# BornAgain tutorial, part II

- Main concept of fitting
- Basic fitting in GUI
- Fitting game

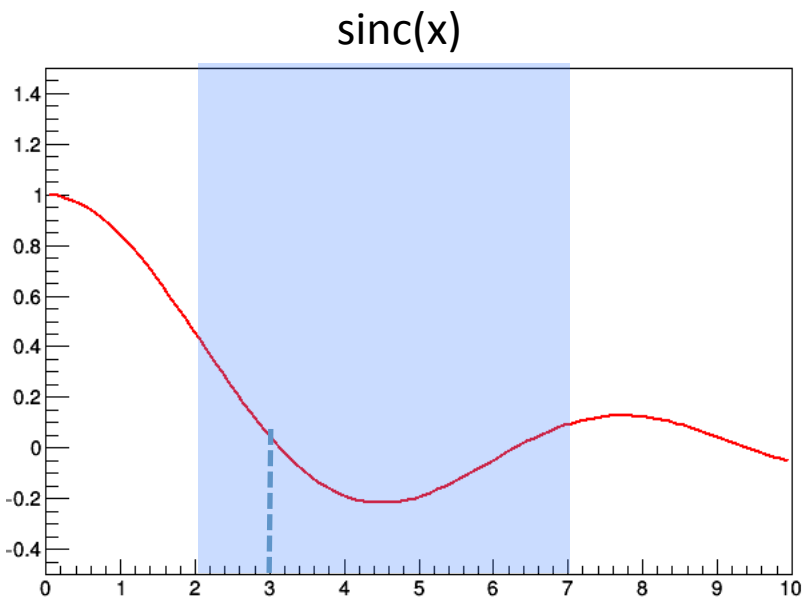
- Advanced fitting in Python
- Import user data
- Fitting game advanced

- Submitting pull request
- Simulating polarized neutrons

# Main concept of fitting

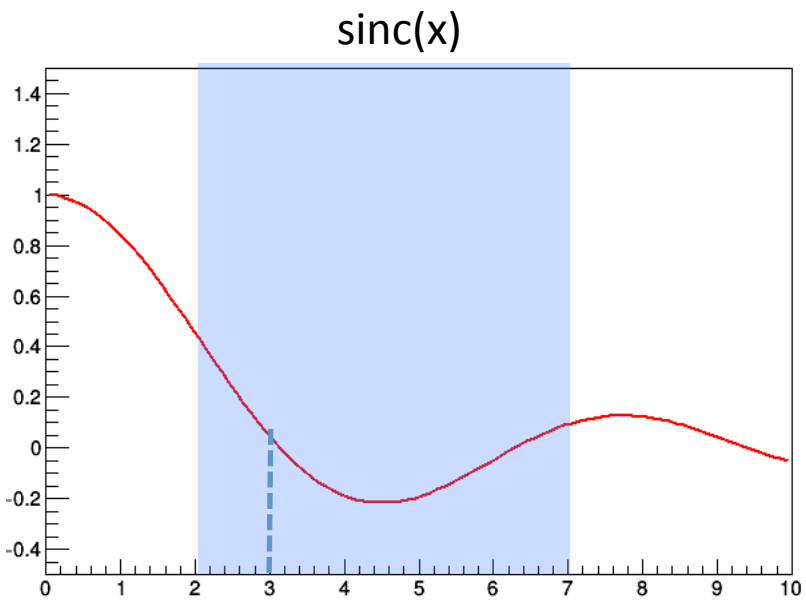
# Minimization

Finding the minimum of a multi-argument function in a given region.



# Minimization

Finding the minimum of a multi-argument function in a given region.




```
float objective_function(p):
    return sinc(p)
```

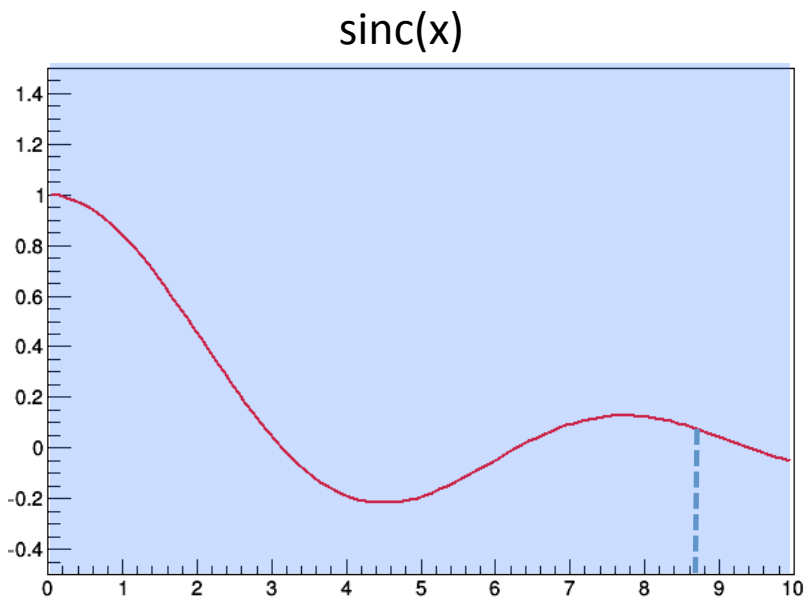
Number of parameters to test: 1
Starting value: 3.0
Limits: [2.0, 7.0]

Minimizer

Found minimum at:
p = 4.4 +/- 0.0001

# Minimization

Finding the minimum of a multi-argument function in a given region.




```
float objective_function(p):
    return sinc(p)
```

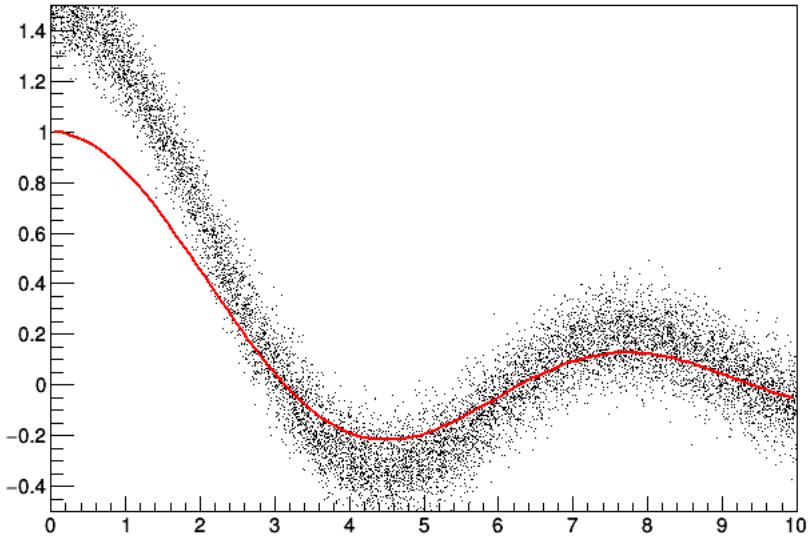
Number of parameters to test: 1
Starting value: 3.0
Limits: [2.0, 7.0]

Minimizer

Found minimum at:
p = 4.4 +/- 0.0001

# Fitting

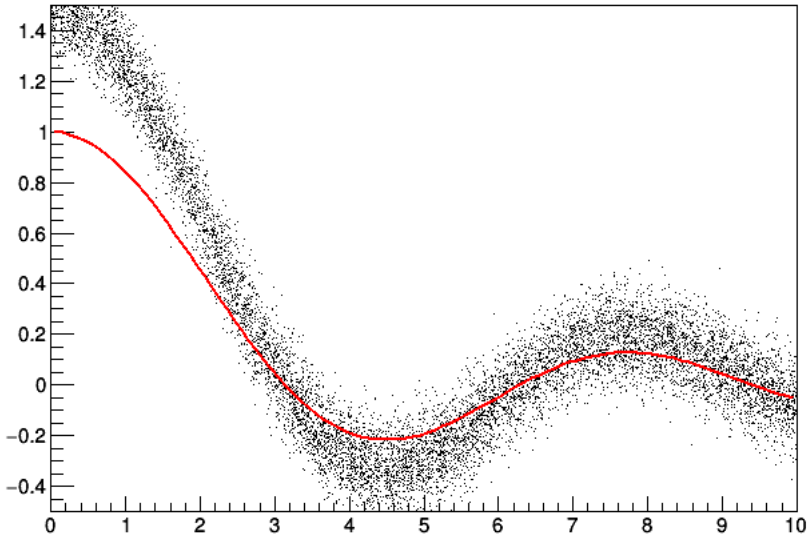
Finding the best set of parameter values for a model function to represent experimental data according to some criteria.




- Data: set of (x,y) points
- Model: p1*sinc(p2*x)
- Fit criteria: minimum of Chi2

# Fitting

Finding the best set of parameter values for a model function to represent experimental data according to some criteria.




- Data: set of (x,y) points
- Model: p1*sinc(p2*x)
- Fit criteria: minimum of Chi2

```
float objective_function(p1, p2):
    foreach (x,y)
        chi2 += (y - p1*sinc(p2*x))**2
    return chi2
```

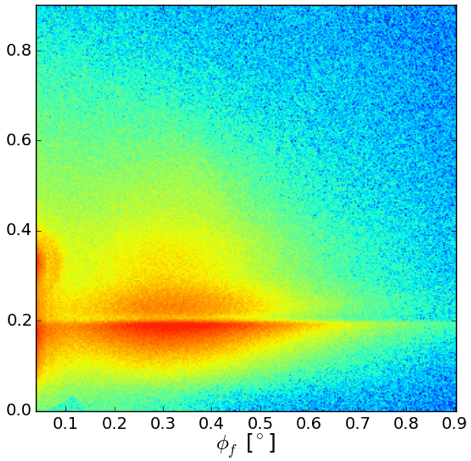
Number of fit parameters: 2
Starting values: p1=1.0; p2=1.0
Limits: p1 = [0.5, 1.5], p2 = [0.9, 1.1]

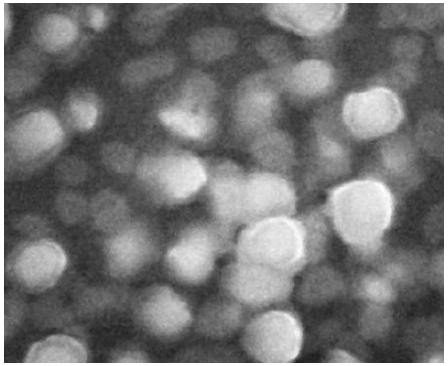
Minimizer

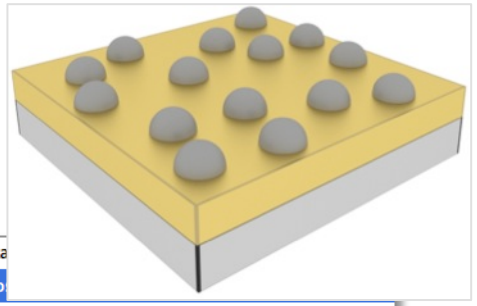
Found minimum at:
p1 = 1.2 +/- 0.025
p2 = 1.0 +/- 0.001

