SasView Documentation

Release 4.2.1

The SasView Project

Feb 10, 2019

CONTENTS

CHAPTER

ONE

SASVIEW USER DOCUMENTATION

Note: In Windows use [Alt]-[Cursor left] to return to the previous page

1.1 Model Functions

1.1.1 Cylinder Functions

barbell

Cylinder with spherical end caps

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
sld	Barbell scattering length density	10^{-6}Å^{-2}	4
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	1
radius_bell	Spherical bell radius	Å	40
radius	Cylindrical bar radius	Å	20
length	Cylinder bar length	Å	400
theta	Barbell axis to beam angle	degree	60
phi	Rotation about beam	degree	60

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

Calculates the scattering from a barbell-shaped cylinder. Like *capped_cylinder*, this is a sphereocylinder with spherical end caps that have a radius larger than that of the cylinder, but with the center of the end cap radius lying outside of the cylinder. See the diagram for the details of the geometry and restrictions on parameter values.

The scattered intensity $I(q)$ is calculated as

$$I(q) = \frac{\Delta\rho^2}{V} \left\langle A^2(q,\alpha).sin(\alpha)\right\rangle$$

where the amplitude $A(q,\alpha)$ with the rod axis at angle α to q is given as

$$\begin{aligned} A(q) =&\ \pi r^2L \frac{\sin\left(\tfrac12 qL\cos\alpha\right)}{\tfrac12 qL\cos\alpha} \frac{2J_1(qr\sin\alpha)}{qr\sin\alpha} \\ &+ 4\pi R^3\int_{-h/R}^1 dt \cos\left[q\cos\alpha \left(Rt + h + {\tfrac12} L\right)\right] \times (1-t^2) \frac{J_1\left[qR\sin\alpha \left(1-t^2\right)^{1/2}\right]}{qR\sin\alpha \left(1-t^2\right)^{1/2}} \end{aligned}$$

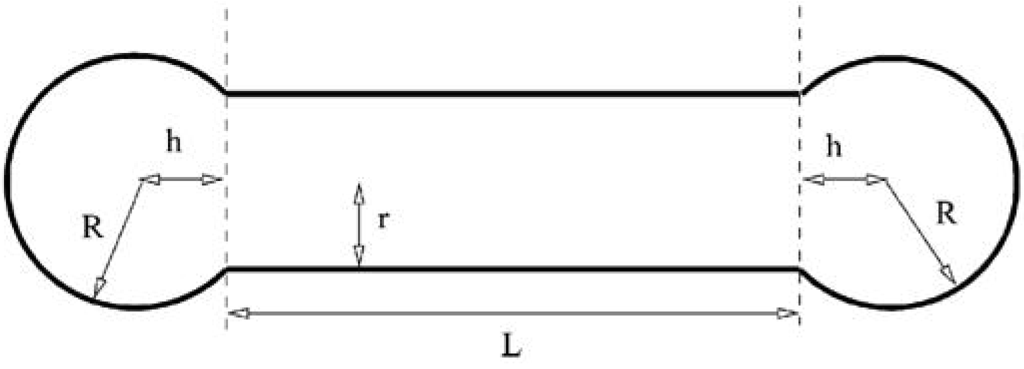


Fig. 1.1: Barbell geometry, where r is *radius*, R is *radius_bell* and L is *length*. Since the end cap radius $R \geq r$ and by definition for this geometry $h < 0$, h is then defined by r and R as $h = -\sqrt{R^2 - r^2}$

The $\langle \ldots \rangle$ brackets denote an average of the structure over all orientations. $\langle A^2(q,\alpha) \rangle$ is then the form factor, $P(q)$. The scale factor is equivalent to the volume fraction of cylinders, each of volume, V. Contrast $\Delta\rho$ is the difference of scattering length densities of the cylinder and the surrounding solvent.

The volume of the barbell is

$$V = \pi r_c^2 L + 2\pi \left(\tfrac{2}{3}R^3 + R^2 h - \tfrac{1}{3}h^3\right)$$

and its radius of gyration is

$$\begin{aligned} R_g^2 = &\left[\tfrac{12}{5}R^5 + R^4\left(6h + \tfrac{3}{2}L\right) + R^2\left(4h^2 + L^2 + 4Lh\right) + R^2\left(3Lh^2 + \tfrac{3}{2}L^2 h\right)\right. \\ &\left. + \tfrac{2}{5}h^5 - \tfrac{1}{2}Lh^4 - \tfrac{1}{2}L^2h^3 + \tfrac{1}{4}L^3r^2 + \tfrac{3}{2}Lr^4\right]\left(4R^3 6R^2 h - 2h^3 + 3r^2 L\right)^{-1} \end{aligned}$$

Note: The requirement that $R \geq r$ is not enforced in the model! It is up to you to restrict this during analysis.

The 2D scattering intensity is calculated similar to the 2D cylinder model.

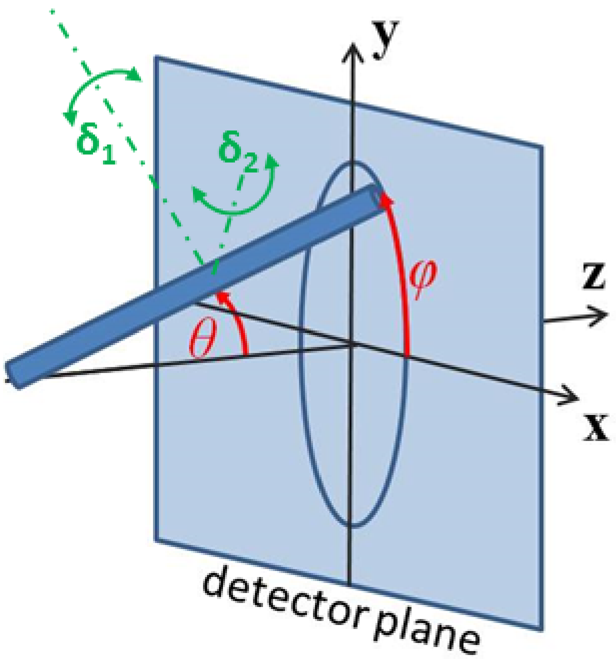


Fig. 1.2: Definition of the angles for oriented 2D barbells.

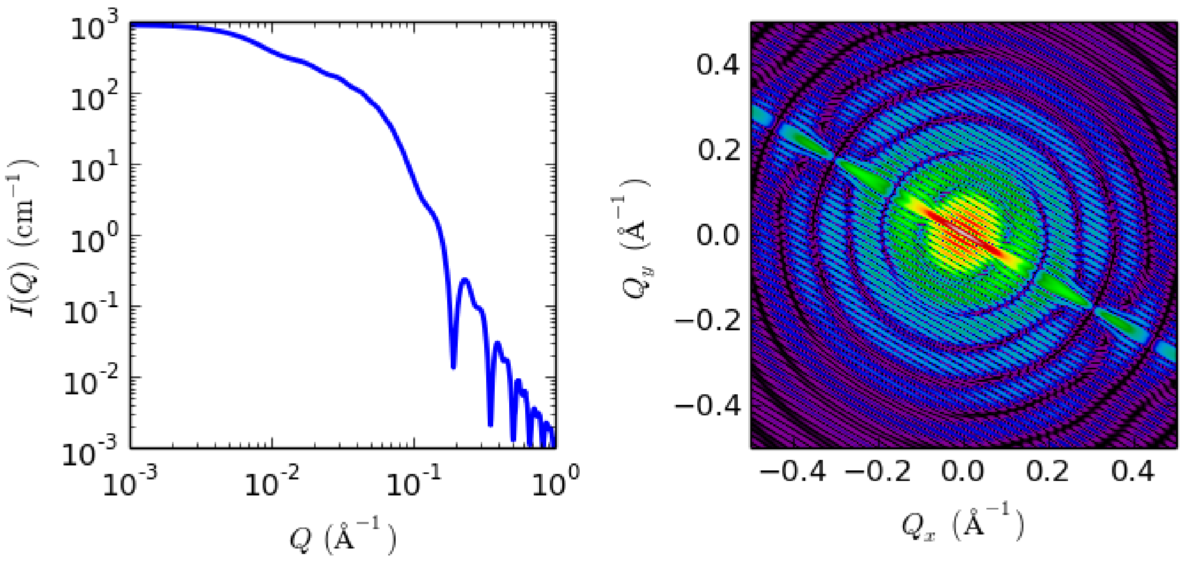


Fig. 1.3: 1D and 2D plots corresponding to the default parameters of the model.

References

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Last Modified by:** Paul Butler **Date:** March 20, 2016
- **Last Reviewed by:** Richard Heenan **Date:** January 4, 2017

capped_cylinder

Right circular cylinder with spherical end caps and uniform SLD

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
sld	Cylinder scattering length density	10^{-6}Å^{-2}	4
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	1
radius	Cylinder radius	Å	20
radius_cap	Cap radius	Å	20
length	Cylinder length	Å	400
theta	cylinder axis to beam angle	degree	60
phi	rotation about beam	degree	60

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definitions

Calculates the scattering from a cylinder with spherical section end-caps. Like *barbell*, this is a sphereocylinder with end caps that have a radius larger than that of the cylinder, but with the center of the end cap radius lying within the cylinder. This model simply becomes a convex lens when the length of the cylinder $L = 0$. See the diagram for the details of the geometry and restrictions on parameter values.

The scattered intensity $I(q)$ is calculated as

$$I(q) = \frac{\Delta\rho^2}{V} \left\langle A^2(q,\alpha).sin(\alpha)\right\rangle$$

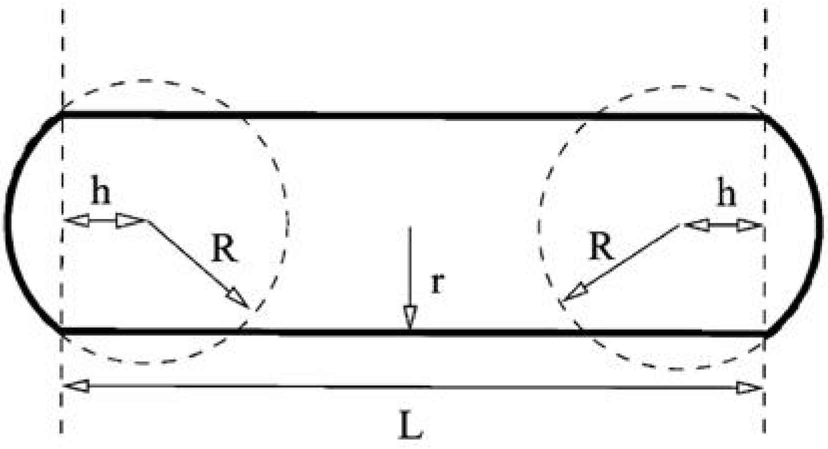


Fig. 1.4: Capped cylinder geometry, where r is *radius*, R is *bell_radius* and L is *length*. Since the end cap radius $R \geq r$ and by definition for this geometry $h < 0$, h is then defined by r and R as $h = -\sqrt{R^2 - r^2}$

where the amplitude $A(q, \alpha)$ with the rod axis at angle α to q is given as

$$\begin{aligned} A(q) =& \pi r^2 L \frac{\sin\left(\frac{1}{2}qL\cos\alpha\right)}{\frac{1}{2}qL\cos\alpha}\frac{2J_1(qr\sin\alpha)}{qr\sin\alpha} \\ &+ 4\pi R^3 \int_{-h/R}^{1} dt \cos\left[q\cos\alpha\left(Rt + h + \tfrac{1}{2}L\right)\right] \times (1-t^2) \frac{J_1\left[qR\sin\alpha\left(1-t^2\right)^{1/2}\right]}{qR\sin\alpha\,(1-t^2)^{1/2}} \end{aligned}$$

The $\langle \ldots \rangle$ brackets denote an average of the structure over all orientations. $\langle A^2(q)\rangle$ is then the form factor, $P(q)$. The scale factor is equivalent to the volume fraction of cylinders, each of volume, V. Contrast $\Delta\rho$ is the difference of scattering length densities of the cylinder and the surrounding solvent.

The volume of the capped cylinder is (with h as a positive value here)

$$V = \pi r_c^2 L + \tfrac{2\pi}{3}(R-h)^2(2R+h)$$

and its radius of gyration is

$$\begin{aligned} R_g^2 =& \left[\tfrac{12}{5}R^5 + R^4\left(6h + \tfrac{3}{2}L\right) + R^2\left(4h^2 + L^2 + 4Lh\right) + R^2\left(3Lh^2 + \tfrac{3}{2}L^2 h\right)\right. \\ &\left. + \tfrac{2}{5}h^5 - \tfrac{1}{2}Lh^4 - \tfrac{1}{2}L^2h^3 + \tfrac{1}{4}L^3r^2 + \tfrac{3}{2}Lr^4\right]\left(4R^3 6R^2 h - 2h^3 + 3r^2 L\right)^{-1} \end{aligned}$$

Note: The requirement that $R \geq r$ is not enforced in the model! It is up to you to restrict this during analysis.

The 2D scattering intensity is calculated similar to the 2D cylinder model.

References

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Last Modified by:** Paul Butler **Date:** September 30, 2016
- **Last Reviewed by:** Richard Heenan **Date:** January 4, 2017

core_shell_bicelle

Circular cylinder with a core-shell scattering length density profile..

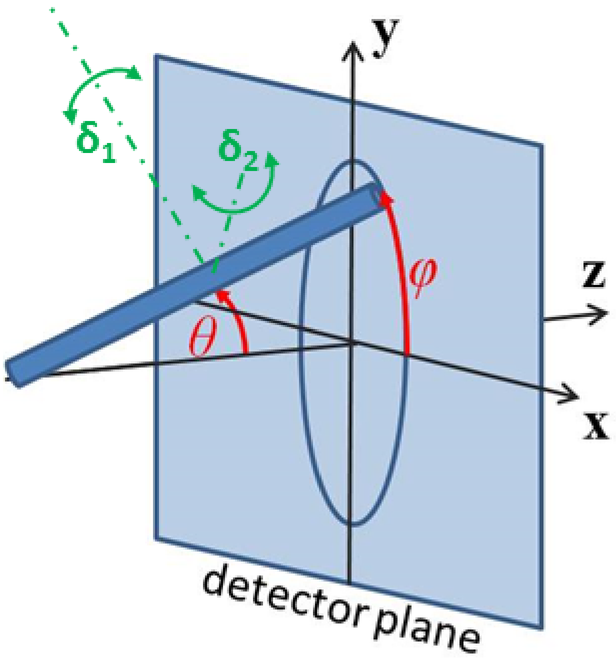


Fig. 1.5: Definition of the angles for oriented 2D cylinders.

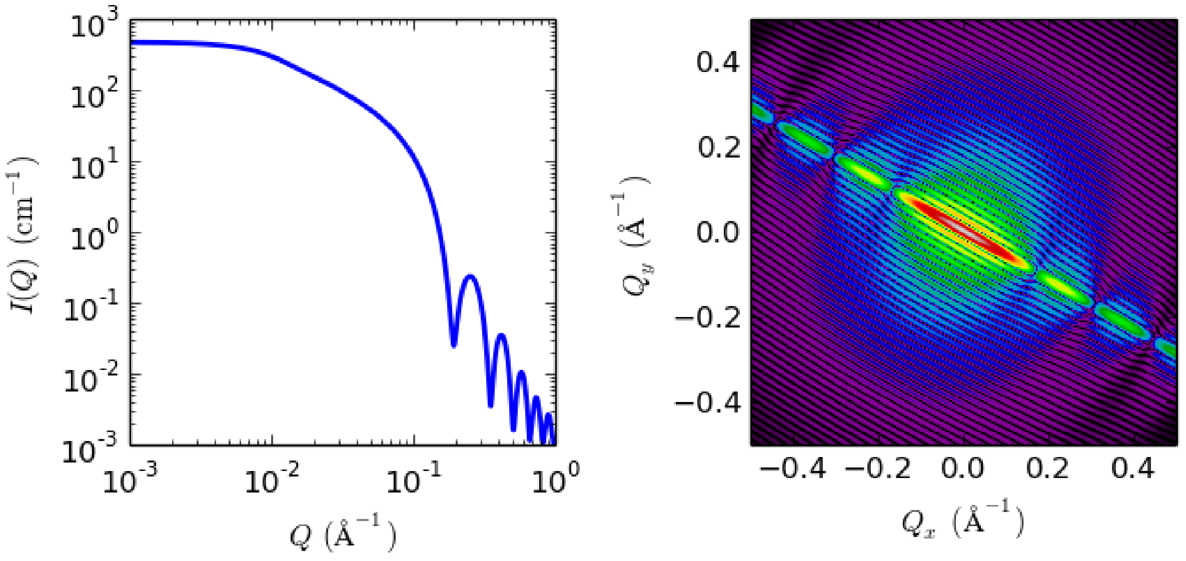


Fig. 1.6: 1D and 2D plots corresponding to the default parameters of the model.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
radius	Cylinder core radius	Å	80
thick_rim	Rim shell thickness	Å	10
thick_face	Cylinder face thickness	Å	10
length	Cylinder length	Å	50
sld_core	Cylinder core scattering length density	$10^{-6}Å^{-2}$	1
sld_face	Cylinder face scattering length density	$10^{-6}Å^{-2}$	4
sld_rim	Cylinder rim scattering length density	$10^{-6}Å^{-2}$	4
sld_solvent	Solvent scattering length density	$10^{-6}Å^{-2}$	1
theta	cylinder axis to beam angle	degree	90
phi	rotation about beam	degree	0

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

This model provides the form factor for a circular cylinder with a core-shell scattering length density profile. Thus this is a variation of a core-shell cylinder or disc where the shell on the walls and ends may be of different thicknesses and scattering length densities. The form factor is normalized by the particle volume.

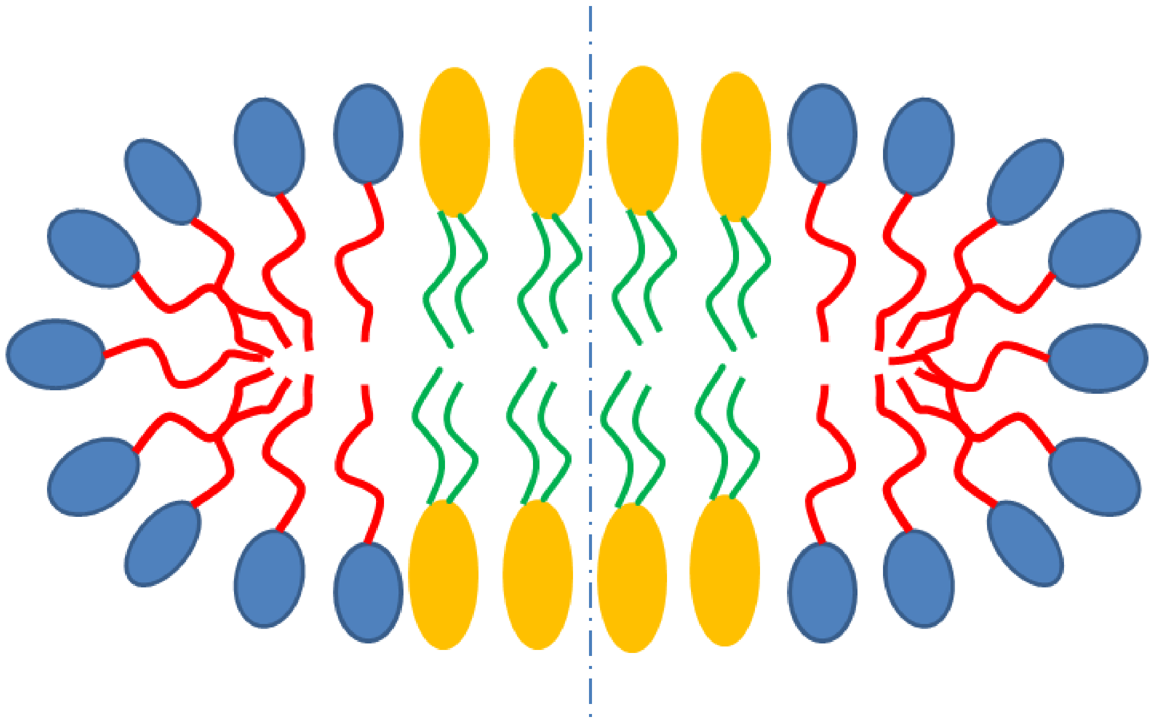


Fig. 1.7: Schematic cross-section of bicelle. Note however that the model here calculates for rectangular, not curved, rims as shown below.

Given the scattering length densities (sld) ρ_c, the core sld, ρ_f, the face sld, ρ_r, the rim sld and ρ_s the solvent sld, the scattering length density variation along the cylinder axis is:

$$\rho(r) = \begin{cases} \rho_c \text{ for } 0 < r < R; -L < z < L \\ \rho_f \text{ for } 0 < r < R; -(L+2t) < z < -L; L < z < (L+2t) \\ \rho_r \text{ for } 0 < r < R; -(L+2t) < z < -L; L < z < (L+2t) \end{cases}$$

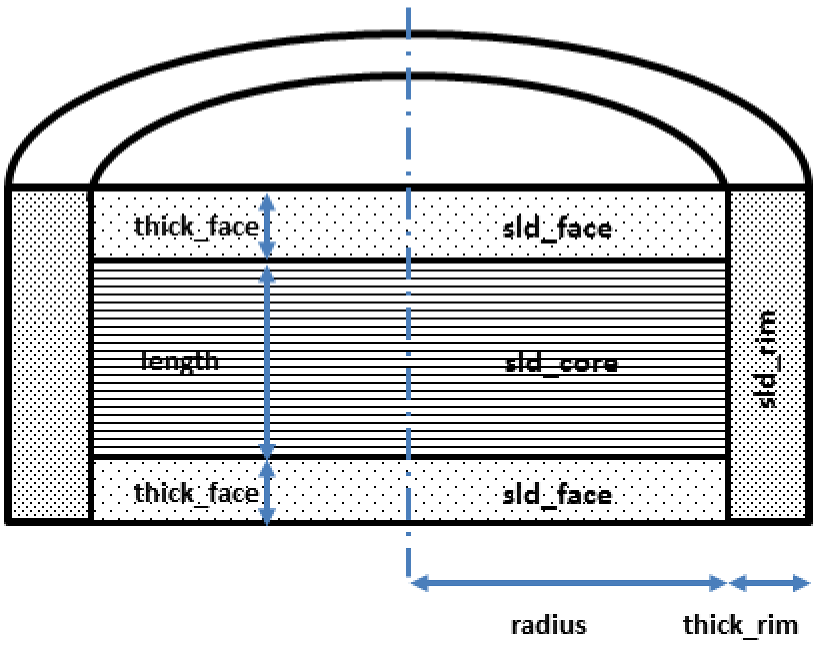


Fig. 1.8: Cross section of cylindrical symmetry model used here. Users will have to decide how to distribute "heads" and "tails" between the rim, face and core regions in order to estimate appropriate starting parameters.

The form factor for the bicelle is calculated in cylindrical coordinates, where α is the angle between the Q vector and the cylinder axis, to give:

$$I(Q,\alpha) = \frac{\text{scale}}{V_t} \cdot F(Q,\alpha)^2 \cdot sin(\alpha) + \text{background}$$

where

$$\begin{aligned} F(Q,\alpha) = \Bigg[& (\rho_c - \rho_f) V_c \frac{2J_1(QRsin\alpha)}{QRsin\alpha} \frac{sin(QLcos\alpha/2)}{Q(L/2)cos\alpha} \\ & + (\rho_f - \rho_r) V_{c+f} \frac{2J_1(QRsin\alpha)}{QRsin\alpha} \frac{sin(Q(L/2+t_f)cos\alpha)}{Q(L/2+t_f)cos\alpha} \\ & + (\rho_r - \rho_s) V_t \frac{2J_1(Q(R+t_r)sin\alpha)}{Q(R+t_r)sin\alpha} \frac{sin(Q(L/2+t_f)cos\alpha)}{Q(L/2+t_f)cos\alpha} \Bigg] \end{aligned}$$

where V_t is the total volume of the bicelle, V_c the volume of the core, V_{c+f} the volume of the core plus the volume of the faces, R is the radius of the core, L the length of the core, t_f the thickness of the face, t_r the thickness of the rim and J_1 the usual first order bessel function.

The output of the 1D scattering intensity function for randomly oriented cylinders is then given by integrating over all possible θ and ϕ.

For oriented bicelles the *theta*, and *phi* orientation parameters will appear when fitting 2D data, see the *cylinder* model for further information. Our implementation of the scattering kernel and the 1D scattering intensity use the c-library from NIST.

References

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Last Modified by:** Paul Butler **Date:** September 30, 2016
- **Last Reviewed by:** Richard Heenan **Date:** January 4, 2017

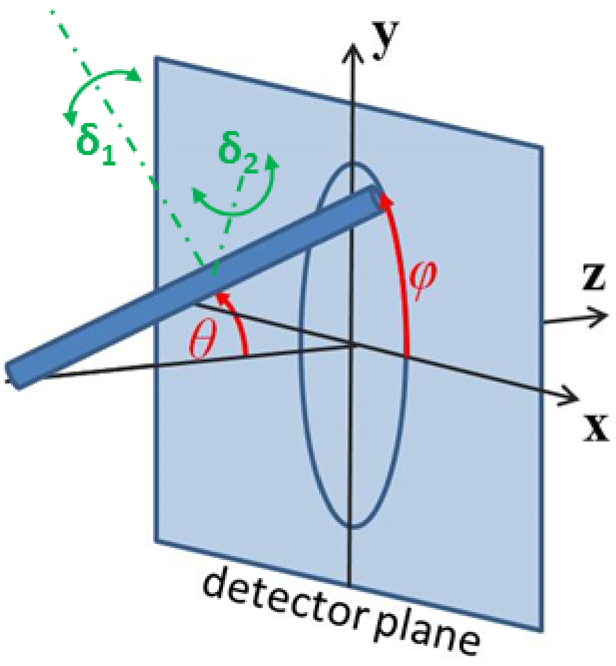


Fig. 1.9: Definition of the angles for the oriented core shell bicelle model, note that the cylinder axis of the bicelle starts along the beam direction when $\theta = \phi = 0$.

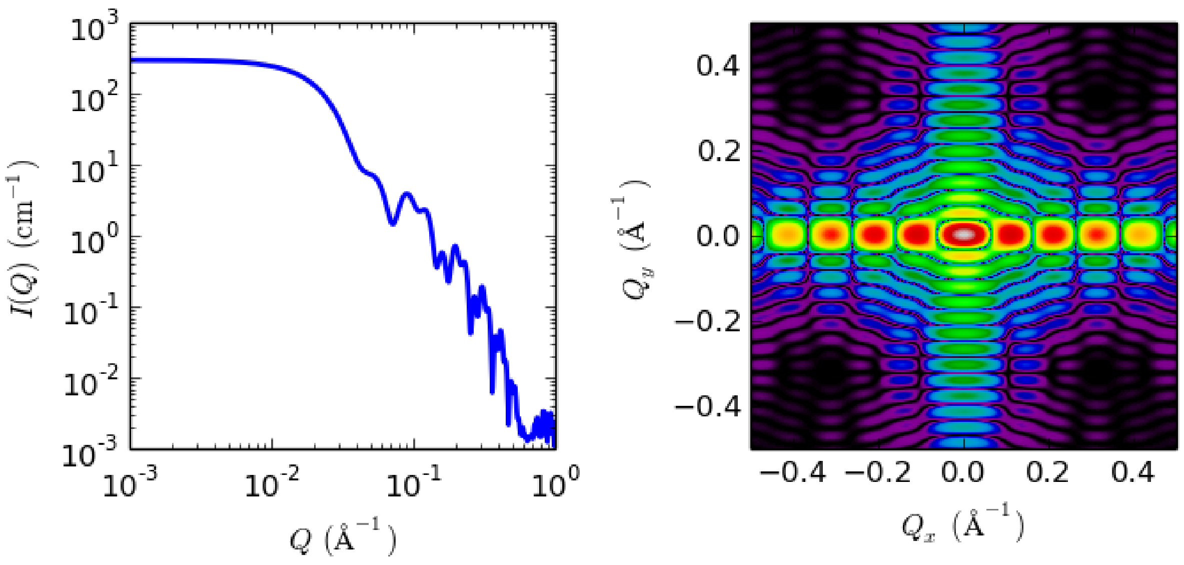


Fig. 1.10: 1D and 2D plots corresponding to the default parameters of the model.

core_shell_bicelle_elliptical

Elliptical cylinder with a core-shell scattering length density profile..

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
radius	Cylinder core radius r_minor	Å	30
x_core	Axial ratio of core, X = r_major/r_minor	None	3
thick_rim	Rim shell thickness	Å	8
thick_face	Cylinder face thickness	Å	14
length	Cylinder length	Å	50
sld_core	Cylinder core scattering length density	$10^{-6}Å^{-2}$	4
sld_face	Cylinder face scattering length density	$10^{-6}Å^{-2}$	7
sld_rim	Cylinder rim scattering length density	$10^{-6}Å^{-2}$	1
sld_solvent	Solvent scattering length density	$10^{-6}Å^{-2}$	6
theta	Cylinder axis to beam angle	degree	90
phi	Rotation about beam	degree	0
psi	Rotation about cylinder axis	degree	0

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

This model provides the form factor for an elliptical cylinder with a core-shell scattering length density profile. Thus this is a variation of the core-shell bicelle model, but with an elliptical cylinder for the core. Outer shells on the rims and flat ends may be of different thicknesses and scattering length densities. The form factor is normalized by the total particle volume.

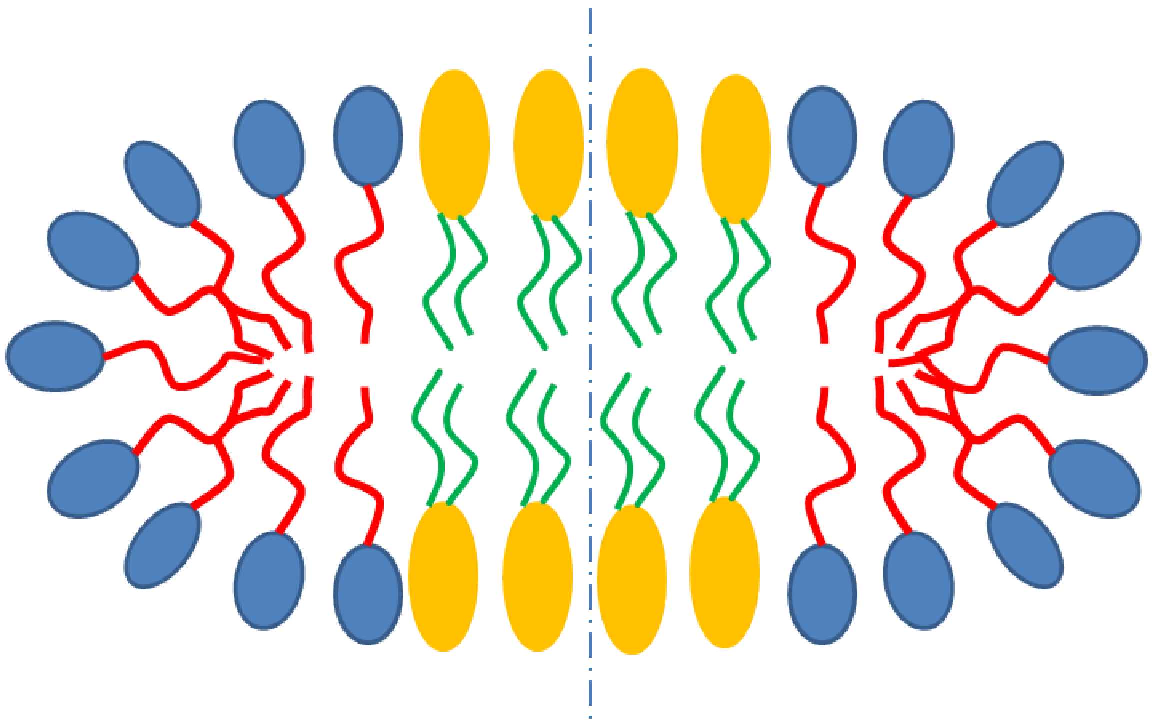


Fig. 1.11: Schematic cross-section of bicelle. Note however that the model here calculates for rectangular, not curved, rims as shown below.

Given the scattering length densities (sld) ρ_c, the core sld, ρ_f, the face sld, ρ_r, the rim sld and ρ_s the solvent sld,

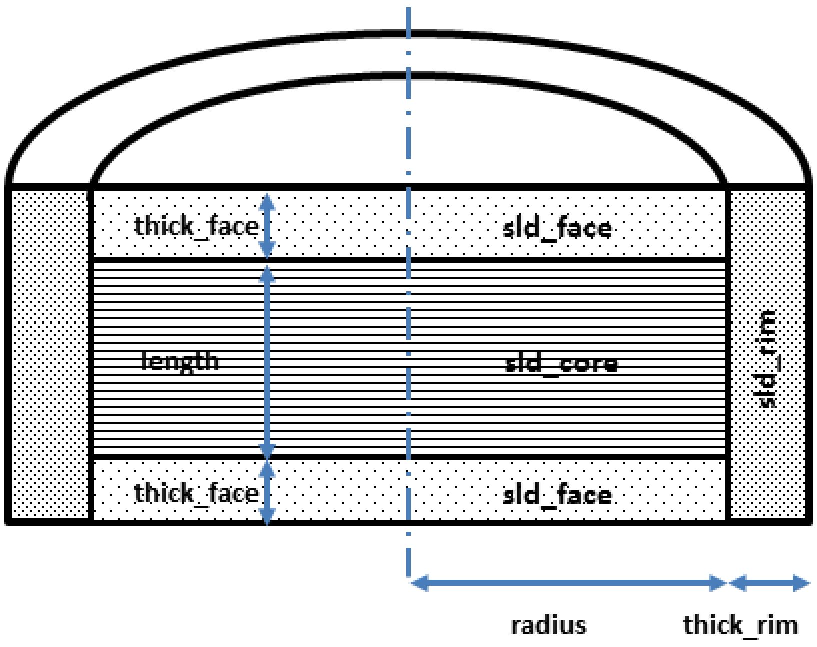


Fig. 1.12: Cross section of model used here. Users will have to decide how to distribute "heads" and "tails" between the rim, face and core regions in order to estimate appropriate starting parameters.

the scattering length density variation along the bicelle axis is:

$$\rho(r) = \begin{cases} \rho_c \text{ for } 0 < r < R; -L < z < L \\ \rho_f \text{ for } 0 < r < R; -(L+2t) < z < -L; L < z < (L+2t) \\ \rho_r \text{ for } 0 < r < R; -(L+2t) < z < -L; L < z < (L+2t) \end{cases}$$

The form factor for the bicelle is calculated in cylindrical coordinates, where α is the angle between the Q vector and the cylinder axis, and ψ is the angle for the ellipsoidal cross section core, to give:

$$I(Q,\alpha,\psi) = \frac{\text{scale}}{V_t} \cdot F(Q,\alpha,\psi)^2 \cdot sin(\alpha) + \text{background}$$

where a numerical integration of $F(Q,\alpha,\psi)^2 \cdot sin(\alpha)$ is carried out over alpha and psi for:

$$\begin{aligned} F(Q,\alpha,\psi) = \bigg[& (\rho_c - \rho_f)V_c \frac{2J_1(QR'sin\alpha)}{QR'sin\alpha}\frac{sin(QLcos\alpha/2)}{Q(L/2)cos\alpha} \\ & + (\rho_f - \rho_r)V_{c+f} \frac{2J_1(QR'sin\alpha)}{QR'sin\alpha}\frac{sin(Q(L/2+t_f)cos\alpha)}{Q(L/2+t_f)cos\alpha} \\ & + (\rho_r - \rho_s)V_t \frac{2J_1(Q(R'+t_r)sin\alpha)}{Q(R'+t_r)sin\alpha}\frac{sin(Q(L/2+t_f)cos\alpha)}{Q(L/2+t_f)cos\alpha} \bigg] \end{aligned}$$

where

$$R' = \frac{R}{\sqrt{2}}\sqrt{(1+X_{core}^2) + (1-X_{core}^2)cos(\psi)}$$

and $V_t = \pi.(R+t_r)(Xcore.R+t_r)^2.(L+2.t_f)$ is the total volume of the bicelle, $V_c = \pi.Xcore.R^2.L$ the volume of the core, $V_{c+f} = \pi.Xcore.R^2.(L+2.t_f)$ the volume of the core plus the volume of the faces, R is the radius of the core, $Xcore$ is the axial ratio of the core, L the length of the core, t_f the thickness of the face, t_r the thickness of the rim and J_1 the usual first order bessel function. The core has radii R and $Xcore.R$ so is circular, as for the core_shell_bicelle model, for $Xcore$ =1. Note that you may need to limit the range of $Xcore$, especially if using the Monte-Carlo algorithm, as setting radius to $R/Xcore$ and axial ratio to $1/Xcore$ gives an equivalent solution!

The output of the 1D scattering intensity function for randomly oriented bicelles is then given by integrating over all possible α and ψ.

For oriented bicelles the *theta*, *phi* and *psi* orientation parameters will appear when fitting 2D data, see the *elliptical_cylinder* model for further information.

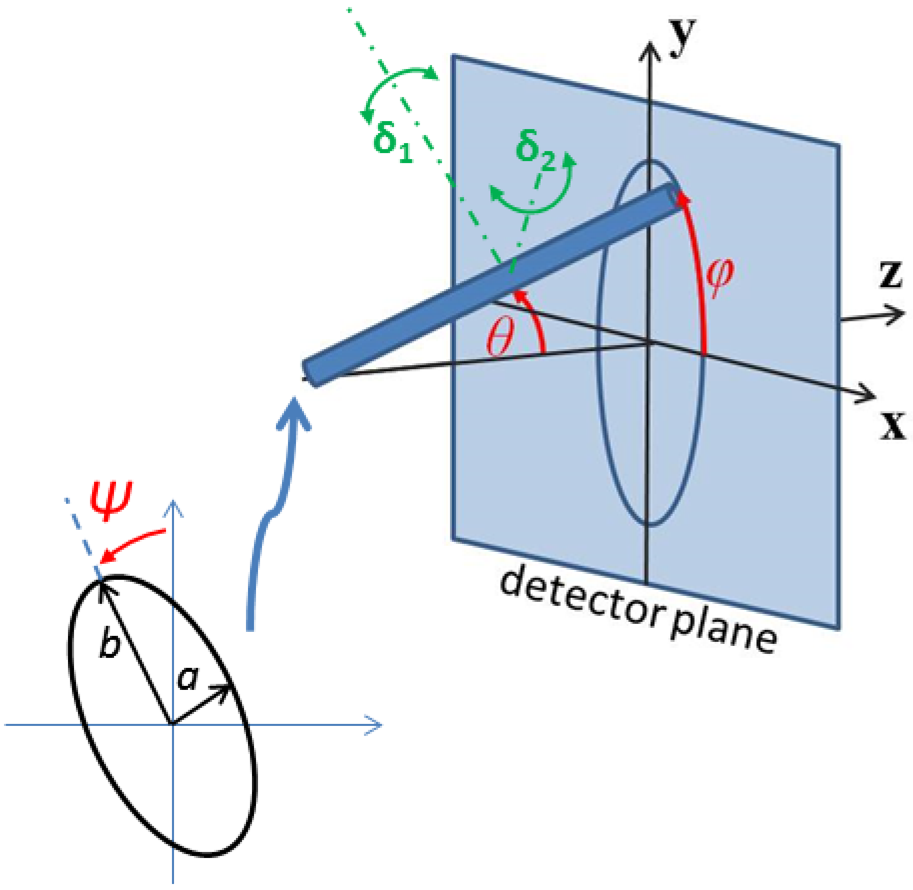


Fig. 1.13: Definition of the angles for the oriented core_shell_bicelle_elliptical particles.

Model verified using Monte Carlo simulation for 1D and 2D scattering.

References

Authorship and Verification

- **Author:** Richard Heenan **Date:** December 14, 2016
- **Last Modified by:** Richard Heenan **Date:** December 14, 2016
- **Last Reviewed by:** Paul Kienzle **Date:** Feb 28, 2018

core_shell_bicelle_elliptical_belt_rough

Elliptical cylinder with a core-shell scattering length density profile..

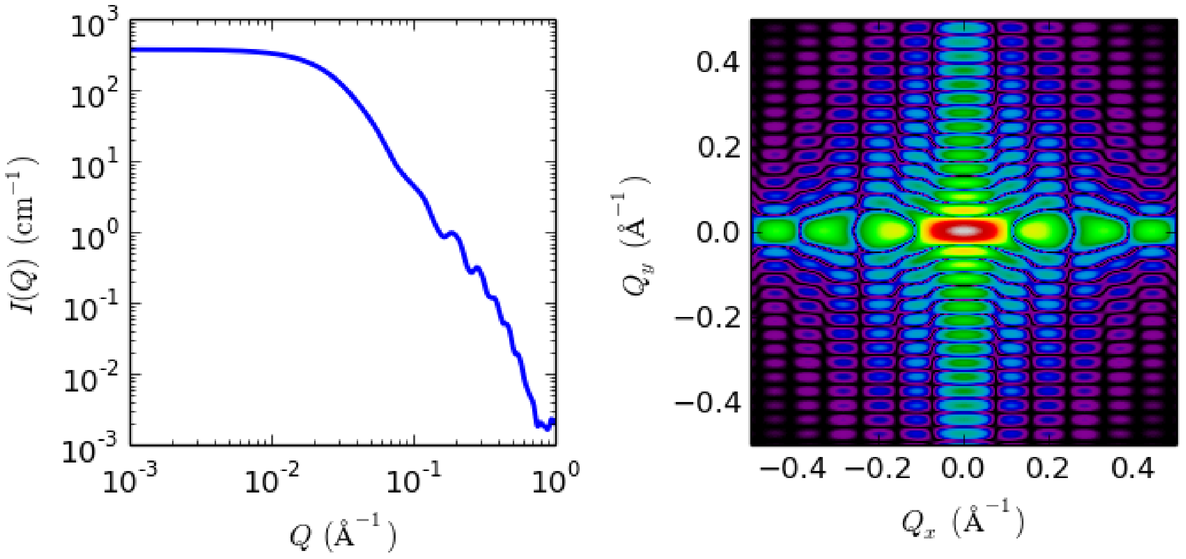


Fig. 1.14: 1D and 2D plots corresponding to the default parameters of the model.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
radius	Cylinder core radius r_minor	Å	30
x_core	Axial ratio of core, X = r_major/r_minor	None	3
thick_rim	Rim or belt shell thickness	Å	8
thick_face	Cylinder face thickness	Å	14
length	Cylinder length	Å	50
sld_core	Cylinder core scattering length density	10^{-6}Å^{-2}	4
sld_face	Cylinder face scattering length density	10^{-6}Å^{-2}	7
sld_rim	Cylinder rim scattering length density	10^{-6}Å^{-2}	1
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	6
sigma	Interfacial roughness	Å	0
theta	Cylinder axis to beam angle	degree	90
phi	Rotation about beam	degree	0
psi	Rotation about cylinder axis	degree	0

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

This model provides the form factor for an elliptical cylinder with a core-shell scattering length density profile. Thus this is a variation of the core-shell bicelle model, but with an elliptical cylinder for the core. In this version the "rim" or "belt" does NOT extend the full length of the particle, but has the same length as the core. Outer shells on the rims and flat ends may be of different thicknesses and scattering length densities. The form factor is normalized by the total particle volume. This version includes an approximate "interfacial roughness".

Given the scattering length densities (sld) ρ_c, the core sld, ρ_f, the face sld, ρ_r, the rim sld and ρ_s the solvent sld, the scattering length density variation along the bicelle axis is:

$$\rho(r) = \begin{cases} \rho_c \text{ for } 0 < r < R; -L/2 < z < L/2 \\ \rho_f \text{ for } 0 < r < R; -(L/2 + t_\text{face}) < z < -L/2; L/2 < z < (L/2 + t_\text{face}) \\ \rho_r \text{ for } R < r < R + t_\text{rim}; -L/2 < z < L/2 \end{cases}$$

The form factor for the bicelle is calculated in cylindrical coordinates, where α is the angle between the Q vector

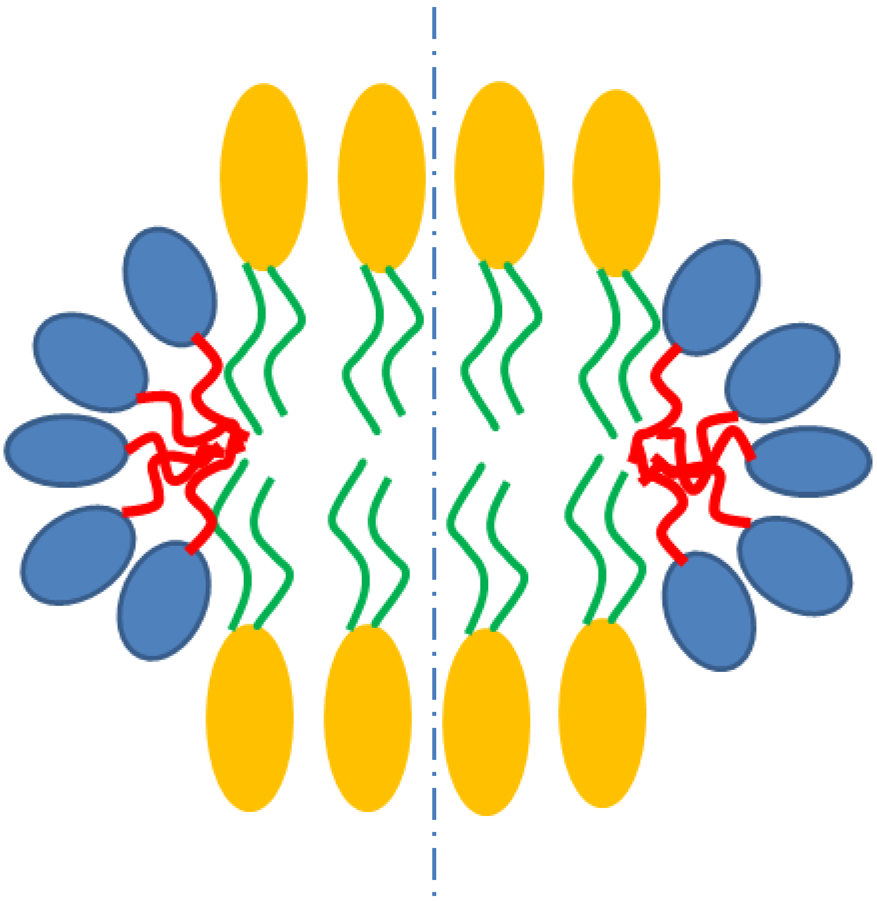


Fig. 1.15: Schematic cross-section of bicelle with belt. Note however that the model here calculates for rectangular, not curved, rims as shown below.

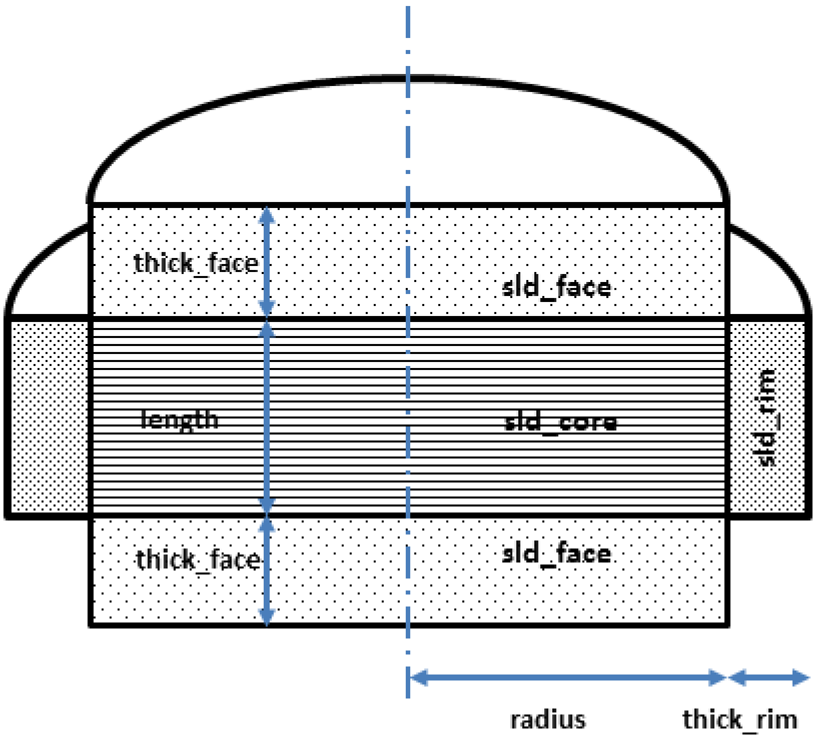


Fig. 1.16: Cross section of model used here. Users will have to decide how to distribute "heads" and "tails" between the rim, face and core regions in order to estimate appropriate starting parameters.

and the cylinder axis, and ψ is the angle for the ellipsoidal cross section core, to give:

$$I(Q, \alpha, \psi) = \frac{\text{scale}}{V_t} \cdot F(Q, \alpha, \psi)^2 \cdot \sin(\alpha) \cdot \exp\left\{-\frac{1}{2}Q^2\sigma^2\right\} + \text{background}$$

where a numerical integration of $F(Q, \alpha, \psi)^2 \sin(\alpha)$ is carried out over α and ψ for:

$$\begin{aligned} F(Q, \alpha, \psi) = \Bigg[&(\rho_c - \rho_r - \rho_f + \rho_s)V_c \frac{2J_1(QR' \sin\alpha)}{QR' \sin\alpha} \frac{\sin(QL\cos\alpha/2)}{Q(L/2)\cos\alpha} \\ &+ (\rho_f - \rho_s)V_{c+f} \frac{2J_1(QR' \sin\alpha)}{QR' \sin\alpha} \frac{\sin(Q(L/2 + t_f)\cos\alpha)}{Q(L/2 + t_f)\cos\alpha} \\ &+ (\rho_r - \rho_s)V_{c+r} \frac{2J_1(Q(R' + t_r)\sin\alpha)}{Q(R' + t_r)\sin\alpha} \frac{\sin(Q(L/2)\cos\alpha)}{Q(L/2)\cos\alpha}\Bigg] \end{aligned}$$

where

$$R' = \frac{R}{\sqrt{2}}\sqrt{(1 + X_{\text{core}}^2) + (1 - X_{\text{core}}^2)\cos(\psi)}$$

and $V_t = \pi(R + t_r)(X_{\text{core}}R + t_r)L + 2\pi X_{\text{core}}R^2 t_f$ is the total volume of the bicelle, $V_c = \pi X_{\text{core}}R^2 L$ the volume of the core, $V_{c+f} = \pi X_{\text{core}}R^2(L + 2t_f)$ the volume of the core plus the volume of the faces, $V_{c+r} = \pi(R + t_r)(X_{\text{core}}R + t_r)L$ the volume of the core plus the rim, R is the radius of the core, X_{core} is the axial ratio of the core, L the length of the core, t_f the thickness of the face, t_r the thickness of the rim and J_1 the usual first order bessel function. The core has radii R and $X_{\text{core}}R$ so is circular, as for the core_shell_bicelle model, for $X_{\text{core}} = 1$. Note that you may need to limit the range of X_{core}, especially if using the Monte-Carlo algorithm, as setting radius to R/X_{core} and axial ratio to $1/X_{\text{core}}$ gives an equivalent solution!

An approximation for the effects of "Gaussian interfacial roughness" σ is included, by multiplying $I(Q)$ by $\exp\left\{-\frac{1}{2}Q^2\sigma^2\right\}$. This applies, in some way, to all interfaces in the model not just the external ones. (Note that for a one dimensional system convolution of the scattering length density profile with a Gaussian of standard deviation σ does exactly this multiplication.) Leave σ set to zero for the usual sharp interfaces.

The output of the 1D scattering intensity function for randomly oriented bicelles is then given by integrating over all possible α and ψ.

For oriented bicelles the *theta*, *phi* and *psi* orientation parameters will appear when fitting 2D data, for further details of the calculation and angular dispersions see *Oriented particles* .

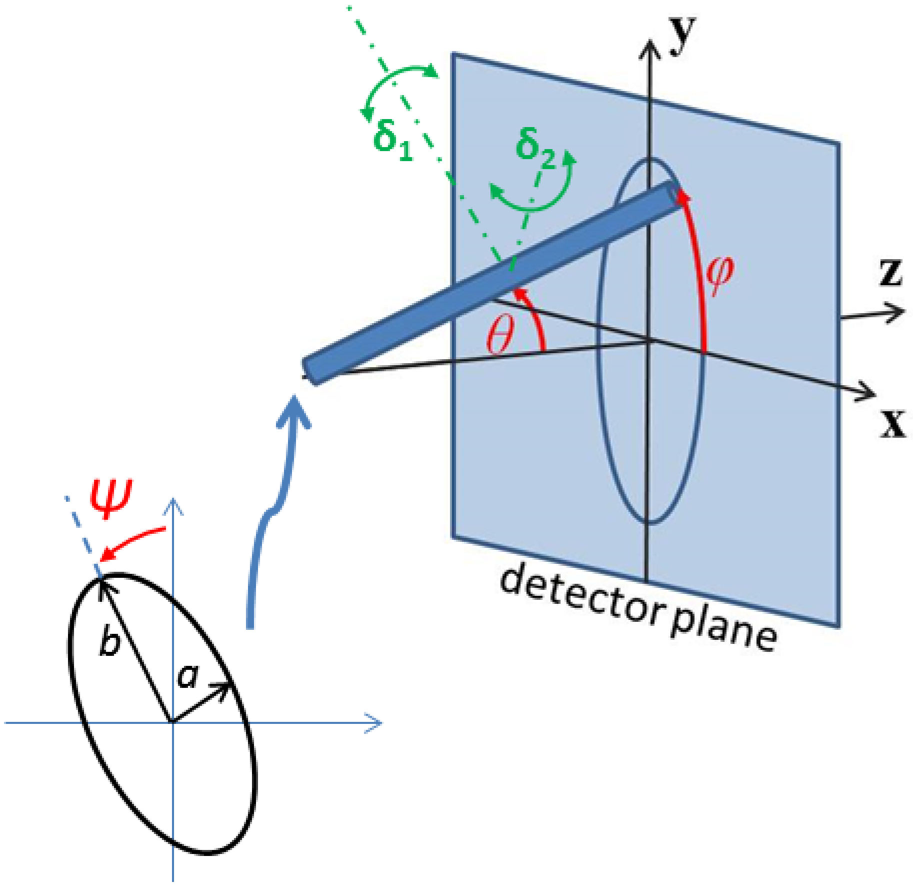


Fig. 1.17: Definition of the angles for the oriented core_shell_bicelle_elliptical particles.

References

Authorship and Verification

- **Author:** Richard Heenan **Date:** October 5, 2017
- **Last Modified by:** Richard Heenan new 2d orientation **Date:** October 5, 2017
- **Last Reviewed by:** Richard Heenan 2d calc seems agree with 1d **Date:** Nov 2, 2017

core_shell_cylinder

Right circular cylinder with a core-shell scattering length density profile.

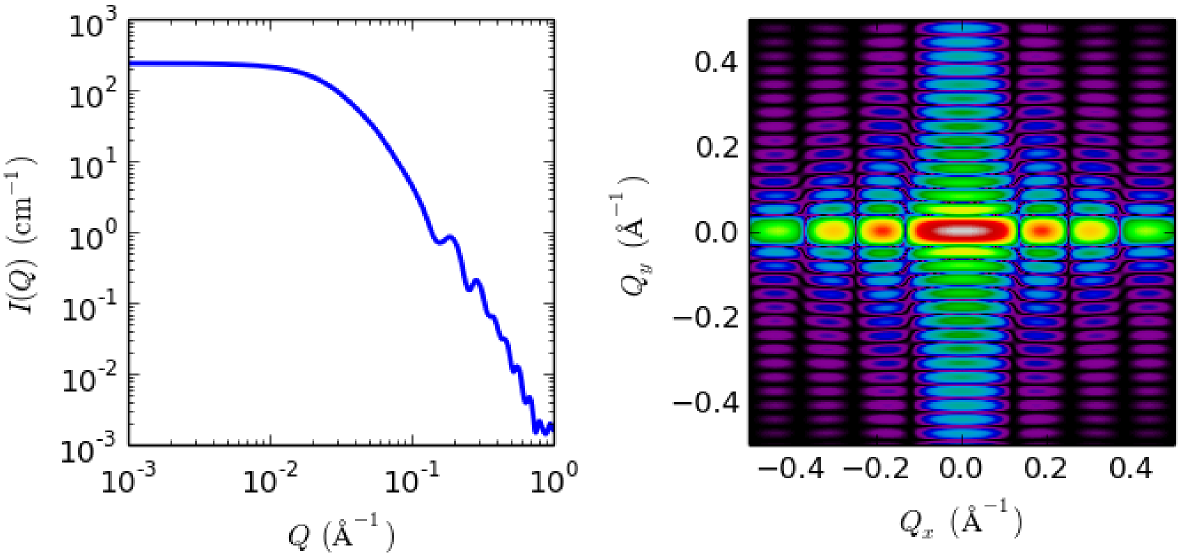


Fig. 1.18: 1D and 2D plots corresponding to the default parameters of the model.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
sld_core	Cylinder core scattering length density	10^{-6}Å^{-2}	4
sld_shell	Cylinder shell scattering length density	10^{-6}Å^{-2}	4
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	1
radius	Cylinder core radius	Å	20
thickness	Cylinder shell thickness	Å	20
length	Cylinder length	Å	400
theta	cylinder axis to beam angle	degree	60
phi	rotation about beam	degree	60

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

The output of the 2D scattering intensity function for oriented core-shell cylinders is given by (Kline, 2006[2]). The form factor is normalized by the particle volume. Note that in this model the shell envelops the entire core so that besides a "sleeve" around the core, the shell also provides two flat end caps of thickness = shell thickness. In other words the length of the total cyclinder is the length of the core cylinder plus twice the thickness of the shell. If no end caps are desired one should use the *core_shell_bicelle* and set the thickness of the end caps (in this case the "thick_face") to zero.

$$I(q,\alpha) = \frac{\text{scale}}{V_s} F^2(q,\alpha).sin(\alpha) + \text{background}$$

where

$$\begin{aligned} F(q,\alpha) = (\rho_c - \rho_s) V_c \frac{\sin\left(q\tfrac{1}{2}L\cos\alpha\right)}{q\tfrac{1}{2}L\cos\alpha} \frac{2J_1\left(qR\sin\alpha\right)}{qR\sin\alpha} \\ + (\rho_s - \rho_\text{solv}) V_s \frac{\sin\left(q\left(\tfrac{1}{2}L+T\right)\cos\alpha\right)}{q\left(\tfrac{1}{2}L+T\right)\cos\alpha} \frac{2J_1\left(q(R+T)\sin\alpha\right)}{q(R+T)\sin\alpha} \end{aligned}$$

and

$$V_s = \pi(R+T)^2(L+2T)$$

[2] S R Kline, *J Appl. Cryst.*, 39 (2006) 895

and α is the angle between the axis of the cylinder and $\vec{q}$, V_s is the total volume (i.e. including both the core and the outer shell), V_c is the volume of the core, L is the length of the core, R is the radius of the core, T is the thickness of the shell, ρ_c is the scattering length density of the core, ρ_s is the scattering length density of the shell, ρ_{solv} is the scattering length density of the solvent, and *background* is the background level. The outer radius of the shell is given by $R + T$ and the total length of the outer shell is given by $L + 2T$. $J1$ is the first order Bessel function.

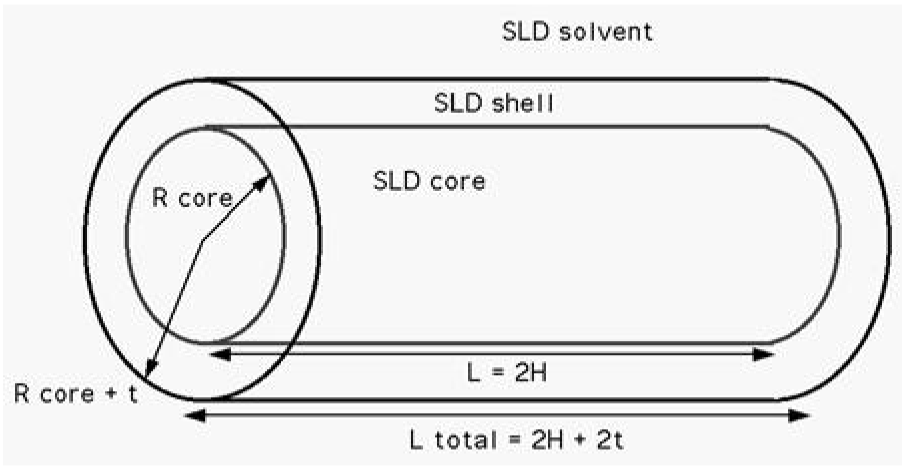


Fig. 1.19: Core shell cylinder schematic.

To provide easy access to the orientation of the core-shell cylinder, we define the axis of the cylinder using two angles θ and ϕ. (see *cylinder model*)

NB: The 2nd virial coefficient of the cylinder is calculated based on the radius and 2 length values, and used as the effective radius for $S(q)$ when $P(q) \cdot S(q)$ is applied.

The θ and ϕ parameters are not used for the 1D output.

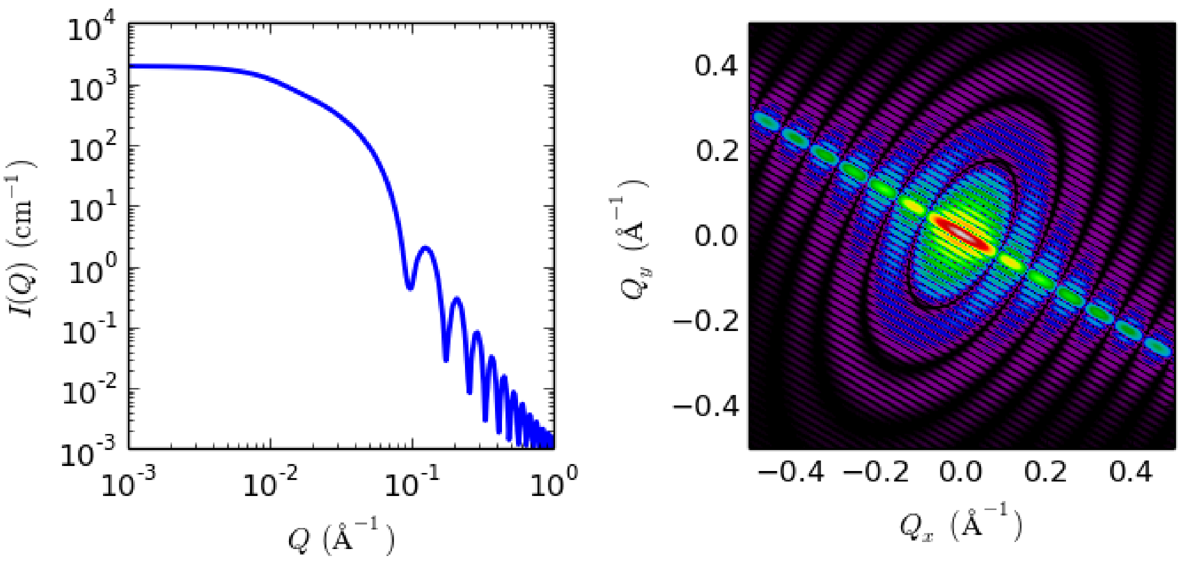


Fig. 1.20: 1D and 2D plots corresponding to the default parameters of the model.

Reference

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Last Modified by:** Paul Kienzle **Date:** Aug 8, 2016

- **Last Reviewed by:** Richard Heenan **Date:** March 18, 2016

cylinder

Right circular cylinder with uniform scattering length density.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
sld	Cylinder scattering length density	10^{-6}Å^{-2}	4
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	1
radius	Cylinder radius	Å	20
length	Cylinder length	Å	400
theta	cylinder axis to beam angle	degree	60
phi	rotation about beam	degree	60

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

For information about polarised and magnetic scattering, see the *Polarisation/Magnetic Scattering* documentation.

Definition

The output of the 2D scattering intensity function for oriented cylinders is given by (Guinier, 1955)

$$P(q,\alpha) = \frac{\text{scale}}{V} F^2(q,\alpha).sin(\alpha) + \text{background}$$

where

$$F(q,\alpha) = 2(\Delta\rho)V \frac{\sin\left(\frac{1}{2}qL\cos\alpha\right)}{\frac{1}{2}qL\cos\alpha} \frac{J_1\left(qR\sin\alpha\right)}{qR\sin\alpha}$$

and α is the angle between the axis of the cylinder and $\vec{q}$, $V = \pi R^2 L$ is the volume of the cylinder, L is the length of the cylinder, R is the radius of the cylinder, and $\Delta\rho$ (contrast) is the scattering length density difference between the scatterer and the solvent. J_1 is the first order Bessel function.

For randomly oriented particles:

$$F^2(q) = \int_0^{\pi/2} F^2(q,\alpha)\sin(\alpha)d\alpha = \int_0^1 F^2(q,u)du$$

Numerical integration is simplified by a change of variable to $u = cos(\alpha)$ with $sin(\alpha) = \sqrt{1-u^2}$.

The output of the 1D scattering intensity function for randomly oriented cylinders is thus given by

$$P(q) = \frac{\text{scale}}{V} \int_0^{\pi/2} F^2(q,\alpha)\sin\alpha\ d\alpha + \text{background}$$

NB: The 2nd virial coefficient of the cylinder is calculated based on the radius and length values, and used as the effective radius for $S(q)$ when $P(q)\cdot S(q)$ is applied.

For 2d scattering from oriented cylinders, we define the direction of the axis of the cylinder using two angles θ (note this is not the same as the scattering angle used in q) and ϕ. Those angles are defined in Fig. 1.21 , for further details see *Oriented particles* .

The θ and ϕ parameters to orient the cylinder only appear in the model when fitting 2d data.

Validation

Validation of the code was done by comparing the output of the 1D model to the output of the software provided by the NIST (Kline, 2006). The implementation of the intensity for fully oriented cylinders was done by averaging over a uniform distribution of orientations using

$$P(q) = \int_0^{\pi/2} d\phi \int_0^{\pi} p(\theta)P_0(q,\theta)\sin\theta\ d\theta$$

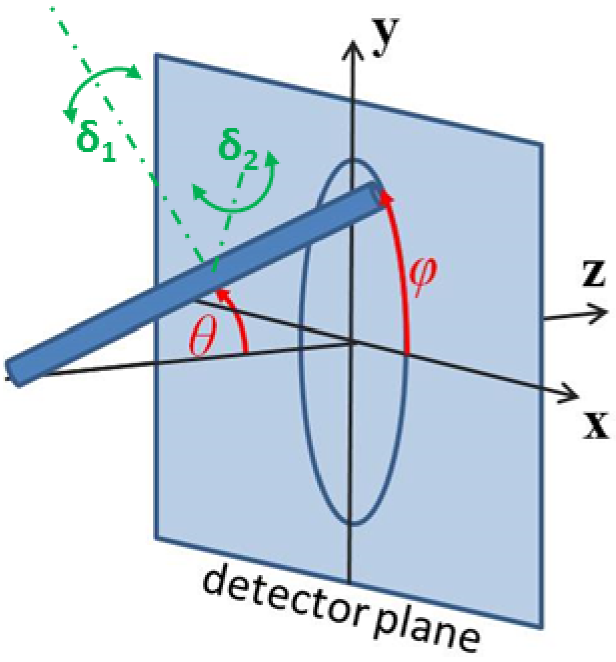


Fig. 1.21: Angles θ and ϕ orient the cylinder relative to the beam line coordinates, where the beam is along the z axis. Rotation θ, initially in the xz plane, is carried out first, then rotation ϕ about the z axis. Orientation distributions are described as rotations about two perpendicular axes δ_1 and δ_2 in the frame of the cylinder itself, which when $\theta = \phi = 0$ are parallel to the Y and X axes.

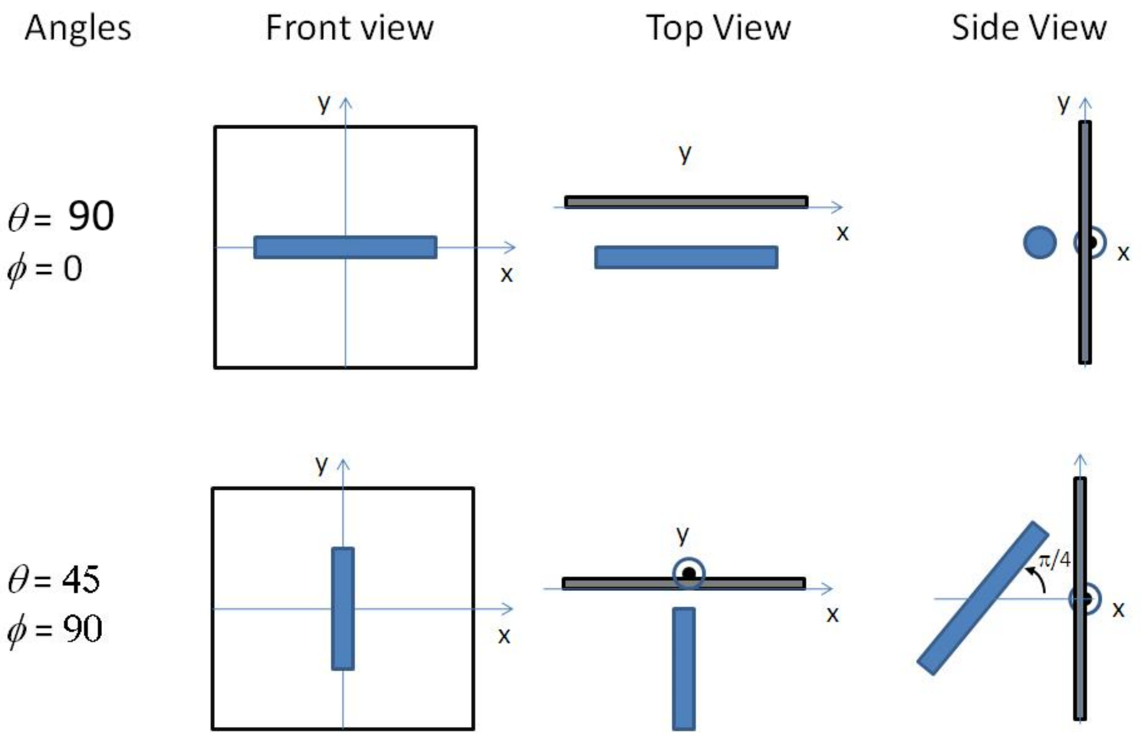


Fig. 1.22: Examples for oriented cylinders.

where $p(\theta, \phi) = 1$ is the probability distribution for the orientation and $P_0(q, \theta)$ is the scattering intensity for the fully oriented system, and then comparing to the 1D result.

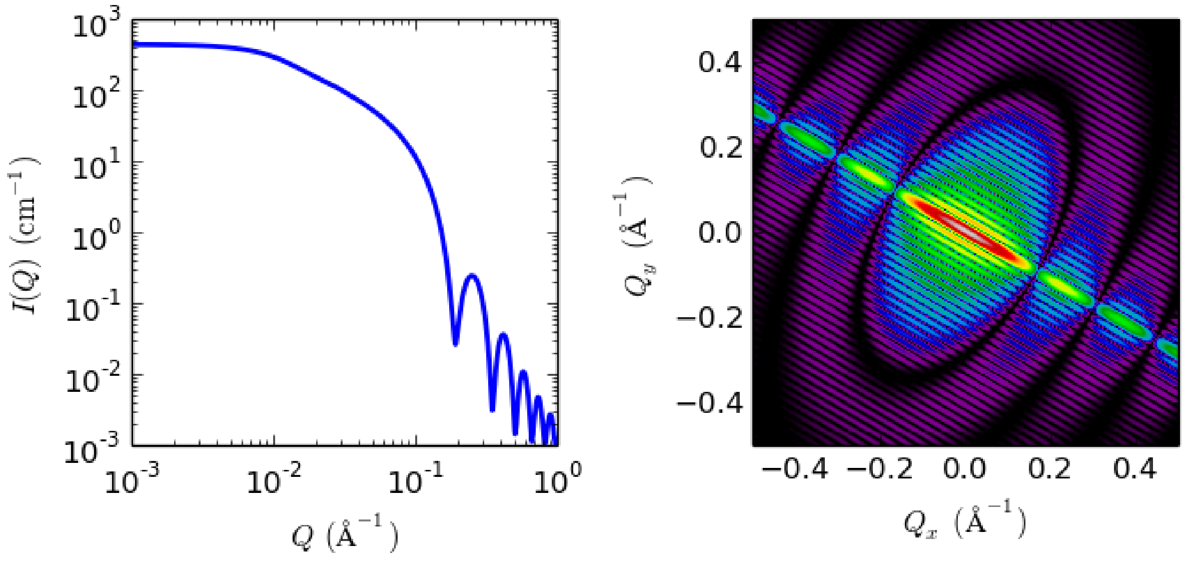


Fig. 1.23: 1D and 2D plots corresponding to the default parameters of the model.

References

J. S. Pedersen, Adv. Colloid Interface Sci. 70, 171-210 (1997). G. Fournet, Bull. Soc. Fr. Mineral. Cristallogr. 74, 39-113 (1951).

elliptical_cylinder

Form factor for an elliptical cylinder.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
radius_minor	Ellipse minor radius	Å	20
axis_ratio	Ratio of major radius over minor radius	None	1.5
length	Length of the cylinder	Å	400
sld	Cylinder scattering length density	10^{-6}Å^{-2}	4
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	1
theta	cylinder axis to beam angle	degree	90
phi	rotation about beam	degree	0
psi	rotation about cylinder axis	degree	0

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

The function calculated is

$$I(\vec{q}) = \frac{1}{V_\text{cyl}} \int d\psi \int d\phi \int p(\theta, \phi, \psi) F^2(\vec{q}, \alpha, \psi) \sin(\alpha) d\alpha$$

with the functions

$$F(q, \alpha, \psi) = 2\frac{J_1(a)\sin(b)}{ab}$$

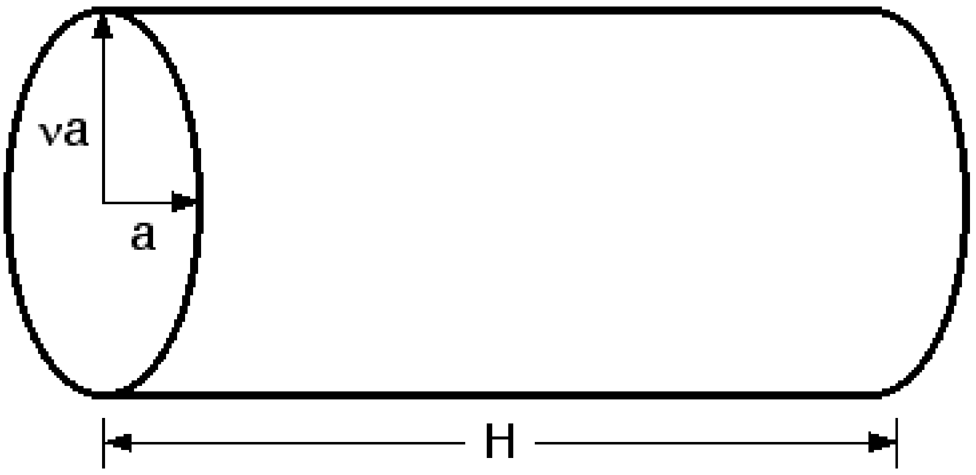


Fig. 1.24: Elliptical cylinder geometry $a = r_{\text{minor}}$ and $\nu = r_{\text{major}}/r_{\text{minor}}$ is the *axis_ratio*.

where

$$a = qr' \sin(\alpha)$$
$$b = q\frac{L}{2}\cos(\alpha)$$
$$r' = \frac{r_{minor}}{\sqrt{2}}\sqrt{(1+\nu^2) + (1-\nu^2)cos(\psi)}$$

and the angle ψ is defined as the orientation of the major axis of the ellipse with respect to the vector $\vec{q}$. The angle α is the angle between the axis of the cylinder and $\vec{q}$.

For 1D scattering, with no preferred orientation, the form factor is averaged over all possible orientations and normalized by the particle volume

$$P(q) = \text{scale} < F^2 > /V$$

For 2d data the orientation of the particle is required, described using a different set of angles as in the diagrams below, for further details of the calculation and angular dispersions see *Oriented particles* .

The θ and ϕ parameters to orient the cylinder only appear in the model when fitting 2d data.

NB: The 2nd virial coefficient of the cylinder is calculated based on the averaged radius $(= \sqrt{r_{\text{minor}}^2 * \text{axis ratio}})$ and length values, and used as the effective radius for $S(Q)$ when $P(Q) * S(Q)$ is applied.

Validation

Validation of our code was done by comparing the output of the 1D calculation to the angular average of the output of the 2D calculation over all possible angles.

In the 2D average, more binning in the angle ϕ is necessary to get the proper result. The following figure shows the results of the averaging by varying the number of angular bins.

References

L A Feigin and D I Svergun, *Structure Analysis by Small-Angle X-Ray and Neutron Scattering*, Plenum, New York, (1987) [see table 3.4]

Authorship and Verification

- **Author:**
- **Last Modified by:**
- **Last Reviewed by:** Richard Heenan - corrected equation in docs **Date:** December 21, 2016

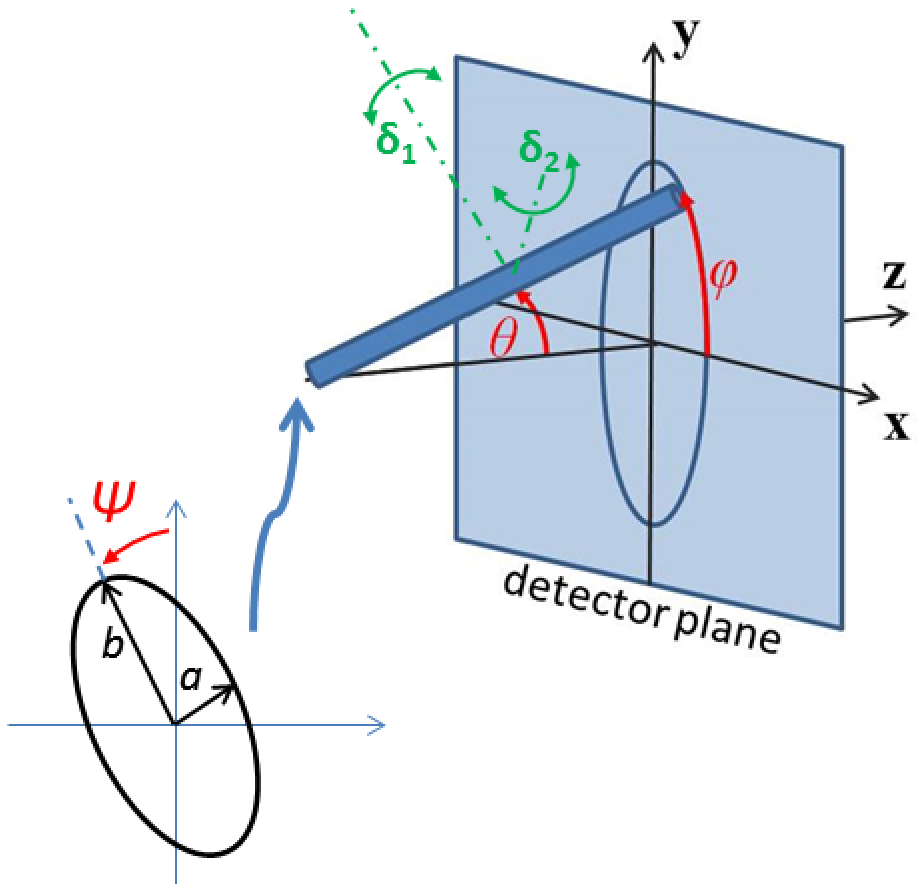


Fig. 1.25: Note that the angles here are not the same as in the equations for the scattering function. Rotation θ, initially in the xz plane, is carried out first, then rotation ϕ about the z axis, finally rotation Ψ is now around the axis of the cylinder. The neutron or X-ray beam is along the z axis.

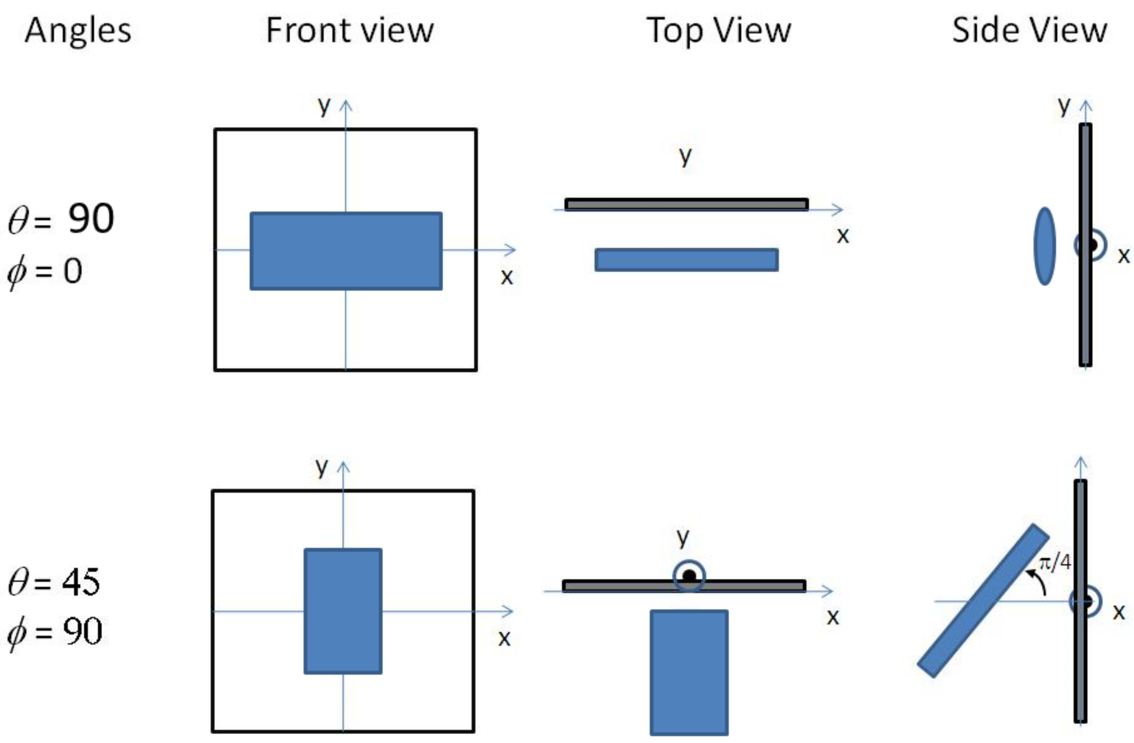


Fig. 1.26: Examples of the angles for oriented elliptical cylinders against the detector plane, with Ψ = 0.

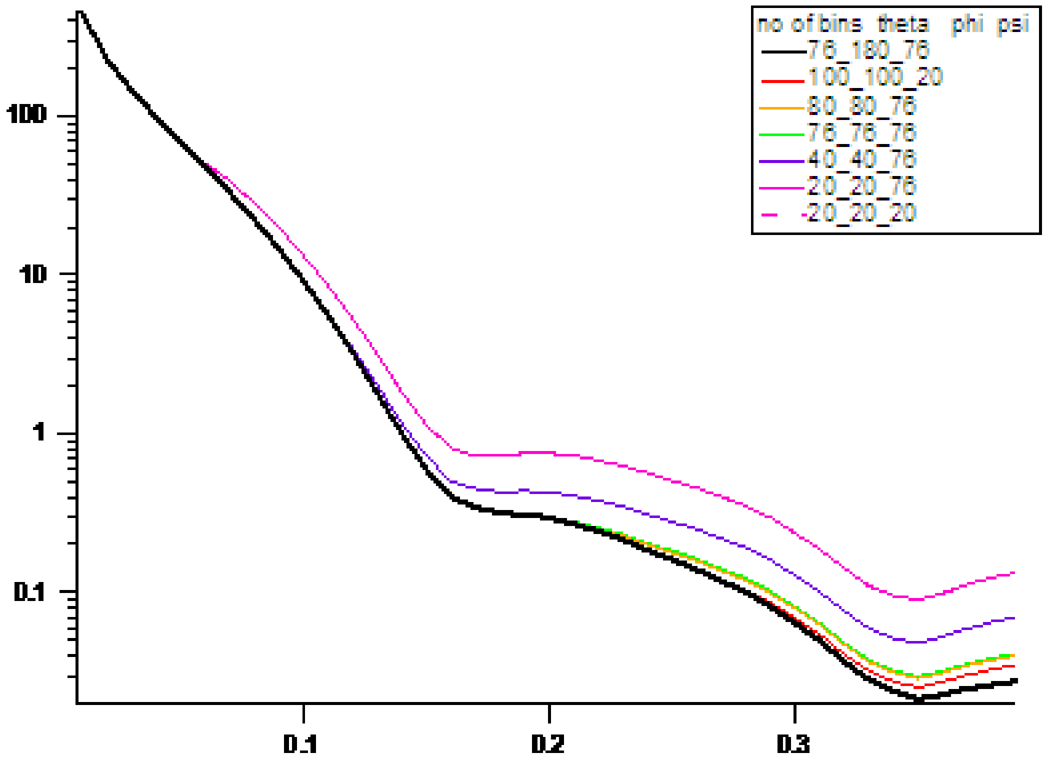


Fig. 1.27: The intensities averaged from 2D over different numbers of bins and angles.

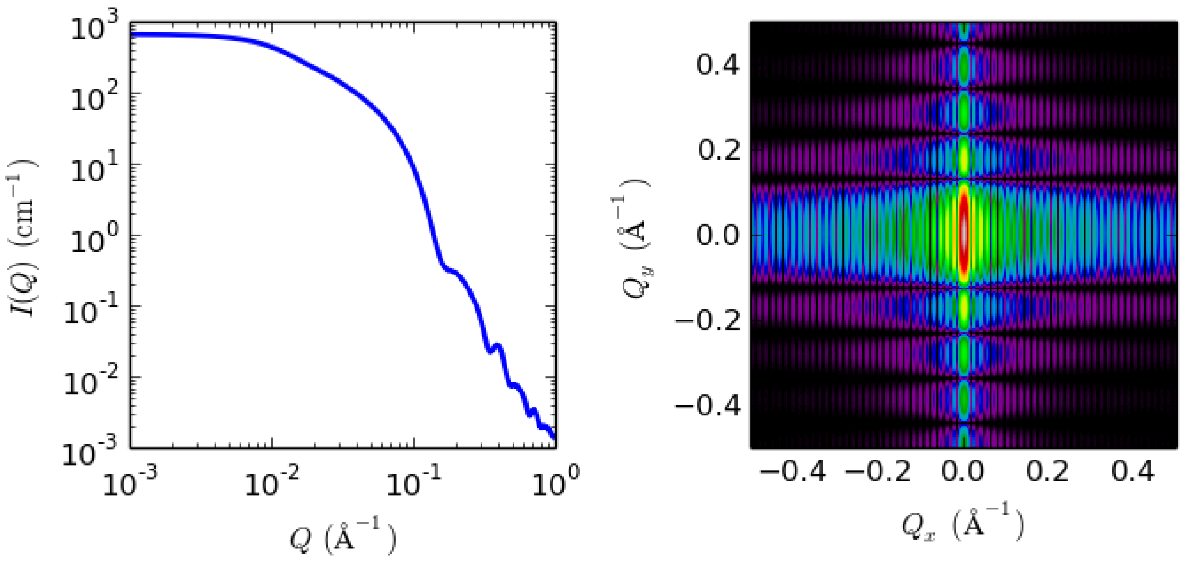


Fig. 1.28: 1D and 2D plots corresponding to the default parameters of the model.

flexible_cylinder

Flexible cylinder where the form factor is normalized by the volumeof the cylinder.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
length	Length of the flexible cylinder	Å	1000
kuhn_length	Kuhn length of the flexible cylinder	Å	100
radius	Radius of the flexible cylinder	Å	20
sld	Cylinder scattering length density	10^{-6}Å^{-2}	1
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	6.3

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

This model provides the form factor, $P(q)$, for a flexible cylinder where the form factor is normalized by the volume of the cylinder. **Inter-cylinder interactions are NOT provided for.**

$$P(q) = \text{scale} \left\langle F^2 \right\rangle / V + \text{background}$$

where the averaging $\langle \ldots \rangle$ is applied only for the 1D calculation

The 2D scattering intensity is the same as 1D, regardless of the orientation of the q vector which is defined as

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

Definitions

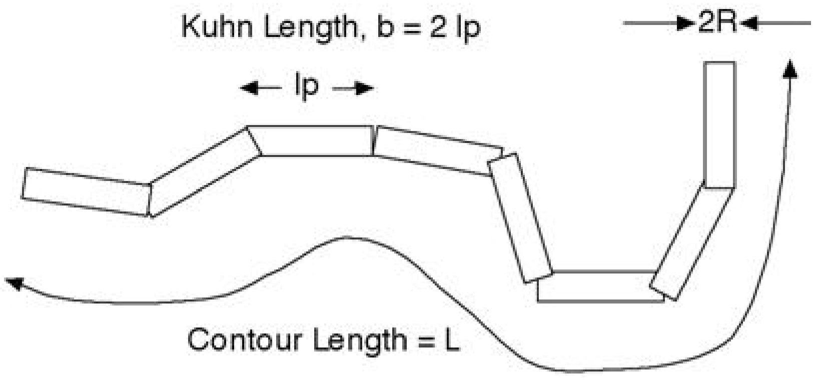
The chain of contour length, L, (the total length) can be described as a chain of some number of locally stiff segments of length l_p, the persistence length (the length along the cylinder over which the flexible cylinder can be considered a rigid rod). The Kuhn length ($b = 2 * l_p$) is also used to describe the stiffness of a chain.

