell,

lattice_angle the angle between the lattice basis vectors **a**,**b** in direct space,

ξ is the angle defining the lattice orientation (set to 0 by default).

`damping_length` is used to introduce finite size effects by applying a multiplicative coefficient equal to exp(-`peak_distance/damping_length`) to the Fourier transform of the probability densities. `damping_length` is equal to 0 by default and, in this case, no correction is applied.

Two predefined interference functions can also be used:

- `createSquare(peak_distance, damping_length, domain_size_1, domain_size_2)` where the angle between the base vectors of the lattice is set to $\pi/2$, it creates a squared lattice,
- `createHexagonal(peak_distance, damping_length, domain_size_1, domain_size_2)` where the angle between the base vectors of the lattice is set to $2\pi/3$,

where `domain_size1, 2` are the dimensions of coherent domains of the paracrystal along the main axes,
`peak_distance` is the same in both directions and $\mathbf{a} \equiv \mathbf{x}$.

Probability distribution functions have to be defined. As the two-dimensional paracrystal is defined from two independent one-dimensional paracrystals, we need two of these functions, using
`setProbabilityDistributions(pdf_1, pdf_2)`, with `pdf_1,2` related to each main axis of the paracrystal (see figure 5.15).

Example The particles deposited on a substrate are half-spheres. The scattered beams interference via the 2DParacrystal distribution function. The paracrystal is based on a 2D hexagonal lattice with a Gaussian probability distribution function in reciprocal space. Script 5.7 shows the implementation of the interference function and fig. 5.16 an example of output intensity using hemi-spherical particles.

Listing 5.7: `Python` script to define a "2DParacrystal" interference function between particles forming an hexagonal monolayer.

```
interference = InterferenceFunction2DParaCrystal.createHexagonal
    (30.0*nanometer,0.0, 40.0*micrometer, 40.0*micrometer)
pdf = FTDistribution2DCauchy(1.0*nanometer, 1.0*nanometer)
interference.setProbabilityDistributions(pdf, pdf)
particle_layout.addInterferenceFunction(interference)
```

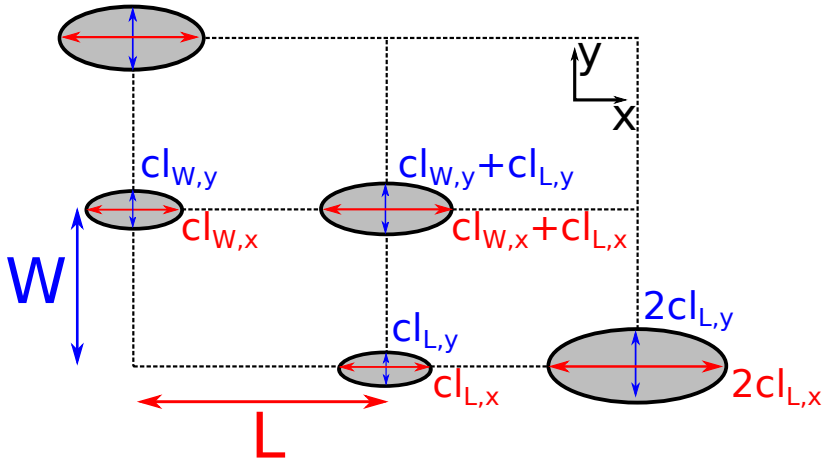


Figure 5.15: Schematics of the ideal 2D paracrystal. The gray-shaded areas mark the regions where the probability to find a node is larger that the width at half-maximum of the distribution. L and W are the mean inter-node distances along the two crystallographic axes. $\mathrm{cl}_{(L,W),(x,y)}$ are the widths of the distribution of distance. The disorder is propagated as we add more nodes. Such a structure would be generated using `InterferenceFunction2DParacrystal(L,W,90.*degrees,0,damp_length)`, with $\mathtt{pdf}_1$ = `FTDistribution2DGauss(`$\mathtt{cl}_{L,x}$`,`$\mathtt{cl}_{L,y}$`)` and $\mathtt{pdf}_2$ = `FTDistribution2DGauss(`$\mathtt{cl}_{W,x}$`,`$\mathtt{cl}_{W,y}$`)`.

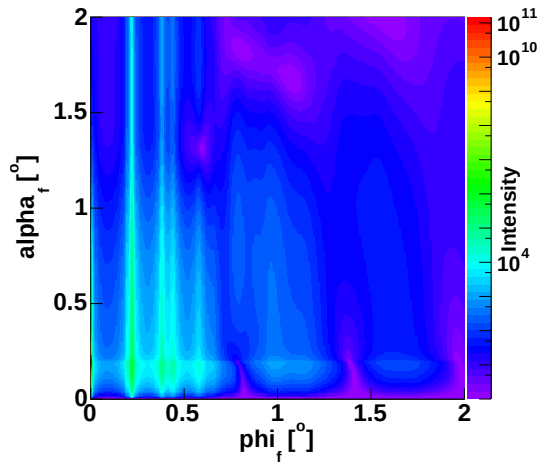


Figure 5.16: Output intensity scattered from a sample made of half-spheres with 2DParacrystal interference function.

5.3.4 Summary

Name	Characteristics	Comments
DA	no correlations	implemented with `setApproximation` default option
LMA	sample = groups of particles of identical sizes and shapes	automatic implementation if several `ParticleLayout`s are defined
SSCA	distance between particles = function of their sizes	- implemented with `setApproximation` - dimensionless coupling coefficient κ is required using `setKappa`.

Table 5.2: List of size-distribution models implemented in `BornAgain`.