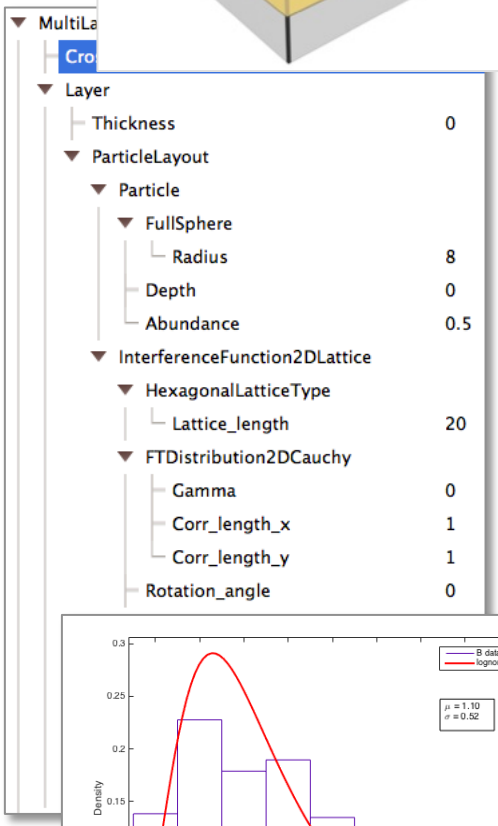
# Fitting of GISAS data*

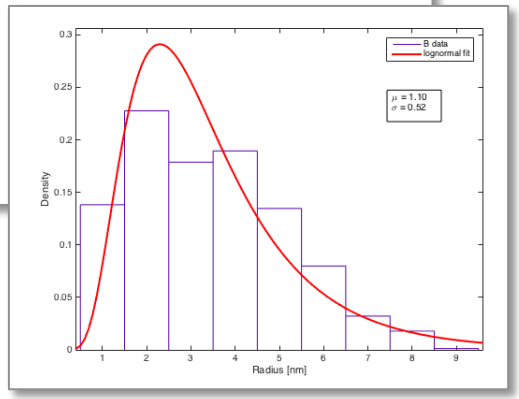
* In ideal world



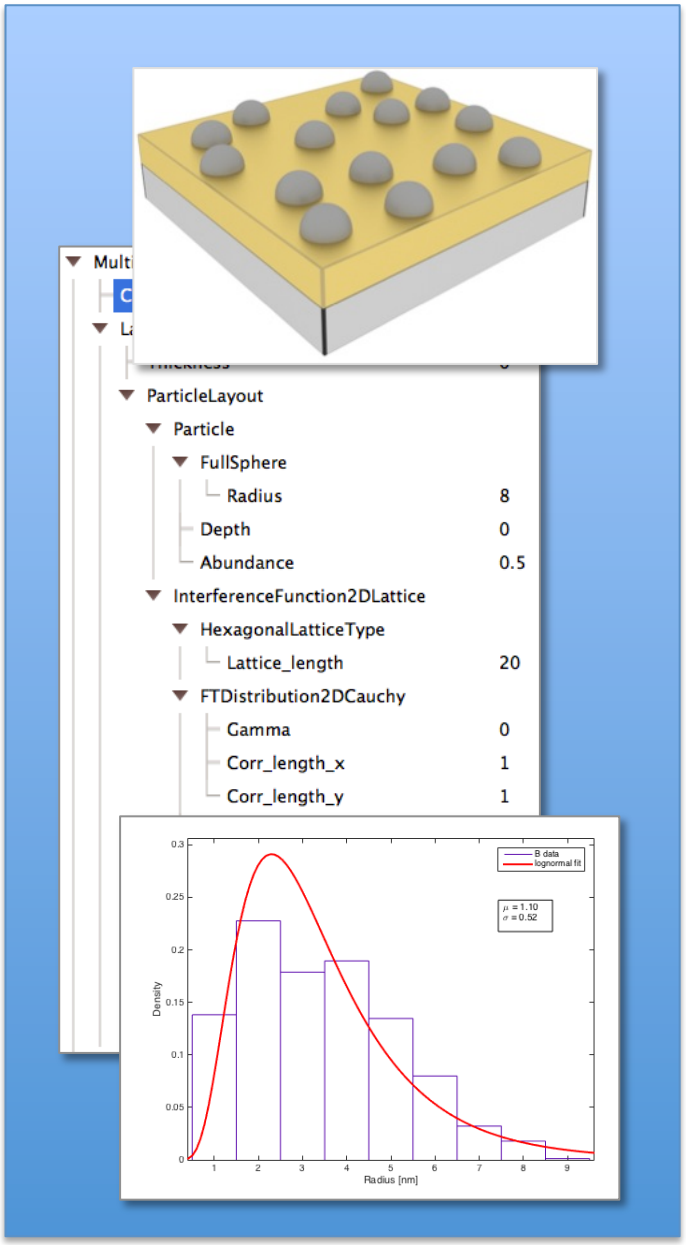



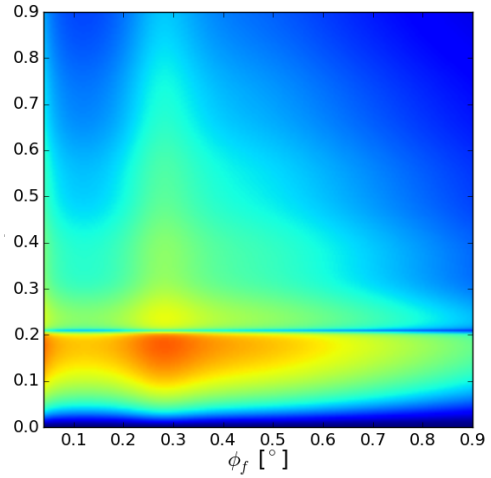


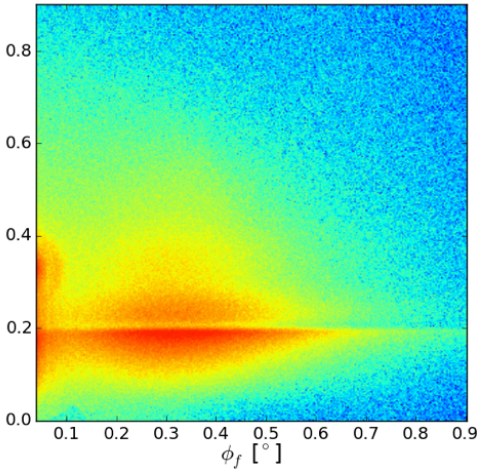


# Fitting of GISAS data




float objective_function(sample parameters):

> call simulation for given set of parameters

> compare simulated image against reference

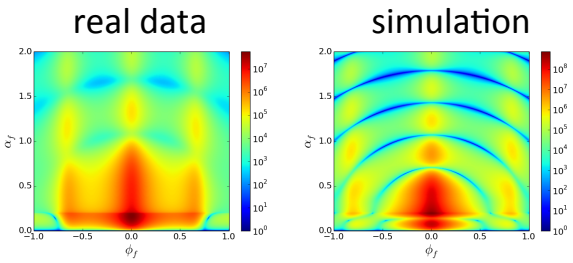
return similarity_metric

Minimizer

# Objective function

*Also known as similarity metric*

- defined in sample parameters space
- One of most common – chi2 function




bin intensity
from experiment

bin intensity provided
by simulation

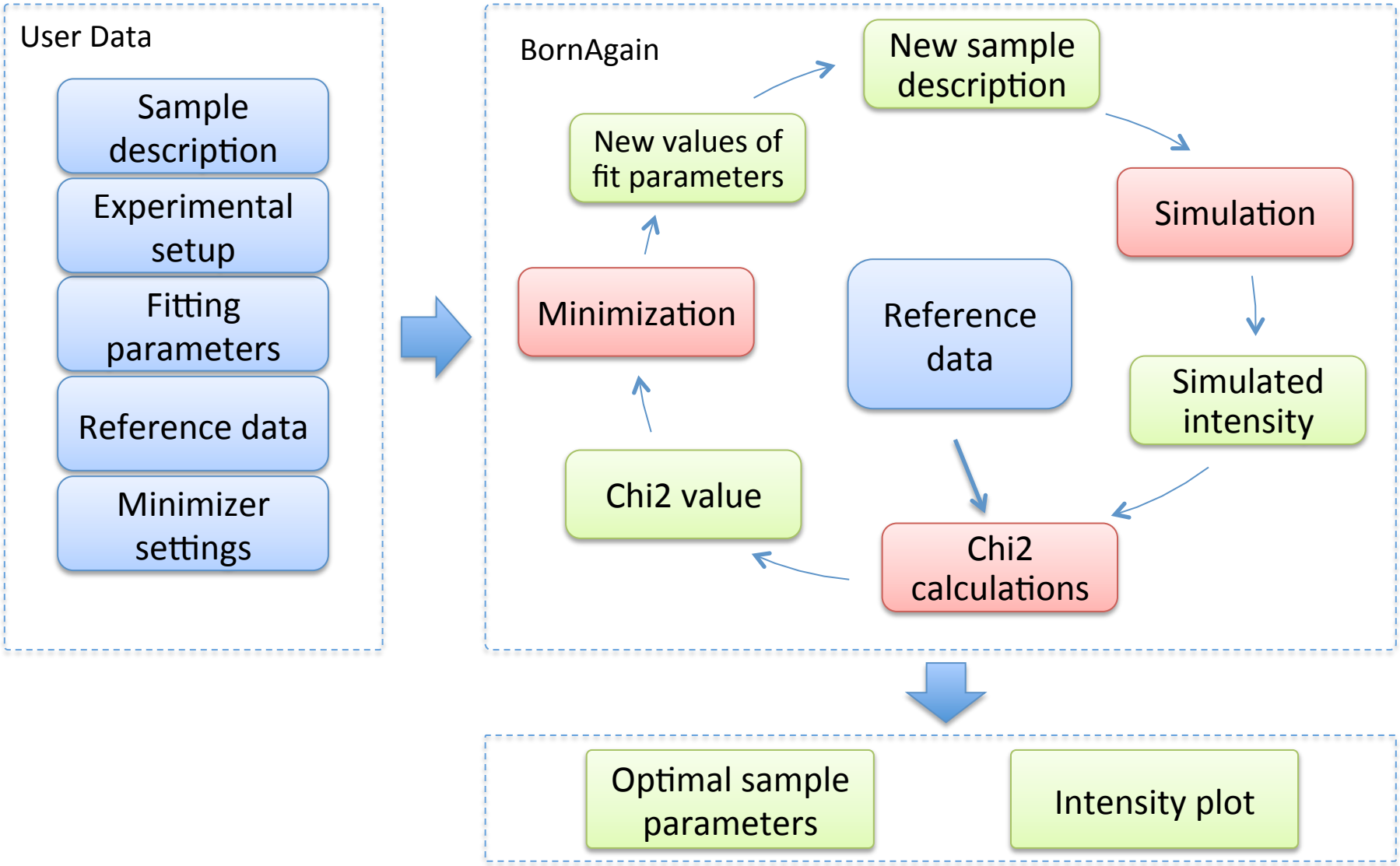
$$\chi^2(\mathbf{p}) = \frac{1}{d}\sum_{i=1}^{N}\frac{(a_i - f(\mathbf{x_i}, \mathbf{p}))^2}{\sigma^2}$$