The returned value is in units of cm^{-1}, on absolute scale.

In the parameters, the sld and sld_solvent represent the SLD of the cylinder and solvent respectively.

Our model uses the form factor calculations implemented in a c-library provided by the NIST Center for Neutron Research (Kline, 2006).

From the reference:



'Method 3 With Excluded Volume' is used. The model is a parametrization of simulations of a discrete representation of the worm-like chain model of Kratky and Porod applied in the pseudocontinuous limit. See equations (13,26-27) in the original reference for the details.

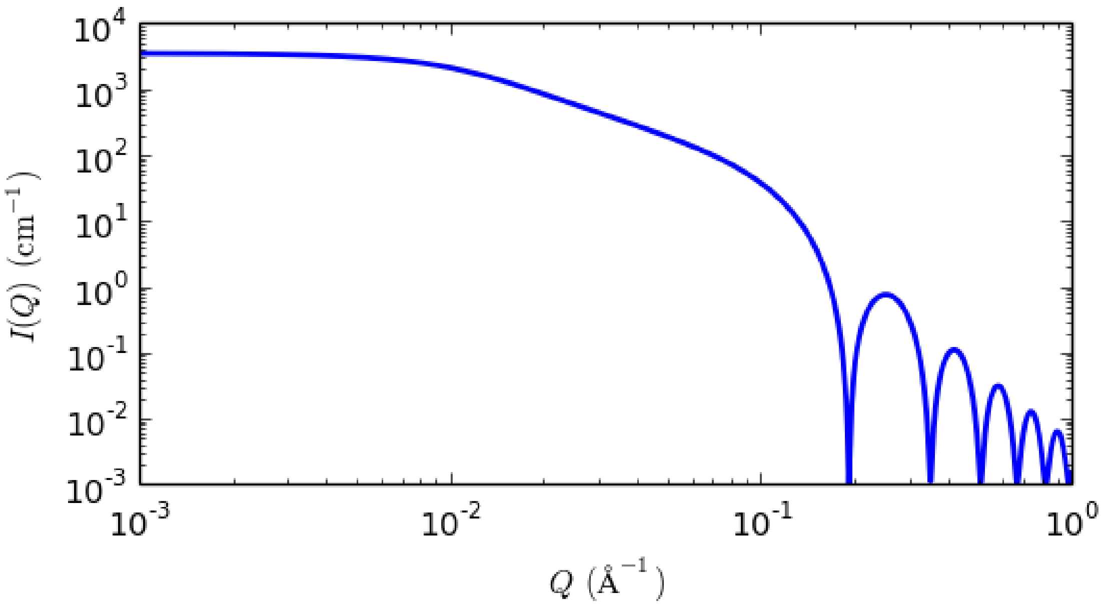


Fig. 1.29: 1D plot corresponding to the default parameters of the model.

References

J S Pedersen and P Schurtenberger. *Scattering functions of semiflexible polymers with and without excluded volume effects.* Macromolecules, 29 (1996) 7602-7612

Correction of the formula can be found in

W R Chen, P D Butler and L J Magid, *Incorporating Intermicellar Interactions in the Fitting of SANS Data from Cationic Wormlike Micelles.* Langmuir, 22(15) 2006 6539-6548

flexible_cylinder_elliptical

Flexible cylinder wth an elliptical cross section and a uniform scattering length density.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
length	Length of the flexible cylinder	Å	1000
kuhn_length	Kuhn length of the flexible cylinder	Å	100
radius	Radius of the flexible cylinder	Å	20
axis_ratio	Axis_ratio (major_radius/minor_radius	None	1.5
sld	Cylinder scattering length density	10^{-6}Å^{-2}	1
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	6.3

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

This model calculates the form factor for a flexible cylinder with an elliptical cross section and a uniform scattering length density. The non-negligible diameter of the cylinder is included by accounting for excluded volume interactions within the walk of a single cylinder. The form factor is normalized by the particle volume such that

$$P(q) = \text{scale}\left\langle F^2\right\rangle / V + \text{background}$$

where the averaging $\langle \ldots \rangle$ is over all possible orientations of the flexible cylinder.

The 2D scattering intensity is the same as 1D, regardless of the orientation of the q vector which is defined as

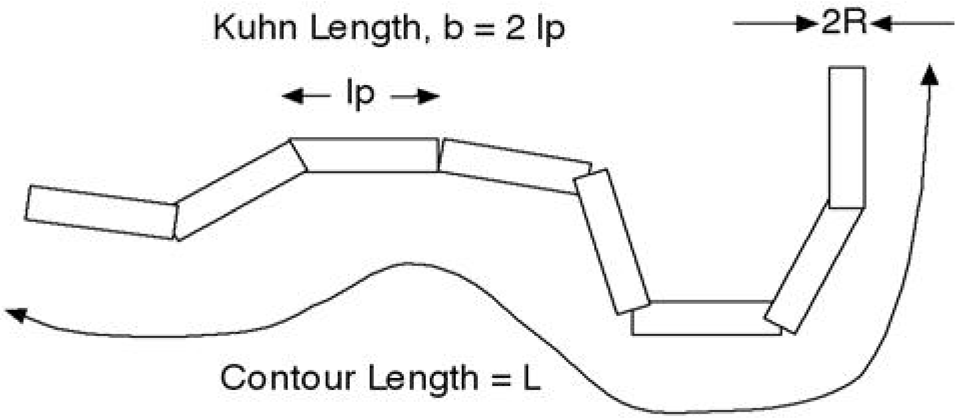
$$q = \sqrt{q_x^2 + q_y^2}$$

Definitions

The function calculated in a similar way to that for the flexible_cylinder model from the reference given below using the author's "Method 3 With Excluded Volume". The model is a parameterization of simulations of a discrete representation of the worm-like chain model of Kratky and Porod applied in the pseudo-continuous limit. See equations (13, 26-27) in the original reference for the details.

Note: There are several typos in the original reference that have been corrected by WRC. Details of the corrections are in the reference below. Most notably

- Equation (13): the term $(1 - w(QR))$ should swap position with $w(QR)$
- Equations (23) and (24) are incorrect; WRC has entered these into Mathematica and solved analytically. The results were then converted to code.
- Equation (27) should be $q0 = max(a3/sqrt(RgSquare), 3)$ instead of $max(a3 * b/sqrt(RgSquare), 3)$
- The scattering function is negative for a range of parameter values and q-values that are experimentally accessible. A correction function has been added to give the proper behavior.



The chain of contour length, L, (the total length) can be described as a chain of some number of locally stiff segments of length l_p, the persistence length (the length along the cylinder over which the flexible cylinder can be considered a rigid rod). The Kuhn length $(b = 2 * l_p)$ is also used to describe the stiffness of a chain.

The cross section of the cylinder is elliptical, with minor radius a . The major radius is larger, so of course, **the axis ratio (parameter 5) must be greater than one.** Simple constraints should be applied during curve fitting to maintain this inequality.

The returned value is in units of cm^{-1}, on absolute scale.

In the parameters, the sld and $sld\_solvent$ represent the SLD of the chain/cylinder and solvent respectively. The *scale*, and the contrast are both multiplicative factors in the model and are perfectly correlated. One or both of these parameters must be held fixed during model fitting.

No inter-cylinder interference effects are included in this calculation.

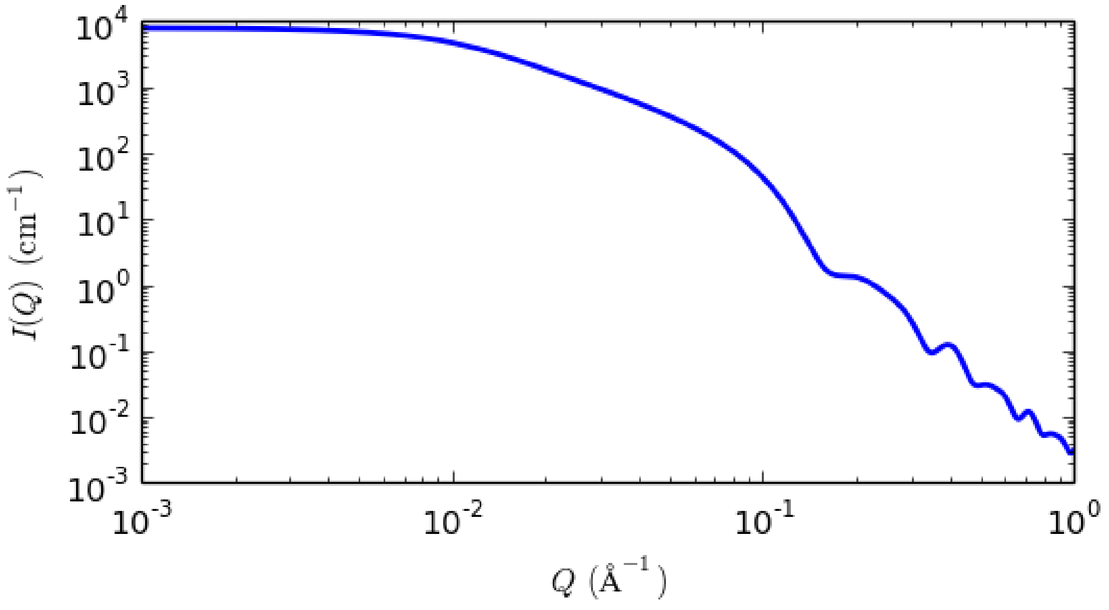


Fig. 1.30: 1D plot corresponding to the default parameters of the model.

References

J S Pedersen and P Schurtenberger. *Scattering functions of semiflexible polymers with and without excluded volume effects.* Macromolecules, 29 (1996) 7602-7612

Correction of the formula can be found in

W R Chen, P D Butler and L J Magid, *Incorporating Intermicellar Interactions in the Fitting of SANS Data from Cationic Wormlike Micelles.* Langmuir, 22(15) 2006 6539-6548

hollow_cylinder

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
radius	Cylinder core radius	Å	20
thickness	Cylinder wall thickness	Å	10
length	Cylinder total length	Å	400
sld	Cylinder sld	10^{-6}Å^{-2}	6.3
sld_solvent	Solvent sld	10^{-6}Å^{-2}	1
theta	Cylinder axis to beam angle	degree	90
phi	Rotation about beam	degree	0

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

This model provides the form factor, $P(q)$, for a monodisperse hollow right angle circular cylinder (rigid tube) where the The inside and outside of the hollow cylinder are assumed to have the same SLD and the form factor is thus normalized by the volume of the tube (i.e. not by the total cylinder volume).

$$P(q) = \text{scale} \left\langle F^2 \right\rangle / V_\text{shell} + \text{background}$$

where the averaging $\langle \ldots \rangle$ is applied only for the 1D calculation. If Intensity is given on an absolute scale, the scale factor here is the volume fraction of the shell. This differs from the *core_shell_cylinder* in that, in that case, scale is the volume fraction of the entire cylinder (core+shell). The application might be for a bilayer which wraps into a hollow tube and the volume fraction of material is all in the shell, whereas the *core_shell_cylinder* model might be used for a cylindrical micelle where the tails in the core have a different SLD than the headgroups (in the shell) and the volume fraction of material comes fromm the whole cyclinder. NOTE: the hollow_cylinder represents a tube whereas the core_shell_cylinder includes a shell layer covering the ends (end caps) as well.

The 1D scattering intensity is calculated in the following way (Guinier, 1955)

$$\begin{aligned}
P(q) &= (\text{scale})V_\text{shell}\Delta\rho^2 \int_0^1 \Psi^2 \left[q_z, R_\text{outer}(1-x^2)^{1/2}, R_\text{core}(1-x^2)^{1/2}\right] \left[\frac{\sin(qHx)}{qHx}\right]^2 dx \\
\Psi[q,y,z] &= \frac{1}{1-\gamma^2}\left[\Lambda(qy) - \gamma^2\Lambda(qz)\right] \\
\Lambda(a) &= 2J_1(a)/a \\
\gamma &= R_\text{core}/R_\text{outer} \\
V_\text{shell} &= \pi\left(R_\text{outer}^2 - R_\text{core}^2\right)L \\
J_1(x) &= (\sin(x) - x\cdot\cos(x))/x^2
\end{aligned}$$

where *scale* is a scale factor, $H = L/2$ and J_1 is the 1st order Bessel function.

NB: The 2nd virial coefficient of the cylinder is calculated based on the outer radius and full length, which give an the effective radius for structure factor $S(q)$ when $P(q) \cdot S(q)$ is applied.

In the parameters,the *radius* is R_core while *thickness* is $R_\text{outer} - R_\text{core}$.

To provide easy access to the orientation of the core-shell cylinder, we define the axis of the cylinder using two angles θ and ϕ (see *cylinder model*).

References

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Last Modified by: Paul Butler Date: September 06, 2018** (corrected VR calculation)
- **Last Reviewed by:** Paul Butler **Date:** September 06, 2018

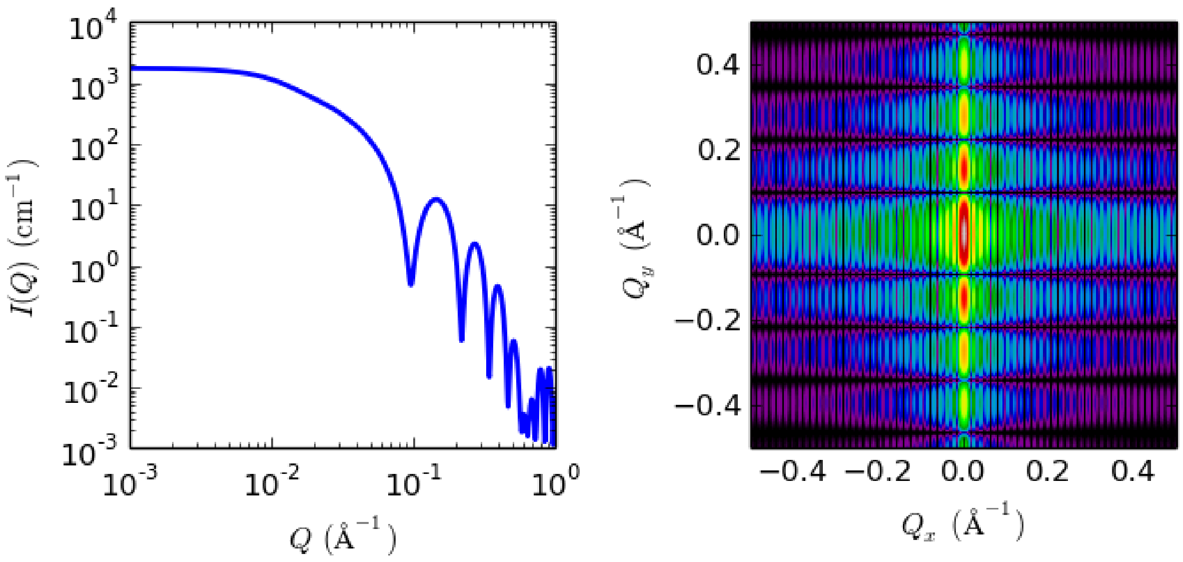


Fig. 1.31: 1D and 2D plots corresponding to the default parameters of the model.

pearl_necklace

Colloidal spheres chained together with no preferential orientation

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
radius	Mean radius of the chained spheres	Å	80
edge_sep	Mean separation of chained particles	Å	350
thick_string	Thickness of the chain linkage	Å	2.5
num_pearls	Number of pearls in the necklace (must be integer)	none	3
sld	Scattering length density of the chained spheres	10^{-6}Å^{-2}	1
sld_string	Scattering length density of the chain linkage	10^{-6}Å^{-2}	1
sld_solvent	Scattering length density of the solvent	10^{-6}Å^{-2}	6.3

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

This model provides the form factor for a pearl necklace composed of two elements: N pearls (homogeneous spheres of radius R) freely jointed by M rods (like strings - with a total mass $Mw = M * m_r + N * m_s$, and the string segment length (or edge separation) l (= A - 2R)). A is the center-to-center pearl separation distance.

Definition

The output of the scattering intensity function for the pearl_necklace is given by (Schweins, 2004)

$$I(q) = \frac{\text{scale}}{V} \cdot \frac{(S_{ss}(q) + S_{ff}(q) + S_{fs}(q))}{(M \cdot m_f + N \cdot m_s)^2} + \text{bkg}$$

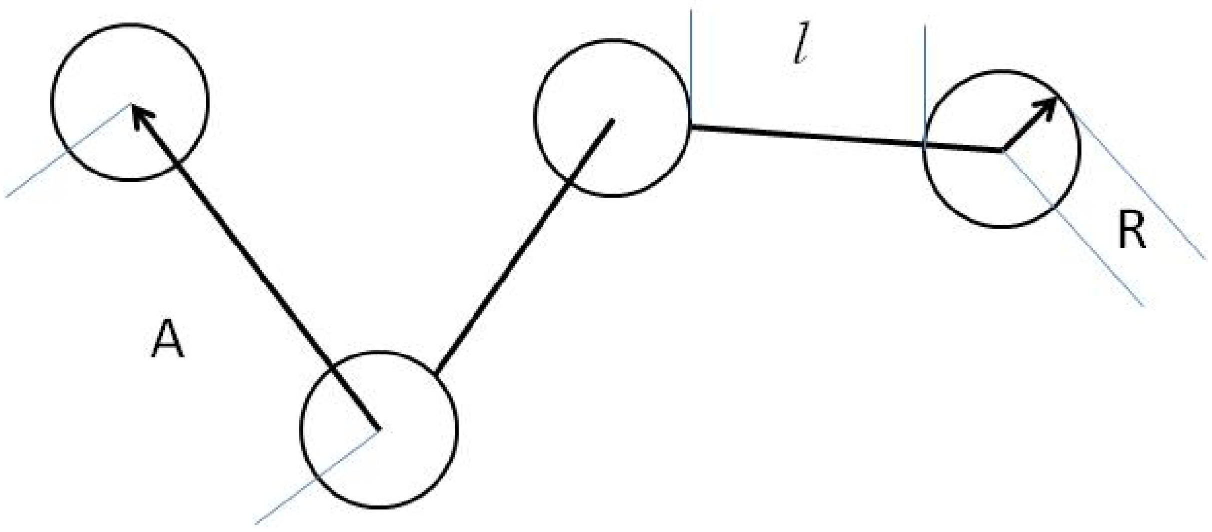


Fig. 1.32: Pearl Necklace schematic

where

$$S_{ss}(q) = sm_s^2\psi^2(q)[\frac{N}{1-sin(qA)/qA} - \frac{N}{2} - \frac{1-(sin(qA)/qA)^N}{(1-sin(qA)/qA)^2} \cdot \frac{sin(qA)}{qA}]$$

$$S_{ff}(q) = sm_r^2[M\{2\Lambda(q) - (\frac{sin(ql/2)}{ql/2})\} + \frac{2M\beta^2(q)}{1-sin(qA)/qA} - 2\beta^2(q) \cdot \frac{1-(sin(qA)/qA)^M}{(1-sin(qA)/qA)^2}]$$

$$S_{fs}(q) = m_r\beta(q) \cdot m_s\psi(q) \cdot 4[\frac{N-1}{1-sin(qA)/qA} - \frac{1-(sin(qA)/qA)^{N-1}}{(1-sin(qA)/qA)^2} \cdot \frac{sin(qA)}{qA}]$$

$$\psi(q) = 3 \cdot \frac{sin(qR) - (qR) \cdot cos(qR)}{(qR)^3}$$

$$\Lambda(q) = \frac{\int_0^{ql} \frac{sin(t)}{t} dt}{ql}$$

$$\beta(q) = \frac{\int_{qR}^{q(A-R)} \frac{sin(t)}{t} dt}{ql}$$

where the mass m_i is (SLD_i - $SLD_{solvent}$) * (volume of the N pearls/rods). V is the total volume of the necklace.

The 2D scattering intensity is the same as $P(q)$ above, regardless of the orientation of the q vector.

The returned value is scaled to units of cm^{-1} and the parameters of the pearl_necklace model are the following

NB: *num_pearls* must be an integer.

References

R Schweins and K Huber, *Particle Scattering Factor of Pearl Necklace Chains*, *Macromol. Symp.* 211 (2004) 25-42 2004

pringle

The Pringle model provides the form factor, $P(q)$, for a 'pringle' or 'saddle-shaped' disc that is bent in two directions.

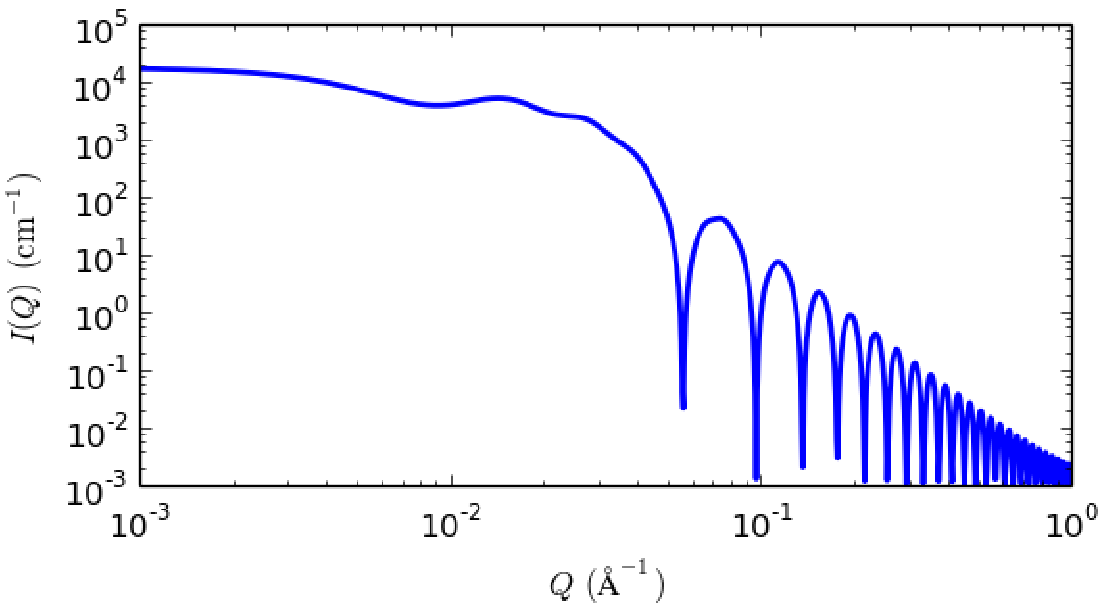


Fig. 1.33: 1D plot corresponding to the default parameters of the model.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
radius	Pringle radius	Å	60
thickness	Thickness of pringle	Å	10
alpha	Curvature parameter alpha	None	0.001
beta	Curvature paramter beta	None	0.02
sld	Pringle sld	10^{-6}Å^{-2}	1
sld_solvent	Solvent sld	10^{-6}Å^{-2}	6.3

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

The form factor for this bent disc is essentially that of a hyperbolic paraboloid and calculated as

$$P(q) = (\Delta\rho)^2 V \int_0^{\pi/2} d\psi \sin\psi sinc^2 \left(\frac{qd\cos\psi}{2}\right) \left[\left(S_0^2 + C_0^2\right) + 2\sum_{n=1}^{\infty}\left(S_n^2 + C_n^2\right)\right]$$

where

$$C_n = \frac{1}{r^2}\int_0^R rdr \cos(qr^2\alpha\cos\psi)J_n\left(qr^2\beta\cos\psi\right)J_{2n}\left(qr\sin\psi\right)$$

$$S_n = \frac{1}{r^2}\int_0^R rdr \sin(qr^2\alpha\cos\psi)J_n\left(qr^2\beta\cos\psi\right)J_{2n}\left(qr\sin\psi\right)$$

and $\Delta\rho$ is $\rho_{pringle} - \rho_{solvent}$, V is the volume of the disc, ψ is the angle between the normal to the disc and the q vector, d and R are the "pringle" thickness and radius respectively, α and β are the two curvature parameters, and J_n is the nth order Bessel function of the first kind.

Reference

Karen Edler, Universtiy of Bath, Private Communication. 2012. Derivation by Stefan Alexandru Rautu.

- **Author:** Andrew Jackson **Date:** 2008

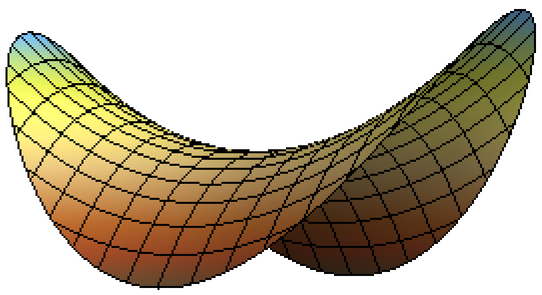


Fig. 1.34: Schematic of model shape (Graphic from Matt Henderson, matt@matthen.com)

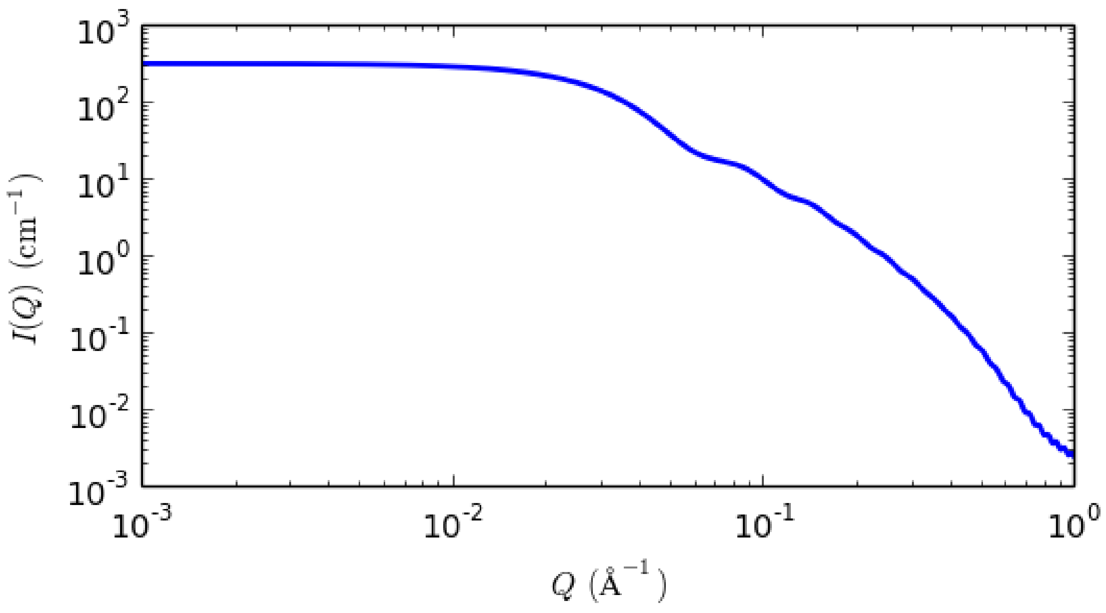


Fig. 1.35: 1D plot corresponding to the default parameters of the model.

- **Last Modified by:** Wojciech Wpotrzebowski **Date:** March 20, 2016
- **Last Reviewed by:** Andrew Jackson **Date:** September 26, 2016

stacked_disks

Form factor for a stacked set of non exfoliated core/shell disks

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
thick_core	Thickness of the core disk	Å	10
thick_layer	Thickness of layer each side of core	Å	10
radius	Radius of the stacked disk	Å	15
n_stacking	Number of stacked layer/core/layer disks	None	1
sigma_d	Sigma of nearest neighbor spacing	Å	0
sld_core	Core scattering length density	$10^{-6}Å^{-2}$	4
sld_layer	Layer scattering length density	$10^{-6}Å^{-2}$	0
sld_solvent	Solvent scattering length density	$10^{-6}Å^{-2}$	5
theta	Orientation of the stacked disk axis w/respect incoming beam	degree	0
phi	Rotation about beam	degree	0

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

This model provides the form factor, $P(q)$, for stacked discs (tactoids) with a core/layer structure which is constructed itself as $P(q)S(Q)$ multiplying a $P(q)$ for individual core/layer disks by a structure factor $S(q)$ proposed by Kratky and Porod in 1949[1] assuming the next neighbor distance (d-spacing) in the stack of parallel discs obeys a Gaussian distribution. As such the normalization of this "composite" form factor is relative to the individual disk volume, not the volume of the stack of disks. This model is appropriate for example for non non exfoliated clay particles such as Laponite.

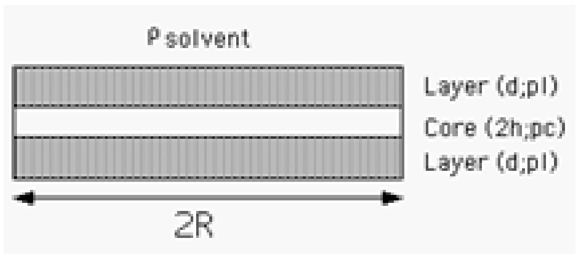


Fig. 1.36: Geometry of a single core/layer disk

The scattered intensity $I(q)$ is calculated as

$$I(q) = N \int_0^{\pi/2} \left[\Delta\rho_t \left(V_t f_t(q,\alpha) - V_c f_c(q,\alpha)\right) + \Delta\rho_c V_c f_c(q,\alpha)\right]^2 S(q,\alpha) \sin\alpha \, d\alpha + \text{background}$$

where the contrast

$$\Delta\rho_i = \rho_i - \rho_{\text{solvent}}$$

and N is the number of individual (single) discs per unit volume, α is the angle between the axis of the disc and

[1] O Kratky and G Porod, *J. Colloid Science*, 4, (1949) 35

q, and V_t and V_c are the total volume and the core volume of a single disc, respectively, and

$$f_t(q,\alpha) = \left(\frac{\sin(q(d+h)\cos\alpha)}{q(d+h)\cos\alpha}\right)\left(\frac{2J_1(qR\sin\alpha)}{qR\sin\alpha}\right)$$
$$f_c(q,\alpha) = \left(\frac{\sin(qh)\cos\alpha)}{qh\cos\alpha}\right)\left(\frac{2J_1(qR\sin\alpha)}{qR\sin\alpha}\right)$$

where d = thickness of the layer (*thick_layer*), $2h$ = core thickness (*thick_core*), and R = radius of the disc (*radius*).

$$S(q,\alpha) = 1 + \frac{1}{2}\sum_{k=1}^{n}(n-k)\cos\left(kDq\cos\alpha\right)\exp\left[-k(q)^2(D\cos\alpha\;\sigma_d)^2/2\right]$$

where n is the total number of the disc stacked (*n_stacking*), $D = 2(d+h)$ is the next neighbor center-to-center distance (d-spacing), and σ_d = the Gaussian standard deviation of the d-spacing (*sigma_d*). Note that $D\cos(\alpha)$ is the component of D parallel to q and the last term in the equation above is effectively a Debye-Waller factor term.

Note: 1. Each assembly in the stack is layer/core/layer, so the spacing of the cores is core plus two layers. The 2nd virial coefficient of the cylinder is calculated based on the *radius* and *length* = *n_stacking* * (*thick_core* + 2 * *thick_layer*) values, and used as the effective radius for $S(Q)$ when $P(Q) * S(Q)$ is applied.

2. the resolution smearing calculation uses 76 Gaussian quadrature points to properly smear the model since the function is HIGHLY oscillatory, especially around the q-values that correspond to the repeat distance of the layers.

2d scattering from oriented stacks is calculated in the same way as for cylinders, for further details of the calculation and angular dispersions see *Oriented particles*.

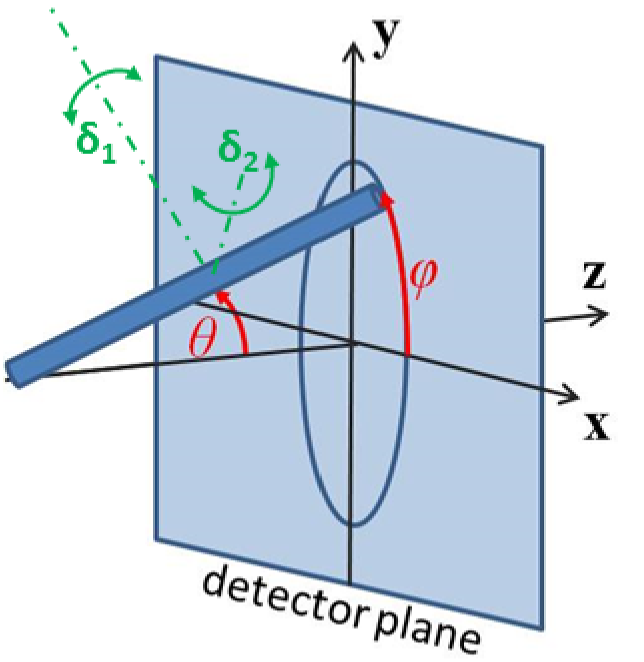


Fig. 1.37: Angles θ and ϕ orient the stack of discs relative to the beam line coordinates, where the beam is along the z axis. Rotation θ, initially in the xz plane, is carried out first, then rotation ϕ about the z axis. Orientation distributions are described as rotations about two perpendicular axes δ_1 and δ_2 in the frame of the cylinder itself, which when $\theta = \phi = 0$ are parallel to the Y and X axes.

Our model is derived from the form factor calculations implemented in a c-library provided by the NIST Center for Neutron Research[2]

[2] S R Kline, *J Appl. Cryst.*, 39 (2006) 895

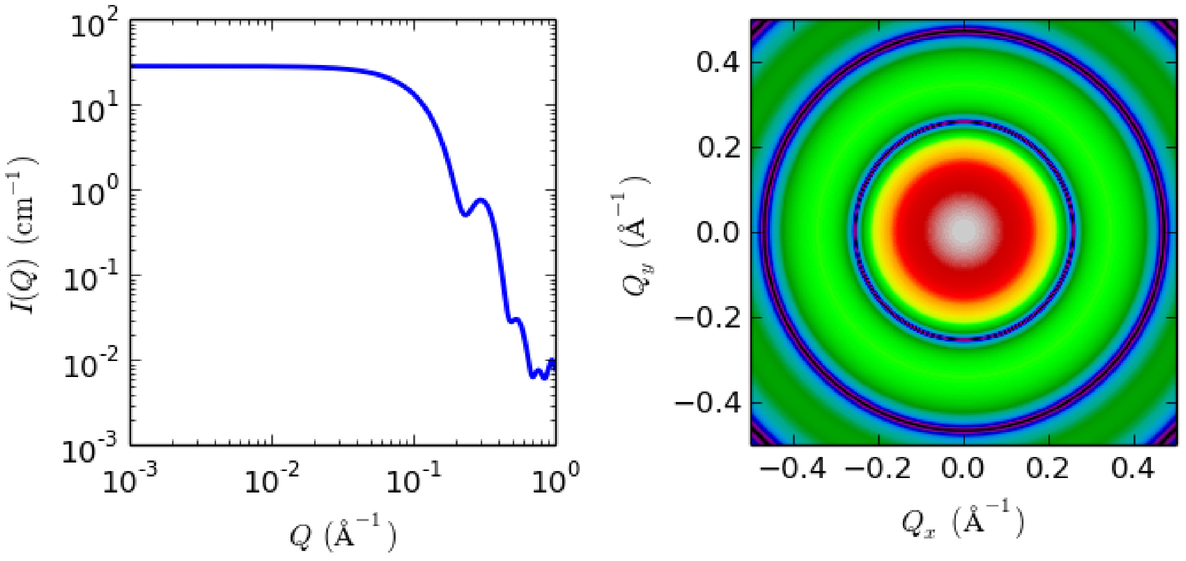


Fig. 1.38: 1D and 2D plots corresponding to the default parameters of the model.

References

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Last Modified by:** Paul Butler and Paul Kienzle **Date:** November 26, 2016
- **Last Reviewed by:** Paul Butler and Paul Kienzle **Date:** November 26, 2016

1.1.2 Ellipsoid Functions

core_shell_ellipsoid

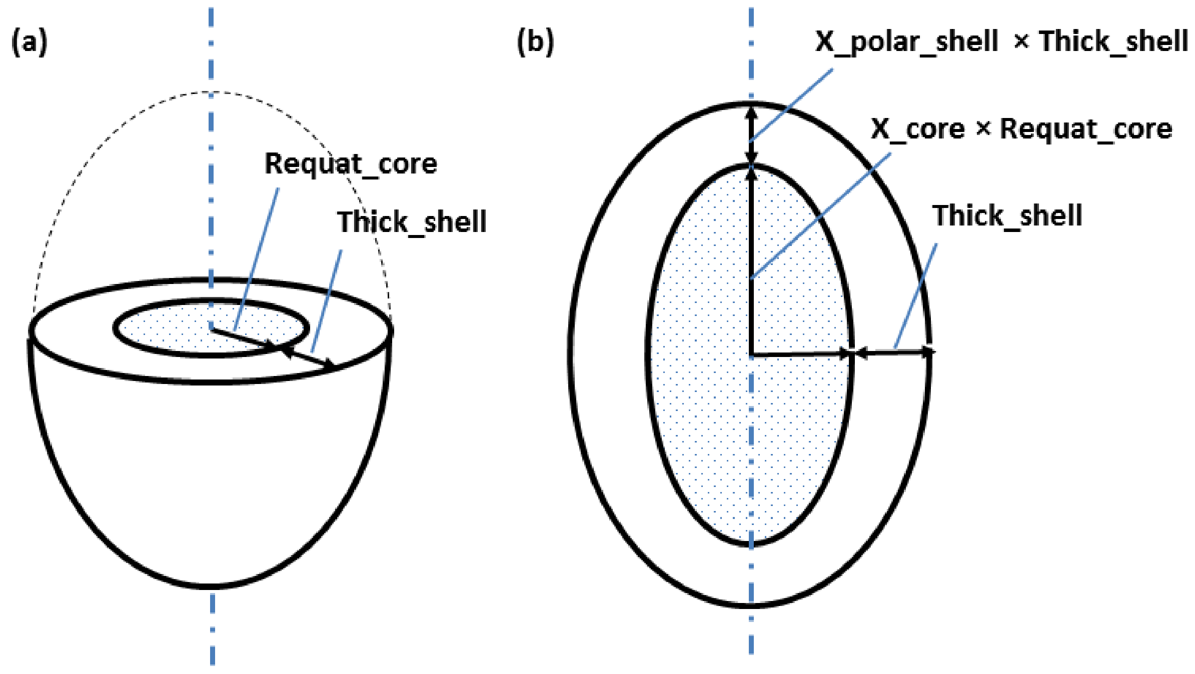
Form factor for an spheroid ellipsoid particle with a core shell structure.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
radius_equat_core	Equatorial radius of core	Å	20
x_core	axial ratio of core, X = r_polar/r_equatorial	None	3
thick_shell	thickness of shell at equator	Å	30
x_polar_shell	ratio of thickness of shell at pole to that at equator	None	1
sld_core	Core scattering length density	10^{-6}Å^{-2}	2
sld_shell	Shell scattering length density	10^{-6}Å^{-2}	1
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	6.3
theta	elipsoid axis to beam angle	degree	0
phi	rotation about beam	degree	0

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

Parameters for this model are the core axial ratio X and a shell thickness, which are more often what we would like to determine and makes the model better behaved, particularly when polydispersity is applied than the four independent radii used in the original parameterization of this model.



The geometric parameters of this model are shown in the diagram above, which shows (a) a cut through at the circular equator and (b) a cross section through the poles, of a prolate ellipsoid.

When *X_core < 1* the core is oblate; when *X_core > 1* it is prolate. *X_core = 1* is a spherical core.

For a fixed shell thickness *XpolarShell = 1*, to scale the shell thickness pro-rata with the radius set or constrain *XpolarShell = X_core*.

When including an $S(q)$, the radius in $S(q)$ is calculated to be that of a sphere with the same 2nd virial coefficient of the outer surface of the ellipsoid. This may have some undesirable effects if the aspect ratio of the ellipsoid is large (ie, if $X << 1$ or $X >> 1$), when the $S(q)$ - which assumes spheres - will not in any case be valid. Generating a custom product model will enable separate effective volume fraction and effective radius in the $S(q)$.

If SAS data are in absolute units, and the SLDs are correct, then scale should be the total volume fraction of the "outer particle". When $S(q)$ is introduced this moves to the $S(q)$ volume fraction, and scale should then be 1.0, or contain some other units conversion factor (for example, if you have SAXS data).

The calculation of intensity follows that for the solid ellipsoid, but with separate terms for the core-shell and shell-solvent boundaries.

$$P(q,\alpha) = \frac{\text{scale}}{V} F^2(q,\alpha) + \text{background}$$

where

$$\begin{aligned}F(q,\alpha) = &f(q, radius\_equat\_core, radius\_equat\_core.x\_core, \alpha)\\ &+ f(q, radius\_equat\_core + thick\_shell, radius\_equat\_core.x\_core + thick\_shell.x\_polar\_shell, \alpha)\end{aligned}$$

where

$$f(q, R_e, R_p, \alpha) = \frac{3\Delta\rho V(\sin[qr(R_p,R_e,\alpha)] - \cos[qr(R_p,R_e,\alpha)])}{[qr(R_p,R_e,\alpha)]^3}$$

and

$$r(R_e,R_p,\alpha) = \left[R_e^2 \sin^2\alpha + R_p^2\cos^2\alpha\right]^{1/2}$$

α is the angle between the axis of the ellipsoid and $\vec{q}$, $V = (4/3)\pi R_p R_e^2$ is the volume of the ellipsoid , R_p is the polar radius along the rotational axis of the ellipsoid, R_e is the equatorial radius perpendicular to the rotational axis of the ellipsoid and $\Delta\rho$ (contrast) is the scattering length density difference, either $(sld\_core - sld\_shell)$ or $(sld\_shell - sld\_solvent)$.

For randomly oriented particles:

$$F^2(q) = \int_0^{\pi/2} F^2(q,\alpha)\sin(\alpha)d\alpha$$

For oriented ellipsoids the *theta*, *phi* and *psi* orientation parameters will appear when fitting 2D data, see the *elliptical_cylinder* model for further information.

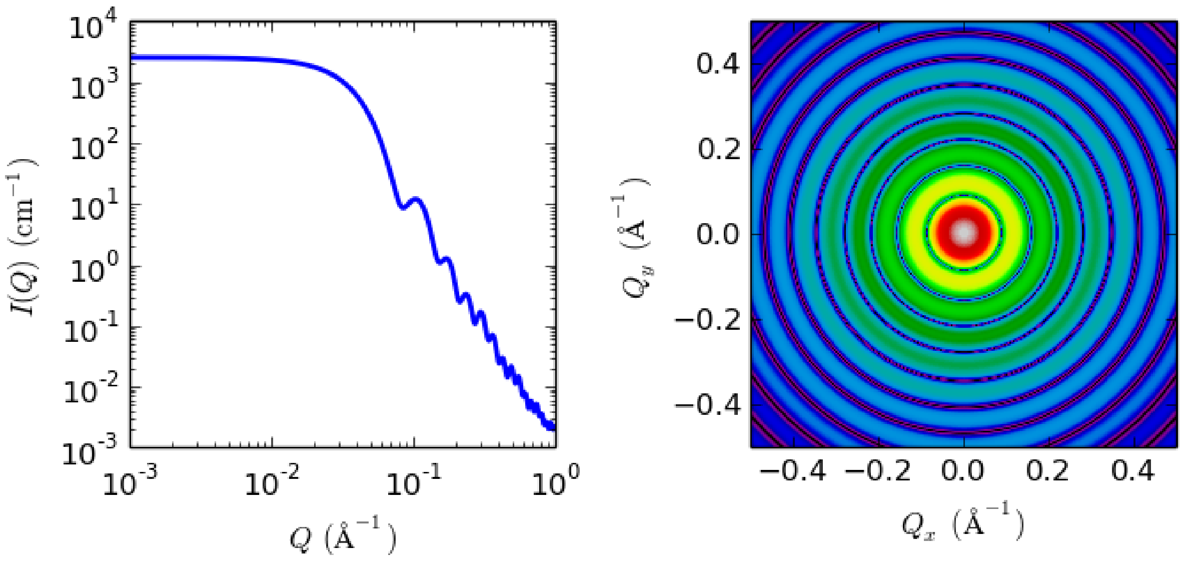


Fig. 1.39: 1D and 2D plots corresponding to the default parameters of the model.

References see for example: Kotlarchyk, M.; Chen, S.-H. J. Chem. Phys., 1983, 79, 2461. Berr, S. J. Phys. Chem., 1987, 91, 4760.

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Last Modified by:** Richard Heenan (reparametrised model) **Date:** 2015
- **Last Reviewed by:** Richard Heenan **Date:** October 6, 2016

ellipsoid

Ellipsoid of revolution with uniform scattering length density.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
sld	Ellipsoid scattering length density	10^{-6}Å^{-2}	4
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	1
radius_polar	Polar radius	Å	20
radius_equatorial	Equatorial radius	Å	400
theta	ellipsoid axis to beam angle	degree	60
phi	rotation about beam	degree	60

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

The form factor is normalized by the particle volume

Definition

The output of the 2D scattering intensity function for oriented ellipsoids is given by (Feigin, 1987)

$$P(q,\alpha) = \frac{\text{scale}}{V} F^2(q,\alpha) + \text{background}$$

where

$$F(q,\alpha) = \Delta\rho V \frac{3(\sin qr - qr\cos qr)}{(qr)^3}$$

for

$$r = \left[R_e^2 \sin^2\alpha + R_p^2 \cos^2\alpha\right]^{1/2}$$

α is the angle between the axis of the ellipsoid and $\vec{q}$, $V = (4/3)\pi R_p R_e^2$ is the volume of the ellipsoid, R_p is the polar radius along the rotational axis of the ellipsoid, R_e is the equatorial radius perpendicular to the rotational axis of the ellipsoid and $\Delta\rho$ (contrast) is the scattering length density difference between the scatterer and the solvent.

For randomly oriented particles use the orientational average,

$$\langle F^2(q)\rangle = \int_0^{\pi/2} F^2(q,\alpha)\sin(\alpha)\,d\alpha$$

computed via substitution of $u = \sin(\alpha)$, $du = \cos(\alpha)\,d\alpha$ as

$$\langle F^2(q)\rangle = \int_0^1 F^2(q,u)\,du$$

with

$$r = R_e\left[1 + u^2\left(R_p^2/R_e^2 - 1\right)\right]^{1/2}$$

For 2d data from oriented ellipsoids the direction of the rotation axis of the ellipsoid is defined using two angles θ and ϕ as for the *cylinder orientation figure*. For the ellipsoid, θ is the angle between the rotational axis and the z -axis in the xz plane followed by a rotation by ϕ in the xy plane, for further details of the calculation and angular dispersions see *Oriented particles* .

NB: The 2nd virial coefficient of the solid ellipsoid is calculated based on the R_p and R_e values, and used as the effective radius for $S(q)$ when $P(q)\cdot S(q)$ is applied.

The θ and ϕ parameters are not used for the 1D output.

Validation

Validation of the code was done by comparing the output of the 1D model to the output of the software provided by the NIST (Kline, 2006).

The implementation of the intensity for fully oriented ellipsoids was validated by averaging the 2D output using a uniform distribution $p(\theta,\phi) = 1.0$ and comparing with the output of the 1D calculation.

The discrepancy above q = 0.3 cm^{-1} is due to the way the form factors are calculated in the c-library provided by NIST. A numerical integration has to be performed to obtain $P(q)$ for randomly oriented particles. The NIST software performs that integration with a 76-point Gaussian quadrature rule, which will become imprecise at high q where the amplitude varies quickly as a function of q. The SasView result shown has been obtained by summing over 501 equidistant points. Our result was found to be stable over the range of q shown for a number of points higher than 500.

Model was also tested against the triaxial ellipsoid model with equal major and minor equatorial radii. It is also consistent with the cyclinder model with polar radius equal to length and equatorial radius equal to radius.

References

L A Feigin and D I Svergun. *Structure Analysis by Small-Angle X-Ray and Neutron Scattering*, Plenum Press, New York, 1987.

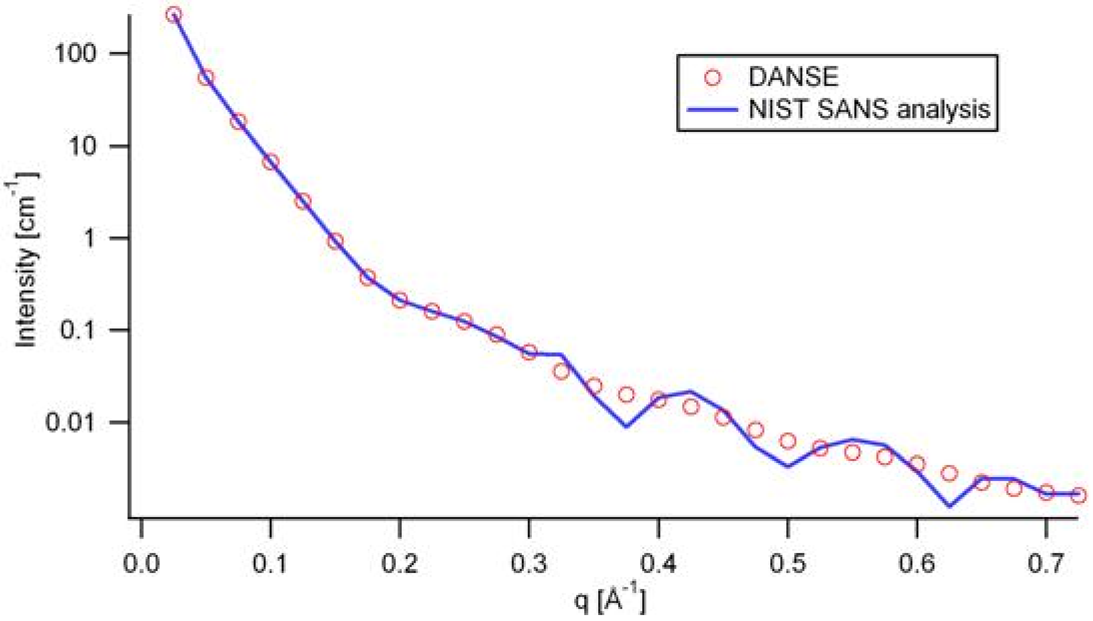


Fig. 1.40: Comparison of the intensity for uniformly distributed ellipsoids calculated from our 2D model and the intensity from the NIST SANS analysis software. The parameters used were: *scale* = 1.0, *radius_polar* = 20 Å, *radius_equatorial* = 400 Å, *contrast* = 3e-6 Å^{-2}, and *background* = 0.0 cm^{-1}.

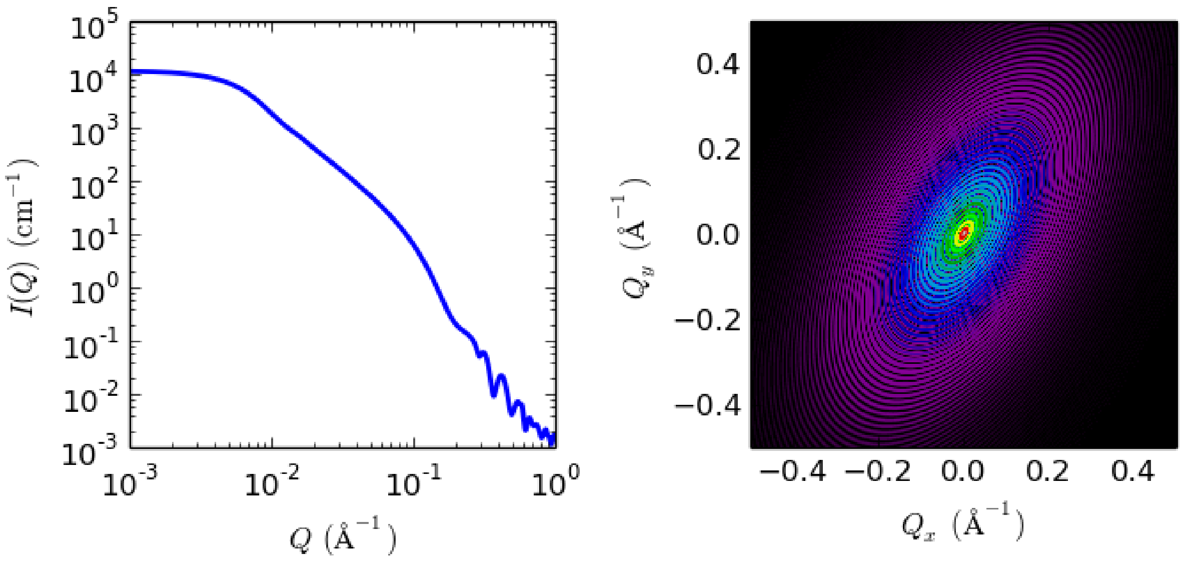


Fig. 1.41: 1D and 2D plots corresponding to the default parameters of the model.

1. Isihara. J. Chem. Phys. 18(1950) 1446-1449

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Converted to sasmodels by:** Helen Park **Date:** July 9, 2014
- **Last Modified by:** Paul Kienzle **Date:** March 22, 2017

triaxial_ellipsoid

Ellipsoid of uniform scattering length density with three independent axes.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
sld	Ellipsoid scattering length density	10^{-6}Å^{-2}	4
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	1
radius_equat_minor	Minor equatorial radius, Ra	Å	20
radius_equat_major	Major equatorial radius, Rb	Å	400
radius_polar	Polar radius, Rc	Å	10
theta	polar axis to beam angle	degree	60
phi	rotation about beam	degree	60
psi	rotation about polar axis	degree	60

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

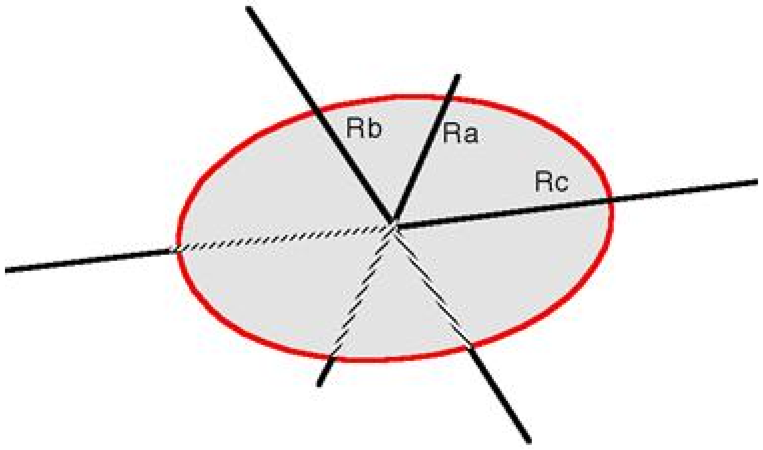


Fig. 1.42: Ellipsoid with R_a as *radius_equat_minor*, R_b as *radius_equat_major* and R_c as *radius_polar*.

Given an ellipsoid

$$\frac{X^2}{R_a^2}+\frac{Y^2}{R_b^2}+\frac{Z^2}{R_c^2}=1$$

the scattering for randomly oriented particles is defined by the average over all orientations Ω of:

$$P(q)=\text{scale}(\Delta\rho)^2\frac{V}{4\pi}\int_\Omega\Phi^2(qr)\,d\Omega+\text{background}$$

where

$$\begin{aligned}\Phi(qr)&=3j_1(qr)/qr=3(\sin qr-qr\cos qr)/(qr)^3\\ r^2&=R_a^2e^2+R_b^2f^2+R_c^2g^2\\ V&=\tfrac{4}{3}\pi R_aR_bR_c\end{aligned}$$

The e, f and g terms are the projections of the orientation vector on X, Y and Z respectively. Keeping the orientation fixed at the canonical axes, we can integrate over the incident direction using polar angle $-\pi/2 \le \gamma \le \pi/2$ and equatorial angle $0 \le \phi \le 2\pi$ (as defined in ref [1]),

$$\langle \Phi^2 \rangle = \int_0^{2\pi} \int_{-\pi/2}^{\pi/2} \Phi^2(qr) \cos\gamma \, d\gamma d\phi$$

with $e = \cos\gamma \sin\phi$, $f = \cos\gamma \cos\phi$ and $g = \sin\gamma$. A little algebra yields

$$r^2 = b^2(p_a \sin^2\phi \cos^2\gamma + 1 + p_c \sin^2\gamma)$$

for

$$p_a = \frac{a^2}{b^2} - 1 \text{ and } p_c = \frac{c^2}{b^2} - 1$$

Due to symmetry, the ranges can be restricted to a single quadrant $0 \le \gamma \le \pi/2$ and $0 \le \phi \le \pi/2$, scaling the resulting integral by 8. The computation is done using the substitution $u = \sin\gamma$, $du = \cos\gamma \, d\gamma$, giving

$$\langle \Phi^2 \rangle = 8 \int_0^{\pi/2} \int_0^1 \Phi^2(qr) du d\phi$$
$$r^2 = b^2(p_a \sin^2(\phi)(1 - u^2) + 1 + p_c u^2)$$

Though for convenience we describe the three radii of the ellipsoid as equatorial and polar, they may be given in *any* size order. To avoid multiple solutions, especially with Monte-Carlo fit methods, it may be advisable to restrict their ranges. For typical small angle diffraction situations there may be a number of closely similar "best fits", so some trial and error, or fixing of some radii at expected values, may help.

To provide easy access to the orientation of the triaxial ellipsoid, we define the axis of the cylinder using the angles θ, ϕ and ψ. These angles are defined analogously to the elliptical_cylinder below, note that angle ϕ is now NOT the same as in the equations above.

For oriented ellipsoids the *theta*, *phi* and *psi* orientation parameters will appear when fitting 2D data, see the *elliptical_cylinder* model for further information.

The radius-of-gyration for this system is $R_g^2 = (R_a R_b R_c)^2/5$.

The contrast $\Delta\rho$ is defined as SLD(ellipsoid) - SLD(solvent). In the parameters, R_a is the minor equatorial radius, R_b is the major equatorial radius, and R_c is the polar radius of the ellipsoid.

NB: The 2nd virial coefficient of the triaxial solid ellipsoid is calculated after sorting the three radii to give the most appropriate prolate or oblate form, from the new polar radius $R_p = R_c$ and effective equatorial radius, $R_e = \sqrt{R_a R_b}$, to then be used as the effective radius for $S(q)$ when $P(q) \cdot S(q)$ is applied.

Validation

Validation of our code was done by comparing the output of the 1D calculation to the angular average of the output of 2D calculation over all possible angles.

References

[1] Finnigan, J.A., Jacobs, D.J., 1971. *Light scattering by ellipsoidal particles in solution*, J. Phys. D: Appl. Phys. 4, 72-77. doi:10.1088/0022-3727/4/1/310

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Last Modified by:** Paul Kienzle (improved calculation) **Date:** April 4, 2017
- **Last Reviewed by:** Paul Kienzle & Richard Heenan **Date:** April 4, 2017

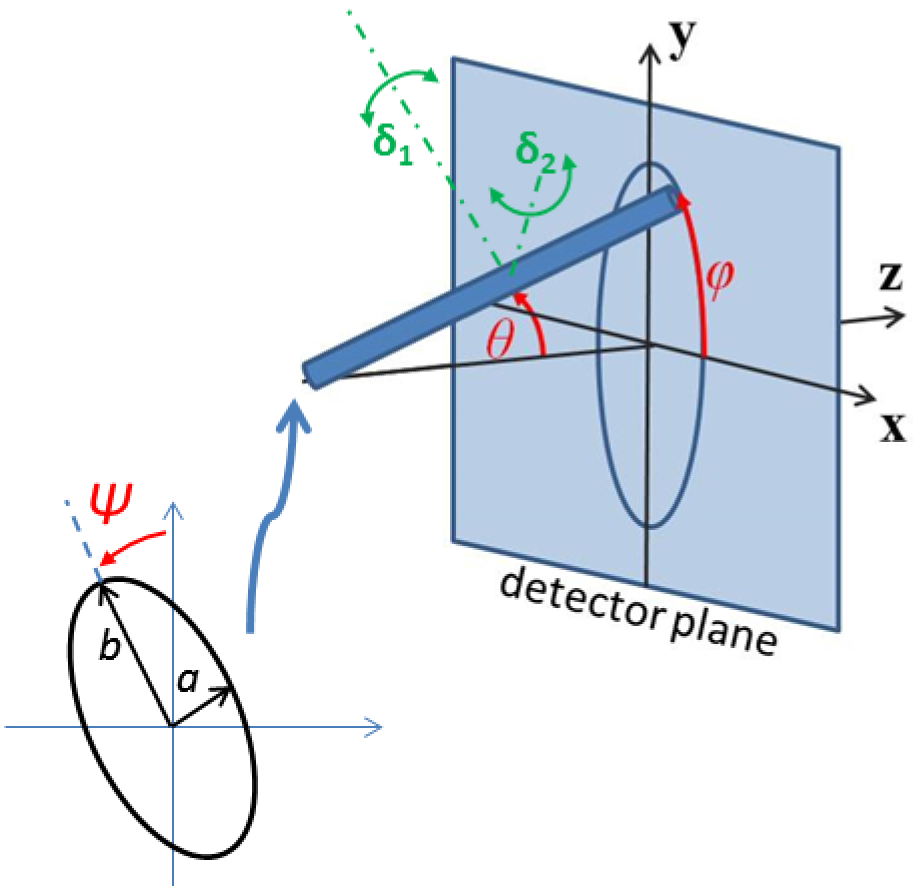


Fig. 1.43: Definition of angles for oriented triaxial ellipsoid, where radii are for illustration here $a < b << c$ and angle Ψ is a rotation around the axis of the particle.

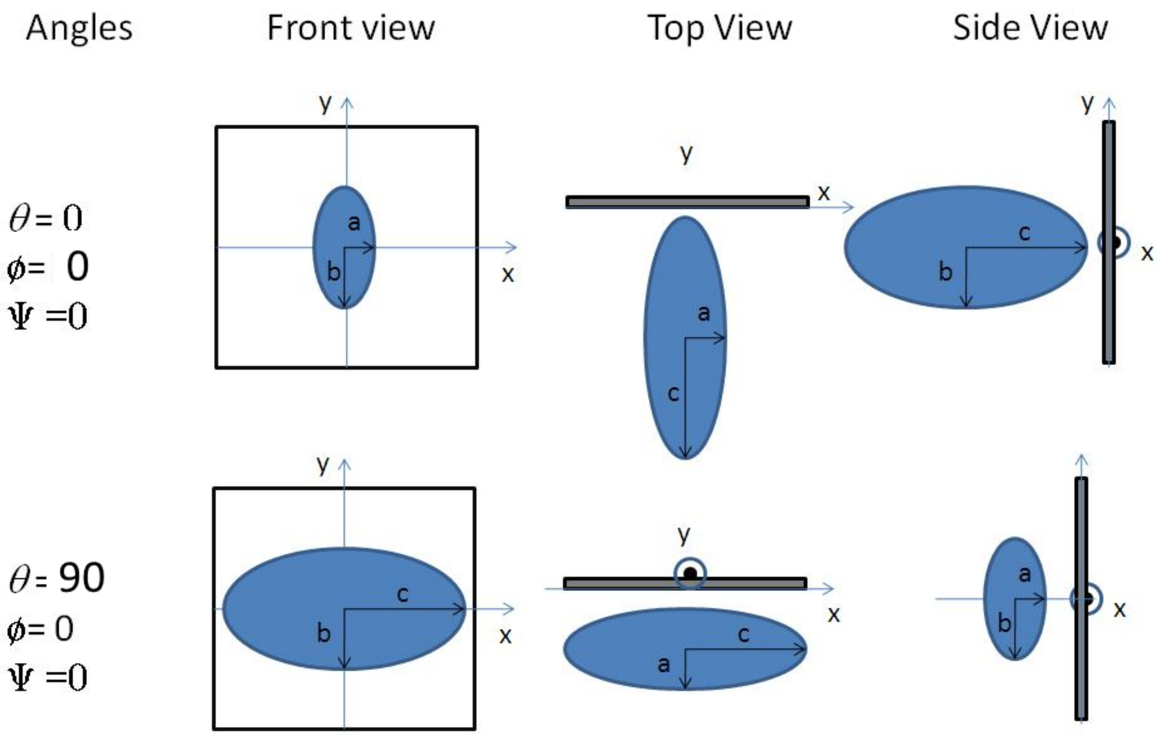


Fig. 1.44: Some examples for an oriented triaxial ellipsoid.

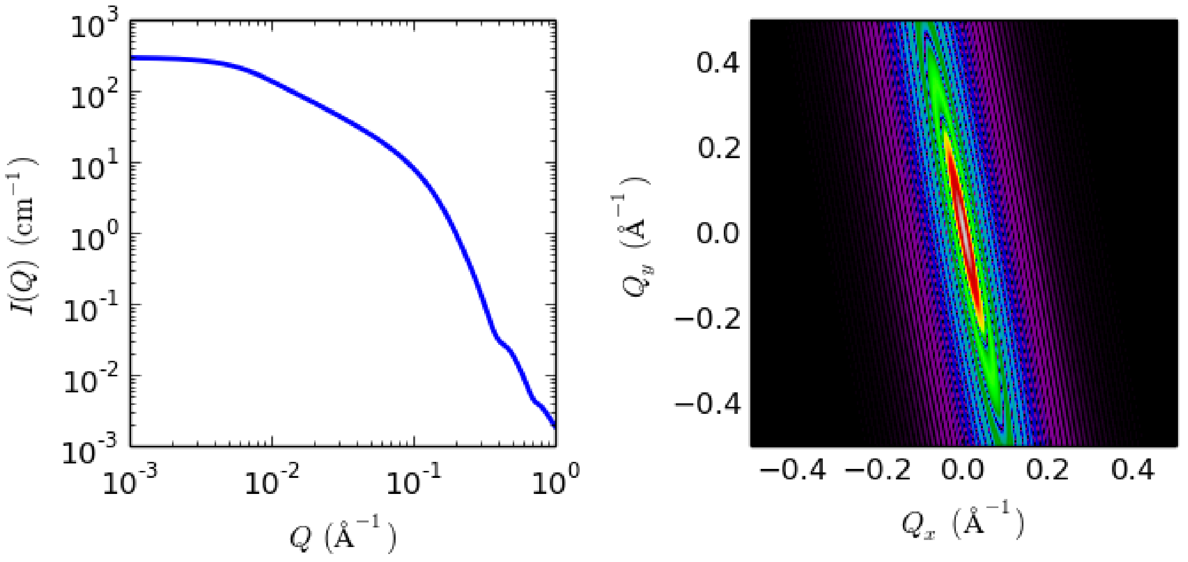


Fig. 1.45: 1D and 2D plots corresponding to the default parameters of the model.

1.1.3 Lamellae Functions

lamellar

Lyotropic lamellar phase with uniform SLD and random distribution

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
thickness	total layer thickness	Å	50
sld	Layer scattering length density	$10^{-6}Å^{-2}$	1
sld_solvent	Solvent scattering length density	$10^{-6}Å^{-2}$	6

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Polydispersity in the bilayer thickness can be applied from the GUI.

Definition

The scattering intensity $I(q)$ for dilute, randomly oriented, "infinitely large" sheets or lamellae is

$$I(q) = \text{scale}\frac{2\pi P(q)}{q^2\delta} + \text{background}$$

The form factor is

$$P(q) = \frac{2\Delta\rho^2}{q^2}(1 - \cos(q\delta)) = \frac{4\Delta\rho^2}{q^2}\sin^2\left(\frac{q\delta}{2}\right)$$

where δ is the total layer thickness and $\Delta\rho$ is the scattering length density difference.

This is the limiting form for a spherical shell of infinitely large radius. Note that the division by δ means that $scale$ in sasview is the volume fraction of sheet, $\phi = S\delta$ where S is the area of sheet per unit volume. S is half the Porod surface area per unit volume of a thicker layer (as that would include both faces of the sheet).

The 2D scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

References

F Nallet, R Laversanne, and D Roux, J. Phys. II France, 3, (1993) 487-502

also in J. Phys. Chem. B, 105, (2001) 11081-11088

lamellar_hg

Random lamellar phase with Head and Tail Groups

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
length_tail	Tail thickness (total = H+T+T+H)	Å	15
length_head	Head thickness	Å	10
sld	Tail scattering length density	$10^{-6}Å^{-2}$	0.4
sld_head	Head scattering length density	$10^{-6}Å^{-2}$	3
sld_solvent	Solvent scattering length density	$10^{-6}Å^{-2}$	6

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

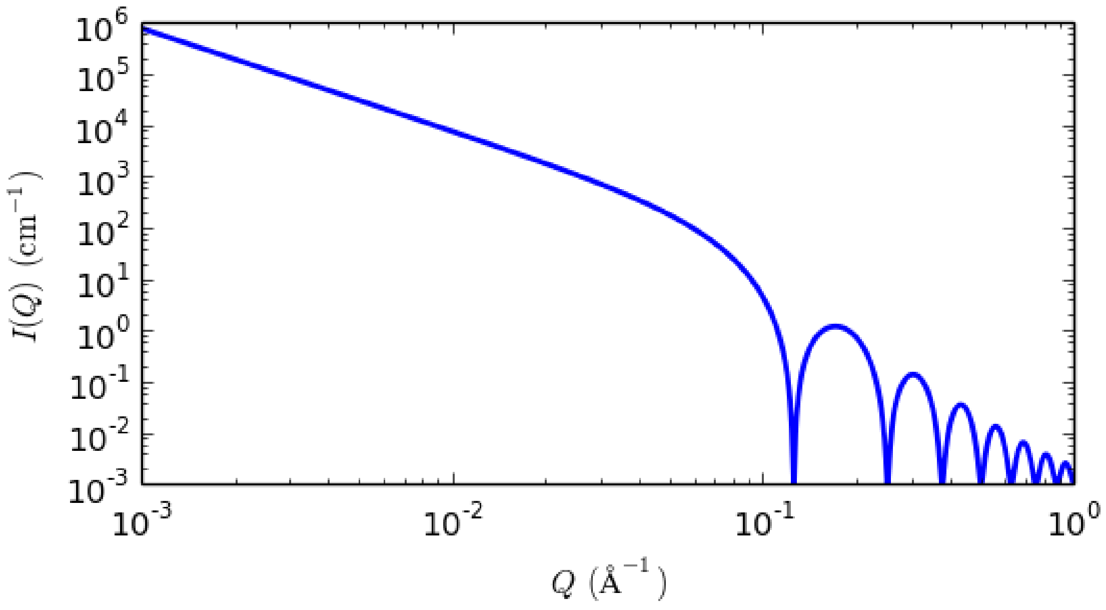


Fig. 1.46: 1D plot corresponding to the default parameters of the model.

This model provides the scattering intensity, $I(q)$, for a lyotropic lamellar phase where a random distribution in solution are assumed. The SLD of the head region is taken to be different from the SLD of the tail region.

Definition

The scattering intensity $I(q)$ is

$$I(q) = 2\pi \frac{\text{scale}}{2(\delta_H + \delta_T)} P(q) \frac{1}{q^2}$$

The form factor $P(q)$ is

$$P(q) = \frac{4}{q^2} \left\{ \Delta\rho_H \left[\sin[q(\delta_H + \delta_T) \ - \sin(q\delta_T)\right] + \Delta\rho_T \sin(q\delta_T) \right\}^2$$

where δ_T is *length_tail*, δ_H is *length_head*, $\Delta\rho_H$ is the head contrast (*sld_head* $-$ *sld_solvent*), and $\Delta\rho_T$ is tail contrast (*sld* $-$ *sld_solvent*).

The total thickness of the lamellar sheet is $\delta_H + \delta_T + \delta_T + \delta_H$. Note that in a non aqueous solvent the chemical "head" group may be the "Tail region" and vice-versa.

The 2D scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

References

F Nallet, R Laversanne, and D Roux, J. Phys. II France, 3, (1993) 487-502

also in J. Phys. Chem. B, 105, (2001) 11081-11088

2014/04/17 - Description reviewed by S King and P Butler.

lamellar_hg_stack_caille

Random lamellar head/tail/tail/head sheet with Caille structure factor

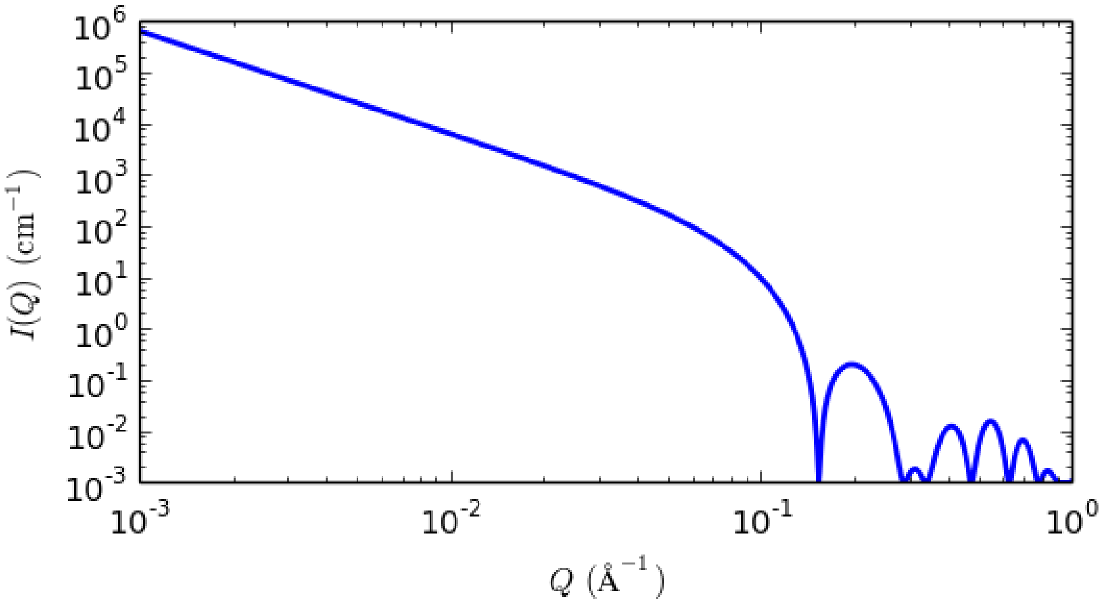


Fig. 1.47: 1D plot corresponding to the default parameters of the model.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
length_tail	Tail thickness	Å	10
length_head	head thickness	Å	2
Nlayers	Number of layers	None	30
d_spacing	lamellar d-spacing of Caille S(Q)	Å	40
Caille_parameter	Caille parameter	None	0.001
sld	Tail scattering length density	10^{-6}Å^{-2}	0.4
sld_head	Head scattering length density	10^{-6}Å^{-2}	2
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	6

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

This model provides the scattering intensity, $I(q) = P(q)S(q)$, for a lamellar phase where a random distribution in solution are assumed. Here a Caille $S(q)$ is used for the lamellar stacks.

The scattering intensity $I(q)$ is

$$I(q) = 2\pi\frac{P(q)S(q)}{q^2\delta}$$

The form factor $P(q)$ is

$$P(q) = \frac{4}{q^2}\left\{\Delta\rho_H\left[\sin[q(\delta_H+\delta_T)] - \sin(q\delta_T)\right] + \Delta\rho_T\sin(q\delta_T)\right\}^2$$

and the structure factor $S(q)$ is

$$S(q) = 1 + 2\sum_1^{N-1}\left(1-\frac{n}{N}\right)\cos(qdn)\exp\left(-\frac{2q^2d^2\alpha(n)}{2}\right)$$

where

$$\begin{aligned}
\alpha(n) &= \frac{\eta_{cp}}{4\pi^2}\left(\ln(\pi n) + \gamma_E\right) \\
\gamma_E &= 0.5772156649 && \text{Euler's constant} \\
\eta_{cp} &= \frac{q_o^2 k_B T}{8\pi\sqrt{KB}} && \text{Caille constant}
\end{aligned}$$

δ_T is the tail length (or *length_tail*), δ_H is the head thickness (or *length_head*), $\Delta\rho_H$ is SLD(headgroup) - SLD(solvent), and $\Delta\rho_T$ is SLD(tail) - SLD(headgroup). Here d is (repeat) spacing, K is smectic bending elasticity, B is compression modulus, and N is the number of lamellar plates (*Nlayers*).

NB: **When the Caille parameter is greater than approximately 0.8 to 1.0, the assumptions of the model are incorrect.** And due to a complication of the model function, users are responsible for making sure that all the assumptions are handled accurately (see the original reference below for more details).

Non-integer numbers of stacks are calculated as a linear combination of results for the next lower and higher values.

Be aware that the computations may be very slow.

The 2D scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

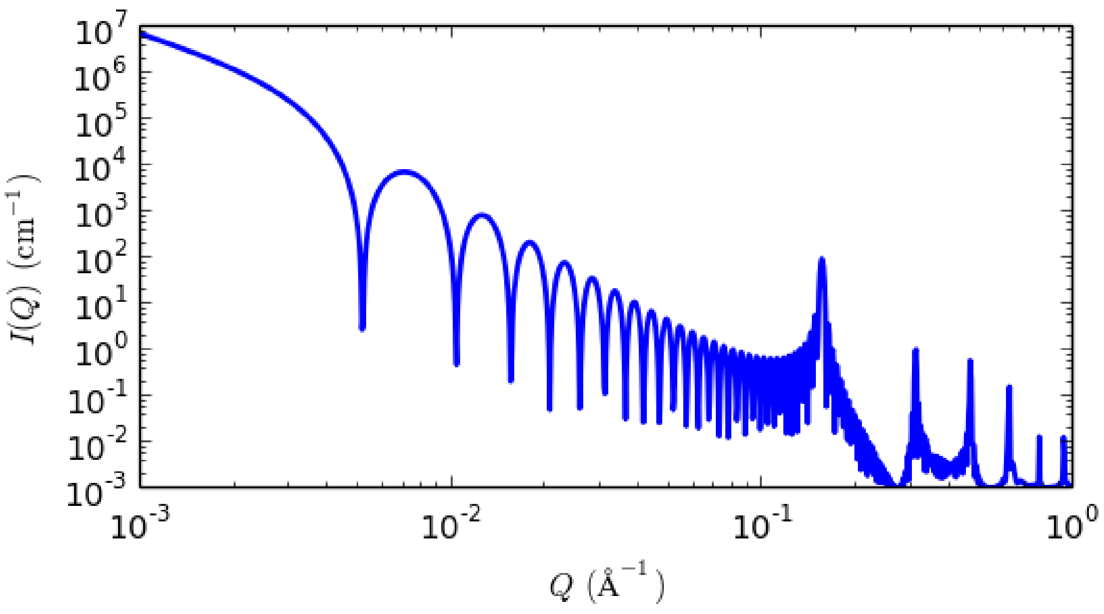


Fig. 1.48: 1D plot corresponding to the default parameters of the model.

References

F Nallet, R Laversanne, and D Roux, J. Phys. II France, 3, (1993) 487-502

also in J. Phys. Chem. B, 105, (2001) 11081-11088

lamellar_stack_caille

Random lamellar sheet with Caille structure factor

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
thickness	sheet thickness	Å	30
Nlayers	Number of layers	None	20
d_spacing	lamellar d-spacing of Caille S(Q)	Å	400
Caille_parameter	Caille parameter	Å^{-2}	0.1
sld	layer scattering length density	10^{-6}Å^{-2}	6.3
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	1

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

This model provides the scattering intensity, $I(q) = P(q)S(q)$, for a lamellar phase where a random distribution in solution are assumed. Here a Caille $S(q)$ is used for the lamellar stacks.

Definition

The scattering intensity $I(q)$ is

$$I(q) = 2\pi \frac{P(q)S(q)}{q^2\delta}$$

The form factor is

$$P(q) = \frac{2\Delta\rho^2}{q^2}(1 - \cos q\delta)$$

and the structure factor is

$$S(q) = 1 + 2\sum_{1}^{N-1}\left(1 - \frac{n}{N}\right)\cos(qdn)\exp\left(-\frac{2q^2d^2\alpha(n)}{2}\right)$$

where

$$\begin{aligned} \alpha(n) &= \frac{\eta_{cp}}{4\pi^2}\left(\ln(\pi n) + \gamma_E\right) & \\ \gamma_E &= 0.5772156649 & \text{Euler's constant} \\ \eta_{cp} &= \frac{q_o^2 k_B T}{8\pi\sqrt{K\overline{B}}} & \text{Caille constant} \end{aligned}$$

Here d = (repeat) d_spacing, δ = bilayer thickness, the contrast $\Delta\rho$ = SLD(headgroup) - SLD(solvent), K = smectic bending elasticity, B = compression modulus, and N = number of lamellar plates (*n_plates*).

NB: **When the Caille parameter is greater than approximately 0.8 to 1.0, the assumptions of the model are incorrect.** And due to a complication of the model function, users are responsible for making sure that all the assumptions are handled accurately (see the original reference below for more details).

Non-integer numbers of stacks are calculated as a linear combination of results for the next lower and higher values.

The 2D scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

References

F Nallet, R Laversanne, and D Roux, J. Phys. II France, 3, (1993) 487-502

also in J. Phys. Chem. B, 105, (2001) 11081-11088

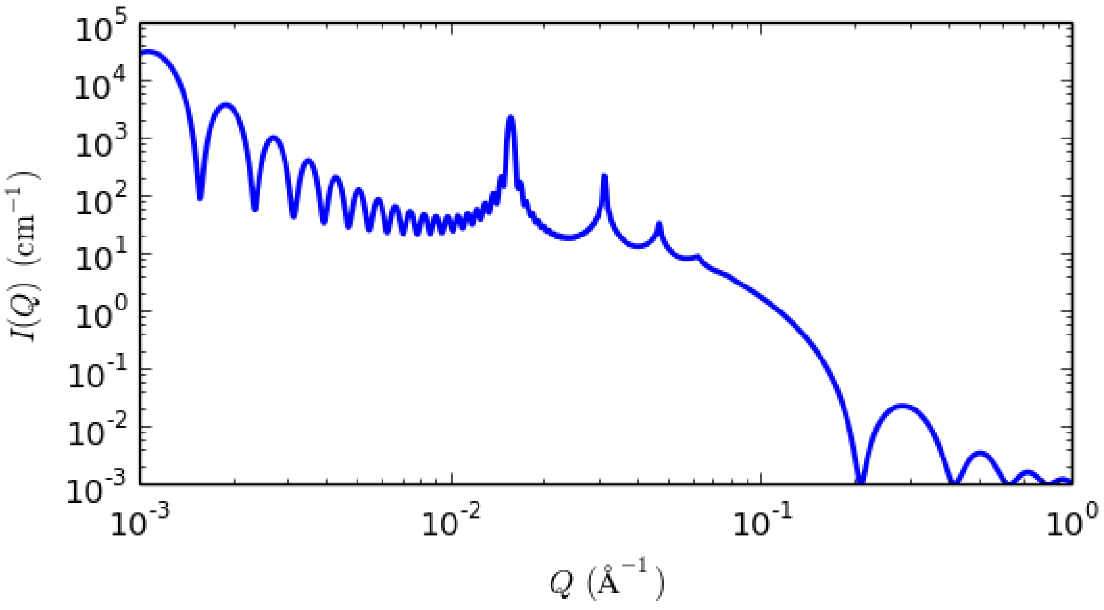


Fig. 1.49: 1D plot corresponding to the default parameters of the model.

lamellar_stack_paracrystal

Random lamellar sheet with paracrystal structure factor

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
thickness	sheet thickness	Å	33
Nlayers	Number of layers	None	20
d_spacing	lamellar spacing of paracrystal stack	Å	250
sigma_d	Sigma (polydispersity) of the lamellar spacing	Å	0
sld	layer scattering length density	10^{-6}Å^{-2}	1
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	6.34

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

This model calculates the scattering from a stack of repeating lamellar structures. The stacks of lamellae (infinite in lateral dimension) are treated as a paracrystal to account for the repeating spacing. The repeat distance is further characterized by a Gaussian polydispersity. **This model can be used for large multilamellar vesicles.**

Definition

In the equations below,

- *scale* is used instead of the mass per area of the bilayer Γ_m (this corresponds to the volume fraction of the material in the bilayer, *not* the total excluded volume of the paracrystal),
- *sld* − *sld_solvent* is the contrast $\Delta\rho$,
- *thickness* is the layer thickness t,
- *Nlayers* is the number of layers N,
- *d_spacing* is the average distance between adjacent layers $\langle D\rangle$, and
- *sigma_d* is the relative standard deviation of the Gaussian layer distance distribution $\sigma_D/\langle D\rangle$.

The scattering intensity $I(q)$ is calculated as

$$I(q) = 2\pi\Delta\rho^2\Gamma_m\frac{P_{\text{bil}}(q)}{q^2}Z_N(q)$$

The form factor of the bilayer is approximated as the cross section of an infinite, planar bilayer of thickness t (compare the equations for the lamellar model).

$$P_{\text{bil}}(q) = \left(\frac{\sin(qt/2)}{qt/2}\right)^2$$

$Z_N(q)$ describes the interference effects for aggregates consisting of more than one bilayer. The equations used are (3-5) from the Bergstrom reference:

$$Z_N(q) = \frac{1-w^2}{1+w^2-2w\cos(q\langle D\rangle)} + x_N S_N + (1-x_N)S_{N+1}$$

where

$$S_N(q) = \frac{a_N}{N}[1+w^2-2w\cos(q\langle D\rangle)]^2$$

and

$$\begin{aligned} a_N &= 4w^2 - 2(w^3+w)\cos(q\langle D\rangle) \\ &\quad - 4w^{N+2}\cos(Nq\langle D\rangle) + 2w^{N+3}\cos[(N-1)q\langle D\rangle] + 2w^{N+1}\cos[(N+1)q\langle D\rangle] \end{aligned}$$

for the layer spacing distribution $w = \exp(-\sigma_D^2 q^2/2)$.

Non-integer numbers of stacks are calculated as a linear combination of the lower and higher values

$$N_L = x_N N + (1-x_N)(N+1)$$

The 2D scattering intensity is the same as 1D, regardless of the orientation of the q vector which is defined as

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

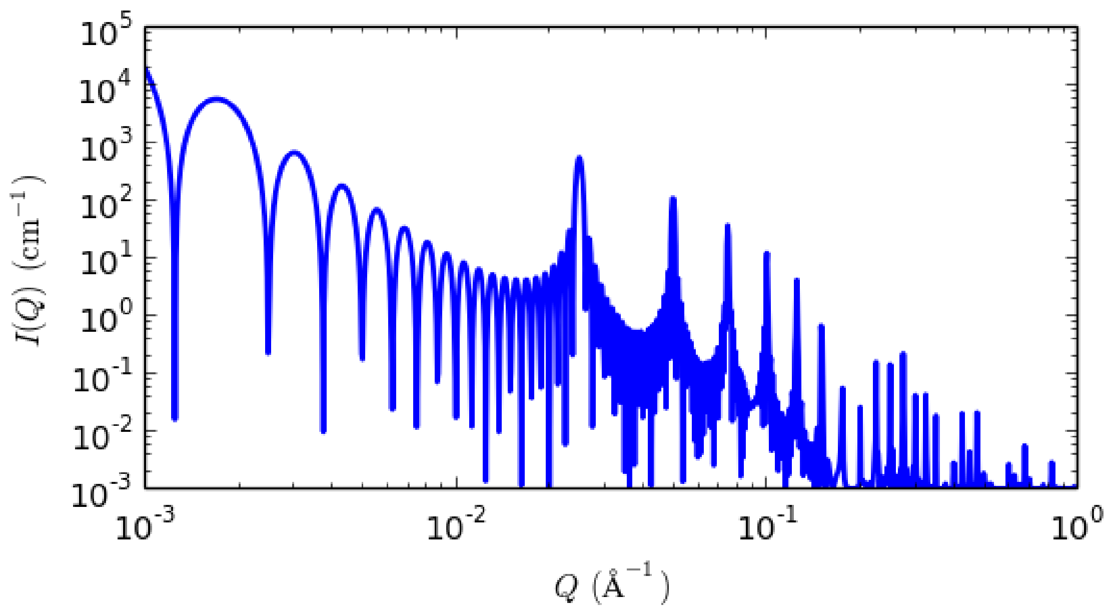


Fig. 1.50: 1D plot corresponding to the default parameters of the model.

Reference

M Bergstrom, J S Pedersen, P Schurtenberger, S U Egelhaaf, *J. Phys. Chem. B*, 103 (1999) 9888-9897

1.1.4 Paracrystal Functions

bcc_paracrystal

Body-centred cubic lattic with paracrystalline distortion

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
dnn	Nearest neighbour distance	Å	220
d_factor	Paracrystal distortion factor	None	0.06
radius	Particle radius	Å	40
sld	Particle scattering length density	10^{-6}Å^{-2}	4
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	1
theta	c axis to beam angle	degree	60
phi	rotation about beam	degree	60
psi	rotation about c axis	degree	60

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Warning: This model and this model description are under review following concerns raised by SasView users. If you need to use this model, please email help@sasview.org for the latest situation. *The SasView Developers. September 2018.*

Definition

Calculates the scattering from a **body-centered cubic lattice** with paracrystalline distortion. Thermal vibrations are considered to be negligible, and the size of the paracrystal is infinitely large. Paracrystalline distortion is assumed to be isotropic and characterized by a Gaussian distribution.