Function	Parameters	Comments
`InterferenceFunctionNone`	None	disordered distribution
`InterferenceFunction1DLattice`	`lattice_length`	use only with infinitely long/wide particles
	$\xi = \widehat{(\mathbf{x}, \mathbf{a})}$	pdf=(Cauchy, Gauss or Voigt) to be defined
`InterferenceFunctionRadialParaCrystal`	peak_distance of pdf	pdf=(Cauchy, Gauss or Voigt) to be defined
	damping_length (optional)	
`InterferenceFunction2DLattice`	L_1, L_2: lattice lengths	pdf=(Cauchy, Gauss or Voigt) to be defined
	lattice_angle=$\widehat{(\mathbf{a}, \mathbf{b})}$	
	$\xi = \widehat{(\mathbf{x}, \mathbf{a})}$	
`InterferenceFunction2DParaCrystal`	L_1, L_2: lattice lengths	2D pdf=(Cauchy, Gauss or Voigt) to be defined
	lattice_angle=$\widehat{(\mathbf{a}, \mathbf{b})}$	(1 pdf per axis)
	$\xi = \widehat{(\mathbf{x}, \mathbf{a})}$	
	damping_length (optional)	same for both axes

Table 5.3: List of interference functions implemented in `BornAgain`. pdf : probability distribution function, $\mathbf{a}, \mathbf{b}$ are the lattice base vectors, and $\mathbf{x}$ is the axis vector perpendicular to the detector plane.

Chapter 6

Scattering by domain structures

... to come

Chapter 7

Scattering by rough interfaces

... to come

Chapter 8

Using the Python API

8.1 Running a simulation

A simulation of GISAS using `BornAgain` consists of the following steps:

- define materials by specifying name and refractive index,
- define layers by specifying thickness, roughness, material,
- define embedded particles by specifying shape, size, constituting material, interference function,
- embed the particles in layers, specifying density, position, orientation,
- assemble a multilayered sample,
- specify input beam and detector characteristics,
- run the simulation,
- save the simulated detector image.

All these steps can be organized in either a Graphical User Interface (GUI) or by providing a Python script with the simulation description. In the following, we describe how to write a `Python` script which runs a `BornAgain` simulation. For tutorials about this programming language, the users are referred to [**?**].

8.1.1 Units:

By default the angles are expressed in radians and the lengths are given in nanometers. But it is possible to use other units by specifying them right after the value of the corresponding parameter like, for example, `20.0*micrometer`.

8.1.2 A first example

In this example, we simulate the scattering from a mixture of cylindrical and prismatic nanoparticles without any interference between them. These particles are placed in air, on top of a substrate.
We are going to go through each step of the simulation. The `Python` code snippet specific to each stage will be given at the beginning of the description. More examples can be found at our project web site `http://www.bornagainproject.org/documentation/python_examples`

8.1.2.1 Importing Python modules

```
1 import numpy
2 import matplotlib
3 import pylab
4 from bornagain import *
```

We start by importing different functions from external modules, for example `NumPy` (lines 1-3), which is a fundamental package for scientific computing with `Python` [?]. In particular, line 4 imports the features of `BornAgain` software.

8.1.2.2 Defining the materials

```
5  def get_sample():
6      """
7      Build and return the sample representing cylinders and pyramids
           on top of substrate without interference.
8      """
9      # defining materials
10     m_air = HomogeneousMaterial("Air", 0.0, 0.0)
11     m_substrate = HomogeneousMaterial("Substrate", 6e-6, 2e-8)
12     m_particle = HomogeneousMaterial("Particle", 6e-4, 2e-8)
```

Line 5 marks the beginning of the function to define our sample. Lines 10, 11 and 12 define different materials using class `HomogeneousMaterial`. The general syntax is the following

```
<material_name> = HomogeneousMaterial("name", delta, beta)
```

where `name` is the name of the material associated with its complex refractive index n=1-`delta` +i `beta`. `<material_name>` is later used when referring to this particular material. The three materials defined in this example are `Air` with a refractive index of 1 (`delta = beta = 0`), a `Substrate` associated with a complex refractive index equal to $1-6\times10^{-6}+i2\times10^{-8}$, and the material of the particles, whose refractive index is n$= 1-6\times10^{-4}+i2\times10^{-8}$.

8.1.2.3 Defining the particles

```
14    # collection of particles
15    cylinder_ff = FormFactorCylinder(5*nanometer, 5*nanometer)
16    cylinder = Particle(m_particle, cylinder_ff)
17    prism_ff = FormFactorPrism3(10*nanometer, 5*nanometer)
18    prism = Particle(m_particle, prism_ff)
```

We implement two different shapes of particles: cylinders and prisms (*i.e.* elongated particles with a constant equilateral triangular cross section).

All particles implemented in `BornAgain` are defined by their form factors (see Appendix **??**), their sizes and the material they are made of. Here, for the cylindrical particle, we input its radius and height. For the prism, the possible inputs are the length of one side of its equilateral triangular base and its height.

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 15). Then we associate this shape with the constituting material as in line 16. The same procedure has been applied for the prism in lines 17 and 18, respectively.

8.1.2.4 Characterizing particles assembly

```
19    particle_layout = ParticleLayout()
20    particle_layout.addParticle(cylinder, 0.0, 0.5)
21    particle_layout.addParticle(prism, 0.0, 0.5)
22    interference = InterferenceFunctionNone()
23    particle_layout.addInterferenceFunction(interference)
```

The object which holds the information about the positions and densities of particles in our sample is called `ParticleLayout` (line 19). We use the associated function `addParticle` for each particle shape (lines 20, 21). Its general syntax is

```
addParticle(<particle_name>, depth, abundance)
```

where `<particle_name>` is the name used to define the particles (lines 16 and 18), `depth` (default value = 0) is the vertical position, 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. Here we have 50% of cylinders and 50% of prisms.

Remark: Depth of particles
The vertical positions of the particles in a layer are given in relative coordinates. For the top layer, the bottom of the layer corresponds to `depth`=0 and negative values would correspond to particles floating above layer 1 since the vertical axis, shown in figure 3.2 is pointing upwards. But for all the other layers, it is the top of the layer which corresponds to `depth`=0.

Finally, lines 22 and 23 specify that there is **no coherent interference** between the waves scattered by these particles. In this case, the intensity is calculated by the

incoherent sum of the scattered waves: $\langle |F_j|^2 \rangle$, where F_j is the form factor associated with the particle of type j. 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 (see Section 5.1.2). On the contrary, the vertical position is imposed when we add the particles in a given layer by parameter `depth`, as shown in lines 20 and 21.

8.1.2.5 Multilayer

```
24 # air layer with particles and substrate form multi layer
25     air_layer = Layer(m_air)
26     air_layer.addLayout(particle_layout)
27     substrate_layer = Layer(m_substrate, 0)
28     multi_layer = MultiLayer()
29     multi_layer.addLayer(air_layer)
30     multi_layer.addLayer(substrate_layer)
31     return multi_layer
```

We now have to configure our sample. For this first example, the particles, *i.e.* 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 down to the bottom one**.