sample parameters
(radius, lattice_length)

sum over detector bins

# Fitting in BornAgain

# Fitting in BornAgain



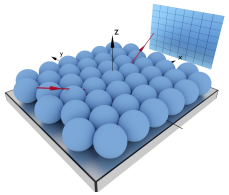
# Demo: basic fitting in GUI

# Local minima problem

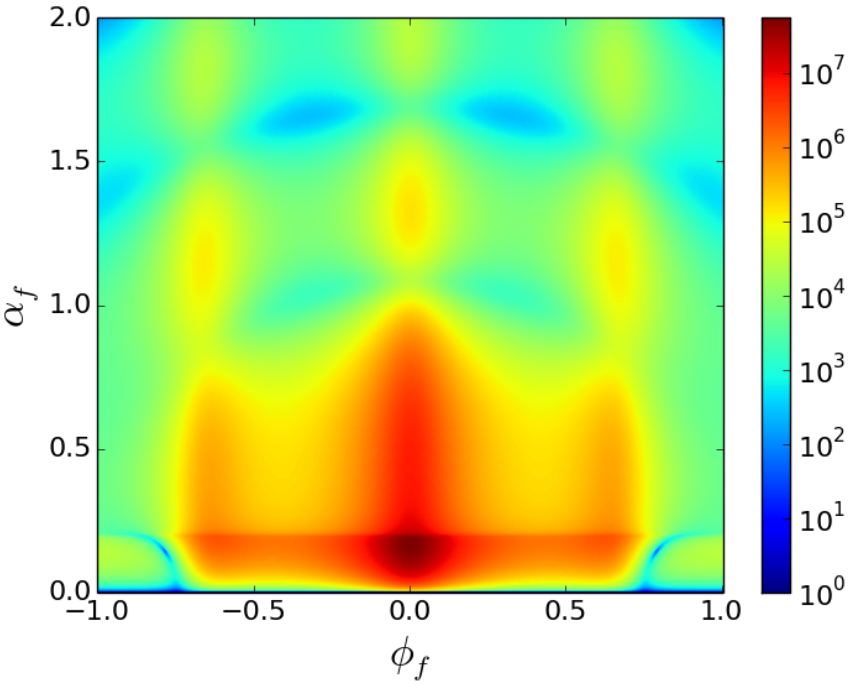
# Toy fitting example

## Spherical nano particles at hexagonal lattice

- Fitting radius of spheres and lattice length



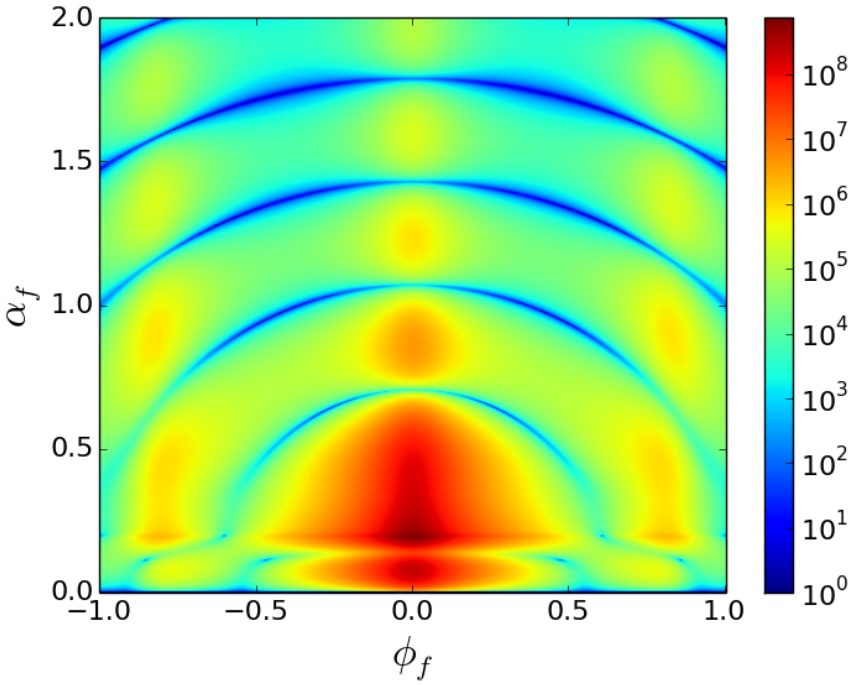
"Real" data




- Radius = 5 nm
- Lattice length = 10 nm

*Values we are going to find during the fit*

Simulated data

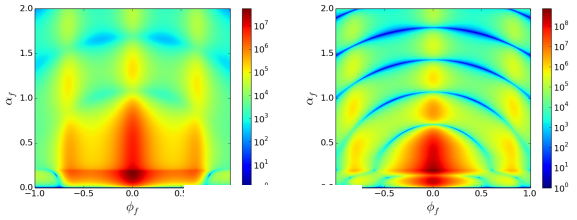



- Radius = 8 nm
- Lattice length = 8 nm

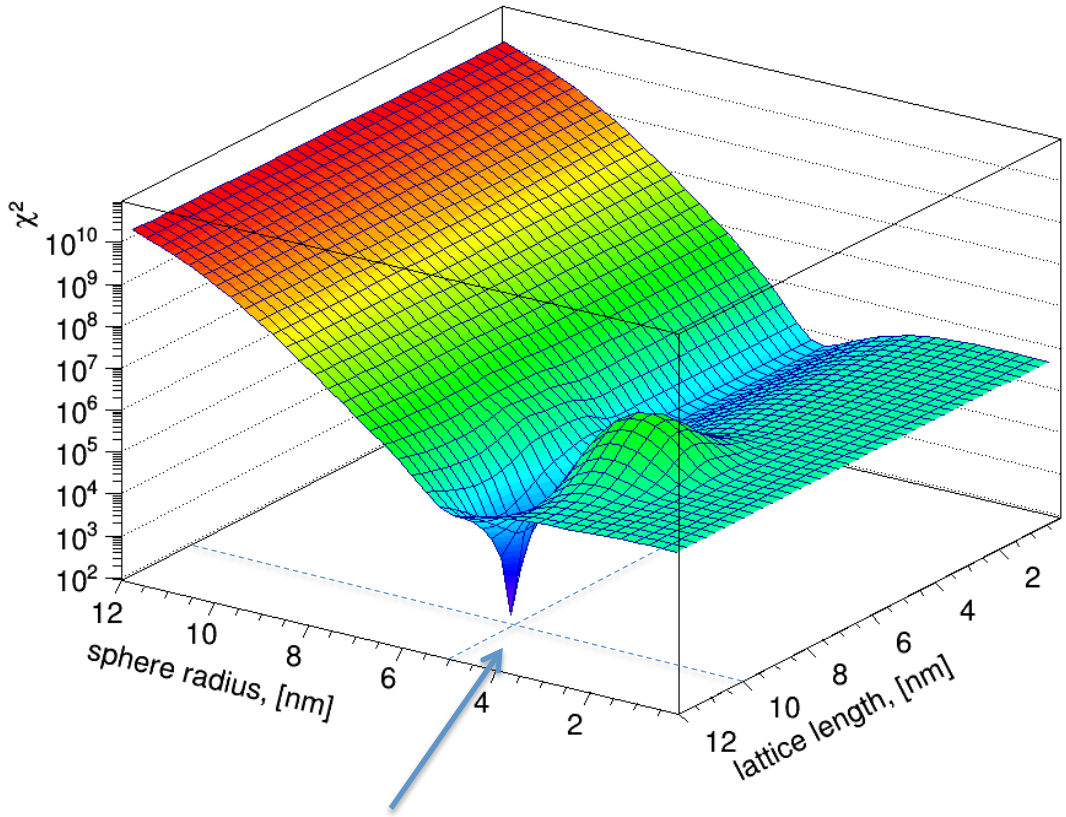
*Starting values of our fit parameters*

# Toy fitting example

How does objective function looks like?



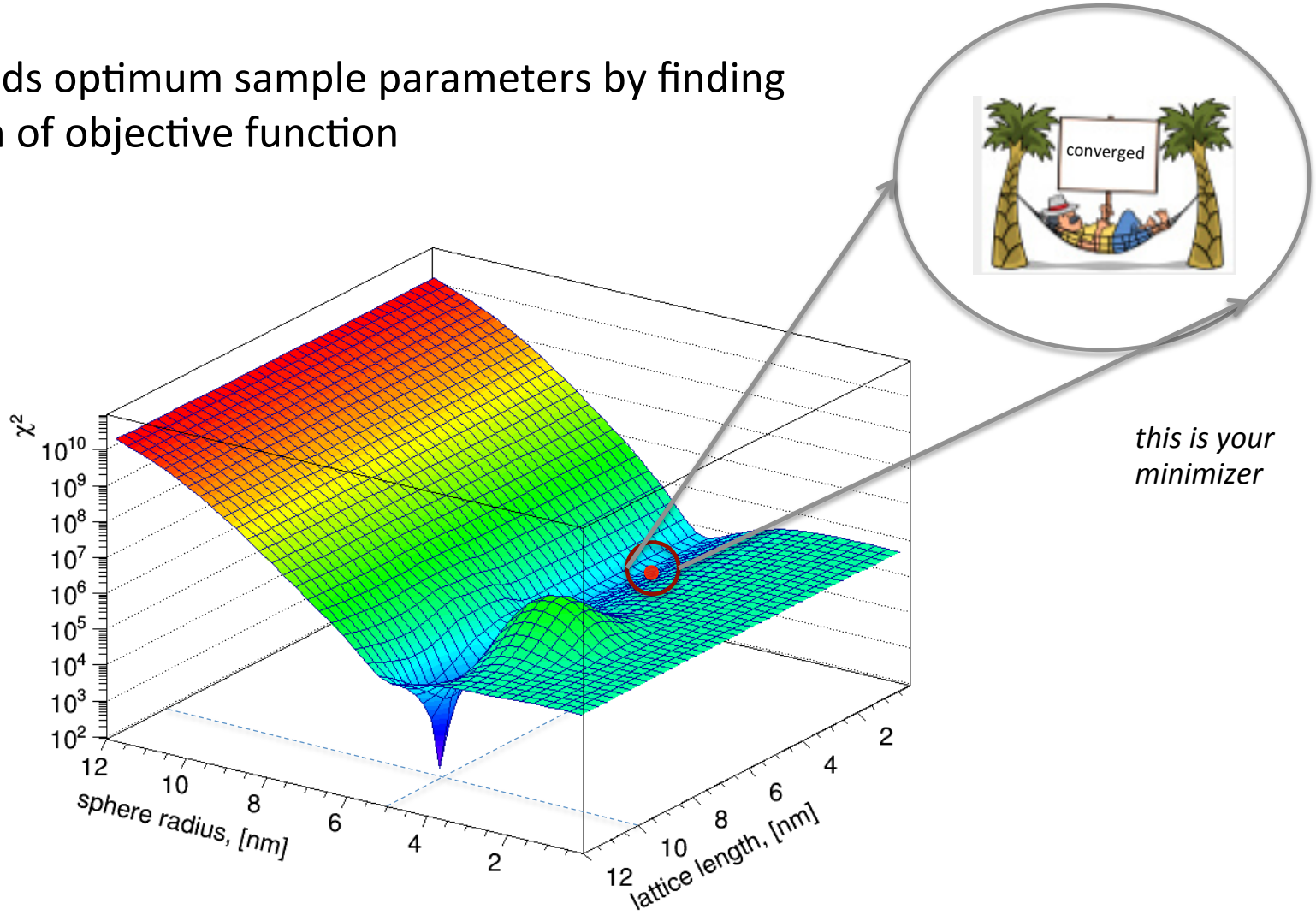
$$\chi^2(\mathbf{p})$$




- Minimum of the function corresponds to optimal sample parameters

# Toy fitting example

Minimizer finds optimum sample parameters by finding the minimum of objective function




*this is your minimizer*

Depending on starting values of fit parameters the minimizer will find one of two local minima.