The scattering intensity $I(q)$ is calculated as

$$I(q) = \frac{\text{scale}}{V_p} V_{\text{lattice}} P(q) Z(q)$$

where *scale* is the volume fraction of spheres, V_p is the volume of the primary particle, V_{lattice} is a volume correction for the crystal structure, $P(q)$ is the form factor of the sphere (normalized), and $Z(q)$ is the paracrystalline structure factor for a body-centered cubic structure.

Equation (1) of the 1990 reference[2] is used to calculate $Z(q)$, using equations (29)-(31) from the 1987 paper[1] for $Z1$, $Z2$, and $Z3$.

The lattice correction (the occupied volume of the lattice) for a body-centered cubic structure of particles of radius R and nearest neighbor separation D is

$$V_{\text{lattice}} = \frac{16\pi}{3} \frac{R^3}{\left(D\sqrt{2}\right)^3}$$

The distortion factor (one standard deviation) of the paracrystal is included in the calculation of $Z(q)$

$$\Delta a = gD$$

where g is a fractional distortion based on the nearest neighbor distance.

For a crystal, diffraction peaks appear at reduced q-values given by

$$\frac{qD}{2\pi} = \sqrt{h^2 + k^2 + l^2}$$

[2] Hideki Matsuoka et. al. *Physical Review B*, 41 (1990) 3854 -3856 (Corrections to FCC and BCC lattice structure calculation)

[1] Hideki Matsuoka et. al. *Physical Review B*, 36 (1987) 1754-1765 (Original Paper)

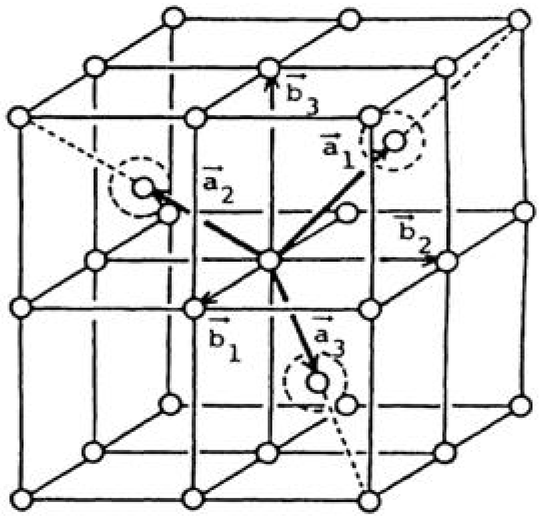


Fig. 1.51: Body-centered cubic lattice.

where for a body-centered cubic lattice, only reflections where $(h + k + l) =$ even are allowed and reflections where $(h + k + l) =$ odd are forbidden. Thus the peak positions correspond to (just the first 5)

q/q_o	1	$\sqrt{2}$	$\sqrt{3}$	$\sqrt{4}$	$\sqrt{5}$
Indices	(110)	(200)	(211)	(220)	(310)

Note: The calculation of $Z(q)$ is a double numerical integral that must be carried out with a high density of points to properly capture the sharp peaks of the paracrystalline scattering. So be warned that the calculation is slow. Fitting of any experimental data must be resolution smeared for any meaningful fit. This makes a triple integral which may be very slow.

This example dataset is produced using 200 data points, *qmin* = 0.001 Å^{-1}, *qmax* = 0.1 Å^{-1} and the above default values.

The 2D (Anisotropic model) is based on the reference below where $I(q)$ is approximated for 1d scattering. Thus the scattering pattern for 2D may not be accurate, particularly at low q. For general details of the calculation and angular dispersions for oriented particles see *Oriented particles* . Note that we are not responsible for any incorrectness of the 2D model computation.

References

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Last Modified by:** Paul Butler **Date:** September 29, 2016
- **Last Reviewed by:** Richard Heenan **Date:** March 21, 2016

fcc_paracrystal

Face-centred cubic lattic with paracrystalline distortion

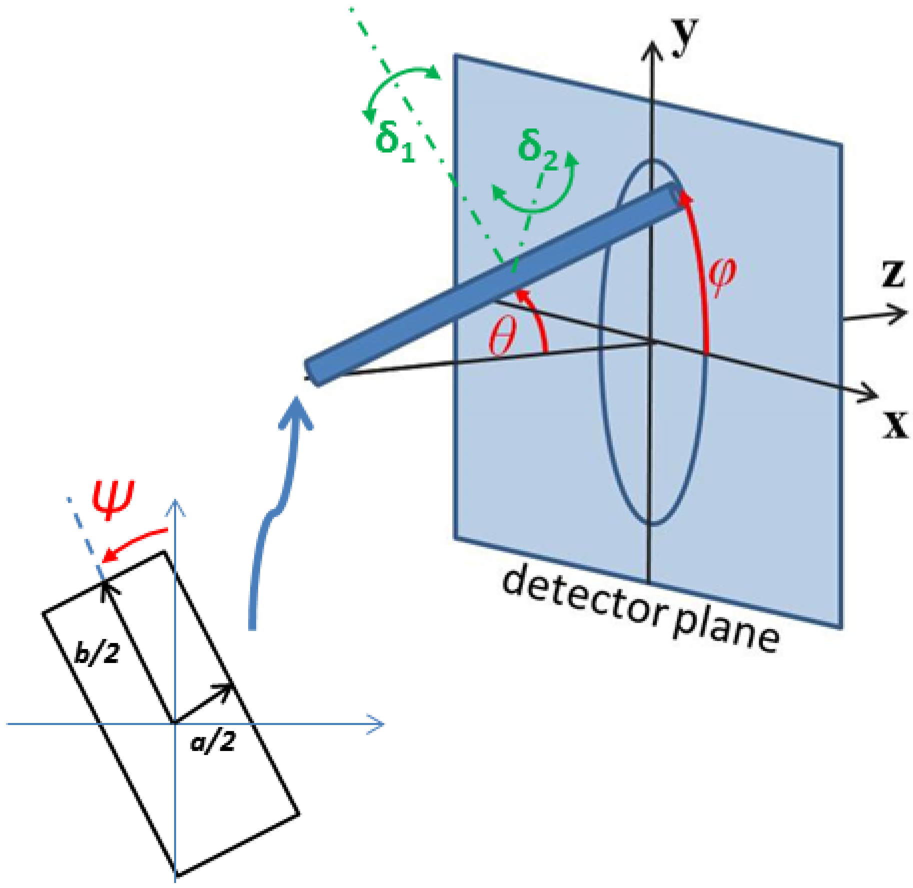


Fig. 1.52: Orientation of the crystal with respect to the scattering plane, when $\theta = \phi = 0$ the c axis is along the beam direction (the z axis).

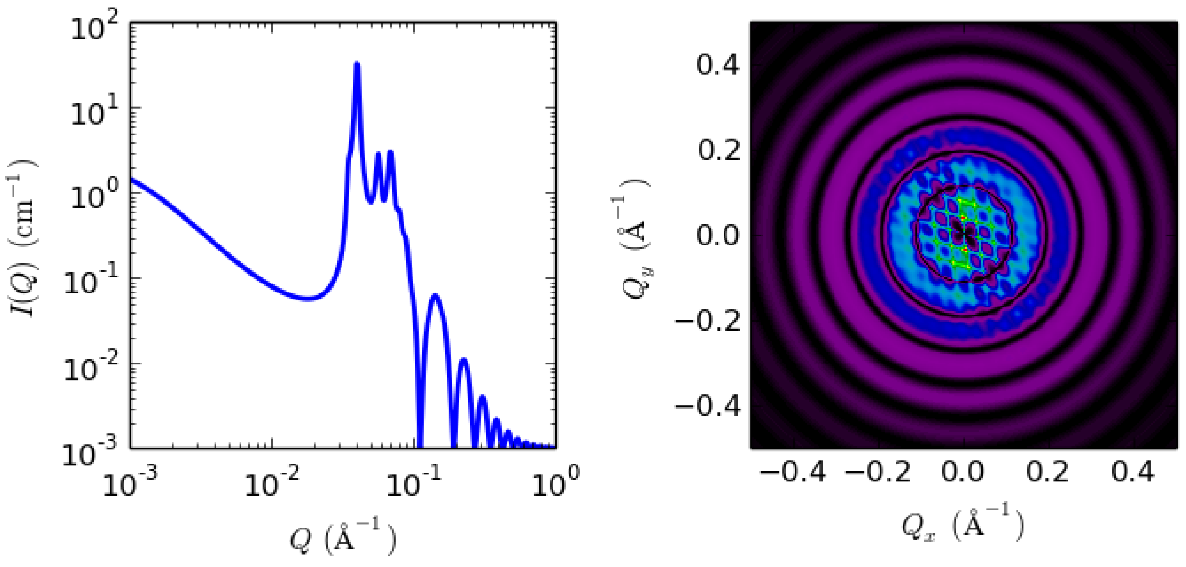


Fig. 1.53: 1D and 2D plots corresponding to the default parameters of the model.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
dnn	Nearest neighbour distance	Å	220
d_factor	Paracrystal distortion factor	None	0.06
radius	Particle radius	Å	40
sld	Particle scattering length density	10^{-6}Å^{-2}	4
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	1
theta	c axis to beam angle	degree	60
phi	rotation about beam	degree	60
psi	rotation about c axis	degree	60

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Warning: This model and this model description are under review following concerns raised by SasView users. If you need to use this model, please email help@sasview.org for the latest situation. *The SasView Developers. September 2018.*

Definition

Calculates the scattering from a **face-centered cubic lattice** with paracrystalline distortion. Thermal vibrations are considered to be negligible, and the size of the paracrystal is infinitely large. Paracrystalline distortion is assumed to be isotropic and characterized by a Gaussian distribution.

The scattering intensity $I(q)$ is calculated as

$$I(q) = \frac{\text{scale}}{V_p} V_{\text{lattice}} P(q) Z(q)$$

where *scale* is the volume fraction of spheres, V_p is the volume of the primary particle, V_{lattice} is a volume correction for the crystal structure, $P(q)$ is the form factor of the sphere (normalized), and $Z(q)$ is the paracrystalline structure factor for a face-centered cubic structure.

Equation (1) of the 1990 reference[2] is used to calculate $Z(q)$, using equations (23)-(25) from the 1987 paper[1] for $Z1$, $Z2$, and $Z3$.

The lattice correction (the occupied volume of the lattice) for a face-centered cubic structure of particles of radius R and nearest neighbor separation D is

$$V_{\text{lattice}} = \frac{16\pi}{3} \frac{R^3}{\left(D\sqrt{2}\right)^3}$$

The distortion factor (one standard deviation) of the paracrystal is included in the calculation of $Z(q)$

$$\Delta a = gD$$

where g is a fractional distortion based on the nearest neighbor distance.

For a crystal, diffraction peaks appear at reduced q-values given by

$$\frac{qD}{2\pi} = \sqrt{h^2 + k^2 + l^2}$$

where for a face-centered cubic lattice h, k, l all odd or all even are allowed and reflections where h, k, l are mixed odd/even are forbidden. Thus the peak positions correspond to (just the first 5)

q/q_0	1	$\sqrt{4/3}$	$\sqrt{8/3}$	$\sqrt{11/3}$	$\sqrt{4}$
Indices	(111)	(200)	(220)	(311)	(222)

[2] Hideki Matsuoka et. al. *Physical Review B*, 41 (1990) 3854 -3856 (Corrections to FCC and BCC lattice structure calculation)

[1] Hideki Matsuoka et. al. *Physical Review B*, 36 (1987) 1754-1765 (Original Paper)

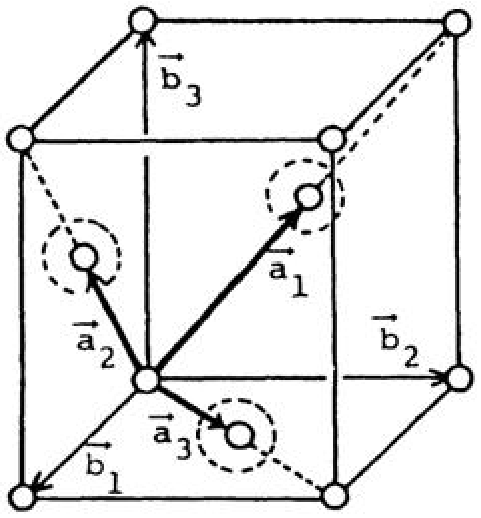


Fig. 1.54: Face-centered cubic lattice.

Note: The calculation of $Z(q)$ is a double numerical integral that must be carried out with a high density of points to properly capture the sharp peaks of the paracrystalline scattering. So be warned that the calculation is slow. Fitting of any experimental data must be resolution smeared for any meaningful fit. This makes a triple integral which may be very slow.

The 2D (Anisotropic model) is based on the reference below where $I(q)$ is approximated for 1d scattering. Thus the scattering pattern for 2D may not be accurate particularly at low q. For general details of the calculation and angular dispersions for oriented particles see *Oriented particles* . Note that we are not responsible for any incorrectness of the 2D model computation.

References

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Last Modified by:** Paul Butler **Date:** September 29, 2016
- **Last Reviewed by:** Richard Heenan **Date:** March 21, 2016

sc_paracrystal

Simple cubic lattice with paracrystalline distortion

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
dnn	Nearest neighbor distance	Å	220
d_factor	Paracrystal distortion factor	None	0.06
radius	Radius of sphere	Å	40
sld	Sphere scattering length density	$10^{-6}Å^{-2}$	3
sld_solvent	Solvent scattering length density	$10^{-6}Å^{-2}$	6.3
theta	c axis to beam angle	degree	0
phi	rotation about beam	degree	0
psi	rotation about c axis	degree	0

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

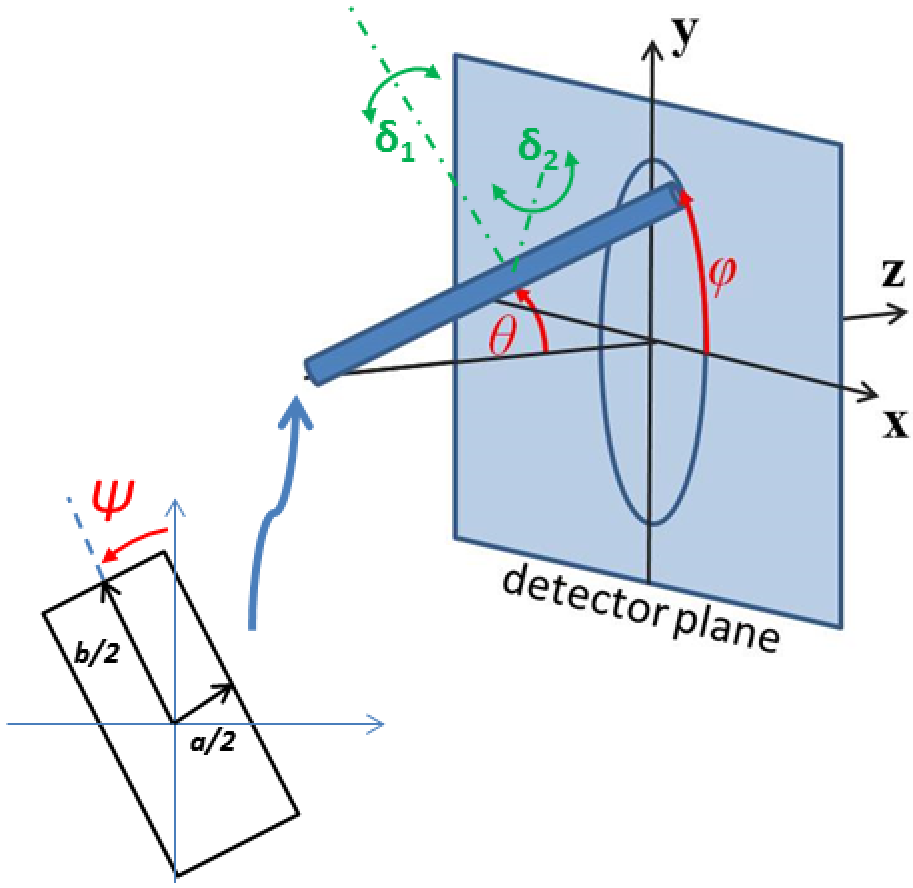


Fig. 1.55: Orientation of the crystal with respect to the scattering plane, when $\theta = \phi = 0$ the c axis is along the beam direction (the z axis).

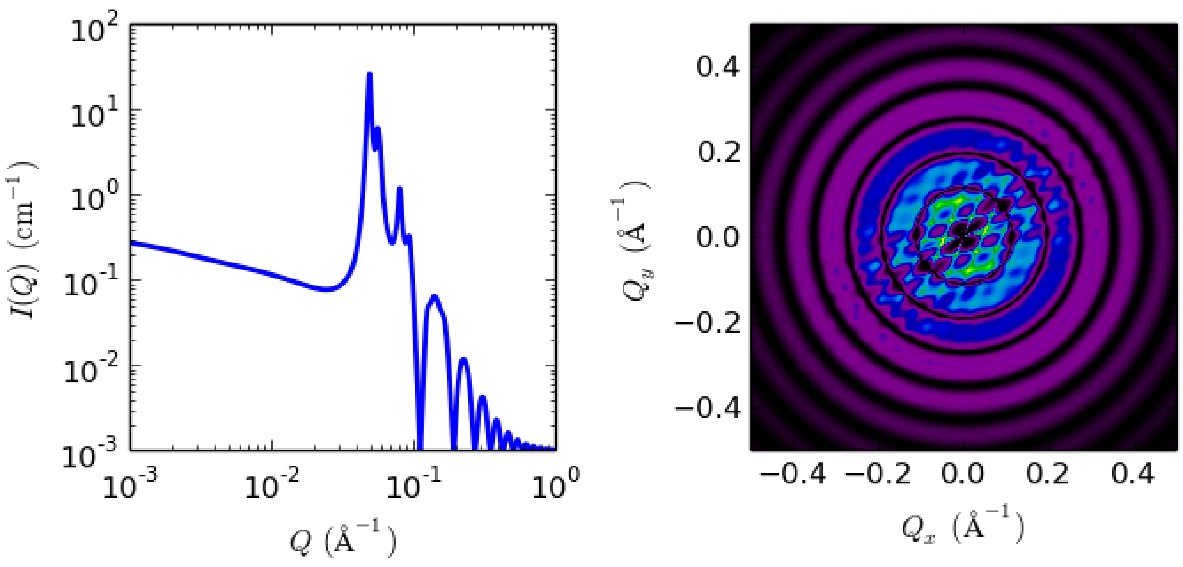


Fig. 1.56: 1D and 2D plots corresponding to the default parameters of the model.

Warning: This model and this model description are under review following concerns raised by SasView users. If you need to use this model, please email help@sasview.org for the latest situation. *The SasView Developers. September 2018.*

Definition

Calculates the scattering from a **simple cubic lattice** with paracrystalline distortion. Thermal vibrations are considered to be negligible, and the size of the paracrystal is infinitely large. Paracrystalline distortion is assumed to be isotropic and characterized by a Gaussian distribution.

The scattering intensity $I(q)$ is calculated as

$$I(q) = \text{scale}\frac{V_\text{lattice} P(q) Z(q)}{V_p} + \text{background}$$

where scale is the volume fraction of spheres, V_p is the volume of the primary particle, V_lattice is a volume correction for the crystal structure, $P(q)$ is the form factor of the sphere (normalized), and $Z(q)$ is the paracrystalline structure factor for a simple cubic structure.

Equation (16) of the 1987 reference[1] is used to calculate $Z(q)$, using equations (13)-(15) from the 1987 paper[1] for $Z1$, $Z2$, and $Z3$.

The lattice correction (the occupied volume of the lattice) for a simple cubic structure of particles of radius *R* and nearest neighbor separation *D* is

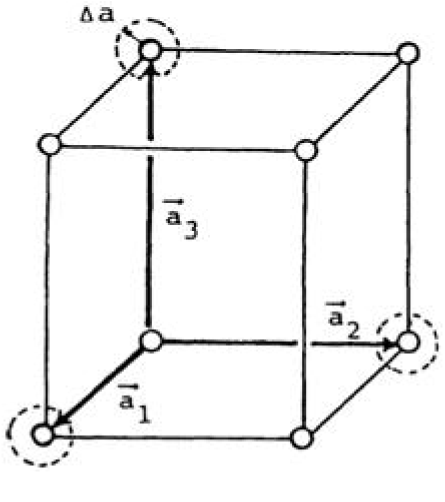
$$V_\text{lattice} = \frac{4\pi}{3}\frac{R^3}{D^3}$$

The distortion factor (one standard deviation) of the paracrystal is included in the calculation of $Z(q)$

$$\Delta a = gD$$

where *g* is a fractional distortion based on the nearest neighbor distance.

The simple cubic lattice is



For a crystal, diffraction peaks appear at reduced q-values given by

$$\frac{qD}{2\pi} = \sqrt{h^2 + k^2 + l^2}$$

where for a simple cubic lattice any h, k, l are allowed and none are forbidden. Thus the peak positions correspond to (just the first 5)

q/q_0	1	$\sqrt{2}$	$\sqrt{3}$	$\sqrt{4}$	$\sqrt{5}$
$Indices$	(100)	(110)	(111)	(200)	(210)

[1] Hideki Matsuoka et. al. *Physical Review B*, 36 (1987) 1754-1765 (Original Paper)

Note: The calculation of *Z(q)* is a double numerical integral that must be carried out with a high density of points to properly capture the sharp peaks of the paracrystalline scattering. So be warned that the calculation is slow. Fitting of any experimental data must be resolution smeared for any meaningful fit. This makes a triple integral which may be very slow.

The 2D (Anisotropic model) is based on the reference below where *I(q)* is approximated for 1d scattering. Thus the scattering pattern for 2D may not be accurate particularly at low q. For general details of the calculation and angular dispersions for oriented particles see *Oriented particles* . Note that we are not responsible for any incorrectness of the 2D model computation.

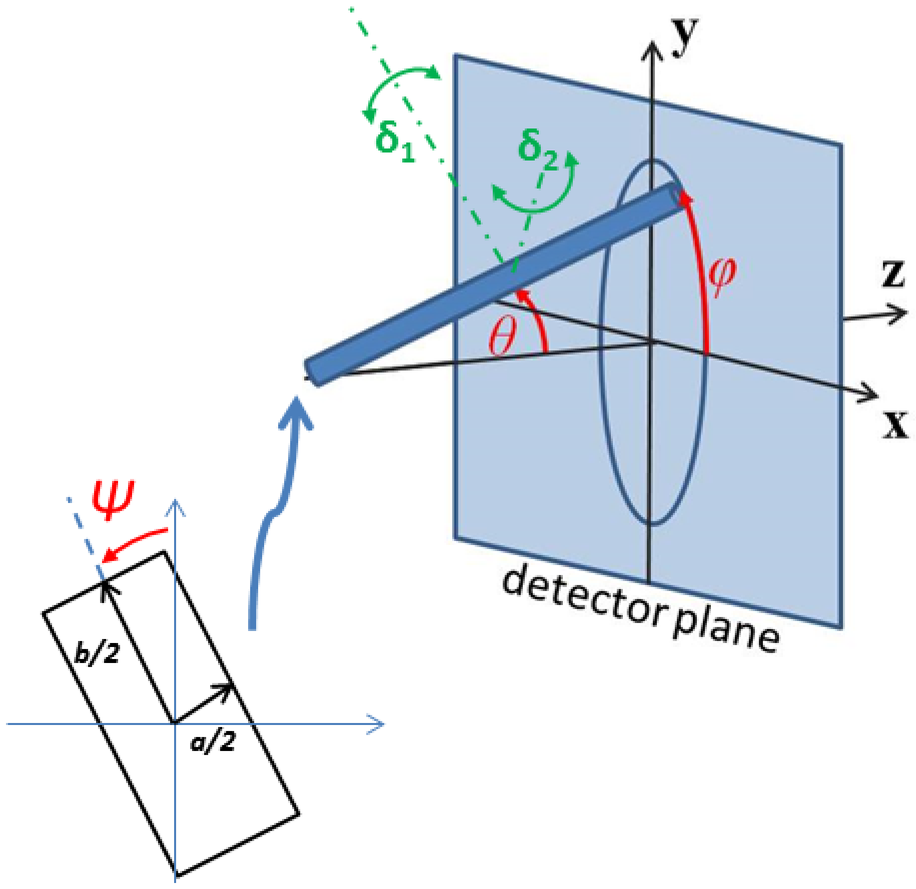


Fig. 1.57: Orientation of the crystal with respect to the scattering plane, when $\theta = \phi = 0$ the c axis is along the beam direction (the z axis).

Reference

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Last Modified by:** Paul Butler **Date:** September 29, 2016
- **Last Reviewed by:** Richard Heenan **Date:** March 21, 2016

1.1.5 Parallelepiped Functions

core_shell_parallelepiped

Rectangular solid with a core-shell structure.

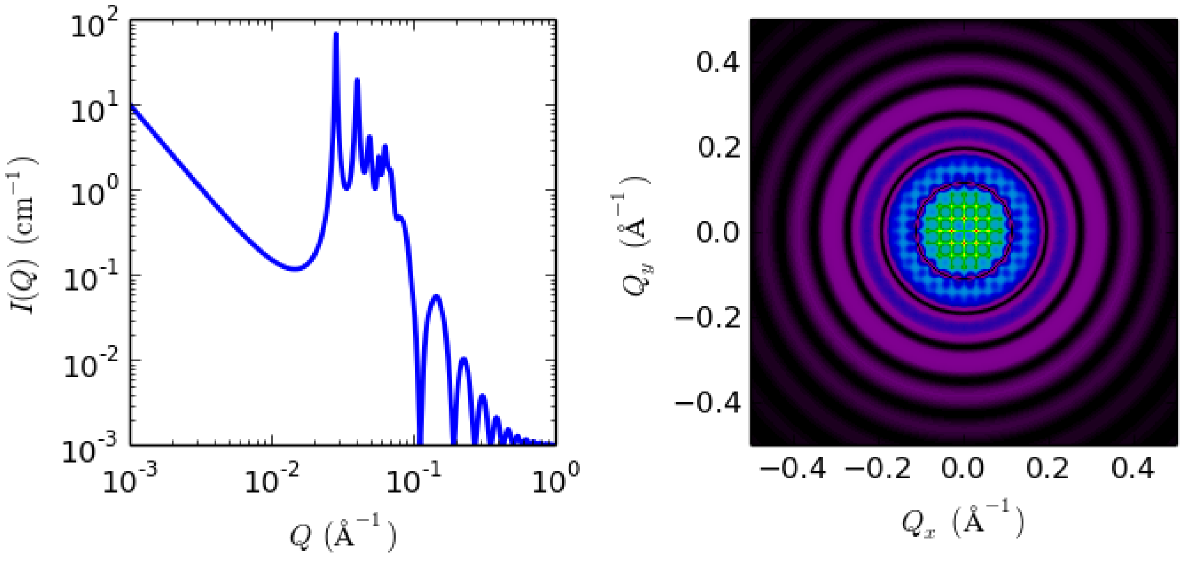


Fig. 1.58: 1D and 2D plots corresponding to the default parameters of the model.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
sld_core	Parallelepiped core scattering length density	10^{-6}Å^{-2}	1
sld_a	Parallelepiped A rim scattering length density	10^{-6}Å^{-2}	2
sld_b	Parallelepiped B rim scattering length density	10^{-6}Å^{-2}	4
sld_c	Parallelepiped C rim scattering length density	10^{-6}Å^{-2}	2
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	6
length_a	Shorter side of the parallelepiped	Å	35
length_b	Second side of the parallelepiped	Å	75
length_c	Larger side of the parallelepiped	Å	400
thick_rim_a	Thickness of A rim	Å	10
thick_rim_b	Thickness of B rim	Å	10
thick_rim_c	Thickness of C rim	Å	10
theta	c axis to beam angle	degree	0
phi	rotation about beam	degree	0
psi	rotation about c axis	degree	0

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

Calculates the form factor for a rectangular solid with a core-shell structure. The thickness and the scattering length density of the shell or "rim" can be different on each (pair) of faces. The three dimensions of the core of the parallelepiped (strictly here a cuboid) may be given in *any* size order as long as the particles are randomly oriented (i.e. take on all possible orientations see notes on 2D below). To avoid multiple fit solutions, especially with Monte-Carlo fit methods, it may be advisable to restrict their ranges. There may be a number of closely similar "best fits", so some trial and error, or fixing of some dimensions at expected values, may help.

The form factor is normalized by the particle volume V such that

$$I(q) = \frac{\text{scale}}{V}\langle P(q,\alpha,\beta)\rangle + \text{background}$$

where $\langle \ldots \rangle$ is an average over all possible orientations of the rectangular solid, and the usual $\Delta\rho^2\, V^2$ term cannot be pulled out of the form factor term due to the multiple slds in the model.

The core of the solid is defined by the dimensions A, B, C here shown such that $A < B < C$.

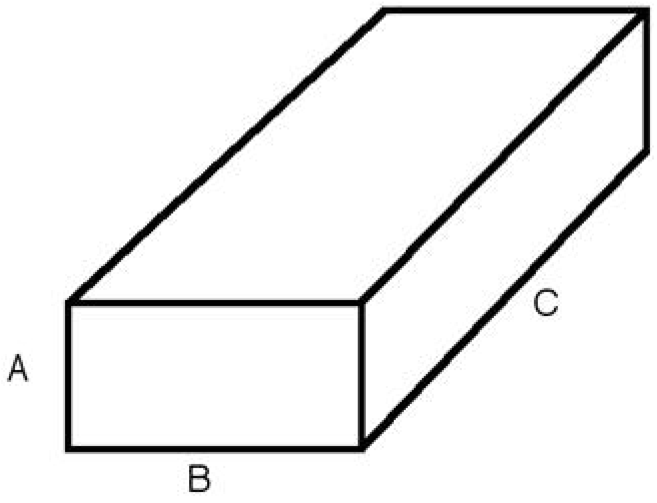


Fig. 1.59: Core of the core shell parallelepiped with the corresponding definition of sides.

There are rectangular "slabs" of thickness t_A that add to the A dimension (on the BC faces). There are similar slabs on the AC $(= t_B)$ and AB $(= t_C)$ faces. The projection in the AB plane is

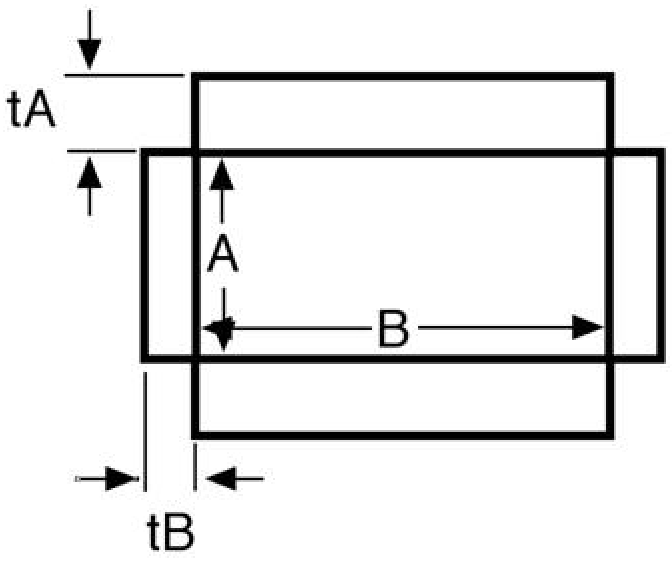


Fig. 1.60: AB cut through the core-shell parallelipiped showing the cross secion of four of the six shell slabs. As can be seen, this model leaves **"gaps"** at the corners of the solid.

The total volume of the solid is thus given as

$$V = ABC + 2t_A BC + 2t_B AC + 2t_C AB$$

The intensity calculated follows the *parallelepiped* model, with the core-shell intensity being calculated as the square of the sum of the amplitudes of the core and the slabs on the edges. The scattering amplitude is computed for a particular orientation of the core-shell parallelepiped with respect to the scattering vector and then averaged over all possible orientations, where α is the angle between the z axis and the C axis of the parallelepiped, and β is the angle between the projection of the particle in the xy detector plane and the y axis.

$$P(q) = \frac{\int_0^{\pi/2} \int_0^{\pi/2} F^2(q, \alpha, \beta) \; sin\alpha \; d\alpha \; d\beta}{\int_0^{\pi/2} \; sin\alpha \; d\alpha \; d\beta}$$

and

$$\begin{aligned}F(q,\alpha,\beta) = &(\rho_{\text{core}} - \rho_{\text{solvent}})S(Q_A,A)S(Q_B,B)S(Q_C,C)\\ &+ (\rho_{\text{A}} - \rho_{\text{solvent}})\left[S(Q_A,A+2t_A) - S(Q_A,A)\right]S(Q_B,B)S(Q_C,C)\\ &+ (\rho_{\text{B}} - \rho_{\text{solvent}})S(Q_A,A)\left[S(Q_B,B+2t_B) - S(Q_B,B)\right]S(Q_C,C)\\ &+ (\rho_{\text{C}} - \rho_{\text{solvent}})S(Q_A,A)S(Q_B,B)\left[S(Q_C,C+2t_C) - S(Q_C,C)\right]\end{aligned}$$

with

$$S(Q_X,L) = L\frac{\sin(\frac{1}{2}Q_X L)}{\frac{1}{2}Q_X L}$$

and

$$\begin{aligned}Q_A &= q\sin\alpha\sin\beta\\ Q_B &= q\sin\alpha\cos\beta\\ Q_C &= q\cos\alpha\end{aligned}$$

where ρ_{core}, ρ_{A}, ρ_{B} and ρ_{C} are the scattering lengths of the parallelepiped core, and the rectangular slabs of thickness t_A, t_B and t_C, respectively. ρ_{solvent} is the scattering length of the solvent.

Note: the code actually implements two substitutions: $d(cos\alpha)$ is substituted for -$sin\alpha\ d\alpha$ (note that in the *parallelepiped* code this is explicitly implemented with $\sigma = cos\alpha$), and β is set to $\beta = u\pi/2$ so that $du = \pi/2\ d\beta$. Thus both integrals go from 0 to 1 rather than 0 to $\pi/2$.

FITTING NOTES

1. There are many parameters in this model. Hold as many fixed as possible with known values, or you will certainly end up at a solution that is unphysical.
2. The 2nd virial coefficient of the core_shell_parallelepiped is calculated based on the the averaged effective radius ($= \sqrt{(A+2t_A)(B+2t_B)/\pi}$) and length $(C+2t_C)$ values, after appropriately sorting the three dimensions to give an oblate or prolate particle, to give an effective radius for $S(q)$ when $P(q)*S(q)$ is applied.
3. For 2d data the orientation of the particle is required, described using angles θ, ϕ and Ψ as in the diagrams below, where θ and ϕ define the orientation of the director in the laboratry reference frame of the beam direction (z) and detector plane ($x-y$ plane), while the angle Ψ is effectively the rotational angle around the particle C axis. For $\theta = 0$ and $\phi = 0$, $\Psi = 0$ corresponds to the B axis oriented parallel to the y-axis of the detector with A along the x-axis. For other θ, ϕ values, the order of rotations matters. In particular, the parallelepiped must first be rotated θ degrees in the $x-z$ plane before rotating ϕ degrees around the z axis (in the $x-y$ plane). Applying orientational distribution to the particle orientation (i.e *jitter* to one or more of these angles) can get more confusing as *jitter* is defined **NOT** with respect to the laboratory frame but the particle reference frame. It is thus highly recmmended to read *Oriented particles* for further details of the calculation and angular dispersions.

Note: For 2d, constraints must be applied during fitting to ensure that the order of sides chosen is not altered, and hence that the correct definition of angles is preserved. For the default choice shown here, that means ensuring that the inequality $A < B < C$ is not violated, The calculation will not report an error, but the results may be not correct.

Validation

Cross-checked against hollow rectangular prism and rectangular prism for equal thickness overlapping sides, and by Monte Carlo sampling of points within the shape for non-uniform, non-overlapping sides.

References

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010

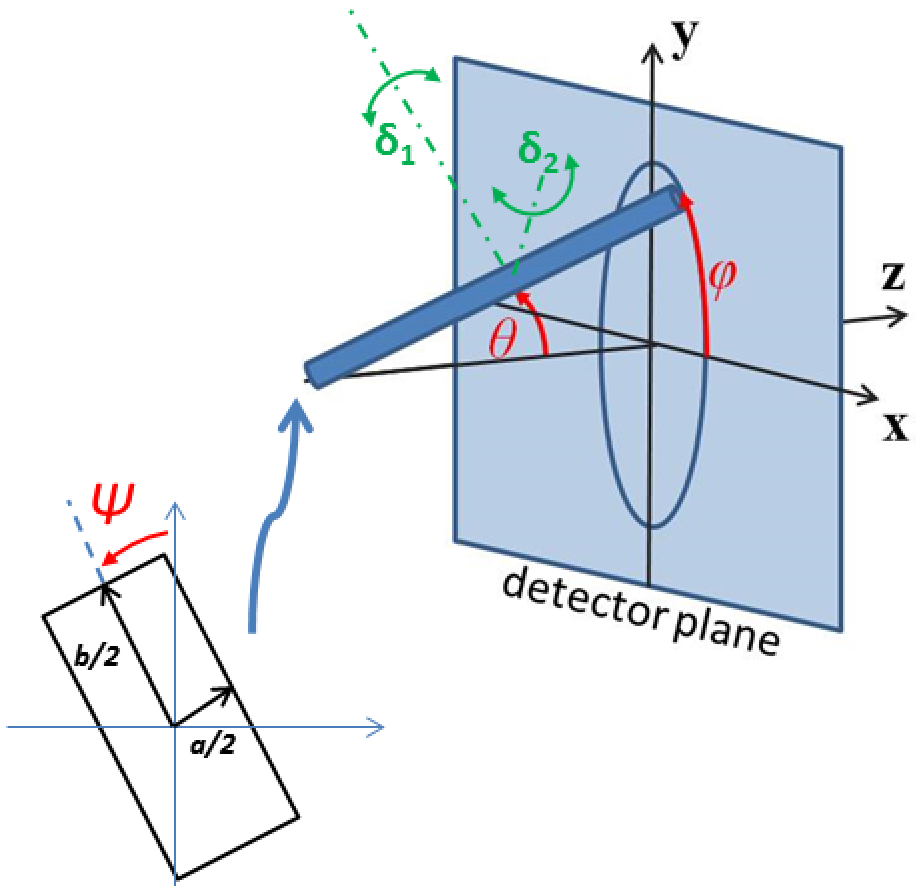


Fig. 1.61: Definition of the angles for oriented core-shell parallelepipeds. Note that rotation θ, initially in the $x - z$ plane, is carried out first, then rotation ϕ about the z axis, finally rotation Ψ is now around the C axis of the particle. The neutron or X-ray beam is along the z axis and the detecotr defines the $x - y$ plane.

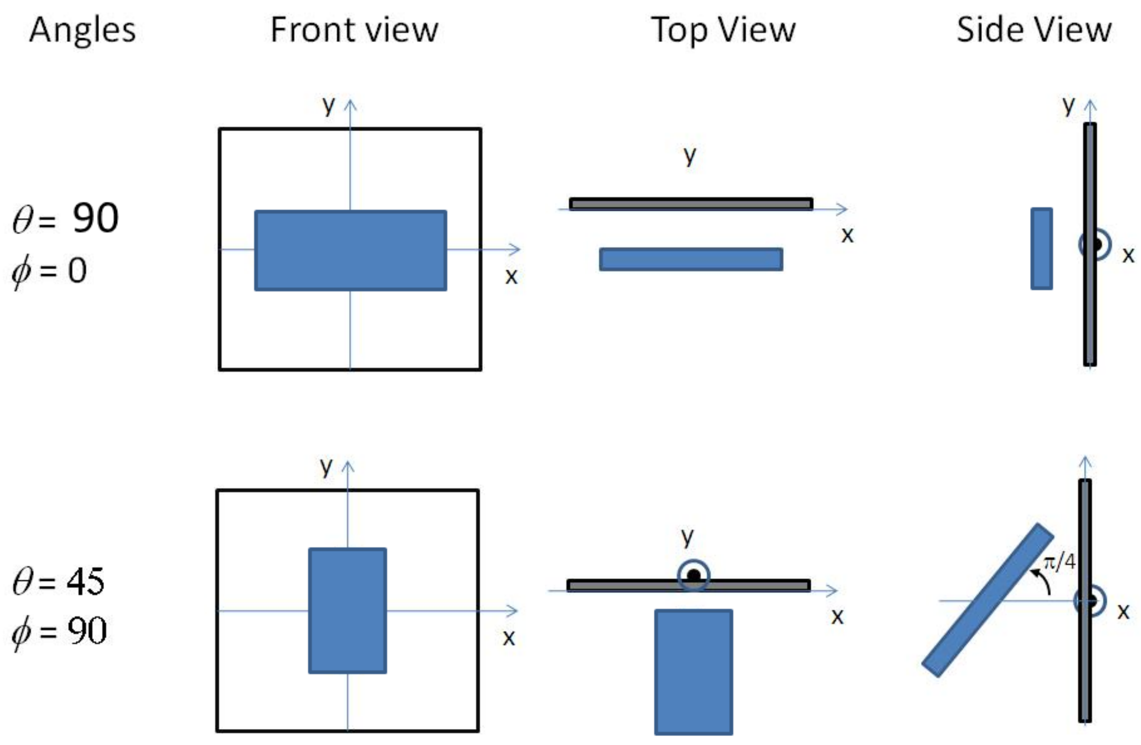


Fig. 1.62: Examples of the angles for oriented core-shell parallelepipeds against the detector plane.

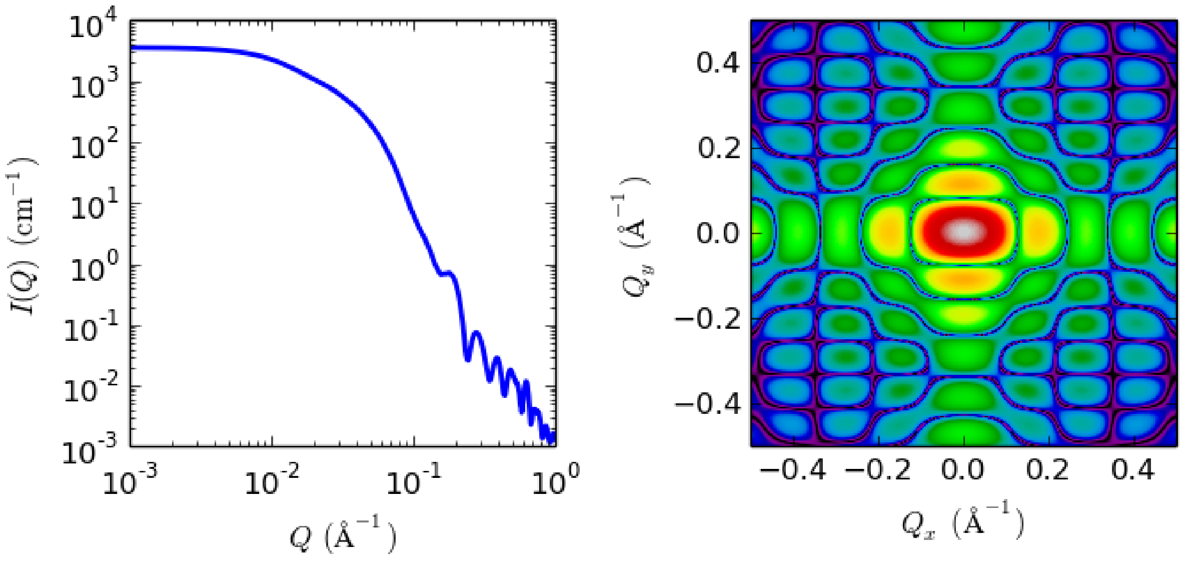


Fig. 1.63: 1D and 2D plots corresponding to the default parameters of the model.

- **Converted to sasmodels by:** Miguel Gonzalez **Date:** February 26, 2016
- **Last Modified by:** Paul Kienzle **Date:** October 17, 2017
- **Last Reviewed by:** Paul Butler **Date:** May 24, 2018 - documentation updated

hollow_rectangular_prism

Hollow rectangular parallelepiped with uniform scattering length density.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
sld	Parallelepiped scattering length density	10^{-6}Å^{-2}	6.3
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	1
length_a	Shorter side of the parallelepiped	Å	35
b2a_ratio	Ratio sides b/a	Å	1
c2a_ratio	Ratio sides c/a	Å	1
thickness	Thickness of parallelepiped	Å	1
theta	c axis to beam angle	degree	0
phi	rotation about beam	degree	0
psi	rotation about c axis	degree	0

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

This model provides the form factor, $P(q)$, for a hollow rectangular parallelepiped with a wall of thickness Δ. The 1D scattering intensity for this model is calculated by forming the difference of the amplitudes of two massive parallelepipeds differing in their outermost dimensions in each direction by the same length increment 2Δ ([1] Nayuk, 2012).

As in the case of the massive parallelepiped model (*rectangular_prism*), the scattering amplitude is computed for a particular orientation of the parallelepiped with respect to the scattering vector and then averaged over all possible orientations, giving

$$P(q) = \frac{1}{V^2}\frac{2}{\pi} \times \int_0^{\frac{\pi}{2}} \int_0^{\frac{\pi}{2}} A_{P\Delta}^2(q)\,\sin\theta\,d\theta\,d\phi$$

where θ is the angle between the z axis and the longest axis of the parallelepiped, ϕ is the angle between the scattering vector (lying in the xy plane) and the y axis, and

$$A_{P\Delta}(q) = ABC\left[\frac{\sin\left(q\frac{C}{2}\cos\theta\right)}{\left(q\frac{C}{2}\cos\theta\right)}\right]\left[\frac{\sin\left(q\frac{A}{2}\sin\theta\sin\phi\right)}{\left(q\frac{A}{2}\sin\theta\sin\phi\right)}\right]\left[\frac{\sin\left(q\frac{B}{2}\sin\theta\cos\phi\right)}{\left(q\frac{B}{2}\sin\theta\cos\phi\right)}\right]$$
$$-8\left(\frac{A}{2}-\Delta\right)\left(\frac{B}{2}-\Delta\right)\left(\frac{C}{2}-\Delta\right)\left[\frac{\sin\left[q\left(\frac{C}{2}-\Delta\right)\cos\theta\right]}{q\left(\frac{C}{2}-\Delta\right)\cos\theta}\right]\left[\frac{\sin\left[q\left(\frac{A}{2}-\Delta\right)\sin\theta\sin\phi\right]}{q\left(\frac{A}{2}-\Delta\right)\sin\theta\sin\phi}\right]\left[\frac{\sin\left[q\left(\frac{B}{2}-\Delta\right)\sin\theta\cos\phi\right.}{q\left(\frac{B}{2}-\Delta\right)\sin\theta\cos\phi}\right.$$

where A, B and C are the external sides of the parallelepiped fulfilling $A \le B \le C$, and the volume V of the parallelepiped is

$$V = ABC\, -\, (A-2\Delta)(B-2\Delta)(C-2\Delta)$$

The 1D scattering intensity is then calculated as

$$I(q) = \text{scale} \times V \times (\rho_\text{p} - \rho_\text{solvent})^2 \times P(q) + \text{background}$$

where ρ_p is the scattering length density of the parallelepiped, ρ_solvent is the scattering length density of the solvent, and (if the data are in absolute units) *scale* represents the volume fraction (which is unitless) of the rectangular shell of material (i.e. not including the volume of the solvent filled core).

[1] R Nayuk and K Huber, *Z. Phys. Chem.*, 226 (2012) 837-854

For 2d data the orientation of the particle is required, described using angles θ, ϕ and Ψ as in the diagrams below, for further details of the calculation and angular dispersions see *Oriented particles* . The angle Ψ is the rotational angle around the long C axis. For example, $\Psi = 0$ when the B axis is parallel to the x-axis of the detector.

For 2d, constraints must be applied during fitting to ensure that the inequality $A < B < C$ is not violated, and hence the correct definition of angles is preserved. The calculation will not report an error if the inequality is *not* preserved, but the results may be not correct.

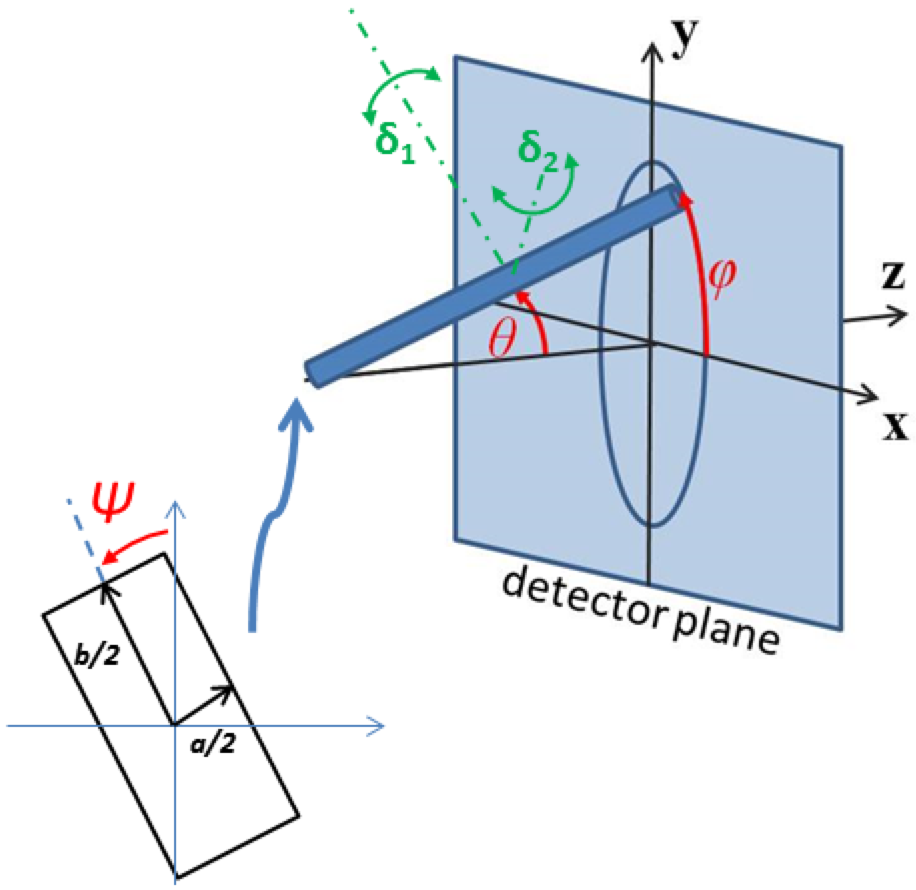


Fig. 1.64: Definition of the angles for oriented hollow rectangular prism. Note that rotation θ, initially in the xz plane, is carried out first, then rotation ϕ about the z axis, finally rotation Ψ is now around the axis of the prism. The neutron or X-ray beam is along the z axis.

Validation

Validation of the code was conducted by qualitatively comparing the output of the 1D model to the curves shown in (Nayuk, 2012).

References

Authorship and Verification

- **Author:** Miguel Gonzales **Date:** February 26, 2016
- **Last Modified by:** Paul Kienzle **Date:** December 14, 2017
- **Last Reviewed by:** Paul Butler **Date:** September 06, 2018

hollow_rectangular_prism_thin_walls

Hollow rectangular parallelepiped with thin walls.

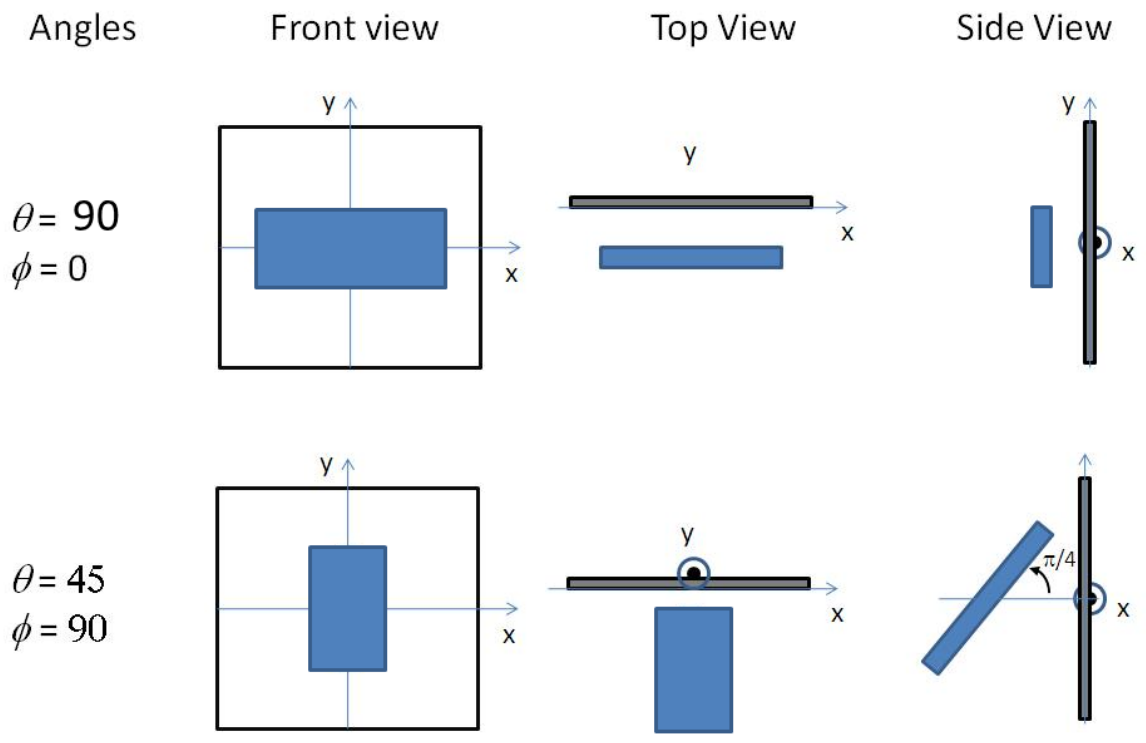


Fig. 1.65: Examples of the angles for oriented hollow rectangular prisms against the detector plane.

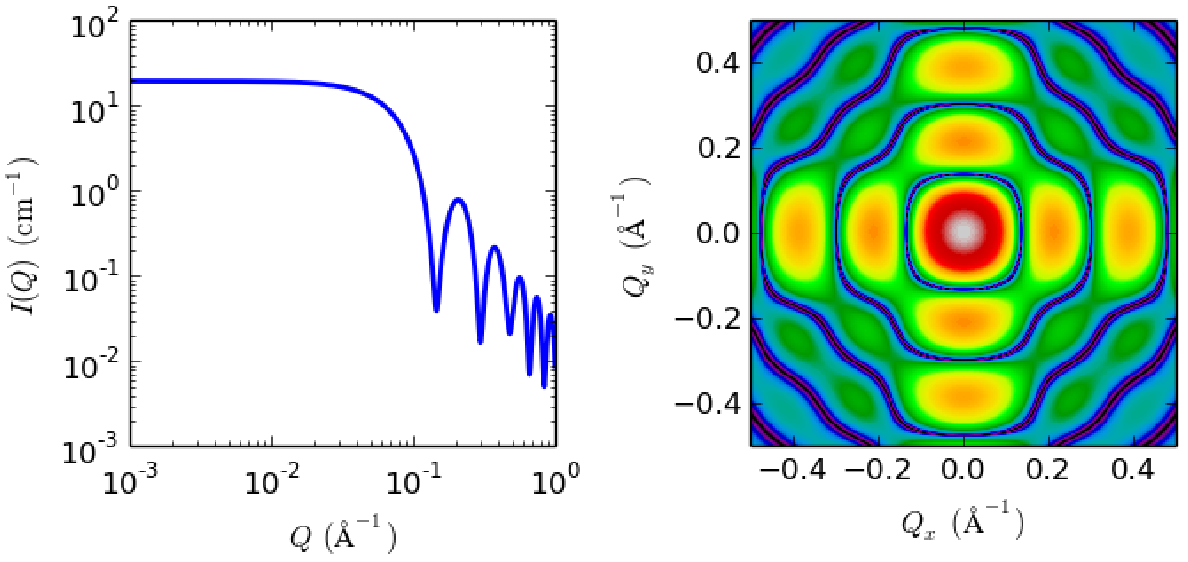


Fig. 1.66: 1D and 2D plots corresponding to the default parameters of the model.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
sld	Parallelepiped scattering length density	$10^{-6}Å^{-2}$	6.3
sld_solvent	Solvent scattering length density	$10^{-6}Å^{-2}$	1
length_a	Shorter side of the parallelepiped	Å	35
b2a_ratio	Ratio sides b/a	Å	1
c2a_ratio	Ratio sides c/a	Å	1

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

This model provides the form factor, $P(q)$, for a hollow rectangular prism with infinitely thin walls. It computes only the 1D scattering, not the 2D. The 1D scattering intensity for this model is calculated according to the equations given by Nayuk and Huber[1].

Assuming a hollow parallelepiped with infinitely thin walls, edge lengths $A \le B \le C$ and presenting an orientation with respect to the scattering vector given by θ and ϕ, where θ is the angle between the z axis and the longest axis of the parallelepiped C, and ϕ is the angle between the scattering vector (lying in the xy plane) and the y axis, the form factor is given by

$$P(q) = \frac{1}{V^2}\frac{2}{\pi}\int_0^{\frac{\pi}{2}}\int_0^{\frac{\pi}{2}}[A_L(q)+A_T(q)]^2 \sin\theta\, d\theta\, d\phi$$

where

$$\begin{aligned} V &= 2AB + 2AC + 2BC \\ A_L(q) &= 8 \times \frac{\sin\left(\frac{1}{2}qA\sin\phi\sin\theta\right)\sin\left(\frac{1}{2}qB\cos\phi\sin\theta\right)\cos\left(\frac{1}{2}qC\cos\theta\right)}{q^2\,\sin^2\theta\,\sin\phi\cos\phi} \\ A_T(q) &= A_F(q) \times \frac{2\,\sin\left(\frac{1}{2}qC\cos\theta\right)}{q\,\cos\theta} \end{aligned}$$

and

$$A_F(q) = 4\frac{\cos\left(\frac{1}{2}qA\sin\phi\sin\theta\right)\sin\left(\frac{1}{2}qB\cos\phi\sin\theta\right)}{q\,\cos\phi\,\sin\theta} + 4\frac{\sin\left(\frac{1}{2}qA\sin\phi\sin\theta\right)\cos\left(\frac{1}{2}qB\cos\phi\sin\theta\right)}{q\,\sin\phi\,\sin\theta}$$

The 1D scattering intensity is then calculated as

$$I(q) = \text{scale} \times V \times (\rho_\text{p} - \rho_\text{solvent})^2 \times P(q)$$

where V is the surface area of the rectangular prism, ρ_p is the scattering length density of the parallelepiped, ρ_solvent is the scattering length density of the solvent, and (if the data are in absolute units) *scale* is related to the total surface area.

The 2D scattering intensity is not computed by this model.

Validation

Validation of the code was conducted by qualitatively comparing the output of the 1D model to the curves shown in (Nayuk, 2012[1]).

References

Authorship and Verification

- **Author:** Miguel Gonzales **Date:** February 26, 2016
- **Last Modified by:** Paul Kienzle **Date:** October 15, 2016
- **Last Reviewed by:** Paul Butler **Date:** September 07, 2018

[1] R Nayuk and K Huber, *Z. Phys. Chem.*, 226 (2012) 837-854

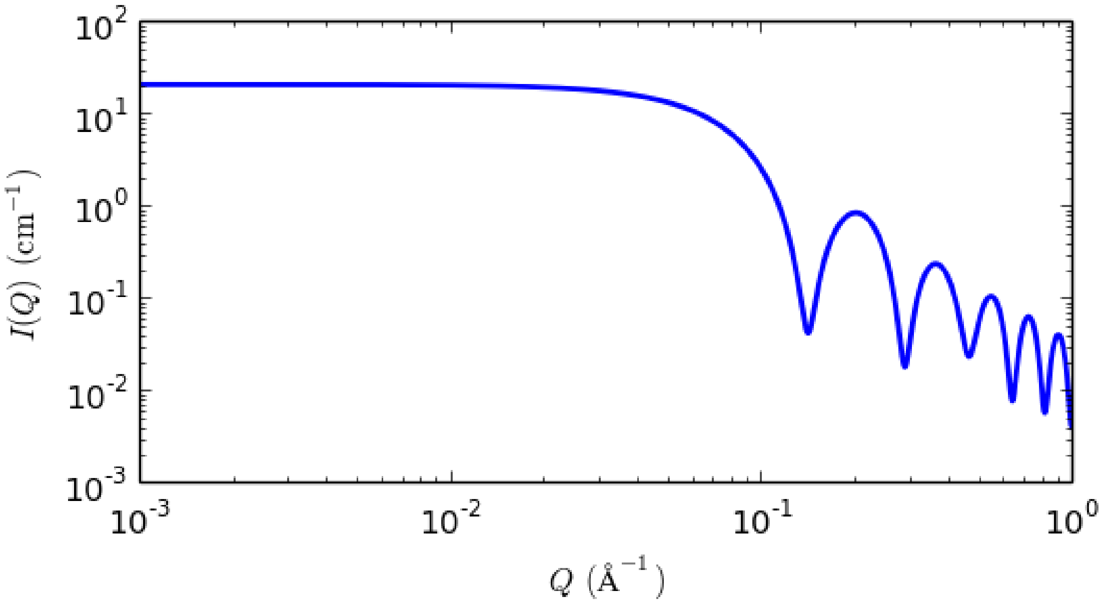


Fig. 1.67: 1D plot corresponding to the default parameters of the model.

parallelepiped

Rectangular parallelepiped with uniform scattering length density.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
sld	Parallelepiped scattering length density	10^{-6}Å^{-2}	4
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	1
length_a	Shorter side of the parallelepiped	Å	35
length_b	Second side of the parallelepiped	Å	75
length_c	Larger side of the parallelepiped	Å	400
theta	c axis to beam angle	degree	60
phi	rotation about beam	degree	60
psi	rotation about c axis	degree	60

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

This model calculates the scattering from a rectangular solid (Fig. 1.68). If you need to apply polydispersity, see also *rectangular_prism*. For information about polarised and magnetic scattering, see the *Polarisation/Magnetic Scattering* documentation.

The three dimensions of the parallelepiped (strictly here a cuboid) may be given in *any* size order as long as the particles are randomly oriented (i.e. take on all possible orientations see notes on 2D below). To avoid multiple fit solutions, especially with Monte-Carlo fit methods, it may be advisable to restrict their ranges. There may be a number of closely similar "best fits", so some trial and error, or fixing of some dimensions at expected values, may help.

The form factor is normalized by the particle volume and the 1D scattering intensity $I(q)$ is then calculated as:

$$I(q) = \frac{\text{scale}}{V}(\Delta\rho \cdot V)^2 \left\langle P(q,\alpha,\beta) \right\rangle + \text{background}$$

where the volume $V = ABC$, the contrast is defined as $\Delta\rho = \rho_\text{p} - \rho_\text{solvent}$, $P(q,\alpha,\beta)$ is the form factor corresponding to a parallelepiped oriented at an angle α (angle between the long axis C and $\vec{q}$), and β (the angle

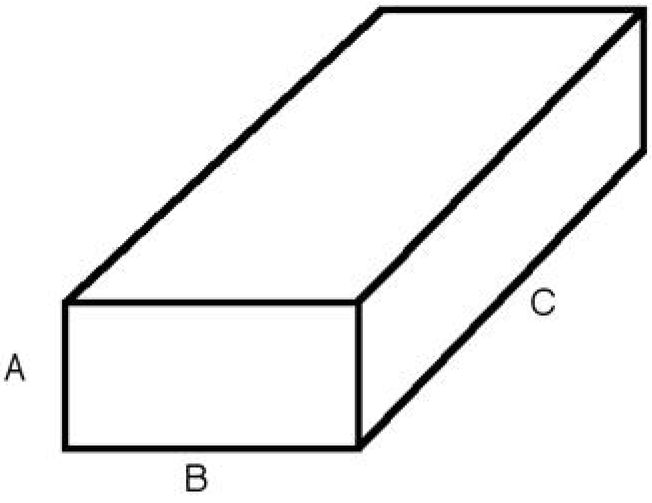


Fig. 1.68: Parallelepiped with the corresponding definition of sides.

between the projection of the particle in the xy detector plane and the y axis) and the averaging $\langle \ldots \rangle$ is applied over all orientations.

Assuming $a = A/B < 1$, $b = B/B = 1$, and $c = C/B > 1$, the form factor is given by (Mittelbach and Porod, 1961[1])

$$P(q, \alpha) = \int_0^1 \phi_Q\left(\mu\sqrt{1-\sigma^2}, a\right) [S(\mu c \sigma/2)]^2 \, d\sigma$$

with

$$\begin{aligned}
\phi_Q(\mu, a) &= \int_0^1 \left\{ S\left[\frac{\mu}{2}\cos\left(\frac{\pi}{2}u\right)\right] S\left[\frac{\mu a}{2}\sin\left(\frac{\pi}{2}u\right)\right] \right\}^2 du \\
S(x) &= \frac{\sin x}{x} \\
\mu &= qB
\end{aligned}$$

where substitution of $\sigma = cos\alpha$ and $\beta = \pi/2 \ u$ have been applied.

For **oriented** particles, the 2D scattering intensity, $I(q_x, q_y)$, is given as:

$$I(q_x, q_y) = \frac{\text{scale}}{V} (\Delta\rho \cdot V)^2 P(q_x, q_y) + \text{background}$$

Where $P(q_x, q_y)$ for a given orientation of the form factor is calculated as

$$P(q_x, q_y) = \left[\frac{\sin(\frac{1}{2}qA\cos\alpha)}{(\frac{1}{2}qA\cos\alpha)}\right]^2 \left[\frac{\sin(\frac{1}{2}qB\cos\beta)}{(\frac{1}{2}qB\cos\beta)}\right]^2 \left[\frac{\sin(\frac{1}{2}qC\cos\gamma)}{(\frac{1}{2}qC\cos\gamma)}\right]^2$$

with

$$\begin{aligned}
\cos\alpha &= \hat{A} \cdot \hat{q}, \\
\cos\beta &= \hat{B} \cdot \hat{q}, \\
\cos\gamma &= \hat{C} \cdot \hat{q}
\end{aligned}$$

FITTING NOTES

1. The 2nd virial coefficient of the parallelepiped is calculated based on the averaged effective radius, after appropriately sorting the three dimensions, to give an oblate or prolate particle, $(= \sqrt{AB/\pi})$ and length $(= C)$ values, and used as the effective radius for $S(q)$ when $P(q) \cdot S(q)$ is applied.

[1] P Mittelbach and G Porod, *Acta Physica Austriaca*, 14 (1961) 185-211

2. For 2d data the orientation of the particle is required, described using angles θ, ϕ and Ψ as in the diagrams below, where θ and ϕ define the orientation of the director in the laboratry reference frame of the beam direction (z) and detector plane ($x - y$ plane), while the angle Ψ is effectively the rotational angle around the particle C axis. For $\theta = 0$ and $\phi = 0$, $\Psi = 0$ corresponds to the B axis oriented parallel to the y-axis of the detector with A along the x-axis. For other θ, ϕ values, the order of rotations matters. In particular, the parallelepiped must first be rotated θ degrees in the $x - z$ plane before rotating ϕ degrees around the z axis (in the $x - y$ plane). Applying orientational distribution to the particle orientation (i.e *jitter* to one or more of these angles) can get more confusing as *jitter* is defined **NOT** with respect to the laboratory frame but the particle reference frame. It is thus highly recmmended to read *Oriented particles* for further details of the calculation and angular dispersions.

Note: For 2d, constraints must be applied during fitting to ensure that the order of sides chosen is not altered, and hence that the correct definition of angles is preserved. For the default choice shown here, that means ensuring that the inequality $A < B < C$ is not violated, The calculation will not report an error, but the results may be not correct.

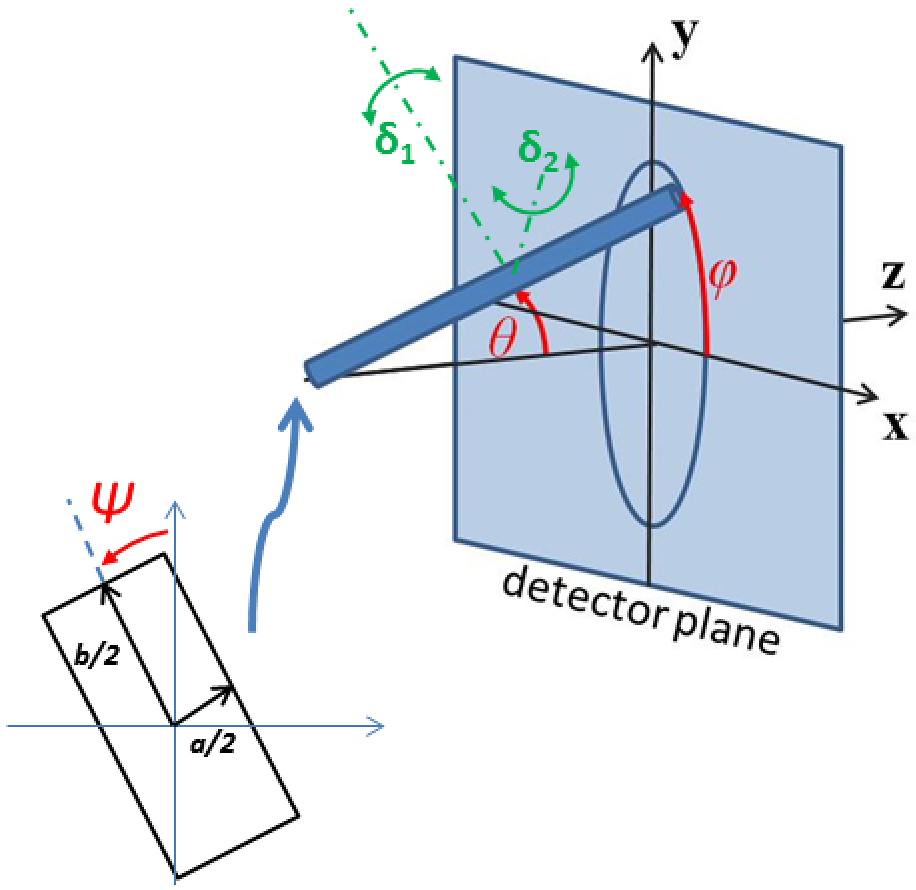


Fig. 1.69: Definition of the angles for oriented parallelepiped, shown with $A < B < C$.

Validation

Validation of the code was done by comparing the output of the 1D calculation to the angular average of the output of a 2D calculation over all possible angles.

References

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Last Modified by:** Paul Kienzle **Date:** April 05, 2017

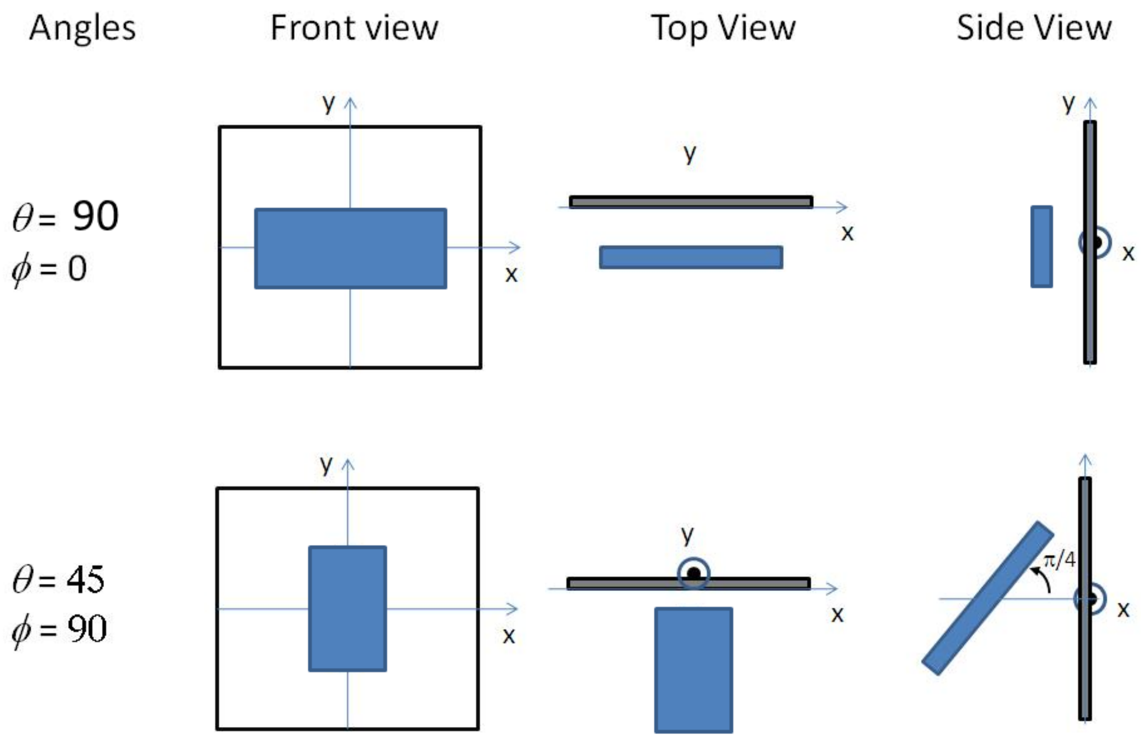


Fig. 1.70: Examples of the angles for an oriented parallelepiped against the detector plane.

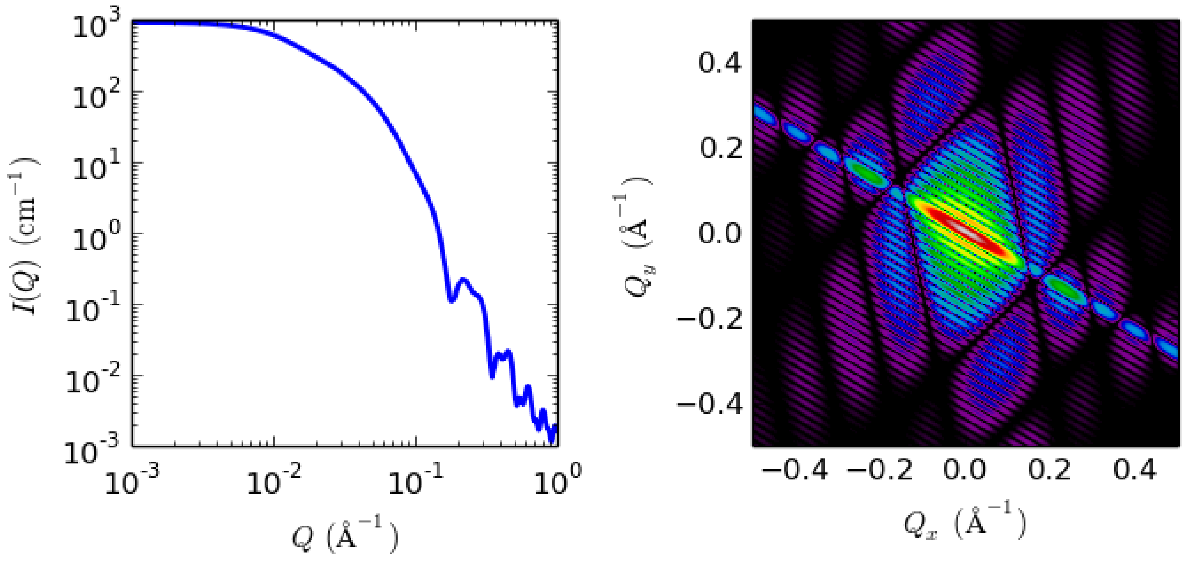


Fig. 1.71: 1D and 2D plots corresponding to the default parameters of the model.

- **Last Reviewed by:** Miguel Gonzales and Paul Butler **Date:** May 24, 2018 - documentation updated

rectangular_prism

Rectangular parallelepiped with uniform scattering length density.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
sld	Parallelepiped scattering length density	$10^{-6}Å^{-2}$	6.3
sld_solvent	Solvent scattering length density	$10^{-6}Å^{-2}$	1
length_a	Shorter side of the parallelepiped	Å	35
b2a_ratio	Ratio sides b/a	None	1
c2a_ratio	Ratio sides c/a	None	1
theta	c axis to beam angle	degree	0
phi	rotation about beam	degree	0
psi	rotation about c axis	degree	0

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

This model provides the form factor, $P(q)$, for a rectangular prism.

Note that this model is almost totally equivalent to the existing *parallelepiped* model. The only difference is that the way the relevant parameters are defined here (a, b/a, c/a instead of a, b, c) which allows use of polydispersity with this model while keeping the shape of the prism (e.g. setting $b/a = 1$ and $c/a = 1$ and applying polydispersity to a will generate a distribution of cubes of different sizes).

Definition

The 1D scattering intensity for this model was calculated by Mittelbach and Porod (Mittelbach, 1961), but the implementation here is closer to the equations given by Nayuk and Huber (Nayuk, 2012). Note also that the angle definitions used in the code and the present documentation correspond to those used in (Nayuk, 2012) (see Fig. 1 of that reference), with θ corresponding to α in that paper, and not to the usual convention used for example in the *parallelepiped* model.

In this model the scattering from a massive parallelepiped with an orientation with respect to the scattering vector given by θ and ϕ

$$A_P\left(q\right) = \frac{\sin\left(\frac{1}{2}qC\cos\theta\right)}{\frac{1}{2}qC\cos\theta} \times \frac{\sin\left(\frac{1}{2}qA\cos\theta\right)}{\frac{1}{2}qA\cos\theta} \times , \frac{\sin\left(\frac{1}{2}qB\cos\theta\right)}{\frac{1}{2}qB\cos\theta}$$

where A, B and C are the sides of the parallelepiped and must fulfill $A \leq B \leq C$, θ is the angle between the z axis and the longest axis of the parallelepiped C, and ϕ is the angle between the scattering vector (lying in the xy plane) and the y axis.

The normalized form factor in 1D is obtained averaging over all possible orientations

$$P(q) = \frac{2}{\pi}\int_0^{\frac{\pi}{2}}\int_0^{\frac{\pi}{2}} A_P^2(q)\,\sin\theta\,d\theta\,d\phi$$

And the 1D scattering intensity is calculated as

$$I(q) = \text{scale} \times V \times (\rho_\text{p} - \rho_\text{solvent})^2 \times P(q)$$

where V is the volume of the rectangular prism, ρ_p is the scattering length of the parallelepiped, ρ_solvent is the scattering length of the solvent, and (if the data are in absolute units) *scale* represents the volume fraction (which is unitless).

For 2d data the orientation of the particle is required, described using angles θ, ϕ and Ψ as in the diagrams below, for further details of the calculation and angular dispersions see *Oriented particles* . The angle Ψ is the rotational angle around the long C axis. For example, $\Psi = 0$ when the B axis is parallel to the x-axis of the detector.

For 2d, constraints must be applied during fitting to ensure that the inequality $A < B < C$ is not violated, and hence the correct definition of angles is preserved. The calculation will not report an error, but the results may be not correct.

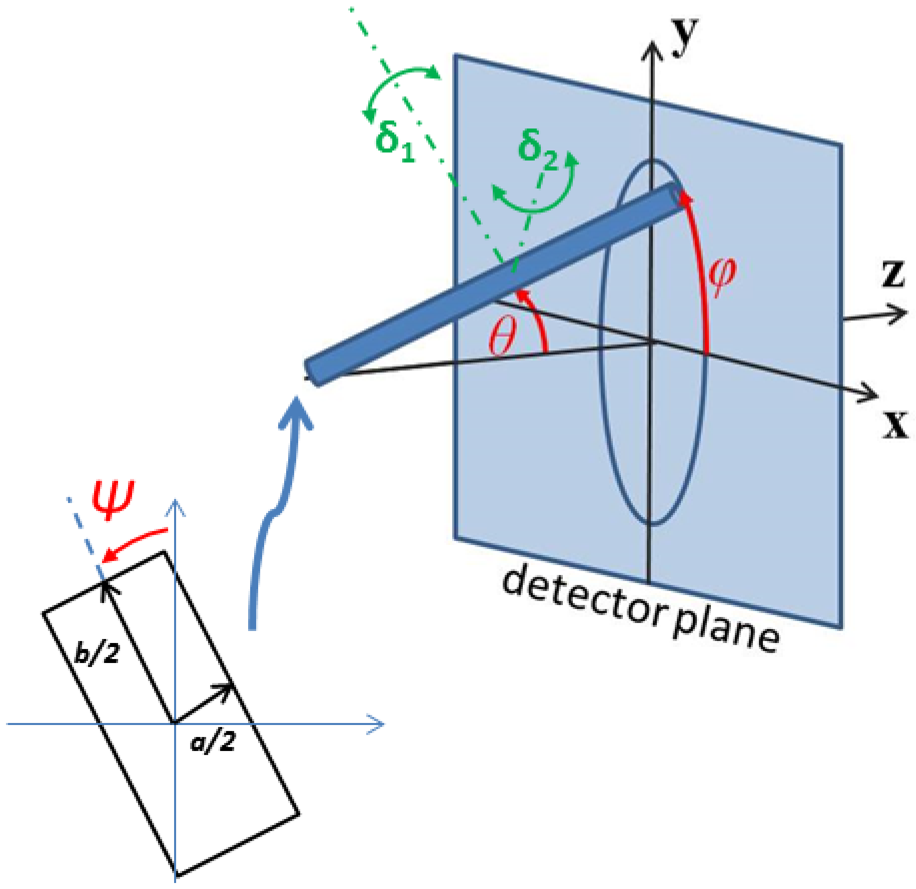


Fig. 1.72: Definition of the angles for oriented core-shell parallelepipeds. Note that rotation θ, initially in the xz plane, is carried out first, then rotation ϕ about the z axis, finally rotation Ψ is now around the axis of the cylinder. The neutron or X-ray beam is along the z axis.

Validation

Validation of the code was conducted by comparing the output of the 1D model to the output of the existing *parallelepiped* model.

References

P Mittelbach and G Porod, *Acta Physica Austriaca*, 14 (1961) 185-211

R Nayuk and K Huber, *Z. Phys. Chem.*, 226 (2012) 837-854

1.1.6 Sphere Functions

adsorbed_layer

Scattering from an adsorbed layer on particles

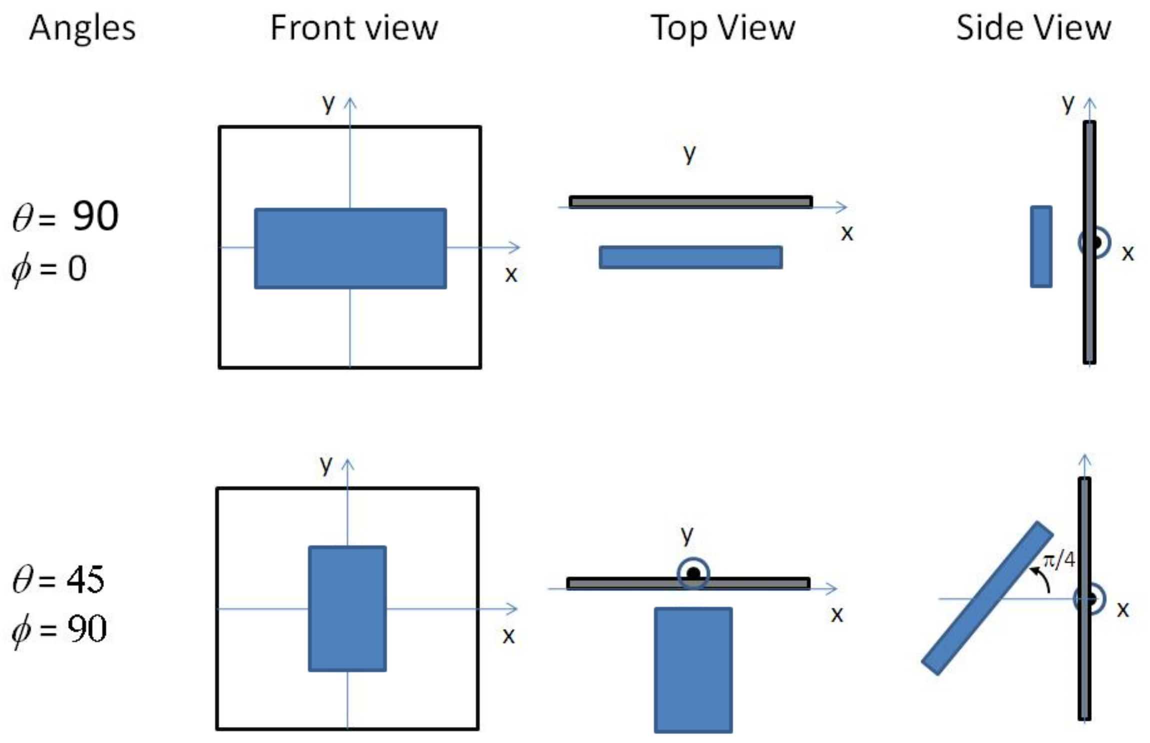


Fig. 1.73: Examples of the angles for oriented rectangular prisms against the detector plane.

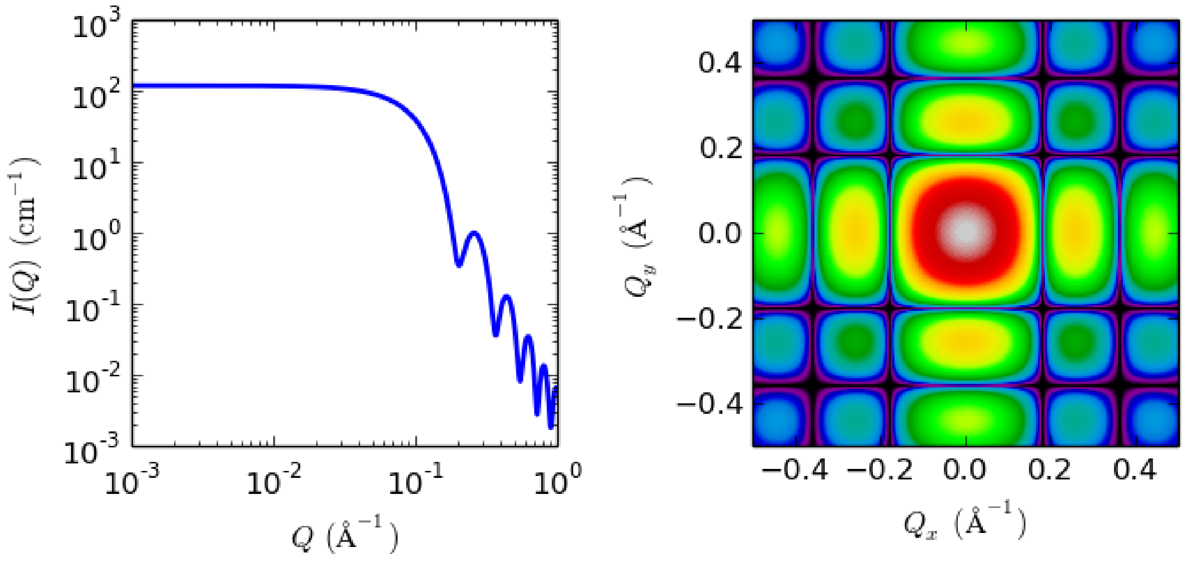


Fig. 1.74: 1D and 2D plots corresponding to the default parameters of the model.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
second_moment	Second moment of polymer distribution	Å	23
adsorbed_amount	Adsorbed amount of polymer	$mg \cdot m^{-2}$	1.9
density_shell	Bulk density of polymer in the shell	$g \cdot cm^{-3}$	0.7
radius	Core particle radius	Å	500
volfraction	Core particle volume fraction	None	0.14
sld_shell	Polymer shell SLD	10^{-6}Å^{-2}	1.5
sld_solvent	Solvent SLD	10^{-6}Å^{-2}	6.3

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

This model describes the scattering from a layer of surfactant or polymer adsorbed on large, smooth, notionally spherical particles under the conditions that (i) the particles (cores) are contrast-matched to the dispersion medium, (ii) $S(Q) \sim 1$ (ie, the particle volume fraction is dilute), (iii) the particle radius is >> layer thickness (ie, the interface is locally flat), and (iv) scattering from excess unadsorbed adsorbate in the bulk medium is absent or has been corrected for.

Unlike many other core-shell models, this model does not assume any form for the density distribution of the adsorbed species normal to the interface (cf, a core-shell model normally assumes the density distribution to be a homogeneous step-function). For comparison, if the thickness of a (traditional core-shell like) step function distribution is t, the second moment about the mean of the density distribution (ie, the distance of the centre-of-mass of the distribution from the interface), $\sigma = \sqrt{t^2/12}$.

$$I(q) = \text{scale} \cdot (\rho_\text{poly} - \rho_\text{solvent})^2 \left[\frac{6\pi\phi_\text{core}}{Q^2} \frac{\Gamma^2}{\delta^2_\text{poly} R_\text{core}} \exp(-Q^2\sigma^2) \right] + \text{background}$$

where *scale* is a scale factor, ρ_poly is the sld of the polymer (or surfactant) layer, ρ_solv is the sld of the solvent/medium and cores, ϕ_core is the volume fraction of the core particles, δ_poly is the bulk density of the polymer, Γ is the adsorbed amount, and σ is the second moment of the thickness distribution.