Let us start with the air layer. It contains the particles. In line 25, we use the previously defined `m_air` (="air" material) (line 10). The command in line 26 shows that this layer contains particles which are defined using particle layout object. The substrate layer only contains the substrate material (line 27).

There are different possible syntaxes to define a layer. As shown in lines 25 and 27, we can use `Layer(<material_name>,thickness)` or `Layer(<material_name>)`. The second case corresponds to the default value of the `thickness`, equal to 0. The `thickness` is expressed in nanometers.

Our two layers are now fully characterized. The sample is assembled using `MultiLayer()` constructor (line 28): 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 30).

8.1.2.6 Characterizing the input beam and output detector

```
32 def get_simulation():
33     """
34     Create and return GISAXS simulation with beam and detector
           defined
35     """
36     simulation = Simulation()
37     simulation.setDetectorParameters(100, -1.0*degree, 1.0*degree,
           100, 0.0*degree, 2.0*degree)
```

```
38      simulation.setBeamParameters(1.0*angstrom, 0.2*degree, 0.0*
            degree)
39      return simulation
```

The first stage is to create the `Simulation()` object (line 36). Then we define the detector (line 37) and beam parameters (line 38). Those functions are part of the Simulation class. The different incident and exit angles are shown in figure 3.2.

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),
```

where number of bins `n_phi`, low edge of first bin `phi_f_min` and upper edge of last bin `phi_f_max` all together define ϕ_f detector axis, while `n_alpha`, `alpha_f_min` and `alpha_f_max` are related to α_f detector axis.

Remark: Axis binning
By default axes are binned to provide constant bin size in k-space, which means slightly non-equidistant binning in angle space. Other possible options, including user defined axes with custom variable bin size are explained elsewhere.

To characterize the beam we use function

```
setBeamParameters(lambda, alpha_i, phi_i),
```

where `lambda` is the incident beam wavelength, `alpha_i` is the incident grazing angle on the surface of the sample, `phi_i` is the in-plane direction of the incident beam (measured with respect to the x-axis).

8.1.2.7 Running the simulation and plotting the results

```
40  def run_simulation():
41      """
42     Run simulation and plot results
43      """
44      sample = get_sample()
45      simulation = get_simulation()
46      simulation.setSample(sample)
47      simulation.runSimulation()
48      result = simulation.getIntensityData().getArray() + 1  # for log
            scale
49      pylab.imshow(numpy.rot90(result, 1), norm=matplotlib.colors.
            LogNorm(), extent=[-1.0, 1.0, 0, 2.0])
50      pylab.show()
```

The function, whose definition starts from line 40, gathers all items. We create the sample and the simulation objects at the lines 44 and 45, using calls to the previously defined functions. We assign the sample to the simulation at line 46 and finally launch the simulation at line 47.

In line 48 we obtain 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. Lines 49-50 produces the two-dimensional contour plot of the intensity as a function of α_f and ϕ_f shown in figure 8.1.

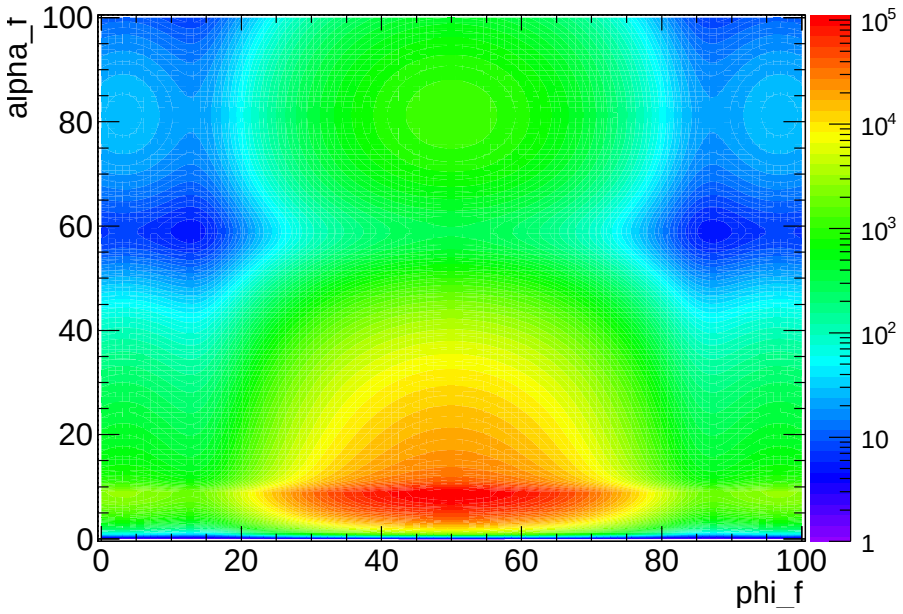


Figure 8.1: Simulated grazing-incidence small-angle X-ray scattering from a mixture of cylindrical and prismatic nanoparticles without any interference, deposited on top of a substrate. The input beam is characterized by a wavelength λ of 1 Å and incident angles $\alpha_i = 0.2°$, $\phi_i = 0°$. The cylinders have a radius and a height both equal to 5 nm, the prisms are characterized by a side length equal to 10 nm and they are 5 nm high. The material of the particles has a refractive index of $1 - 6 \times 10^{-4} + i2 \times 10^{-8}$. For the substrate it is equal to $1 - 6 \times 10^{-6} + i2 \times 10^{-8}$. The color scale is associated with the output intensity in arbitrary units.

8.1.3 Working with sample parameters

This section gives additional details about the manipulation of sample parameters during run time; that is after the sample has already been constructed. For a single simulation this is normally not necessary. However it might be useful during interactive work when the user tries to find optimal sample parameters by running a series of simulations. A similar task also arises when the theoretical model, composed of the description of the sample and of the simulation, is used for fitting real data. In this case, the fitting kernel requires a list of the existing sample parameters and a mechanism for changing the values of these parameters in order to find their optima.

In BornAgain this is done using the so-called sample parameter pool mechanism. We are going to briefly explain this approach using the example of Section 8.1.2.

In BornAgain a sample is described by a hierarchical tree of objects. For the multilayer created in the previous section this tree can be graphically represented as shown in Fig. 8.2. Similar trees can be printed in a Python session by running multi_layer.printSampleTree()

The top MultiLayer object is composed of three children, namely Layer #0, Layer Interface #0 and Layer #1. The children objects might themselves also be decomposed into tree-like structures. For example, Layer #0 contains a ParticleLayout object, which holds information related to the two types of particles populating the layer. All numerical values used during the sample construction (thickness of layers, size of particles, roughness parameters) are part of the same tree structure. They are marked in the figure with shaded gray boxes.

These values are registered in the sample parameter pool using the name composed of the corresponding nodes' names. And they can be accessed/changed during run time. For example, the height of the cylinders populating the first layer can be changed from the current value of 5 nm to 1 nm by running the command