# Local minima problem solution

- Use reliable model of the sample
- Provide a good initial guess for the fit parameters
- Use small number of fit parameters
- Avoid correlated fit parameters
- Use stochastic minimizers to explore large parameter space

- Invent new GISAS images similarity metric  instead of chi2

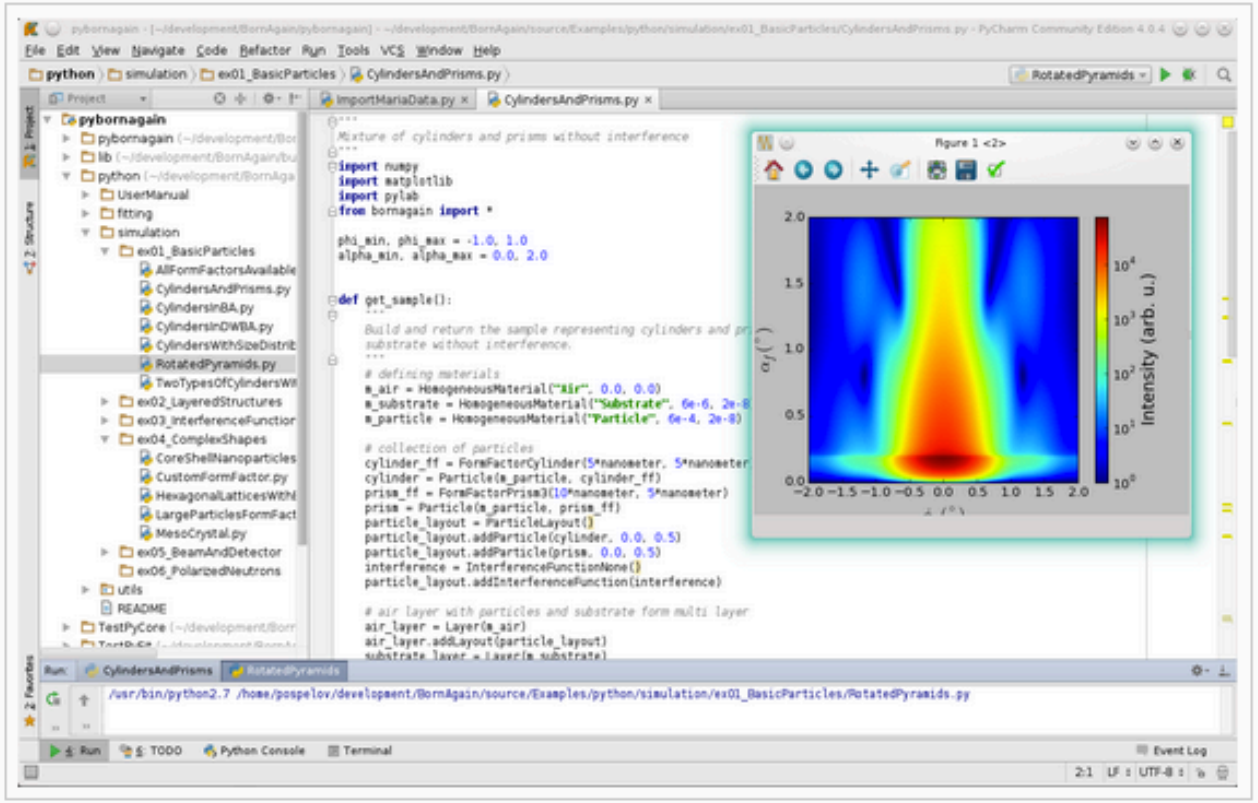
# Fitting game: nanodots

# Working with Python scripts

# Working with Python scripts

BornAgain can be used from Python to run GISAS simulation or to do a fit of the data. This is more flexible than using BornAgain from the Graphical User Interface, and it will be a natural solution for those who have reached the limitations of our GUI.

The user creates a Python script with a sample description and simulation settings. The user then runs the simulation by executing the script in the Python interpreter and assesses the simulation results using the graphics or analysis library of his choice, e.g. Python + numpy + matplotlib.

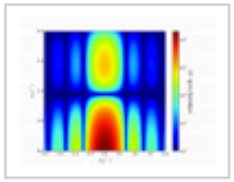
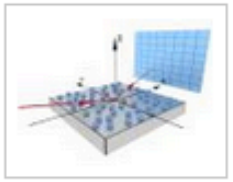
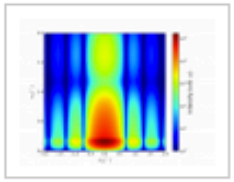
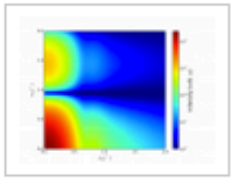
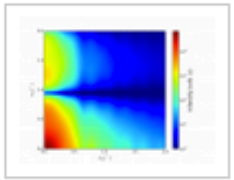
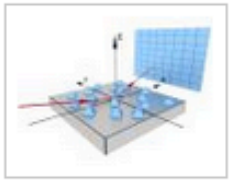
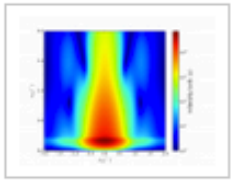



SEARCH

DOCUMENTATION

- Welcome to BornAgain
- Functionality overview
- Architectural overview
- Installation instructions
- Working with BornAgain
  - Using Graphical User Interface
  - Working with Python scripts
    - How to setup a PyCharm project
    - Basic simulation tutorial
    - Detector types
    - Accessing simulation results
    - Particles positioning
    - Particles rotation
    - Particle composition
    - Working with sample parameters
    - Introduction to fitting
    - Basic fitting tutorial
    - Importing experimental data
- Examples of BornAgain usage
- API References
- Troubleshooting
- Frequently Asked Questions

# Embedded Particles

| | |
|---|---|
|   | Cylinders in Born Approximation |
|   | Cylinders in Distorted Wave Born Approximation |
|   | Cylinders with size distribution |
|   | Two types of cylinders with size distribution |
|   | Rotated pyramids |


## SEARCH

## DOCUMENTATION

- Welcome to BornAgain
- Functionality overview
- Architectural overview
- Installation instructions
- Working with BornAgain
- Examples of BornAgain usage
  - Embedded Particles
    - Cylinders in Born Approximation
    - Cylinders in Distorted Wave Born Approximation
    - Cylinders with size distribution
    - Two types of cylinders with size distribution
    - Rotated pyramids
    - Cylinders and Prisms
    - All available form factors
  - Layered Structures
  - Interference Functions
  - Complex Shapes
  - Beam and Detector
  - Fitting
  - Miscellaneous
- API References
- Troubleshooting
- Frequently Asked Questions

Basic simulation example

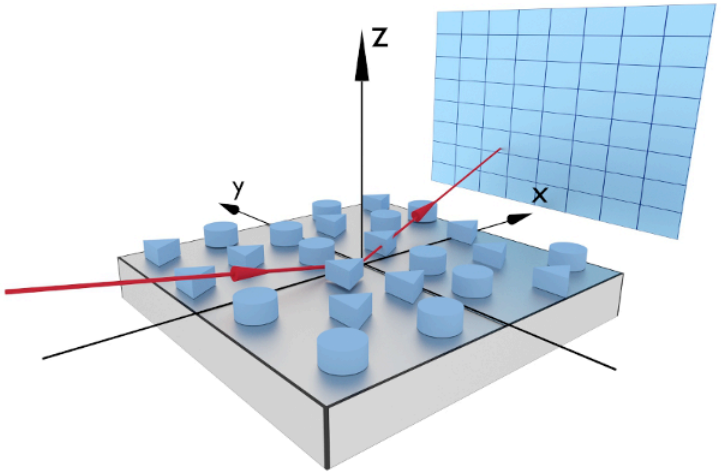
https://github.com/scgmlz/BornAgain-tutorial/blob/master/bornagain-python/fitting/advanced-tutorial/step0_simulation.py

How to plot chi2 in two fit parameters space

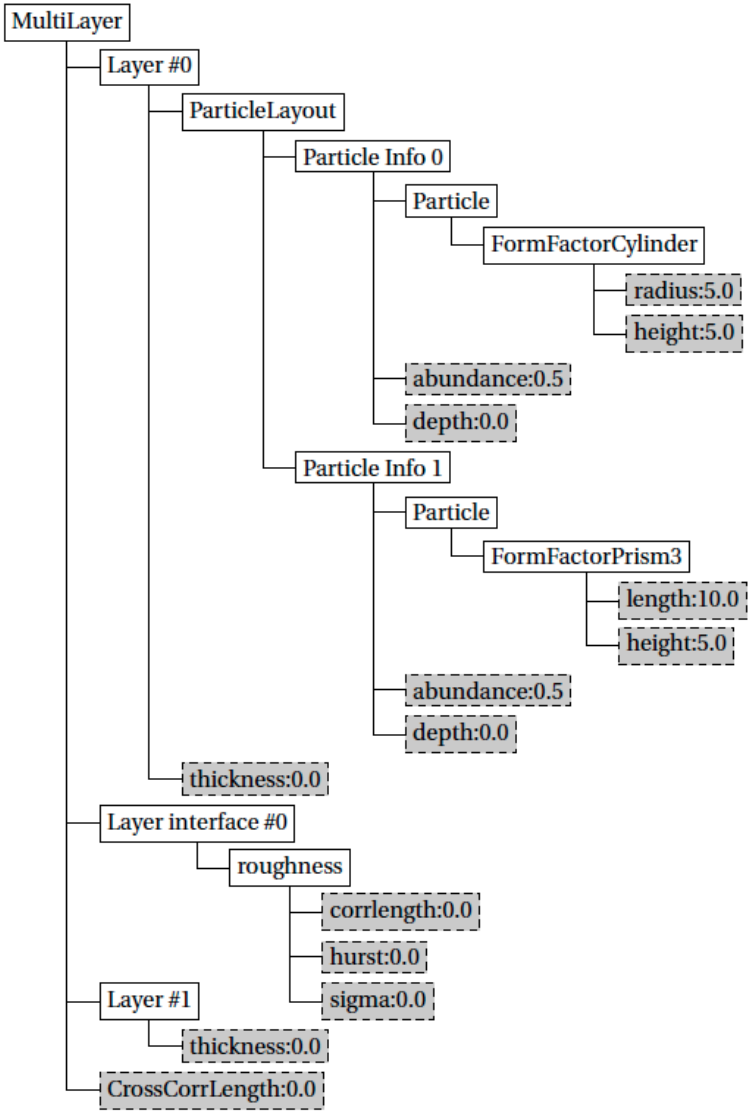
https://github.com/scgmlz/BornAgain-tutorial/blob/master/bornagain-python/fitting/advanced-tutorial/step1_chi2.py