Note that all parameters except σ are correlated so fitting more than one of these parameters will generally fail. Also note that unlike other shape models, no volume normalization is applied to this model (the calculation is exact).

The code for this model is based originally on a a fortran implementation by Steve King at ISIS in the SANDRA package c. 1990.

References

Authorship and Verification

- **Author:** Jae-Hi Cho **Date:** pre 2010
- **Last Modified by:** Paul Kienzle **Date:** April 14, 2016
- **Last Reviewed by:** Steve King **Date:** March 18, 2016

binary_hard_sphere

binary mixture of hard spheres with hard sphere interactions.

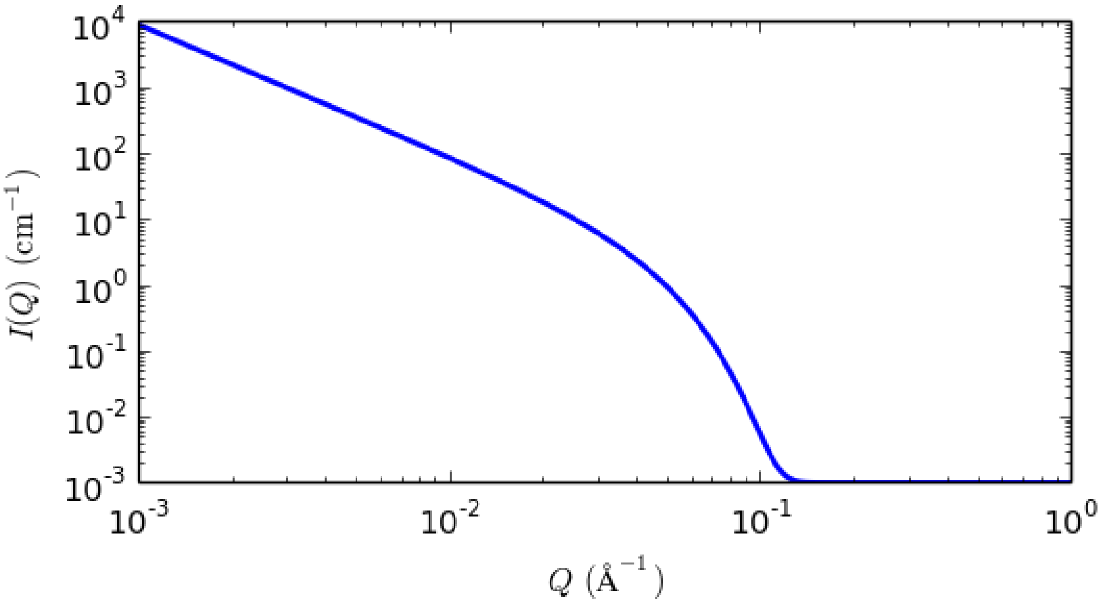


Fig. 1.75: 1D plot corresponding to the default parameters of the model.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
radius_lg	radius of large particle	Å	100
radius_sm	radius of small particle	Å	25
volfraction_lg	volume fraction of large particle	None	0.1
volfraction_sm	volume fraction of small particle	None	0.2
sld_lg	scattering length density of large particle	10^{-6}Å^{-2}	3.5
sld_sm	scattering length density of small particle	10^{-6}Å^{-2}	0.5
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	6.36

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

The binary hard sphere model provides the scattering intensity, for binary mixture of hard spheres including hard sphere interaction between those particles, using rhw Percus-Yevick closure. The calculation is an exact multi-component solution that properly accounts for the 3 partial structure factors as follows:

$$I(q) = (1-x)f_1^2(q)S_{11}(q) + 2[x(1-x)]^{1/2}f_1(q)f_2(q)S_{12}(q) + x\,f_2^2(q)S_{22}(q)$$

where S_{ij} are the partial structure factors and f_i are the scattering amplitudes of the particles. The subscript 1 is for the smaller particle and 2 is for the larger. The number fraction of the larger particle, ($x = n2/(n1+n2)$, where n = the number density) is internally calculated based on the diameter ratio and the volume fractions.

$$x = \frac{(\phi_2/\phi)\alpha^3}{(1-(\phi_2/\phi)+(\phi_2/\phi)\alpha^3)}$$
$$\phi = \phi_1 + \phi_2 = \text{total volume fraction}$$
$$\alpha = R_1/R_2 = \text{size ratio}$$

The 2D scattering intensity is the same as 1D, regardless of the orientation of the q vector which is defined as

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

NOTE 1: The volume fractions and the scattering contrasts are loosely correlated, so holding as many parameters fixed to known values during fitting will improve the robustness of the fit.

NOTE 2: Since the calculation uses the Percus-Yevick closure, all of the limitations of that closure relation apply here. Specifically, one should be wary of results for (total) volume fractions greater than approximately 40%. Depending on the size ratios or number fractions, the limit on total volume fraction may be lower.

NOTE 3: The heavy arithmatic operations also mean that at present the function is poorly behaved at very low qr. In some cases very large qr may also be poorly behaved. These should however be outside any useful region of qr.

The code for this model is based originally on a c-library implementation by the NIST Center for Neutron Research (Kline, 2006).

See the references for details.

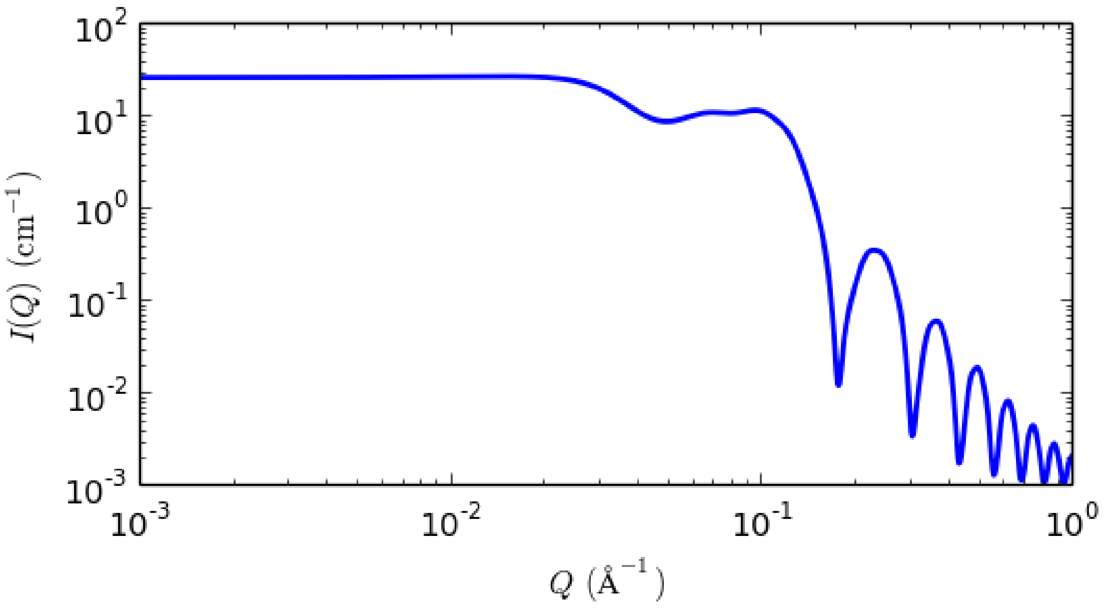


Fig. 1.76: 1D plot corresponding to the default parameters of the model.

References

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Last Modified by:** Paul Butler **Date:** March 20, 2016
- **Last Reviewed by:** Paul Butler **Date:** March 20, 2016

core_multi_shell

This model provides the scattering from a spherical core with 1 to 4 concentric shell structures. The SLDs of the core and each shell are individually specified.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
sld_core	Core scattering length density	$10^{-6}Å^{-2}$	1
radius	Radius of the core	Å	200
sld_solvent	Solvent scattering length density	$10^{-6}Å^{-2}$	6.4
n	number of shells	None	1
sld[n]	scattering length density of shell k	$10^{-6}Å^{-2}$	1.7
thickness[n]	Thickness of shell k	Å	40

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

This model is a trivial extension of the CoreShell function to a larger number of shells. The scattering length density profile for the default sld values (w/ 4 shells).

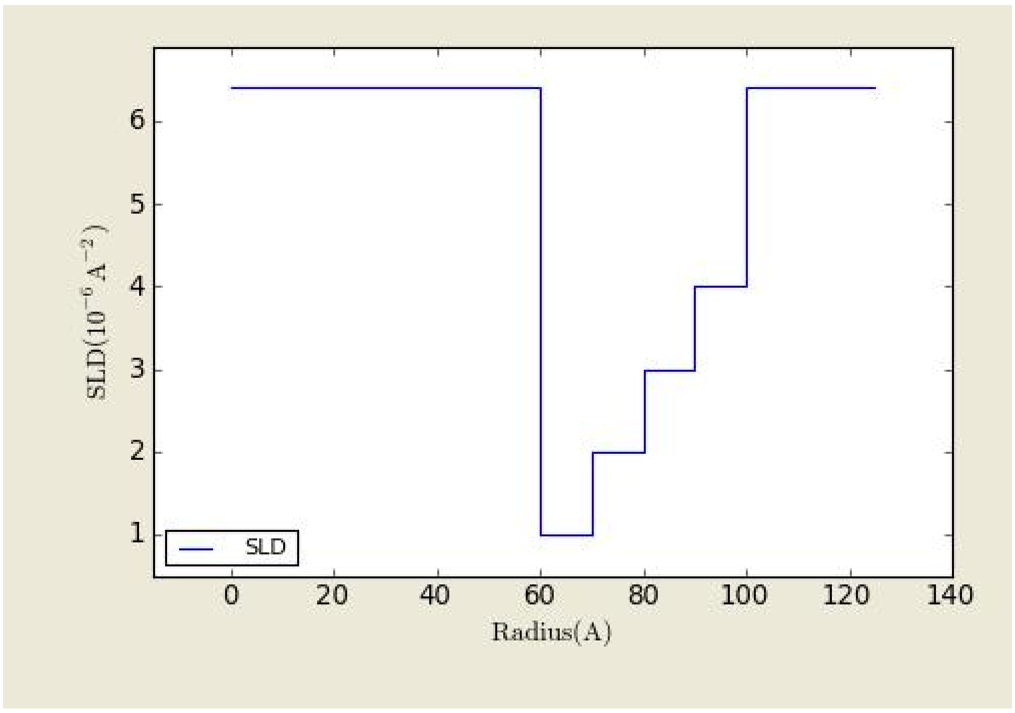


Fig. 1.77: SLD profile of the core_multi_shell object from the center of sphere out for the default SLDs.*

The 2D scattering intensity is the same as $P(q)$ above, regardless of the orientation of the $\vec{q}$ vector which is defined as

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

Note: Be careful! The SLDs and scale can be highly correlated. Hold as many of these parameters fixed as possible.

Note: The outer most radius (= *radius* + *thickness*) is used as the effective radius for $S(Q)$ when $P(Q) * S(Q)$ is applied.

For information about polarised and magnetic scattering, see the *Polarisation/Magnetic Scattering* documentation.

Our model uses the form factor calculations implemented in a c-library provided by the NIST Center for Neutron Research (Kline, 2006)[2].

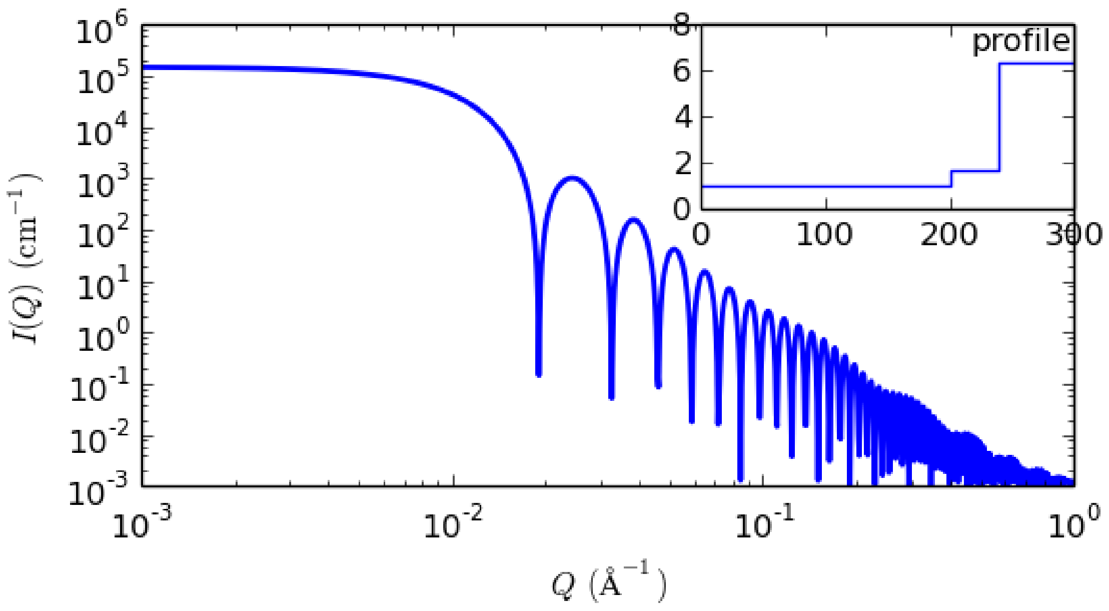


Fig. 1.78: 1D plot corresponding to the default parameters of the model.

References

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Last Modified by:** Paul Kienzle **Date:** September 12, 2016
- **Last Reviewed by:** Paul Kienzle **Date:** September 12, 2016

core_shell_sphere

Form factor for a monodisperse spherical particle with particle with a core-shell structure.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
radius	Sphere core radius	Å	60
thickness	Sphere shell thickness	Å	10
sld_core	core scattering length density	10^{-6}Å^{-2}	1
sld_shell	shell scattering length density	10^{-6}Å^{-2}	2
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	3

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

This model provides the form factor, $P(q)$, for a spherical particle with a core-shell structure. The form factor is normalized by the particle volume.

For information about polarised and magnetic scattering, see the *Polarisation/Magnetic Scattering* documentation.

Definition

[2] S R Kline, *J Appl. Cryst.*, 39 (2006) 895

The 1D scattering intensity is calculated in the following way (Guinier, 1955)

$$P(q) = \frac{\text{scale}}{V} F^2(q) + \text{background}$$

where

$$F(q) = \frac{3}{V_s}\left[V_c(\rho_c - \rho_s)\frac{\sin(qr_c) - qr_c\cos(qr_c)}{(qr_c)^3} + V_s(\rho_s - \rho_{\text{solv}})\frac{\sin(qr_s) - qr_s\cos(qr_s)}{(qr_s)^3}\right]$$

where V_s is the volume of the whole particle, V_c is the volume of the core, $r_s = radius + thickness$ is the radius of the particle, r_c is the radius of the core, ρ_c is the scattering length density of the core, ρ_s is the scattering length density of the shell, ρ_{solv}, is the scattering length density of the solvent.

The 2D scattering intensity is the same as $P(q)$ above, regardless of the orientation of the q vector.

NB: The outer most radius (ie, = radius + thickness) is used as the effective radius for $S(Q)$ when $P(Q) \cdot S(Q)$ is applied.

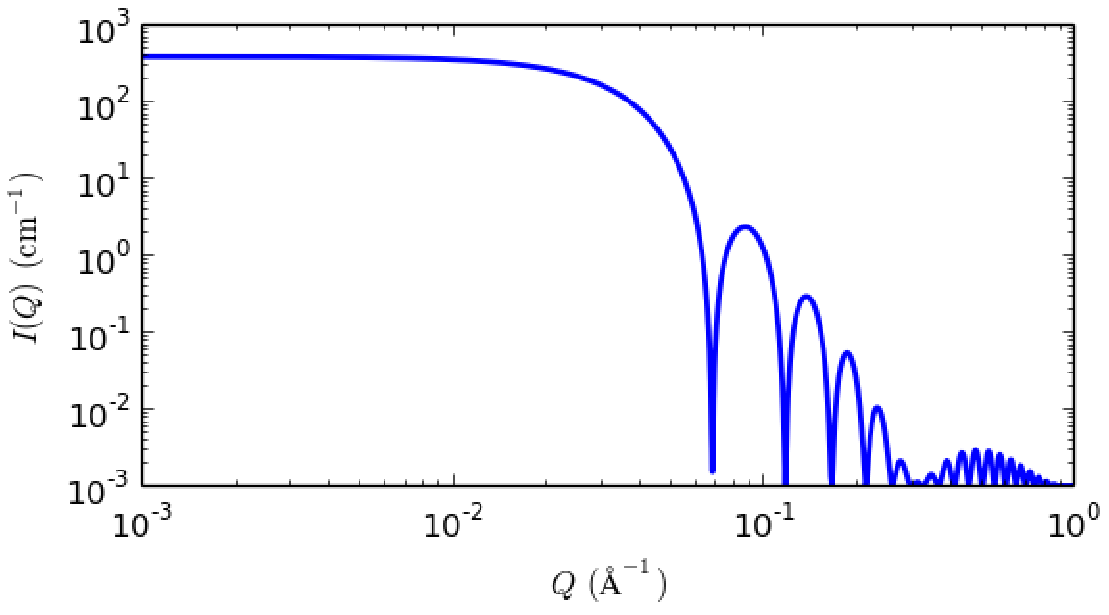


Fig. 1.79: 1D plot corresponding to the default parameters of the model.

References

A Guinier and G Fournet, *Small-Angle Scattering of X-Rays*, John Wiley and Sons, New York, (1955)

Validation

Validation of our code was done by comparing the output of the 1D model to the output of the software provided by NIST (Kline, 2006). Figure 1 shows a comparison of the output of our model and the output of the NIST software.

fuzzy_sphere

Scattering from spherical particles with a fuzzy surface.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
sld	Particle scattering length density	$10^{-6}Å^{-2}$	1
sld_solvent	Solvent scattering length density	$10^{-6}Å^{-2}$	3
radius	Sphere radius	Å	60
fuzziness	std deviation of Gaussian convolution for interface (must be << radius)	Å	10

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

For information about polarised and magnetic scattering, see the *Polarisation/Magnetic Scattering* documentation.

Definition

The scattering intensity $I(q)$ is calculated as:

$$I(q) = \frac{\text{scale}}{V}(\Delta\rho)^2 A^2(q) S(q) + \text{background}$$

where the amplitude $A(q)$ is given as the typical sphere scattering convoluted with a Gaussian to get a gradual drop-off in the scattering length density:

$$A(q) = \frac{3\left[\sin(qR) - qR\cos(qR)\right]}{(qR)^3}\exp\left(\frac{-(\sigma_{\text{fuzzy}}q)^2}{2}\right)$$

Here $A(q)^2$ is the form factor, $P(q)$. The scale is equivalent to the volume fraction of spheres, each of volume, V. Contrast $(\Delta\rho)$ is the difference of scattering length densities of the sphere and the surrounding solvent.

Poly-dispersion in radius and in fuzziness is provided for, though the fuzziness must be kept much smaller than the sphere radius for meaningful results.

From the reference:

> The "fuzziness" of the interface is defined by the parameter σ_{fuzzy}. The particle radius R represents the radius of the particle where the scattering length density profile decreased to 1/2 of the core density. σ_{fuzzy} is the width of the smeared particle surface; i.e., the standard deviation from the average height of the fuzzy interface. The inner regions of the microgel that display a higher density are described by the radial box profile extending to a radius of approximately $R_{\text{box}} \sim R - 2\sigma$. The profile approaches zero as $R_{\text{sans}} \sim R + 2\sigma$.

For 2D data: The 2D scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

References

M Stieger, J. S Pedersen, P Lindner, W Richtering, *Langmuir*, 20 (2004) 7283-7292

linear_pearls

Linear pearls model of scattering from spherical pearls.

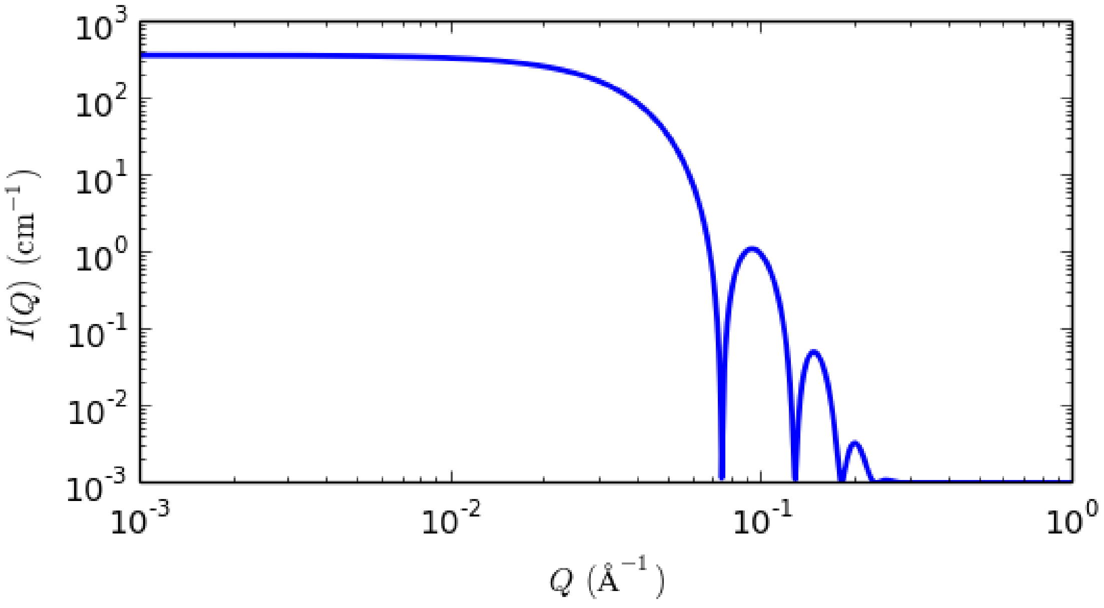
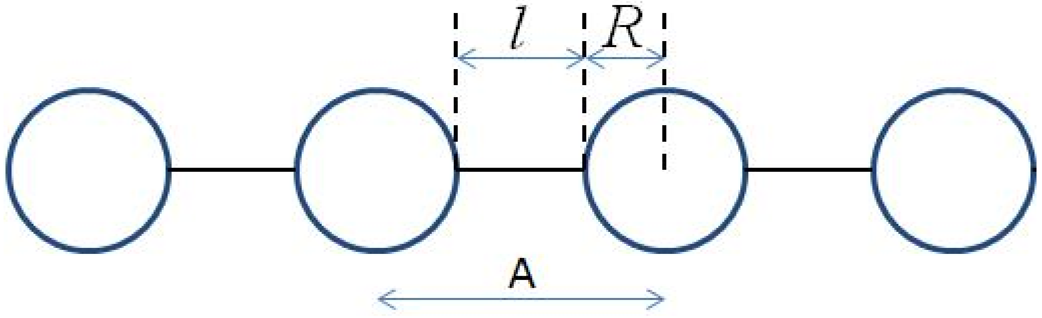


Fig. 1.80: 1D plot corresponding to the default parameters of the model.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
radius	Radius of the pearls	Å	80
edge_sep	Length of the string segment - surface to surface	Å	350
num_pearls	Number of the pearls	None	3
sld	SLD of the pearl spheres	10^{-6}Å^{-2}	1
sld_solvent	SLD of the solvent	10^{-6}Å^{-2}	6.3

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

This model provides the form factor for N spherical pearls of radius R linearly joined by short strings (or segment length or edge separation) l $(= A - 2R)$. A is the center-to-center pearl separation distance. The thickness of each string is assumed to be negligible.



Definition

The output of the scattering intensity function for the linear_pearls model is given by (Dobrynin, 1996)

$$P(Q) = \frac{\text{scale}}{V}\left[m_p^2\left(N + 2\sum_{n-1}^{N-1}(N-n)\frac{\sin(qnl)}{qnl}\right)\left(3\frac{\sin(qR) - qR\cos(qR)}{(qr)^3}\right)^2\right]$$

where the mass m_p is $(SLD_{pearl} - SLD_{solvent}) * (volume\ of\ N\ pearls)$. V is the total volume.

The 2D scattering intensity is the same as P(q) above, regardless of the orientation of the q vector.

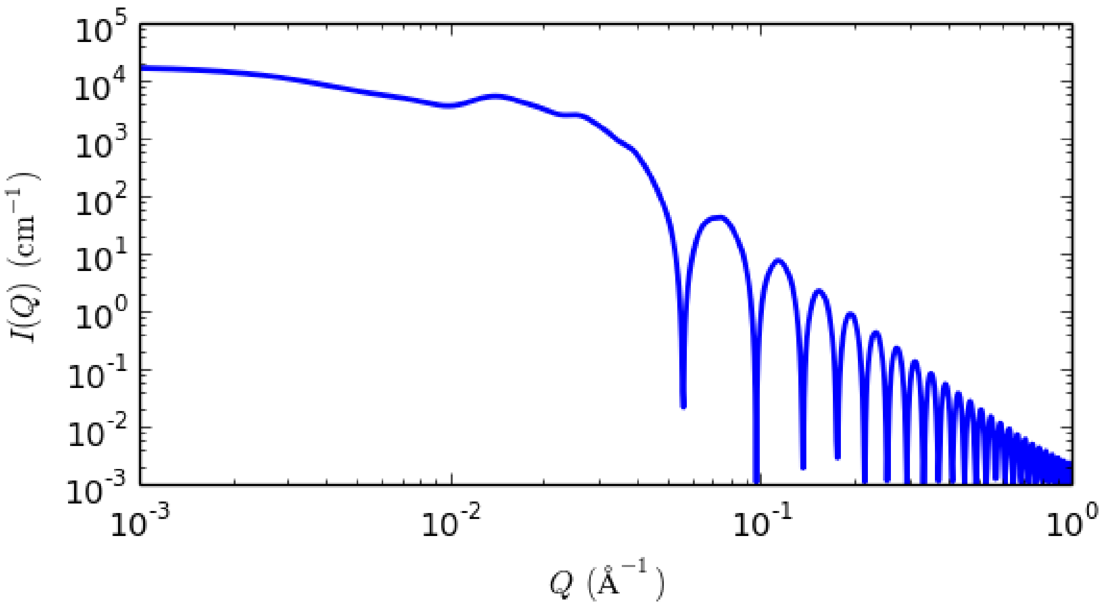


Fig. 1.81: 1D plot corresponding to the default parameters of the model.

References

A V Dobrynin, M Rubinstein and S P Obukhov, *Macromol.*, 29 (1996) 2974-2979

multilayer_vesicle

P(Q) for a Multi-lamellar vesicle

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
volfraction	volume fraction of vesicles	None	0.05
radius	radius of solvent filled core	Å	60
thick_shell	thickness of one shell	Å	10
thick_solvent	solvent thickness between shells	Å	10
sld_solvent	solvent scattering length density	$10^{-6}\mathrm{Å}^{-2}$	6.4
sld	Shell scattering length density	$10^{-6}\mathrm{Å}^{-2}$	0.4
n_shells	Number of shell plus solvent layer pairs	None	2

The returned value is scaled to units of $\mathrm{cm}^{-1}\ \mathrm{sr}^{-1}$, absolute scale.

Definition

This model is a trivial extension of the core_shell_sphere function where the core is filled with solvent and is surrounded by N shells of material (such as lipids) interleaved with $N-1$ layers of solvent. For $N=1$, this returns the same as the vesicle model, except for the normalisation, which here is to outermost volume. The shell thicknesses and SLD are constant for all shells as expected for a multilayer vesicle.

See the *core_shell_sphere* model for more documentation.

The 1D scattering intensity is calculated in the following way (Guinier, 1955)

$$P(q) = \text{scale} \cdot \frac{\phi}{V(R_N)} F^2(q) + \text{background}$$

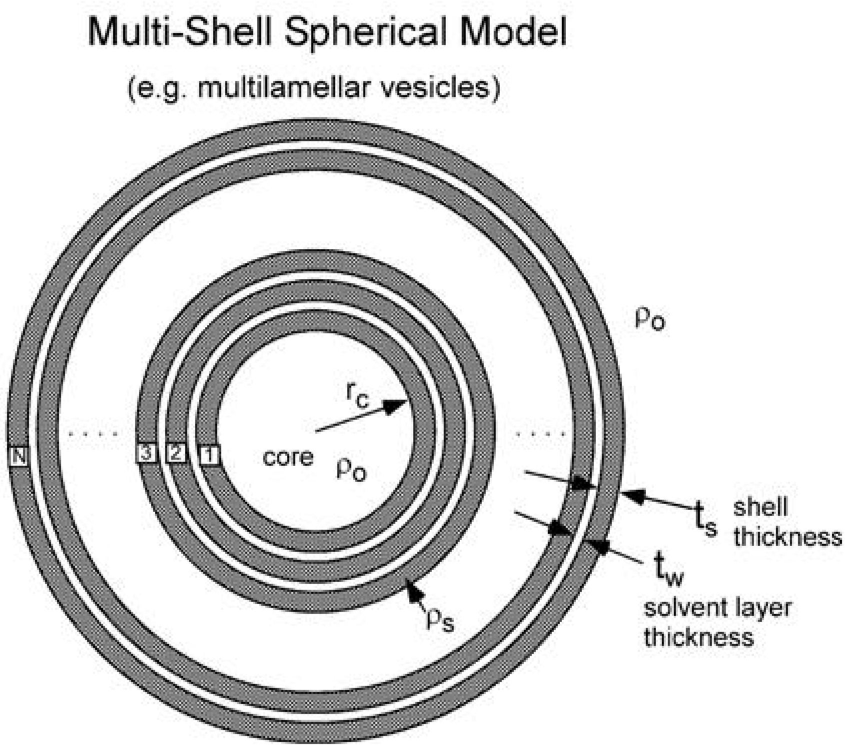

Multi-Shell Spherical Model
(e.g. multilamellar vesicles)
ρ_o
r_c
core ρ_o
N
3
2
1
t_s shell thickness
t_w solvent layer thickness
ρ_s

Fig. 1.82: Geometry of the multilayer_vesicle model.

where

$$F(q) = (\rho_{\text{shell}} - \rho_{\text{solv}}) \sum_{i=1}^{N} \left[3V(r_i) \frac{\sin(qr_i) - qr_i \cos(qr_i)}{(qr_i)^3} - 3V(R_i) \frac{\sin(qR_i) - qR_i \cos(qR_i)}{(qR_i)^3} \right]$$

for

$$r_i = r_c + (i-1)(t_s + t_w) \text{ solvent radius before shell } i$$
$$R_i = r_i + t_s \text{ shell radius for shell } i$$

ϕ is the volume fraction of particles, $V(r)$ is the volume of a sphere of radius r, r_c is the radius of the core, t_s is the thickness of the shell, t_w is the thickness of the solvent layer between the shells, ρ_{shell} is the scattering length density of a shell, and ρ_{solv} is the scattering length density of the solvent.

USAGE NOTES

- The outer-most shell radius R_N is used as the effective radius for $P(Q)$ when $P(Q) * S(Q)$ is applied. calculations rather slow.
- The number of shells is always rounded to an integer value as a non interger number of layers is not physical.
- Thus Polydispersity should only be applied to number of shells **VERY CAREFULLY**. A possible legitimate use would be for mixed systems in which some vesicles have 1 shell, some have 2, etc. A polydispersity on N can be used to model the data by using the "array distriubtion" feature. First create a file such as *shell_dist.txt* containing the relative portion of each vesicle size:

```
1 20
2  4
3  1
```

  Turn on polydispersity and select an array distribution for the *n_shells* parameter. Choose the above *shell_dist.txt* file, and the model will be computed with 80% 1-shell vesicles, 16% 2-shell vesicles and 4% 3-shell vesicles.

- This is a highly non-linear, highly oscillatory (especially around the q-values that correspond to the repeat distance of the layers), model function complicated by the fact that the number of water/shell pairs must physically be an integer value, although the optimization treats it as a floating point value. Thus it may be that the resolution interpolation is not sufficiently fine grained in certain cases. Please report any such occurences to the SasView team. Generally, for the best possible experience:
- Start with the best possible guess
- Using a priori knowledge, hold as many parameters fixed as possible
- if N=1, tw (water thickness) must by definition be zero. Both N and tw should be fixed during fitting.
- If N>1, use constraints to keep N > 1
- Because N only really moves in integer steps, it may get "stuck" if the optimizer step size is too small so care should be taken If you experience problems with this please contact the SasView team and let them know the issue preferably with example data and model which fail to converge.

The 2D scattering intensity is the same as 1D, regardless of the orientation of the q vector which is defined as:

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

For information about polarised and magnetic scattering, see the *Polarisation/Magnetic Scattering* documentation.

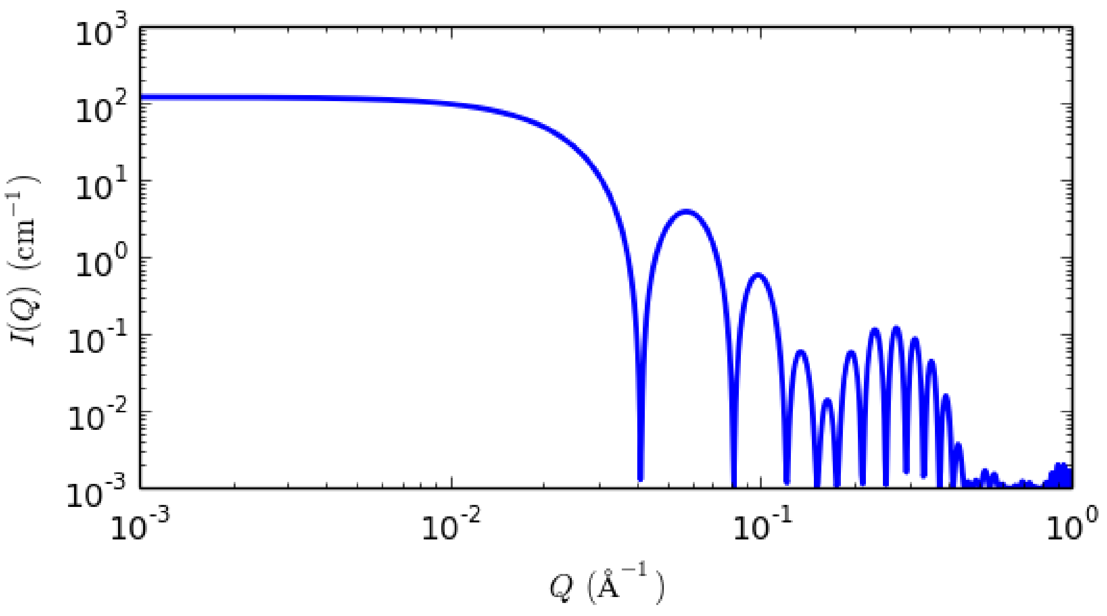


Fig. 1.83: 1D plot corresponding to the default parameters of the model.

References

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Converted to sasmodels by:** Piotr Rozyczko **Date:** Feb 24, 2016
- **Last Modified by:** Paul Kienzle **Date:** Feb 7, 2017
- **Last Reviewed by:** Paul Butler **Date:** March 12, 2017

onion

Onion shell model with constant, linear or exponential density

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
sld_core	Core scattering length density	$10^{-6}Å^{-2}$	1
radius_core	Radius of the core	Å	200
sld_solvent	Solvent scattering length density	$10^{-6}Å^{-2}$	6.4
n_shells	number of shells	None	1
sld_in[n_shells]	scattering length density at the inner radius of shell k	$10^{-6}Å^{-2}$	1.7
sld_out[n_shells]	scattering length density at the outer radius of shell k	$10^{-6}Å^{-2}$	2
thickness[n_shells]	Thickness of shell k	Å	40
A[n_shells]	Decay rate of shell k	None	1

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

This model provides the form factor, $P(q)$, for a multi-shell sphere where the scattering length density (SLD) of each shell is described by an exponential, linear, or constant function. The form factor is normalized by the volume of the sphere where the SLD is not identical to the SLD of the solvent. We currently provide up to 9 shells with this model.

NB: *radius* represents the core radius r_0 and *thickness[k]* represents the thickness of the shell, $r_{k+1} - r_k$.

Definition

The 1D scattering intensity is calculated in the following way

$$P(q) = [f]^2 / V_\text{particle}$$

where

$$f = f_\text{core} + \left(\sum_{\text{shell}=1}^N f_\text{shell}\right) + f_\text{solvent}$$

The shells are spherically symmetric with particle density $\rho(r)$ and constant SLD within the core and solvent, so

$$\begin{aligned}
f_\text{core} &= 4\pi\int_0^{r_\text{core}} \rho_\text{core}\frac{\sin(qr)}{qr}\, r^2\,\mathrm{d}r &&= 3\rho_\text{core}V(r_\text{core})\frac{j_1(qr_\text{core})}{qr_\text{core}} \\
f_\text{shell} &= 4\pi\int_{r_{\text{shell}-1}}^{r_\text{shell}} \rho_\text{shell}(r)\frac{\sin(qr)}{qr}\, r^2\,\mathrm{d}r \\
f_\text{solvent} &= 4\pi\int_{r_N}^\infty \rho_\text{solvent}\frac{\sin(qr)}{qr}\, r^2\,\mathrm{d}r &&= -3\rho_\text{solvent}V(r_N)\frac{j_1(qr_N)}{qr_N}
\end{aligned}$$

where the spherical bessel function j_1 is

$$j_1(x) = \frac{\sin(x)}{x^2} - \frac{\cos(x)}{x}$$

and the volume is $V(r) = \frac{4\pi}{3}r^3$. The volume of the particle is determined by the radius of the outer shell, so $V_\text{particle} = V(r_N)$.

Now lets consider the SLD of a shell defined by

$$\rho_\text{shell}(r) = \begin{cases} B\exp\left(A(r-r_{\text{shell}-1})/\Delta t_\text{shell}\right) + C & \text{for } A \neq 0 \\ \rho_\text{in} = \text{constant} & \text{for } A = 0 \end{cases}$$

An example of a possible SLD profile is shown below where ρ_in and Δt_shell stand for the SLD of the inner side of the k^th shell and the thickness of the k^th shell in the equation above, respectively.

For $A > 0$,

$$\begin{aligned}
f_\text{shell} &= 4\pi\int_{r_{\text{shell}-1}}^{r_\text{shell}} \left[B\exp\left(A(r-r_{\text{shell}-1})/\Delta t_\text{shell}\right) + C\right]\frac{\sin(qr)}{qr}\, r^2\,\mathrm{d}r \\
&= 3BV(r_\text{shell})e^A h(\alpha_\text{out},\beta_\text{out}) - 3BV(r_{\text{shell}-1})h(\alpha_\text{in},\beta_\text{in}) + 3CV(r_\text{shell})\frac{j_1(\beta_\text{out})}{\beta_\text{out}} - 3CV(r_{\text{shell}-1})\frac{j_1(\beta_\text{in})}{\beta_\text{in}}
\end{aligned}$$

for

$$B = \frac{\rho_{\text{out}} - \rho_{\text{in}}}{e^A - 1} \qquad C = \frac{\rho_{\text{in}} e^A - \rho_{\text{out}}}{e^A - 1}$$
$$\alpha_{\text{in}} = A \frac{r_{\text{shell}-1}}{\Delta t_{\text{shell}}} \qquad \alpha_{\text{out}} = A \frac{r_{\text{shell}}}{\Delta t_{\text{shell}}}$$
$$\beta_{\text{in}} = q r_{\text{shell}-1} \qquad \beta_{\text{out}} = q r_{\text{shell}}$$

where h is

$$h(x,y) = \frac{x\sin(y) - y\cos(y)}{(x^2+y^2)y} - \frac{(x^2-y^2)\sin(y) - 2xy\cos(y)}{(x^2+y^2)^2 y}$$

For $A \sim 0$, e.g., $A = -0.0001$, this function converges to that of the linear SLD profile with $\rho_{\text{shell}}(r) \approx A(r - r_{\text{shell}-1})/\Delta t_{\text{shell}}) + B$, so this case is equivalent to

$$\begin{aligned} f_{\text{shell}} &= 3V(r_{\text{shell}}) \frac{\Delta\rho_{\text{shell}}}{\Delta t_{\text{shell}}} \left[\frac{2\cos(qr_{\text{out}}) + qr_{\text{out}}\sin(qr_{\text{out}})}{(qr_{\text{out}})^4} \right] \\ &- 3V(r_{\text{shell}}) \frac{\Delta\rho_{\text{shell}}}{\Delta t_{\text{shell}}} \left[\frac{2\cos(qr_{\text{in}}) + qr_{\text{in}}\sin(qr_{\text{in}})}{(qr_{\text{in}})^4} \right] \\ &+ 3\rho_{\text{out}} V(r_{\text{shell}}) \frac{j_1(qr_{\text{out}})}{qr_{\text{out}}} - 3\rho_{\text{in}} V(r_{\text{shell}-1}) \frac{j_1(qr_{\text{in}})}{qr_{\text{in}}} \end{aligned}$$

For $A = 0$, the exponential function has no dependence on the radius (so that ρ_{out} is ignored in this case) and becomes flat. We set the constant to ρ_{in} for convenience, and thus the form factor contributed by the shells is

$$f_{\text{shell}} = 3\rho_{\text{in}} V(r_{\text{shell}}) \frac{j_1(qr_{\text{out}})}{qr_{\text{out}}} - 3\rho_{\text{in}} V(r_{\text{shell}-1}) \frac{j_1(qr_{\text{in}})}{qr_{\text{in}}}$$

The 2D scattering intensity is the same as $P(q)$ above, regardless of the orientation of the q vector which is defined as

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

NB: The outer most radius is used as the effective radius for $S(q)$ when $P(q)S(q)$ is applied.

References

L A Feigin and D I Svergun, *Structure Analysis by Small-Angle X-Ray and Neutron Scattering*, Plenum Press, New York, 1987.

polymer_micelle

Polymer micelle model

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
ndensity	Number density of micelles	$10^{15}cm^3$	8.94
v_core	Core volume	$Å^3$	62624
v_corona	Corona volume	$Å^3$	61940
sld_solvent	Solvent scattering length density	$10^{-6}Å^{-2}$	6.4
sld_core	Core scattering length density	$10^{-6}Å^{-2}$	0.34
sld_corona	Corona scattering length density	$10^{-6}Å^{-2}$	0.8
radius_core	Radius of core (must be >> rg)	Å	45
rg	Radius of gyration of chains in corona	Å	20
d_penetration	Factor to mimic non-penetration of Gaussian chains	None	1
n_aggreg	Aggregation number of the micelle	None	6

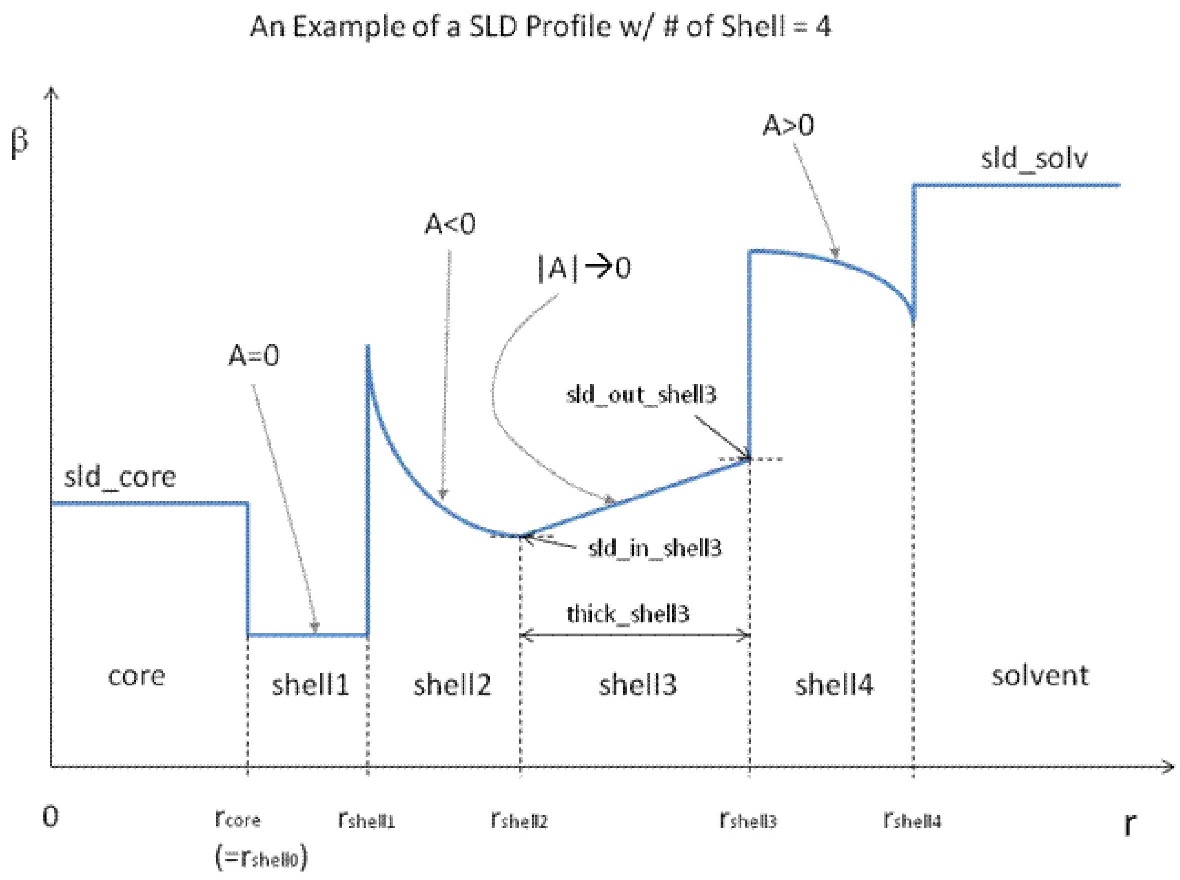


Fig. 1.84: Example of an onion model profile.

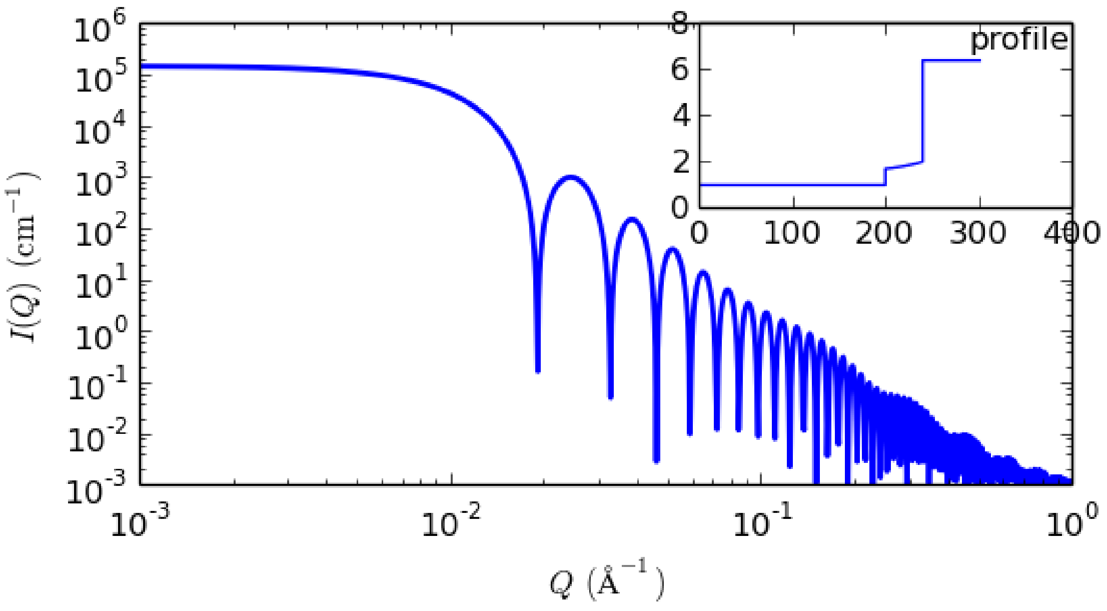


Fig. 1.85: 1D plot corresponding to the default parameters of the model.

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

This model provides the form factor, $P(q)$, for a micelle with a spherical core and Gaussian polymer chains attached to the surface, thus may be applied to block copolymer micelles. To work well the Gaussian chains must be much smaller than the core, which is often not the case. Please study the reference carefully.

Definition

The 1D scattering intensity for this model is calculated according to the equations given by Pedersen (Pedersen, 2000), summarised briefly here.

The micelle core is imagined as N = *n_aggreg* polymer heads, each of volume V_{core}, which then defines a micelle core of radius r = *r_core*, which is a separate parameter even though it could be directly determined. The Gaussian random coil tails, of gyration radius R_g, are imagined uniformly distributed around the spherical core, centred at a distance $r + d \cdot R_g$ from the micelle centre, where d = *d_penetration* is of order unity. A volume V_{corona} is defined for each coil. The model in detail seems to separately parametrise the terms for the shape of $I(Q)$ and the relative intensity of each term, so use with caution and check parameters for consistency. The spherical core is monodisperse, so it's intensity and the cross terms may have sharp oscillations (use q resolution smearing if needs be to help remove them).

$$\begin{aligned}
P(q) &= N^2\beta_s^2\Phi(qr)^2 + N\beta_c^2P_c(q) + 2N^2\beta_s\beta_cS_{sc}(q) + N(N-1)\beta_c^2S_{cc}(q)\\
\beta_s &= V_{\text{core}}(\rho_{\text{core}} - \rho_{\text{solvent}})\\
\beta_c &= V_{\text{corona}}(\rho_{\text{corona}} - \rho_{\text{solvent}})
\end{aligned}$$

where ρ_{core}, ρ_{corona} and ρ_{solvent} are the scattering length densities *sld_core*, *sld_corona* and *sld_solvent*. For the spherical core of radius r

$$\Phi(qr) = \frac{\sin(qr) - qr\cos(qr)}{(qr)^3}$$

whilst for the Gaussian coils

$$\begin{aligned}
P_c(q) &= 2[\exp(-Z) + Z - 1]/Z^2\\
Z &= (qR_g)^2
\end{aligned}$$

The sphere to coil (core to corona) and coil to coil (corona to corona) cross terms are approximated by:

$$\begin{aligned}
S_{sc}(q) &= \Phi(qr)\psi(Z)\frac{\sin(q(r + d\cdot R_g))}{q(r + d\cdot R_g)}\\
S_{cc}(q) &= \psi(Z)^2\left[\frac{\sin(q(r + d\cdot R_g))}{q(r + d\cdot R_g)}\right]^2\\
\psi(Z) &= \frac{[1 - \exp^{-Z}]}{Z}
\end{aligned}$$

Validation

$P(q)$ above is multiplied by *ndensity*, and a units conversion of 10^{-13}, so *scale* is likely 1.0 if the scattering data is in absolute units. This model has not yet been independently validated.

References

J Pedersen, *J. Appl. Cryst.*, 33 (2000) 637-640

- **Modified by:** Richard Heenan **Date:** March 20, 2016
- **Verified by:** Paul Kienzle **Date:** November 29, 2017
- **Description modified by:** Paul Kienzle **Date:** November 29, 2017
- **Description reviewed by:** Steve King **Date:** November 30, 2017

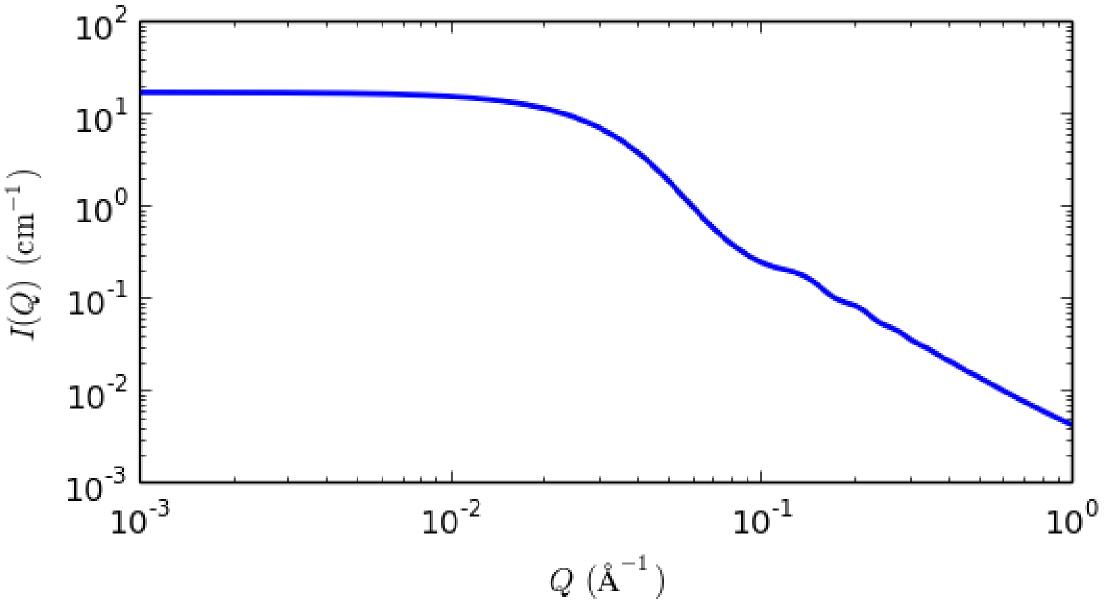


Fig. 1.86: 1D plot corresponding to the default parameters of the model.

raspberry

Calculates the form factor, *P(q)*, for a 'Raspberry-like' structure where there are smaller spheres at the surface of a larger sphere, such as the structure of a Pickering emulsion.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
sld_lg	large particle scattering length density	10^{-6}Å^{-2}	-0.4
sld_sm	small particle scattering length density	10^{-6}Å^{-2}	3.5
sld_solvent	solvent scattering length density	10^{-6}Å^{-2}	6.36
volfraction_lg	volume fraction of large spheres	None	0.05
volfraction_sm	volume fraction of small spheres	None	0.005
surface_fraction	fraction of small spheres at surface	None	0.4
radius_lg	radius of large spheres	Å	5000
radius_sm	radius of small spheres	Å	100
penetration	fractional penetration depth of small spheres into large sphere	Å	0

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

The figure below shows a schematic of a large droplet surrounded by several smaller particles forming a structure similar to that of Pickering emulsions.

In order to calculate the form factor of the entire complex, the self-correlation of the large droplet, the self-correlation of the particles, the correlation terms between different particles and the cross terms between large droplet and small particles all need to be calculated.

Consider two infinitely thin shells of radii R_1 and R_2 separated by distance r. The general structure of the equation is then the form factor of the two shells multiplied by the phase factor that accounts for the separation of their centers.

$$S(q) = \frac{\sin(qR_1)}{qR_1}\frac{\sin(qR_2)}{qR_2}\frac{\sin(qr)}{qr}$$

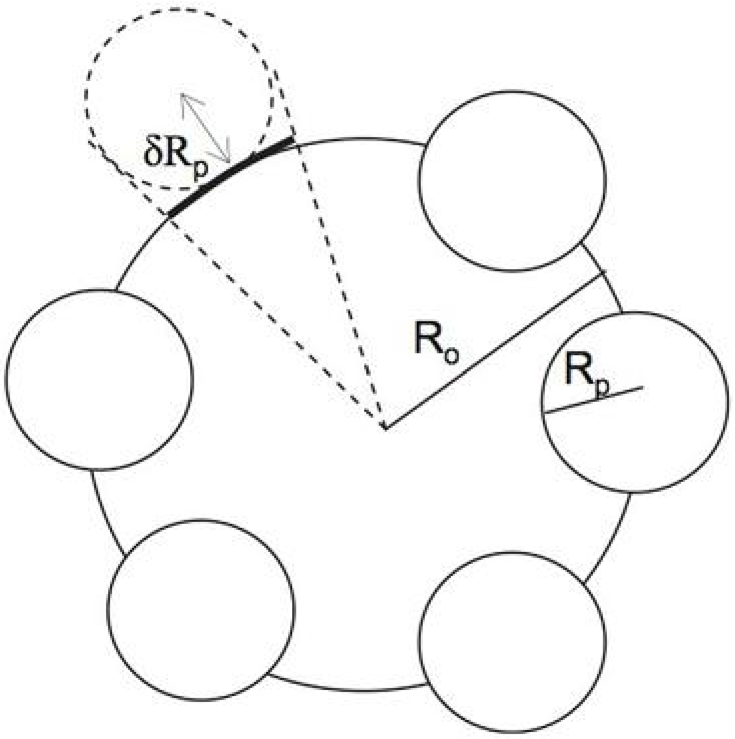


Fig. 1.87: Schematic of the raspberry model

In this case, the large droplet and small particles are solid spheres rather than thin shells. Thus the two terms must be integrated over R_L and R_S respectively using the weighting function of a sphere. We then obtain the functions for the form of the two spheres:

$$\Psi_L = \int_0^{R_L} (4\pi R_L^2)\frac{\sin(qR_L)}{qR_L}dR_L = \frac{3[\sin(qR_L) - qR_L\cos(qR_L)]}{(qR_L)^2}$$

$$\Psi_S = \int_0^{R_S} (4\pi R_S^2)\frac{\sin(qR_S)}{qR_S}dR_S = \frac{3[\sin(qR_S) - qR_L\cos(qR_S)]}{(qR_S)^2}$$

The cross term between the large droplet and small particles is given by:

$$S_{LS} = \Psi_L\Psi_S\frac{\sin(q(R_L + \delta R_S))}{q(R_L + \delta\ R_S)}$$

and the self term between small particles is given by:

$$S_{SS} = \Psi_S^2\left[\frac{\sin(q(R_L + \delta R_S))}{q(R_L + \delta\ R_S)}\right]^2$$

The number of small particles per large droplet, N_p, is given by:

$$N_p = \frac{\phi_S\phi_{\text{surface}}V_L}{\phi_L V_S}$$

where ϕ_S is the volume fraction of small particles in the sample, ϕ_{surface} is the fraction of the small particles that are adsorbed to the large droplets, ϕ_L is the volume fraction of large droplets in the sample, and V_S and V_L are the volumes of individual small particles and large droplets respectively.

The form factor of the entire complex can now be calculated including the excess scattering length densities of the components $\Delta\rho_L$ and $\Delta\rho_S$, where $\Delta\rho_x = |\rho_x - \rho_{\text{solvent}}|$:

$$P_{LS} = \frac{1}{M^2}\left[(\Delta\rho_L)^2V_L^2\Psi_L^2 + N_p(\Delta\rho_S)^2V_S^2\Psi_S^2 + N_p(1-N_p)(\Delta\rho_S)^2V_S^2S_{SS} + 2N_p\Delta\rho_L\Delta\rho_SV_LV_SS_{LS}\right]$$

where M is the total scattering length of the whole complex :

$$M = \Delta\rho_L V_L + N_p \Delta\rho_S V_S$$

In a real system, there will ususally be an excess of small particles such that some fraction remain unbound. Therefore the overall scattering intensity is given by:

$$I(Q) = I_{LS}(Q) + I_S(Q) = (\phi_L(\Delta\rho_L)^2 V_L + \phi_S \phi_{\text{surface}} N_p (\Delta\rho_S)^2 V_S) P_{LS} + \phi_S (1 - \phi_{\text{surface}})(\Delta\rho_S)^2 V_S \Psi_S^2$$

A useful parameter to extract is the fraction of the surface area of the large droplets that is covered by small particles. This can be calculated from the model parameters as:

$$\chi = \frac{4\phi_L \phi_{\text{surface}}(R_L + \delta R_S)}{\phi_L R_S}$$

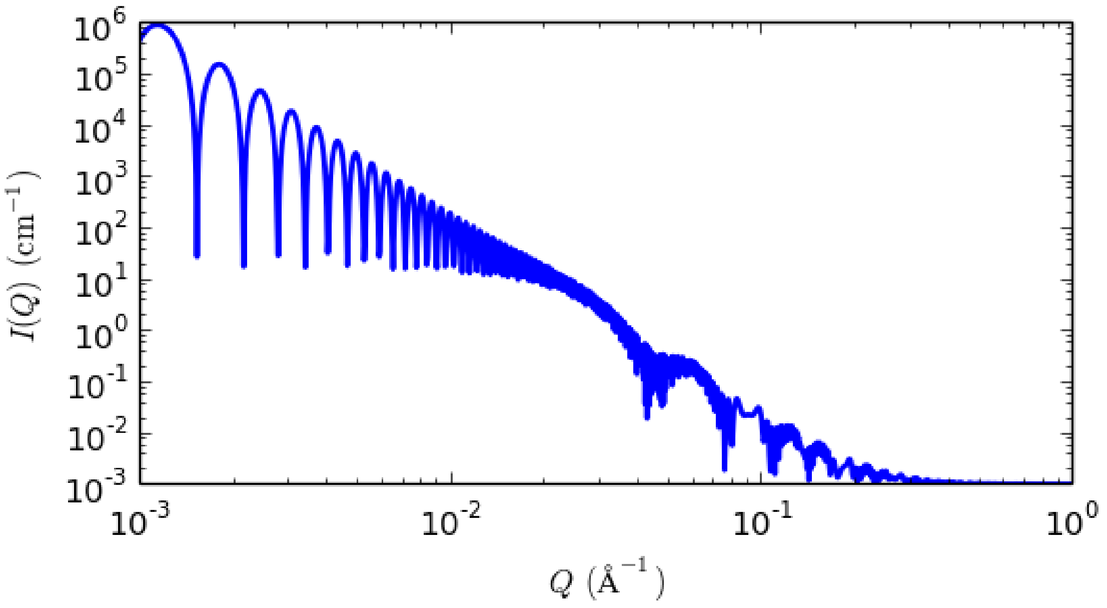


Fig. 1.88: 1D plot corresponding to the default parameters of the model.

References

K Larson-Smith, A Jackson, and D C Pozzo, *Small angle scattering model for Pickering emulsions and raspberry particles*, *Journal of Colloid and Interface Science*, 343(1) (2010) 36-41

- **Author:** Andrew Jackson **Date:** 2008
- **Modified by:** Andrew Jackson **Date:** March 20, 2016
- **Reviewed by:** Andrew Jackson **Date:** March 20, 2016

sphere

Spheres with uniform scattering length density

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
sld	Layer scattering length density	10^{-6}Å^{-2}	1
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	6
radius	Sphere radius	Å	50

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

For information about polarised and magnetic scattering, see the *Polarisation/Magnetic Scattering* documentation.

Definition

The 1D scattering intensity is calculated in the following way (Guinier, 1955)

$$I(q) = \frac{\text{scale}}{V} \cdot \left[3V(\Delta\rho) \cdot \frac{\sin(qr) - qr\cos(qr))}{(qr)^3}\right]^2 + \text{background}$$

where *scale* is a volume fraction, V is the volume of the scatterer, r is the radius of the sphere and *background* is the background level. *sld* and *sld_solvent* are the scattering length densities (SLDs) of the scatterer and the solvent respectively, whose difference is $\Delta\rho$.

Note that if your data is in absolute scale, the *scale* should represent the volume fraction (which is unitless) if you have a good fit. If not, it should represent the volume fraction times a factor (by which your data might need to be rescaled).

The 2D scattering intensity is the same as above, regardless of the orientation of $\vec{q}$.

Validation

Validation of our code was done by comparing the output of the 1D model to the output of the software provided by the NIST (Kline, 2006).

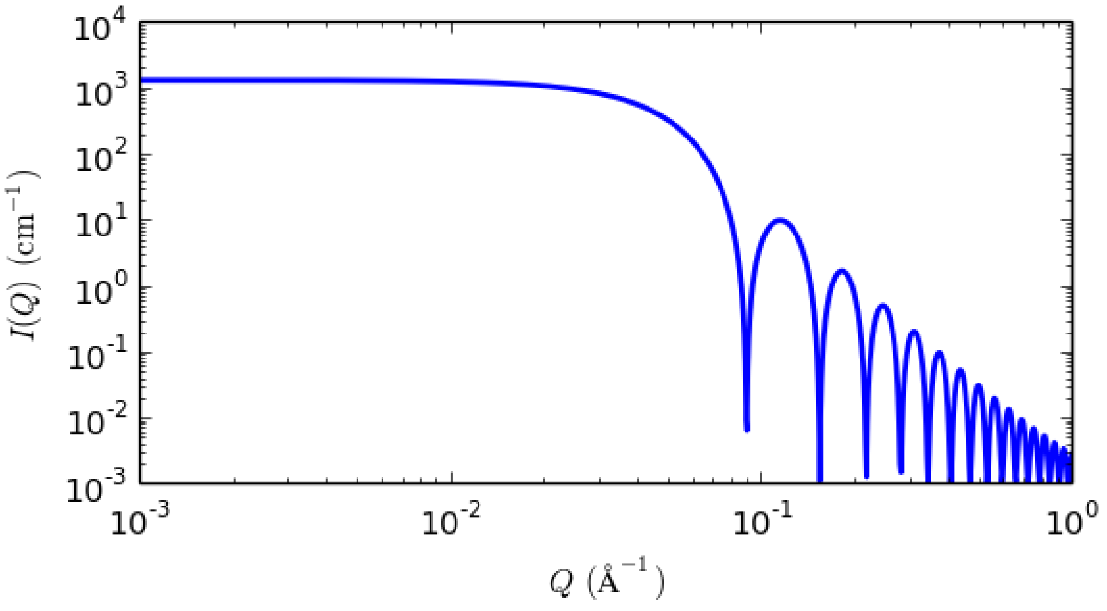


Fig. 1.89: 1D plot corresponding to the default parameters of the model.

References

A Guinier and G. Fournet, *Small-Angle Scattering of X-Rays*, John Wiley and Sons, New York, (1955)

- **Last Reviewed by:** S King and P Parker **Date:** 2013/09/09 and 2014/01/06

spherical_sld

Sperical SLD intensity calculation

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
n_shells	number of shells	None	1
sld_solvent	solvent sld	10^{-6}Å^{-2}	1
sld[n_shells]	sld of the shell	10^{-6}Å^{-2}	4.06
thickness[n_shells]	thickness shell	Å	100
interface[n_shells]	thickness of the interface	Å	50
shape[n_shells]	interface shape	None	0
nu[n_shells]	interface shape exponent	None	2.5
n_steps	number of steps in each interface (must be an odd integer)	None	35

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

Similarly to the onion, this model provides the form factor, $P(q)$, for a multi-shell sphere, where the interface between the each neighboring shells can be described by the error function, power-law, or exponential functions. The scattering intensity is computed by building a continuous custom SLD profile along the radius of the particle. The SLD profile is composed of a number of uniform shells with interfacial shells between them.

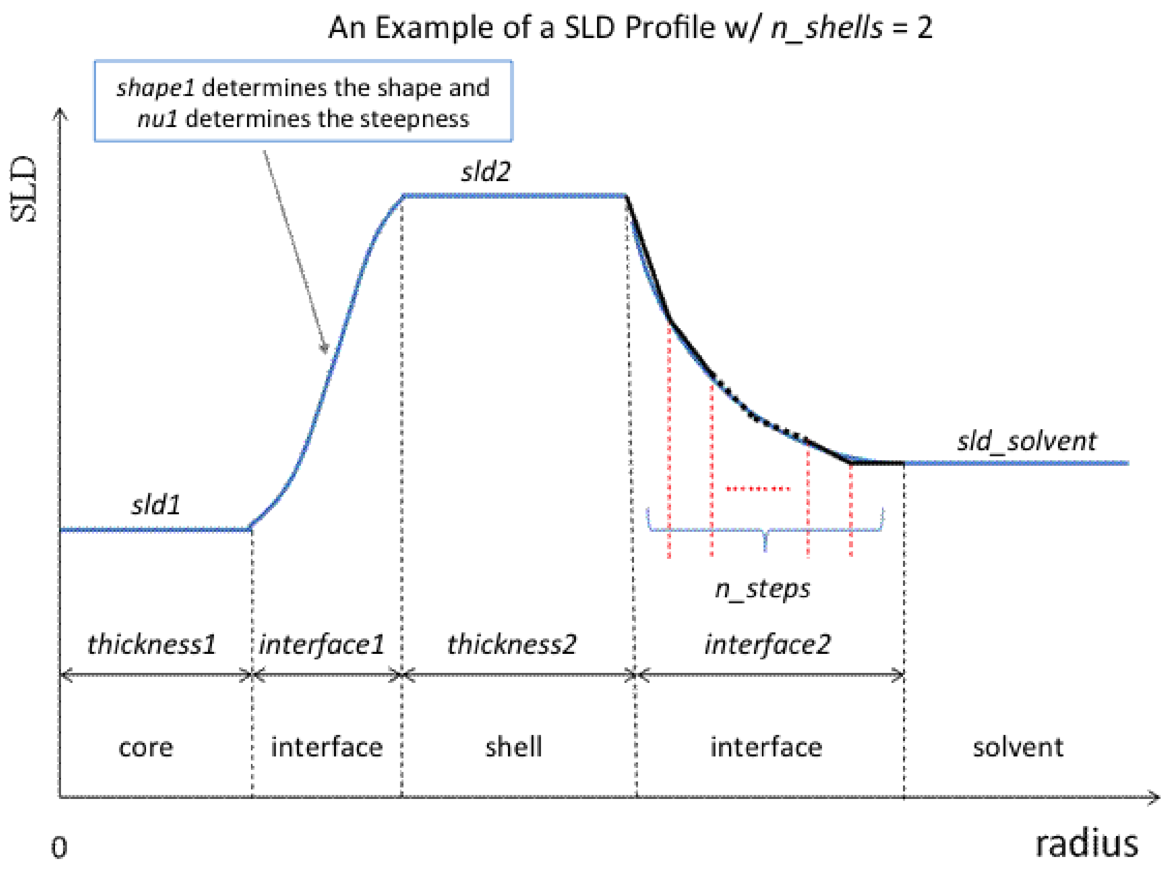


Fig. 1.90: Example SLD profile

Unlike the <onion> model (using an analytical integration), the interfacial shells here are sub-divided and numerically integrated assuming each sub-shell is described by a line function, with *n_steps* sub-shells per interface. The form factor is normalized by the total volume of the sphere.

Interface shapes are as follows:

0: erf(νz)

1: Rpow(z^ν)

2: Lpow(z^ν)

3: Rexp($-\nu z$)

4: Lexp($-\nu z$)

The form factor $P(q)$ in 1D is calculated by:

$$P(q) = \frac{f^2}{V_\text{particle}} \text{ where } f = f_\text{core} + \sum_{\text{inter}_i=0}^{N} f_{\text{inter}_i} + \sum_{\text{flat}_i=0}^{N} f_{\text{flat}_i} + f_\text{solvent}$$

For a spherically symmetric particle with a particle density $\rho_x(r)$ the sld function can be defined as:

$$f_x = 4\pi \int_0^\infty \rho_x(r) \frac{\sin(qr)}{qr^2} r^2 dr$$

so that individual terms can be calculated as follows:

$$f_\text{core} = 4\pi \int_0^{r_\text{core}} \rho_\text{core} \frac{\sin(qr)}{qr} r^2 dr = 3\rho_\text{core} V(r_\text{core}) \Big[\frac{\sin(qr_\text{core}) - qr_\text{core}\cos(qr_\text{core})}{qr_\text{core}^3} \Big]$$

$$f_{\text{inter}_i} = 4\pi \int_{\Delta t_{\text{inter}_i}} \rho_{\text{inter}_i} \frac{\sin(qr)}{qr} r^2 dr$$

$$f_{\text{shell}_i} = 4\pi \int_{\Delta t_{\text{inter}_i}} \rho_{\text{flat}_i} \frac{\sin(qr)}{qr} r^2 dr = 3\rho_{\text{flat}_i} V(r_{\text{inter}_i} + \Delta t_{\text{inter}_i}) \Big[\frac{\sin(qr_{\text{inter}_i} + \Delta t_{\text{inter}_i}) - q(r_{\text{inter}_i} + \Delta t_{\text{inter}_i})\cos(q(r_{\text{inter}_i} + \Delta t_{\text{inte}}}{q(r_{\text{inter}_i} + \Delta t_{\text{inter}_i})^3}$$

$$f_\text{solvent} = 4\pi \int_{r_N}^\infty \rho_\text{solvent} \frac{\sin(qr)}{qr} r^2 dr = 3\rho_\text{solvent} V(r_N) \Big[\frac{\sin(qr_N) - qr_N\cos(qr_N)}{qr_N^3} \Big]$$

Here we assumed that the SLDs of the core and solvent are constant in r. The SLD at the interface between shells, ρ_{inter_i} is calculated with a function chosen by an user, where the functions are

Exp:

$$\rho_{inter_i}(r) = \begin{cases} B\exp\Big(\frac{\pm A(r - r_{\text{flat}_i})}{\Delta t_{\text{inter}_i}}\Big) + C & \text{for } A \neq 0 \\ B\Big(\frac{(r - r_{\text{flat}_i})}{\Delta t_{\text{inter}_i}}\Big) + C & \text{for } A = 0 \end{cases}$$

Power-Law

$$\rho_{inter_i}(r) = \begin{cases} \pm B\Big(\frac{(r - r_{\text{flat}_i})}{\Delta t_{\text{inter}_i}}\Big)^A + C & \text{for } A \neq 0 \\ \rho_{\text{flat}_{i+1}} & \text{for } A = 0 \end{cases}$$

Erf:

$$\rho_{inter_i}(r) = \begin{cases} B\text{erf}\Big(\frac{A(r - r_{\text{flat}_i})}{\sqrt{2}\Delta t_{\text{inter}_i}}\Big) + C & \text{for } A \neq 0 \\ B\Big(\frac{(r - r_{\text{flat}_i})}{\Delta t_{\text{inter}_i}}\Big) + C & \text{for } A = 0 \end{cases}$$

The functions are normalized so that they vary between 0 and 1, and they are constrained such that the SLD is continuous at the boundaries of the interface as well as each sub-shell. Thus B and C are determined.

Once ρ_{inter_i} is found at the boundary of the sub-shell of the interface, we can find its contribution to the form factor $P(q)$

$$f_{\text{inter}_i} = 4\pi \int_{\Delta t_{\text{inter}_i}} \rho_{\text{inter}_i} \frac{\sin(qr)}{qr} r^2 dr = 4\pi \sum_{j=}^{n_\text{ste}}$$

$$\approx 4\pi \sum_{j=1}^{n_\text{steps}} \Big[3(\rho_{\text{inter}_i}(r_{j+1}) - \rho_{\text{inter}_i}(r_j) V(r_j) \Big[\frac{r_j^2 \beta_\text{out}^2 \sin(\beta_\text{out}) - (\beta_\text{out}^2 - 2)\cos(\beta_\text{out})}{\beta_\text{out}^4} \Big]$$

$$- 3(\rho_{\text{inter}_i}(r_{j+1}) - \rho_{\text{inter}_i}(r_j) V(r_{j-1}) \Big[\frac{r_{j-1}^2 \sin(\beta_\text{in}) - (\beta_\text{in}^2 - 2)\cos(\beta_\text{in})}{\beta_\text{in}^4} \Big]$$

$$+ 3\rho_{\text{inter}_i}(r_{j+1}) V(r_j) \Big[\frac{\sin(\beta_\text{out}) - \cos(\beta_\text{out})}{\beta_\text{out}^4} \Big] - 3\rho_{\text{inter}_i}(r_j) V(r_j) \Big[\frac{\sin(\beta_\text{in}) - \cos(\beta_\text{in})}{\beta_\text{in}^4} \Big] \Big]$$

where

$$V(a) = \frac{4\pi}{3}a^3$$

$$a_{\text{in}} \sim \frac{r_j}{r_{j+1} - r_j}, a_{\text{out}} \sim \frac{r_{j+1}}{r_{j+1} - r_j}$$

$$\beta_{\text{in}} = qr_j, \qquad \beta_{\text{out}} = qr_{j+1}$$

We assume $\rho_{\text{inter}_j}(r)$ is approximately linear within the sub-shell j.

Finally the form factor can be calculated by

$$P(q) = \frac{[f]^2}{V_{\text{particle}}} \text{ where } V_{\text{particle}} = V(r_{\text{shell}_N})$$

For 2D data the scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

Note: The outer most radius is used as the effective radius for $S(Q)$ when $P(Q) * S(Q)$ is applied.

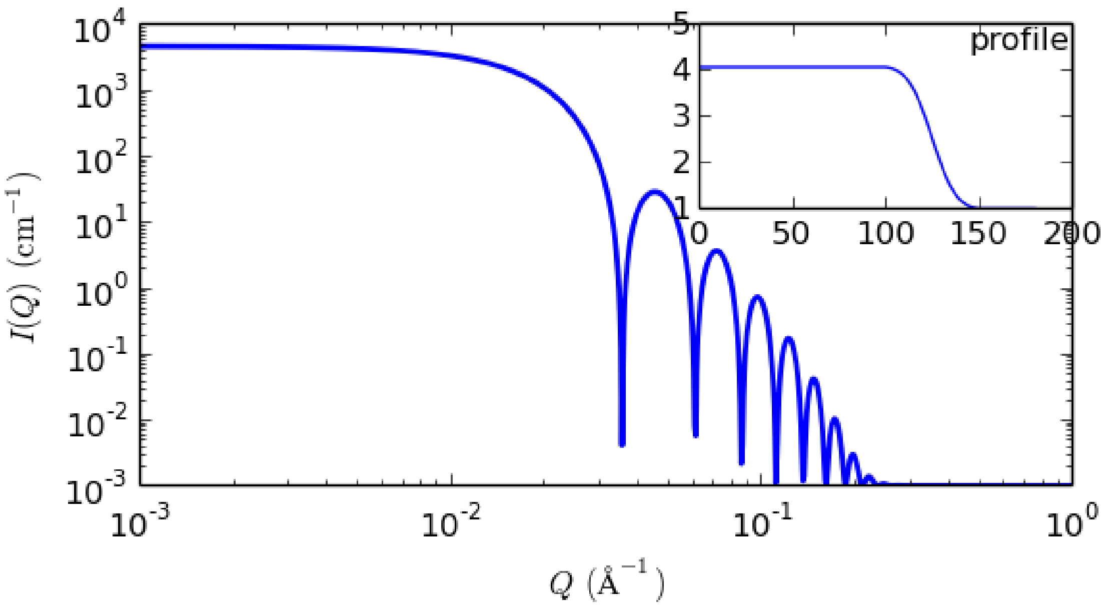


Fig. 1.91: 1D plot corresponding to the default parameters of the model.

References

Authorship and Verification

- **Author:** Jae-Hie Cho **Date:** Nov 1, 2010
- **Last Modified by:** Paul Kienzle **Date:** Dec 20, 2016
- **Last Reviewed by:** Paul Butler **Date:** September 8, 2018

vesicle

Vesicle model representing a hollow sphere

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
sld	vesicle shell scattering length density	10^{-6}Å^{-2}	0.5
sld_solvent	solvent scattering length density	10^{-6}Å^{-2}	6.36
volfraction	volume fraction of shell	None	0.05
radius	vesicle core radius	Å	100
thickness	vesicle shell thickness	Å	30

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

This model provides the form factor, *P(q)*, for an unilamellar vesicle and is effectively identical to the hollow sphere reparameterized to be more intuitive for a vesicle and normalizing the form factor by the volume of the shell. The 1D scattering intensity is calculated in the following way (Guinier,1955[1])

$$P(q) = \frac{\phi}{V_{\text{shell}}}\left[\frac{3V_{\text{core}}(\rho_{\text{solvent}} - \rho_{\text{shell}})j_1(qR_{\text{core}})}{qR_{\text{core}}} + \frac{3V_{\text{tot}}(\rho_{\text{shell}} - \rho_{\text{solvent}})j_1(qR_{\text{tot}})}{qR_{\text{tot}}}\right]^2 + \text{background}$$

where ϕ is the volume fraction of shell material, V_{shell} is the volume of the shell, V_{cor} is the volume of the core, V_{tot} is the total volume, R_{core} is the radius of the core, R_{tot} is the outer radius of the shell, ρ_{solvent} is the scattering length density of the solvent (which is the same as for the core in this case), ρ_{scale} is the scattering length density of the shell, background is a flat background level (due for example to incoherent scattering in the case of neutrons), and j_1 is the spherical bessel function $j_1 = (\sin(x) - x\cos(x))/x^2$.

The functional form is identical to a "typical" core-shell structure, except that the scattering is normalized by the volume that is contributing to the scattering, namely the volume of the shell alone, the scattering length density of the core is fixed the same as that of the solvent, the scale factor when the data are on an absolute scale is equivalent to the volume fraction of material in the shell rather than the entire core+shell sphere, and the parameterization is done in terms of the core radius = R_{core} and the shell thickness = $R_{\text{tot}} - R_{\text{core}}$.

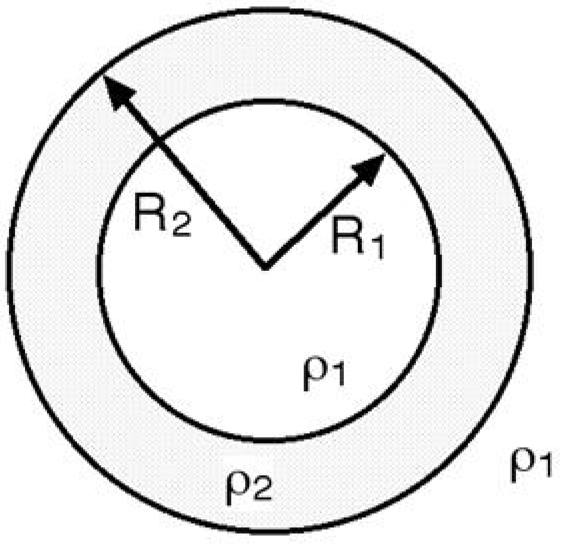


Fig. 1.92: Vesicle geometry.

The 2D scattering intensity is the same as *P(q)* above, regardless of the orientation of the *q* vector which is defined as

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

NB: The outer most radius (= *radius* + *thickness*) is used as the effective radius for *S(Q)* when *P(Q)* * *S(Q)* is applied.

[1] A Guinier and G. Fournet, *Small-Angle Scattering of X-Rays*, John Wiley and Sons, New York, (1955)

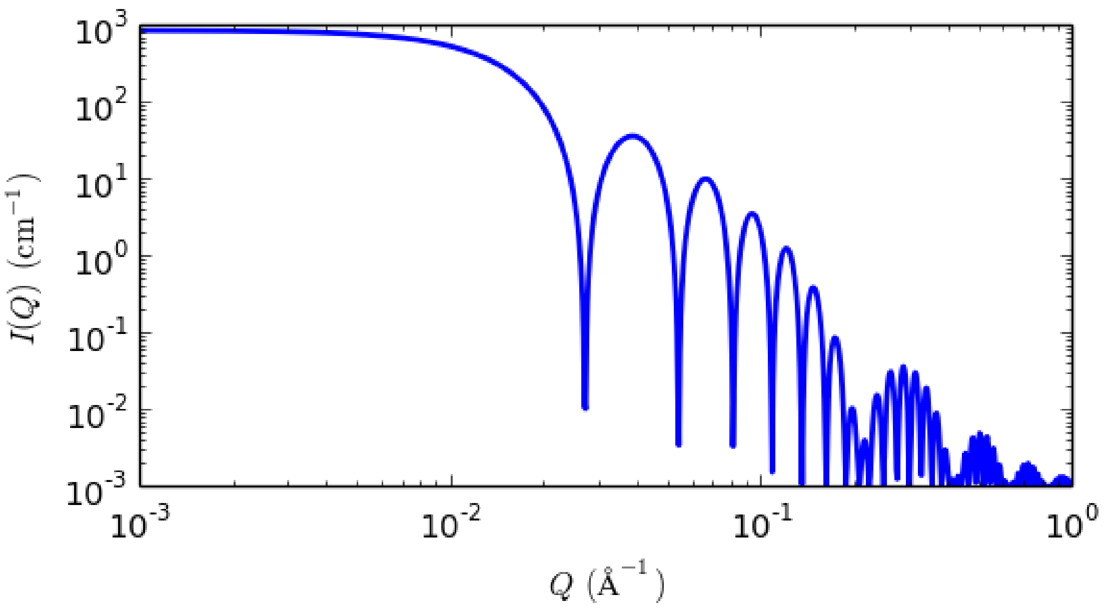


Fig. 1.93: 1D plot corresponding to the default parameters of the model.

References

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Last Modified by:** Paul Butler **Date:** March 20, 2016
- **Last Reviewed by:** Paul Butler **Date:** September 7, 2018

1.1.7 Shape-Independent Functions

be_polyelectrolyte

Polyelectrolyte with the RPA expression derived by Borue and Erukhimovich

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
contrast_factor	Contrast factor of the polymer	barns	10
bjerrum_length	Bjerrum length	Å	7.1
virial_param	Virial parameter	$\mathrm{Å}^3$	12
monomer_length	Monomer length	Å	10
salt_concentration	Concentration of monovalent salt	mol/L	0
ionization_degree	Degree of ionization	None	0.05
polymer_concentration	Polymer molar concentration	mol/L	0.7

The returned value is scaled to units of $\mathrm{cm}^{-1}\ \mathrm{sr}^{-1}$, absolute scale.

Note: Please read the Validation section below.

Definition This model calculates the structure factor of a polyelectrolyte solution with the RPA expression derived by Borue and Erukhimovich[1]. Note however that the fitting procedure here does not follow the notation in that reference as 's' and 't' are **not** decoupled. Instead the scattering intensity $I(q)$ is calculated as

$$I(q) = K\frac{q^2+k^2}{4\pi L_b\alpha^2}\frac{1}{1+r_0^4(q^2+k^2)(q^2-12hC_a/b^2)} + background$$
$$k^2 = 4\pi L_b(2C_s+\alpha C_a)$$
$$r_0^2 = \frac{b}{\alpha\sqrt{C_a}48\pi L_b}$$

where

K is the contrast factor for the polymer which is defined differently than in other models and is given in barns where 1 $barn = 10^{-24}\ cm^2$. K is defined as:

$$K = a^2$$
$$a = b_p - (v_p/v_s)b_s$$

where:

- b_p and b_s are **sum of the scattering lengths of the atoms** constituting the polymer monomer and the solvent molecules, respectively.
- v_p and v_s are the partial molar volume of the polymer and the solvent, respectively.
- L_b is the Bjerrum length (Å) - **Note:** This parameter needs to be kept constant for a given solvent and temperature!
- h is the virial parameter (Å^3) - **Note:** See[1] for the correct interpretation of this parameter. It incorporates second and third virial coefficients and can be *negative*.
- b is the monomer length (Å).
- C_s is the concentration of monovalent salt(1/Å^3 - internally converted from mol/L).
- α is the degree of ionization (the ratio of charged monomers to the total number of monomers)
- C_a is the polymer molar concentration (1/Å^3 - internally converted from mol/L)
- $background$ is the incoherent background.

For 2D data the scattering intensity is calculated in the same way as 1D, where the $\vec{q}$ vector is defined as

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

Validation

As of the last revision, this code is believed to be correct. However it needs further validation and should be used with caution at this time. The history of this code goes back to a 1998 implementation. It was recently noted that in that implementation, while both the polymer concentration and salt concentration were converted from experimental units of mol/L to more dimensionally useful units of 1/Å^3, only the converted version of the polymer concentration was actually being used in the calculation while the unconverted salt concentration (still in apparent units of mol/L) was being used. This was carried through to Sasmodels as used for SasView 4.1 (though the line of code converting the salt concentration to the new units was removed somewhere along the line). Simple dimensional analysis of the calculation shows that the converted salt concentration should be used, which the original code suggests was the intention, so this has now been corrected (for SasView 4.2). Once better validation has been performed this note will be removed.

References

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Last Modified by:** Paul Butler **Date:** September 25, 2018
- **Last Reviewed by:** Paul Butler **Date:** September 25, 2018

[1] V Y Borue, I Y Erukhimovich, *Macromolecules*, 21 (1988) 3240

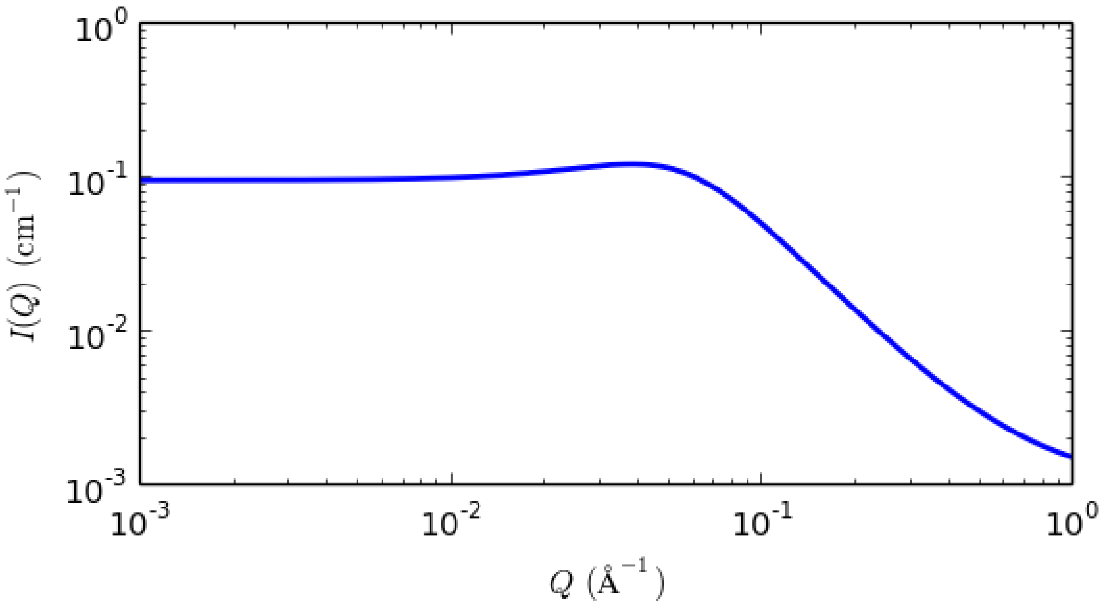


Fig. 1.94: 1D plot corresponding to the default parameters of the model.

broad_peak

Broad Lorentzian type peak on top of a power law decay

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
porod_scale	Power law scale factor	None	1e-05
porod_exp	Exponent of power law	None	3
lorentz_scale	Scale factor for broad Lorentzian peak	None	10
lorentz_length	Lorentzian screening length	Å	50
peak_pos	Peak position in q	Å^{-1}	0.1
lorentz_exp	Exponent of Lorentz function	None	2

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

This model calculates an empirical functional form for SAS data characterized by a broad scattering peak. Many SAS spectra are characterized by a broad peak even though they are from amorphous soft materials. For example, soft systems that show a SAS peak include copolymers, polyelectrolytes, multiphase systems, layered structures, etc.