```
multi_layer.setParameterValue('/MultiLayer/Layer0/ParticleLayout/
    ParticleInfo0/Particle/FormFactorCylinder/height', 1.0)
```

A list of the names and values of all registered sample's parameters can be displayed using the command

```
> multi_layer.printParameters()
The sample contains following parameters ('name':value)
'/MultiLayer/Layer0/ParticleLayout/ParticleInfo0/Particle/
    FormFactorCylinder/height':5
'/MultiLayer/Layer0/ParticleLayout/ParticleInfo0/Particle/
    FormFactorCylinder/radius':5
'/MultiLayer/Layer0/ParticleLayout/ParticleInfo0/abundance':0.5
'/MultiLayer/Layer0/ParticleLayout/ParticleInfo0/depth':0
'/MultiLayer/Layer0/ParticleLayout/ParticleInfo1/Particle/
    FormFactorPrism3/length':5
'/MultiLayer/Layer0/ParticleLayout/ParticleInfo1/Particle/
    FormFactorPrism3/height':5
'/MultiLayer/Layer0/ParticleLayout/ParticleInfo1/abundance':0.5
'/MultiLayer/Layer0/ParticleLayout/ParticleInfo1/depth':0
'/MultiLayer/Layer0/thickness':0
'/MultiLayer/Layer1/thickness':0
'/MultiLayer/LayerInterface/roughness/corrlength':0
'/MultiLayer/LayerInterface/roughness/hurst':0
'/MultiLayer/LayerInterface/roughness/sigma':0
'/MultiLayer/crossCorrLength':0
```

Wildcards '*' can be used to reduce typing or to work on a group of parameters. In the example below, the first command will change the height of all cylinders in the same way, as in the previous example. The second line will change simultaneously the

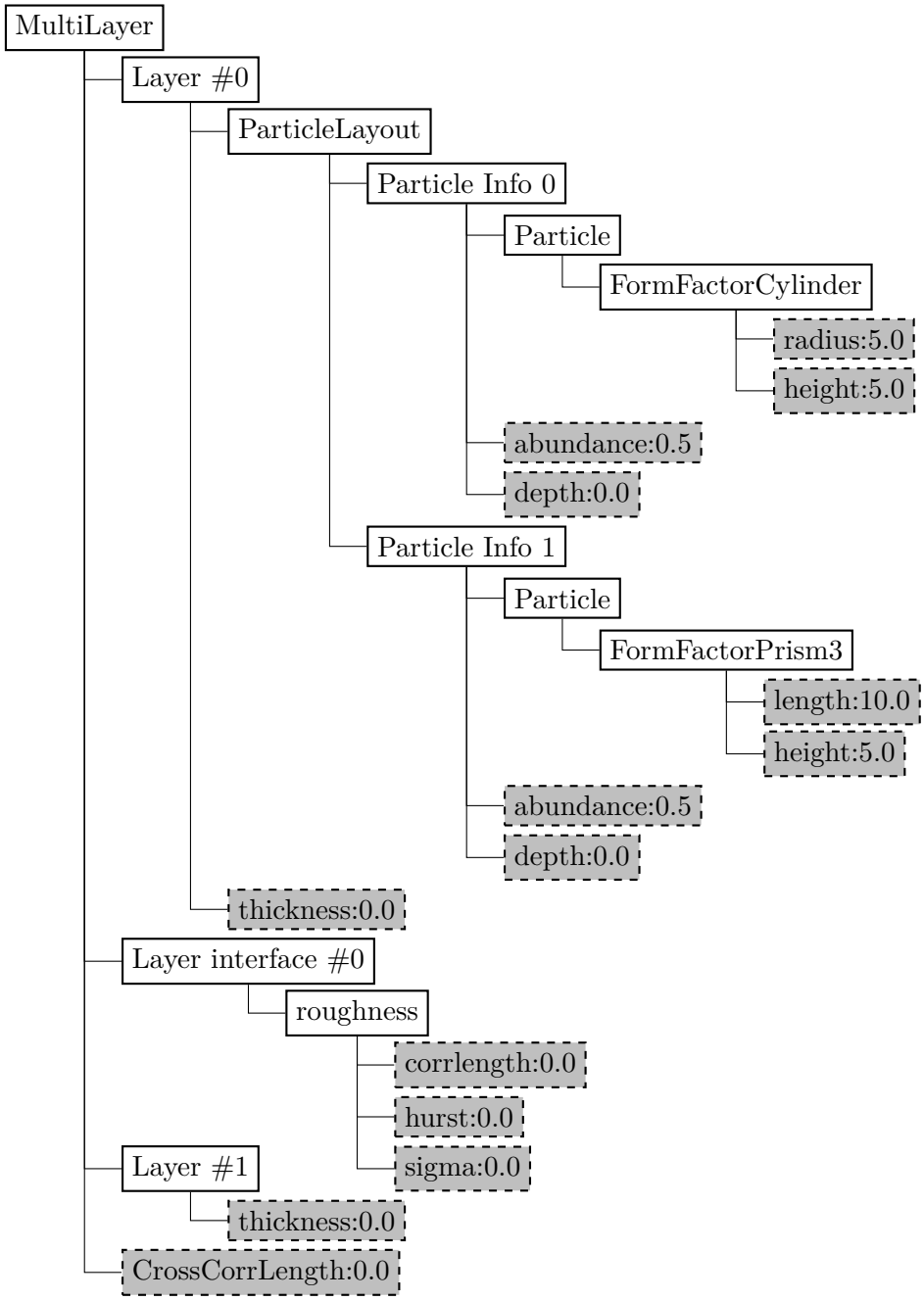


Figure 8.2: Tree representation of the sample structure.

height of *both* cylinders and prisms.