Sample parameters

# Sample parameters available for fitting




- Sample is represented by a hierarchical tree of objects
- Every number used for sample construction can be used as a fit parameter*




* Current limitations
One fit parameter can fit one sample parameter only
Some of sample parameters can't become (yet) a fit parameter (e.g. PDF distributions)

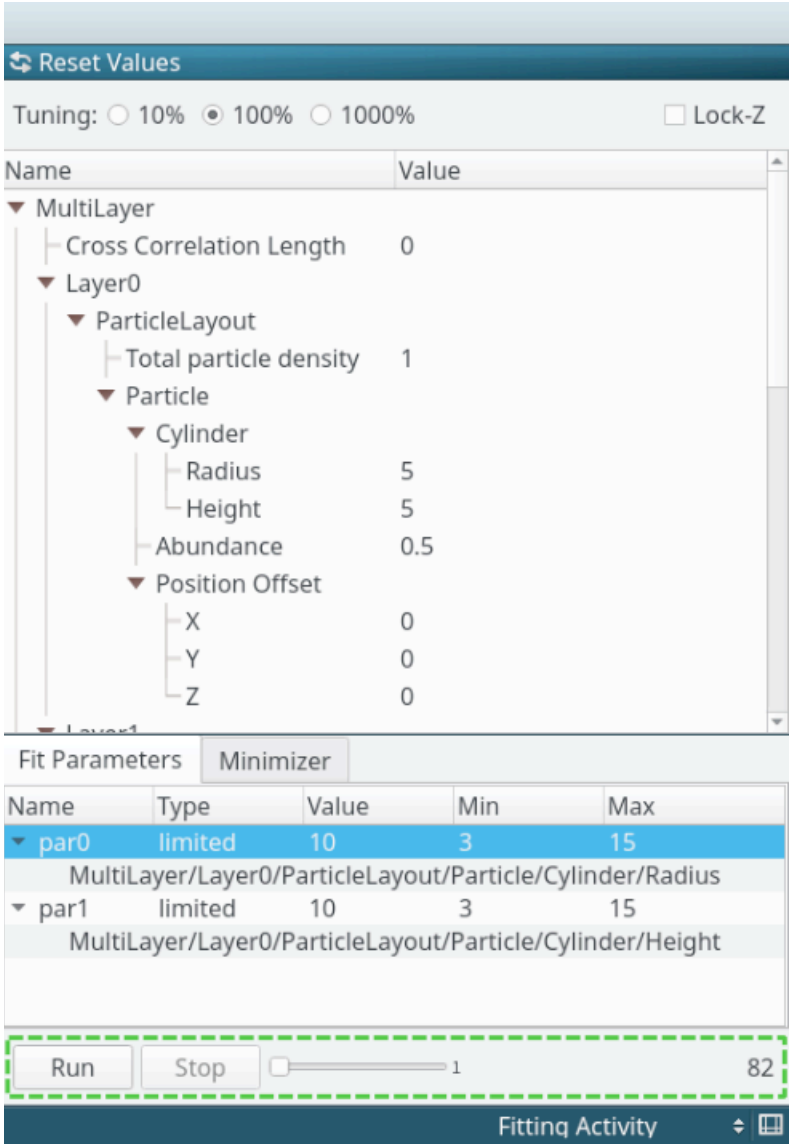
# Fit parameters construction in GUI

- Sample parameters used in real time tuning widget become a fit parameter via drag-and-drop

## Current limitations

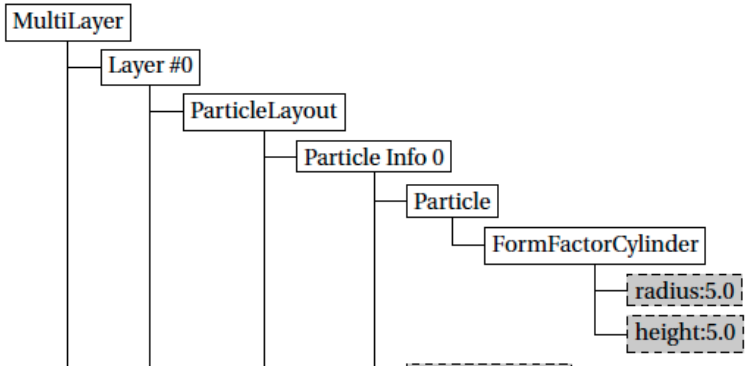
- One fit parameter can fit one sample parameter only
- Some of sample parameters can't become (yet) a fit parameter (e.g. PDF distributions)

This limitations will be addressed in BornAgain-1.8



# Fit parameters construction in Python

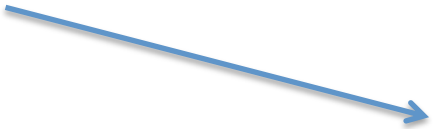
- To define fit parameter in Python one have to know corresponding sample parameter name
- The name is composed from names of all parents




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

"/Cylinder/Radius"
"/Cylinder/Height"

```
m_particle = ba.HomogeneousMaterial("Particle", 6e-4, 2e-8)

cylinder = ba.Particle(m_particle, cylinder_ff)
cylinder.printParameters()
```



"/Particle/PositionX"
"/Particle/PositionY"
"/Particle/PositionZ"
"/Particle/Cylinder/Radius"
"/Particle/Cylinder/Height"

How to construct a simulation and then change some of sample parameters at runtime

https://github.com/scgmlz/BornAgain-tutorial/blob/master/bornagain-python/fitting/advanced-tutorial/step2_sample_parameters.py

# Basic fitting from Python

# Basic fitting from Python

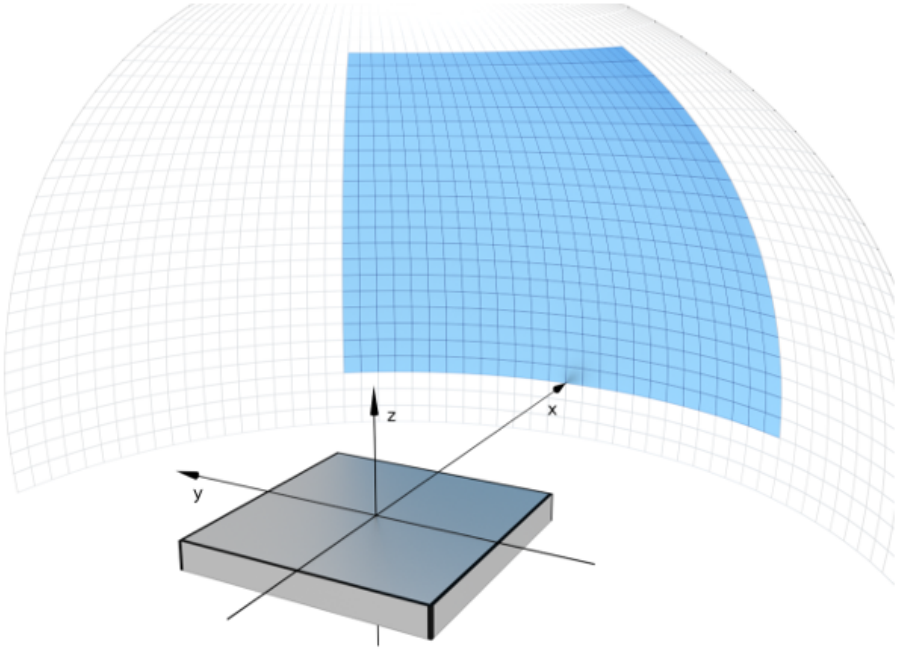
```
# construct sample
sample = MultiLayer()

# construct simulation
simulation = GISASSimulation()
simulation.setSample(sample)

# read real data from disk
real_data = IntensityDataIOFactory.readIntensityData("filename.txt")

# setup fit kernel
fit_suite = FitSuite()
fit_suite.addSimulationAndRealData(simulation, real_data)
fit_suite.addFitParameter("*/MultiLayer/")

# run fit
fit_suite.runFit()
```

https://github.com/scgmlz/BornAgain-tutorial/blob/master/bornagain-python/fitting/advanced-tutorial/step3_basicfit.py