The d-spacing corresponding to the broad peak is a characteristic distance between the scattering inhomogeneities (such as in lamellar, cylindrical, or spherical morphologies, or for bicontinuous structures).

The scattering intensity $I(q)$ is calculated as

$$I(q) = \frac{A}{q^n} + \frac{C}{1 + (|q - q_0|\xi)^m} + B$$

Here the peak position is related to the d-spacing as $q_0 = 2\pi/d_0$.

A is the Porod law scale factor, n the Porod exponent, C is the Lorentzian scale factor, m the exponent of q, ξ the screening length, and B the flat background.

For 2D data the scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

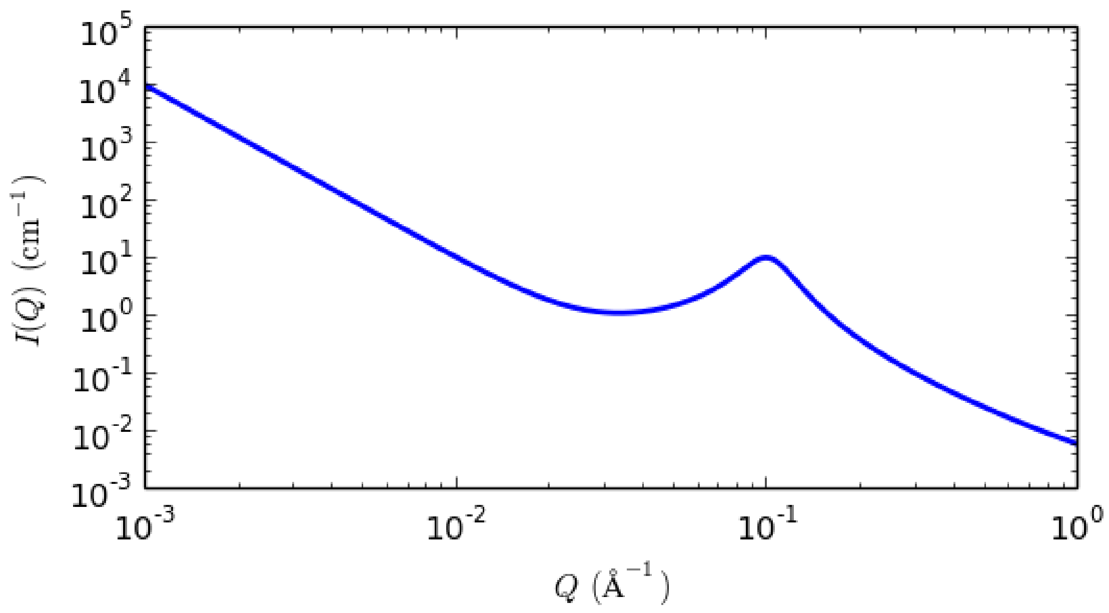


Fig. 1.95: 1D plot corresponding to the default parameters of the model.

References

None.

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Last Modified by:** Paul kienle **Date:** July 24, 2016
- **Last Reviewed by:** Richard Heenan **Date:** March 21, 2016

correlation_length

Calculates an empirical functional form for SAS data characterized by a low-Q signal and a high-Q signal.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
lorentz_scale	Lorentzian Scaling Factor	None	10
porod_scale	Porod Scaling Factor	None	1e-06
cor_length	Correlation length, xi, in Lorentzian	Å	50
porod_exp	Porod Exponent, n, in q^-n	None	3
lorentz_exp	Lorentzian Exponent, m, in 1/(1 + (q.xi)^m)	Å^{-2}	2

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

The scattering intensity I(q) is calculated as

$$I(Q) = \frac{A}{Q^n} + \frac{C}{1 + (Q\xi)^m} + \text{background}$$

The first term describes Porod scattering from clusters (exponent = n) and the second term is a Lorentzian function describing scattering from polymer chains (exponent = m). This second term characterizes the polymer/solvent interactions and therefore the thermodynamics. The two multiplicative factors A and C, and the two exponents n and m are used as fitting parameters. (Respectively *porod_scale*, *lorentz_scale*, *porod_exp* and *lorentz_exp* in the parameter list.) The remaining parameter ξ (*cor_length* in the parameter list) is a correlation length for the polymer chains. Note that when $m = 2$ this functional form becomes the familiar Lorentzian function. Some interpretation of the values of A and C may be possible depending on the values of m and n.

For 2D data: The 2D scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

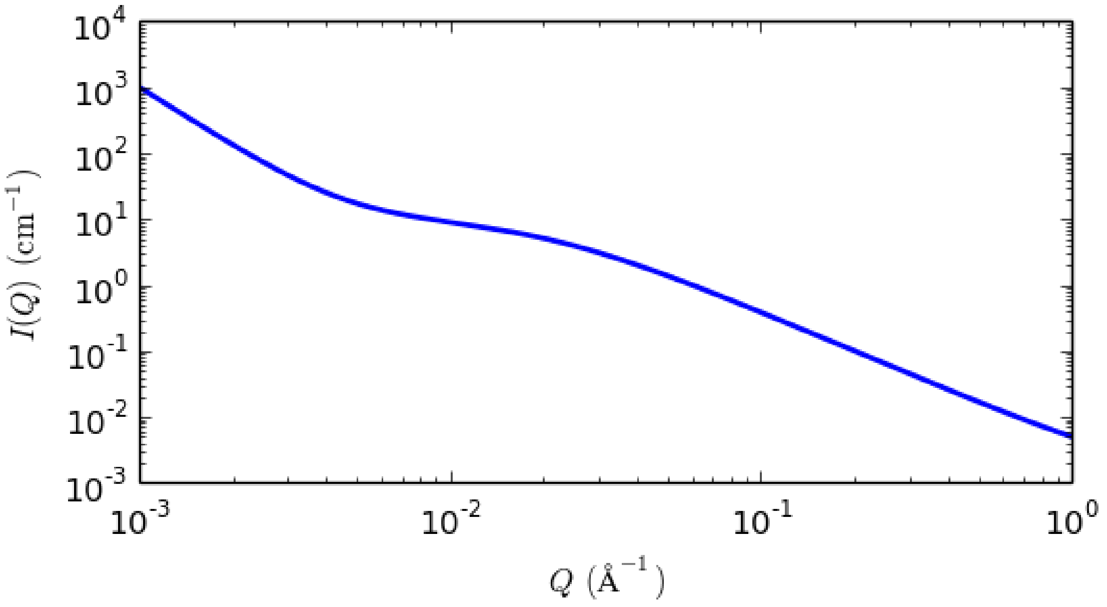


Fig. 1.96: 1D plot corresponding to the default parameters of the model.

References

B Hammouda, D L Ho and S R Kline, Insight into Clustering in Poly(ethylene oxide) Solutions, Macromolecules, 37 (2004) 6932-6937

dab

DAB (Debye Anderson Brumberger) Model

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
cor_length	correlation length	Å	50

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Calculates the scattering from a randomly distributed, two-phase system based on the Debye-Anderson-Brumberger (DAB) model for such systems. The two-phase system is characterized by a single length scale, the correlation length, which is a measure of the average spacing between regions of phase 1 and phase 2. **The model also assumes smooth interfaces between the phases** and hence exhibits Porod behavior $(I \sim q^{-4})$ at large q, $(qL \gg 1)$.

The DAB model is ostensibly a development of the earlier Debye-Bueche model.

Definition

$$I(q) = \text{scale} \cdot \frac{L^3}{(1 + (q \cdot L)^2)^2} + \text{background}$$

where scale is

$$\text{scale} = 8\pi\phi(1 - \phi)\Delta\rho^2$$

and the parameter L is the correlation length.

For 2D data the scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

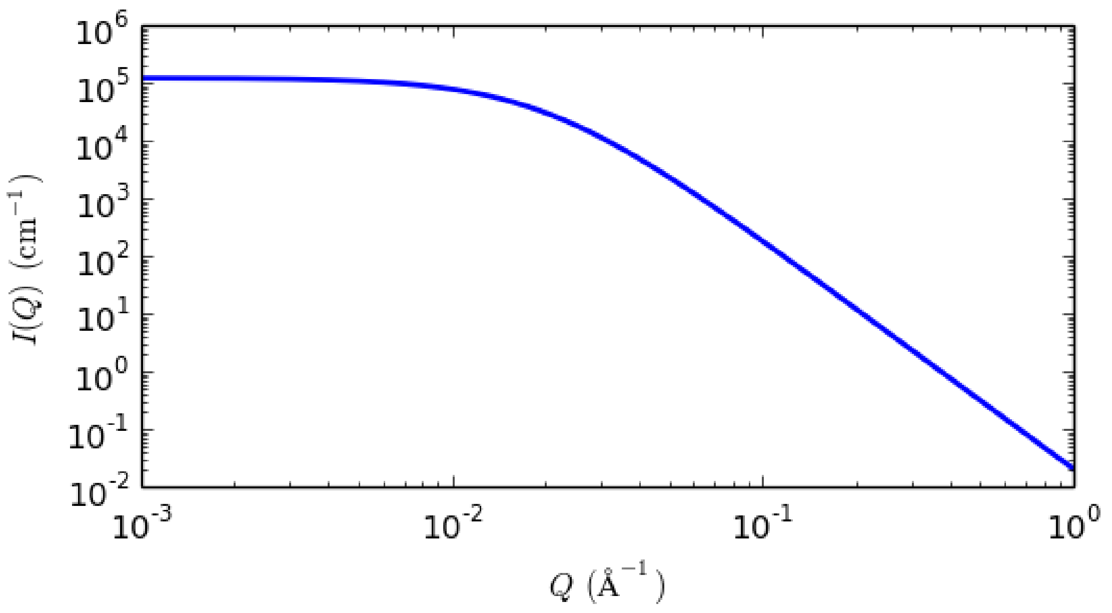


Fig. 1.97: 1D plot corresponding to the default parameters of the model.

References

P Debye, H R Anderson, H Brumberger, *Scattering by an Inhomogeneous Solid. II. The Correlation Function and its Application*, *J. Appl. Phys.*, 28(6) (1957) 679

P Debye, A M Bueche, *Scattering by an Inhomogeneous Solid*, *J. Appl. Phys.*, 20 (1949) 518

2013/09/09 - Description reviewed by King, S and Parker, P.

fractal

Calculates the scattering from fractal-like aggregates of spheres following theTexiera reference.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
volfraction	volume fraction of blocks	None	0.05
radius	radius of particles	Å	5
fractal_dim	fractal dimension	None	2
cor_length	cluster correlation length	Å	100
sld_block	scattering length density of particles	$10^{-6}Å^{-2}$	2
sld_solvent	scattering length density of solvent	$10^{-6}Å^{-2}$	6.4

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition This model calculates the scattering from fractal-like aggregates of spherical building blocks according the following equation:

$$I(q) = \phi\, V_\text{block}(\rho_\text{block} - \rho_\text{solvent})^2 P(q)S(q) + \text{background}$$

where ϕ is The volume fraction of the spherical "building block" particles of radius R_0, V_{block} is the volume of a single building block, $\rho_{solvent}$ is the scattering length density of the solvent, and ρ_{block} is the scattering length density of the building blocks, and P(q), S(q) are the scattering from randomly distributed spherical particles (the building blocks) and the interference from such building blocks organized in a fractal-like clusters. P(q) and S(q) are calculated as:

$$\begin{aligned}
P(q) &= F(qR_0)^2 \\
F(q) &= \frac{3(\sin x - x\cos x)}{x^3} \\
V_\text{particle} &= \frac{4}{3}\,\pi R_0 \\
S(q) &= 1 + \frac{D_f\ \Gamma(D_f - 1)}{[1 + 1/(q\xi)^2\ ]^{(D_f-1)/2}} \frac{\sin[(D_f - 1)\tan^{-1}(q\xi)]}{(qR_0)^{D_f}}
\end{aligned}$$

where ξ is the correlation length representing the cluster size and D_f is the fractal dimension, representing the self similarity of the structure. Note that S(q) here goes negative if D_f is too large, and the Gamma function diverges at $D_f = 0$ and $D_f = 1$.

Polydispersity on the radius is provided for.

For 2D data: The 2D scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

References

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Converted to sasmodels by:** Paul Butler **Date:** March 19, 2016
- **Last Modified by:** Paul Butler **Date:** March 12, 2017
- **Last Reviewed by:** Paul Butler **Date:** March 12, 2017

fractal_core_shell

Scattering from a fractal structure formed from core shell spheres

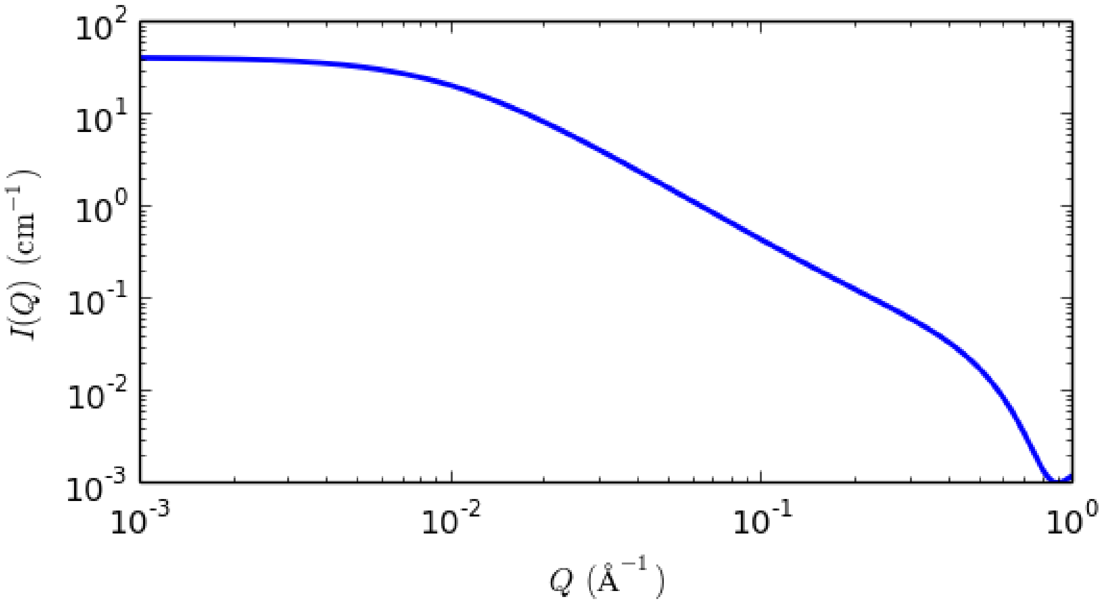


Fig. 1.98: 1D plot corresponding to the default parameters of the model.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
radius	Sphere core radius	Å	60
thickness	Sphere shell thickness	Å	10
sld_core	Sphere core scattering length density	10^{-6}Å^{-2}	1
sld_shell	Sphere shell scattering length density	10^{-6}Å^{-2}	2
sld_solvent	Solvent scattering length density	10^{-6}Å^{-2}	3
volfraction	Volume fraction of building block spheres	None	0.05
fractal_dim	Fractal dimension	None	2
cor_length	Correlation length of fractal-like aggregates	Å	100

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition Calculates the scattering from a fractal structure with a primary building block of core-shell spheres, as opposed to just homogeneous spheres in the fractal model. It is an extension of the well known Teixeira[1] fractal model replacing the $P(q)$ of a solid sphere with that of a core-shell sphere. This model could find use for aggregates of coated particles, or aggregates of vesicles for example.

$$I(q) = P(q)S(q) + \text{background}$$

Where $P(q)$ is the core-shell form factor and $S(q)$ is the Teixeira[1] fractal structure factor both of which are given again below:

$$P(q) = \frac{\phi}{V_s}\left[3V_c(\rho_c - \rho_s)\frac{\sin(qr_c) - qr_c\cos(qr_c)}{(qr_c)^3} + 3V_s(\rho_s - \rho_{solv})\frac{\sin(qr_s) - qr_s\cos(qr_s)}{(qr_s)^3}\right]^2$$

$$S(q) = 1 + \frac{D_f\ \Gamma(D_f - 1)}{[1 + 1/(q\xi)^2]^{(D_f-1)/2}}\frac{\sin[(D_f - 1)\tan^{-1}(q\xi)]}{(qr_s)^{D_f}}$$

where ϕ is the volume fraction of particles, V_s is the volume of the whole particle, V_c is the volume of the core, ρ_c, ρ_s, and ρ_{solv} are the scattering length densities of the core, shell, and solvent respectively, r_c and r_s are the radius of the core and the radius of the whole particle respectively, D_f is the fractal dimension, and ξ the correlation length.

[1] J Teixeira, *J. Appl. Cryst.*, 21 (1988) 781-785

Polydispersity of radius and thickness are also provided for.

This model does not allow for anisotropy and thus the 2D scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

Our model is derived from the form factor calculations implemented in IGOR macros by the NIST Center for Neutron Research[2]

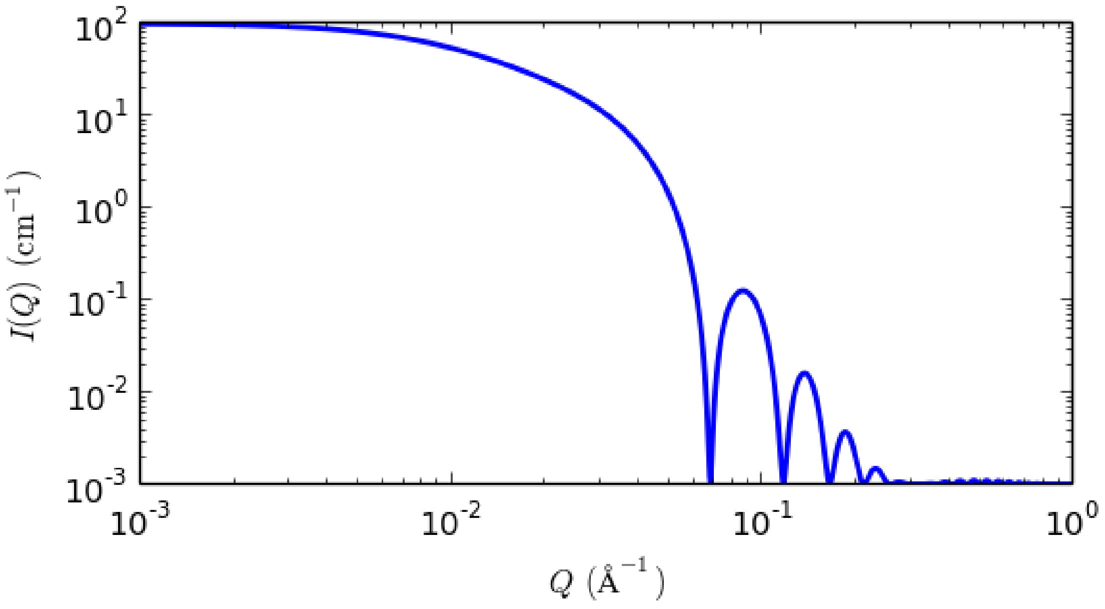


Fig. 1.99: 1D plot corresponding to the default parameters of the model.

References

Authorship and Verification

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Last Modified by:** Paul Butler and Paul Kienzle **Date:** November 27, 2016
- **Last Reviewed by:** Paul Butler and Paul Kienzle **Date:** November 27, 2016

gauss_lorentz_gel

Gauss Lorentz Gel model of scattering from a gel structure

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
gauss_scale	Gauss scale factor	None	100
cor_length_static	Static correlation length	Å	100
lorentz_scale	Lorentzian scale factor	None	50
cor_length_dynamic	Dynamic correlation length	Å	20

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

[2] S R Kline, *J Appl. Cryst.*, 39 (2006) 895

This model calculates the scattering from a gel structure, but typically a physical rather than chemical network. It is modeled as a sum of a low-q exponential decay (which happens to give a functional form similar to Guinier scattering, so interpret with care) plus a Lorentzian at higher-q values. See also the gel_fit model.

Definition

The scattering intensity $I(q)$ is calculated as (Eqn. 5 from the reference)

$$I(q) = I_G(0)\exp(-q^2\Xi^2/2) + I_L(0)/(1 + q^2\xi^2)$$

Ξ is the length scale of the static correlations in the gel, which can be attributed to the "frozen-in" crosslinks. ξ is the dynamic correlation length, which can be attributed to the fluctuating polymer chains between crosslinks. $I_G(0)$ and $I_L(0)$ are the scaling factors for each of these structures. Think carefully about how these map to your particular system!

Note: The peaked structure at higher q values (Figure 2 from the reference) is not reproduced by the model. Peaks can be introduced into the model by summing this model with the *gaussian_peak* model.

For 2D data the scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

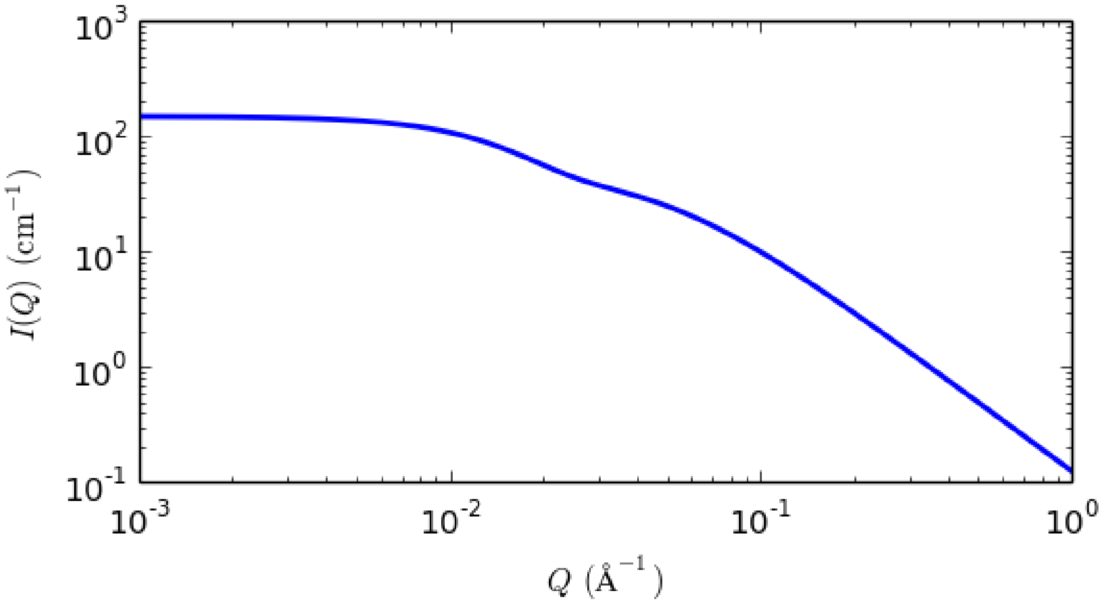


Fig. 1.100: 1D plot corresponding to the default parameters of the model.

References

G Evmenenko, E Theunissen, K Mortensen, H Reynaers, *Polymer*, 42 (2001) 2907-2913

gaussian_peak

Gaussian shaped peak

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
peak_pos	Peak position	Å^{-1}	0.05
sigma	Peak width (standard deviation)	Å^{-1}	0.005

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

This model describes a Gaussian shaped peak on a flat background

$$I(q) = (\text{scale}) \exp \left[-\tfrac{1}{2} (q - q_0)^2 / \sigma^2 \right] + \text{background}$$

with the peak having height of *scale* centered at q_0 and having a standard deviation of σ. The FWHM (full-width half-maximum) is 2.354σ.

For 2D data, scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

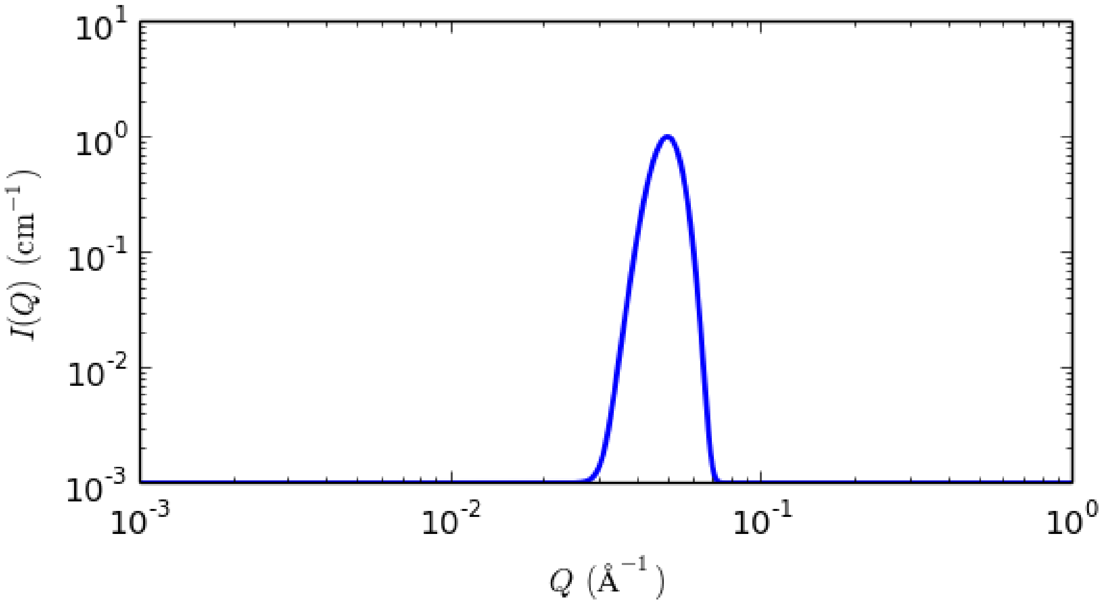


Fig. 1.101: 1D plot corresponding to the default parameters of the model.

References

None.

gel_fit

Fitting using fine-scale polymer distribution in a gel.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
guinier_scale	Guinier length scale	cm^-1	1.7
lorentz_scale	Lorentzian length scale	cm^-1	3.5
rg	Radius of gyration	Å	104
fractal_dim	Fractal exponent	None	2
cor_length	Correlation length	Å	16

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

This model was implemented by an interested user!

Unlike a concentrated polymer solution, the fine-scale polymer distribution in a gel involves at least two characteristic length scales, a shorter correlation length ($a1$) to describe the rapid fluctuations in the position of the polymer chains that ensure thermodynamic equilibrium, and a longer distance (denoted here as $a2$) needed to account for the static accumulations of polymer pinned down by junction points or clusters of such points. The latter is derived from a simple Guinier function. Compare also the gauss_lorentz_gel model.

Definition

The scattered intensity $I(q)$ is calculated as

$$I(Q) = I(0)_L \frac{1}{(1 + [((D + 1/3)Q^2 a_1^2])^{D/2}} + I(0)_G exp\left(-Q^2 a_2^2\right) + B$$

where

$$a_2^2 \approx \frac{R_g^2}{3}$$

Note that the first term reduces to the Ornstein-Zernicke equation when $D = 2$; ie, when the Flory exponent is 0.5 (theta conditions). In gels with significant hydrogen bonding D has been reported to be ~2.6 to 2.8.

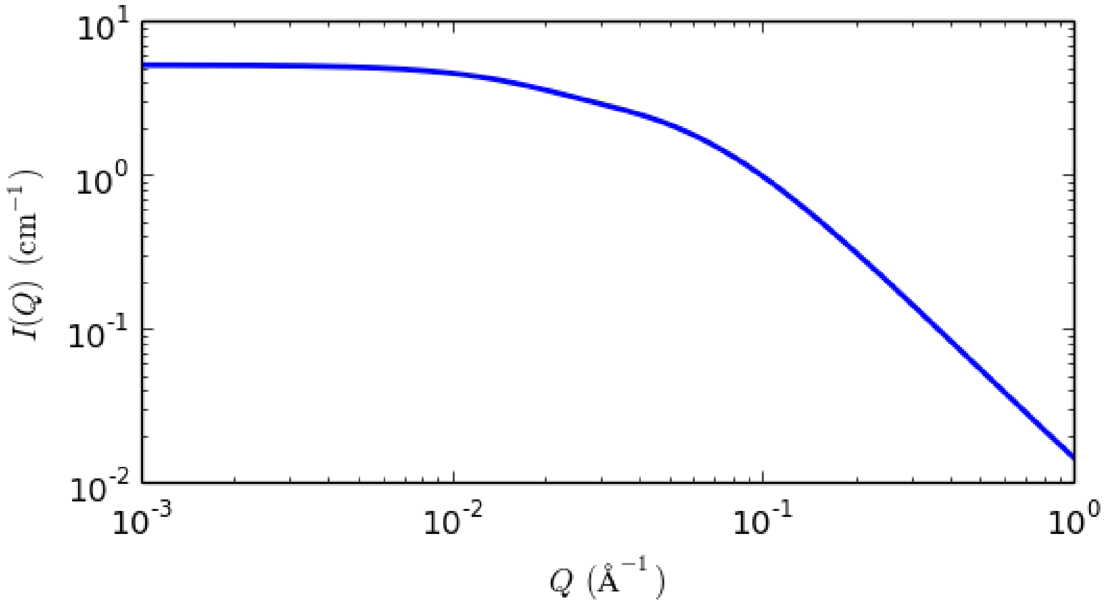


Fig. 1.102: 1D plot corresponding to the default parameters of the model.

References

Mitsuhiro Shibayama, Toyoichi Tanaka, Charles C Han, *J. Chem. Phys.* 1992, 97 (9), 6829-6841

Simon Mallam, Ferenc Horkay, Anne-Marie Hecht, Adrian R Rennie, Erik Geissler, *Macromolecules* 1991, 24, 543-548

guinier

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
rg	Radius of Gyration	Å	60

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

This model fits the Guinier function

$$I(q) = \text{scale} \cdot \exp\left[\frac{-Q^2 R_g^2}{3}\right] + \text{background}$$

to the data directly without any need for linearisation (*cf.* the usual plot of $\ln I(q)$ vs q^2). Note that you may have to restrict the data range to include small q only, where the Guinier approximation actually applies. See also the guinier_porod model.

For 2D data the scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

In scattering, the radius of gyration R_g quantifies the objects's distribution of SLD (not mass density, as in mechanics) from the objects's SLD centre of mass. It is defined by

$$R_g^2 = \frac{\sum_i \rho_i \left(r_i - r_0\right)^2}{\sum_i \rho_i}$$

where r_0 denotes the object's SLD centre of mass and ρ_i is the SLD at a point i.

Notice that R_g^2 may be negative (since SLD can be negative), which happens when a form factor $P(Q)$ is increasing with Q rather than decreasing. This can occur for core/shell particles, hollow particles, or for composite particles with domains of different SLDs in a solvent with an SLD close to the average match point. (Alternatively, this might be regarded as there being an internal inter-domain "structure factor" within a single particle which gives rise to a peak in the scattering).

To specify a negative value of R_g^2 in SasView, simply give R_g a negative value (R_g^2 will be evaluated as $R_g|R_g|$). Note that the physical radius of gyration, of the exterior of the particle, will still be large and positive. It is only the apparent size from the small Q data that will give a small or negative value of R_g^2.

References

A Guinier and G Fournet, *Small-Angle Scattering of X-Rays*, John Wiley & Sons, New York (1955)

guinier_porod

Guinier-Porod function

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
rg	Radius of gyration	Å	60
s	Dimension variable	None	1
porod_exp	Porod exponent	None	3

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Calculates the scattering for a generalized Guinier/power law object. This is an empirical model that can be used to determine the size and dimensionality of scattering objects, including asymmetric objects such as rods or

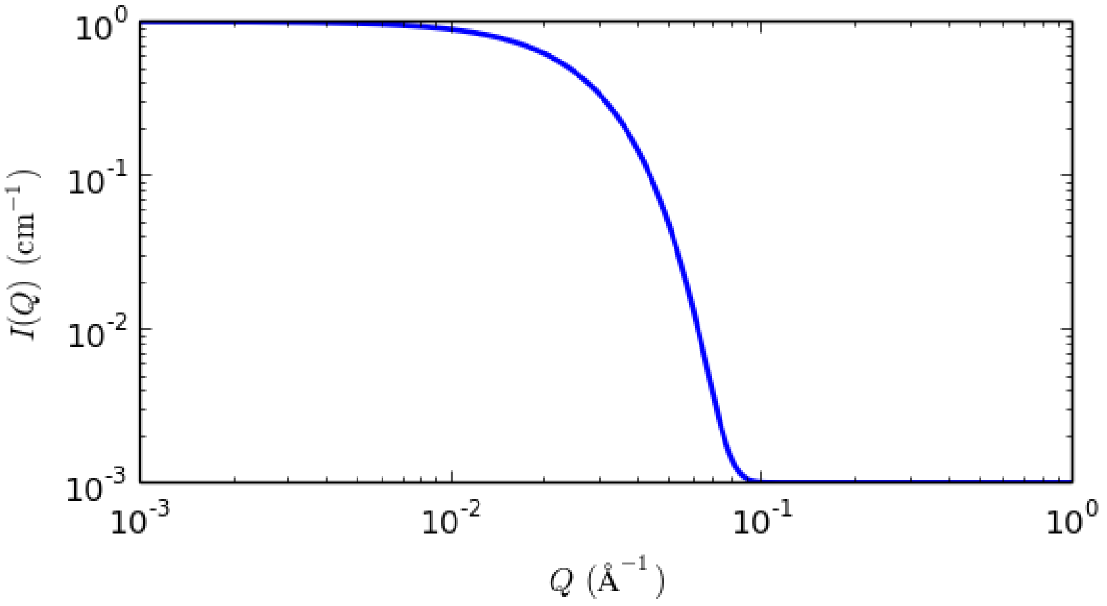


Fig. 1.103: 1D plot corresponding to the default parameters of the model.

platelets, and shapes intermediate between spheres and rods or between rods and platelets, and overcomes some of the deficiencies of the (Beaucage) Unified_Power_Rg model (see Hammouda, 2010).

Definition

The following functional form is used

$$I(q) = \begin{cases} \frac{G}{Q^s}\ \exp\left[\frac{-Q^2R_g^2}{3-s}\right] & Q \leq Q_1 \\ D/Q^m & Q \geq Q_1 \end{cases}$$

This is based on the generalized Guinier law for such elongated objects (see the Glatter reference below). For 3D globular objects (such as spheres), $s = 0$ and one recovers the standard Guinier formula. For 2D symmetry (such as for rods) $s = 1$, and for 1D symmetry (such as for lamellae or platelets) $s = 2$. A dimensionality parameter $(3 - s)$ is thus defined, and is 3 for spherical objects, 2 for rods, and 1 for plates.

Enforcing the continuity of the Guinier and Porod functions and their derivatives yields

$$Q_1 = \frac{1}{R_g} \sqrt{(m-s)(3-s)/2}$$

and

$$\begin{aligned} D &= G\ \exp\left[\frac{-Q_1^2 R_g^2}{3-s}\right] Q_1^{m-s} \\ &= \frac{G}{R_g^{m-s}}\ \exp\left[-\frac{m-s}{2}\right] \left(\frac{(m-s)(3-s)}{2}\right)^{\frac{m-s}{2}} \end{aligned}$$

Note that the radius of gyration for a sphere of radius R is given by $R_g = R\sqrt{3/5}$. For a cylinder of radius R and length L, $R_g^2 = \frac{L^2}{12} + \frac{R^2}{2}$ from which the cross-sectional radius of gyration for a randomly oriented thin cylinder is $R_g = R/\sqrt{2}$ and the cross-sectional radius of gyration of a randomly oriented lamella of thickness T is given by $R_g = T/\sqrt{12}$.

For 2D data: The 2D scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

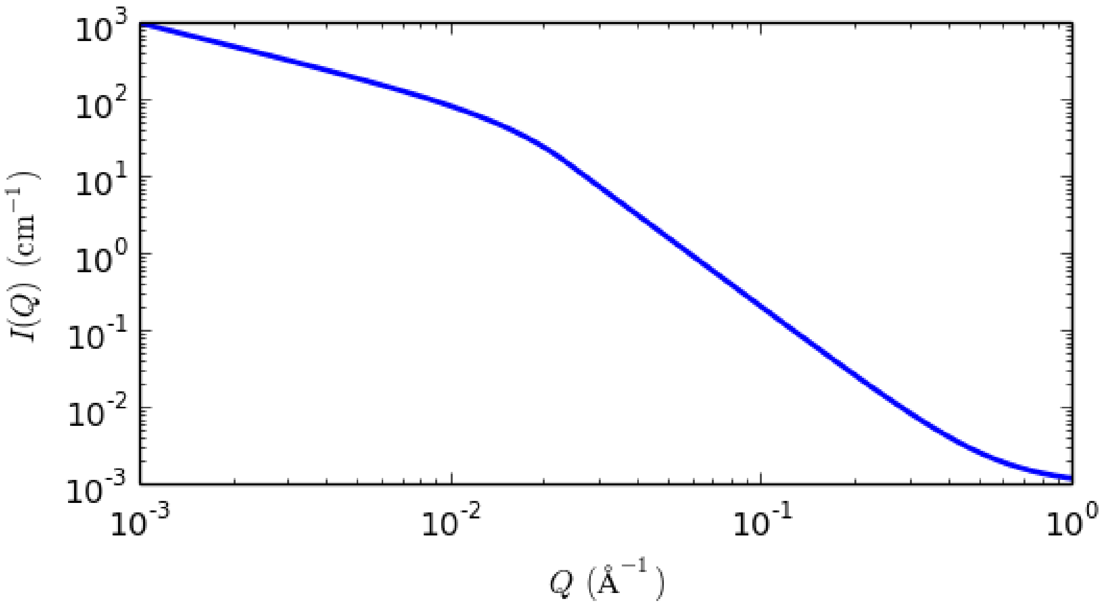


Fig. 1.104: 1D plot corresponding to the default parameters of the model.

Reference

B Hammouda, *A new Guinier-Porod model, J. Appl. Cryst.*, (2010), 43, 716-719

B Hammouda, *Analysis of the Beaucage model, J. Appl. Cryst.*, (2010), 43, 1474-1478

line

Line model

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
intercept	intercept in linear model	cm^{-1}	1
slope	slope in linear model	Å·cm^{-1}	1

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

This model calculates intensity using simple linear function

Definition

The scattering intensity $I(q)$ is calculated as

$$I(q) = \text{scale}(A + B \cdot q) + \text{background}$$

Note: For 2D plots intensity has different definition than other shape independent models

$$I(q) = \text{scale}(I(qx) \cdot I(qy)) + \text{background}$$

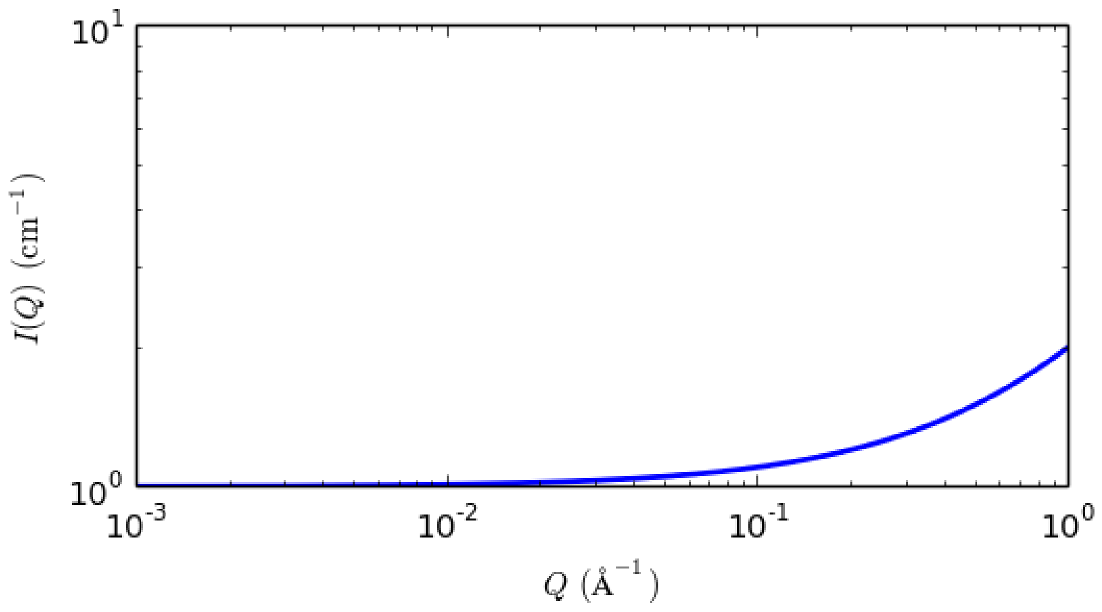


Fig. 1.105: 1D plot corresponding to the default parameters of the model.

References

None.

lorentz

Ornstein-Zernicke correlation length model

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
cor_length	Screening length	Å	50

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Lorentz (Ornstein-Zernicke Model)

Definition

The Ornstein-Zernicke model is defined by

$$I(q) = \frac{\text{scale}}{1 + (qL)^2} + \text{background}$$

The parameter L is the screening length *cor_length*.

For 2D data the scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

References

L.S. Qrnstein and F. Zernike, *Proc. Acad. Sci. Amsterdam* 17, 793 (1914), and *Z. Phys.* 19, 134 (1918), and 27, 761 {1926); referred to as QZ.

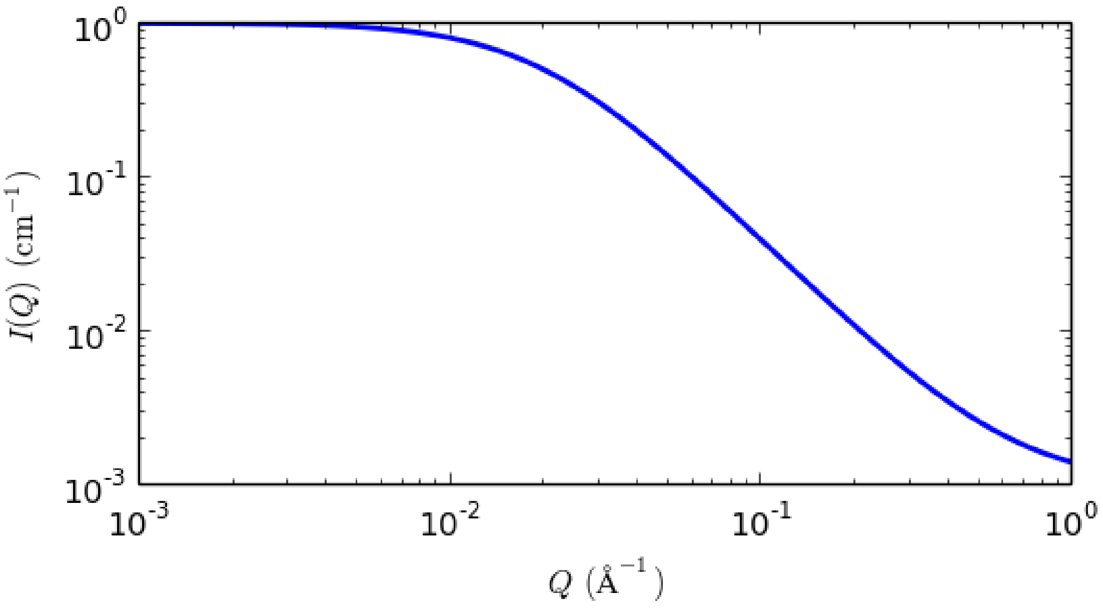


Fig. 1.106: 1D plot corresponding to the default parameters of the model.

mass_fractal

Mass Fractal model

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
radius	Particle radius	Å	10
fractal_dim_mass	Mass fractal dimension	None	1.9
cutoff_length	Cut-off length	Å	100

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Calculates the scattering from fractal-like aggregates based on the Mildner reference.

Definition

The scattering intensity $I(q)$ is calculated as

$$I(q) = scale \times P(q)S(q) + background$$

$$P(q) = F(qR)^2$$

$$F(x) = \frac{3\left[sin(x) - xcos(x)\right]}{x^3}$$

$$S(q) = \frac{\Gamma(D_m - 1)\zeta^{D_m - 1}}{\left[1 + (q\zeta)^2\right]^{(D_m - 1)/2}} \frac{sin\left[(D_m - 1)tan^{-1}(q\zeta)\right]}{q}$$

$$scale = scale\_factor \times NV^2(\rho_{\text{particle}} - \rho_{\text{solvent}})^2$$

$$V = \frac{4}{3}\pi R^3$$

where R is the radius of the building block, D_m is the **mass** fractal dimension, ζ is the cut-off length, ρ_{solvent} is the scattering length density of the solvent, and ρ_{particle} is the scattering length density of particles.

Note: The mass fractal dimension (D_m) is only valid if $1 < mass\_dim < 6$. It is also only valid over a limited q range (see the reference for details).

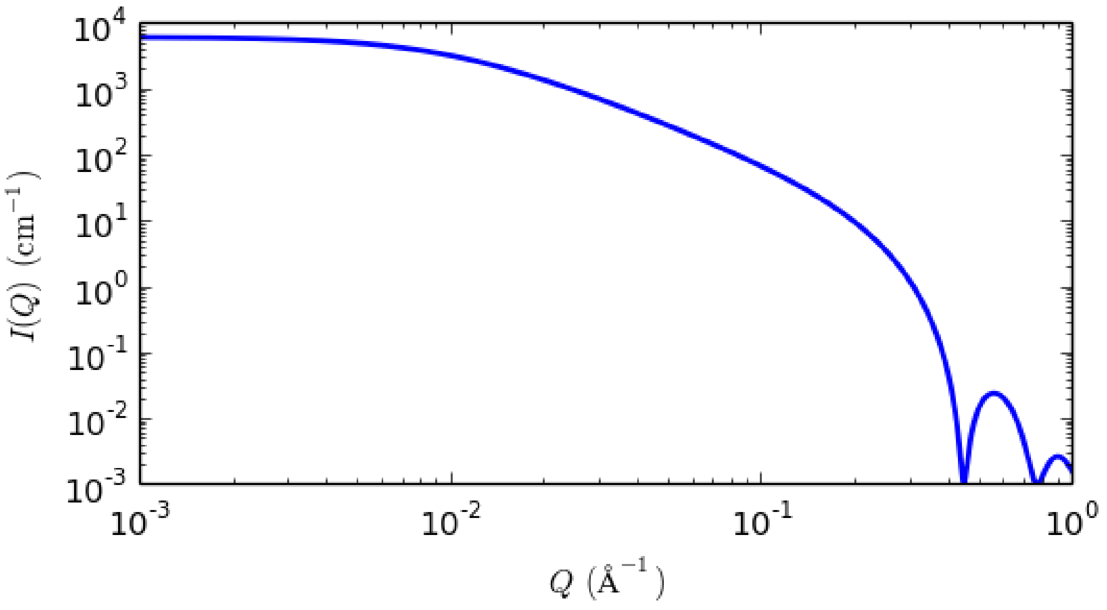


Fig. 1.107: 1D plot corresponding to the default parameters of the model.

References

D Mildner and P Hall, *J. Phys. D: Appl. Phys.*, 19 (1986) 1535-1545 Equation(9)

mass_surface_fractal

Mass Surface Fractal model

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
fractal_dim_mass	Mass fractal dimension	None	1.8
fractal_dim_surf	Surface fractal dimension	None	2.3
rg_cluster	Cluster radius of gyration	Å	4000
rg_primary	Primary particle radius of gyration	Å	86.7

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

A number of natural and commercial processes form high-surface area materials as a result of the vapour-phase aggregation of primary particles. Examples of such materials include soots, aerosols, and fume or pyrogenic silicas. These are all characterised by cluster mass distributions (sometimes also cluster size distributions) and internal surfaces that are fractal in nature. The scattering from such materials displays two distinct breaks in log-log representation, corresponding to the radius-of-gyration of the primary particles, rg, and the radius-of-gyration of the clusters (aggregates), Rg. Between these boundaries the scattering follows a power law related to the mass fractal dimension, Dm, whilst above the high-Q boundary the scattering follows a power law related to the surface fractal dimension of the primary particles, Ds.

Definition

The scattered intensity I(q) is calculated using a modified Ornstein-Zernicke equation

$$I(q) = scale \times P(q) + background$$
$$P(q) = \left\{ \left[1 + (q^2 a)\right]^{D_m/2} \times \left[1 + (q^2 b)\right]^{(6-D_s-D_m)/2} \right\}^{-1}$$
$$a = R_g^2/(3D_m/2)$$
$$b = r_g^2/[-3(D_s + D_m - 6)/2]$$
$$scale = scale\_factor \times NV^2(\rho_{particle} - \rho_{solvent})^2$$

where R_g is the size of the cluster, r_g is the size of the primary particle, D_s is the surface fractal dimension, D_m is the mass fractal dimension, $\rho_{solvent}$ is the scattering length density of the solvent, and $\rho_{particle}$ is the scattering length density of particles.

Note: The surface (D_s) and mass (D_m) fractal dimensions are only valid if $0 < surface\_dim < 6$, $0 < mass\_dim < 6$, and $(surface\_dim + mass\_dim) < 6$. Older versions of sasview may have the default primary particle radius larger than the cluster radius, this was an error, also present in the Schmidt review paper below. The primary particle should be the smaller as described in the original Hurd et.al. who also point out that polydispersity in the primary particle sizes may affect their apparent surface fractal dimension.

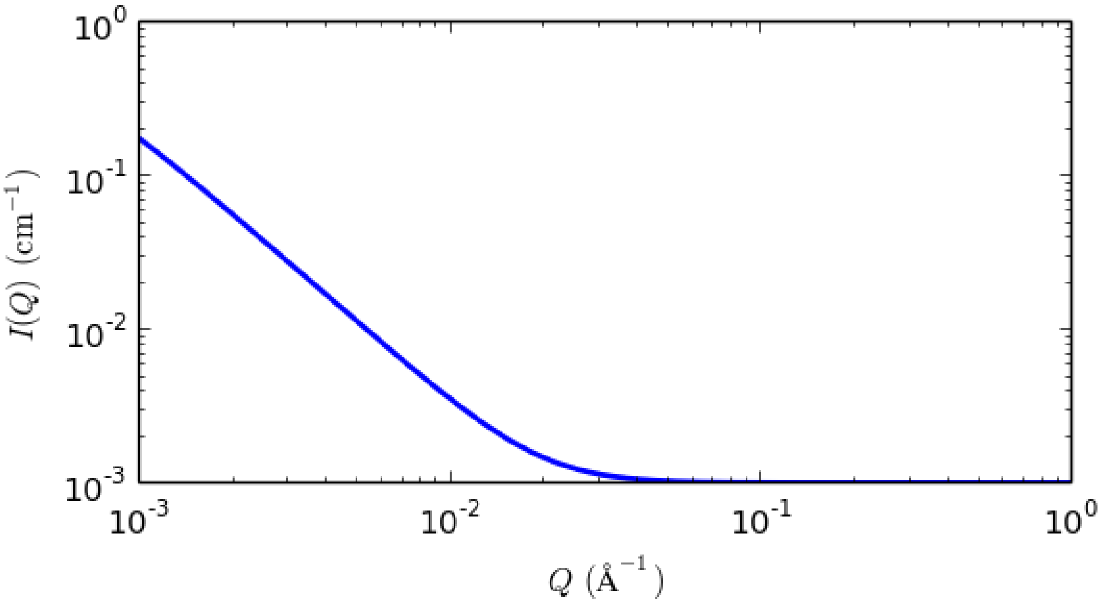


Fig. 1.108: 1D plot corresponding to the default parameters of the model.

References

Authorship and Verification

- **Converted to sasmodels by:** Piotr Rozyczko **Date:** Jan 20, 2016
- **Last Reviewed by:** Richard Heenan **Date:** May 30, 2018

mono_gauss_coil

Scattering from monodisperse polymer coils

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
i_zero	Intensity at q=0	cm^{-1}	70
rg	Radius of gyration	Å	75

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

This Debye Gaussian coil model strictly describes the scattering from *monodisperse* polymer chains in theta solvents or polymer melts, conditions under which the distances between segments follow a Gaussian distribution. Provided the number of segments is large (ie, high molecular weight polymers) the single-chain form factor P(Q) is that described by Debye (1947).

To describe the scattering from *polydisperse* polymer chains see the *poly_gauss_coil* model.

Definition

$$I(q) = \text{scale} \cdot I_0 \cdot P(q) + \text{background}$$

where

$$\begin{aligned} I_0 &= \phi_\text{poly} \cdot V \cdot (\rho_\text{poly} - \rho_\text{solv})^2 \\ P(q) &= 2[\exp(-Z) + Z - 1]/Z^2 \\ Z &= (qR_g)^2 \\ V &= M/(N_A \delta) \end{aligned}$$

Here, ϕ_poly is the volume fraction of polymer, V is the volume of a polymer coil, *M* is the molecular weight of the polymer, N_A is Avogadro's Number, δ is the bulk density of the polymer, ρ_poly is the sld of the polymer, ρsolv is the sld of the solvent, and R_g is the radius of gyration of the polymer coil.

The 2D scattering intensity is calculated in the same way as the 1D, but where the q vector is redefined as

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

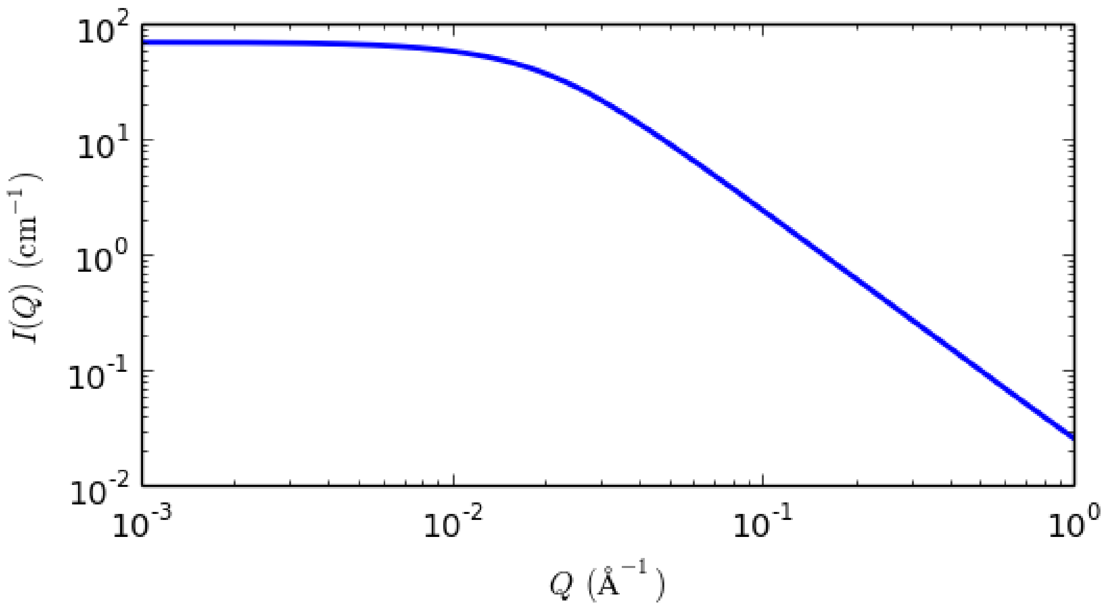


Fig. 1.109: 1D plot corresponding to the default parameters of the model.

References

P Debye, *J. Phys. Colloid. Chem.*, 51 (1947) 18.

R J Roe, *Methods of X-Ray and Neutron Scattering in Polymer Science*, Oxford University Press, New York (2000).

http://www.ncnr.nist.gov/staff/hammouda/distance_learning/chapter_28.pdf

peak_lorentz

A Lorentzian peak on a flat background

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
peak_pos	Peak postion in q	Å^{-1}	0.05
peak_hwhm	HWHM of peak	Å^{-1}	0.005

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

This model describes a Lorentzian shaped peak on a flat background.

Definition

The scattering intensity $I(q)$ is calculated as

$$I(q) = \frac{scale}{\left(1 + \left(\frac{q-q_0}{B}\right)^2\right)} + background$$

with the peak having height of I_0 centered at q_0 and having a HWHM (half-width half-maximum) of B.

For 2D data the scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

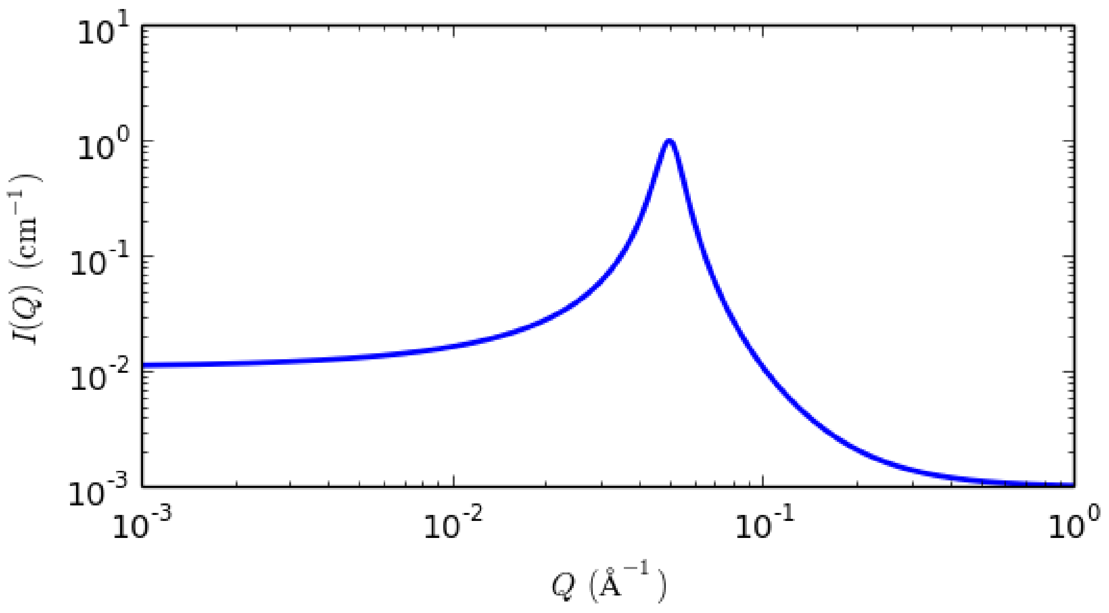


Fig. 1.110: 1D plot corresponding to the default parameters of the model.

References

None.

poly_gauss_coil

Scattering from polydisperse polymer coils

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
i_zero	Intensity at q=0	cm^{-1}	70
rg	Radius of gyration	Å	75
polydispersity	Polymer Mw/Mn	None	2

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

This empirical model describes the scattering from *polydisperse* polymer chains in theta solvents or polymer melts, assuming a Schulz-Zimm type molecular weight distribution.

To describe the scattering from *monodisperse* polymer chains, see the *mono_gauss_coil* model.

Definition

$$I(q) = \text{scale} \cdot I_0 \cdot P(q) + \text{background}$$

where

$$\begin{aligned} I_0 &= \phi_\text{poly} \cdot V \cdot (\rho_\text{poly} - \rho_\text{solv})^2 \\ P(q) &= 2[(1 + UZ)^{-1/U} + Z - 1]/[(1 + U)Z^2] \\ Z &= [(qR_g)^2]/(1 + 2U) \\ U &= (Mw/Mn) - 1 = \text{polydispersity ratio} - 1 \\ V &= M/(N_A\delta) \end{aligned}$$

Here, ϕ_poly, is the volume fraction of polymer, V is the volume of a polymer coil, M is the molecular weight of the polymer, N_A is Avogadro's Number, δ is the bulk density of the polymer, ρ_poly is the sld of the polymer, ρ_solv is the sld of the solvent, and R_g is the radius of gyration of the polymer coil.

The 2D scattering intensity is calculated in the same way as the 1D, but where the q vector is redefined as

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

References

O Glatter and O Kratky (editors), *Small Angle X-ray Scattering*, Academic Press, (1982) Page 404.

J S Higgins, H C Benoit, *Polymers and Neutron Scattering*, Oxford Science Publications, (1996).

S M King, *Small Angle Neutron Scattering* in *Modern Techniques for Polymer Characterisation*, Wiley, (1999).

http://www.ncnr.nist.gov/staff/hammouda/distance_learning/chapter_28.pdf

polymer_excl_volume

Polymer Excluded Volume model

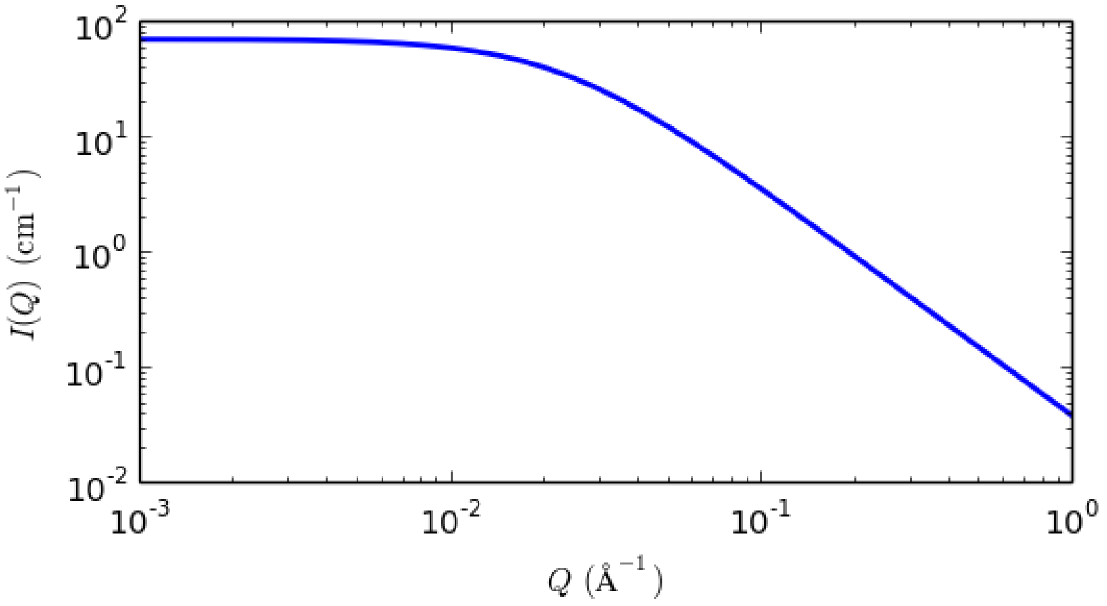


Fig. 1.111: 1D plot corresponding to the default parameters of the model.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
rg	Radius of Gyration	Å	60
porod_exp	Porod exponent	None	3

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

This model describes the scattering from polymer chains subject to excluded volume effects and has been used as a template for describing mass fractals.

Definition

The form factor was originally presented in the following integral form (Benoit, 1957)

$$P(Q) = 2\int_0^1 dx(1-x)exp\left[-\frac{Q^2a^2}{6}n^{2v}x^{2v}\right]$$

where ν is the excluded volume parameter (which is related to the Porod exponent m as $\nu = 1/m$), a is the statistical segment length of the polymer chain, and n is the degree of polymerization.

This integral was put into an almost analytical form as follows (Hammouda, 1993)

$$P(Q) = \frac{1}{\nu U^{1/2\nu}}\left\{\gamma\left(\frac{1}{2\nu}, U\right) - \frac{1}{U^{1/2\nu}}\gamma\left(\frac{1}{\nu}, U\right)\right\}$$

and later recast as (for example, Hore, 2013; Hammouda & Kim, 2017)

$$P(Q) = \frac{1}{\nu U^{1/2\nu}}\gamma\left(\frac{1}{2\nu}, U\right) - \frac{1}{\nu U^{1/\nu}}\gamma\left(\frac{1}{\nu}, U\right)$$

where $\gamma(x, U)$ is the incomplete gamma function

$$\gamma(x, U) = \int_0^U dt\ \exp(-t)t^{x-1}$$

and the variable U is given in terms of the scattering vector Q as

$$U = \frac{Q^2a^2n^{2\nu}}{6} = \frac{Q^2R_g^2(2\nu+1)(2\nu+2)}{6}$$

The two analytic forms are equivalent. In the 1993 paper

$$\frac{1}{\nu U^{1/2\nu}}$$

has been factored out.

SasView implements the 1993 expression.

The square of the radius-of-gyration is defined as

$$R_g^2 = \frac{a^2 n^{2\nu}}{(2\nu+1)(2\nu+2)}$$

Note: This model applies only in the mass fractal range (ie, $5/3 <= m <= 3$) and **does not apply** to surface fractals ($3 < m <= 4$). It also does not reproduce the rigid rod limit (m=1) because it assumes chain flexibility from the outset. It may cover a portion of the semi-flexible chain range ($1 < m < 5/3$).

A low-Q expansion yields the Guinier form and a high-Q expansion yields the Porod form which is given by

$$P(Q\rightarrow\infty) = \frac{1}{\nu U^{1/2\nu}}\Gamma\left(\frac{1}{2\nu}\right) - \frac{1}{\nu U^{1/\nu}}\Gamma\left(\frac{1}{\nu}\right)$$

Here $\Gamma(x) = \gamma(x,\infty)$ is the gamma function.

The asymptotic limit is dominated by the first term

$$P(Q\rightarrow\infty) \sim \frac{1}{\nu U^{1/2\nu}}\Gamma\left(\frac{1}{2\nu}\right) = \frac{m}{(QR_g)^m}\left[\frac{6}{(2\nu+1)(2\nu+2)}\right]^{m/2}\Gamma(m/2)$$

The special case when $\nu = 0.5$ (or $m = 1/\nu = 2$) corresponds to Gaussian chains for which the form factor is given by the familiar Debye function.

$$P(Q) = \frac{2}{Q^4 R_g^4}\left[\exp(-Q^2R_g^2) - 1 + Q^2R_g^2\right]$$

For 2D data: The 2D scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

References

H Benoit, *Comptes Rendus*, 245 (1957) 2244-2247

B Hammouda, *SANS from Homogeneous Polymer Mixtures - A Unified Overview, Advances in Polym. Sci.* 106 (1993) 87-133

M Hore et al, *Co-Nonsolvency of Poly(n-isopropylacrylamide) in Deuterated Water/Ethanol Mixtures* 46 (2013) 7894-7901

B Hammouda & M-H Kim, *The empirical core-chain model* 247 (2017) 434-440

porod

Porod function

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001

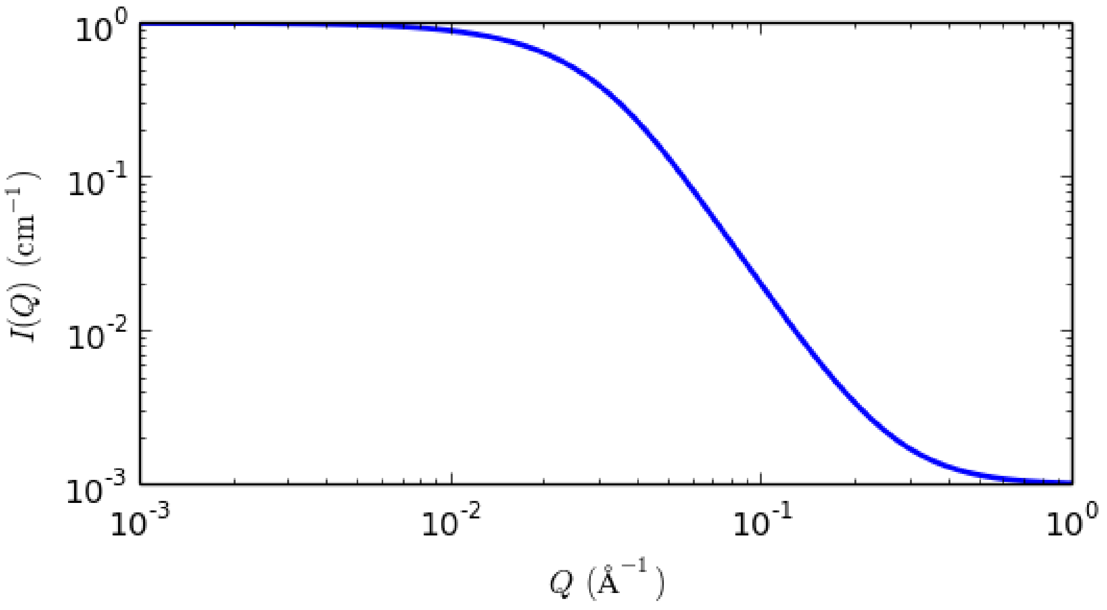


Fig. 1.112: 1D plot corresponding to the default parameters of the model.

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

This model fits the Porod function

$$I(q) = C/q^4$$

to the data directly without any need for linearisation (cf. Log I(q) vs Log q).

Here $C = 2\pi(\Delta\rho)^2 S_v$ is the scale factor where S_v is the specific surface area (ie, surface area / volume) of the sample, and $\Delta\rho$ is the contrast factor.

For 2D data: The 2D scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

References

G Porod. *Kolloid Zeit.* 124 (1951) 83.

L A Feigin, D I Svergun, G W Taylor. *Structure Analysis by Small-Angle X-ray and Neutron Scattering.* Springer. (1987)

power_law

Simple power law with a flat background

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
power	Power law exponent	None	4

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

This model calculates a simple power law with a flat background.

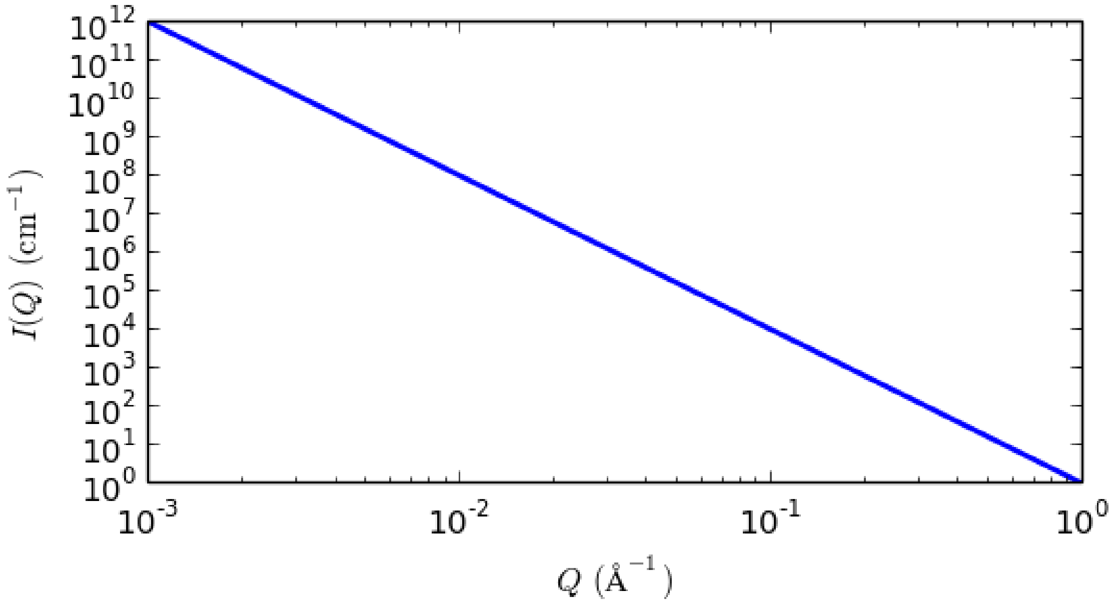


Fig. 1.113: 1D plot corresponding to the default parameters of the model.

Definition

$$I(q) = \text{scale} \cdot q^{-\text{power}} + \text{background}$$

Note the minus sign in front of the exponent. The exponent *power* should therefore be entered as a **positive** number for fitting.

Also note that unlike many other models, *scale* in this model is NOT explicitly related to a volume fraction. Be careful if combining this model with other models.

References

None.

rpa

Random Phase Approximation

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
case_num	Component organization	None	1
N[4]	Degree of polymerization	None	1000
Phi[4]	volume fraction	None	0.25
v[4]	molar volume	mL/mol	100
L[4]	scattering length	fm	10
b[4]	segment length	Å	5
K12	A:B interaction parameter	None	-0.0004
K13	A:C interaction parameter	None	-0.0004
K14	A:D interaction parameter	None	-0.0004
K23	B:C interaction parameter	None	-0.0004
K24	B:D interaction parameter	None	-0.0004
K34	C:D interaction parameter	None	-0.0004

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

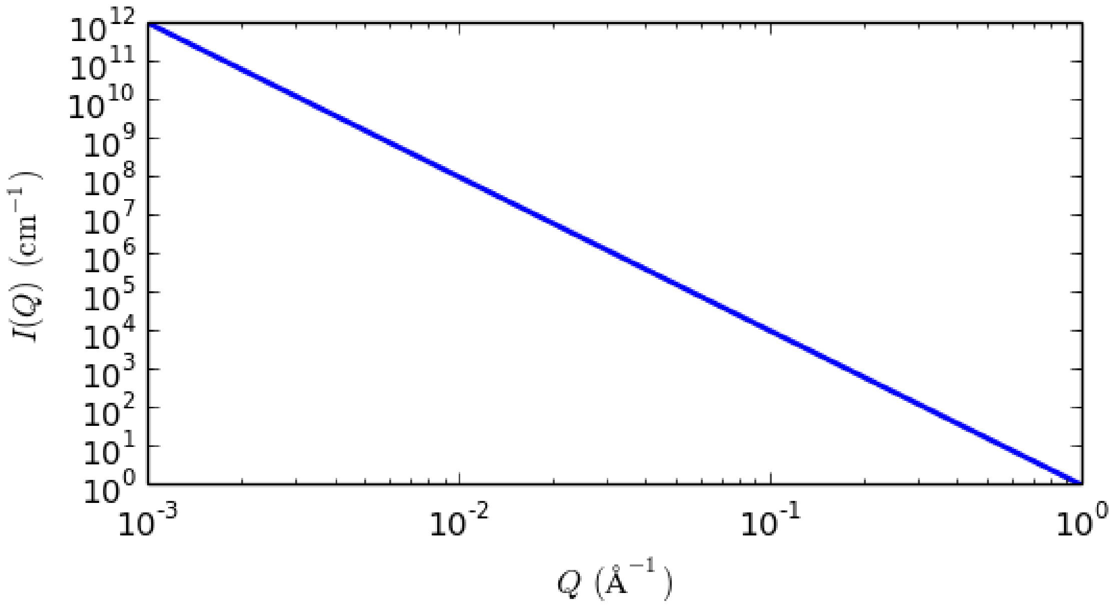


Fig. 1.114: 1D plot corresponding to the default parameters of the model.

Definition

Calculates the macroscopic scattering intensity for a multi-component homogeneous mixture of polymers using the Random Phase Approximation. This general formalism contains 10 specific cases

Case 0: C/D binary mixture of homopolymers

Case 1: C-D diblock copolymer

Case 2: B/C/D ternary mixture of homopolymers

Case 3: C/C-D mixture of a homopolymer B and a diblock copolymer C-D

Case 4: B-C-D triblock copolymer

Case 5: A/B/C/D quaternary mixture of homopolymers

Case 6: A/B/C-D mixture of two homopolymers A/B and a diblock C-D

Case 7: A/B-C-D mixture of a homopolymer A and a triblock B-C-D

Case 8: A-B/C-D mixture of two diblock copolymers A-B and C-D

Case 9: A-B-C-D tetra-block copolymer

Note: These case numbers are different from those in the NIST SANS package!

The models are based on the papers by Akcasu *et al.* and by Hammouda assuming the polymer follows Gaussian statistics such that $R_g^2 = nb^2/6$ where b is the statistical segment length and n is the number of statistical segment lengths. A nice tutorial on how these are constructed and implemented can be found in chapters 28 and 39 of Boualem Hammouda's 'SANS Toolbox'.

In brief the macroscopic cross sections are derived from the general forms for homopolymer scattering and the multiblock cross-terms while the inter polymer cross terms are described in the usual way by the χ parameter.

USAGE NOTES:

- Only one case can be used at any one time.
- The RPA (mean field) formalism only applies only when the multicomponent polymer mixture is in the homogeneous mixed-phase region.

- **Component D is assumed to be the "background" component (ie, all contrasts are calculated with respect to component D).** So the scattering contrast for a C/D blend = [SLD(component C) - SLD(component D)]^2.
- Depending on which case is being used, the number of fitting parameters can vary.

Note:

- In general the degrees of polymerization, the volume fractions, the molar volumes, and the neutron scattering lengths for each component are obtained from other methods and held fixed while The *scale* parameter should be held equal to unity.
- The variables are normally the segment lengths (b_a, b_b, etc.) and χ parameters (K_{ab}, K_{ac}, etc).

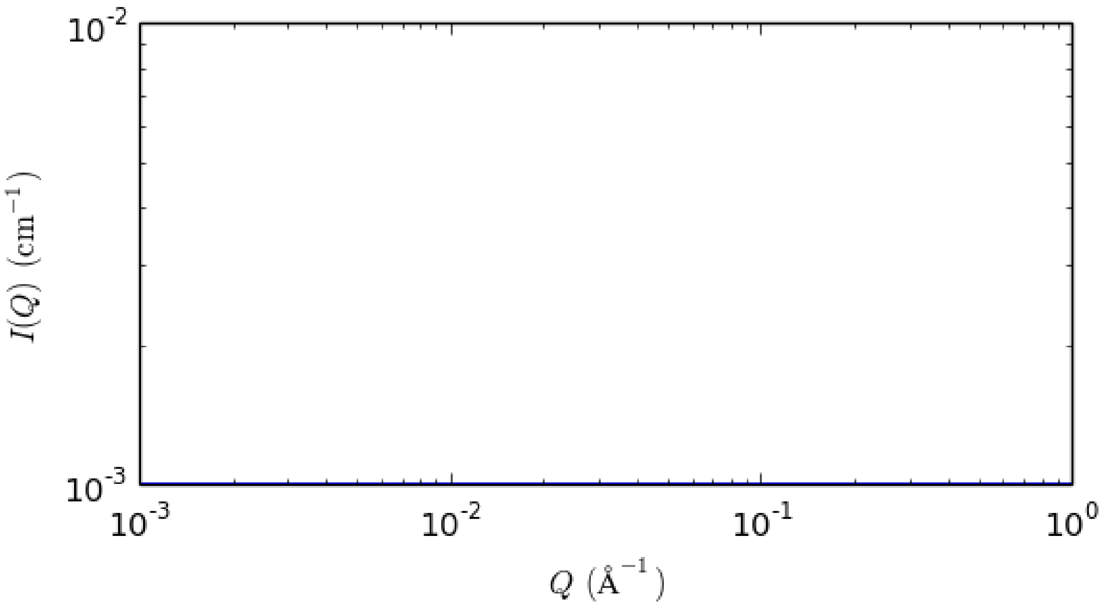


Fig. 1.115: 1D plot corresponding to the default parameters of the model.

References

A Z Akcasu, R Klein and B Hammouda, *Macromolecules*, 26 (1993) 4136.

2. Hammouda, *Advances in Polymer Science* 106 (1993) 87.

B. Hammouda, *SANS Toolbox* https://www.ncnr.nist.gov/staff/hammouda/the_sans_toolbox.pdf.

Authorship and Verification

- **Author:** Boualem Hammouda - NIST IGOR/DANSE **Date:** pre 2010
- **Converted to sasmodels by:** Paul Kienzle **Date:** July 18, 2016
- **Last Modified by:** Paul Butler **Date:** March 12, 2017
- **Last Reviewed by:** Paul Butler **Date:** March 12, 2017

spinodal

Spinodal decomposition model

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
gamma	Exponent	None	3
q_0	Correlation peak position	Å^{-1}	0.1

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

This model calculates the SAS signal of a phase separating system undergoing spinodal decomposition. The scattering intensity $I(q)$ is calculated as

$$I(q) = I_{max}\frac{(1+\gamma/2)x^2}{\gamma/2 + x^{2+\gamma}} + B$$

where $x = q/q_0$, q_0 is the peak position, I_{max} is the intensity at q_0 (parameterised as the $scale$ parameter), and B is a flat background. The spinodal wavelength, Λ, is given by $2\pi/q_0$.

The definition of I_{max} in the literature varies. Hashimoto *et al* (1991) define it as

$$I_{max} = \Lambda^3\Delta\rho^2$$

whereas Meier & Strobl (1987) give

$$I_{max} = V_z\Delta\rho^2$$

where V_z is the volume per monomer unit.

The exponent γ is equal to $d + 1$ for off-critical concentration mixtures (smooth interfaces) and $2d$ for critical concentration mixtures (entangled interfaces), where d is the dimensionality (ie, 1, 2, 3) of the system. Thus 2 <= γ <= 6. A transition from $\gamma = d + 1$ to $\gamma = 2d$ is expected near the percolation threshold.

As this function tends to zero as q tends to zero, in practice it may be necessary to combine it with another function describing the low-angle scattering, or to simply omit the low-angle scattering from the fit.

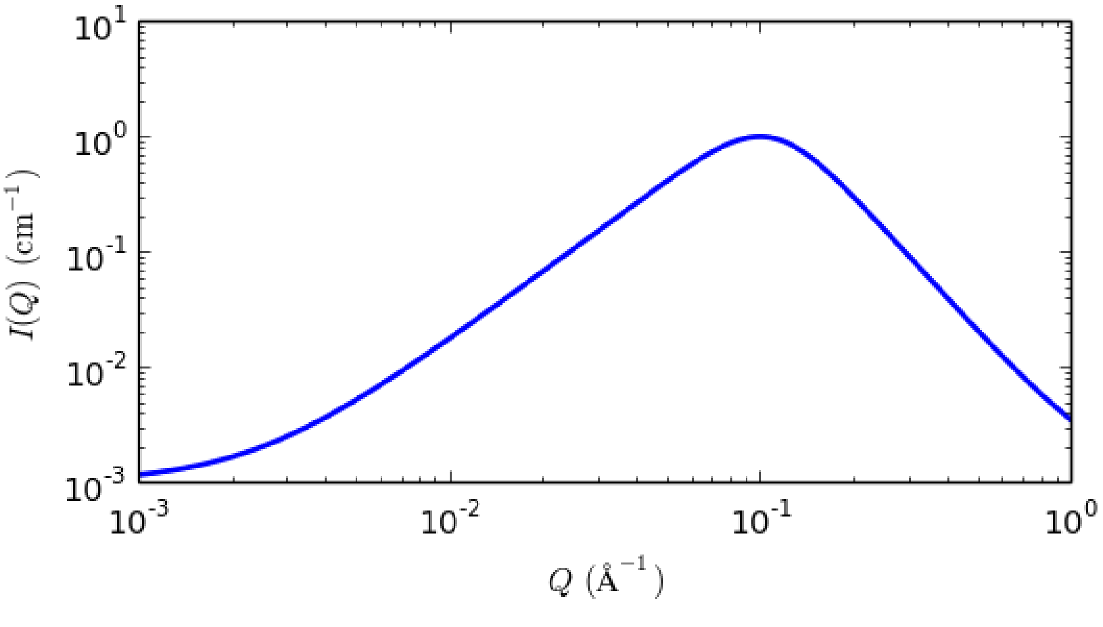


Fig. 1.116: 1D plot corresponding to the default parameters of the model.

References

H. Furukawa. Dynamics-scaling theory for phase-separating unmixing mixtures: Growth rates of droplets and scaling properties of autocorrelation functions. Physica A 123, 497 (1984).

H. Meier & G. Strobl. Small-Angle X-ray Scattering Study of Spinodal Decomposition in Polystyrene/Poly(styrene-co-bromostyrene) Blends. Macromolecules 20, 649-654 (1987).

T. Hashimoto, M. Takenaka & H. Jinnai. Scattering Studies of Self-Assembling Processes of Polymer Blends in Spinodal Decomposition. J. Appl. Cryst. 24, 457-466 (1991).

Revision History

- **Author:** Dirk Honecker **Date:** Oct 7, 2016
- **Revised:** Steve King **Date:** Oct 25, 2018

star_polymer

Star polymer model with Gaussian statistics

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
rg_squared	Ensemble radius of gyration SQUARED of the full polymer	$Å^2$	100
arms	Number of arms in the model	None	3

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

Calcuates the scattering from a simple star polymer with f equal Gaussian coil arms. A star being defined as a branched polymer with all the branches emanating from a common central (in the case of this model) point. It is derived as a special case of on the Benoit model for general branched polymers[1] as also used by Richter *et al.*[2]

For a star with f arms the scattering intensity $I(q)$ is calculated as

$$I(q) = \frac{2}{fv^2}\left[v - 1 + \exp(-v) + \frac{f-1}{2}\left[1 - \exp(-v)\right]^2\right]$$

where

$$v = \frac{uf}{(3f-2)}$$

and

$$u = \left\langle R_g^2 \right\rangle q^2$$

contains the square of the ensemble average radius-of-gyration of the full polymer while v contains the radius of gyration of a single arm R_{arm}. The two are related as:

$$R_{arm}^2 = \frac{f}{3f-2}R_g^2$$

Note that when there is only one arm, $f = 1$, the Debye Gaussian coil equation is recovered.

Note: Star polymers in solutions tend to have strong interparticle and osmotic effects. Thus the Benoit equation may not work well for many real cases. A newer model for star polymer incorporating excluded volume has been developed by Li et al in arXiv:1404.6269 [physics.chem-ph]. Also, at small q the scattering, i.e. the Guinier term, is not sensitive to the number of arms, and hence 'scale' here is simply $I(q = 0)$ as described for the *mono_gauss_coil* model, using volume fraction ϕ and volume V for the whole star polymer.

[1] H Benoit *J. Polymer Science*, 11, 507-510 (1953)

[2] D Richter, B. Farago, J. S. Huang, L. J. Fetters, B Ewen *Macromolecules*, 22, 468-472 (1989)

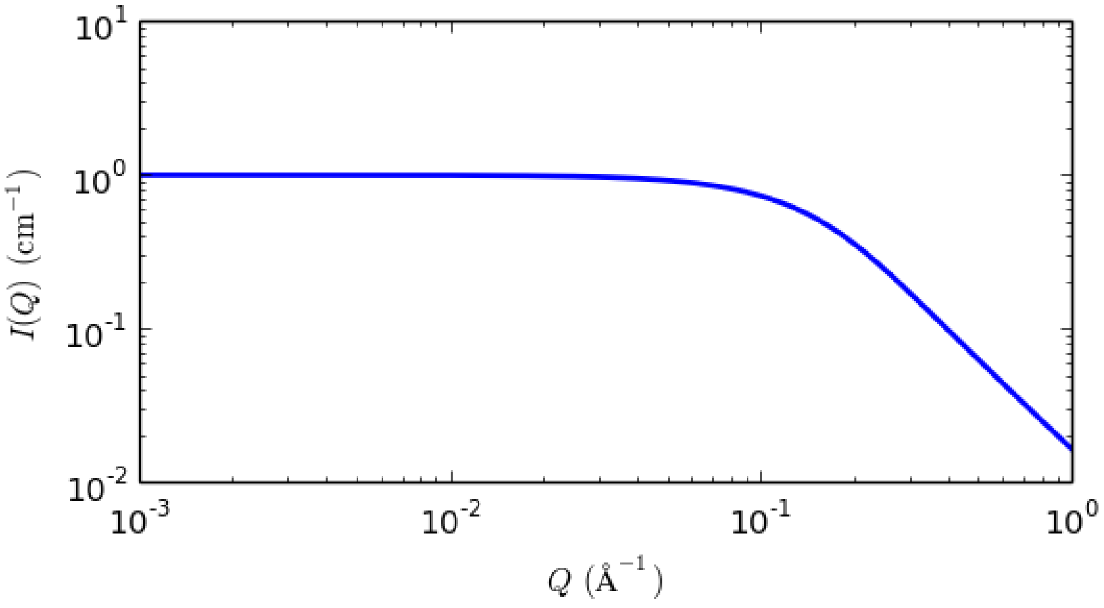


Fig. 1.117: 1D plot corresponding to the default parameters of the model.

References

Authorship and Verification

- **Author:** Kieran Campbell **Date:** July 24, 2012
- **Last Modified by:** Paul Butler **Date:** Auguts 26, 2017
- **Last Reviewed by:** Ziang Li and Richard Heenan **Date:** May 17, 2017

surface_fractal

Fractal-like aggregates based on the Mildner reference

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
radius	Particle radius	Å	10
fractal_dim_surf	Surface fractal dimension	None	2
cutoff_length	Cut-off Length	Å	500

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

This model calculates the scattering from fractal-like aggregates based on the Mildner reference.