```
multi_layer.setParameterValue('*FormFactorCylinder/height', 1.0)
multi_layer.setParameterValue('*height', 1.0)
```

The complete example described in this section can be found at

```
./Examples/python/fitting/ex001_SampleParametersIntro/
   SampleParametersIntro.py
```

8.2 Fitting

In addition to the simulation of grazing incidence X-ray and neutron scattering by multilayered samples, `BornAgain` also offers the option to fit the numerical model to reference data by modifying a selection of sample parameters from the numerical model. This aspect of the software is discussed in the current chapter.

Section 8.2.0.1 details the implementation of fittings in `BornAgain` . `Python` fitting examples with detailed explanations of every fitting step are given in Section 8.2.3. Advanced fitting techniques, including fine tuning of minimization algorithms, simultaneous fits of different data sets, parameters correlation, are covered in Section 8.2.4. Section 8.2.5 contains some practical advice, which might help the user to get right answers from `BornAgain` fitting.

8.2.0.1 Implementation in BornAgain

Fitting in `BornAgain` deals with estimating the optimum parameters in the numerical model by minimizing the difference between numerical and reference data. The features include

- a variety of multidimensional minimization algorithms and strategies.
- the choice over possible fitting parameters, their properties and correlations.
- the full control on objective function calculations, including applications of different normalizations and assignments of different masks and weights to different areas of reference data.
- the possibility to fit simultaneously an arbitrary number of data sets.

Figure 8.3 shows the general work flow of a typical fitting procedure.

Before running the fitting the user is required to prepare some data and to configure the fitting kernel of `BornAgain` . The required stages are

- Preparing the sample and the simulation description (multilayer, beam, detector parameters).
- Choosing the fitting parameters.

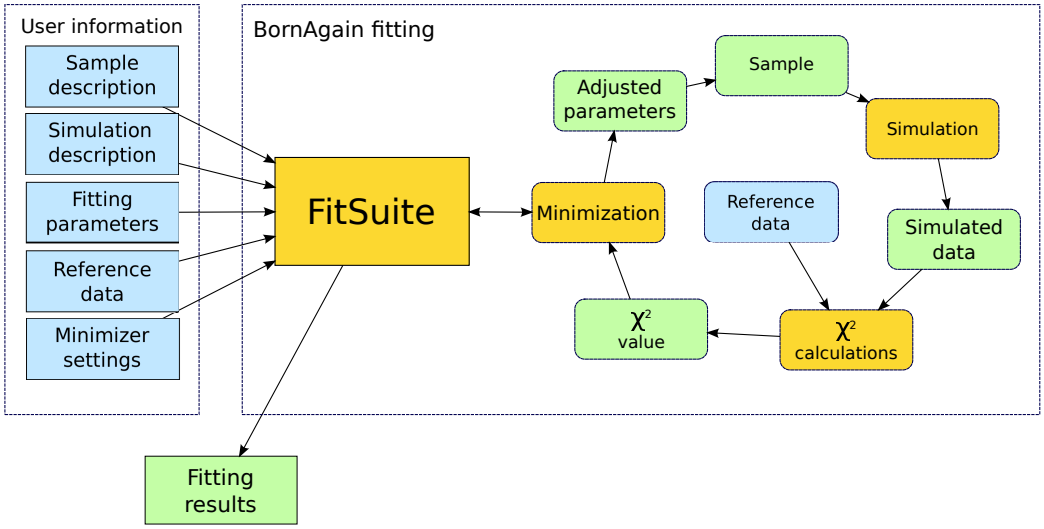


Figure 8.3: Fitting work flow.

- Loading the reference data.
- Defining the minimization settings.

The class `FitSuite` contains the main functionalities to be used for the fit and serves as the main interface between the user and the fitting work flow. The later involves iterations during which

- The minimizer makes an assumption about the optimal sample parameters.
- These parameters are propagated to the sample.
- The simulation is performed for the given state of the sample.
- The simulated data (intensities) are propagated to the χ^2 module.
- The later calculates χ^2 using the simulated and reference data.
- The value of χ^2 is propagated to the minimizer, which makes new assumptions about optimal sample parameters.

The iteration process is going on under the control of the selected minimization algorithm, without any intervention from the user. It stops

- when the maximum number of iteration steps has been exceeded,
- when the function's minimum has been reached within the tolerance window,
- if the minimizer could not improve the values of the parameters.

After the control is returned, fitting results can be retrieved. They consist in the best χ^2 value found, the corresponding optimal sample parameters and the intensity map simulated with this set of parameters.

The following parts of this section will detail each of the main stages necessary to run a fitting procedure.

8.2.0.2 Preparing the sample and the simulation description

This step is similar for any simulation using `BornAgain` (see Section **??**). It consists in first characterizing the geometry of the system: the particles (shapes, sizes, refractive indices), the different layers (thickness, order, refractive index, a possible roughness of the interface), the interference between the particles and the way they are distributed in the layers (buried particles or particles sitting on top of a layer). Then we specify the parameters of the input beam and of the output detector.

8.2.0.3 Choice of parameters to be fitted

In principle, every parameter used in the construction of the sample can be used as a fitting parameter. For example, the particles' heights, radii or the layer's roughness or thickness could be selected using the parameter pool mechanism. This mechanism is explained in detail in Section 8.1.3 and it is therefore recommended to read it before proceeding any further.

The user specifies selected sample parameters as fit parameters using `FitSuite` and its `addFitParameter` method