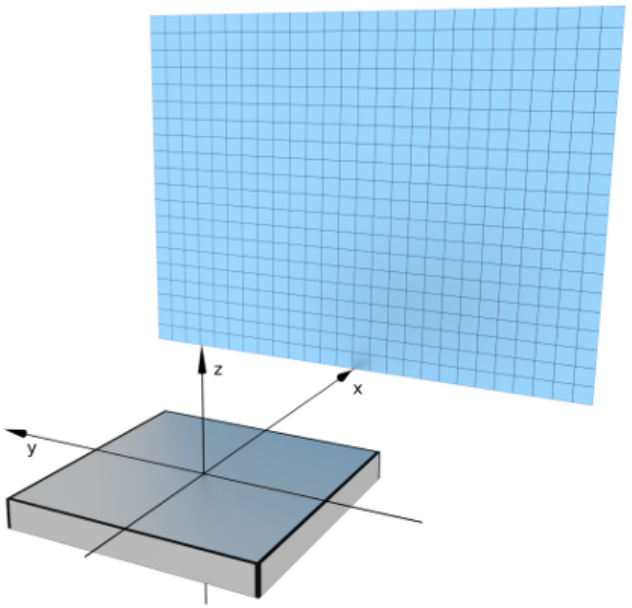
# Detector types

# Detector types

Spherical detector

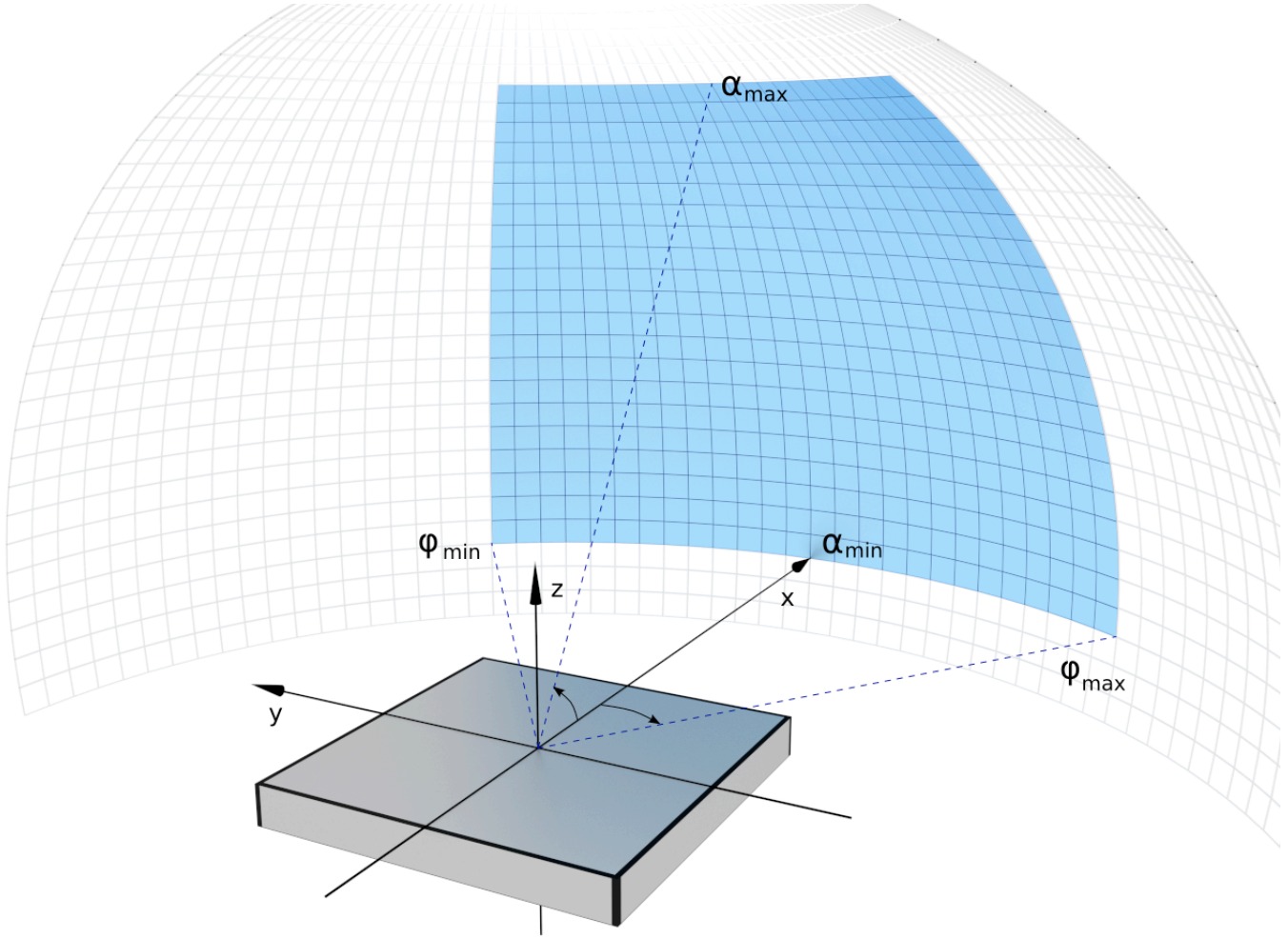



Rectangular detector



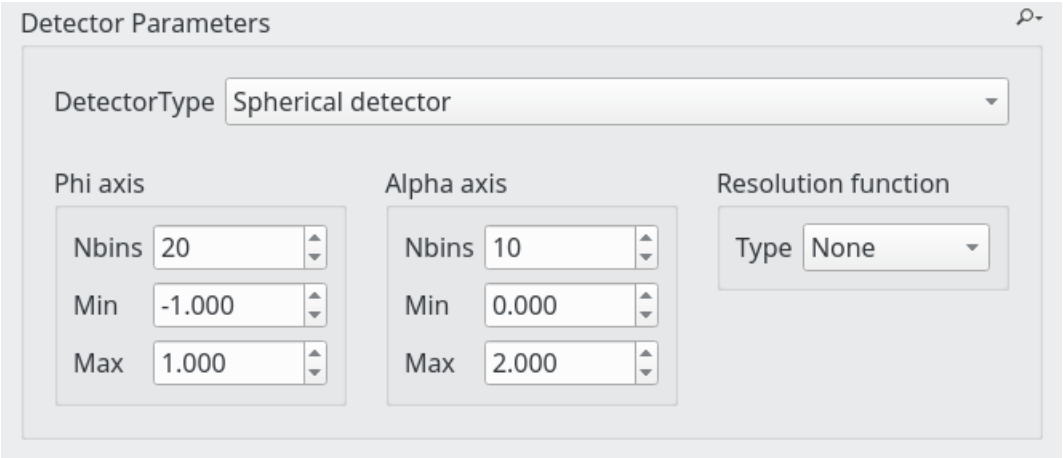

http://bornagainproject.org/documentation/usage/scripting/spherical_detector_tutorial
http://bornagainproject.org/documentation/usage/scripting/rectangular_detector_tutorial

# Spherical detector

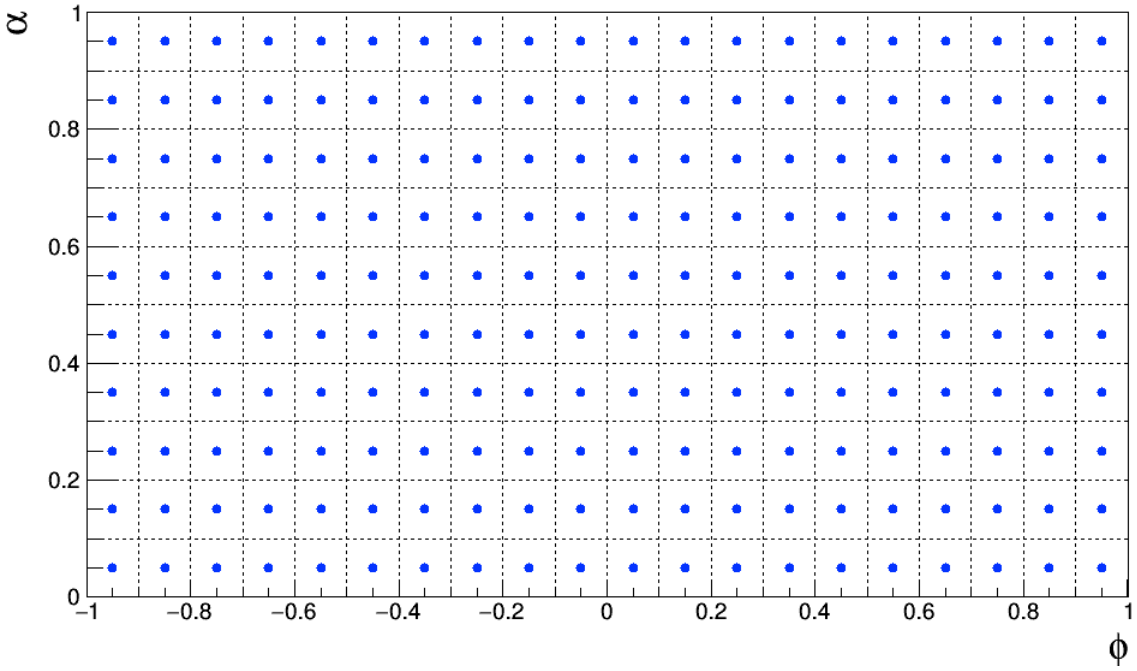



Spherical detector represents portion of a sphere, whose center is located at the origin of the sample coordinate system.

# Spherical detector




GUI angles are in degrees.

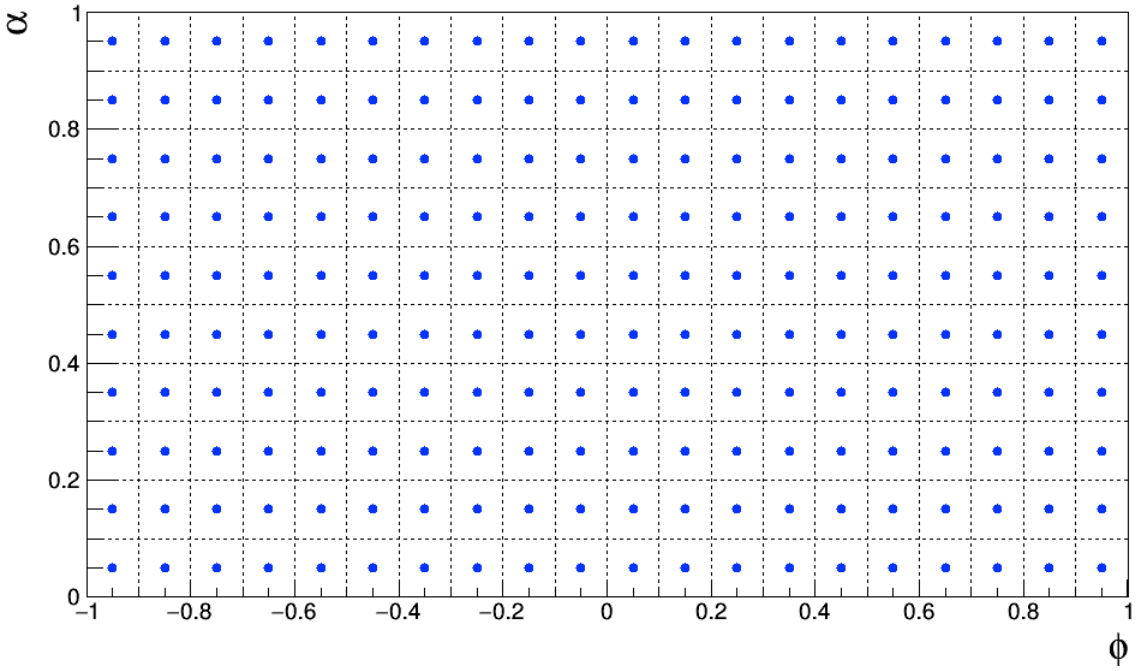


# Spherical detector

n_phi, phi_min, phi_max, n_alpha, alpha_min, alpha_max

```
simulation = GISASSimulation()

# way 1
simulation.setDetectorParameters(20, -1.0*deg, 1.0*deg, 10, 0.0*deg, 1.0*deg)

# way 2
detector = SphericalDetector(20, -1.0*deg, 1.0*deg, 10, 0.0*deg, 1.0*deg)
simulation.setDetector(detector)
```