Definition

The scattering intensity $I(q)$ is calculated as

$$\begin{aligned} I(q) &= \text{scale} \times P(q)S(q) + \text{background} \\ P(q) &= F(qR)^2 \\ F(x) &= \frac{3\left[\sin(x) - x\cos(x)\right]}{x^3} \\ S(q) &= \Gamma(5-D_S)\xi^{\,5-D_S}\left[1+(q\xi)^2\right]^{-(5-D_S)/2}\sin\left[-(5-D_S)\tan^{-1}(q\xi)\right]q^{-1} \\ \text{scale} &= \text{scale factor}\, NV^1(\rho_\text{particle} - \rho_\text{solvent})^2 \\ V &= \frac{4}{3}\pi R^3 \end{aligned}$$

where R is the radius of the building block, D_S is the **surface** fractal dimension, ξ is the cut-off length, ρ_solvent is the scattering length density of the solvent and ρ_particle is the scattering length density of particles.

Note: The surface fractal dimension is only valid if $1 < D_S < 3$. The result is only valid over a limited q range, $\frac{5}{3-D_S}\xi^{\,-1} < q < R^{-1}$. See the reference for details.

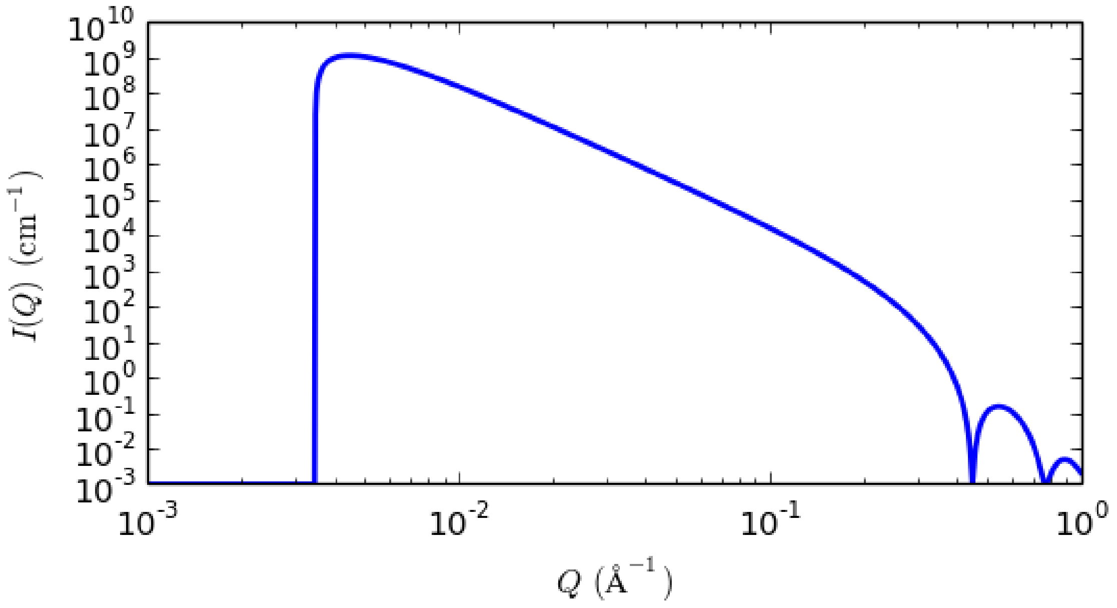


Fig. 1.118: 1D plot corresponding to the default parameters of the model.

References

D Mildner and P Hall, *J. Phys. D: Appl. Phys.*, 19 (1986) 1535-1545

teubner_strey

Teubner-Strey model of microemulsions

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
volfraction_a	Volume fraction of phase a	None	0.5
sld_a	SLD of phase a	10^{-6}Å^{-2}	0.3
sld_b	SLD of phase b	10^{-6}Å^{-2}	6.3
d	Domain size (periodicity)	Å	100
xi	Correlation length	Å	30

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

This model calculates the scattered intensity of a two-component system using the Teubner-Strey model. Unlike *dab* this function generates a peak. A two-phase material can be characterised by two length scales - a correlation length and a domain size (periodicity).

The original paper by Teubner and Strey defined the function as:

$$I(q) \propto \frac{1}{a_2 + c_1 q^2 + c_2 q^4} + \text{background}$$

where the parameters a_2, c_1 and c_2 are defined in terms of the periodicity, d, and correlation length ξ as:

$$\begin{aligned} a_2 &= \left[1 + \left(\frac{2\pi\xi}{d}\right)^2\right]^2 \\ c_1 &= -2\xi^2\left(\frac{2\pi\xi}{d}\right)^2 + 2\xi^2 \\ c_2 &= \xi^4 \end{aligned}$$

and thus, the periodicity, d is given by

$$d = 2\pi\left[\frac{1}{2}\left(\frac{a_2}{c_2}\right)^{1/2} - \frac{1}{4}\frac{c_1}{c_2}\right]^{-1/2}$$

and the correlation length, ξ, is given by

$$\xi = \left[\frac{1}{2}\left(\frac{a_2}{c_2}\right)^{1/2} + \frac{1}{4}\frac{c_1}{c_2}\right]^{-1/2}$$

Here the model is parameterised in terms of d and ξ and with an explicit volume fraction for one phase, ϕ_a, and contrast, $\delta\rho^2 = (\rho_a - \rho_b)^2$:

$$I(q) = \frac{8\pi\phi_a(1-\phi_a)(\Delta\rho)^2 c_2/\xi}{a_2 + c_1 q^2 + c_2 q^4}$$

where $8\pi\phi_a(1-\phi_a)(\Delta\rho)^2 c_2/\xi$ is the constant of proportionality from the first equation above.

In the case of a microemulsion, $a_2 > 0$, $c_1 < 0$, and $c_2 > 0$.

For 2D data, scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

References

M Teubner, R Strey, *J. Chem. Phys.*, 87 (1987) 3195

K V Schubert, R Strey, S R Kline and E W Kaler, *J. Chem. Phys.*, 101 (1994) 5343

H Endo, M Mihailescu, M. Monkenbusch, J Allgaier, G Gompper, D Richter, B Jakobs, T Sottmann, R Strey, and I Grillo, *J. Chem. Phys.*, 115 (2001), 580

two_lorentzian

This model calculates an empirical functional form for SAS data characterized by two Lorentzian-type functions.

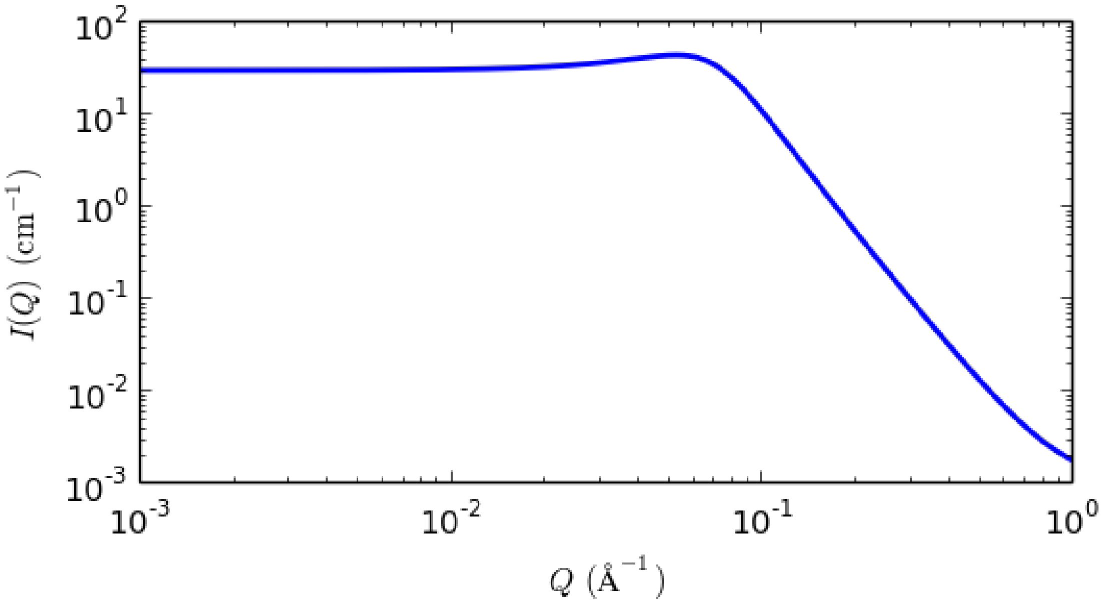


Fig. 1.119: 1D plot corresponding to the default parameters of the model.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
lorentz_scale_1	First power law scale factor	None	10
lorentz_length_1	First Lorentzian screening length	Å	100
lorentz_exp_1	First exponent of power law	None	3
lorentz_scale_2	Second scale factor for broad Lorentzian peak	None	1
lorentz_length_2	Second Lorentzian screening length	Å	10
lorentz_exp_2	Second exponent of power law	None	2

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

The scattering intensity $I(q)$ is calculated as

$$I(q) = \frac{A}{1 + (Q\xi_1)^n} + \frac{C}{1 + (Q\xi_2)^m} + \text{B}$$

where A = Lorentzian scale factor #1, C = Lorentzian scale #2, ξ_1 and ξ_2 are the corresponding correlation lengths, and n and m are the respective power law exponents (set $n = m = 2$ for Ornstein-Zernicke behaviour).

For 2D data the scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

References

None.

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Last Modified by:** Piotr rozyczko **Date:** January 29, 2016
- **Last Reviewed by:** Paul Butler **Date:** March 21, 2016

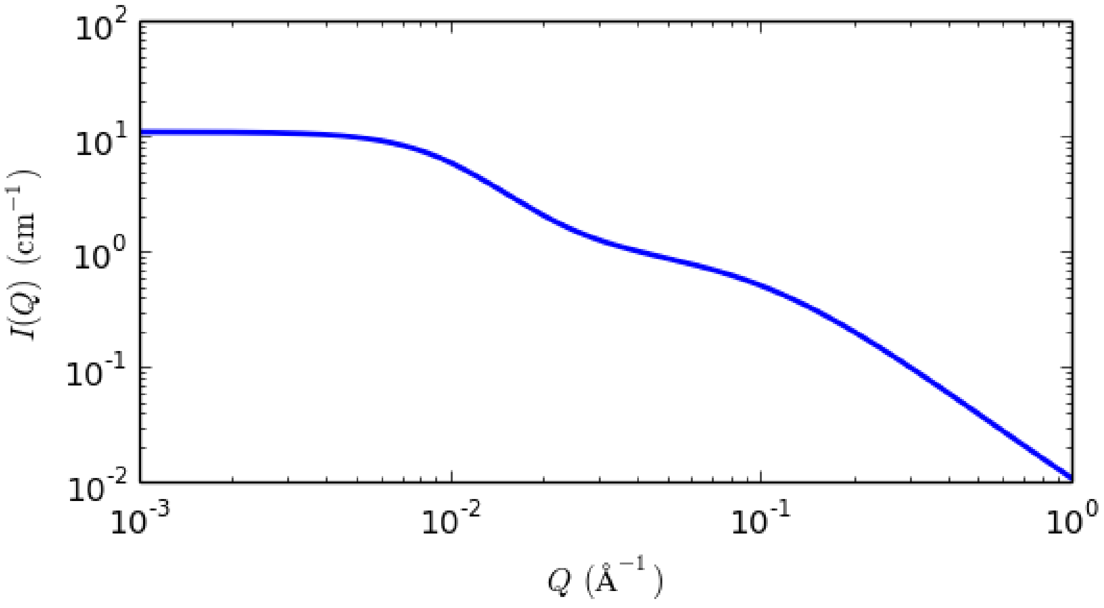


Fig. 1.120: 1D plot corresponding to the default parameters of the model.

two_power_law

This model calculates an empirical functional form for SAS data characterized by two power laws.

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
coefficent_1	coefficent A in low Q region	None	1
crossover	crossover location	$\mathrm{\AA}^{-1}$	0.04
power_1	power law exponent at low Q	None	1
power_2	power law exponent at high Q	None	4

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

The scattering intensity $I(q)$ is calculated as

$$I(q) = \begin{cases} Aq^{-m1} + \text{background} & q <= q_c \\ Cq^{-m2} + \text{background} & q > q_c \end{cases}$$

where q_c = the location of the crossover from one slope to the other, A = the scaling coefficent that sets the overall intensity of the lower Q power law region, $m1$ = power law exponent at low Q, and $m2$ = power law exponent at high Q. The scaling of the second power law region (coefficent C) is then automatically scaled to match the first by following formula:

$$C = \frac{Aq_c^{m2}}{q_c^{m1}}$$

Note: Be sure to enter the power law exponents as positive values!

For 2D data the scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

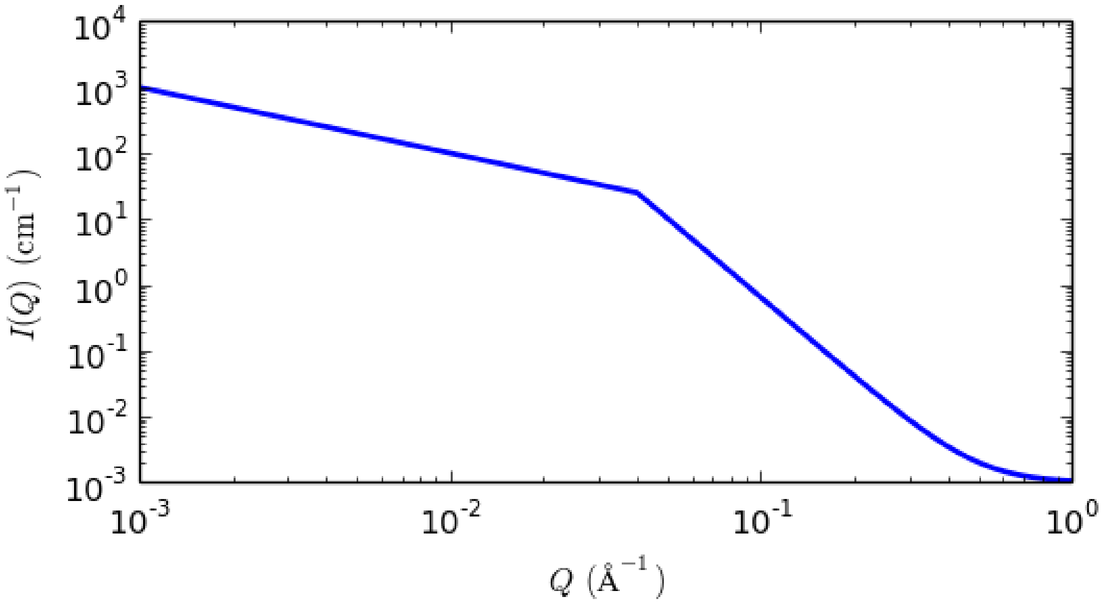


Fig. 1.121: 1D plot corresponding to the default parameters of the model.

References

None.

- **Author:** NIST IGOR/DANSE **Date:** pre 2010
- **Last Modified by:** Wojciech Wpotrzebowski **Date:** February 18, 2016
- **Last Reviewed by:** Paul Butler **Date:** March 21, 2016

unified_power_Rg

Unified Power Rg

Parameter	Description	Units	Default value
scale	Source intensity	None	1
background	Source background	cm^{-1}	0.001
level	Level number	None	1
rg[level]	Radius of gyration	Å	15.8
power[level]	Power	None	4
B[level]		cm^{-1}	4.5e-06
G[level]		cm^{-1}	400

The returned value is scaled to units of cm^{-1} sr^{-1}, absolute scale.

Definition

This model employs the empirical multiple level unified Exponential/Power-law fit method developed by Beaucage. Four functions are included so that 1, 2, 3, or 4 levels can be used. In addition a 0 level has been added which simply calculates

$$I(q) = \text{scale}/q + \text{background}$$

The Beaucage method is able to reasonably approximate the scattering from many different types of particles, including fractal clusters, random coils (Debye equation), ellipsoidal particles, etc.

The model works best for mass fractal systems characterized by Porod exponents between 5/3 and 3. It should not be used for surface fractal systems. Hammouda (2010) has pointed out a deficiency in the way this model handles the transitioning between the Guinier and Porod regimes and which can create artefacts that appear as kinks in the fitted model function.

Also see the Guinier_Porod model.

The empirical fit function is:

$$I(q) = \text{background} + \sum_{i=1}^{N} \Big[G_i \exp\Big(-\frac{q^2 R_{gi}^2}{3}\Big) + B_i \exp\Big(-\frac{q^2 R_{g(i+1)}^2}{3}\Big) \Big(\frac{1}{q_i^*}\Big)^{P_i} \Big]$$

where

$$q_i^* = q \left[\operatorname{erf}\left(\frac{q R_{gi}}{\sqrt{6}} \right) \right]^{-3}$$

For each level, the four parameters G_i, R_{gi}, B_i and P_i must be chosen. Beaucage has an additional factor k in the definition of q_i^* which is ignored here.

For example, to approximate the scattering from random coils (Debye equation), set R_{gi} as the Guinier radius, $P_i = 2$, and $B_i = 2G_i/R_{gi}$

See the references for further information on choosing the parameters.

For 2D data: The 2D scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

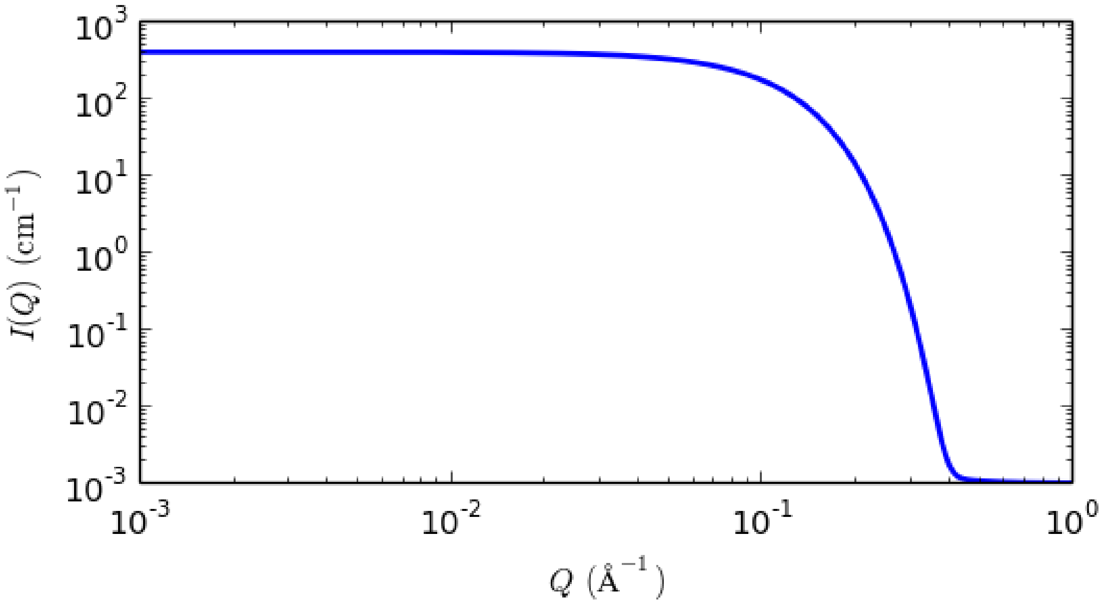


Fig. 1.122: 1D plot corresponding to the default parameters of the model.

References

G Beaucage, *J. Appl. Cryst.*, 28 (1995) 717-728

G Beaucage, *J. Appl. Cryst.*, 29 (1996) 134-146

B Hammouda, *Analysis of the Beaucage model, J. Appl. Cryst.*, (2010), 43, 1474-1478

1.1.8 Structure Factors

hardsphere

Hard sphere structure factor, with Percus-Yevick closure

Parameter	Description	Units	Default value
radius_effective	effective radius of hard sphere	Å	50
volfraction	volume fraction of hard spheres	None	0.2

The returned value is a dimensionless structure factor, $S(q)$.

Calculate the interparticle structure factor for monodisperse spherical particles interacting through hard sphere (excluded volume) interactions. May be a reasonable approximation for other shapes of particles that freely rotate, and for moderately polydisperse systems. Though strictly the maths needs to be modified (no Beta(Q) correction yet in sasview).

radius_effective is the effective hard sphere radius. volfraction is the volume fraction occupied by the spheres.

In sasview the effective radius may be calculated from the parameters used in the form factor $P(q)$ that this $S(q)$ is combined with.

For numerical stability the computation uses a Taylor series expansion at very small qR, there may be a very minor glitch at the transition point in some circumstances.

The S(Q) uses the Percus-Yevick closure where the interparticle potential is

$$U(r) = \begin{cases} \infty & r < 2R \\ 0 & r \geq 2R \end{cases}$$

where r is the distance from the center of the sphere of a radius R.

For a 2D plot, the wave transfer is defined as

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

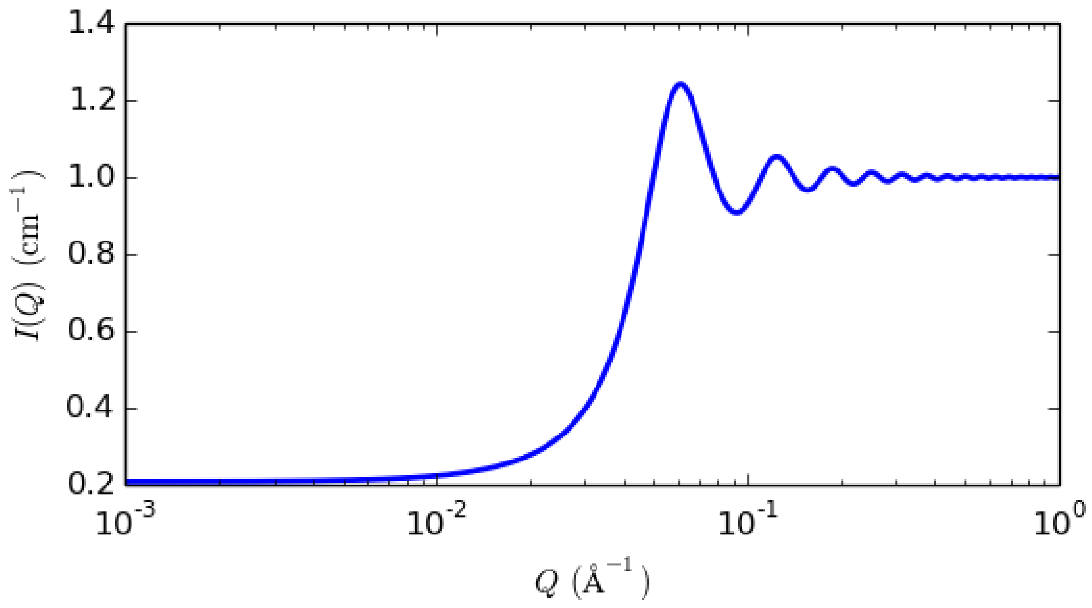


Fig. 1.123: 1D plot corresponding to the default parameters of the model.

References

J K Percus, J Yevick, *J. Phys. Rev.*, 110, (1958) 1

hayter_msa

Hayter-Penfold rescaled MSA, charged sphere, interparticle S(Q) structure factor

Parameter	Description	Units	Default value
radius_effective	effective radius of charged sphere	Å	20.75
volfraction	volume fraction of spheres	None	0.0192
charge	charge on sphere (in electrons)	e	19
temperature	temperature, in Kelvin, for Debye length calculation	K	318.16
concentration_salt	conc of salt, moles/litre, 1:1 electolyte, for Debye length	M	0
dielectconst	dielectric constant (relative permittivity) of solvent, default water, for Debye length	None	71.08

The returned value is a dimensionless structure factor, $S(q)$.

This calculates the structure factor (the Fourier transform of the pair correlation function $g(r)$) for a system of charged, spheroidal objects in a dielectric medium. When combined with an appropriate form factor (such as sphere, core+shell, ellipsoid, etc), this allows for inclusion of the interparticle interference effects due to screened coulomb repulsion between charged particles.

This routine only works for charged particles. If the charge is set to zero the routine may self-destruct! For non-charged particles use a hard sphere potential.

The salt concentration is used to compute the ionic strength of the solution which in turn is used to compute the Debye screening length. At present there is no provision for entering the ionic strength directly nor for use of any multivalent salts, though it should be possible to simulate the effect of this by increasing the salt concentration. The counterions are also assumed to be monovalent.

In sasview the effective radius may be calculated from the parameters used in the form factor $P(q)$ that this $S(q)$ is combined with.

The computation uses a Taylor series expansion at very small rescaled qR, to avoid some serious rounding error issues, this may result in a minor artefact in the transition region under some circumstances.

For 2D data, the scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

References

J B Hayter and J Penfold, *Molecular Physics*, 42 (1981) 109-118

J P Hansen and J B Hayter, *Molecular Physics*, 46 (1982) 651-656

squarewell

Square well structure factor, with MSA closure

Parameter	Description	Units	Default value
radius_effective	effective radius of hard sphere	Å	50
volfraction	volume fraction of spheres	None	0.04
welldepth	depth of well, epsilon	kT	1.5
wellwidth	width of well in diameters (=2R) units, must be > 1	diameters	1.2

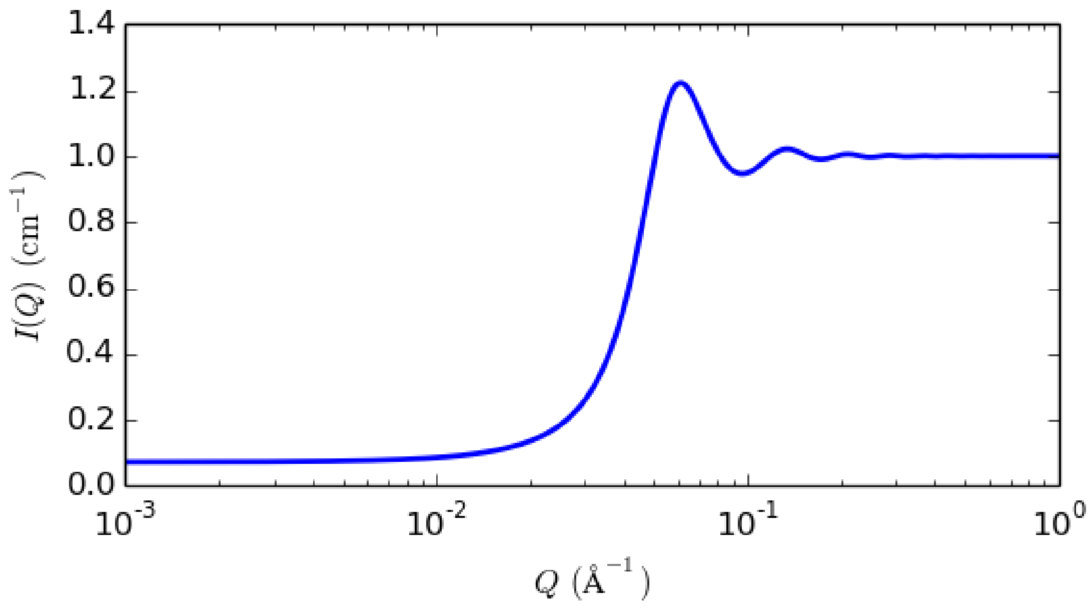


Fig. 1.124: 1D plot corresponding to the default parameters of the model.

The returned value is a dimensionless structure factor, $S(q)$.

This calculates the interparticle structure factor for a square well fluid spherical particles. The mean spherical approximation (MSA) closure was used for this calculation, and is not the most appropriate closure for an attractive interparticle potential. This solution has been compared to Monte Carlo simulations for a square well fluid, showing this calculation to be limited in applicability to well depths $\epsilon < 1.5$ kT and volume fractions $\phi < 0.08$.

Positive well depths correspond to an attractive potential well. Negative well depths correspond to a potential "shoulder", which may or may not be physically reasonable. The stickyhardsphere model may be a better choice in some circumstances. Computed values may behave badly at extremely small qR.

The well width (λ) is defined as multiples of the particle diameter $(2R)$.

The interaction potential is:

$$U(r) = \begin{cases} \infty, r < 2R \\ -\varepsilon, 2R \leq r \leq 2R\lambda \\ 0, r \geq 2R \end{cases}$$

$$U(r) = \begin{cases} \infty & r < 2R \\ -\epsilon & 2R \leq r < 2R\lambda \\ 0 & r \geq 2R\lambda \end{cases}$$

where r is the distance from the center of the sphere of a radius R.

In sasview the effective radius may be calculated from the parameters used in the form factor $P(q)$ that this $S(q)$ is combined with.

For 2D data: The 2D scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

References

R V Sharma, K C Sharma, *Physica*, 89A (1977) 213.

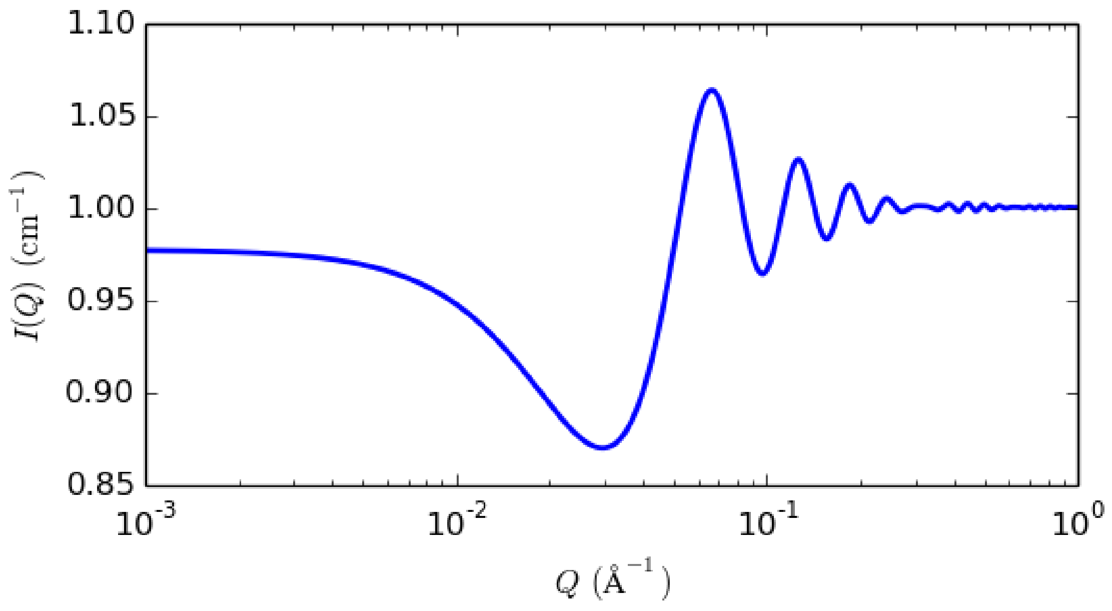


Fig. 1.125: 1D plot corresponding to the default parameters of the model.

stickyhardsphere

Sticky hard sphere structure factor, with Percus-Yevick closure

Parameter	Description	Units	Default value
radius_effective	effective radius of hard sphere	Å	50
volfraction	volume fraction of hard spheres	None	0.2
perturb	perturbation parameter, epsilon	None	0.05
stickiness	stickiness, tau	None	0.2

The returned value is a dimensionless structure factor, $S(q)$.

This calculates the interparticle structure factor for a hard sphere fluid with a narrow attractive well. A perturbative solution of the Percus-Yevick closure is used. The strength of the attractive well is described in terms of "stickiness" as defined below.

The perturb (perturbation parameter), ϵ, should be held between 0.01 and 0.1. It is best to hold the perturbation parameter fixed and let the "stickiness" vary to adjust the interaction strength. The stickiness, τ, is defined in the equation below and is a function of both the perturbation parameter and the interaction strength. τ and ϵ are defined in terms of the hard sphere diameter ($\sigma = 2R$), the width of the square well, Δ (same units as R), and the depth of the well, U_o, in units of kT. From the definition, it is clear that smaller τ means stronger attraction.

$$\tau = \frac{1}{12\epsilon}\exp(u_o/kT)$$
$$\epsilon = \Delta/(\sigma + \Delta)$$

where the interaction potential is

$$U(r) = \begin{cases} \infty & r < \sigma \\ -U_o & \sigma \le r \le \sigma + \Delta \\ 0 & r > \sigma + \Delta \end{cases}$$

The Percus-Yevick (PY) closure was used for this calculation, and is an adequate closure for an attractive interparticle potential. This solution has been compared to Monte Carlo simulations for a square well fluid, with good agreement.

The true particle volume fraction, ϕ, is not equal to h, which appears in most of the reference. The two are related in equation (24) of the reference. The reference also describes the relationship between this perturbation solution and the original sticky hard sphere (or adhesive sphere) model by Baxter.

NB: The calculation can go haywire for certain combinations of the input parameters, producing unphysical solutions - in this case errors are reported to the command window and the $S(q)$ is set to -1 (so it will disappear on a log-log plot). Use tight bounds to keep the parameters to values that you know are physical (test them) and keep nudging them until the optimization does not hit the constraints.

In sasview the effective radius may be calculated from the parameters used in the form factor $P(q)$ that this $S(q)$ is combined with.

For 2D data the scattering intensity is calculated in the same way as 1D, where the q vector is defined as

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

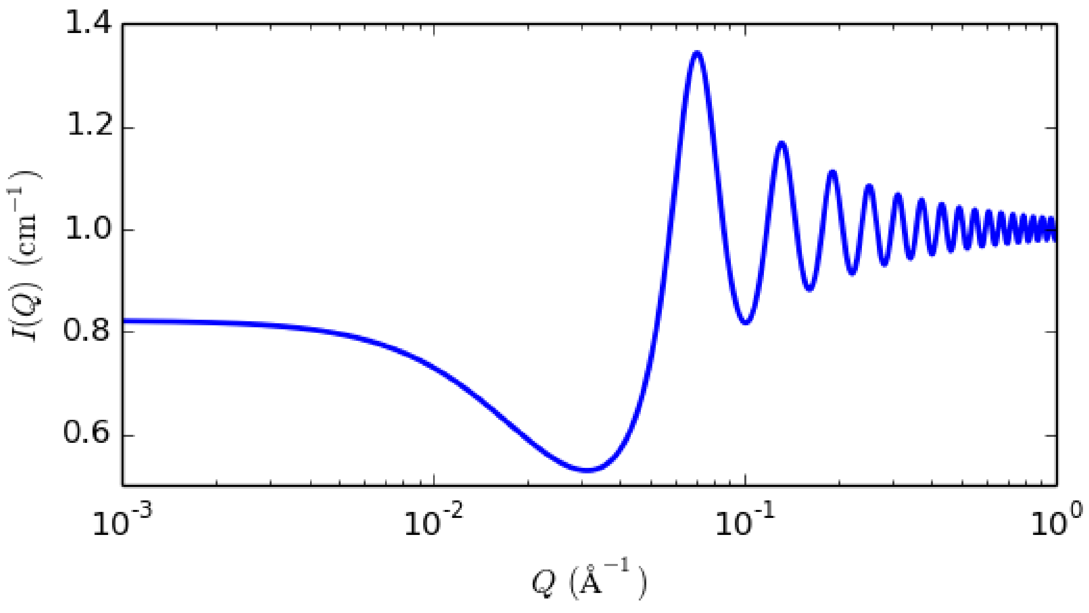


Fig. 1.126: 1D plot corresponding to the default parameters of the model.

References

S V G Menon, C Manohar, and K S Rao, *J. Chem. Phys.*, 95(12) (1991) 9186-9190

1.2 Menu Bar

The menu bar at the top of the *SasView* window gives you access to additional features of the program:

1.2.1 File

The File option allows you load data into *SasView* for analysis, or to save the work you have been doing.

Data can be loaded one file at a time, or by selecting multiple files, or by loading an entire folder of files (in which case *SasView* will attempt to make an intelligent guess as to what to load based on the file formats it recognises in the folder!).

A *SasView* session can also be saved and reloaded as an 'Analysis' (an individual model fit or invariant calculation, etc), or as a 'Project' (everything you have done since starting your *SasView* session).

1.2.2 Edit

The Edit option allows you to:

- undo/redo your recent changes;
- copy and paste parameters between *SasView* analysis windows;
- copy parameters from a *SasView* analysis window to the Clipboard as either tab-delimited text (compatible with Microsoft Excel) or LaTex-wrapped text;
- generate a summary 'Report' of the most recent analysis performed;
- reset parameter values in the P(r) Inversion analysis page.

1.2.3 View

The View option allows you to:

- show the Batch Fitting Results Panel if it has been closed;
- show/hide the Data Explorer Panel;
- show/hide the Toolbar of icons below the Menu Bar;
- select the default location that *SasView* looks in for data to analyse (the *SasView* installation directory, the initial default, or a custom folder). NB: any change only takes effect when *SasView* is restarted;
- change the default assignment of categories (*Shapes*, *Shape-independent*, *Structure Factor*) for fitting model functions.

1.2.4 Tools

The Tools option provides access to a comprehensive range of tools and utilities. See *Tools & Utilities* for more information.

1.2.5 Window

The Window option allows you to select which *SasView* windows are visible.

1.2.6 Analysis

The Analysis option provides access to the key functionality of *SasView*:

- Model Fitting;
- P(r) Inversion;
- Invariant Analysis;
- Correlation Function Analysis (*SasView* 4.1 and later)

See *Fitting & Other Analyses* for more information.

1.2.7 Fitting

The Fitting option allows you to:

- create a new FitPage;
- change optimiser (under Fit Options);
- view fit parameter correlations, distributions, and convergence traces (under Fit Results);

- create/edit a Plugin Model.

Additional functionality is available under this menu option during particular types of model fitting, including:

- setting up a Constrained or Simultaneous Fit;
- combining a Batch Fit (an obscure capability);
- setting up Chain Fitting.

1.2.8 Help

The Help option provides access to:

- this help documentation;
- a *Tutorials* on using *SasView* (in pdf format);
- information on how to acknowledge *SasView* in publications;
- information about the version of *SasView* you are using;
- the *Model Marketplace*;
- a check to see if there is a more recent version of *SasView*.

Note: This help document was last changed by Steve King, 10Oct2016

1.3 Fitting & Other Analyses

Note: In Windows use [Alt]-[Cursor left] to return to the previous page

1.3.1 Fitting Documentation

Note: In Windows use [Alt]-[Cursor left] to return to the previous page

Fitting

Note: If some code blocks are not readable, expand the documentation window

Preparing to fit data

To fit some data you must first load some data, activate one or more data sets, send those data sets to fitting, and select a model to fit to each data set.

Instructions on how to load and activate data are in the section *Loading Data*.

SasView can fit data in one of three ways:

- in *Single* fit mode - individual data sets are fitted independently one-by-one

- in *Simultaneous* fit mode - multiple data sets are fitted simultaneously to the *same* model with/without constrained parameters (this might be useful, for example, if you have measured the same sample at different contrasts)
- in *Batch* fit mode - multiple data sets are fitted sequentially to the *same* model (this might be useful, for example, if you have performed a kinetic or time-resolved experiment and have *lots* of data sets!)

Selecting a model

The models in SasView are grouped into categories. By default these consist of:

- *Cylinder* - cylindrical shapes (disc, right cylinder, cylinder with endcaps etc)
- *Ellipsoid* - ellipsoidal shapes (oblate,prolate, core shell, etc)
- *Parellelepiped* - as the name implies
- *Sphere* - sheroidal shapes (sphere, core multishell, vesicle, etc)
- *Lamellae* - lamellar shapes (lamellar, core shell lamellar, stacked lamellar, etc)
- *Shape-Independent* - models describing structure in terms of density correlation functions, fractals, peaks, power laws, etc
- *Paracrystal* - semi ordered structures (bcc, fcc, etc)
- *Structure Factor* - S(Q) models
- *Plugin Models* - User-created (custom/non-library) Python models

Use the *Category* drop-down menu to chose a category of model, then select a model from the drop-down menu beneath. A graph of the chosen model, calculated using default parameter values, will appear. The graph will update dynamically as the parameter values are changed.

You can decide your own model categorizations using the *Category Manager*.

Once you have selected a model you can read its help documentation by clicking on the *Description* button to the right.

Show 1D/2D

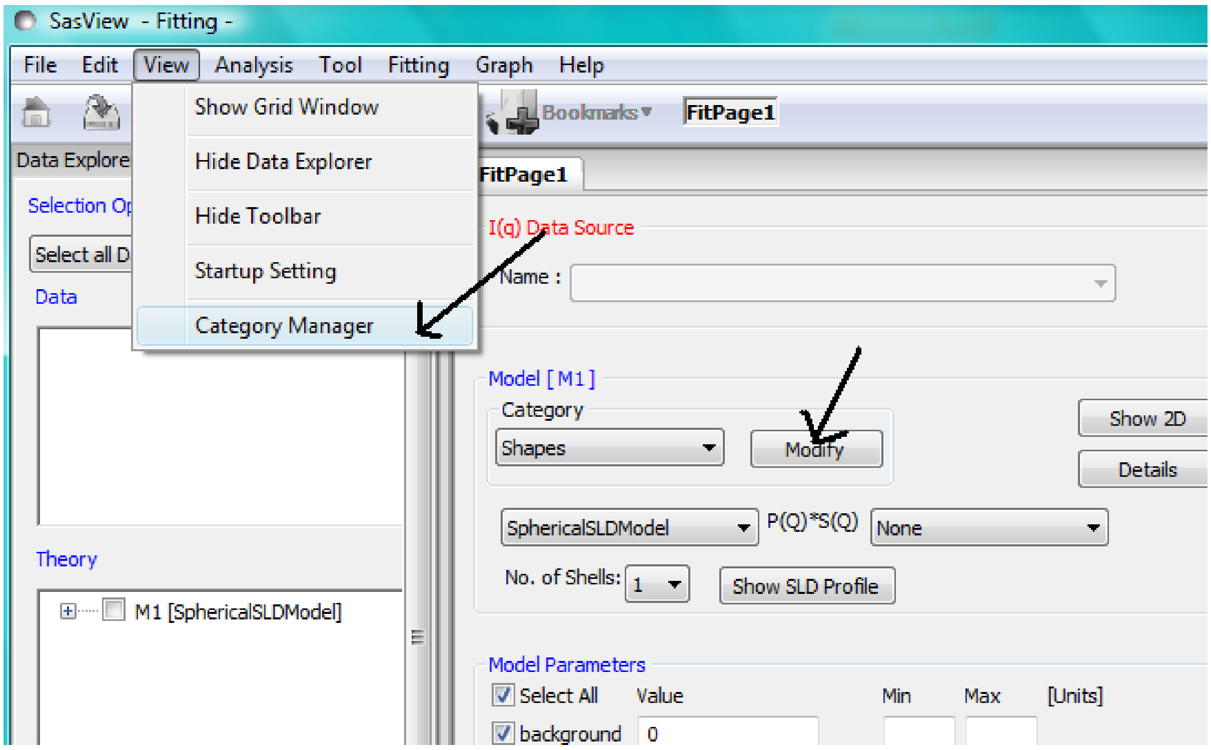
Models are normally fitted to 1D (ie, I(Q) vs Q) data sets, but some models in SasView can also be fitted to 2D (ie, I(Qx,Qy) vs Qx vs Qy) data sets.

NB: Magnetic scattering can only be fitted in SasView in 2D.

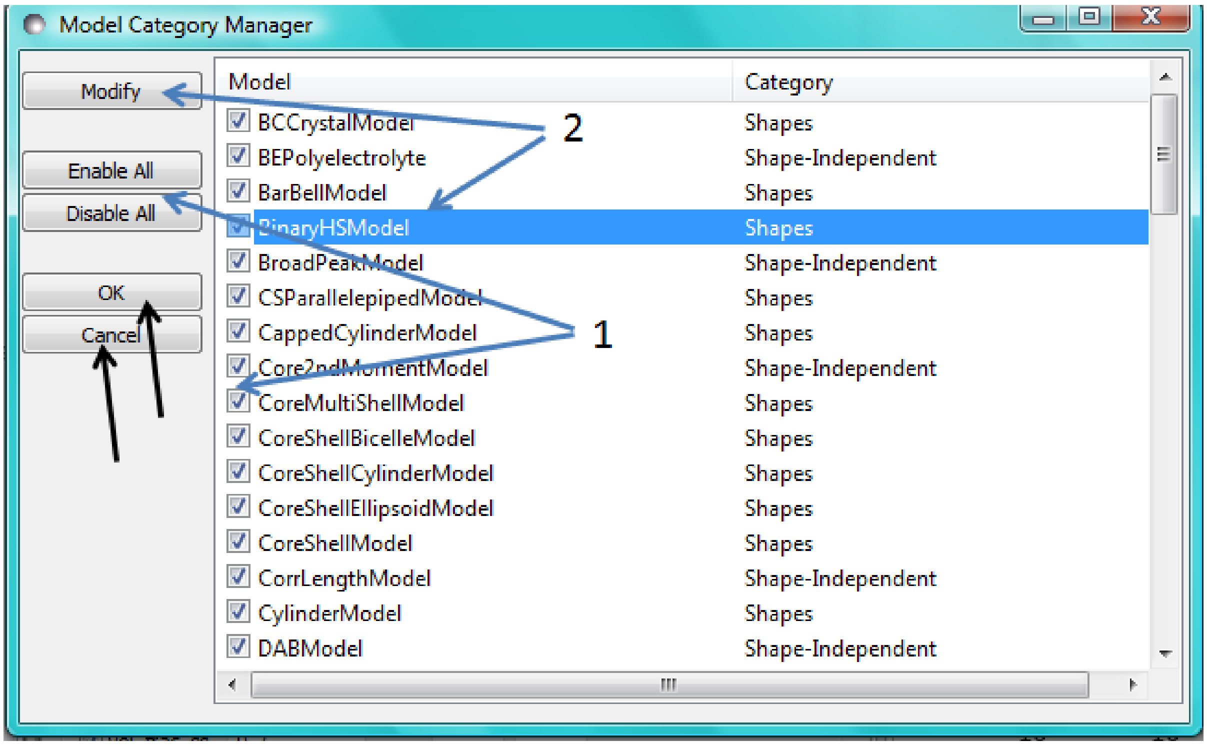
To activate 2D fitting mode, click the *Show 2D* button on the *Fit Page*. To return to 1D fitting model, click the same button (which will now say *Show 1D*).

Category Manager

To change the model categorizations, either choose *Category Manager* from the *View* option on the menubar, or click on the *Modify* button on the *Fit Page*.



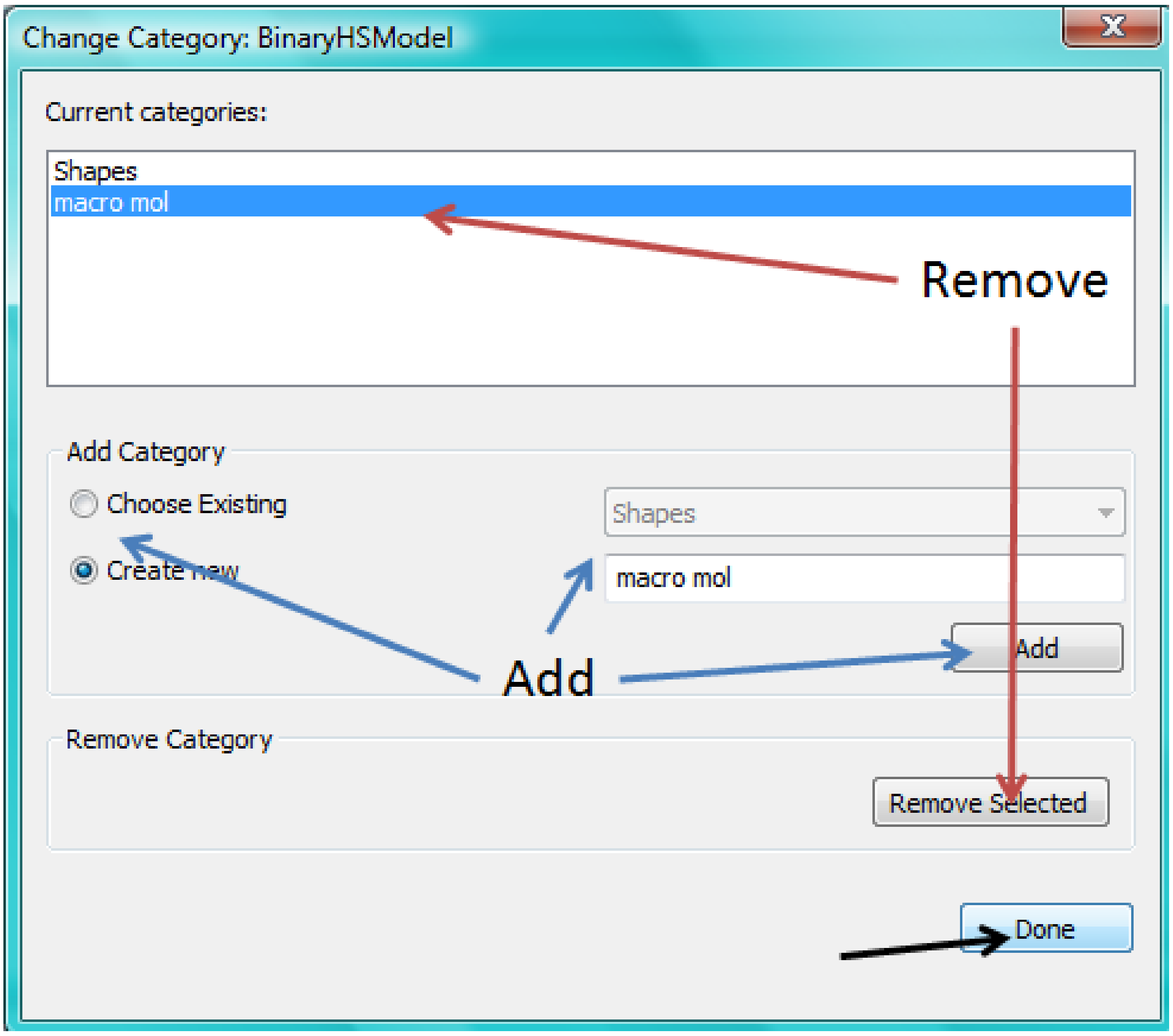
The categorization of all models except the user supplied Plugin Models can be reassigned, added to, and removed using *Category Manager*. Models can also be hidden from view in the drop-down menus.



Model	Category
☑ BCCrystalModel	Shapes
☑ BEPolyelectrolyte	Shape-Independent
☑ BarBellModel	Shapes
☑ BinaryHSModel	Shapes
☑ BroadPeakModel	Shape-Independent
☑ CSParallelepipedModel	Shapes
☑ CappedCylinderModel	Shapes
☑ Core2ndMomentModel	Shape-Independent
☑ CoreMultiShellModel	Shapes
☑ CoreShellBicelleModel	Shapes
☑ CoreShellCylinderModel	Shapes
☑ CoreShellEllipsoidModel	Shapes
☑ CoreShellModel	Shapes
☑ CorrLengthModel	Shape-Independent
☑ CylinderModel	Shapes
☑ DABModel	Shape-Independent

Changing category

To change category, highlight a model in the list by left-clicking on its entry and then click the *Modify* button. Use the *Change Category* panel that appears to make the required changes.



To create a category for the selected model, click the *Add* button. In order to delete a category, select the category name and click the *Remove Selected* button. Then click *Done*.

Showing/hiding models

Use the *Enable All / Disable All* buttons and the check boxes beside each model to select the models to show/hide. To apply the selection, click *Ok*. Otherwise click *Cancel*.

NB: It may be necessary to change to a different category and then back again before any changes take effect.

Model Functions

For a complete list of all the library models available in SasView, see the Model Documentation .

It is also possible to add your own models.

Adding your own Models

There are essentially three ways to generate new fitting models for SasView:

- Using the SasView *New Plugin Model* helper dialog (best for beginners and/or relatively simple models)

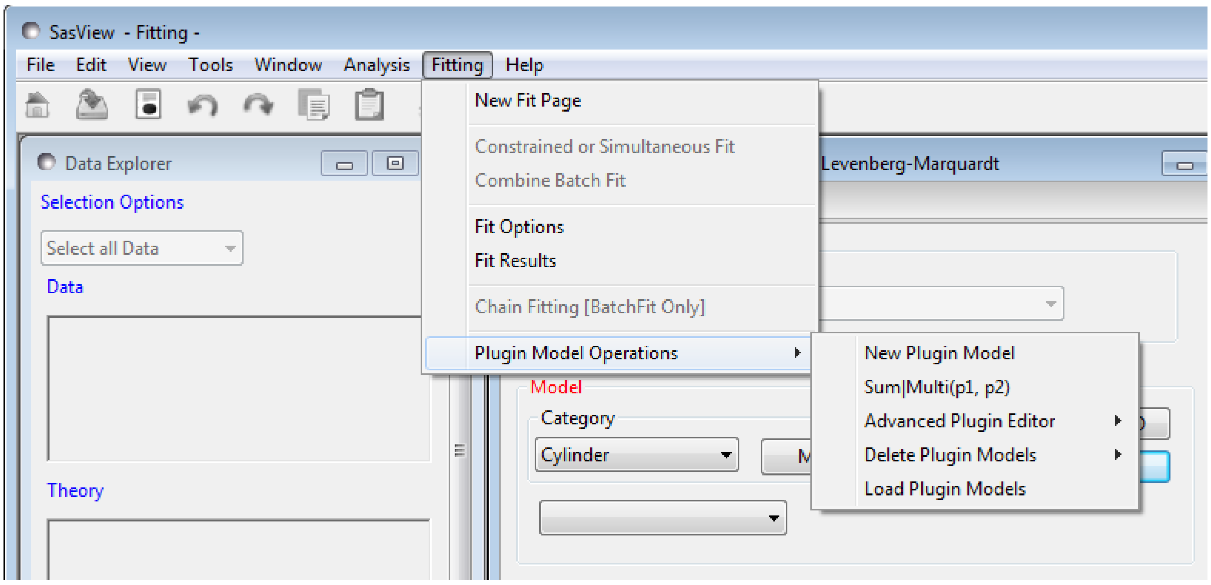
- By copying/editing an existing model (this can include models generated by the New Plugin Model* dialog) in the *Python Shell-Editor Tool* or *Advanced Plugin Editor* (suitable for all use cases)
- By writing a model from scratch outside of SasView (only recommended for code monkeys!)

Please read the guidance on *Writing a Plugin Model* before proceeding.

To be found by SasView your model must reside in the *~.sasview\plugin_models* folder.

Plugin Model Operations

From the *Fitting* option in the menu bar, select *Plugin Model Operations*

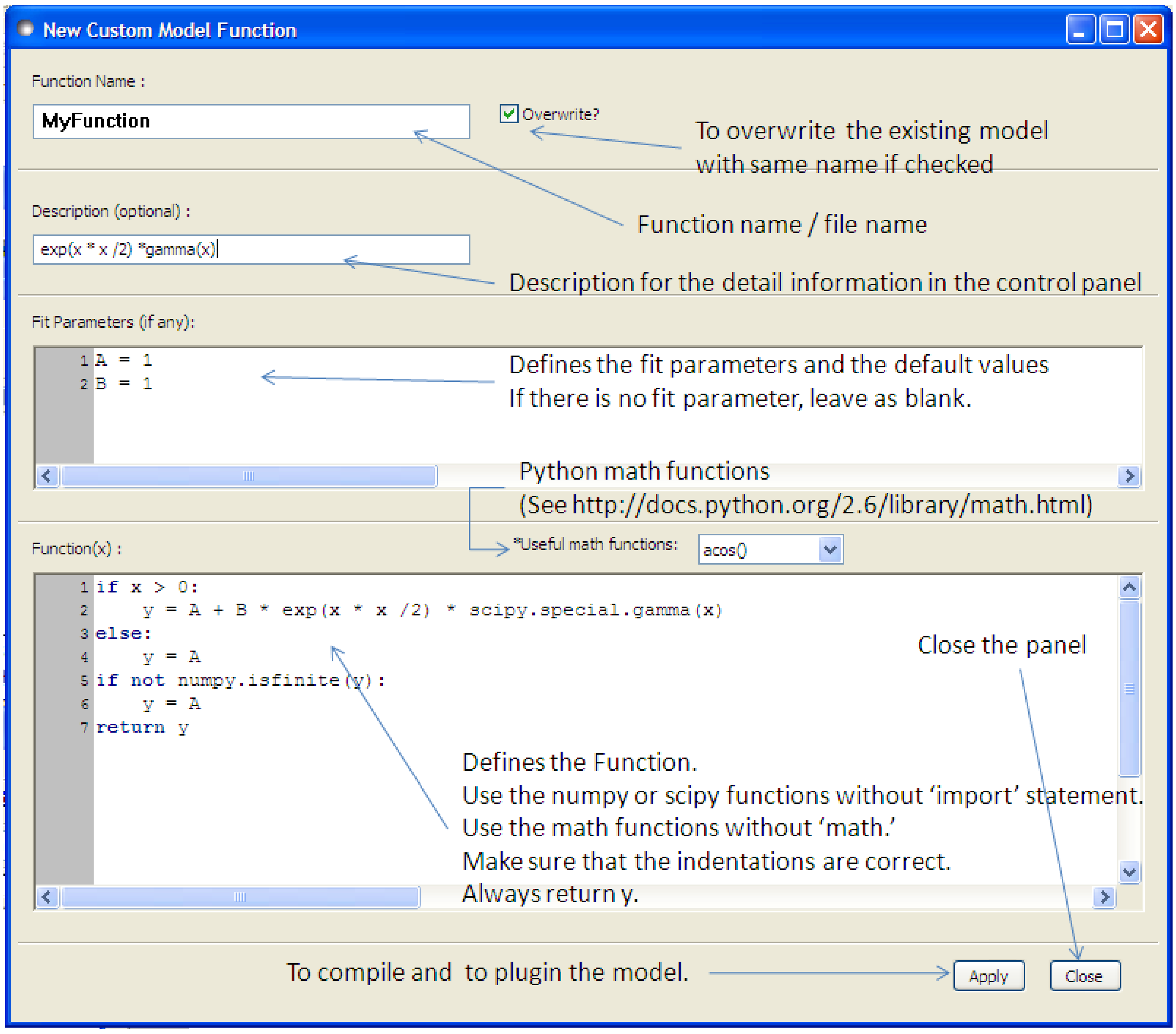


and then one of the sub-options

- *New Plugin Model* - to create a plugin model template with a helper dialog
- *Sum|Multi(p1,p2)* - to create a plugin model by summing/multiplying *existing models* in the model library
- *Advanced Plugin Editor* - to create/edit a plugin model in a Python shell
- *Delete Plugin Models* - to delete a plugin model
- *Load Plugin Models* - to (re-)load plugin models

New Plugin Model

Relatively straightforward models can be programmed directly from the SasView GUI using the *New Plugin Model Function*.



Function Name : MyFunction

Overwrite? ✓

To overwrite the existing model with same name if checked

Function name / file name

Description (optional) : exp(x * x /2) *gamma(x)

Description for the detail information in the control panel

Fit Parameters (if any):

```
A = 1
B = 1
```

Defines the fit parameters and the default values
If there is no fit parameter, leave as blank.

Python math functions
(See http://docs.python.org/2.6/library/math.html)

*Useful math functions: acos()

Function(x) :

```
if x > 0:
    y = A + B * exp(x * x /2) * scipy.special.gamma(x)
else:
    y = A
if not numpy.isfinite(y):
    y = A
return y
```

Defines the Function.
Use the numpy or scipy functions without 'import' statement.
Use the math functions without 'math.'
Make sure that the indentations are correct.
Always return y.

Close the panel

To compile and to plugin the model.

When using this feature, be aware that even if your code has errors, including syntax errors, a model file is still generated. When you then correct the errors and click 'Apply' again to re-compile you will get an error informing you that the model already exists if the 'Overwrite' box is not checked. In this case you will need to supply a new model function name. By default the 'Overwrite' box is *checked*.

A model file generated by this option can be viewed and further modified using the *Advanced Plugin Editor* .

Note that the New Plugin Model Feature currently does not allow for parameters to be polydisperse. However they can be edited in the Advanced Editor.

SasView version 4.2 made it possible to specify whether a plugin created with the *New Plugin Model* dialog is actually a form factor P(Q) or a structure factor S(Q). To do this, simply add one or other of the following lines under the *import* statements.

For a form factor:

```
form_factor = True
```

or for a structure factor:

```
structure_factor = True
```

If the plugin is a structure factor it is *also* necessary to add two variables to the parameter list:

```
parameters = [
                ['radius_effective', '', 1, [0.0, numpy.inf], 'volume', ''],
```

(continues on next page)

(continued from previous page)

```
                    ['volfraction', '', 1, [0.0, 1.0], '', ''],
                    [...],
```

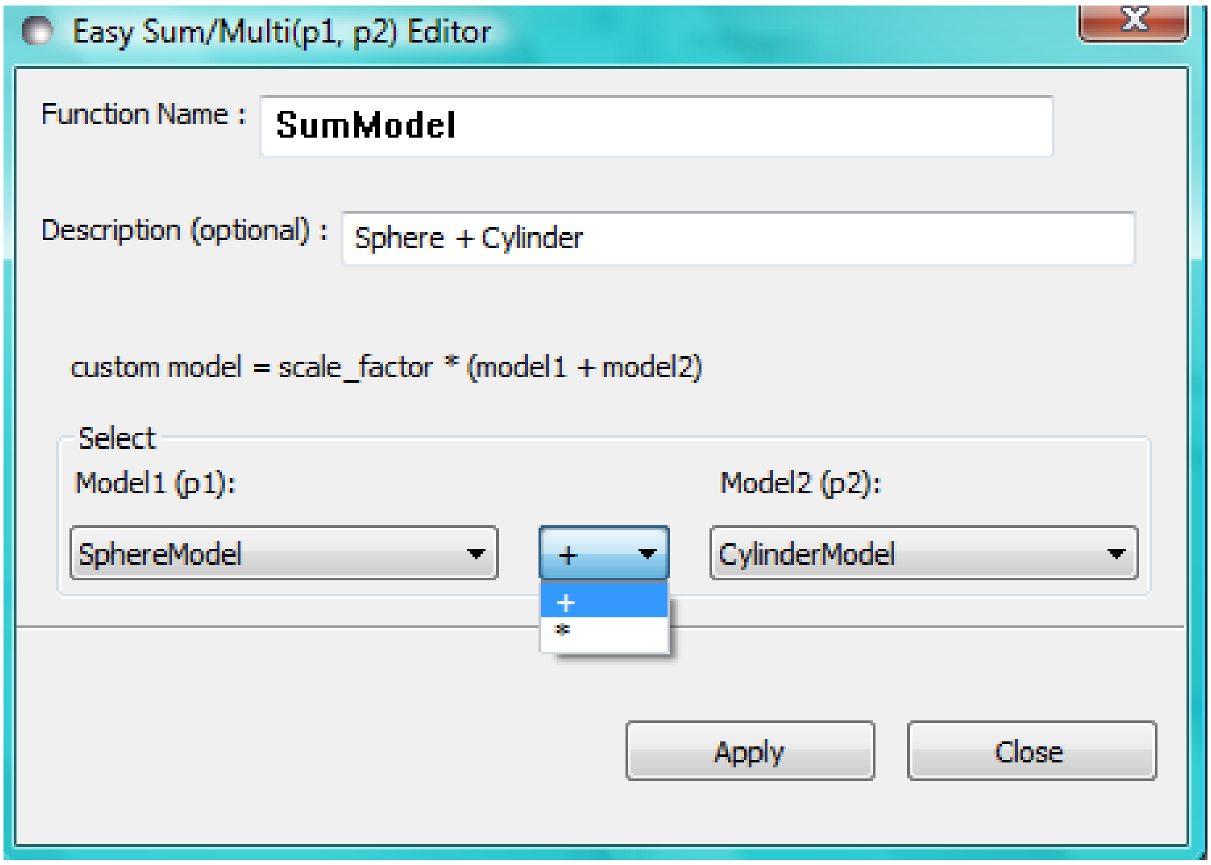
and to the declarations of the functions Iq and Iqxy::

```
def Iq(x , radius_effective, volfraction, ...):

def Iqxy(x, y, radius_effective, volfraction, ...):
```

Such a plugin should then be available in the S(Q) drop-down box on a FitPage (once a P(Q) model has been selected).

Sum|Multi(p1,p2)


Easy Sum/Multi(p1, p2) Editor

Function Name : SumModel

Description (optional) : Sphere + Cylinder

custom model = scale_factor * (model1 + model2)

Select

Model1 (p1): SphereModel

+

Model2 (p2): CylinderModel

This option creates a custom Plugin Model of the form:

```
Plugin Model = scale_factor * {(scale_1 * model_1) +/- (scale_2 * model_2)} +␣
↪background
```

or:

```
Plugin Model = scale_factor * (model1 * model2) + background
```

In the *Easy Sum/Multi Editor* give the new model a function name and brief description (to appear under the *Details* button on the *FitPage*). Then select two existing models, as p1 and p2, and the required operator, '+' or '*' between them. Finally, click the *Apply* button to generate and test the model and then click *Close*.

Any changes to a plugin model generated in this way only become effective *after* it is re-selected from the plugin models drop-down menu on the FitPage. If the model is not listed you can force a recompilation of the plugins by selecting *Fitting* > *Plugin Model Operations* > *Load Plugin Models*.

SasView version 4.2 introduced a much simplified and more extensible structure for plugin models generated through the Easy Sum/Multi Editor. For example, the code for a combination of a sphere model with a power law model now looks like this:

```
from sasmodels.core import load_model_info
from sasmodels.sasview_model import make_model_from_info

model_info = load_model_info('sphere+power_law')
model_info.name = 'MyPluginModel'
model_info.description = 'sphere + power_law'
Model = make_model_from_info(model_info)
```

To change the models or operators contributing to this plugin it is only necessary to edit the string in the brackets after *load_model_info*, though it would also be a good idea to update the model name and description too!!!

The model specification string can handle multiple models and combinations of operators (+ or *) which are processed according to normal conventions. Thus 'model1+model2*model3' would be valid and would multiply model2 by model3 before adding model1. In this example, parameters in the *FitPage* would be prefixed A (for model2), B (for model3) and C (for model1). Whilst this might appear a little confusing, unless you were creating a plugin model from multiple instances of the same model the parameter assignments ought to be obvious when you load the plugin.

If you need to include another plugin model in the model specification string, just prefix the name of that model with *custom*. For instance:

```
sphere+custom.MyPluginModel
```

To create a P(Q)*S(Q) model use the @ symbol instead of * like this:

```
sphere@hardsphere
```

This streamlined approach to building complex plugin models from existing library models, or models available on the *Model Marketplace*, also permits the creation of P(Q)*S(Q) plugin models, something that was not possible in earlier versions of SasView.

Advanced Plugin Editor

Selecting this option shows all the plugin models in the plugin model folder, on Windows this is

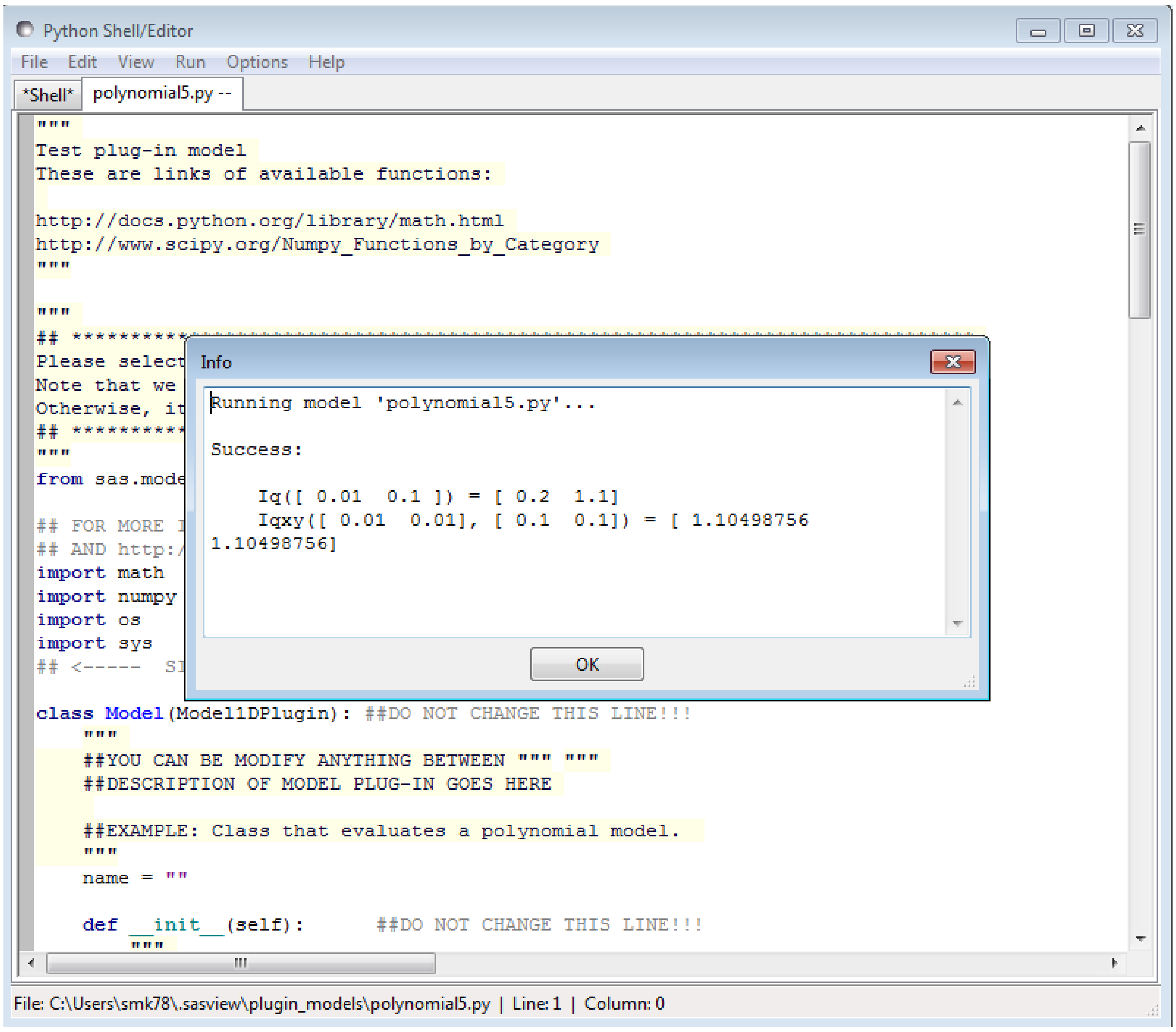
C:\Users\{username}.sasview\plugin_models

You can edit, modify, and save the Python code in any of these models using the *Advanced Plugin Model Editor*. Note that this is actually the same tool as the *Python Shell-Editor Tool* .

For details of the SasView plugin model format see *Writing a Plugin Model* .

Note: Model files generated with the Sum/Multi option are still using the SasView 3.x model format. Unless you are confident about what you are doing, it is recommended that you only modify lines denoted with the ## <—— comments!

When editing is complete, select *Run* > *Check Model* from the *Advanced Plugin Model Editor* menu bar. An *Info* box will appear with the results of the compilation and model unit tests. The model will only be usable if the tests 'pass'.



```
"""
Test plug-in model
These are links of available functions:

http://docs.python.org/library/math.html
http://www.scipy.org/Numpy_Functions_by_Category
"""

"""
## **********
Please select
Note that we
Otherwise, it
## **********
"""
from sas.mode

## FOR MORE I
## AND http:/
import math
import numpy
import os
import sys
## <-----   SI

class Model(Model1DPlugin): ##DO NOT CHANGE THIS LINE!!!
    """
    ##YOU CAN BE MODIFY ANYTHING BETWEEN """ """
    ##DESCRIPTION OF MODEL PLUG-IN GOES HERE

    ##EXAMPLE: Class that evaluates a polynomial model.
    """
    name = ""

    def __init__(self):      ##DO NOT CHANGE THIS LINE!!!
        """
```

```
Running model 'polynomial5.py'...

Success:

    Iq([ 0.01  0.1 ]) = [ 0.2  1.1]
    Iqxy([ 0.01  0.01], [ 0.1  0.1]) = [ 1.10498756
1.10498756]
```

File: C:\Users\smk78\.sasview\plugin_models\polynomial5.py | Line: 1 | Column: 0

To use the model, go to the relevant *Fit Page*, select the *Plugin Models* category and then select the model from the drop-down menu.

Any changes to a plugin model generated in this way only become effective *after* it is re-selected from the model drop-down menu on the FitPage.

Delete Plugin Models

Simply highlight the plugin model to be removed. The operation is final!!!

NB: Models shipped with SasView cannot be removed in this way.

Load Plugin Models

This option loads (or re-loads) all models present in the *~.sasview\plugin_models* folder.

Fitting Options

It is possible to specify which optimiser SasView should use to fit the data, and to modify some of the configurational parameters for each optimiser.

From *Fitting* in the menu bar select *Fit Options*, then select one of the following optimisers:

- DREAM
- Levenberg-Marquardt
- Quasi-Newton BFGS
- Differential Evolution
- Nelder-Mead Simplex

The DREAM optimiser is the most sophisticated, but may not necessarily be the best option for fitting simple models. If uncertain, try the Levenberg-Marquardt optimiser initially.

These optimisers form the *Bumps* package written by P Kienzle. For more information on each optimiser, see the *Fitting Documentation*.

Fitting Limits

By default, *SasView* will attempt to model fit the full range of the data; ie, across all *Q* values. If necessary, however, it is possible to specify only a sub-region of the data for fitting.

In a *FitPage* or *BatchPage* change the *Q* values in the *Min* and/or *Max* text boxes. Vertical coloured bars will appear on the graph with the data and 'theory' indicating the current *Q* limits (red = *Qmin*, purple = *Qmax*).

To return to including all data in the fit, click the *Reset* button.

Shortcuts

Copy/Paste Parameters

It is possible to copy the parameters from one *Fit Page* and to paste them into another *Fit Page* using the same model.

To *copy* parameters, either:

- Select *Edit -> Copy Params* from the menu bar, or
- Use Ctrl(Cmd on Mac) + Left Mouse Click on the *Fit Page*.

To *paste* parameters, either:

- Select *Edit -> Paste Params* from the menu bar, or
- Use Ctrl(Cmd on Mac) + Shift + Left-click on the *Fit Page*.

If either operation is successful a message will appear in the info line at the bottom of the SasView window.

Bookmark

To *Bookmark* a *Fit Page* either:

- Select a *Fit Page* and then click on *Bookmark* in the tool bar, or
- Right-click and select the *Bookmark* in the popup menu.

Status Bar & Console

The status bar is located at the bottom of the SasView window and displays messages, hints, warnings and errors.

At the right-hand side of the status bar is a button marked *Console*. The *Console* displays available message history and some run-time traceback information.

During a long task the *Console* can also be used to monitor the progress.

Single Fit Mode

NB: Before proceeding, ensure that the Single Mode radio button at the bottom of the Data Explorer is checked (see the section Loading Data).

This mode fits one data set.

When data is sent to the fitting it is plotted in a graph window as markers.

If a graph does not appear, or a graph window appears but is empty, then the data has not loaded correctly. Check to see if there is a message in the *Status Bar & Console* or in the *Console* window.

Assuming the data has loaded correctly, when a model is selected a green model calculation (or what SasView calls a 'Theory') line will appear in the earlier graph window, and a second graph window will appear displaying the residuals (the difference between the experimental data and the theory) at the same X-data values. See *Assessing Fit Quality*.

The objective of model-fitting is to find a *physically-plausible* model, and set of model parameters, that generate a theory that reproduces the experimental data and minimizes the values of the residuals.

Change the default values of the model parameters by hand until the theory line starts to represent the experimental data. Then check the tick boxes alongside the 'background' and 'scale' parameters. Click the *Fit* button. SasView will optimise the values of the 'background' and 'scale' and also display the corresponding uncertainties on the optimised values.

Note: If the uncertainty on a fitted parameter is unrealistically large, or if it displays as NaN, the model is most likely a poor representation of the data, the parameter in question is highly correlated with one or more of the other fitted parameters, or the model is relatively insensitive to the value of that particular parameter.

In the bottom left corner of the *Fit Page* is a box displaying a normalised value of the statistical χ^2 parameter (the reduced χ^2, See *Assessing Fit Quality*) returned by the optimiser.

Now check the box for another model parameter and click *Fit* again. Repeat this process until all relevant parameters are checked and have been optimised. As the fit of the theory to the experimental data improves, the value of 'Reduced Chi2' will decrease. A good model fit should produce values of Reduced Chi2 close to one, and certainly << 100. See *Assessing Fit Quality*.

SasView has a number of different optimisers (see the section *Fitting Options*). The DREAM optimiser is the most sophisticated, but may not necessarily be the best option for fitting simple models. If uncertain, try the Levenberg-Marquardt optimiser initially.

Simultaneous Fit Mode

NB: Before proceeding, ensure that the Single Mode radio button at the bottom of the Data Explorer is checked (see the section Loading Data).

This mode is an extension of the *Single Fit Mode* that allows for some relatively extensive constraints between fitted parameters in a single *FitPage* or between several *FitPage*'s (eg, to constrain all fitted parameters to be the same in a contrast series of *FitPages* except for the solvent sld parameter, constrain the length to be twice that of the radius in a single *FitPage*, fix the radius of the sphere in one *FitPage* to be the same as the radius of the cylinder in a second *FitPage*, etc).

If the data to be fit are in multiple files, load each file, then select each file in the *Data Explorer*, and *Send To Fitting*. If multiple data sets are in one file, load that file, *Unselect All Data*, select just those data sets to be fitted, and *Send To Fitting*. Either way, the result should be that for n data sets you have $2n$ graphs (n of the data and model fit, and n of the resulting residuals). But it may be helpful to minimise the residuals plots for clarity. Also see *Assessing Fit Quality*.

NB: If you need to use a custom Plugin Model, you must ensure that model is available first (see Adding your own Models).

Method

Now go to each *FitPage* in turn and:

> Select the required category and model;
>
> Unselect all the model parameters;
>
> Enter some starting guesses for the parameters;
>
> Enter any parameter limits (recommended);
>
> Select which parameters will refine (selecting all is generally a bad idea…);

When done, select *Constrained or Simultaneous Fit* under *Fitting* in the menu bar.

In the *Const & Simul Fit* page that appears, select which data sets are to be simultaneously fitted (this will probably be all of them or you would not have loaded them in the first place!).

To tie parameters between the data sets with constraints, check the 'yes' radio button next to *Add Constraint?* in the *Fit Constraints* box.

To constrain all identically named parameters to fit *simultaneously* to the same value across all the *Fitpages* use the *Easy Setup* drop-down buttons in the *Const & Simul Fit* page.

NB: You can only constrain parameters that are set to refine.

Constraints will generally be of the form

> Mi Parameter1 = Mj.Parameter1

however the text box after the '=' sign can be used to adjust this relationship; for example

> Mi Parameter1 = scalar * Mj.Parameter1

A 'free-form' constraint box is also provided.

Many constraints can be entered for a single fit.

When ready, click the *Fit* button on the *Const & Simul Fit* page, NOT the *Fit* button on the individual *FitPage*'s.

The results of the model-fitting will be returned to each of the individual *FitPage*'s.

Note that the Reduced Chi2 value returned is the SUM of the Reduced Chi2 of each fit. To see the Reduced Chi2 value for a specific *FitPage*, click the *Compute* button at the bottom of that *FitPage* to recalculate. Note that in doing so the degrees of freedom will be set to Npts. See *Assessing Fit Quality*. Moreover in the case of constraints the degrees of freedom are less than one might think due to those constraints.

Batch Fit Mode

NB: Before proceeding, ensure that the Single Mode radio button at the bottom of the Data Explorer is checked (see the section Loading Data). The Batch Mode button will be used later on!

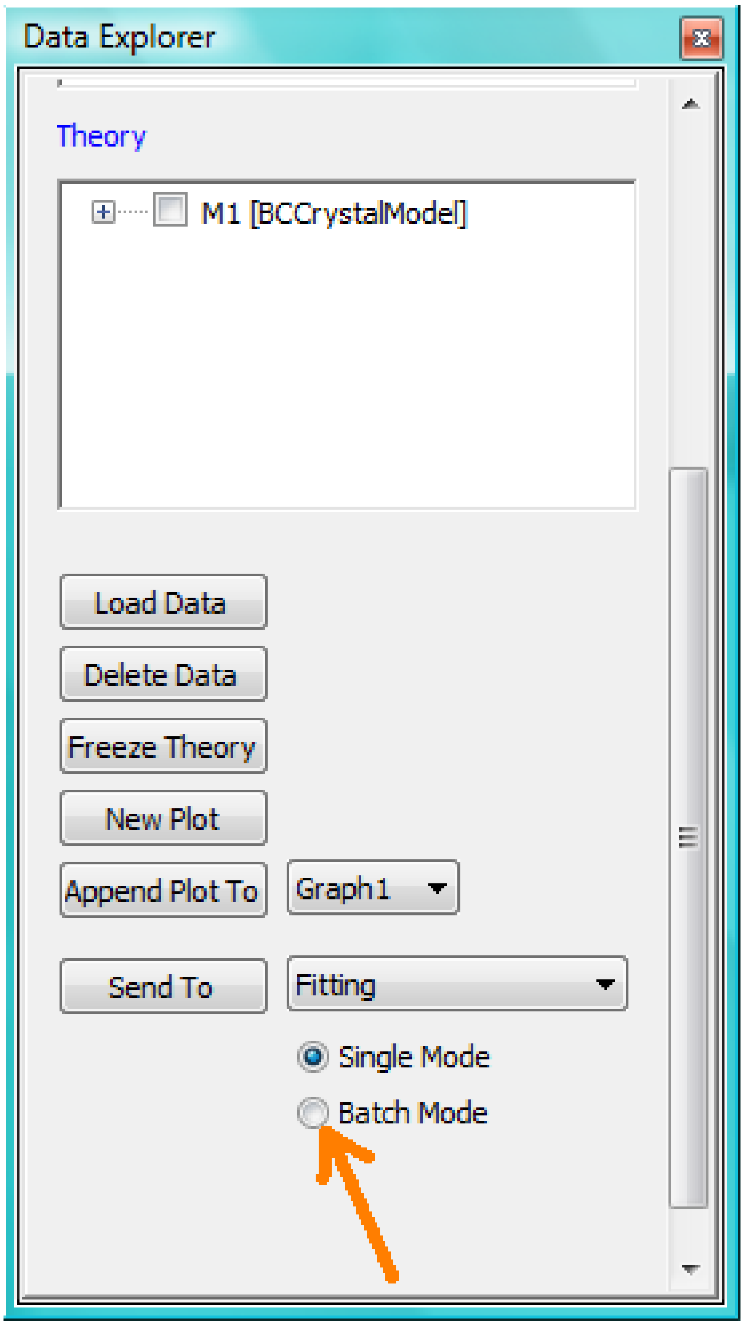
This mode *sequentially* fits two or more data sets *to the same model*. Unlike in simultaneous fitting, in batch fitting it is not possible to constrain fit parameters between data sets.

If the data to be fit are in multiple files, load each file in the *Data Explorer*. If multiple data sets are in one file, load just that file. *Unselect All Data*, then select a single initial data set to be fitted. Fit that selected data set as described above under *Single Fit Mode*.

NB: If you need to use a custom Plugin Model, you must ensure that model is available first (see Adding your own Models).

Method

Now *Select All Data* in the *Data Explorer*, check the *Batch Mode* radio button at the bottom of that panel and *Send To Fitting*. A *BatchPage* will be created.



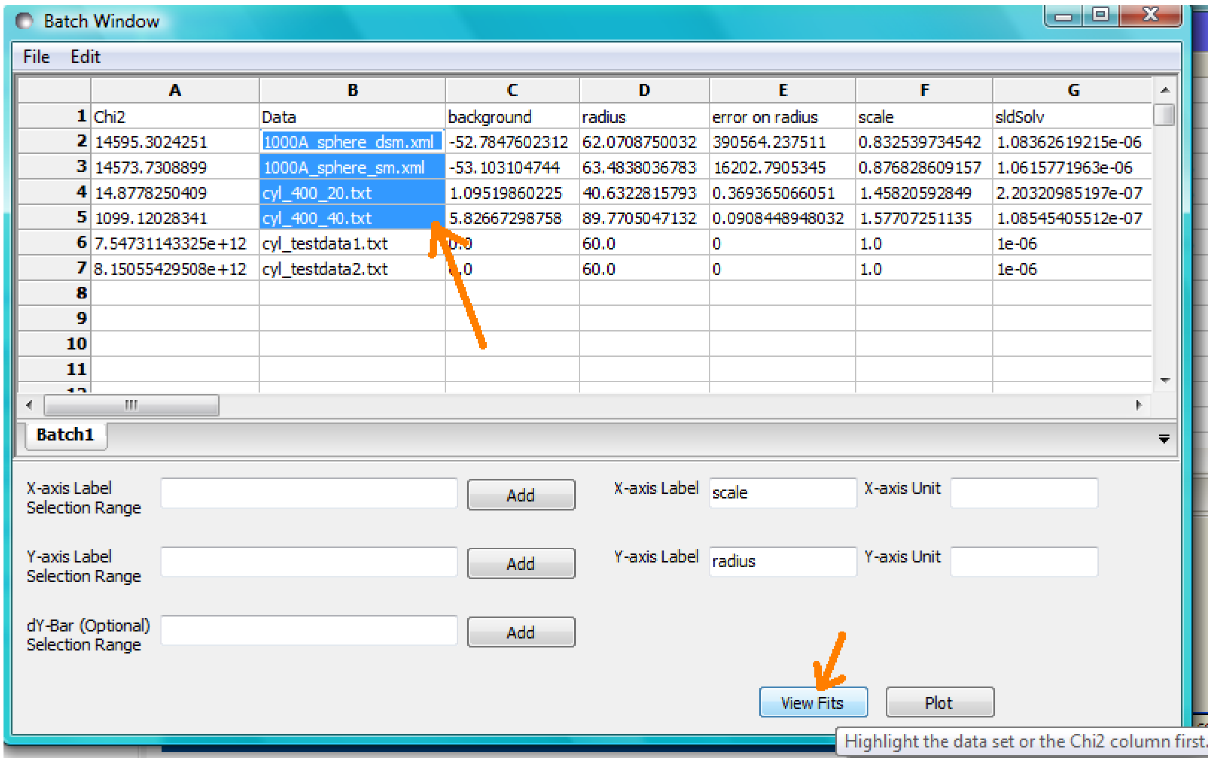
NB: The Batch Page can also be created by checking the Batch Mode radio button and selecting New Fit Page under Fitting in the menu bar.

Using the drop-down menus in the *BatchPage*, now set up the *same* data set with the *same* model that you just fitted in single fit mode. A quick way to set the model parameter values is to just copy them from the earlier Single Fit. To do this, go back to the Single Fit *FitPage*, select *Copy Params* under *Edit* in the menu bar, then go back to the *BatchPage* and *Paste Params*.

When ready, use the *Fit* button on the *BatchPage* to perform the fitting, NOT the *Fit* button on the individual *FitPage*'s.

Unlike in single fit mode, the results of batch fits are not returned to the *BatchPage*. Instead, a spreadsheet-like *Grid Window* will appear.

If you want to visually check a graph of a particular fit, click on the name of a *Data set* in the *Grid Window* and then click the *View Fits* button. The data and the model fit will be displayed. If you select mutliple data sets they will all appear on one graph.



	A	B	C	D	E	F	G
1	Chi2	Data	background	radius	error on radius	scale	sldSolv
2	14595.3024251	1000A_sphere_dsm.xml	-52.7847602312	62.0708750032	390564.237511	0.832539734542	1.08362619215e-06
3	14573.7308899	1000A_sphere_sm.xml	-53.103104744	63.4838036783	16202.7905345	0.876828609157	1.0615771963e-06
4	14.8778250409	cyl_400_20.txt	1.09519860225	40.6322815793	0.369365066051	1.45820592849	2.20320985197e-07
5	1099.12028341	cyl_400_40.txt	5.82667298758	89.7705047132	0.0908448948032	1.57707251135	1.08545405512e-07
6	7.54731143325e+12	cyl_testdata1.txt	0.0	60.0	0	1.0	1e-06
7	8.15055429508e+12	cyl_testdata2.txt	0.0	60.0	0	1.0	1e-06
8							
9							
10							
11							

X-axis Label Selection Range
X-axis Label scale
X-axis Unit
Y-axis Label Selection Range
Y-axis Label radius
Y-axis Unit
dY-Bar (Optional) Selection Range

Highlight the data set or the Chi2 column first.

NB: In theory, returning to the BatchPage and changing the name of the I(Q) data source should also work, but at the moment whilst this does change the data set displayed it always superimposes the 'theory' corresponding to the starting parameters.

If you select a 'Chi2' value and click the *View Fits* button a graph of the residuals for that data set is displayed. Again, if you select multiple 'Chi2' values then all the residuals data will appear on one graph. Also see *Assessing Fit Quality*.