```
fit_suite = FitSuite()
fit_suite.addFitParameter(<name>, <initial value>, <step>, <limits>)
```

where `<name>` corresponds to the parameter name in the sample's parameter pool. By using wildcards in the parameter name, a group of sample parameters, corresponding to the given pattern, can be associated with a single fitting parameter and fitted simultaneously to get a common optimal value (see Section 8.1.3).

The second parameter <initial value> correspond to the initial value of the fitting parameter, while the third one is responsible to the initial iteration steps size. The last parameter `<AttLimits>` corresponds to the boundaries imposed on parameter value. It can be

- `limitless()` by default,
- `fixed()`,
- `lowerLimited(<min_value>)`,
- `upperLimited(<max_value>)`,
- `limited(<min_value>, <max_value>)`.

where `<min_value>` and `<max_value>` are double values corresponding to the lower and higher boundary, respectively.

8.2.0.4 Associating reference and simulated data

The minimization procedure deals with a pair of reference data (normally associated with experimental data) and the theoretical model (presented by the sample and the simulation descriptions).

We assume that the experimental data are a two-dimensional intensity matrix as function of the output scattering angles α_f and ϕ_f (see Fig. 3.2). The user is required to provide the data in the form of an ASCII file containing an axes binning description and the intensity data itself.

Remark: We recognize the importance of supporting the most common data formats. We are going to provide this feature in the following releases and welcome users' requests on this subject.

To associate the simulation and the reference data to the fitting engine, method `addSimulationAndRealData` has to be used as shown

```
fit_suite = FitSuite()
fit_suite.addSimulationAndRealData(<simulation>, <reference>, <
    chi2_module>)
```

Here `<simulation>` corresponds to a `BornAgain` simulation object with the sample, beam and detector fully defined, `<reference>` corresponds to the experimental data object obtained from the ASCII file and `<chi2_module>` is an optional parameter for advanced control of χ^2 calculations.

It is possible to call this given method more than once to submit more than one pair of `<simulation>, <reference>` to the fitting procedure. In this way, simultaneous fits of some combined data sets are performed.

By using the third parameter, `<chi2_module>`, different normalizations and weights can be applied to give user full control of the way χ^2 is calculated. This feature will be explained in Section 8.2.4.

8.2.1 Minimizer settings

`BornAgain` contains a variety of minimization engines from `ROOT` and `GSL` libraries. They are listed in Table 8.1. By default `Minuit2` minimizer with default settings will be used and no additional configuration needs to be done. The remainder of this section explains some of the expert settings, which can be applied to get better fit results.

The default minimization algorithm can be changed using `MinimizerFactory` as shown below

```
fit_suite = FitSuite()
minimizer = MinimizerFactory.createMinimizer("<Minimizer name>","<
    algorithm>")
fit_suite.setMinimizer(minimizer)
```

where `<Minimizer name>` and `<algorithm>` can be chosen from the first and second column of Table 8.1 respectively. The list of minimization algorithms implemented in `BornAgain` can also be obtained using `MinimizerFactory.printCatalogue()` command.

Minimizer name	Algorithm	Description
`Minuit2` [?]	`Migrad`	According to [?] best minimizer for nearly all functions, variable-metric method with inexact line search, a stable metric updating scheme, and checks for positive-definiteness.
	`Simplex`	simplex method of Nelder and Mead usually slower than `Migrad`, rather robust with respect to gross fluctuations in the function value, gives no reliable information about parameter errors,
	`Combined`	minimization with `Migrad` but switches to Simplex if Migrad fails to converge.
	`Scan`	not intended to minimize, just scans the function, one parameter at a time, retains the best value after each scan
	`Fumili`	optimized method for least square and log likelihood minimizations
`GSLMultiMin` [?]	`ConjugateFR`	Fletcher-Reeves conjugate gradient algorithm,
	`ConjugatePR`	Polak-Ribiere conjugate gradient algorithm,
	`BFGS`	Broyden-Fletcher-Goldfarb-Shanno algorithm,
	`BFGS2`	improved version of BFGS,
	`SteepestDescent`	follows the downhill gradient of the function at each step
`GSLLMA` [?]		Levenberg-Marquardt Algorithm
`GSLSimAn` [?]		Simulated Annealing Algorithm

Table 8.1: List of minimizers implemented in `BornAgain`.

There are several options common to every minimization algorithm, which can be changed before starting the minimization. They are handled by `MinimizerOptions` class:

```
fit_suite.getMinimizer().getOptions().setMaxFunctionCalls(10)
```

In the above code snippet, a number of “maximum function calls”, namely the maximum number of times the minimizer is allowed to call the simulation, is limited to 10.

There are also expert-level options common for all minimizers as well as a number of options to tune individual minimization algorithms. They will be explained in Section 8.2.4.

8.2.2 Running the fitting ant retrieving the results

After the initial configuration of `FitSuite` has been performed, the fitting can be started using the command

```
fit_suite.runFit()
```

Depending on the complexity of the sample and the number of free sample parameters the fitting process can take from tens to thousands of iterations. The results of the fit can be printed on the screen using the command

```
fit_suite.printResults()
```

Section 8.2.3 gives more details about how to access the fitting results.

8.2.3 Basic Python fitting example

In this section we are going to go through a complete example of fitting using `BornAgain`. Each step will be associated with a detailed piece of code written in `Python`. The complete listing of the script is given in Appendix (see Listing **??**). The script can also be found at

```
./Examples/python/fitting/ex002_FitCylindersAndPrisms/
   FitCylindersAndPrisms.py
```

This example uses the same sample geometry as in Section 8.1.2. Cylindrical and prismatic particles in equal proportion are deposited on a substrate layer, with no interference between the particles. We consider the following parameters to be unkown

- the radius of cylinders,
- the height of cylinders,
- the length of the prisms' triangular basis,
- the height of prisms.

Our reference data are a "noisy" two-dimensional intensity map obtained from the simulation of the same geometry with a fixed value of 5 nm for the height and radius of cylinders and for the height of prisms which have a 10-nanometer-long side length. Then we run our fitting using default minimizer settings starting with a cylinder's height of 4 nm, a cylinder's radius of 6 nm, a prism's half side of 6 nm and a height equal to 4 nm. As a result, the fitting procedure is able to find the correct value of 5 nm for all four parameters.

Importing Python libraries

```
1 from libBornAgainCore import *
2 from libBornAgainFit import *
```

We start from importing two `BornAgain` libraries required to create the sample description and to run the fitting.

Building the sample