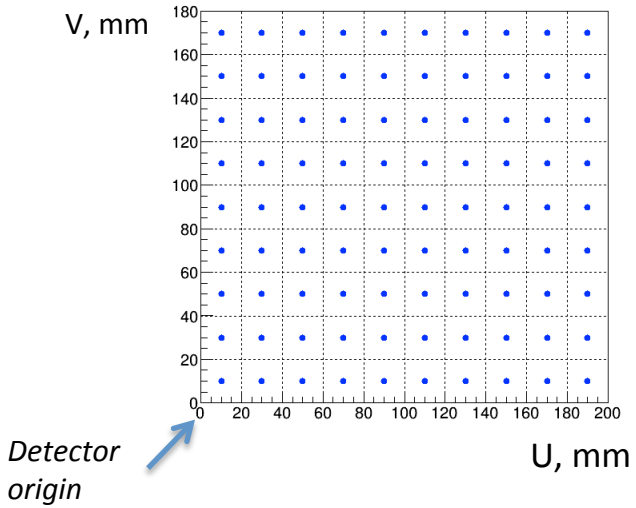
deg = π/180.0

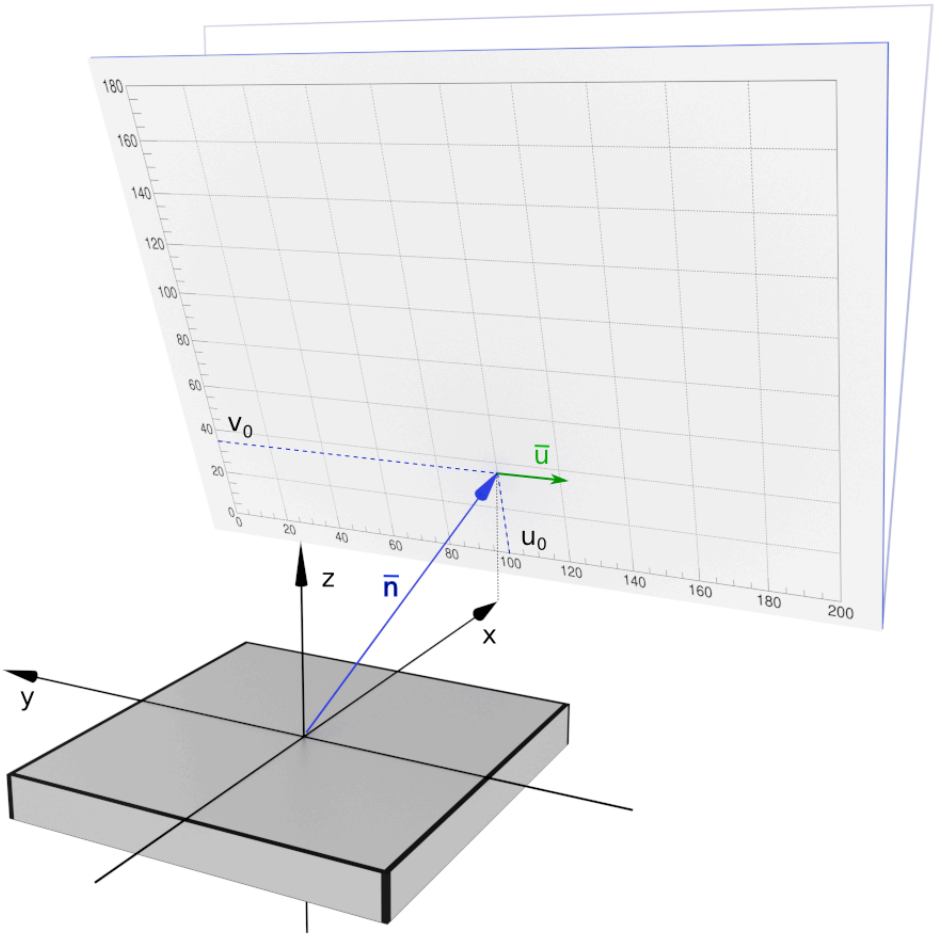


# Rectangular detector

## Detector plane:

- Number of bins in horizontal direction, total width in mm
- Number of bins in vertical direction, total height in mm




## Detector orientation:

- Normal **n** to the detector plane in sample coordinate system
- ($u_0$, $v_0$) of intersection of **n** and the detector plane in detector coordinates
- **u**-rotation vector

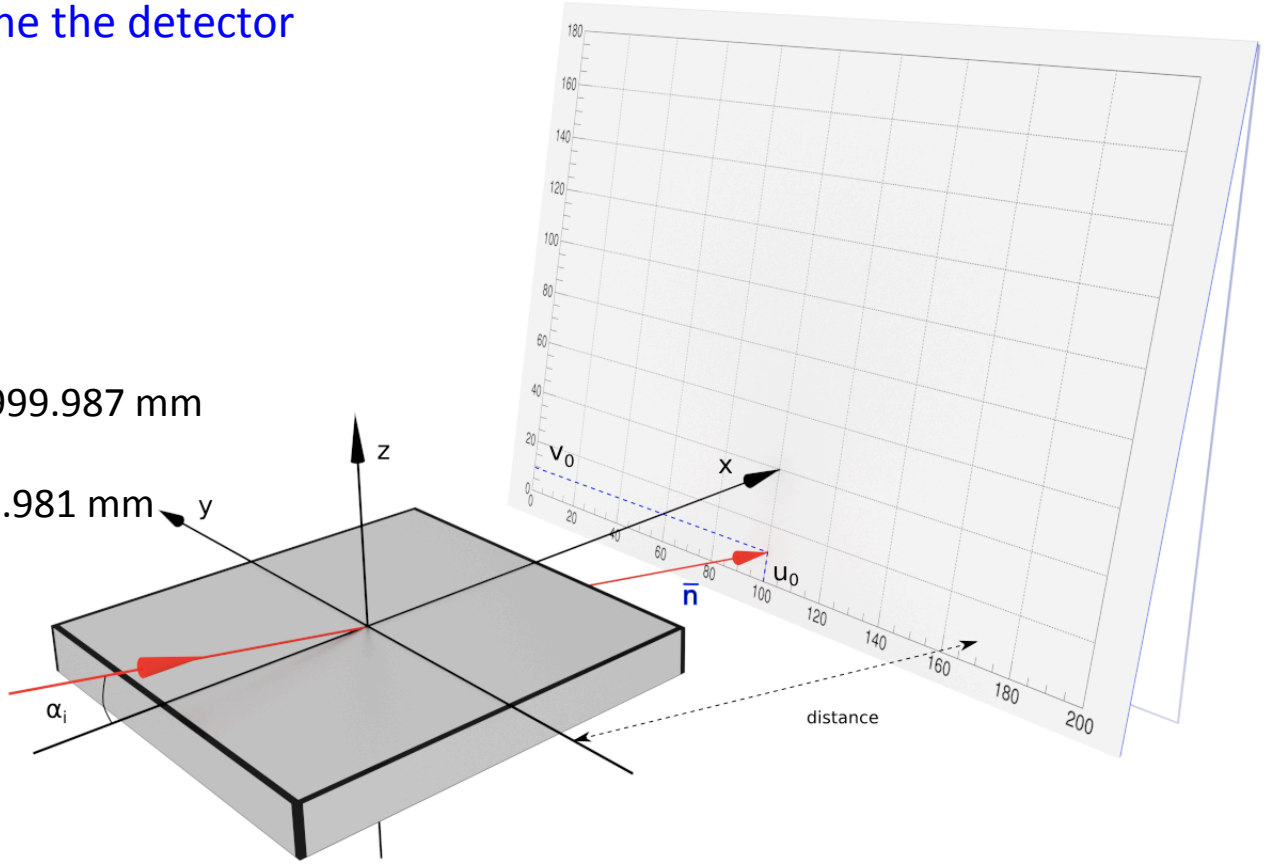
# Rectangular detector

## Example: detector is perpendicular to direct beam

- $\alpha_i$ is the beam inclination angle
- Normal **n** coincide with the beam direction
- Length of **n** is equal to sample-detector distance
- $(u_0, v_0)$ denote the point where direct beam hit the detector

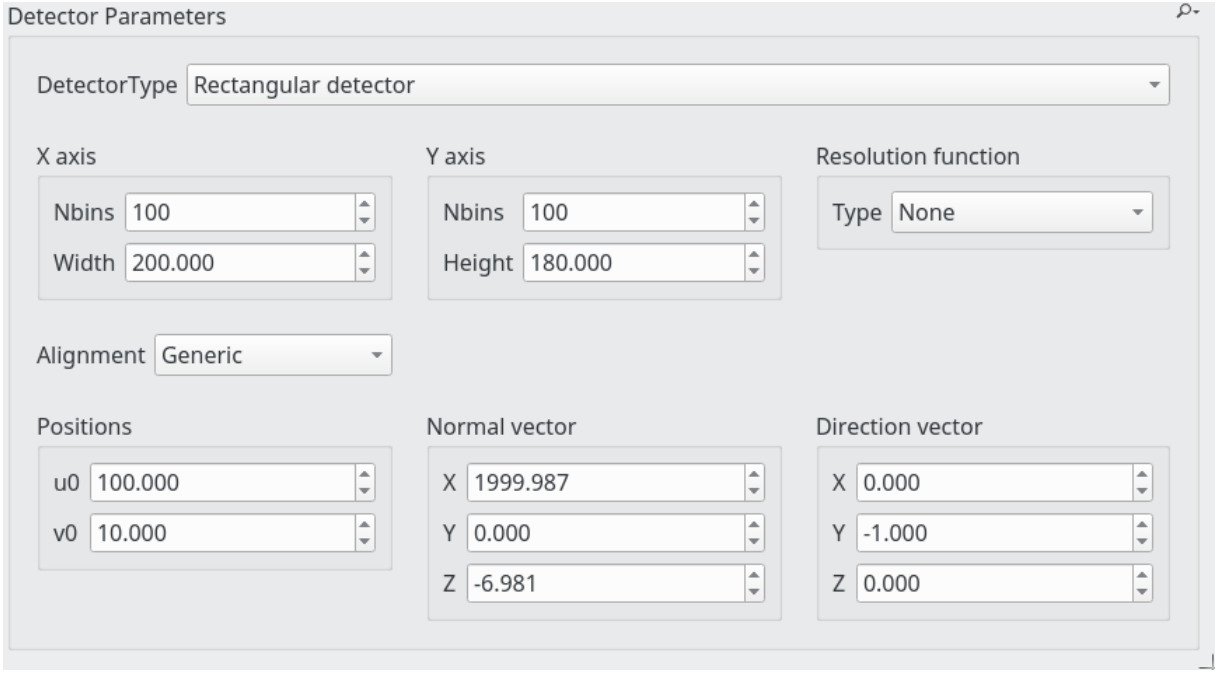
## Numbers necessary to define the detector

nxbins = 100
width = 200 mm
nybins = 100
height = 180 mm
$\alpha_i$ = 0.2 degree
distance = 2000 mm
$n_x$ = distance*cos(alpha_i) = 1999.987 mm
$n_y$ = 0
$n_z$ = -distance*sin(alpha_i) = -6.981 mm
$u_0$ = 100 mm
$v_0$ = 10 mm