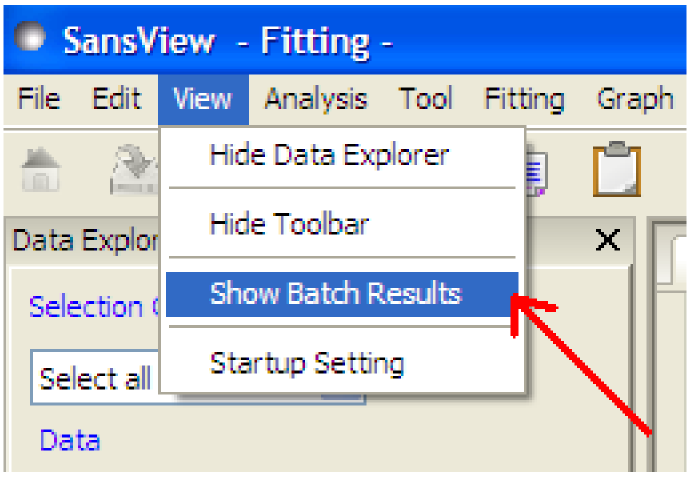
Chain Fitting

By default, the *same* parameter values copied from the initial single fit into the *BatchPage* will be used as the starting parameters for all batch fits. It is, however, possible to get *SasView* to use the results of a fit to a preceding data set as the starting parameters for the next fit in the sequence. This variation of batch fitting is called *Chain Fitting*, and will considerably speed up model-fitting if you have lots of very similar data sets where a few parameters are gradually changing. Do not use chain fitting on disparate data sets.

To use chain fitting, select *Chain Fitting* under *Fitting* in the menu bar. It toggles on/off, so selecting it again will switch back to normal batch fitting.

Grid Window

The *Grid Window* provides an easy way to view the results from batch fitting. It will be displayed automatically when a batch fit completes, but may be opened at any time by selecting *Show Grid Window* under *View* in the menu bar.

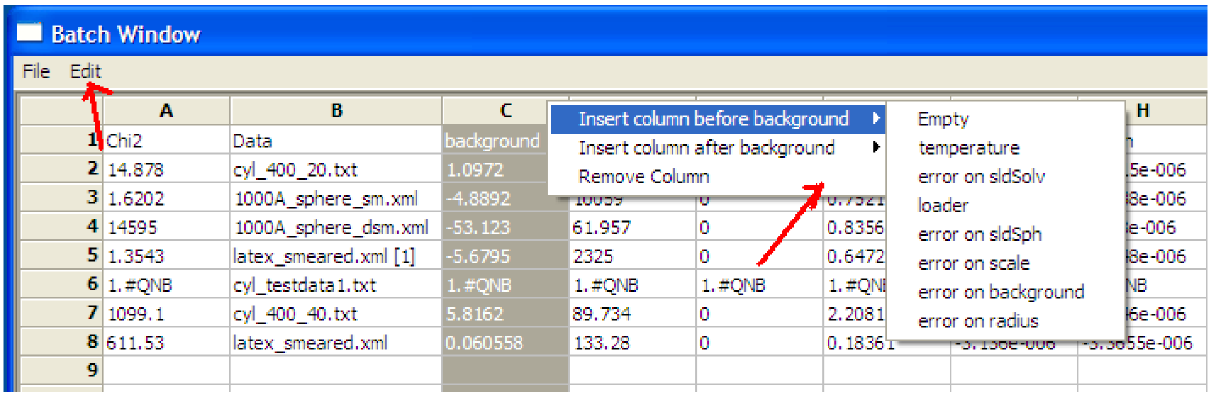


Once a batch fit is completed, all model parameters are displayed but *not* their uncertainties. To view the uncertainties, click on a given column then go to *Edit* in the menu bar, select *Insert Column Before* and choose the required data from the list. An empty column can be inserted in the same way.

To remove a column from the grid, click on the column header and choose *Remove Column* under *Edit* in the menu bar. The same functionality also allows you to re-order columns.

NB: You cannot insert/remove/re-order the rows in the Grid Window.

All of the above functions are also available by right-clicking on a column label.

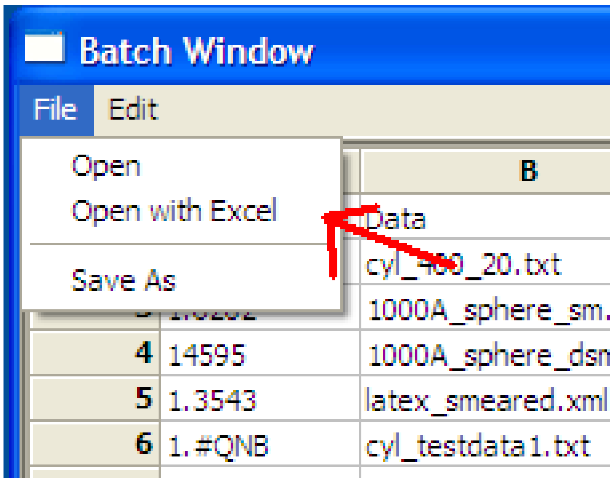


	A	B	C					H
1	Chi2	Data	background					
2	14.878	cyl_400_20.txt	1.0972					5e-006
3	1.6202	1000A_sphere_sm.xml	-4.8892					8e-006
4	14595	1000A_sphere_dsm.xml	-53.123	61.957	0	0.8356		e-006
5	1.3543	latex_smeared.xml [1]	-5.6795	2325	0	0.6472		8e-006
6	1.#QNB	cyl_testdata1.txt	1.#QNB	1.#QNB	1.#QNB	1.#QN		NB
7	1099.1	cyl_400_40.txt	5.8162	89.734	0	2.2081		6e-006
8	611.53	latex_smeared.xml	0.060558	133.28	0	0.18361		3655e-006
9								

NB: If there is an existing Grid Window and another batch fit is performed, an additional 'Table' tab will be added to the Grid Window.

The parameter values in the *currently selected* table of the *Grid Window* can be output to a CSV file by choosing *Save As* under *File* in the (*Grid Window*) menu bar. The default filename includes the date and time that the batch fit was performed.

Saved CSV files can be reloaded by choosing *Open* under *File* in the *Grid Window* menu bar. The loaded parameters will appear in a new table tab.

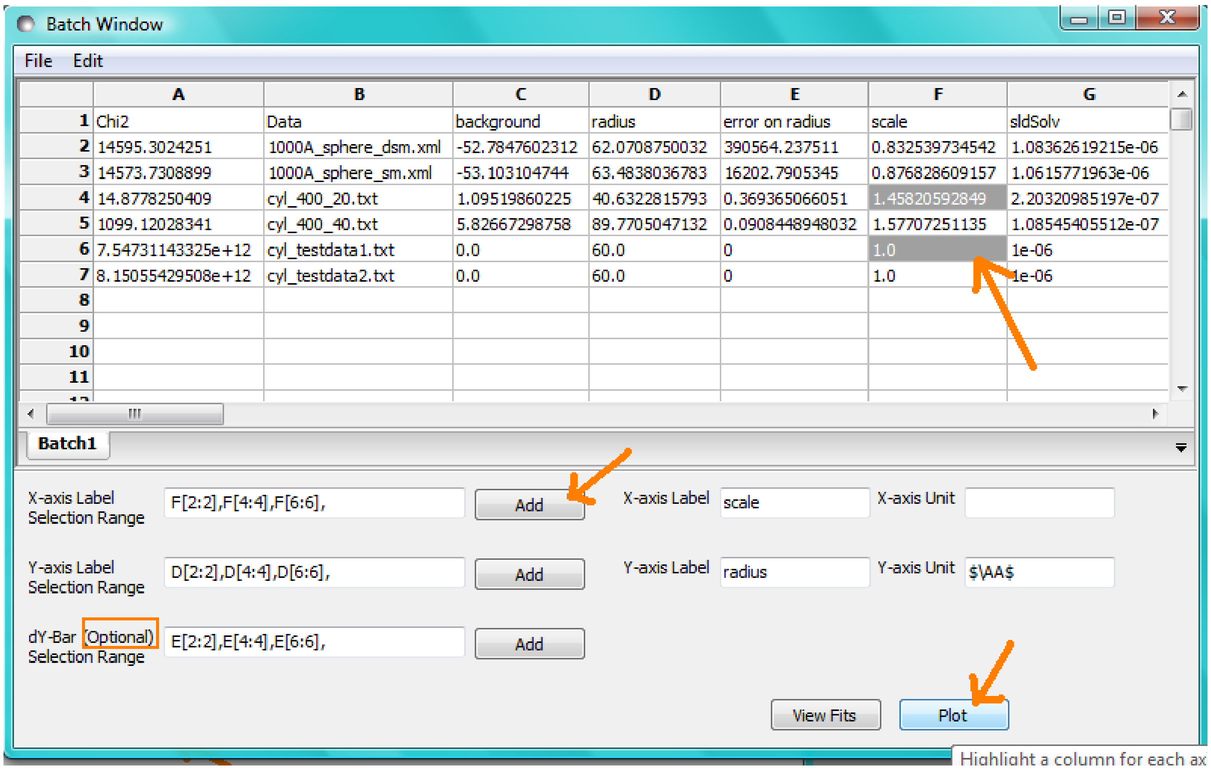


NB: Saving the Grid Window does not save any experimental data, residuals or actual model fits. Consequently if you reload a saved CSV file the ability to View Fits will be lost.

Parameter Plots

Any column of *numeric* parameter values can be plotted against another using the *Grid Window*. Simply select one column at the time and click the *Add* button next to the required *X/Y-axis Selection Range* text box. When both the X and Y axis boxes have been completed, click the *Plot* button.

When the *Add* button is clicked, *SasView* also automatically completes the *X/Y-axis Label* text box with the heading from Row 1 of the selected table, but different labels and units can be entered manually.



	A	B	C	D	E	F	G
1	Chi2	Data	background	radius	error on radius	scale	sldSolv
2	14595.3024251	1000A_sphere_dsm.xml	-52.7847602312	62.0708750032	390564.237511	0.832539734542	1.08362619215e-06
3	14573.7308899	1000A_sphere_sm.xml	-53.103104744	63.4838036783	16202.7905345	0.876828609157	1.0615771963e-06
4	14.8778250409	cyl_400_20.txt	1.09519860225	40.6322815793	0.369365066051	1.45820592849	2.20320985197e-07
5	1099.12028341	cyl_400_40.txt	5.82667298758	89.7705047132	0.0908448948032	1.57707251135	1.08545405512e-07
6	7.54731143325e+12	cyl_testdata1.txt	0.0	60.0	0	1.0	1e-06
7	8.15055429508e+12	cyl_testdata2.txt	0.0	60.0	0	1.0	1e-06
8							
9							
10							
11							

X-axis Label Selection Range F[2:2],F[4:4],F[6:6],
X-axis Label scale
X-axis Unit

Y-axis Label Selection Range D[2:2],D[4:4],D[6:6],
Y-axis Label radius
Y-axis Unit $\AA$

dY-Bar (Optional) Selection Range E[2:2],E[4:4],E[6:6],

The *X/Y-axis Selection Range* can be edited manually. The text control box recognises the operators +, -, *, /, or 'pow', and allows the following types of expression :

1. if an axis label range is a function of 1 or more *columns*, write this type of expression

   constant1 * column_name1 [minimum row index : maximum row index] operator constant2 * column_name2 [minimum row index : maximum row index]

Example: radius [2 : 5] -3 * scale [2 : 5]

2. if only some *values* of a given column are needed but the range between the first row and the last row used is not continuous, write this type of expression

   column_name1 [minimum row index1 : maximum row index1] , column_name1 [minimum row index2 : maximum row index2]

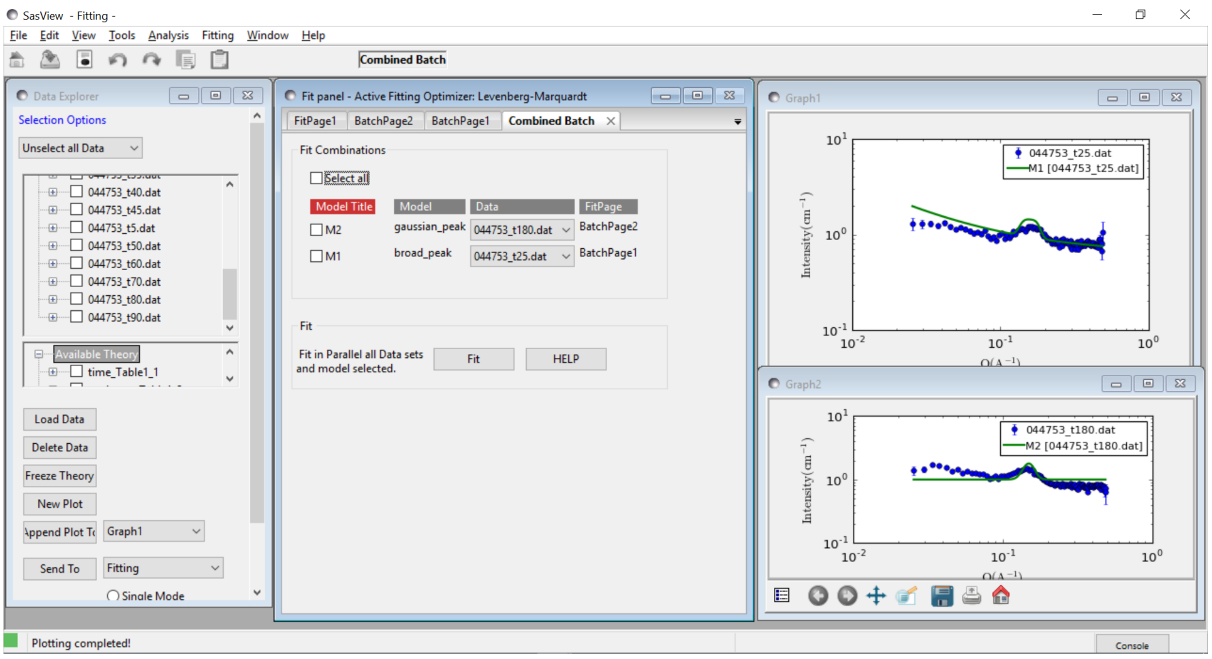
   Example: radius [2 : 5] , radius [10 : 25]

Combined Batch Fit Mode

The purpose of the Combined Batch Fit is to allow running two or more batch fits in sequence without overwriting the output table of results. This may be of interest for example if one is fitting a series of data sets where there is a shape change occurring in the series that requires changing the model part way through the series; for example a sphere to rod transition. Indeed the regular batch mode does not allow for multiple models and requires all the files in the series to be fit with single model and set of parameters. While it is of course possible to just run part of the series as a batch fit using model one followed by running another batch fit on the rest of the series with model two (and/or model three etc), doing so will overwrite the table of outputs from the previous batch fit(s). This may not be desirable if one is interested in comparing the parameters: for example the sphere radius of set one and the cylinder radius of set two.

Method

In order to use the *Combined Batch Fit*, first load all the data needed as described in *Loading Data*. Next start up two or more *BatchPage* fits following the instructions in *Batch Fit Mode* but **DO NOT PRESS FIT**. At this point the *Combine Batch Fit* menu item under the *Fitting menu* should be active (if there is one or no *BatchPage* the menu item will be greyed out and inactive). Clicking on *Combine Batch Fit* will bring up a new panel, similar to the *Const & Simult Fit* panel. In this case there will be a checkbox for each *BatchPage* instead of each *FitPage* that should be included in the fit. Once all are selected, click the Fit button on the *BatchPage* to run each batch fit in *sequence*

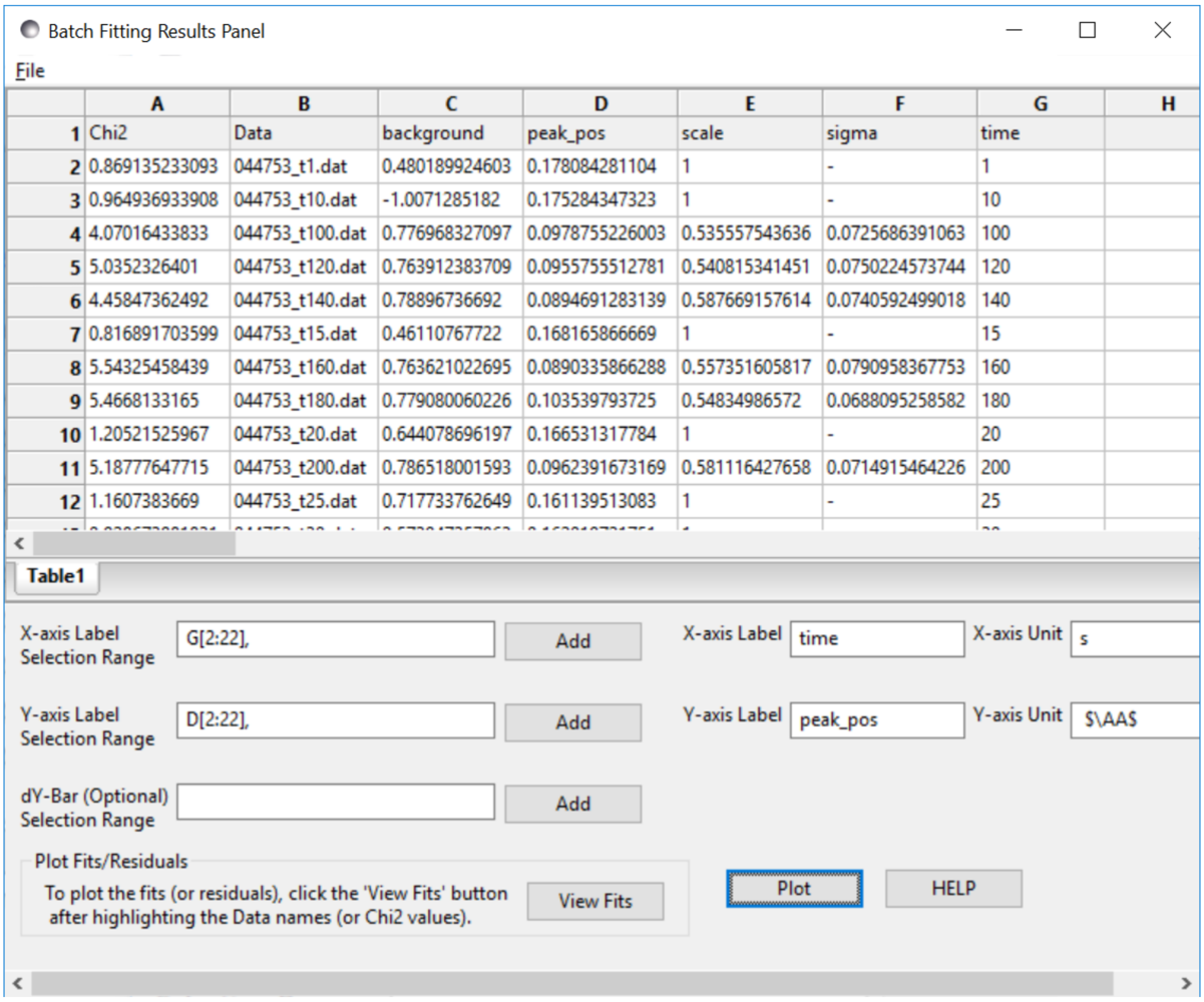


Model Title	Model	Data	FitPage
☐ M2	gaussian_peak	044753_t180.dat	BatchPage2
☐ M1	broad_peak	044753_t25.dat	BatchPage1

The batch table will then pop up at the end as for the case of the simple Batch Fitting with the following caveats:

Note: The order matters. The parameters in the table will be taken from the model used in the first *BatchPage* of the list. Any parameters from the second and later *BatchPage* s that have the same name as a parameter in the first will show up allowing for plotting of that parameter across the models. The other parameters will not be available in the grid.

Note: a corralary of the above is that currently models created as a sum|multiply model will not work as desired because the generated model parameters have a p#_ appended to the beginning and thus radius and p1_radius will not be recognized as the same parameter.


Batch Fitting Results Panel

File

	A	B	C	D	E	F	G	H
1	Chi2	Data	background	peak_pos	scale	sigma	time	
2	0.869135233093	044753_t1.dat	0.480189924603	0.178084281104	1	-	1	
3	0.964936933908	044753_t10.dat	-1.0071285182	0.175284347323	1	-	10	
4	4.07016433833	044753_t100.dat	0.776968327097	0.0978755226003	0.535557543636	0.0725686391063	100	
5	5.0352326401	044753_t120.dat	0.763912383709	0.0955755512781	0.540815341451	0.0750224573744	120	
6	4.45847362492	044753_t140.dat	0.78896736692	0.0894691283139	0.587669157614	0.0740592499018	140	
7	0.816891703599	044753_t15.dat	0.46110767722	0.168165866669	1	-	15	
8	5.54325458439	044753_t160.dat	0.763621022695	0.0890335866288	0.557351605817	0.0790958367753	160	
9	5.4668133165	044753_t180.dat	0.779080060226	0.103539793725	0.54834986572	0.0688095258582	180	
10	1.20521525967	044753_t20.dat	0.644078696197	0.166531317784	1	-	20	
11	5.18777647715	044753_t200.dat	0.786518001593	0.0962391673169	0.581116427658	0.0714915464226	200	
12	1.1607383669	044753_t25.dat	0.717733762649	0.161139513083	1	-	25	

Table1

X-axis Label Selection Range G[2:22],
X-axis Label time
X-axis Unit s

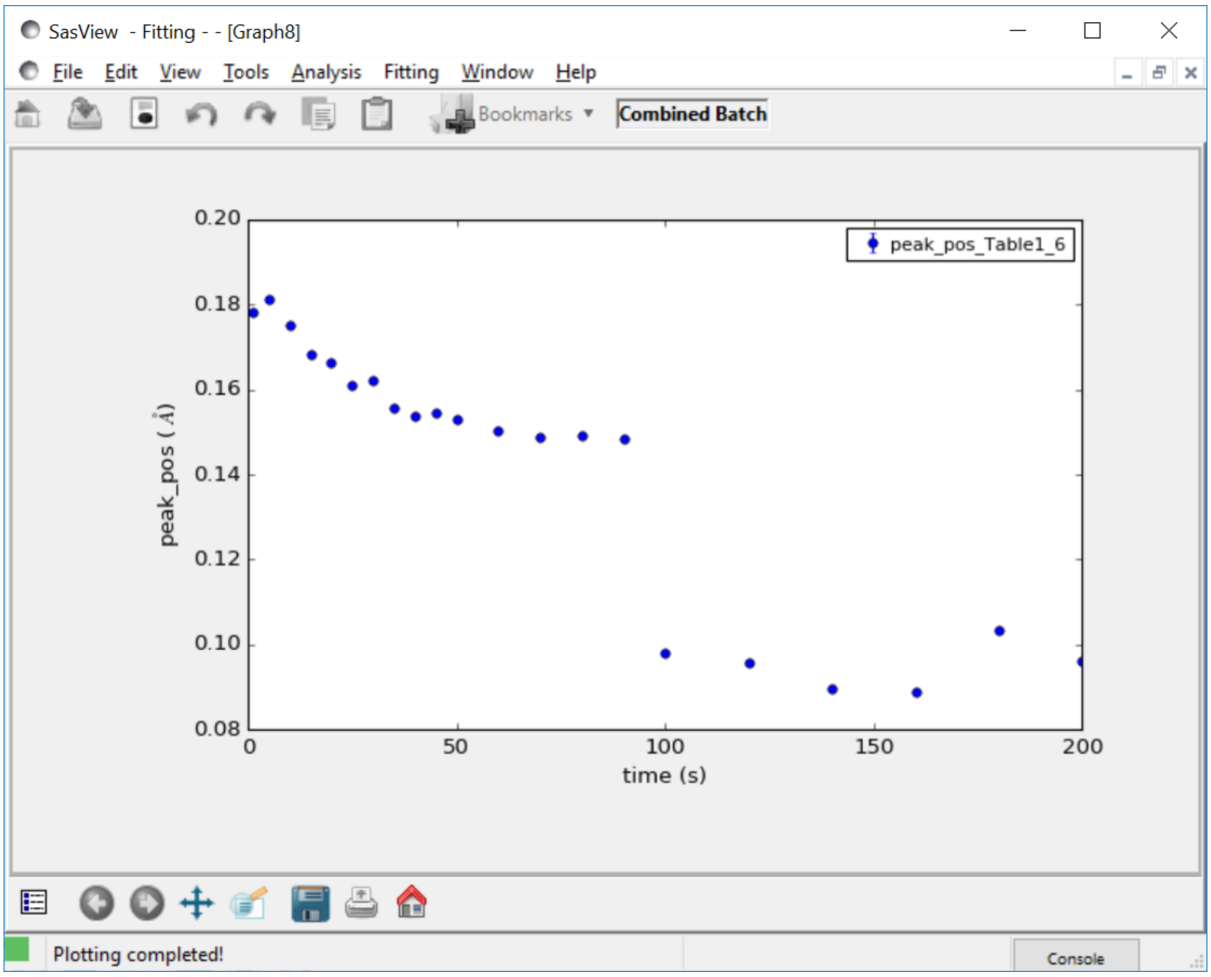
Y-axis Label Selection Range D[2:22],
Y-axis Label peak_pos
Y-axis Unit $\AA$

dY-Bar (Optional) Selection Range

Plot Fits/Residuals

To plot the fits (or residuals), click the 'View Fits' button after highlighting the Data names (or Chi2 values).

In the example shown above the data is a time series with a shifting peak. The first part of the series was fitted using the *broad_peak* model, while the rest of the data were fit using the *gaussian_peak* model. Unfortunately the time is not listed in the file but the file name contains the information. As described in *Grid Window*, a column can be added manually, in this case called time, and the peak position plotted against time.



Note the discontinuity in the peak position. This reflects the fact that the Gaussian fit is a rather poor model for the data and is not actually finding the peak.

.*Document History*

2017-09-10 Paul Butler
2017-09-15 Steve King
2018-03-05 Paul Butler

Assessing Fit Quality

When performing model-fits to some experimental data it is helpful to be able to gauge how good an individual fit is, how it compares to a fit of the *same model to another set of data*, or how it compares to a fit of a *different model to the same data*.

One way is obviously to just inspect the graph of the experimental data and to see how closely (or not!) the 'theory' calculation matches it. But *SasView* also provides two other measures of the quality of a fit:

- χ^2 (or 'Chi2'; pronounced 'chi-squared')
- *Residuals*

Chi2

χ^2 is a statistical parameter that quantifies the differences between an observed data set and an expected dataset (or 'theory') calculated as

$$\chi^2 = \sum[(Y_i - \mathrm{theory}_i)^2/\mathrm{error}_i^2]$$

Fitting typically minimizes the value of χ^2. For assessing the quality of the model and its "fit" however, *SasView* displays the traditional reduced χ^2_R which normalizes this parameter by dividing it by the number of degrees of freedom (or DOF). The DOF is the number of data points being considered, N_pts, reduced by the number of free (i.e. fitted) parameters, N_par. Note that model parameters that are kept fixed do *not* contribute to the DOF (they are not "free"). This reduced value is then given as

$$\chi^2_R = \sum[(Y_i - \mathrm{theory}_i)^2/\mathrm{error}_i^2]/[N_\mathrm{pts} - N_\mathrm{par}]$$

Note that this means the displayed value will vary depending on the number of parameters used in the fit. In particular, when doing a calculation without a fit (e.g. manually changing a parameter) the DOF will now equal N_pts and the χ^2_R will be the smallest possible for that combination of model, data set, and set of parameter values.

When $N_\mathrm{pts} \gg N_\mathrm{par}$ as it should for proper fitting, the value of the reduced χ^2_R will not change very much.

For a good fit, χ^2_R tends to 1.

χ^2_R is sometimes referred to as the 'goodness-of-fit' parameter.

Residuals

A residual is the difference between an observed value and an estimate of that value, such as a 'theory' calculation (whereas the difference between an observed value and its *true* value is its error).

SasView calculates 'normalized residuals', R_i, for each data point in the fit:

$$R_i = (Y_i - \mathrm{theory}_i)/\mathrm{error}_i$$

Think of each normalized residual as the number of standard deviations between the measured value and the theory. For a good fit, 68% of R_i will be within one standard deviation, which will show up in the Residuals plot as R_i values between -1 and $+1$. Almost all the values should be between -3 and $+3$.

Residuals values larger than ± 3 indicate that the model is not fit correctly, the wrong model was chosen (e.g., because there is more than one phase in your system), or there are problems in the data reduction. Since the goodness of fit is calculated from the sum-squared residuals, these extreme values will drive the choice of fit parameters. Any uncertainties calculated for the fitting parameters will be meaningless.

Document History

2015-06-08 Steve King
2017-09-28 Paul Kienzle
2018-03-04 Paul Butler

Polydispersity & Orientational Distributions

For some models we can calculate the average intensity for a population of particles that possess size and/or orientational (ie, angular) distributions. In SasView we call the former *polydispersity* but use the parameter *PD* to parameterise both. In other words, the meaning of *PD* in a model depends on the actual parameter it is being applied too.

The resultant intensity is then normalized by the average particle volume such that

$$P(q) = \mathrm{scale}\langle F^* F\rangle/V + \mathrm{background}$$

where F is the scattering amplitude and $\langle\cdot\rangle$ denotes an average over the distribution $f(x;\bar{x},\sigma)$, giving

$$P(q) = \frac{\text{scale}}{V} \int_{\mathbb{R}} f(x;\bar{x},\sigma) F^2(q,x)\, dx + \text{background}$$

Each distribution is characterized by a center value $\bar{x}$ or x_{med}, a width parameter σ (note this is *not necessarily* the standard deviation, so read the description of the distribution carefully), the number of sigmas N_σ to include from the tails of the distribution, and the number of points used to compute the average. The center of the distribution is set by the value of the model parameter.

The distribution width applied to *volume* (ie, shape-describing) parameters is relative to the center value such that $\sigma = \text{PD} \cdot \bar{x}$. However, the distribution width applied to *orientation* parameters is just $\sigma = \text{PD}$.

N_σ determines how far into the tails to evaluate the distribution, with larger values of N_σ required for heavier tailed distributions. The scattering in general falls rapidly with qr so the usual assumption that $f(r - 3\sigma_r)$ is tiny and therefore $f(r-3\sigma_r)f(r-3\sigma_r)$ will not contribute much to the average may not hold when particles are large. This, too, will require increasing N_σ.

Users should note that the averaging computation is very intensive. Applying polydispersion and/or orientational distributions to multiple parameters at the same time, or increasing the number of points in the distribution, will require patience! However, the calculations are generally more robust with more data points or more angles.

The following distribution functions are provided:

- *Uniform Distribution*
- *Rectangular Distribution*
- *Gaussian Distribution*
- *Boltzmann Distribution*
- *Lognormal Distribution*
- *Schulz Distribution*
- *Array Distribution*

These are all implemented as *number-average* distributions.

Additional distributions are under consideration.

Beware: when the Polydispersity & Orientational Distribution panel in SasView is first opened, the default distribution for all parameters is the Gaussian Distribution. This may not be suitable. See Suggested Applications below.

Note: In 2009 IUPAC decided to introduce the new term 'dispersity' to replace the term 'polydispersity' (see Pure Appl. Chem., (2009), 81(2), 351-353 in order to make the terminology describing distributions of chemical properties unambiguous. However, these terms are unrelated to the proportional size distributions and orientational distributions used in SasView models.

Suggested Applications

If applying polydispersion to parameters describing particle sizes, consider using the Lognormal or Schulz distributions.

If applying polydispersion to parameters describing interfacial thicknesses or angular orientations, consider using the Gaussian or Boltzmann distributions.

If applying polydispersion to parameters describing angles, use the Uniform distribution. Beware of using distributions that are always positive (eg, the Lognormal) because angles can be negative!

The array distribution allows a user-defined distribution to be applied.

Uniform Distribution

The Uniform Distribution is defined as

$$f(x) = \frac{1}{\text{Norm}} \begin{cases} 1 & \text{for } |x - \bar x| \le \sigma \\ 0 & \text{for } |x - \bar x| > \sigma \end{cases}$$

where $\bar x$ (x_mean in the figure) is the mean of the distribution, σ is the half-width, and *Norm* is a normalization factor which is determined during the numerical calculation.

The polydispersity in sasmodels is given by

$$\text{PD} = \sigma / \bar x$$

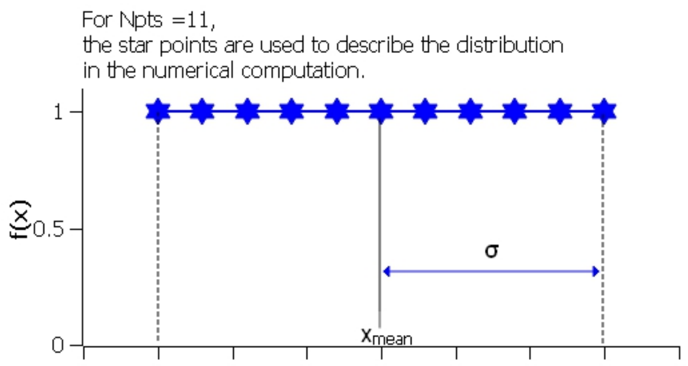


Fig. 1.127: Uniform distribution.

The value N_σ is ignored for this distribution.

Rectangular Distribution

The Rectangular Distribution is defined as

$$f(x) = \frac{1}{\text{Norm}} \begin{cases} 1 & \text{for } |x - \bar x| \le w \\ 0 & \text{for } |x - \bar x| > w \end{cases}$$

where $\bar x$ (x_mean in the figure) is the mean of the distribution, w is the half-width, and *Norm* is a normalization factor which is determined during the numerical calculation.

Note that the standard deviation and the half width w are different!

The standard deviation is

$$\sigma = w / \sqrt{3}$$

whilst the polydispersity in sasmodels is given by

$$\text{PD} = \sigma / \bar x$$

Note: The Rectangular Distribution is deprecated in favour of the Uniform Distribution above and is described here for backwards compatibility with earlier versions of SasView only.

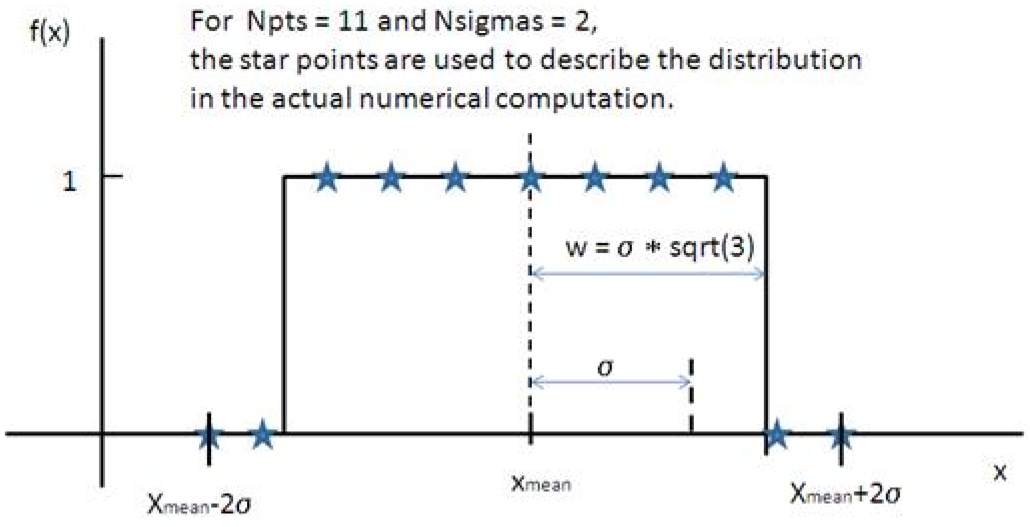

f(x)
For Npts = 11 and Nsigmas = 2,
the star points are used to describe the distribution
in the actual numerical computation.
1
w = σ * sqrt(3)
σ
Xmean-2σ
Xmean
Xmean+2σ
x

Fig. 1.128: Rectangular distribution.

Gaussian Distribution

The Gaussian Distribution is defined as

$$f(x) = \frac{1}{\text{Norm}} \exp\left(-\frac{(x-\bar{x})^2}{2\sigma^2}\right)$$

where $\bar{x}$ (x_{mean} in the figure) is the mean of the distribution and *Norm* is a normalization factor which is determined during the numerical calculation.

The polydispersity in sasmodels is given by

$$\text{PD} = \sigma/\bar{x}$$

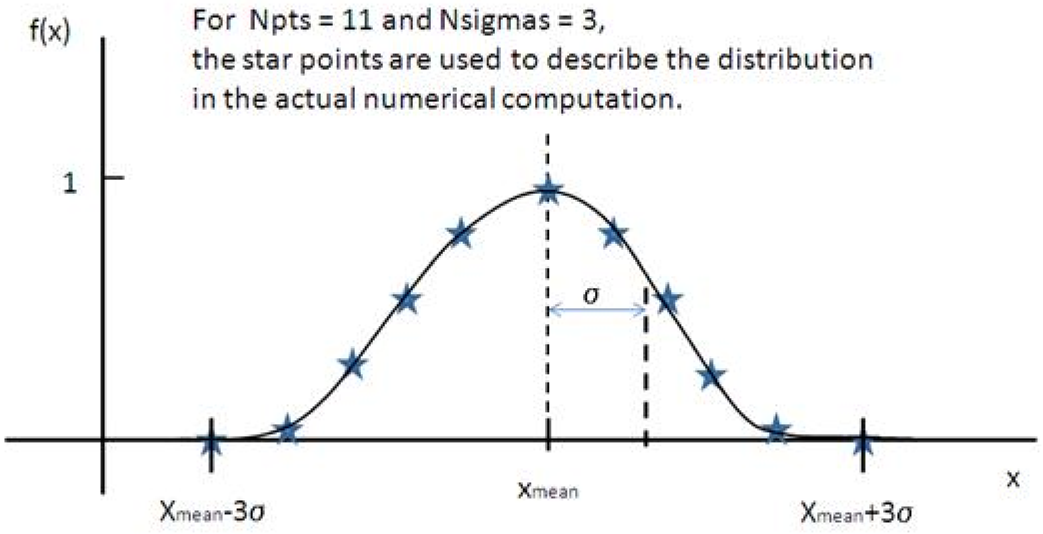

f(x)
For Npts = 11 and Nsigmas = 3,
the star points are used to describe the distribution
in the actual numerical computation.
1
σ
Xmean-3σ
Xmean
Xmean+3σ
x

Fig. 1.129: Normal distribution.

Boltzmann Distribution

The Boltzmann Distribution is defined as

$$f(x) = \frac{1}{\text{Norm}} \exp\left(-\frac{|x-\bar{x}|}{\sigma}\right)$$

where $\bar{x}$ (x_{mean} in the figure) is the mean of the distribution and *Norm* is a normalization factor which is determined during the numerical calculation.

The width is defined as

$$\sigma = \frac{kT}{E}$$

which is the inverse Boltzmann factor, where k is the Boltzmann constant, T the temperature in Kelvin and E a characteristic energy per particle.

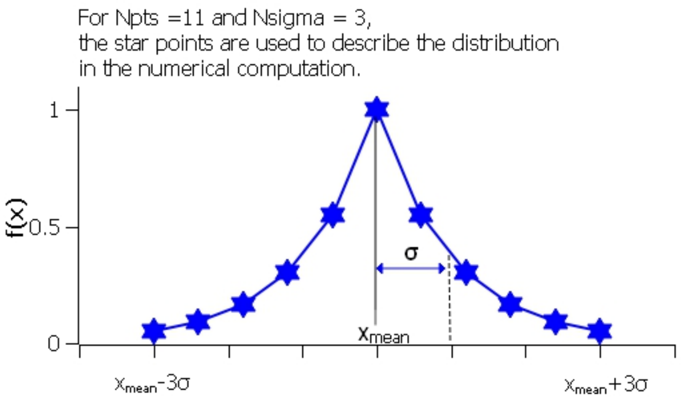


Fig. 1.130: Boltzmann distribution.

Lognormal Distribution

The Lognormal Distribution describes a function of x where $\ln(x)$ has a normal distribution. The result is a distribution that is skewed towards larger values of x.

The Lognormal Distribution is defined as

$$f(x) = \frac{1}{\text{Norm}} \frac{1}{x\sigma} \exp\left(-\frac{1}{2}\left(\frac{\ln(x)-\mu}{\sigma}\right)^2\right)$$

where *Norm* is a normalization factor which will be determined during the numerical calculation, $\mu = \ln(x_{\text{med}})$ and x_{med} is the *median* value of the *lognormal* distribution, but σ is a parameter describing the width of the underlying *normal* distribution.

x_{med} will be the value given for the respective size parameter in sasmodels, for example, *radius=60*.

The polydispersity in sasmodels is given by

$$\text{PD} = \sigma = p/x_{\text{med}}$$

The mean value of the distribution is given by $\bar{x} = \exp(\mu + \sigma^2/2)$ and the peak value by $\max x = \exp(\mu - \sigma^2)$.

The variance (the square of the standard deviation) of the *lognormal* distribution is given by

$$\nu = [\exp(\sigma^2) - 1] \exp(2\mu + \sigma^2)$$

Note that larger values of PD might need a larger number of points and N_σ.

For further information on the Lognormal distribution see: http://en.wikipedia.org/wiki/Log-normal_distribution and http://mathworld.wolfram.com/LogNormalDistribution.html

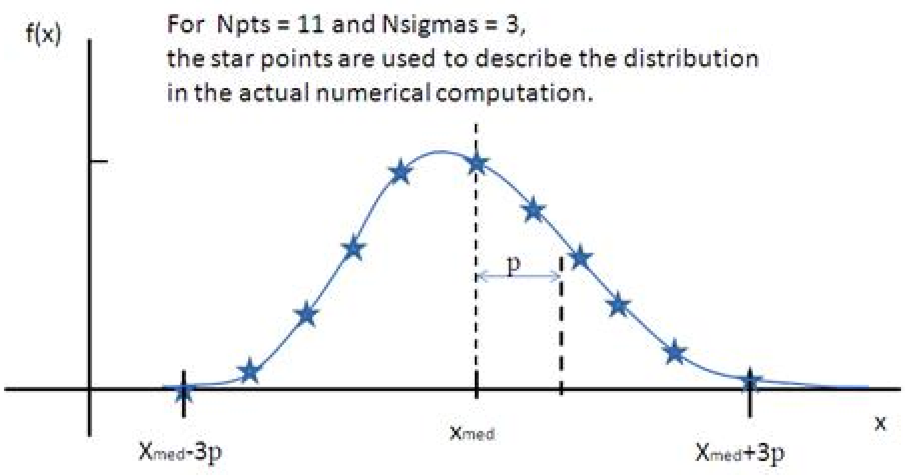

f(x)
For Npts = 11 and Nsigmas = 3,
the star points are used to describe the distribution
in the actual numerical computation.
p
Xmed-3p
Xmed
Xmed+3p
x

Fig. 1.131: Lognormal distribution for PD=0.1.

Schulz Distribution

The Schulz (sometimes written Schultz) distribution is similar to the Lognormal distribution, in that it is also skewed towards larger values of x, but which has computational advantages over the Lognormal distribution.

The Schulz distribution is defined as

$$f(x) = \frac{1}{\text{Norm}}(z+1)^{z+1}(x/\bar{x})^z \frac{\exp[-(z+1)x/\bar{x}]}{\bar{x}\Gamma(z+1)}$$

where $\bar{x}$ (x_{mean} in the figure) is the mean of the distribution, *Norm* is a normalization factor which is determined during the numerical calculation, and z is a measure of the width of the distribution such that

$$z = (1-p^2)/p^2$$

where p is the polydispersity in sasmodels given by

$$PD = p = \sigma/\bar{x}$$

and σ is the RMS deviation from $\bar{x}$.

Note that larger values of PD might need a larger number of points and N_σ. For example, for PD=0.7 with radius=60 Å, at least Npts>=160 and Nsigmas>=15 are required.

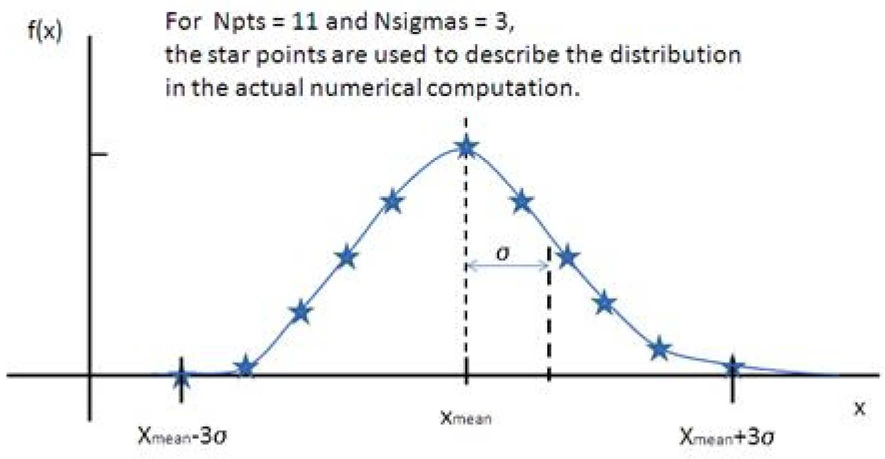

f(x)
For Npts = 11 and Nsigmas = 3,
the star points are used to describe the distribution
in the actual numerical computation.
σ
Xmean-3σ
Xmean
Xmean+3σ
x

Fig. 1.132: Schulz distribution.

For further information on the Schulz distribution see: M Kotlarchyk & S-H Chen, *J Chem Phys*, (1983), 79, 2461 and M Kotlarchyk, RB Stephens, and JS Huang, *J Phys Chem*, (1988), 92, 1533

Array Distribution

This user-definable distribution should be given as a simple ASCII text file where the array is defined by two columns of numbers: x and $f(x)$. The $f(x)$ will be normalized to 1 during the computation.

Example of what an array distribution file should look like:

30	0.1
32	0.3
35	0.4
36	0.5
37	0.6
39	0.7
41	0.9

Only these array values are used computation, therefore the parameter value given for the model will have no affect, and will be ignored when computing the average. This means that any parameter with an array distribution will not be fitable.

Note about DLS polydispersity

Several measures of polydispersity abound in Dynamic Light Scattering (DLS) and it should not be assumed that any of the following can be simply equated with the polydispersity *PD* parameter used in SasView.

The dimensionless **Polydispersity Index (PI)** is a measure of the width of the distribution of autocorrelation function decay rates (*not* the distribution of particle sizes itself, though the two are inversely related) and is defined by ISO 22412:2017 as

$$PI = \mu_2/\bar{\Gamma}^2$$

where μ_2 is the second cumulant, and $\bar{\Gamma}^2$ is the intensity-weighted average value, of the distribution of decay rates.

If the distribution of decay rates is Gaussian then

$$PI = \sigma^2/2\bar{\Gamma}^2$$

where σ is the standard deviation, allowing a **Relative Polydispersity (RP)** to be defined as

$$RP = \sigma/\bar{\Gamma} = \sqrt{2 \cdot PI}$$

PI values smaller than 0.05 indicate a highly monodisperse system. Values greater than 0.7 indicate significant polydispersity.

The **size polydispersity P-parameter** is defined as the relative standard deviation coefficient of variation

$$P = \sqrt{\nu}/\bar{R}$$

where ν is the variance of the distribution and $\bar{R}$ is the mean value of R. Here, the product $P\bar{R}$ is *equal* to the standard deviation of the Lognormal distribution.

P values smaller than 0.13 indicate a monodisperse system.

For more information see:

ISO 22412:2017, International Standards Organisation (2017).

Polydispersity: What does it mean for DLS and Chromatography.

Dynamic Light Scattering: Common Terms Defined, Whitepaper WP111214. Malvern Instruments (2011).

S King, C Washington & R Heenan, *Phys Chem Chem Phys*, (2005), 7, 143.

T Allen, in *Particle Size Measurement*, 4th Edition, Chapman & Hall, London (1990).

Document History

2015-05-01 Steve King
2017-05-08 Paul Kienzle
2018-03-20 Steve King
2018-04-04 Steve King
2018-08-09 Steve King

Resolution Functions

Sometimes the instrumental geometry used to acquire the experimental data has an impact on the clarity of features in the reduced scattering curve. For example, peaks or fringes might be slightly broadened. This is known as *Q resolution smearing*. To compensate for this effect one can either try and remove the resolution contribution - a process called *desmearing* - or add the resolution contribution into a model calculation/simulation (which by definition will be exact) to make it more representative of what has been measured experimentally - a process called *smearing*. Sasmodels does the latter.

Both smearing and desmearing rely on functions to describe the resolution effect. Sasmodels provides three smearing algorithms:

- *Slit Smearing*
- *Pinhole Smearing*
- *2D Smearing*

The Q resolution values should be determined by the data reduction software for the instrument and stored with the data file. If not, they will need to be set manually before fitting.

Slit Smearing

This type of smearing is normally only encountered with data from X-ray Kratky cameras or X-ray/neutron Bonse-Hart USAXS/USANS instruments.

The slit-smeared scattering intensity is defined by

$$I_s = \frac{1}{\text{Norm}} \int_{-\infty}^{\infty} dv\, W_v(v) \int_{-\infty}^{\infty} du\, W_u(u)\, I\left(\sqrt{(q+v)^2 + |u|^2}\right)$$

where *Norm* is given by

$$\int_{-\infty}^{\infty} dv\, W_v(v) \int_{-\infty}^{\infty} du\, W_u(u)$$

[Equation 1]

The functions $W_v(v)$ and $W_u(u)$ refer to the slit width weighting function and the slit height weighting determined at the given q point, respectively. It is assumed that the weighting function is described by a rectangular function, such that

$$W_v(v) = \delta(|v| \le \Delta q_v)$$

[Equation 2]

and

$$W_u(u) = \delta(|u| \le \Delta q_u)$$

[Equation 3]

so that $\Delta q_\alpha = \int_0^\infty d\alpha\, W_\alpha(\alpha)$ for α as v and u.

Here Δq_u and Δq_v stand for the the slit height (FWHM/2) and the slit width (FWHM/2) in q space.

This simplifies the integral in Equation 1 to

$$I_s(q) = \frac{2}{\text{Norm}} \int_{-\Delta q_v}^{\Delta q_v} dv \int_{0}^{\Delta q_u} du\, I\left(\sqrt{(q+v)^2 + u^2}\right)$$

[Equation 4]

which may be solved numerically, depending on the nature of Δq_u and Δq_v.

Solution 1

For $\Delta q_v = 0$ **and** $\Delta q_u =$ constant

$$I_s(q) \approx \int_{0}^{\Delta q_u} du\, I\left(\sqrt{q^2 + u^2}\right) = \int_{0}^{\Delta q_u} d\left(\sqrt{q'^2 - q^2}\right) I(q')$$

For discrete q values, at the q values of the data points and at the q values extended up to $q_N = q_i + \Delta q_u$ the smeared intensity can be approximately calculated as

$$I_s(q_i) \approx \sum_{j=i}^{N-1} \left[\sqrt{q_{j+1}^2 - q_i^2} - \sqrt{q_j^2 - q_i^2}\right] I(q_j) \sum_{j=1}^{N-1} W_{ij}\, I(q_j)$$

[Equation 5]

where $W_{ij} = 0$ for I_s when $j < i$ or $j > N - 1$.

Solution 2

For $\Delta q_v =$ constant **and** $\Delta q_u = 0$

Similar to Case 1

$$I_s(q_i) \approx \sum_{j=p}^{N-1} [q_{j+1} - q_i]\, I(q_j) \approx \sum_{j=p}^{N-1} W_{ij}\, I(q_j)$$

for $q_p = q_i - \Delta q_v$ and $q_N = q_i + \Delta q_v$

[Equation 6]

where $W_{ij} = 0$ for I_s when $j < p$ or $j > N - 1$.

Solution 3

For $\Delta q_v =$ constant **and** $\Delta q_u =$ constant

In this case, the best way is to perform the integration of Equation 1 numerically for both slit height and slit width. However, the numerical integration is imperfect unless a large number of iterations, say, at least 10000 by 10000 for each element of the matrix W, is performed. This is usually too slow for routine use.

An alternative approach is used in sasmodels which assumes slit width << slit height. This method combines Solution 1 with the numerical integration for the slit width. Then

$$\begin{aligned} I_s(q_i) &\approx \sum_{j=p}^{N-1} \sum_{k=-L}^{L} \left[\sqrt{q_{j+1}^2 - (q_i + (k\Delta q_v/L))^2} - \sqrt{q_j^2 - (q_i + (k\Delta q_v/L))^2}\right] (\Delta q_v/L)\, I(q_j) \\ &\approx \sum_{j=p}^{N-1} W_{ij}\, I(q_j) \end{aligned}$$

[Equation 7]

for $q_p = q_i - \Delta q_v$ and $q_N = q_i + \Delta q_v$

where $W_{ij} = 0$ for I_s when $j < p$ or $j > N - 1$.

Pinhole Smearing

This is the type of smearing normally encountered with data from synchrotron SAXS cameras and SANS instruments.

The pinhole smearing computation is performed in a similar fashion to the slit-smeared case above except that the weight function used is a Gaussian. Thus Equation 6 becomes

$$\begin{aligned} I_s(q_i) &\approx \sum_{j=0}^{N-1} [\operatorname{erf}(q_{j+1}) - \operatorname{erf}(q_j)]\, I(q_j) \\ &\approx \sum_{j=0}^{N-1} W_{ij}\, I(q_j) \end{aligned}$$

[Equation 8]

2D Smearing

The 2D smearing computation is performed in a similar fashion to the 1D pinhole smearing above except that the weight function used is a 2D elliptical Gaussian. Thus

$$\begin{aligned} I_s(x_0,\, y_0) &= A \iint dx'dy' \, \exp\left[-\left(\frac{(x'-x'_0)^2}{2\sigma^2_{x'_0}} + \frac{(y'-y'_0)^2}{2\sigma_{y'_0}}\right)\right] I(x',\, y') \\ &= A\sigma_{x'_0}\sigma_{y'_0} \iint dXdY \, \exp\left[-\frac{(X^2+Y^2)}{2}\right] I(\sigma_{x'_0} X x'_0,\, \sigma_{y'_0} Y + y'_0) \\ &= A\sigma_{x'_0}\sigma_{y'_0} \iint dR d\Theta \, R \exp\left(-\frac{R^2}{2}\right) I(\sigma_{x'_0} R\cos\Theta + x'_0,\, \sigma_{y'_0} R\sin\Theta + y'_0) \end{aligned}$$

[Equation 9]

In Equation 9, $x_0 = q\cos(\theta)$, $y_0 = q\sin(\theta)$, and the primed axes are all in the coordinate rotated by an angle θ about the z-axis (see the figure below) so that $x'_0 = x_0\cos(\theta) + y_0\sin(\theta)$ and $y'_0 = -x_0\sin(\theta) + y_0\cos(\theta)$. Note that the rotation angle is zero for a x-y symmetric elliptical Gaussian distribution. The A is a normalization factor.

Now we consider a numerical integration where each of the bins in θ and R are *evenly* (this is to simplify the equation below) distributed by $\Delta\theta$ and ΔR respectively, and it is further assumed that $I(x', y')$ is constant within the bins. Then

$$\begin{aligned} I_s(x_0,\, y_0) &\approx A\sigma_{x'_0}\sigma_{y'_0} \sum_i^n \Delta\Theta \left[\exp\left(\frac{(R_i - \Delta R/2)^2}{2}\right) - \exp\left(\frac{(R_i + \Delta R/2)^2}{2}\right)\right] I(\sigma_{x'_0} R_i\cos\Theta_i + x'_0,\, \sigma_{y'_0} R_i\sin\Theta_i + y'_0) \\ &\approx \sum_i^n W_i\, I(\sigma_{x'_0} R_i\cos\Theta_i + x'_0,\, \sigma_{y'_0} R_i\sin\Theta_i + y'_0) \end{aligned}$$

[Equation 10]

Since the weighting factor on each of the bins is known, it is convenient to transform x'-y' back to x-y coordinates (by rotating it by $-\theta$ around the z-axis).

Then, for a polar symmetric smear

$$I_s(x_0,\, y_0) \approx \sum_i^n W_i\, I(x'\cos\theta - y'\sin\theta,\, x' sin\theta + y'\cos\theta)$$

[Equation 11]

where

$$\begin{aligned} x' &= \sigma_{x'_0} R_i \cos\Theta_i + x'_0 \\ y' &= \sigma_{y'_0} R_i \sin\Theta_i + y'_0 \\ x'_0 &= q = \sqrt{x_0^2 + y_0^2} \\ y'_0 &= 0 \end{aligned}$$

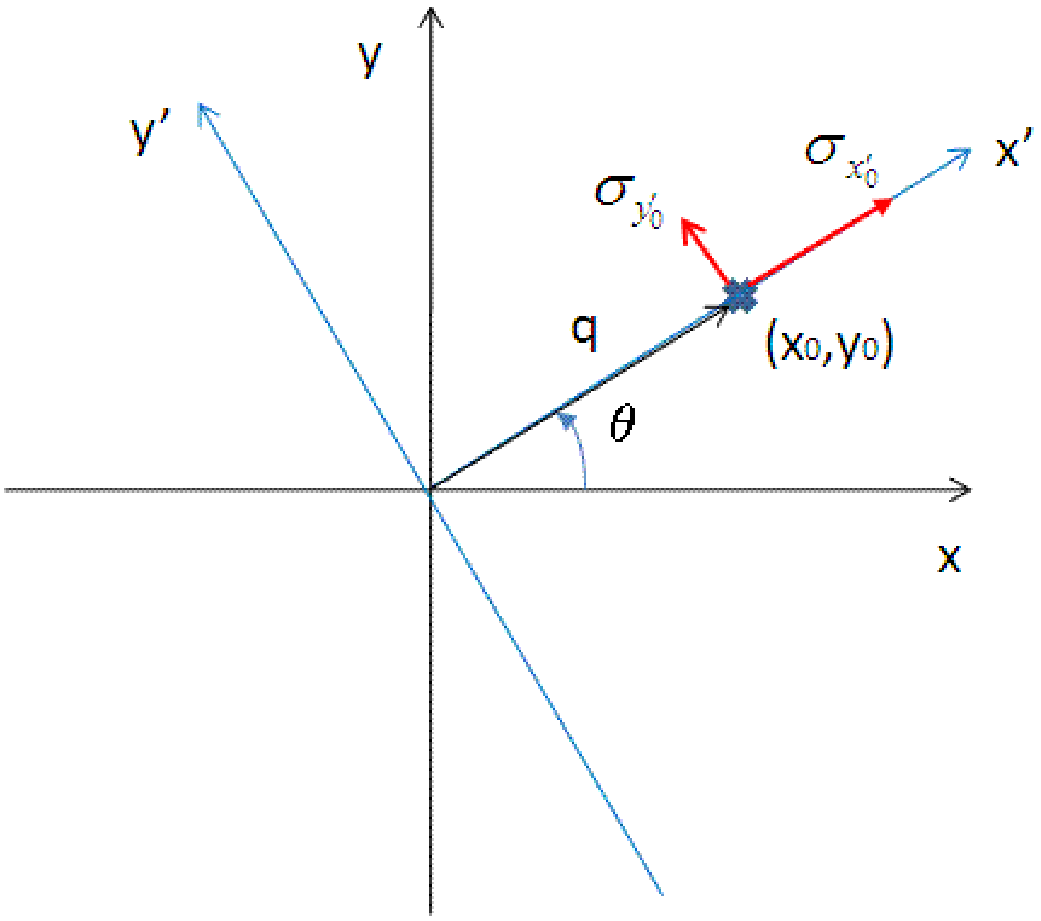


Fig. 1.133: Coordinate axis rotation for 2D resolution calculation.

while for a x-y symmetric smear

$$I_s(x_0,\, y_0) \approx \sum_i^n W_i \, I(x',\, y')$$

[Equation 12]

where

$$\begin{aligned} x' &= \sigma_{x'_0} R_i \cos\Theta_i + x'_0 \\ y' &= \sigma_{y'_0} R_i \sin\Theta_i + y'_0 \\ x'_0 &= x_0 = q_x \\ y'_0 &= y_0 = q_y \end{aligned}$$

The current version of sasmodels uses Equation 11 for 2D smearing, assuming that all the Gaussian weighting functions are aligned in the polar coordinate.

Weighting & Normalization

In all the cases above, the weighting matrix W is calculated on the first call to a smearing function, and includes ~60 q values (finely and evenly binned) below (>0) and above the q range of data in order to smear all data points for a given model and slit/pinhole size. The *Norm* factor is found numerically with the weighting matrix and applied on the computation of I_s.

Document History

2015-05-01 Steve King
2017-05-08 Paul Kienzle

Polarisation/Magnetic Scattering

Models which define a scattering length density parameter can be evaluated as magnetic models. In general, the scattering length density (SLD = β) in each region where the SLD is uniform, is a combination of the nuclear and magnetic SLDs and, for polarised neutrons, also depends on the spin states of the neutrons.

For magnetic scattering, only the magnetization component $\mathbf{M}_\perp$ perpendicular to the scattering vector $\mathbf{Q}$ contributes to the magnetic scattering length.

The magnetic scattering length density is then

$$\beta_M = \frac{\gamma r_0}{2\mu_B}\sigma \cdot \mathbf{M}_\perp = D_M \sigma \cdot \mathbf{M}_\perp$$

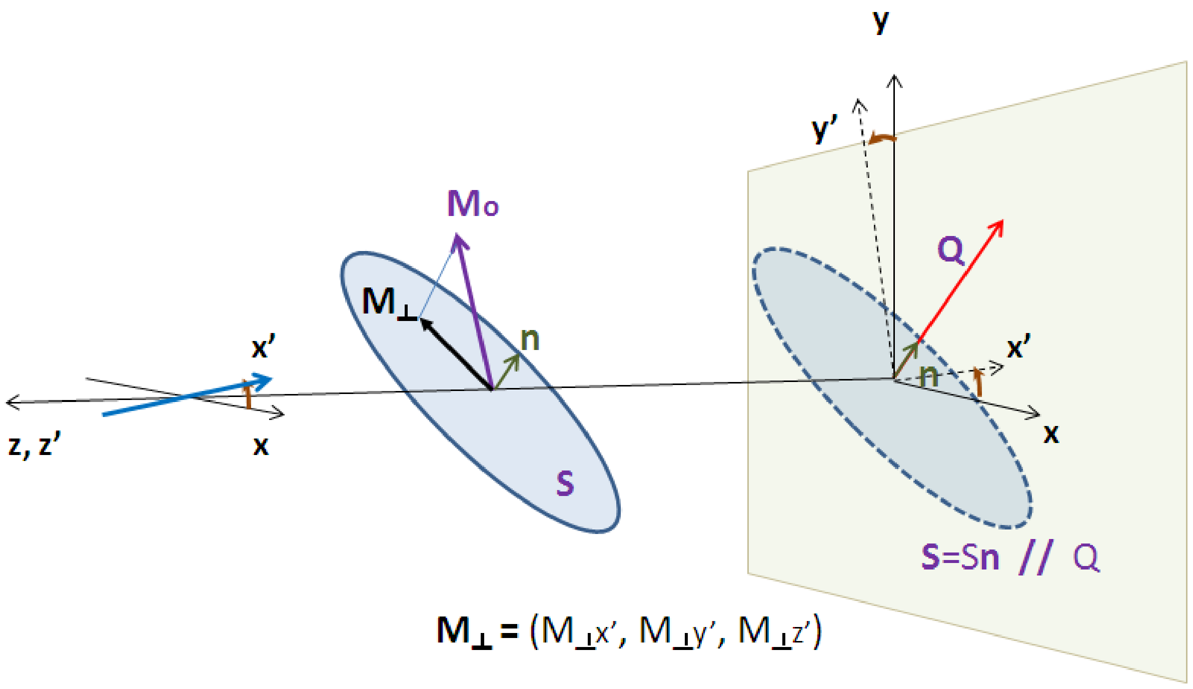
where $\gamma = -1.913$ is the gyromagnetic ratio, μ_B is the Bohr magneton, r_0 is the classical radius of electron, and σ is the Pauli spin.

Assuming that incident neutrons are polarized parallel $(+)$ and anti-parallel $(-)$ to the x' axis, the possible spin states after the sample are then:

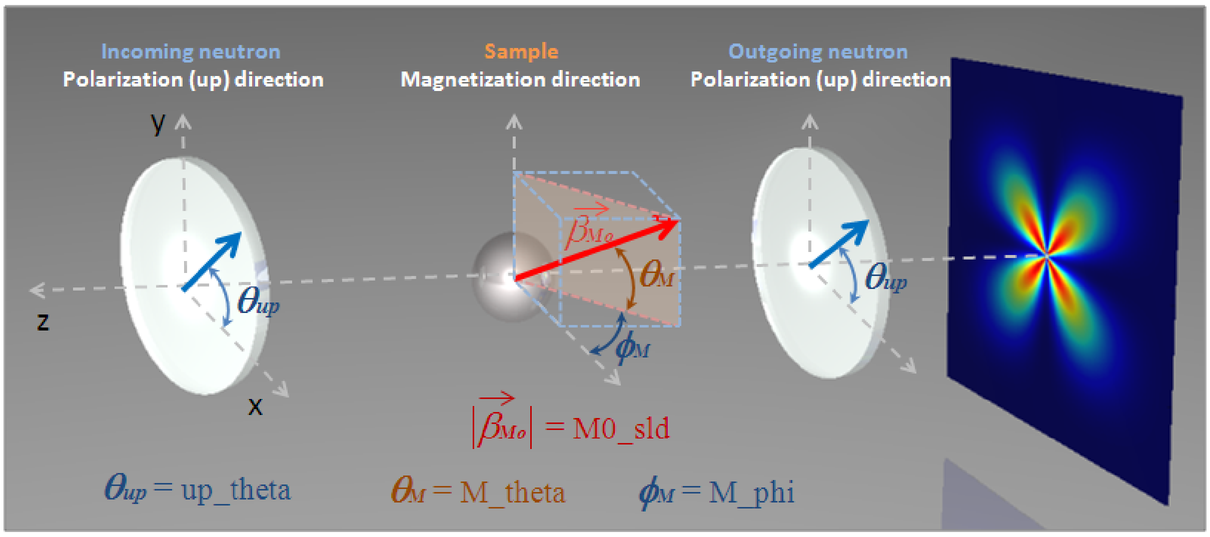
- Non spin-flip $(++)$ and $(--)$
- Spin-flip $(+-)$ and $(-+)$

Each measurement is an incoherent mixture of these spin states based on the fraction of $+$ neutrons before (u_i) and after (u_f) the sample, with weighting:

$$\begin{aligned} -- &= (1 - u_i)(1 - u_f) \\ -+ &= (1 - u_i)(u_f) \\ +- &= (u_i)(1 - u_f) \\ ++ &= (u_i)(u_f) \end{aligned}$$



Ideally the experiment would measure the pure spin states independently and perform a simultaneous analysis of the four states, tying all the model parameters together except u_i and u_f.



If the angles of the Q vector and the spin-axis x' to the x - axis are ϕ and θ_{up}, respectively, then, depending on the spin state of the neutrons, the scattering length densities, including the nuclear scattering length density (β_N) are

$$\beta_{\pm\pm} = \beta_N \mp D_M M_{\perp x'} \text{ for non spin-flip states}$$

and

$$\beta_{\pm\mp} = -D_M(M_{\perp y'} \pm iM_{\perp z'}) \text{ for spin-flip states}$$

where

$$\begin{aligned}
M_{\perp x'} &= M_{0q_x}\cos(\theta_{up}) + M_{0q_y}\sin(\theta_{up})\\
M_{\perp y'} &= M_{0q_y}\cos(\theta_{up}) - M_{0q_x}\sin(\theta_{up})\\
M_{\perp z'} &= M_{0z}\\
M_{0q_x} &= (M_{0x}\cos\phi - M_{0y}\sin\phi)\cos\phi\\
M_{0q_y} &= (M_{0y}\sin\phi - M_{0x}\cos\phi)\sin\phi
\end{aligned}$$

Here, M_{0x}, M_{0x}, M_{0z} are the x, y and z components of the magnetization vector given in the laboratory xyz frame given by

$$\begin{aligned} M_{0x} &= M_0 \cos\theta_M \cos\phi_M \\ M_{0y} &= M_0 \sin\theta_M \\ M_{0z} &= -M_0 \cos\theta_M \sin\phi_M \end{aligned}$$

and the magnetization angles θ_M and ϕ_M are defined in the figure above.

The user input parameters are:

sld_M0	$D_M M_0$
sld_mtheta	θ_M
sld_mphi	ϕ_M
up_frac_i	u_i = (spin up)/(spin up + spin down) *before* the sample
up_frac_f	u_f = (spin up)/(spin up + spin down) *after* the sample
up_angle	$\theta_{\rm up}$

Note: The values of the 'up_frac_i' and 'up_frac_f' must be in the range 0 to 1.

Document History

2015-05-02 Steve King
2017-11-15 Paul Kienzle
2018-06-02 Adam Washington

Oriented particles

With two dimensional small angle diffraction data sasmodels will calculate scattering from oriented particles, applicable for example to shear flow or orientation in a magnetic field.

In general we first need to define the reference orientation of the particle's a-b-c axes with respect to the incoming neutron or X-ray beam. This is done using three angles: θ and ϕ define the orientation of the c-axis of the particle, and angle Ψ is defined as the orientation of the major axis of the particle cross section with respect to its starting position along the beam direction (or equivalently, as rotation about the c axis). There is an unavoidable ambiguity when c is aligned with z in that ϕ and Ψ both serve to rotate the particle about c, but this symmetry is destroyed when θ is not a multiple of 180.

The figures below are for an elliptical cross section cylinder, but may be applied analogously to other shapes of particle.

Note: It is very important to note that these angles, in particular θ and ϕ, are NOT in general the same as the θ and ϕ appearing in equations for the scattering form factor which gives the scattered intensity or indeed in the equation for scattering vector Q. The θ rotation must be applied before the ϕ rotation, else there is an ambiguity.

Having established the mean direction of the particle (the view) we can then apply angular orientation distributions (jitter). This is done by a numerical integration over a range of angles in a similar way to particle size dispersity. The orientation dispersity is defined with respect to the a-b-c axes of the particle, with roll angle Ψ about the c-axis, yaw angle θ about the b-axis and pitch angle ϕ about the a-axis.

You can explore the view and jitter angles interactively using `sasmodels.jitter.run()`. Enter the following into the python interpreter:

```
from sasmodels import jitter
jitter.run()
```

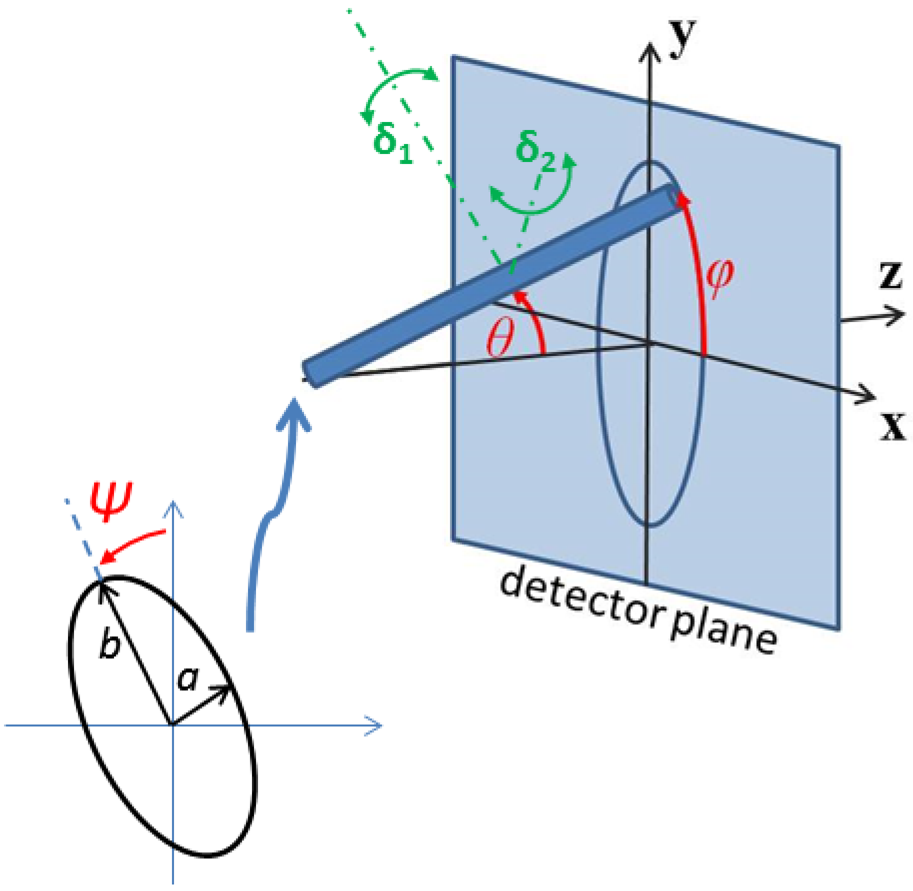


Fig. 1.134: Definition of angles for oriented elliptical cylinder, where axis_ratio b/a is shown >1. Note that rotation θ, initially in the x-z plane, is carried out first, then rotation ϕ about the z-axis, finally rotation Ψ is around the axis of the cylinder. The neutron or X-ray beam is along the $-z$ axis.

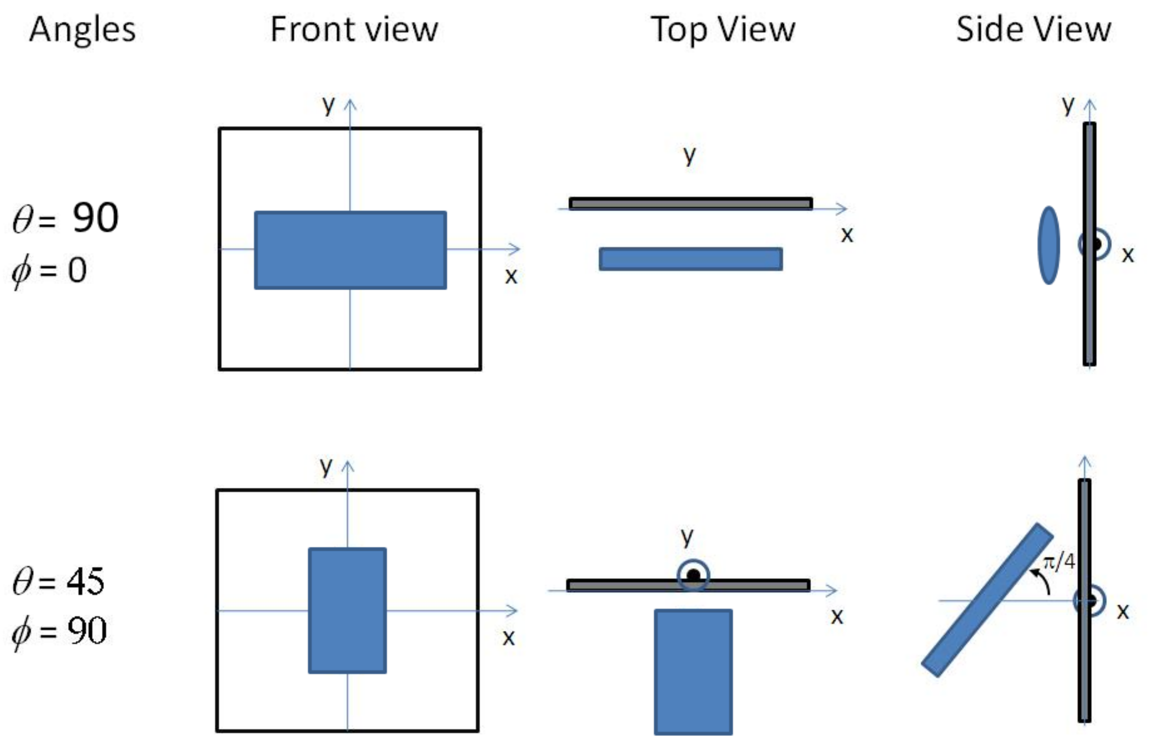


Fig. 1.135: Some examples of the orientation angles for an elliptical cylinder, with $\Psi = 0$.

More formally, starting with axes a-b-c of the particle aligned with axes x-y-z of the laboratory frame, the orientation dispersity is applied first, using the Tait-Bryan x-y'-z'' convention with angles $\Delta\phi$-$\Delta\theta$-$\Delta\Psi$. The reference orientation then follows, using the Euler angles z-y'-z'' with angles ϕ-θ-Ψ. This is implemented using rotation matrices as

$$R = R_z(\phi)\,R_y(\theta)\,R_z(\Psi)\,R_x(\Delta\phi)\,R_y(\Delta\theta)\,R_z(\Delta\Psi)$$

To transform detector (q_x, q_y) values into (q_a, q_b, q_c) for the shape in its canonical orientation, use

$$[q_a, q_b, q_c]^T = R^{-1}\,[q_x, q_y, 0]^T$$

The inverse rotation is easily calculated by rotating the opposite directions in the reverse order, so

$$R^{-1} = R_z(-\Delta\Psi)\,R_y(-\Delta\theta)\,R_x(-\Delta\phi)\,R_z(-\Psi)\,R_y(-\theta)\,R_z(-\phi)$$

The θ and ϕ orientation parameters for the cylinder only appear when fitting 2d data. On introducing "Orientational Distribution" in the angles, "distribution of theta" and "distribution of phi" parameters will appear. These are actually rotations about the axes δ_1 and δ_2 of the cylinder, which correspond to the b and a axes of the cylinder cross section. (When $\theta = \phi = 0$ these are parallel to the Y and X axes of the instrument.) The third orientation distribution, in Ψ, is about the c axis of the particle. Some experimentation may be required to understand the 2d patterns fully. A number of different shapes of distribution are available, as described for size dispersity, see *Polydispersity & Orientational Distributions*.

Given that the angular dispersion distribution is defined in cartesian space, over a cube defined by

$$[-\Delta\theta, \Delta\theta] \times [-\Delta\phi, \Delta\phi] \times [-\Delta\Psi, \Delta\Psi]$$

but the orientation is defined over a sphere, we are left with a map projection problem, with different tradeoffs depending on how values in $\Delta\theta$ and $\Delta\phi$ are translated into latitude/longitude on the sphere.

Sasmodels is using the equirectangular projection. In this projection, square patches in angular dispersity become wedge-shaped patches on the sphere. To correct for the changing point density, there is a scale factor of $\sin(\Delta\theta)$ that applies to each point in the integral. This is not enough, though. Consider a shape which is tumbling freely around the b axis, with $\Delta\theta$ uniform in $[-180, 180]$. At ± 90, all points in $\Delta\phi$ map to the pole, so the jitter will have a distinct angular preference. If the spin axis is along the beam (which will be the case for $\theta = 90$ and $\Psi = 90$) the scattering pattern should be circularly symmetric, but it will go to zero at $q_x = 0$ due to the $\sin(\Delta\theta)$ correction. This problem does not appear for a shape that is tumbling freely around the a axis, with $\Delta\phi$ uniform in $[-180, 180]$, so swap the a and b axes so $\Delta\theta < \Delta\phi$ and adjust Ψ by 90. This works with the current sasmodels shapes due to symmetry.

Alternative projections were considered. The sinusoidal projection works by scaling $\Delta\phi$ as $\Delta\theta$ increases, and dropping those points outside $[-180, 180]$. The distortions are a little less for middle ranges of $\Delta\theta$, but they are still severe for large $\Delta\theta$ and the model is much harder to explain. The azimuthal equidistance projection also improves on the equirectangular projection by extending the range of reasonable values for the $\Delta\theta$ range, with $\Delta\phi$ forming a wedge that cuts to the opposite side of the sphere rather than cutting to the pole. This projection has the nice property that distance from the center are preserved, and that $\Delta\theta$ and $\Delta\phi$ act the same. The azimuthal equal area projection is like the azimuthal equidistance projection, but it preserves area instead of distance. It also has the same behaviour for $\Delta\theta$ and $\Delta\phi$. The Guyou projection has an excellent balance with reasonable distortion in both $\Delta\theta$ and $\Delta\phi$, as well as preserving small patches. However, it requires considerably more computational overhead, and we have not yet derived the formula for the distortion correction, measuring the degree of stretch at the point $(\Delta\theta, \Delta\phi)$ on the map.

Note: Note that the form factors for oriented particles are performing numerical integrations over one or more variables, so care should be taken, especially with very large particles or more extreme aspect ratios. In such cases results may not be accurate, particularly at very high Q, unless the model has been specifically coded to use limiting forms of the scattering equations.

For best numerical results keep the θ distribution narrower than the ϕ distribution. Thus for asymmetric particles, such as elliptical_cylinder, you may need to reorder the sizes of the three axes to acheive the desired result. This is due to the issues of mapping a rectanglar distribution onto the surface of a sphere.

Users can experiment with the values of *Npts* and *Nsigs*, the number of steps used in the integration and the range spanned in number of standard deviations. The standard deviation is entered in units of degrees. For a "rectangular" distribution the full width should be $\pm\sqrt(3)$ ~ 1.73 standard deviations. The new "uniform" distribution avoids this by letting you directly specify the half width.

The angular distributions may be truncated outside of the range -180 to +180 degrees, so beware of using saying a broad Gaussian distribution with large value of *Nsigs*, as the array of *Npts* may be truncated to many fewer points than would give a good integration,as well as becoming rather meaningless. (At some point in the future the actual dispersion arrays may be made available to the user for inspection.)

Some more detailed technical notes are provided in the developer section of this manual *Orientation and Numerical Integration* .

This definition of orientation is new to SasView 4.2. In earlier versions, the orientation distribution appeared as a distribution of view angles. This led to strange effects when c was aligned with z, where changes to the ϕ angle served only to rotate the shape about c, rather than having a consistent interpretation as the pitch of the shape relative to the flow field defining the reference orientation. Prior to SasView 4.1, the reference orientation was defined using a Tait-Bryan convention, making it difficult to control. Now, rotation in θ modifies the spacings in the refraction pattern, and rotation in ϕ rotates it in the detector plane.

Document History

2017-11-06 Richard Heenan
2017-12-20 Paul Kienzle

Optimizer Selection

Bumps has a number of different optimizers available, each with its own control parameters:

- *Levenberg-Marquardt*
- *Nelder-Mead Simplex*
- *DREAM*
- *Differential Evolution*
- *Quasi-Newton BFGS*
- *Random Lines* [experimental]
- *Particle Swarm* [experimental]
- *Parallel Tempering* [experimental]

In general there is a trade-off between convergence rate and robustness, with the fastest algorithms most likely to find a local minimum rather than a global minimum. The gradient descent algorithms (*Levenberg-Marquardt*, *Quasi-Newton BFGS*) tend to be fast but they will find local minima only, while the population algorithms (*DREAM*, *Differential Evolution*) are more robust and likely slower. *Nelder-Mead Simplex* is somewhere between, with a small population keeping the search local but more robust than the gradient descent algorithms.

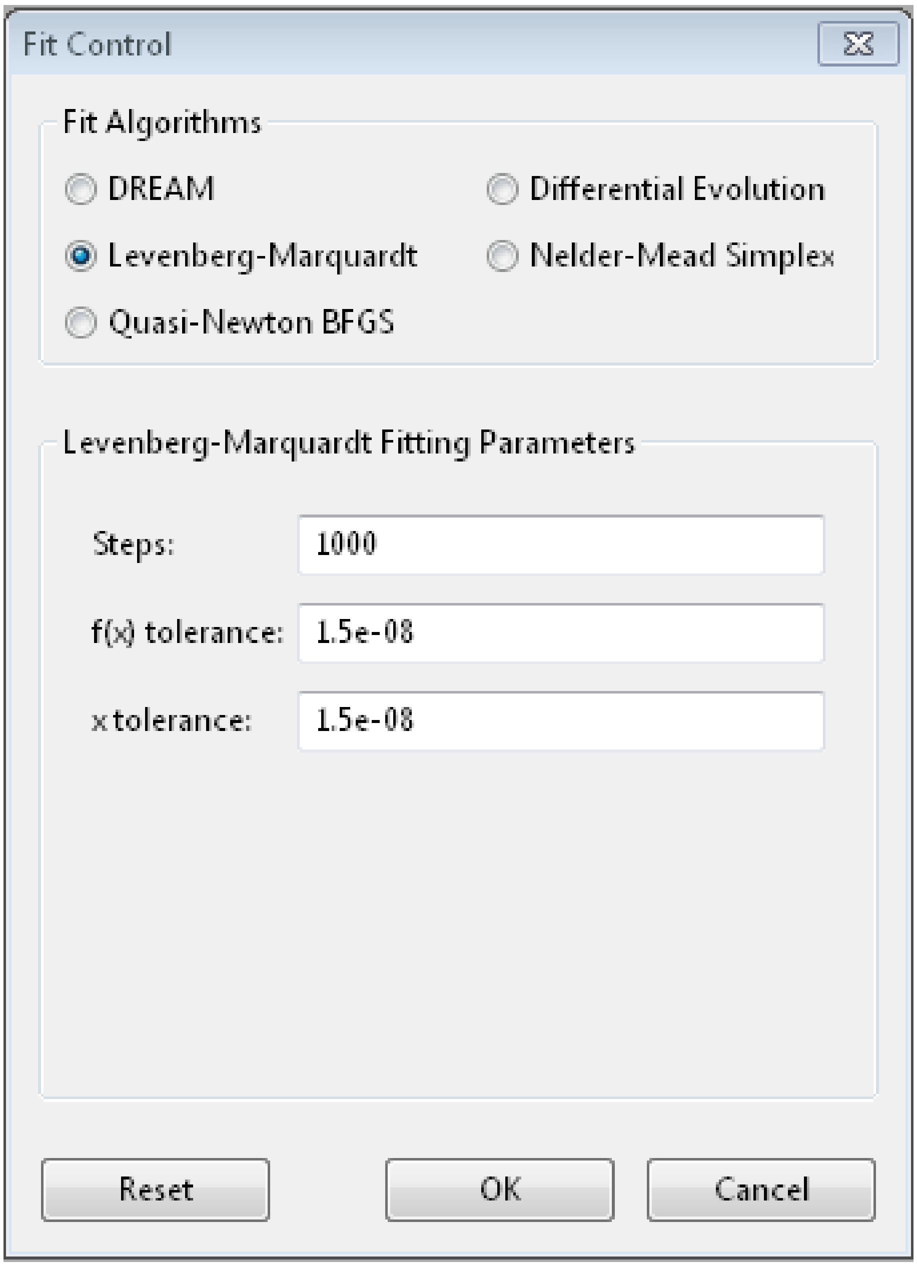
Each algorithm has its own set of control parameters for adjusting the search process and the stopping conditions. The same option may mean slightly different things to different optimizers. The bumps package provides a dialog box for selecting the optimizer and its options when running the fit wx application. This only includes the common options for the most useful optimizers. For full control, the fit will need to be run from the command line interface or through a python script.

For parameter uncertainty, most algorithms use the covariance matrix at the optimum to estimate an uncertainty ellipse. This is okay for a preliminary analysis, but only works reliably for weakly correlated parameters. For full uncertainty analysis, *DREAM* uses a random walk to explore the parameter space near the minimum, showing pairwise correlations amongst the parameter values. In order for *DREAM* to return the correct uncertainty, the function to be optimized should be a conditional probability density, with *nllf* as the negative log likelihood function of seeing point x in the parameter space. Other functions can be fitted, but uncertainty estimates will be meaningless.

Most algorithms have been adapted to run in parallel at least to some degree. The implementation is not heavily tuned, either in terms of minimizing the overhead per function evaluation or for distributing the problem across multiple processors. If the theory function is implemented in parallel, then the optimizer should be run in serial. Mixed mode is also possible when running on a cluster with a multi-threaded theory function. In this case, only one theory function will be evaluated on each cluster node, but the optimizer will distribute the parameters values to the cluster nodes in parallel. Do not run serial algorithms (*Levenberg-Marquardt*, *Quasi-Newton BFGS*) on a cluster.

We have included a number of optimizers in Bumps that did not perform particularly well on our problem sets. However, they may be perfect for your problem, so we have left them in the package for you to explore. They are not available in the GUI selection.

Levenberg-Marquardt



[✓ Levenberg-Marquardt]

Levenberg-Marquardt Fitting Parameters

Steps: 1000

f(x) tolerance: 1.5e-08

x tolerance: 1.5e-08

The Levenberg-Marquardt algorithm has been the standard method for non-linear data fitting. As a gradient descent trust region method, it starts at the initial value of the function and steps in the direction of the derivative until it reaches the minimum. Set up as an explicit minimization of the sum of square differences between theory and model, it uses a numerical approximation of the Jacobian matrix to set the step direction and an adaptive algorithm to set the size of the trust region.

When to use

Use this method when you have a reasonable fit near the minimum, and you want to get the best possible value. This can then be used as the starting point for uncertainty analysis using *DREAM*. This method requires that the problem definition includes a *residuals* method, but this should always be true when fitting data.

When modeling the results of an experiment, the best fit value is an accident of the measurement. Redo the same measurement, and the slightly different values you measure will lead to a different best fit. The important quantity to report is the credible interval covering 68% (1-σ) or 95% (2-σ) of the range of parameter values that are somewhat consistent with the data.

This method uses *lmfit* from *scipy*, and does not run in parallel.

Options

Steps is the number of gradient steps to take. Each step requires a calculation of the Jacobian matrix to determine the direction. This needs $2mn$ function evaluations, where n is the number of parameters and each function is evaluated and m data points (assuming center point formula for finite difference estimate of the derivative). The resulting linear equation is then solved, but for small n and expensive function evaluation this overhead can be ignored. Use `--steps=n` from the command line.