```
5  def get_sample():
6      """
7      Build the sample representing cylinders and pyramids on top of
           substrate without interference.
8      """
9      # defining materials
10     m_air = HomogeneousMaterial("Air", 0.0, 0.0)
11     m_substrate = HomogeneousMaterial("Substrate", 6e-6, 2e-8)
12     m_particle = HomogeneousMaterial("Particle", 6e-4, 2e-8)
13
14     # collection of particles
15     cylinder_ff = FormFactorCylinder(1.0*nanometer, 1.0*nanometer)
16     cylinder = Particle(m_particle, cylinder_ff)
17     prism_ff = FormFactorPrism3(2.0*nanometer, 1.0*nanometer)
18     prism = Particle(m_particle, prism_ff)
19     particle_layout = ParticleLayout()
20     particle_layout.addParticle(cylinder, 0.0, 0.5)
21     particle_layout.addParticle(prism, 0.0, 0.5)
22     interference = InterferenceFunctionNone()
23     particle_layout.addInterferenceFunction(interference)
24
25     # air layer with particles and substrate form multi layer
26     air_layer = Layer(m_air)
27     air_layer.addLayout(particle_layout)
28     substrate_layer = Layer(m_substrate)
29     multi_layer = MultiLayer()
30     multi_layer.addLayer(air_layer)
31     multi_layer.addLayer(substrate_layer)
32     return multi_layer
```

The function starting at line 5 creates a multilayered sample with cylinders and prisms using arbitrary 1nm value for all size's of particles. The details about the generation of this multilayered sample are given in Section 8.1.2.

Creating the simulation

```
35 def get_simulation():
36     """
```

```
37      Create GISAXS simulation with beam and detector defined
38      """
39      simulation = Simulation()
40      simulation.setDetectorParameters(100, -1.0*degree, 1.0*degree,
            100, 0.0*degree, 2.0*degree)
41      simulation.setBeamParameters(1.0*angstrom, 0.2*degree, 0.0*
            degree)
42      return simulation
```

The function starting at line 35 creates the simulation object with the definition of the beam and detector parameters.

Preparing the fitting pair

```
45  def run_fitting():
46      """
47      run fitting
48      """
49      sample = get_sample()
50      simulation = get_simulation()
51      simulation.setSample(sample)
52
53      real_data = IntensityDataIOFactory.readIntensityData('
            refdata_fitcylinderprisms.int.gz')
```

Lines 49- 51 generate the sample and simulation description and assign the sample to the simulation. Our reference data are contained in the file `'refdata_fitcylinderprisms.int.gz'`. This reference had been generated by adding noise on the scattered intensity from a numerical sample with a fixed length of 5 nm for the four fitting parameters (*i.e.* the dimensions of the cylinders and prisms). Line 53 creates the real data object by loading the ASCII data from the input file.

Setting up FitSuite

```
55      fit_suite = FitSuite()
56      fit_suite.addSimulationAndRealData(simulation, real_data)
57      fit_suite.initPrint(10)
```

Line 55 creates a `FitSuite` object which provides the main interface to the minimization kernel of `BornAgain` . Line 56 submits simulation description and real data pair to the subsequent fitting. Line 57 sets up `FitSuite` to print on the screen the information about fit progress once per 10 iterations.

```
60      fit_suite.addFitParameter("*FormFactorCylinder/height", 4.*
            nanometer, 0.01*nanometer, AttLimits.lowerLimited(0.01))
61      fit_suite.addFitParameter("*FormFactorCylinder/radius", 6.*
            nanometer, 0.01*nanometer, AttLimits.lowerLimited(0.01))
62      fit_suite.addFitParameter("*FormFactorPrism3/height", 4.*
            nanometer, 0.01*nanometer, AttLimits.lowerLimited(0.01))
```

```
63      fit_suite.addFitParameter("*FormFactorPrism3/length", 12.*
            nanometer, 0.02*nanometer, AttLimits.lowerLimited(0.01))
```

Lines 60– 63 enter the list of fitting parameters. Here we use the cylinders' height and radius and the prisms' height and side length. The cylinder's length and prism half side are initially equal to 4 nm, whereas the cylinder's radius and the prism half side length are equal to 6 nm before the minimization. The iteration step is equal to 0.01 nm and only the lower boundary is imposed to be equal to 0.01 nm.

Running the fit and accessing results

```
66      fit_suite.runFit()
67
68      print "Fitting completed."
69      fit_suite.printResults()
70      print "chi2:", fit_suite.getMinimizer().getMinValue()
71      fitpars = fit_suite.getFitParameters()
72      for i in range(0, fitpars.size()):
73          print fitpars[i].getName(), fitpars[i].getValue(), fitpars[i
                ].getError()
```

Line 66 shows the command to start the fitting process. During the fitting the progress will be displayed on the screen. Lines 69– 73 shows different ways of accessing the fit results.

More details about fitting, access to its results and visualization of the fit progress using matplotlib libraries can be learned from the following detailed example

```
./Examples/python/fitting/ex002_FitCylindersAndPrisms/
   FitCylindersAndPrisms_detailed.py
```

8.2.4 Advanced fitting

8.2.4.1 Affecting $\chi 2$ calculations

8.2.4.2 Simultaneous fits of several data sets

8.2.4.3 Using fitting strategies

8.2.4.4 Masking the real data

8.2.4.5 Tuning fitting algorithms

8.2.4.6 Fitting with correlated sample parameters

8.2.5 How to get the right answer from fitting

One of the main difficulties in fitting the data with the model is the presence of multiple local minima in the objective function. Many problems can cause the fit to fail, for example:

- an unreliable physical model,
- an unappropriate choice of objective function
- multiple local minima,
- an unphysical behavior of the objective function, unphysical regions in the parameters space,
- an unreliable parameter error calculation in the presence of limits on the parameter value,
- an exponential behavior of the objective function and the corresponding numerical inaccuracies, excessive numerical roundoff in the calculation of its value and derivatives,
- large correlations between parameters,
- very different scales of parameters involved in the calculation,
- not positive definite error matrix even at minimum.

The given list, of course, is not only related to `BornAgain` fitting. It remains applicable to any fitting program and any kind of theoretical model. Below we give some recommendations which might help the user to achieve reliable fit results.

General recommendations

- initially choose a small number of free fitting parameters,
- eliminate redundant parameters,
- provide a good initial guess for the fit parameters,
- start from the default minimizer settings and perform some fine tuning after some experience has been acquired,
- repeat the fit using different starting values for the parameters or their limits,
- repeat the fit, fixing and varying different groups of parameters,

to be continued...

8.3 User API

8.3.1 IntensityData

The `IntensityData` object stores the simulated or real intensity data together with the axes definition of the detector in BornAgain's internal format. During the simulation setup it is created automatically when the user specifies the detector characteristics and is filled with the simulated intensities after the simulation is completed.