# Rectangular detector

## Example: detector is perpendicular to direct beam



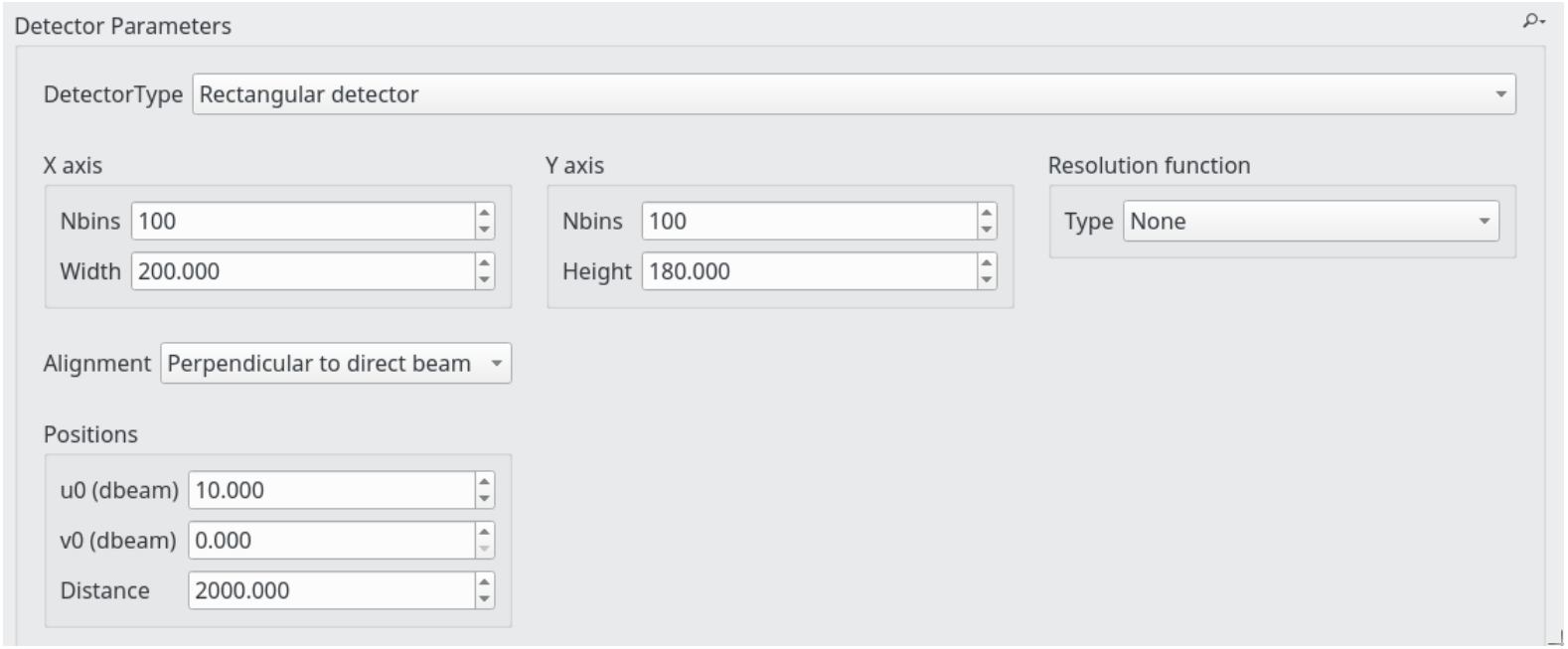

In GUI

```
width, height = 200.0, 180.0
distance = 2000.0
u0, v0 = 100.0, 10.0
alpha_i = 0.2*degree

detector = RectangularDetector(100, width, 100, height)
normal = kvector_t(distance*cos(alpha_i), 0.0, -1.0*distance*sin(alpha_i))
detector.setPosition(normal, u0, v0)

simulation = GISASSimulation()
simulation.setDetector(detector)
```

In Python

# Rectangular detector

## Example: detector is perpendicular to direct beam

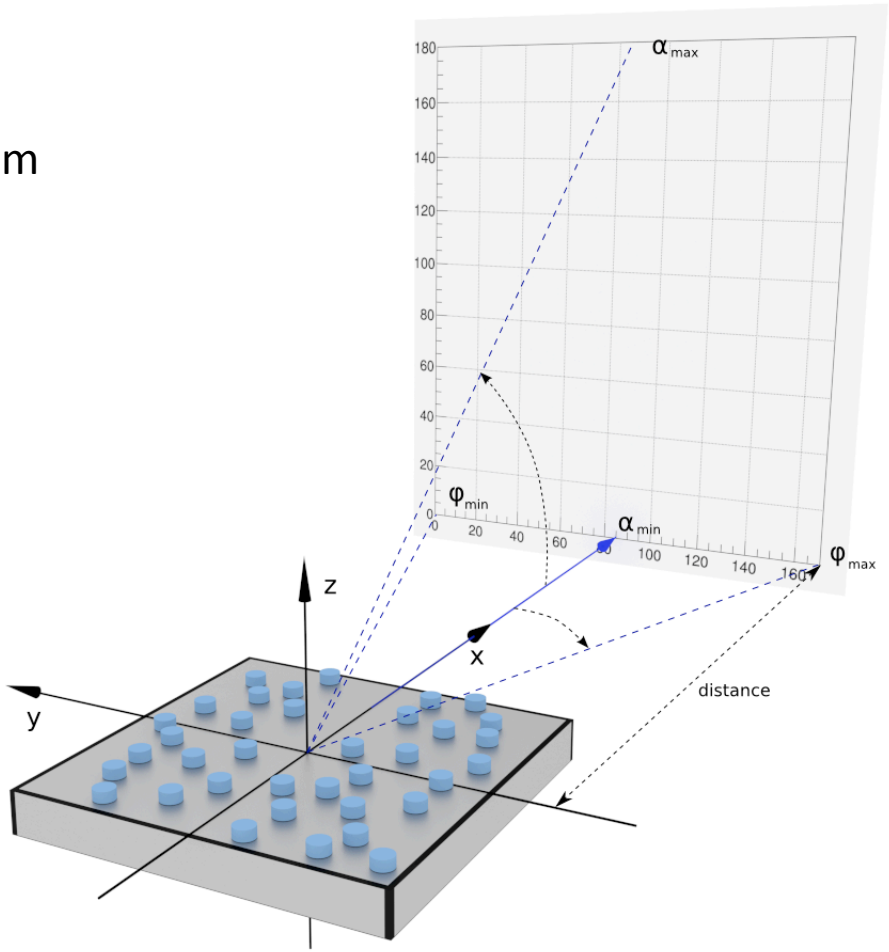



```
width, height = 200.0, 180.0
distance = 2000.0
u0, v0 = 100.0, 10.0

detector = RectangularDetector(100, width, 100, height)
detector.setPerpendicularToDirectBeam(distance, u0, v0)

simulation = GISASSimulation()
simulation.setDetector(detector)
```

# Which detector to choose: spherical or rectangular?


Spherical to give a quick try, rectangular to describe instrument as good as possible.
See example

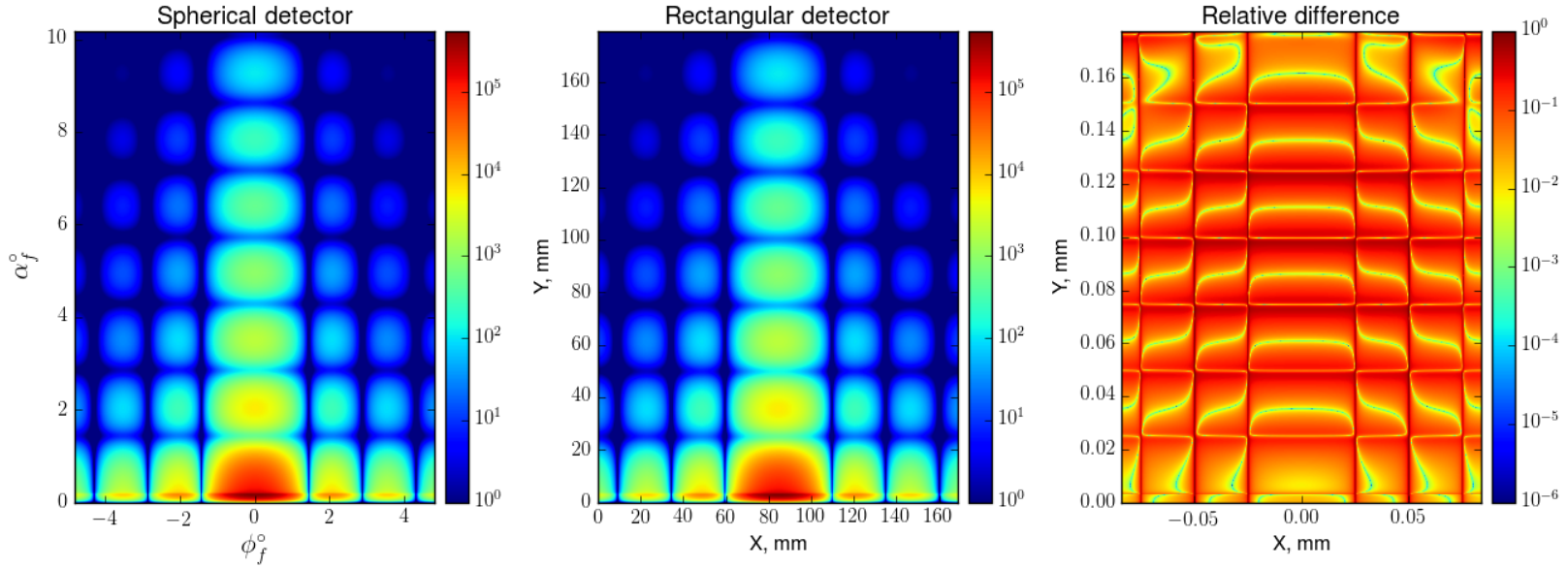
- Pilatus detector
  981x1043 pixels
- Distance 1000 mm



# Which detector to choose: spherical or rectangular?


Spherical to give a quick try, rectangular to describe instrument as good as possible.
See example

https://github.com/scgmlz/BornAgain-tutorial/blob/master/bornagain-python/fitting/advanced-tutorial/step4_rectangular_detector.py




*Example demonstrates that with increase of the detector_distance the difference in detector bin amplitudes becomes smaller.*

How to import experimental data

https://github.com/scgmlz/BornAgain-tutorial/blob/master/bornagain-python/fitting/advanced-tutorial/step5_import_data.py

What is SampleBuilder and how to fit parameters which are not there.

https://github.com/scgmlz/BornAgain-tutorial/blob/master/bornagain-python/fitting/advanced-tutorial/step6_sample_builder.py

# Advanced fitting game: nanocomposite