```
1 simulation = Simulation()
2 simulation.setDetectorParameters(10, -5.0*degree, 5.0*degree, 5,
      0.0*degree, 1.0*degree)
3 ...
4 simulation.runSimulation()
5 intensity = simulation.getIntensityData()
```

The `IntensityData` object retrieved in line 5 corresponds to the two dimensional detector pixel array as shown in Fig. 8.4.

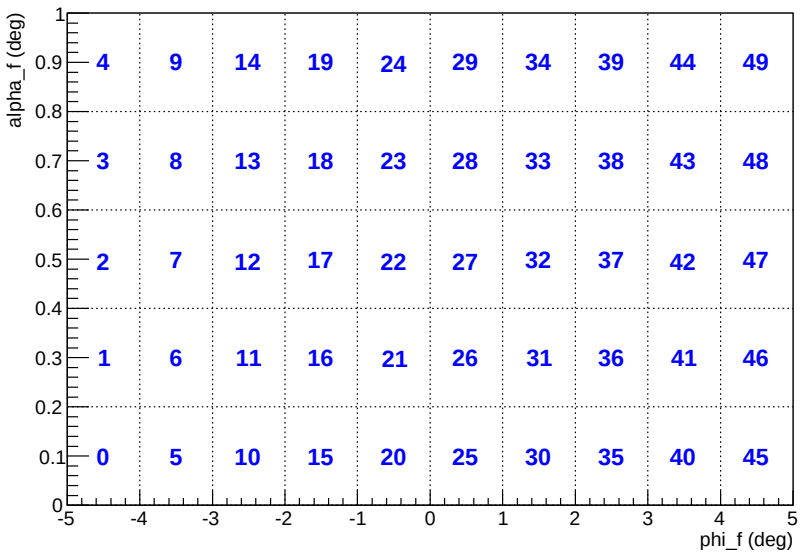


Figure 8.4: The axes layout of IntensityData object.

The x-axis and y-axis of the figure correspond to the ϕ_f and α_f axes of the detector. The x-axis is divided into 10 bins, with low edge of the first bin set to $-5.0\,\text{deg}$ and upper edge of the last bin set to $+5.0\,\text{deg}$. The y-axis is divided into 5 bins, with low edge of the first bin set to $0.0\,\text{deg}$ and upper edge of the last bin set to $1.0\,\text{deg}$. There are 50 bins in total (they are marked on the plot with indexes from 0 to 49), each bin will contain one intensity value.

During a standard simulation (i.e. no Monte-Carlo integration involved) intensities are calculated for ϕ_f, α_f values corresponding to the bin centers, e.g. the intensity stored in bin#42 will correspond to $\phi_f = 3.5\,\text{deg}, \alpha_f = 0.5\,\text{deg}$.

The `IntensityData` object is not intended for direct usage from Python API. The idea is that the API provides the user with the possibility to export the data from BornAgain internal format to the format of his choice as well as import user's data into BornAgain. For the moment this functionality is limited to a few options explained below. We encourage users feedback to implement the support of most requested formats.

8.3.1.1 Import/export of intensity data

For the moment we provide following options:

- Import/export of `IntensityData` object from/to `numpy` array.
- Import/export of `IntensityData` object from/to text file.

Export to numpy array To export intensity data into `numpy` array the method `getArray()` should be used on `IntensityData` object as shown in line 2 of following code snippet.

```
1 intensity = simulation.getIntensityData()
2 array = intensity.getArray()
3 ...
4 pylab.imshow(numpy.rot90(array, 1))
5 pylab.show()
```

For the detector settings defined in the previous paragraph the dimensions of the resulting array will be (10,5). By using `numpy` indexes the user can get access to the intensity values, e.g. `array[0][0]` corresponds to the intensity in bin#0 of Fig. 8.4, `array[0][4]` to bin#4, `array[1][0]` to bin#5, `array[8][2]` to bin#42, `array[9][4]` to bin#49.

To plot this resulting numpy array with `matplotlib` it has to be rotated counter-clockwise to match `matplotlib` conventions as shown in line 4.

8.3.1.2 Importing from numpy array

To use fitting the user has to load experimental data into BornAgain fitting kernel. To read experimental data the user has to create IntensityData object, fill it with the experimental intensity values and pass this object to the fitting kernel.

First, the user creates empty `IntensityData` as shown in line 1 of the following code snippet.

```
1 data = IntensityData()
2 data.addAxis(FixedBinAxis("phi_f", 10, -5.0*degree, 5.0*degree))
3 data.addAxis(FixedBinAxis("alpha_f", 5, 0.0*degree, 1.0*degree))
```

```
4  ...
5  array = numpy.zeros((10, 5)) # fill array with experimental
       intensities
6  ...
7  data.setRawDataVector(array.flatten().tolist())
8
9  fitSuite = FitSuite()
10 fitSuite.addSimulationAndRealData(simulation, data)
```

In lines 2, 3 two axes with fixed bin sizes are defined to represent the detector layout as shown in Fig. 8.4. The constructor of `FixedBinAxis` object has the following signature

```
FixedBinAxis(title, nbins, min_angle, max_angle)
```

The created `IntensityData` object has to be filled with experimental intensities using `numpy` array prepared by the user (lines 5- 7). In lines 9,10 the fitting kernel is created and initialized with `Simulation` object and `IntensityData` object representing the experimental data.

8.3.1.3 Saving intensity data to text file.

The special class `IntensityDataIOFactory` is intended for saving the intensity data in different datafile formats. For the moment, it only supports saving the data in specific BornAgain's text files (the file extention `*.int`).

```
1 intensity = simulation.getIntensityData()
2 IntensityDataIOFactory.writeIntensityData(intensity, 'file_name.int'
      )
```

8.3.1.4 Reading intensity data from a text file.

The same class is also intended for reading intensity data from files with different formats. For the moment, it only supports reading the data from text files of special BornAgain's format (the file extention `*.int`).

```
1 intensity = IntensityDataIOFactory.readIntensityData('file_name.int'
      )
```