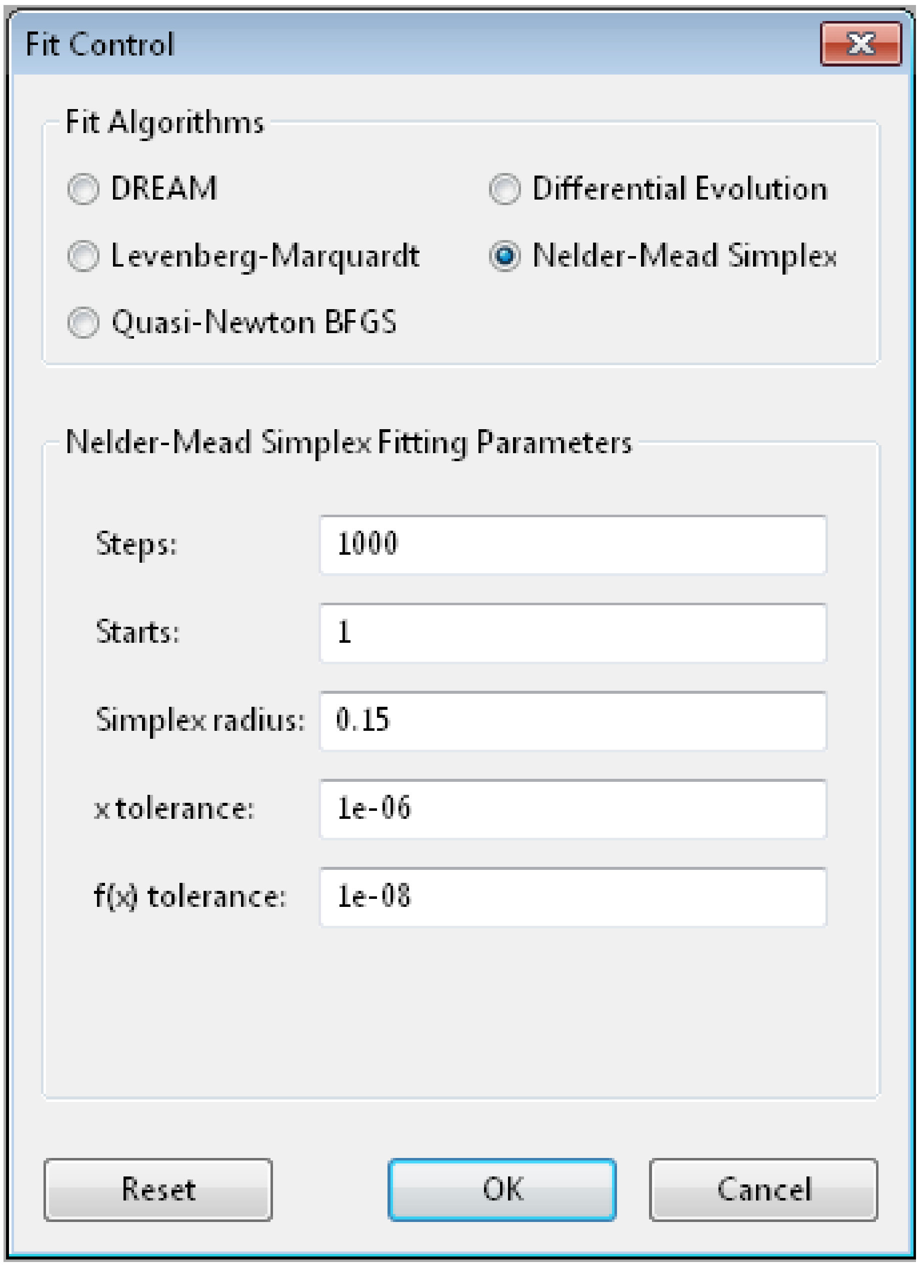
f(x) tolerance and *x tolerance* are used to determine when the fit has reached the point where no significant improvement is expected. If the function value does not improve significantly within the step, or the step is too short, then the fit will terminate. Use `--ftol=v` and `--xtol=v` from the command line.

From the command line, `--starts=n` will automatically restart the algorithm after it has converged so that a slightly better value can be found. If `--keep_best` is included then restart will use a value near the minimum, otherwise it will restart the fit from a random point in the parameter space.

Use `--fit=lm` to select the Levenberg-Marquardt fitter from the command line.

References

Nelder-Mead Simplex


Fit Control

Fit Algorithms

- DREAM
- Differential Evolution
- Levenberg-Marquardt
- [✓ Nelder-Mead Simplex]
- Quasi-Newton BFGS

Nelder-Mead Simplex Fitting Parameters

Steps:	1000
Starts:	1
Simplex radius:	0.15
x tolerance:	1e-06
f(x) tolerance:	1e-08

The Nelder-Mead downhill simplex algorithm is a robust optimizer which does not require the function to be continuous or differentiable. It uses the relative values of the function at the corners of a simplex (an n-dimensional triangle) to decide which points of the simplex to update. It will take the worst value and try moving it inward or outward, or reflect it through the centroid of the remaining values stopping if it finds a better value. If none of these values are better, then it will shrink the simplex and start again. The name amoeba comes from the book *Numerical Recipes* [Press1992] wherein they describe the search as acting like an amoeba, squeezing through narrow valleys as it makes its way down to the minimum.

When to use

Use this method as a first fit to your model. If your fitting function is well behaved with few local minima this will give a quick estimate of the model, and help you decide if the model needs to be refined. If your function is poorly behaved, you will need to select a good initial value before fitting, or use a more robust method such as *Differential Evolution* or *DREAM*.

The uncertainty reported comes from a numerical derivative estimate at the minimum.

This method requires a series of function updates, and does not benefit much from running in parallel.

Options

Steps is the simplex update iterations to perform. Most updates require one or two function evaluations, but shrinking the simplex evaluates every value in the simplex. Use `--steps=n` from the command line.

Starts tells the optimizer to restart a given number of times. Each time it restarts it uses a random starting point. Use `--starts=n` from the command line.

Simplex radius is the initial size of the simplex, as a portion of the bounds defining the parameter space. If a parameter is unbounded, then the radius will be treated as a portion of the parameter value. Use `--radius=n` from the command line.

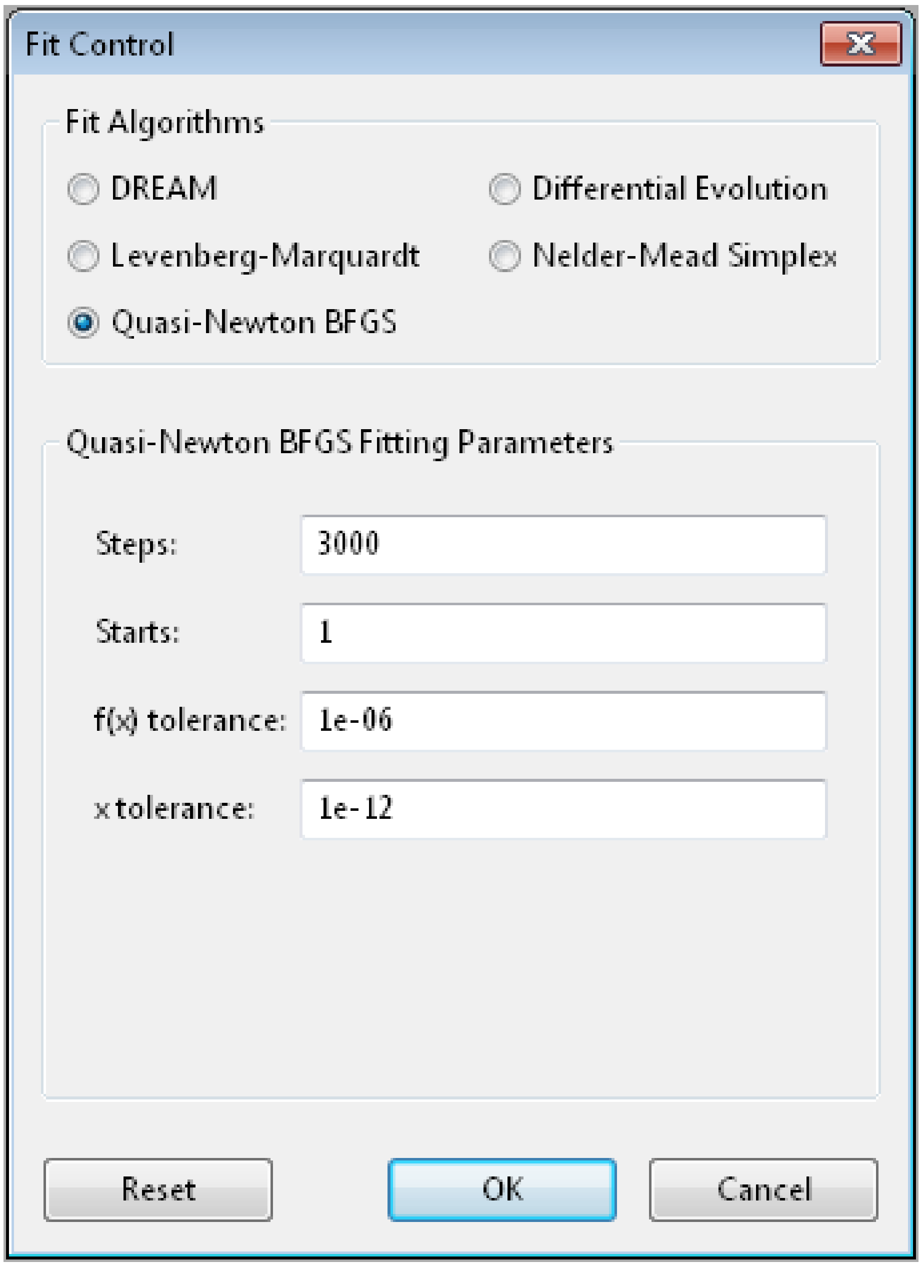
x tolerance and *f(x) tolerance* are used to determine when the fit has reached the point where no significant improvement is expected. If the simplex is tiny (that is, the corners are close to each other) and flat (that is, the values at the corners are close to each other), then the fit will terminate. Use `--xtol=v` and `--ftol=v` from the command line.

From the command line, use `--keep_best` so that restarts are centered on a value near the minimum rather than restarting from a random point within the parameter bounds.

Use `--fit=amoeba` to select the Nelder-Mead simplex fitter from the command line.

References

Quasi-Newton BFGS



Fit Control

Fit Algorithms

DREAM [ ] Differential Evolution [ ] Levenberg-Marquardt [ ] Nelder-Mead Simplex [ ] [✓ Quasi-Newton BFGS]

Quasi-Newton BFGS Fitting Parameters

Steps:	3000
Starts:	1
f(x) tolerance:	1e-06
x tolerance:	1e-12

Broyden-Fletcher-Goldfarb-Shanno is a gradient descent method which uses the gradient to determine the step direction and an approximation of the Hessian matrix to estimate the curvature and guess a step size. The step is further refined with a one-dimensional search in the direction of the gradient.

When to use

Like *Levenberg-Marquardt*, this method converges quickly to the minimum. It does not assume that the problem is in the form of a sum of squares and does not require a *residuals* method.

The n partial derivatives are computed in parallel.

Options

Steps is the number of gradient steps to take. Each step requires a calculation of the Jacobian matrix to determine the direction. This needs $2mn$ function evaluations, where n is the number of parameters and each function is evaluated and m data points (assuming center point formula for finite difference estimate of the derivative). The resulting linear equation is then solved, but for small n and expensive function evaluation this overhead can be ignored. Use `--steps=n` from the command line.

Starts tells the optimizer to restart a given number of times. Each time it restarts it uses a random starting point. Use `--starts=n` from the command line.

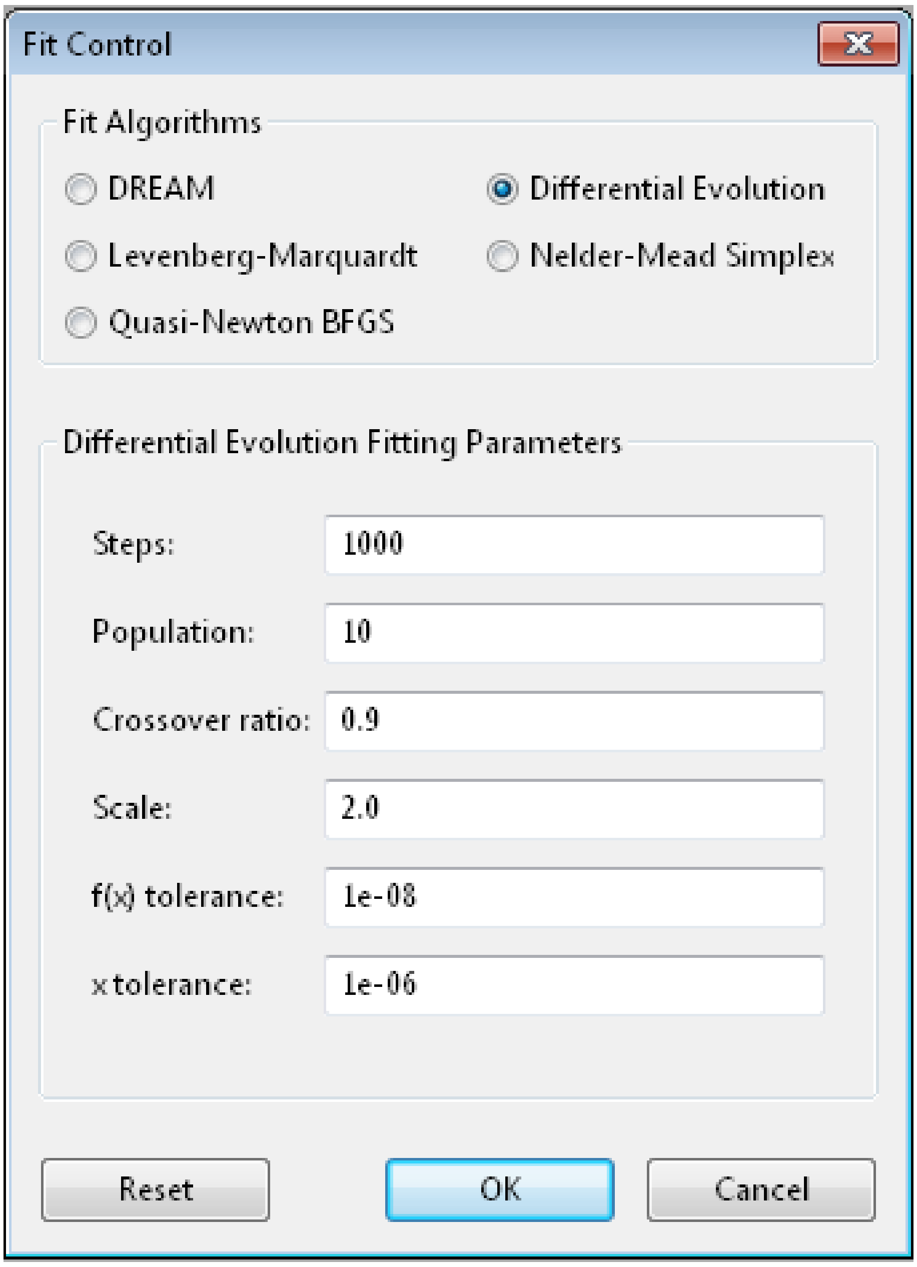
f(x) tolerance and *x tolerance* are used to determine when the fit has reached the point where no significant improvement is expected. If the function is small or the step is too short then the fit will terminate. Use `--ftol=v` and `--xtol=v` from the command line.

From the command line, `--keep_best` uses a value near the previous minimum when restarting instead of using a random value within the parameter bounds.

Use `--fit=newton` to select BFGS from the commandline.

References

Differential Evolution



Fit Algorithms

DREAM
[✓ Differential Evolution]
Levenberg-Marquardt
Nelder-Mead Simplex
Quasi-Newton BFGS

Differential Evolution Fitting Parameters

Steps:	1000
Population:	10
Crossover ratio:	0.9
Scale:	2.0
f(x) tolerance:	1e-08
x tolerance:	1e-06

Differential evolution is a population based algorithm which uses differences between points as a guide to selecting new points. For each member of the population a pair of points is chosen at random, and a difference vector is computed. This vector is scaled, and a random subset of its components are added to the current point based on crossover ratio. This new point is evaluated, and if its value is lower than the current point, it replaces it in the population. There are many variations available within DE that have not been exposed in Bumps. Interested users can modify `bumps.fitters.DEFit` and experiment with different crossover and mutation algorithms, and perhaps add them as command line options.

Differential evolution is a robust directed search strategy. Early in the search, when the population is disperse, the difference vectors are large and the search remains broad. As the search progresses, more of the population goes into the valleys and eventually all the points end up in local minima. Now the differences between random pairs will often be small and the search will become more localized.

The population is initialized according to the prior probability distribution for each each parameter. That is, if the parameter is bounded, it will use a uniform random number generate within the bounds. If it is unbounded, it will use a uniform value in [0,1]. If the parameter corresponds to the result of a previous measurement with mean μ and standard deviation σ, then the initial values will be pulled from a gaussian random number generator.

When to use

Convergence with differential evolution will be slower, but more robust.

Each update will evaluate k points in parallel, where k is the size of the population.

Options

Steps is the number of iterations. Each step updates each member of the population. The population size scales with the number of fitted parameters. Use `--steps=n` from the command line.

Population determines the size of the population. The number of individuals, k, is equal to the number of fitted parameters times the population scale factor. Use `--pop=k` from the command line.

Crossover ratio determines what proportion of the dimensions to update at each step. Smaller values will likely lead to slower convergence, but more robust results. Values must be between 0 and 1. Use `--CR=v` from the command line.

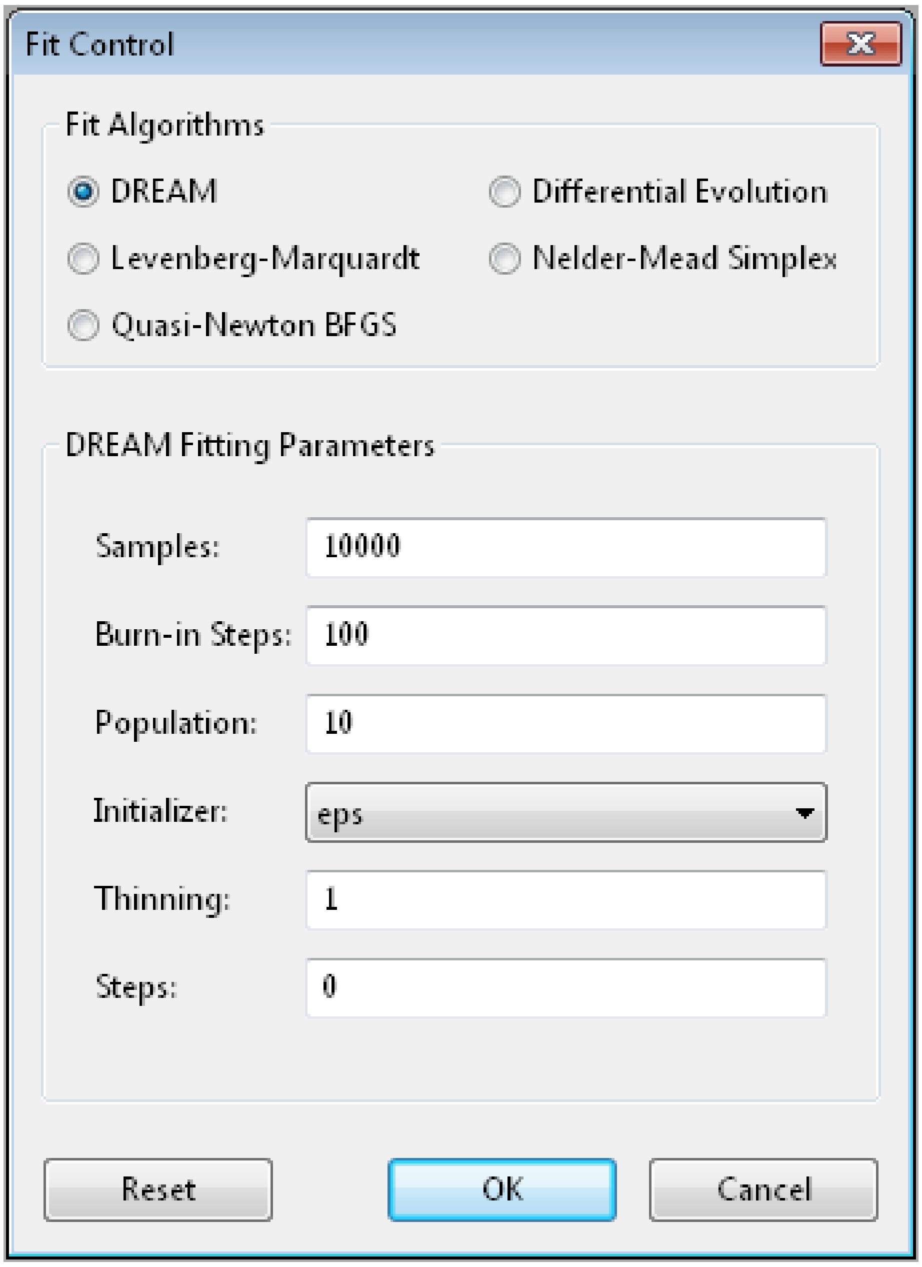
Scale determines how much to scale each difference vector before adding it to the candidate point. The selected mutation algorithm chooses a scale factor uniformly in $[0, F]$. Use `--F=v` from the command line.

f(x) tolerance and *x tolerance* are used to determine when the fit has reached the point where no significant improvement is expected. If the population is flat (that is, the minimum and maximum values are within tolerance) and tiny (that is, all the points are close to each other) then the fit will terminate. Use `ftol=v` and `xtol=v` from the command line.

Use `--fit=de` to select diffrential evolution from the commandline.

References

DREAM



Fit Algorithms: [✓ DREAM] Differential Evolution Levenberg-Marquardt Nelder-Mead Simplex Quasi-Newton BFGS

DREAM Fitting Parameters

Samples:	10000
Burn-in Steps:	100
Population:	10
Initializer:	eps
Thinning:	1
Steps:	0

DREAM is a population based algorithm like differential evolution, but instead of only keeping individuals which improve each generation, it will sometimes keep individuals which get worse. Although it is not fast and does not give the very best value for the function, we have found it to be a robust fitting engine which will give a good value given enough time.

The progress of each individual in the population from generation to generation can considered a Markov chain, whose transition probability is equal to the probability of taking the step times the probability that it keeps the step based on the difference in value between the points. By including a purely random stepper with some probability, the detailed balance condition is preserved, and the Markov chain converges onto the underlying equilibrium distribution. If the theory function represents the conditional probability of selecting each point in the parameter space, then the resulting chain is a random draw from the posterior distribution.

This means that the DREAM algorithm can be used to determine the parameter uncertainties. Unlike the hessian estimate at the minimum that is used to report uncertainties from the other fitters, the resulting uncertainty need not gaussian. Indeed, the resulting distribution can even be multi-modal. Fits to measured data using theory functions that have symmetric solutions have shown all equivalent solutions with approximately equal probability.

When to use

Use DREAM when you need a robust fitting algorithm. It takes longer but it does an excellent job of exploring different minima and getting close to the global optimum.

Use DREAM when you want a detailed analysis of the parameter uncertainty.

Like differential evolution, DREAM will evaluate k points in parallel, where k is the size of the population.

Options

Samples is the number of points to be drawn from the Markov chain. To estimate the 68% interval to two digits of precision, at least 1e5 (or 100,000) samples are needed. For the 95% interval, 1e6 (or 1,000,000) samples are needed. The default 1e4 samples gives a rough approximation of the uncertainty relatively quickly. Use `--samples=n` from the command line.

Burn-in Steps is the number of iterations to required for the Markov chain to converge to the equilibrium distribution. If the fit ends early, the tail of the burn will be saved to the start of the steps. Use `--burn=n` from the command line.

Population determines the size of the population. The number of individuals, k, is equal to the number of fitted parameters times the population scale factor. Use `--pop=k` from the command line.

Initializer determines how the population will be initialized. The options are as follows:

> *eps* (epsilon ball), in which the entire initial population is chosen at random from within a tiny hypersphere centered about the initial point
>
> *lhs* (latin hypersquare), which chops the bounds within each dimension in k equal sized chunks where k is the size of the population and makes sure that each parameter has at least one value within each chunk across the population.
>
> *cov* (covariance matrix), in which the uncertainty is estimated using the covariance matrix at the initial point, and points are selected at random from the corresponding gaussian ellipsoid
>
> *random* (uniform random), in which the points are selected at random within the bounds of the parameters

Use `--init=type` from the command line.

Thinning is the amount of thinning to use when collecting the population. If the fit is somewhat stuck, with most steps not improving the fit, then you will need to thin the population to get proper statistics. Use `--thin=k` from the command line.

Calculate entropy, if true, computes the entropy for the fit. This is an estimate of the amount of information in the data. Use `--entropy` from the command line.

Steps, if not zero, determines the number of iterations to use for drawing samples after burn in. Each iteration updates the full population, which is (population x number of fitted parameters) points. This option is available for compatibility; it is more useful to set the number of samples directly. Use `--steps=n` from the command line.

Use `--fit=dream` to select DREAM from the commandline.

Output

DREAM produces a number of different outputs, and there are a number of things to check before using its reported uncertainty values. The main goal of selecting `--burn=n` is to wait long enough to reach the equilibrium distribution.

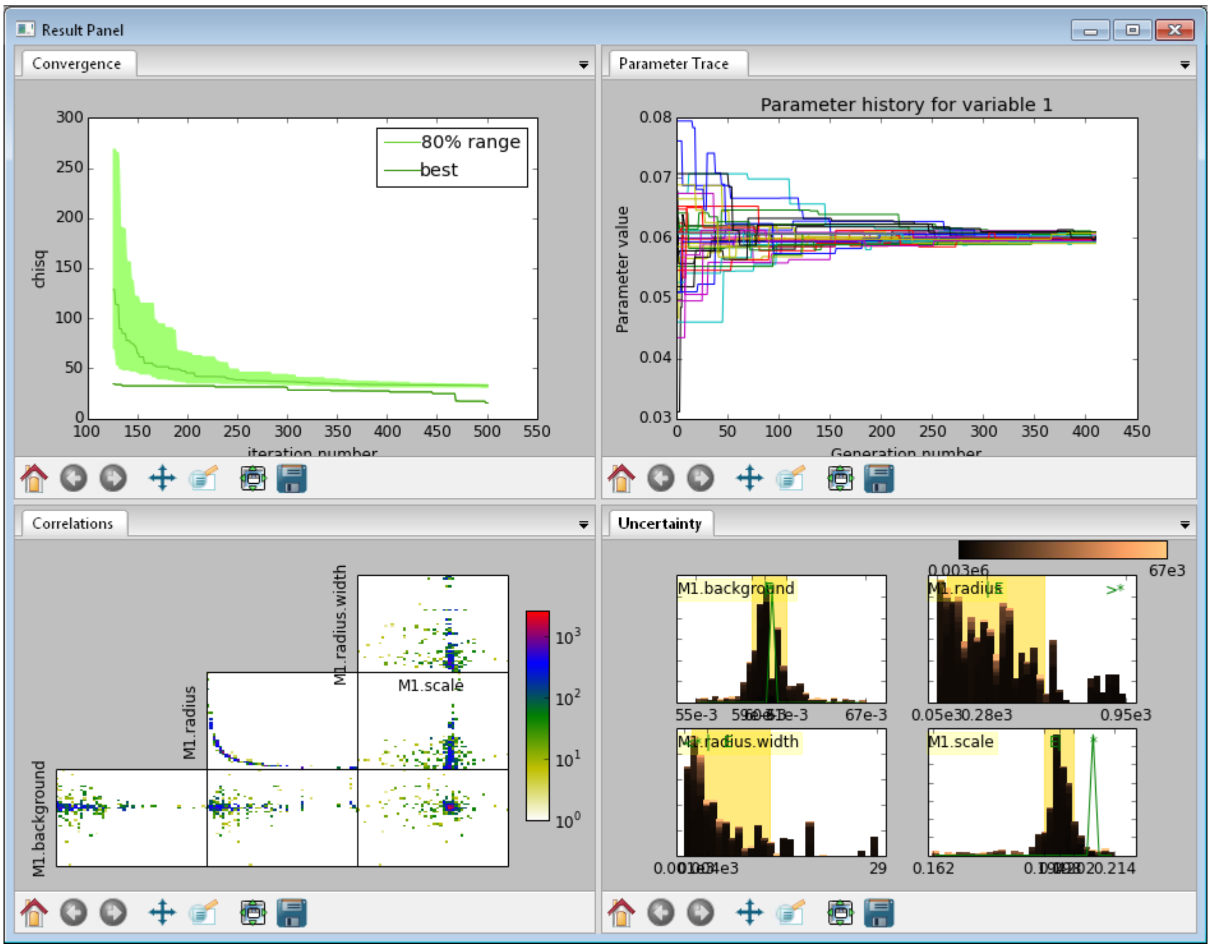


Fig. 1.136: This DREAM fit is incomplete, as can be seen on all four plots. The *Convergence* plot is still decreasing, *Parameter Trace* plot does not show random mixing of Markov chain values, the *Correlations* plots are fuzzy and mostly empty, the *Uncertainty* plot shows black histograms (indicating that there are a few stray values far away from the best) and green maximum likelihood spikes not matching the histogram (indicating that the region around the best value has not been adequately explored).

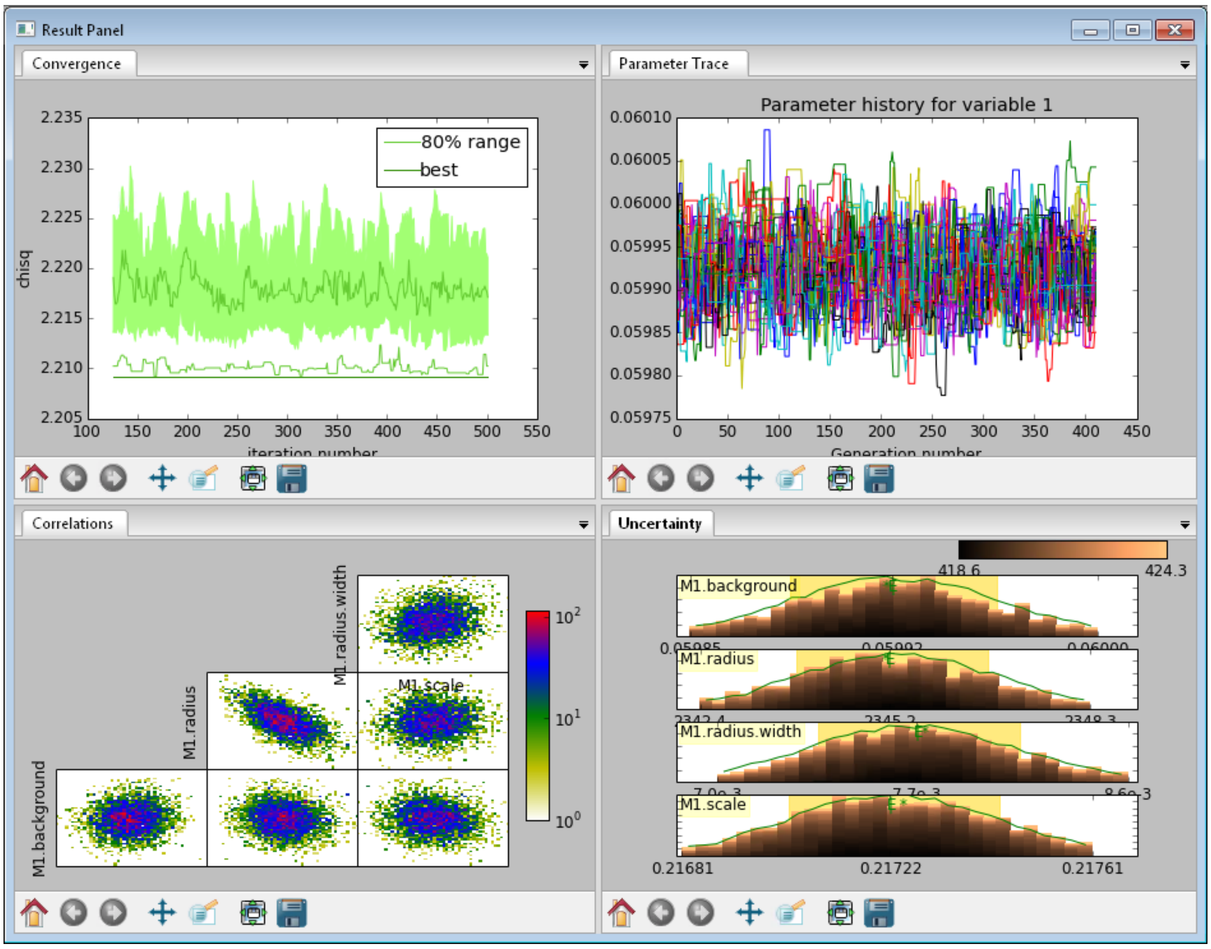


Fig. 1.137: This DREAM fit completed successfully. The *Convergence* plot is flat, the *Parameter Trace* plot is flat and messy, the *Correlateions* plots show nice blobs (and a bit of correlation between the *M1.radius* parameter and the *M1.radius.width* parameter), and the uncertainty plots show a narrow range of -log(P) values in the mostly brown histograms and a good match to the green constrained maximum likelihood line.

For each parameter in the fit, DREAM finds the mean, median and best value, as well as the 68% and 95% credible intervals. The mean value is defined as $\int xP(x)dx$, which is just the expected value of the probability distribution for the parameter. The median value is the 50% point in the probability distribution, and the best value is the maximum likelihood value seen in the random walk. The credible intervals are the central intervals which capture 68% and 95% of the parameter values respectively. You need approximately 100,000 samples to get two digits of precision on the 68% interval, and 1,000,000 samples for the 95% interval.

Table 1.1: Example fit output

#	Parameter	mean	median	best	[68% interval]	[95% interval]
1	M1.background	0.059925(41)	0.059924	0.059922	[0.05988 0.05997]	[0.05985 0.06000]
2	M1.radius	2345.3(15)	2345.234	2345.174	[2343.83 2346.74]	[2342.36 2348.29]
3	M1.radius.width	0.00775(41)	0.00774	0.00777	[0.0074 0.0081]	[0.0070 0.0086]
4	M1.scale	0.21722(20)	0.217218	0.217244	[0.21702 0.21743]	[0.21681 0.21761]

The *Convergence* plot shows the range of χ^2 values in the population for each iteration. The band shows the 68% of values around the median, and the solid line shows the minimum value. If the distribution has reached equilibrium, then convergence graph should be roughly flat, with little change in the minimum value throughout the graph. If there is no convergence, then the remaining plots don't mean much.

The *Correlations* plot shows cross correlation between each pair of parameters. If the parameters are completely uncorrelated then the boxes should contain circles. Diagonals indicate strong correlation. Square blocks indicate that the fit is not sensitive to one of the parameters. The range plotted on the correlation plot is determined by the 95% interval of the data. The individual correlation plots are too small to show the range of values for the parameters. These can instead be read from the *Uncertainty* plot for each parameter, which covers the same range of values and indicates 68% and 95% intervals. If there are some chains that are wandering around away from the minimum, then the plot will look fuzzy, and not have a nice blob in the center. If a correlation plot has multiple blobs, then there are multiple minima in your problem space, usually because there are symmetries in the problem definition. For example, a model fitting $x + a^2$ will have identical solutions for $\pm\, a$.

The *Uncertainty* plot shows histograms for each fitted parameter generated from the values for that parameter across all chains. Within each histogram bar the values are sorted and displayed as a gradient from black to copper, with black values having the lowest χ^2 and copper values having the highest. The resulting histogram should be dark brown, with a black hump in the center and light brown tips. If there are large lumps of light brown, or excessive black then its likely that the optimizer did not converge. The green line over the histogram shows the best value seen within each histogram bin (the maximum likelihood given $p_k == x$). With enough samples and proper convergence, it should roughly follow the outline of the histogram. The yellow band in the center of the plot represents the 68% interval for the data. The histogram cuts off at 95%. These values along with the median are shown as labels along the x axis. The green asterisk represents the best value, the green *E* the mean value and the vertical green line the median value. If the fit is not sensitive to a parameter, or if two parameters are strongly correlated, the parameter histogram will show a box rather than a hump. Spiky shapes (either in the histogram or the maximum likelihood line) indicate lack of convergence or maybe not enough steps. A chopped histograms indicates that the range for that parameter is too small.

The *Parameter Trace* plot is diagnostic for models which have poor mixing. In this cases no matter how the parameter values are changing, they are landing on much worse values for the χ^2. This can happen if the problem is highly constrained with many tight and twisty values.

The *Data and Theory* plot should show theory and data lining up pretty well, with the theory overlaying about 2/3 of the error bars on the data (1-σ = 68%). The *Residuals* plot shows the difference between theory and data divided by uncertainty. The residuals should be 2/3 within [-1, 1], They should not show any structure, such as humps where the theory misses the data for long stretches. This indicates some feature missing from the model, or a lack of convergence to the best model.

If entropy is requested, then bumps will show the total number of bits of information in the fit. This derives from the entropy term:

Using entropy and simulation we hope to be able to make experiment planning decisions in a way that maximizes information, by estimating whether it is better to measure more precisely or to measure different but related values and fit them with shared parameters.

References

Particle Swarm

Inspired by bird flocking behaviour, the particle swarm algorithm is a population-based method which updates an individual according to its momentum and a force toward the current best fit parameter values. We did not explore variations of this algorithm in any detail.

When to use

Particle swarm performed well enough in our low dimensional test problems, but made little progress when more fit parameters were added.

The population updates can run in parallel, but the tiny population size limits the amount of parallelism.

Options

`--steps=n` is the number of iterations. Each step updates each member of the population. The population size scales with the number of fitted parameters.

`--pop=k` determines the size of the population. The number of individuals, k, is equal to the number of fitted parameters times the population scale factor. The default scale factor is 1.

Use `--fit=ps` to select particle swarm from the commandline.

Add a few more lines

References

Random Lines

Most of the population based algorithms ignore the value of the function when choosing the points in the next iteration. Random lines is a new style of algorithm which fits a quadratic model to a selection from the population, and uses that model to propose a new point in the next generation of the population. The hope is that the method will inherit the robustness of the population based algorithms as well as the rapid convergence of the newton descent algorithms.

When to use

Random lines works very well for some of our test problems, showing rapid convergence to the optimum, but on other problems it makes very little progress.

The population updates can run in parallel.

Options

`--steps=n` is the number of iterations. Each step updates each member of the population. The population size scales with the number of fitted parameters.

`--pop=k` determines the size of the population. The number of individuals, k, is equal to the number of fitted parameters times the population scale factor. The default scale factor is 0.5.

`--CR=v` is the crossover ratio, determining what proportion of the dimensions to update at each step. Values must be between 0 and 1.

`--starts=n` tells the optimizer to restart a given number of times. Each time it restarts it uses a random starting point.

`--keep_best` uses a value near the previous minimum when restarting instead of using a random value within the parameter bounds. This option is not available in the options dialog.

Use `--fit=rl` to select random lines from the commandline.

References

Parallel Tempering

Parallel tempering is an MCMC algorithm for uncertainty analysis. This version runs at multiple temperatures simultaneously, with chains at high temperature able to more easily jump between minima and chains at low temperature to fully explore the minima. Like *DREAM* it has a differential evolution stepper, but this version uses the chain history as the population rather than maintaining a population at each temperature.

This is an experimental algorithm which does not yet perform well.

When to use

When complete, parallel tempering should be used for problems with widely spaced local minima which dream cannot fit.

Options

`--steps=n` is the number of iterations to include in the Markov chain. Each iteration updates the full population. The population size scales with the number of fitted parameters.

`--burn=n` is the number of iterations to required for the Markov chain to converge to the equilibrium distribution. If the fit ends early, the tail of the burn will be saved to the start of the steps.

`--CR=v` is the differential evolution crossover ratio to use when computing step size and direction. Use a small value to step through the dimensions one at a time, or a large value to step through all at once.

`-nT=k`, `-Tmin=v` and `--Tmax=v` specify a log-spaced initial distribution of temperatures. The default is 25 points between 0.1 and 10. *DREAM* runs at a fixed temperature of 1.0.

Use `--fit=pt` to select parallel tempering from the commandline.

References

SANS to SESANS conversion

The conversion from SANS into SESANS in absolute units is a simple Hankel transformation when all the small-angle scattered neutrons are detected. First we calculate the Hankel transform including the absolute intensities by

$$G(\delta) = 2\pi \int_0^\infty J_0(Q\delta)\frac{d\Sigma}{d\Omega}(Q)QdQ,$$

in which J_0 is the zeroth order Bessel function, δ the spin-echo length, Q the wave vector transfer and $\frac{d\Sigma}{d\Omega}(Q)$ the scattering cross section in absolute units.

Out of necessity, a 1-dimensional numerical integral approximates the exact Hankel transform. The upper bound of the numerical integral is Q_{max}, which is calculated from the wavelength and the instrument's maximum acceptance angle, both of which are included in the file. While the true Hankel transform has a lower bound of zero, most scattering models are undefined at :math: *Q=0*, so the integral requires an effective lower bound. The lower bound of the integral is $Q_{min} = 0.1 \times 2\pi/R_{max}$, in which R_{max} is the maximum length scale probed by the instrument multiplied by the number of data points. This lower bound is the minimum expected Q value for the given length scale multiplied by a constant. The constant, 0.1, was chosen empirically by integrating multiple curves and finding where the value at which the integral was stable. A constant value of 0.3 gave numerical stability to the integral, so a factor of three safety margin was included to give the final value of 0.1.

From the equation above we can calculate the polarisation that we measure in a SESANS experiment:

$$P(\delta) = e^{t\left(\frac{\lambda}{2\pi}\right)^2 (G(\delta)-G(0))},$$

in which t is the thickness of the sample and λ is the wavelength of the neutrons.

Fitting SESANS Data

Note: A proper installation of the developers setup of SasView (http://trac.sasview.org/wiki/AnacondaSetup) is a prerequisite for using these instructions.

It is possible to fit SESANS measurements from the command line in Python.

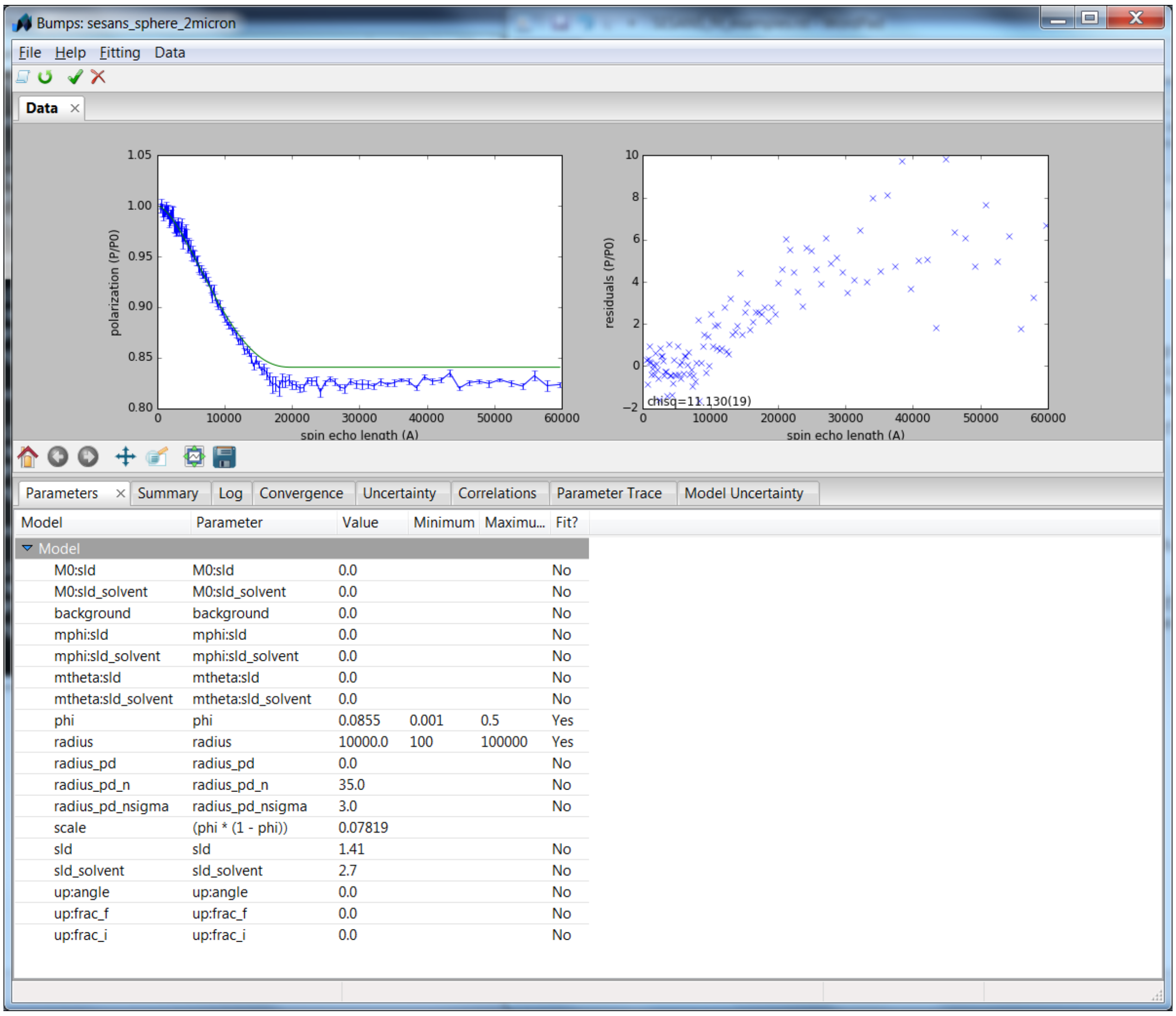
Simple Fits

In the folder sasmodels/example the file sesans_sphere_2micron.py gives an example of how to fit a shape to a measurement.

The command:

```
>python fit_sesans.py sesans_sphere_2micron.py
```

then results in a GUI from which you can control the fit.


Bumps: sesans_sphere_2micron
File Help Fitting Data
Data
polarization (P/P0)
spin echo length (A)
residuals (P/P0)
chisq=11.130(19)
Parameters Summary Log Convergence Uncertainty Correlations Parameter Trace Model Uncertainty

Model	Parameter	Value	Minimum	Maximu...	Fit?
Model					
M0:sld	M0:sld	0.0			No
M0:sld_solvent	M0:sld_solvent	0.0			No
background	background	0.0			No
mphi:sld	mphi:sld	0.0			No
mphi:sld_solvent	mphi:sld_solvent	0.0			No
mtheta:sld	mtheta:sld	0.0			No
mtheta:sld_solvent	mtheta:sld_solvent	0.0			No
phi	phi	0.0855	0.001	0.5	Yes
radius	radius	10000.0	100	100000	Yes
radius_pd	radius_pd	0.0			No
radius_pd_n	radius_pd_n	35.0			No
radius_pd_nsigma	radius_pd_nsigma	3.0			No
scale	(phi * (1 - phi))	0.07819			
sld	sld	1.41			No
sld_solvent	sld_solvent	2.7			No
up:angle	up:angle	0.0			No
up:frac_f	up:frac_f	0.0			No
up:frac_i	up:frac_i	0.0			No

All the parameters and names in sesans_sphere_2micron.py (shown below) can be adjusted to fit your own problem:

```
"""
This is a data file  used to load in sesans data and fit it using the bumps engine
"""
from bumps.names import *

import sesansfit

# Enter the model name to use
model_name = "sphere"

# DO NOT MODIFY THIS LINE
model = sesansfit.get_bumps_model(model_name)

# Enter any custom parameters
# name = Parameter(initial_value, name='name')
phi = Parameter(0.0855, name='phi')
# Add the parameters to this list that should be displayed in the fitting window
custom_params = {"phi" : phi}

# SESANS data file name
sesans_file = "spheres2micron.ses"

# Initial parameter values (if other than defaults)
```

(continues on next page)

(continued from previous page)

```
# "model_parameter_name" : value
initial_vals = {
  "sld" : 1.41,
  "radius" : 10000,
  "sld_solvent" : 2.70,
}

# Ranges for parameters if other than default
# "model_parameter_name" : [min, max]
param_range = {
  "phi" : [0.001, 0.5],
  "radius" : [100, 100000]
}

# Constraints
# model.param_name = f(other params)
# EXAMPLE: model.scale = model.radius*model.radius*(1 - phi) - where radius
# and scale are model functions and phi is a custom parameter
model.scale = phi*(1-phi)

# Send to the fitting engine
# DO NOT MODIFY THIS LINE
problem = sesansfit.sesans_fit(sesans_file, model, initial_vals, custom_params,␣
↪param_range)
```

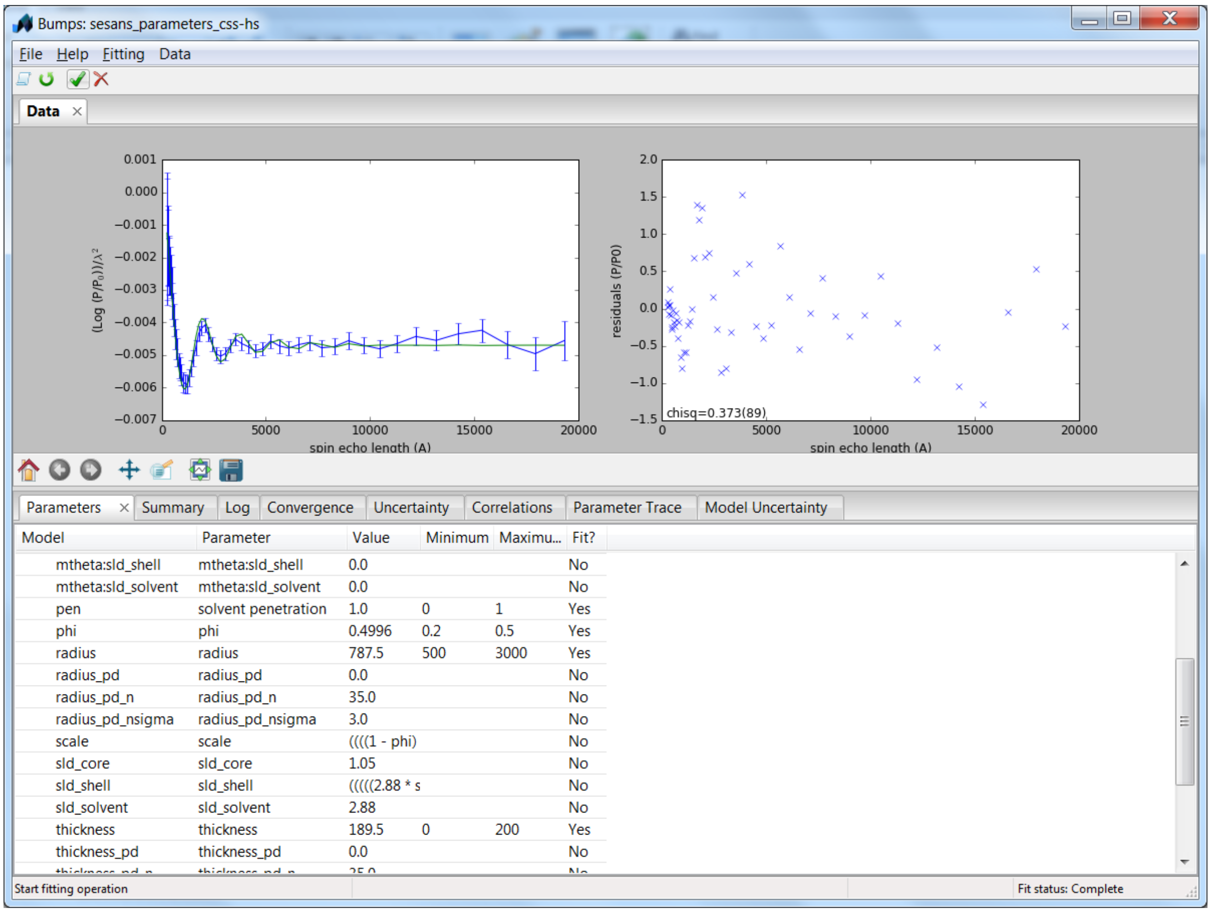
Incorporating a Structure Factor

An example of how to also include a structure factor can be seen in the following example taken from Washington et al., *Soft Matter*, (2014), 10, 3016 (dx.doi.org/10.1039/C3SM53027B). These are time-of-flight measurements, which is the reason that not the polarisation is plotted, but the $\frac{log(P/P_0)}{\lambda^2}$. The sample is a dispersion of core-shell colloids at a high volume fraction with hard sphere interactions.

The fit can be started by:

```
>python fit_sesans.py sesans_parameters_css-hs.py
```

This yields after the fitting:



Model	Parameter	Value	Minimum	Maximu...	Fit?
mtheta:sld_shell	mtheta:sld_shell	0.0			No
mtheta:sld_solvent	mtheta:sld_solvent	0.0			No
pen	solvent penetration	1.0	0	1	Yes
phi	phi	0.4996	0.2	0.5	Yes
radius	radius	787.5	500	3000	Yes
radius_pd	radius_pd	0.0			No
radius_pd_n	radius_pd_n	35.0			No
radius_pd_nsigma	radius_pd_nsigma	3.0			No
scale	scale	((((1 - phi)			No
sld_core	sld_core	1.05			No
sld_shell	sld_shell	(((((2.88 * s			No
sld_solvent	sld_solvent	2.88			No
thickness	thickness	189.5	0	200	Yes
thickness_pd	thickness_pd	0.0			No

The code sesans_parameters_css-hs.py can then be used as a template for a fitting problem with a structure factor:

```
"""
This is a data file  used to load in sesans data and fit it using the bumps engine
"""
from bumps.names import *

import sesansfit

# Enter the model name to use
model_name = "core_shell_sphere*hardsphere"

# DO NOT MODIFY THIS LINE
model = sesansfit.get_bumps_model(model_name)

# Enter any custom parameters
phi = Parameter(0.45, name='phi')
pen = Parameter(0.95, name='solvent penetration')
custom_params = {"phi" : phi, "pen" : pen}

# SESANS data file
sesans_file = "core_shell.ses"

# Initial parameter values (if other than defaults)
initial_vals = {
   "sld_core" : 1.05,
   "sld_shell" : 2.88*pen-0.05*(1-pen),
   "sld_solvent" : 2.88,
   "radius" : 730,
   "thickness" : 20,
   "volfraction" : phi,
```

(continues on next page)

(continued from previous page)

```
    "scale" : (1-phi)
}

# Ranges for parameters if other than default
param_range = {
    "phi" : [0.2, 0.5],
    "pen" : [0,1],
    "radius" : [500, 3000],
    "thickness" : [0,200]
}

# Constraints
# model.param_name = f(other params)
# EXAMPLE: model.scale = model.radius*model.radius*(1 - phi) - where radius
# and scale are model functions and phi is a custom parameter
model.scale = phi*(1-phi)
model.volfraction = phi
model.shell_sld = pen*2.88

# Send to the fitting engine
problem = sesansfit.sesans_fit(sesans_file, model_name, initial_vals, custom_
↪params, param_range)
```

Writing a Plugin Model

Overview

In addition to the models provided with the sasmodels package, you are free to create your own models.

Models can be of three types:

- A pure python model : Example - broadpeak.py
- A python model with embedded C : Example - sphere.py
- A python wrapper with separate C code : Example - cylinder.py, cylinder.c

When using SasView, plugin models should be saved to the SasView *plugin_models* folder *C:\Users\{username}.sasview\plugin_models* (on Windows) or */Users/{username}/.sasview\plugin_models* (on Mac). The next time SasView is started it will compile the plugin and add it to the list of *Plugin Models* in a FitPage. Scripts can load the models from anywhere.

The built-in modules are available in the *models* subdirectory of the sasmodels package. For SasView on Windows, these will be found in *C:\Program Files (x86)\SasView\sasmodels-data\models*. On Mac OSX, these will be within the application bundle as */Applications/SasView 4.0.app/Contents/Resources/sasmodels-data/models*.

Other models are available for download from the Model Marketplace. You can contribute your own models to the Marketplace as well.

Create New Model Files

Copy the appropriate files to your plugin models directory (we recommend using the examples above as templates) as mymodel.py (and mymodel.c, etc) as required, where “mymodel” is the name for the model you are creating.

Please follow these naming rules:

- No capitalization and thus no CamelCase
- If necessary use underscore to separate words (i.e. barbell not BarBell or broad_peak not BroadPeak)
- Do not include “model” in the name (i.e. barbell not BarBellModel)

Edit New Model Files

Model Contents

The model interface definition is in the .py file. This file contains:

- **a model name:**
    - this is the **name** string in the *.py* file
    - titles should be:
    - all in *lower* case
    - without spaces (use underscores to separate words instead)
    - without any capitalization or CamelCase
    - without incorporating the word "model"
    - examples: *barbell* **not** *BarBell*; *broad_peak* **not** *BroadPeak*; *barbell* **not** *BarBellModel*
- **a model title:**
    - this is the **title** string in the *.py* file
    - this is a one or two line description of the model, which will appear at the start of the model documentation and as a tooltip in the SasView GUI
- **a short description:**
    - this is the **description** string in the *.py* file
    - this is a medium length description which appears when you click *Description* on the model FitPage
- **a parameter table:**
    - this will be auto-generated from the *parameters* in the *.py* file
- **a long description:**
    - this is ReStructuredText enclosed between the r"“" and “"" delimiters at the top of the *.py* file
    - what you write here is abstracted into the SasView help documentation
    - this is what other users will refer to when they want to know what your model does; so please be helpful!
- **a definition of the model:**
    - as part of the **long description**
- **a formula defining the function the model calculates:**
    - as part of the **long description**
- **an explanation of the parameters:**
    - as part of the **long description**
    - explaining how the symbols in the formula map to the model parameters
- **a plot of the function, with a figure caption:**
    - this is automatically generated from your default parameters
- **at least one reference:**
    - as part of the **long description**
    - specifying where the reader can obtain more information about the model
- **the name of the author**

  - as part of the **long description**
  - the *.py* file should also contain a comment identifying *who* converted/created the model file

Models that do not conform to these requirements will *never* be incorporated into the built-in library.

Model Documentation

The *.py* file starts with an r (for raw) and three sets of quotes to start the doc string and ends with a second set of three quotes. For example:

```
r"""
Definition
----------

The 1D scattering intensity of the sphere is calculated in the following
way (Guinier, 1955)

.. math::

    I(q) = \frac{\text{scale}}{V} \cdot \left[
        3V(\Delta\rho) \cdot \frac{\sin(qr) - qr\cos(qr))}{(qr)^3}
        \right]^2 + \text{background}

where *scale* is a volume fraction, :math:`V` is the volume of the scatterer,
:math:`r` is the radius of the sphere and *background* is the background level.
*sld* and *sld_solvent* are the scattering length densities (SLDs) of the
scatterer and the solvent respectively, whose difference is :math:`\Delta\rho`.

You can included figures in your documentation, as in the following
figure for the cylinder model.

.. figure:: img/cylinder_angle_definition.jpg

    Definition of the angles for oriented cylinders.

References
----------

A Guinier, G Fournet, *Small-Angle Scattering of X-Rays*,
John Wiley and Sons, New York, (1955)
"""
```

This is where the FULL documentation for the model goes (to be picked up by the automatic documentation system). Although it feels odd, you should start the documentation immediately with the **definition**—the model name, a brief description and the parameter table are automatically inserted above the definition, and the a plot of the model is automatically inserted before the **reference**.

Figures can be included using the *figure* command, with the name of the *.png* file containing the figure and a caption to appear below the figure. Figure numbers will be added automatically.

See this Sphinx cheat sheet for a quick guide to the documentation layout commands, or the Sphinx Documentation for complete details.

The model should include a **formula** written using LaTeX markup. The example above uses the *math* command to make a displayed equation. You can also use *$formula$* for an inline formula. This is handy for defining the relationship between the model parameters and formula variables, such as the phrase "$r$ is the radius" used above. The live demo MathJax page http://www.mathjax.org/ is handy for checking that the equations will look like you intend.

Math layout uses the amsmath package for aligning equations (see amsldoc.pdf on that page for complete documentation). You will automatically be in an aligned environment, with blank lines separating the lines of the equation. Place an ampersand before the operator on which to align. For example:

```
.. math::

  x + y &= 1 \\
  y &= x - 1
```

produces

$$
\begin{aligned}
x + y &= 1 \\
y &= x - 1
\end{aligned}
$$

If you need more control, use:

```
.. math::
    :nowrap:
```

Model Definition

Following the documentation string, there are a series of definitions:

```
name = "sphere"  # optional: defaults to the filename without .py

title = "Spheres with uniform scattering length density"

description = """\
P(q)=(scale/V)*[3V(sld-sld_solvent)*(sin(qr)-qr cos(qr))
                /(qr)^3]^2 + background
    r: radius of sphere
    V: The volume of the scatter
    sld: the SLD of the sphere
    sld_solvent: the SLD of the solvent
"""

category = "shape:sphere"

single = True   # optional: defaults to True

opencl = False  # optional: defaults to False

structure_factor = False  # optional: defaults to False
```

name = "mymodel" defines the name of the model that is shown to the user. If it is not provided it will use the name of the model file. The name must be a valid variable name, starting with a letter and contains only letters, numbers or underscore. Spaces, dashes, and other symbols are not permitted.

title = "short description" is short description of the model which is included after the model name in the automatically generated documentation. The title can also be used for a tooltip.

description = """doc string"""" is a longer description of the model. It shows up when you press the "Description" button of the SasView FitPage. It should give a brief description of the equation and the parameters without the need to read the entire model documentation. The triple quotes allow you to write the description over multiple lines. Keep the lines short since the GUI will wrap each one separately if they are too long. **Make sure the parameter names in the description match the model definition!**

category = "shape:sphere" defines where the model will appear in the model documentation. In this example, the model will appear alphabetically in the list of spheroid models in the *Shape* category.

single = True indicates that the model can be run using single precision floating point values. Set it to False if the numerical calculation for the model is unstable, which is the case for about 20 of the built in models. It is worthwhile modifying the calculation to support single precision, allowing models to run up to 10 times faster. The section *Test_Your_New_Model* describes how to compare model values for single vs. double precision so you can decide if you need to set single to False.

opencl = False indicates that the model should not be run using OpenCL. This may be because the model definition includes code that cannot be compiled for the GPU (for example, goto statements). It can also be used for large models which can't run on most GPUs. This flag has not been used on any of the built in models; models which were failing were streamlined so this flag was not necessary.

structure_factor = True indicates that the model can be used as a structure factor to account for interactions between particles. See *Form_Factors* for more details.

model_info = . . . lets you define a model directly, for example, by loading and modifying existing models. This is done implicitly by `sasmodels.core.load_model_info()`, which can create a mixture model from a pair of existing models. For example:

```
from sasmodels.core import load_model_info
model_info = load_model_info('sphere+cylinder')
```

See `sasmodels.modelinfo.ModelInfo` for details about the model attributes that are defined.

Model Parameters

Next comes the parameter table. For example:

```
# pylint: disable=bad-whitespace, line-too-long
#    ["name",          "units", default, [min, max], "type",    "description"],
parameters = [
    ["sld",           "1e-6/Ang^2",  1, [-inf, inf], "sld",     "Layer scattering␣
↪length density"],
    ["sld_solvent", "1e-6/Ang^2",  6, [-inf, inf], "sld",     "Solvent scattering␣
↪length density"],
    ["radius",       "Ang",         50, [0, inf],    "volume", "Sphere radius"],
]
# pylint: enable=bad-whitespace, line-too-long
```

parameters = [["name", "units", default, [min,max], "type", "tooltip"],. . .] defines the parameters that form the model.

Note: The order of the parameters in the definition will be the order of the parameters in the user interface and the order of the parameters in Iq(), Iqac(), Iqabc() and form_volume(). And *scale* and *background* parameters are implicit to all models, so they do not need to be included in the parameter table.

- **"name"** is the name of the parameter shown on the FitPage.
  - the name must be a valid variable name, starting with a letter and containing only letters, numbers and underscore.
  - parameter names should follow the mathematical convention; e.g., *radius_core* not *core_radius*, or *sld_solvent* not *solvent_sld*.
  - model parameter names should be consistent between different models, so *sld_solvent*, for example, should have exactly the same name in every model.
  - to see all the parameter names currently in use, type the following in the python shell/editor under the Tools menu:

    ```
    import sasmodels.list_pars
    sasmodels.list_pars.list_pars()
    ```

    re-use as many as possible!!!
  - use "name[n]" for multiplicity parameters, where *n* is the name of the parameter defining the number of shells/layers/segments, etc.
- **"units"** are displayed along with the parameter name
  - every parameter should have units; use "None" if there are no units.

  - **sld's should be given in units of 1e-6/Ang^2, and not simply 1/Ang^2 to be consistent with the builtin models. Adjust your formulas appropriately.**
  - fancy units markup is available for some units, including:

    ```
    Ang, 1/Ang, 1/Ang^2, 1e-6/Ang^2, degrees, 1/cm, Ang/cm, g/cm^3, mg/m^2
    ```

  - the list of units is defined in the variable *RST_UNITS* within sasmodels/generate.py
    * new units can be added using the macros defined in *doc/rst_prolog* in the sasmodels source.
    * units should be properly formatted using sub-/super-scripts and using negative exponents instead of the / operator, though the unit name should use the / operator for consistency.
    * please post a message to the SasView developers mailing list with your changes.
- **default** is the initial value for the parameter.
  - **the parameter default values are used to auto-generate a plot of the model function in the documentation.**
- **[min, max]** are the lower and upper limits on the parameter.
  - lower and upper limits can be any number, or *-inf* or *inf*.
  - the limits will show up as the default limits for the fit making it easy, for example, to force the radius to always be greater than zero.
  - these are hard limits defining the valid range of parameter values; polydisperity distributions will be truncated at the limits.
- **"type"** can be one of: "", "sld", "volume", or "orientation".
  - "sld" parameters can have magnetic moments when fitting magnetic models; depending on the spin polarization of the beam and the q value being examined, the effective sld for that material will be used to compute the scattered intensity.
  - "volume" parameters are passed to Iq(), Iqac(), Iqabc() and form_volume(), and have polydispersity loops generated automatically.
  - "orientation" parameters are not passed, but instead are combined with orientation dispersity to translate *qx* and *qy* to *qa*, *qb* and *qc*. These parameters should appear at the end of the table with the specific names *theta*, *phi* and for asymmetric shapes *psi*, in that order.

Some models will have integer parameters, such as number of pearls in the pearl necklace model, or number of shells in the multi-layer vesicle model. The optimizers in BUMPS treat all parameters as floating point numbers which can take arbitrary values, even for integer parameters, so your model should round the incoming parameter value to the nearest integer inside your model you should round to the nearest integer. In C code, you can do this using:

```
static double
Iq(double q, ..., double fp_n, ...)
{
    int n = (int)(fp_n + 0.5);
    ...
}
```

in python:

```
def Iq(q, ..., n, ...):
    n = int(n+0.5)
    ...
```

Derivative based optimizers such as Levenberg-Marquardt will not work for integer parameters since the partial derivative is always zero, but the remaining optimizers (DREAM, differential evolution, Nelder-Mead simplex) will still function.

Model Computation

Models can be defined as pure python models, or they can be a mixture of python and C models. C models are run on the GPU if it is available, otherwise they are compiled and run on the CPU.

Models are defined by the scattering kernel, which takes a set of parameter values defining the shape, orientation and material, and returns the expected scattering. Polydispersity and angular dispersion are defined by the computational infrastructure. Any parameters defined as "volume" parameters are polydisperse, with polydispersity defined in proportion to their value. "orientation" parameters use angular dispersion defined in degrees, and are not relative to the current angle.

Based on a weighting function $G(x)$ and a number of points n, the computed value is

$$\hat{I}(q) = \frac{\int G(x) I(q, x)\, dx}{\int G(x) V(x)\, dx} \approx \frac{\sum_{i=1}^{n} G(x_i) I(q, x_i)}{\sum_{i=1}^{n} G(x_i) V(x_i)}$$

That is, the individual models do not need to include polydispersity calculations, but instead rely on numerical integration to compute the appropriately smeared pattern.

Each .py file also contains a function:

```
def random():
...
```

This function provides a model-specific random parameter set which shows model features in the USANS to SANS range. For example, core-shell sphere sets the outer radius of the sphere logarithmically in *[20, 20,000]*, which sets the Q value for the transition from flat to falling. It then uses a beta distribution to set the percentage of the shape which is shell, giving a preference for very thin or very thick shells (but never 0% or 100%). Using *-sets=10* in sascomp should show a reasonable variety of curves over the default sascomp q range. The parameter set is returned as a dictionary of *{parameter: value, ...}*. Any model parameters not included in the dictionary will default according to the code in the *_randomize_one()* function from sasmodels/compare.py.

Python Models

For pure python models, define the *Iq* function:

```
import numpy as np
from numpy import cos, sin, ...

def Iq(q, par1, par2, ...):
    return I(q, par1, par2, ...)
Iq.vectorized = True
```

The parameters *par1, par2, ...* are the list of non-orientation parameters to the model in the order that they appear in the parameter table. **Note that the auto-generated model file uses x rather than q.**

The *.py* file should import trigonometric and exponential functions from numpy rather than from math. This lets us evaluate the model for the whole range of q values at once rather than looping over each q separately in python. With q as a vector, you cannot use if statements, but must instead do tricks like

```
a = x*q*(q>0) + y*q*(q<=0)
```

or

```
a = np.empty_like(q)
index = q>0
a[index] = x*q[index]
a[~index] = y*q[~index]
```

which sets a to $q \cdot x$ if q is positive or $q \cdot y$ if q is zero or negative. If you have not converted your function to use q vectors, you can set the following and it will only receive one q value at a time:

```
Iq.vectorized = False
```

Return np.NaN if the parameters are not valid (e.g., cap_radius < radius in barbell). If I(q; pars) is NaN for any q, then those parameters will be ignored, and not included in the calculation of the weighted polydispersity.

Models should define *form_volume(par1, par2, . . .)* where the parameter list includes the *volume* parameters in order. This is used for a weighted volume normalization so that scattering is on an absolute scale. If *form_volume* is not defined, then the default *form_volume = 1.0* will be used.

Embedded C Models

Like pure python models, inline C models need to define an *Iq* function:

```
Iq = """
    return I(q, par1, par2, ...);
"""
```

This expands into the equivalent C code:

```
#include <math.h>
double Iq(double q, double par1, double par2, ...);
double Iq(double q, double par1, double par2, ...)
{
    return I(q, par1, par2, ...);
}
```

form_volume defines the volume of the shape. As in python models, it includes only the volume parameters.

source=['fn.c', . . .] includes the listed C source files in the program before *Iq* and *form_volume* are defined. This allows you to extend the library of C functions available to your model.

c_code includes arbitrary C code into your kernel, which can be handy for defining helper functions for *Iq* and *form_volume*. Note that you can put the full function definition for *Iq* and *form_volume* (include function declaration) into *c_code* as well, or put them into an external C file and add that file to the list of sources.

Models are defined using double precision declarations for the parameters and return values. When a model is run using single precision or long double precision, each variable is converted to the target type, depending on the precision requested.

Floating point constants must include the decimal point. This allows us to convert values such as 1.0 (double precision) to 1.0f (single precision) so that expressions that use these values are not promoted to double precision expressions. Some graphics card drivers are confused when functions that expect floating point values are passed integers, such as 4*atan(1); it is safest to not use integers in floating point expressions. Even better, use the builtin constant M_PI rather than 4*atan(1); it is faster and smaller!

The C model operates on a single q value at a time. The code will be run in parallel across different q values, either on the graphics card or the processor.

Rather than returning NAN from Iq, you must define the *INVALID(v)*. The *v* parameter lets you access all the parameters in the model using *v.par1*, *v.par2*, etc. For example:

```
#define INVALID(v) (v.bell_radius < v.radius)
```

The INVALID define can go into *Iq*, or *c_code*, or an external C file listed in *source*.

Oriented Shapes

If the scattering is dependent on the orientation of the shape, then you will need to include *orientation* parameters *theta*, *phi* and *psi* at the end of the parameter table. As described in the section *Oriented particles*, the individual (q_x, q_y) points on the detector will be rotated into (q_a, q_b, q_c) points relative to the sample in its canonical orientation with a-b-c aligned with x-y-z in the laboratory frame and beam travelling along $-z$.

The oriented C model is called using *Iqabc(qa, qb, qc, par1, par2, . . .)* where *par1*, etc. are the parameters to the model. If the shape is rotationally symmetric about *c* then *psi* is not needed, and the model is called as *Iqac(qab, qc, par1, par2, . . .)*. In either case, the orientation parameters are not included in the function call.

For 1D oriented shapes, an integral over all angles is usually needed for the *Iq* function. Given symmetry and the substitution $u = \cos(\alpha)$, $du = -\sin(\alpha)\, d\alpha$ this becomes

$$\begin{aligned}
I(q) &= \frac{1}{4\pi}\int_{-\pi/2}^{pi/2}\int_{-pi}^{pi} F(q_a, q_b, q_c)^2 \sin(\alpha)\, d\beta\, d\alpha \\
&= \frac{8}{4\pi}\int_{0}^{pi/2}\int_{0}^{\pi/2} F^2 \sin(\alpha)\, d\beta\, d\alpha \\
&= \frac{8}{4\pi}\int_{1}^{0}\int_{0}^{\pi/2} -F^2\, d\beta\, du \\
&= \frac{8}{4\pi}\int_{0}^{1}\int_{0}^{\pi/2} F^2\, d\beta\, du
\end{aligned}$$

for

$$\begin{aligned}
q_a &= q\sin(\alpha)\sin(\beta) = q\sqrt{1-u^2}\sin(\beta) \\
q_b &= q\sin(\alpha)\cos(\beta) = q\sqrt{1-u^2}\cos(\beta) \\
q_c &= q\cos(\alpha) = qu
\end{aligned}$$

Using the z, w values for Gauss-Legendre integration in “lib/gauss76.c”, the numerical integration is then:

```
double outer_sum = 0.0;
for (int i = 0; i < GAUSS_N; i++) {
    const double cos_alpha = 0.5*GAUSS_Z[i] + 0.5;
    const double sin_alpha = sqrt(1.0 - cos_alpha*cos_alpha);
    const double qc = cos_alpha * q;
    double inner_sum = 0.0;
    for (int j = 0; j < GAUSS_N; j++) {
        const double beta = M_PI_4 * GAUSS_Z[j] + M_PI_4;
        double sin_beta, cos_beta;
        SINCOS(beta, sin_beta, cos_beta);
        const double qa = sin_alpha * sin_beta * q;
        const double qb = sin_alpha * cos_beta * q;
        const double form = Fq(qa, qb, qc, ...);
        inner_sum += GAUSS_W[j] * form * form;
    }
    outer_sum += GAUSS_W[i] * inner_sum;
}
outer_sum *= 0.25; // = 8/(4 pi) * outer_sum * (pi/2) / 4
```

The *z* values for the Gauss-Legendre integration extends from -1 to 1, so the double sum of *w[i]w[j]* explains the factor of 4. Correcting for the average *dz[i]dz[j]* gives $(1-0)\cdot(\pi/2-0) = \pi/2$. The $8/(4\pi)$ factor comes from the integral over the quadrant. With less symmetry (eg., in the bcc and fcc paracrystal models), then an integral over the entire sphere may be necessary.

For simpler models which are rotationally symmetric a single integral suffices:

$$\begin{aligned}
I(q) &= \frac{1}{\pi}\int_{-\pi/2}^{\pi/2} F(q_{ab}, q_c)^2 \sin(\alpha)\, d\alpha/\pi \\
&= \frac{2}{\pi}\int_{0}^{1} F^2\, du
\end{aligned}$$

for

$$\begin{aligned}
q_{ab} &= q\sin(\alpha) = q\sqrt{1-u^2} \\
q_c &= q\cos(\alpha) = qu
\end{aligned}$$

with integration loop:

```
double sum = 0.0;
for (int i = 0; i < GAUSS_N; i++) {
    const double cos_alpha = 0.5*GAUSS_Z[i] + 0.5;
    const double sin_alpha = sqrt(1.0 - cos_alpha*cos_alpha);
    const double qab = sin_alpha * q;
    const double qc = cos_alpha * q;
    const double form = Fq(qab, qc, ...);
    sum += GAUSS_W[j] * form * form;
}
sum *= 0.5; // = 2/pi * sum * (pi/2) / 2
```

Magnetism

Magnetism is supported automatically for all shapes by modifying the effective SLD of particle according to the Halpern-Johnson vector describing the interaction between neutron spin and magnetic field. All parameters marked as type *sld* in the parameter table are treated as possibly magnetic particles with magnitude *M0* and direction *mtheta* and *mphi*. Polarization parameters are also provided automatically for magnetic models to set the spin state of the measurement.

For more complicated systems where magnetism is not uniform throughout the individual particles, you will need to write your own models. You should not mark the nuclear sld as type *sld*, but instead leave them unmarked and provide your own magnetism and polarization parameters. For 2D measurements you will need (q_x, q_y) values for the measurement to compute the proper magnetism and orientation, which you can implement using *Iqxy(qx, qy, par1, par2, . . .)*.

Special Functions

The C code follows the C99 standard, with the usual math functions, as defined in OpenCL. This includes the following:

M_PI, M_PI_2, M_PI_4, M_SQRT1_2, M_E: π, $\pi/2$, $\pi/4$, $1/\sqrt{2}$ and Euler's constant e

exp, log, pow(x,y), expm1, log1p, sqrt, cbrt: Power functions e^x, $\ln x$, x^y, $e^x - 1$, $\ln 1 + x$, $\sqrt{x}$, $\sqrt[3]{x}$. The functions expm1(x) and log1p(x) are accurate across all x, including x very close to zero.

sin, cos, tan, asin, acos, atan: Trigonometry functions and inverses, operating on radians.

sinh, cosh, tanh, asinh, acosh, atanh: Hyperbolic trigonometry functions.

atan2(y,x): Angle from the x-axis to the point (x, y), which is equal to $\tan^{-1}(y/x)$ corrected for quadrant. That is, if x and y are both negative, then atan2(y,x) returns a value in quadrant III where atan(y/x) would return a value in quadrant I. Similarly for quadrants II and IV when x and y have opposite sign.

fabs(x), fmin(x,y), fmax(x,y), trunc, rint: Floating point functions. rint(x) returns the nearest integer.

NAN: NaN, Not a Number, $0/0$. Use isnan(x) to test for NaN. Note that you cannot use `x == NAN` to test for NaN values since that will always return false. NAN does not equal NAN! The alternative, `x != x` may fail if the compiler optimizes the test away.

INFINITY: $\infty, 1/0$. Use isinf(x) to test for infinity, or isfinite(x) to test for finite and not NaN.

erf, erfc, tgamma, lgamma: do not use Special functions that should be part of the standard, but are missing or inaccurate on some platforms. Use sas_erf, sas_erfc, sas_gamma and sas_lgamma instead (see below).

Some non-standard constants and functions are also provided:

M_PI_180, M_4PI_3: $\frac{\pi}{180}$, $\frac{4\pi}{3}$

SINCOS(x, s, c): Macro which sets s=sin(x) and c=cos(x). The variables c and s must be declared first.

square(x): x^2

cube(x): x^3

sas_sinx_x(x): $\sin(x)/x$, with limit $\sin(0)/0 = 1$.

powr(x, y): x^y for $x \geq 0$; this is faster than general x^y on some GPUs.

pown(x, n): x^n for n integer; this is faster than general x^n on some GPUs.

FLOAT_SIZE: The number of bytes in a floating point value. Even though all variables are declared double, they may be converted to single precision float before running. If your algorithm depends on precision (which is not uncommon for numerical algorithms), use the following:

```
#if FLOAT_SIZE>4
... code for double precision ...
#else
... code for single precision ...
#endif
```

SAS_DOUBLE: A replacement for `double` so that the declared variable will stay double precision; this should generally not be used since some graphics cards do not support double precision. There is no provision for forcing a constant to stay double precision.

The following special functions and scattering calculations are defined in sasmodels/models/lib. These functions have been tuned to be fast and numerically stable down to $q = 0$ even in single precision. In some cases they work around bugs which appear on some platforms but not others, so use them where needed. Add the files listed in `source = ["lib/file.c", ...]` to your *model.py* file in the order given, otherwise these functions will not be available.

polevl(x, c, n): Polynomial evaluation $p(x) = \sum_{i=0}^{n} c_i x^i$ using Horner's method so it is faster and more accurate.

$c = \{c_n, c_{n-1}, \ldots, c_0\}$ is the table of coefficients, sorted from highest to lowest.

`source = ["lib/polevl.c", ...]` (link to code)

p1evl(x, c, n): Evaluation of normalized polynomial $p(x) = x^n + \sum_{i=0}^{n-1} c_i x^i$ using Horner's method so it is faster and more accurate.

$c = \{c_{n-1}, c_{n-2} \ldots, c_0\}$ is the table of coefficients, sorted from highest to lowest.

`source = ["lib/polevl.c", ...]` (polevl.c)

sas_gamma(x): Gamma function $\text{sas\_gamma}(x) = \Gamma(x)$.

The standard math function, tgamma(x), is unstable for $x < 1$ on some platforms.

`source = ["lib/sas_gamma.c", ...]` (sas_gamma.c)

sas_gammaln(x): log gamma function $\text{sas\_gammaln}(x) = \log \Gamma(|x|)$.

The standard math function, lgamma(x), is incorrect for single precision on some platforms.

`source = ["lib/sas_gammainc.c", ...]` (sas_gammainc.c)

sas_gammainc(a, x), sas_gammaincc(a, x): Incomplete gamma function $\text{sas\_gammainc}(a, x) = \int_0^x t^{a-1} e^{-t}\, dt / \Gamma(a)$ and complementary incomplete gamma function $\text{sas\_gammaincc}(a, x) = \int_x^\infty t^{a-1} e^{-t}\, dt / \Gamma(a)$

`source = ["lib/sas_gammainc.c", ...]` (sas_gammainc.c)

sas_erf(x), sas_erfc(x): Error function $\text{sas\_erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt$ and complementary error function $\text{sas\_erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2}\, dt$.

The standard math functions erf(x) and erfc(x) are slower and broken on some platforms.

`source = ["lib/polevl.c", "lib/sas_erf.c", ...]` (sas_erf.c)

sas_J0(x): Bessel function of the first kind sas_J0(x) $=$ $J_0(x)$ where $J_0(x) = \frac{1}{\pi}\int_0^\pi \cos(x\sin(\tau))\,d\tau$.

The standard math function j0(x) is not available on all platforms.

`source = ["lib/polevl.c", "lib/sas_J0.c", ...]` (sas_J0.c)

sas_J1(x): Bessel function of the first kind sas_J1(x) $=$ $J_1(x)$ where $J_1(x) = \frac{1}{\pi}\int_0^\pi \cos(\tau - x\sin(\tau))\,d\tau$.

The standard math function j1(x) is not available on all platforms.

`source = ["lib/polevl.c", "lib/sas_J1.c", ...]` (sas_J1.c)

sas_JN(n, x): Bessel function of the first kind and integer order n, sas_JN$(n, x) = J_n(x)$ where $J_n(x) = \frac{1}{\pi}\int_0^\pi \cos(n\tau - x\sin(\tau))\,d\tau$. If n = 0 or 1, it uses sas_J0(x) or sas_J1(x), respectively.

Warning: JN(n,x) can be very inaccurate (0.1%) for x not in [0.1, 100].

The standard math function jn(n, x) is not available on all platforms.

`source = ["lib/polevl.c", "lib/sas_J0.c", "lib/sas_J1.c", "lib/sas_JN.c", ...]` (sas_JN.c)

sas_Si(x): Sine integral Si$(x) = \int_0^x \frac{\sin t}{t}\,dt$.

Warning: Si(x) can be very inaccurate (0.1%) for x in [0.1, 100].

This function uses Taylor series for small and large arguments:

For large arguments use the following Taylor series,

$$\text{Si}(x) \sim \frac{\pi}{2} - \frac{\cos(x)}{x}\left(1 - \frac{2!}{x^2} + \frac{4!}{x^4} - \frac{6!}{x^6}\right) - \frac{\sin(x)}{x}\left(\frac{1}{x} - \frac{3!}{x^3} + \frac{5!}{x^5} - \frac{7!}{x^7}\right)$$

For small arguments,

$$\text{Si}(x) \sim x - \frac{x^3}{3\times 3!} + \frac{x^5}{5\times 5!} - \frac{x^7}{7\times 7!} + \frac{x^9}{9\times 9!} - \frac{x^{11}}{11\times 11!}$$

`source = ["lib/Si.c", ...]` (Si.c)

sas_3j1x_x(x): Spherical Bessel form sph_j1c$(x) = 3j_1(x)/x = 3(\sin(x) - x\cos(x))/x^3$, with a limiting value of 1 at $x = 0$, where $j_1(x)$ is the spherical Bessel function of the first kind and first order.

This function uses a Taylor series for small x for numerical accuracy.

`source = ["lib/sas_3j1x_x.c", ...]` (sas_3j1x_x.c)

sas_2J1x_x(x): Bessel form sas_J1c$(x) = 2J_1(x)/x$, with a limiting value of 1 at $x = 0$, where $J_1(x)$ is the Bessel function of first kind and first order.

`source = ["lib/polevl.c", "lib/sas_J1.c", ...]` (sas_J1.c)

Gauss76Z[i], Gauss76Wt[i]: Points z_i and weights w_i for 76-point Gaussian quadrature, respectively, computing $\int_{-1}^1 f(z)\,dz \approx \sum_{i=1}^{76} w_i\,f(z_i)$.

Similar arrays are available in `gauss20.c` for 20-point quadrature and in `gauss150.c` for 150-point quadrature. The macros `GAUSS_N`, `GAUSS_Z` and `GAUSS_W` are defined so that you can change the order of the integration by selecting an different source without touching the C code.

`source = ["lib/gauss76.c", ...]` (gauss76.c)

Problems with C models

The graphics processor (GPU) in your computer is a specialized computer tuned for certain kinds of problems. This leads to strange restrictions that you need to be aware of. Your code may work fine on some platforms or for some models, but then return bad values on other platforms. Some examples of particular problems:

(1) Code is too complex, or uses too much memory. GPU devices only have a limited amount of memory available for each processor. If you run programs which take too much memory, then rather than running multiple values in parallel as it usually does, the GPU may only run a single version of the code at a time, making it slower than running on the CPU. It may fail to run on some platforms, or worse, cause the screen to go blank or the system to reboot.

(2) Code takes too long. Because GPU devices are used for the computer display, the OpenCL drivers are very careful about the amount of time they will allow any code to run. For example, on OS X, the model will stop running after 5 seconds regardless of whether the computation is complete. You may end up with only some of your 2D array defined, with the rest containing random data. Or it may cause the screen to go blank or the system to reboot.

(3) Memory is not aligned. The GPU hardware is specialized to operate on multiple values simultaneously. To keep the GPU simple the values in memory must be aligned with the different GPU compute engines. Not following these rules can lead to unexpected values being loaded into memory, and wrong answers computed. The conclusion from a very long and strange debugging session was that any arrays that you declare in your model should be a multiple of four. For example:

```
double Iq(q, p1, p2, ...)
{
    double vector[8];  // Only going to use seven slots, but declare 8
    ...
}
```

The first step when your model is behaving strangely is to set **single=False**. This automatically restricts the model to only run on the CPU, or on high-end GPU cards. There can still be problems even on high-end cards, so you can force the model off the GPU by setting **opencl=False**. This runs the model as a normal C program without any GPU restrictions so you know that strange results are probably from your code rather than the environment. Once the code is debugged, you can compare your output to the output on the GPU.

Although it can be difficult to get your model to work on the GPU, the reward can be a model that runs 1000x faster on a good card. Even your laptop may show a 50x improvement or more over the equivalent pure python model.

Form Factors

Away from the dilute limit you can estimate scattering including particle-particle interactions using $I(q) = P(q) * S(q)$ where $P(q)$ is the form factor and $S(q)$ is the structure factor. The simplest structure factor is the *hardsphere* interaction, which uses the effective radius of the form factor as an input to the structure factor model. The effective radius is the average radius of the form averaged over all the polydispersity values.

```
def ER(radius, thickness):
    """Effective radius of a core-shell sphere."""
    return radius + thickness
```

Now consider the *core_shell_sphere*, which has a simple effective radius equal to the radius of the core plus the thickness of the shell, as shown above. Given polydispersity over *(r1, r2, ..., rm)* in radius and *(t1, t2, ..., tn)* in thickness, *ER* is called with a mesh grid covering all possible combinations of radius and thickness. That is, *radius* is *(r1, r2, ..., rm, r1, r2, ..., rm, ...)* and *thickness* is *(t1, t1, ... t1, t2, t2, ..., t2, ...)*. The *ER* function returns one effective radius for each combination. The effective radius calculator weights each of these according to the polydispersity distributions and calls the structure factor with the average *ER*.

```
def VR(radius, thickness):
    """Sphere and shell volumes for a core-shell sphere."""
    whole = 4.0/3.0 * pi * (radius + thickness)**3
    core = 4.0/3.0 * pi * radius**3
    return whole, whole - core
```

Core-shell type models have an additional volume ratio which scales the structure factor. The *VR* function returns the volume of the whole sphere and the volume of the shell. Like *ER*, there is one return value for each point in the mesh grid.

NOTE: we may be removing or modifying this feature soon. As of the time of writing, core-shell sphere returns (1., 1.) for VR, giving a volume ratio of 1.0.

Unit Tests

THESE ARE VERY IMPORTANT. Include at least one test for each model and PLEASE make sure that the answer value is correct (i.e. not a random number).

```
tests = [
    [{}, 0.2, 0.726362],
    [{"scale": 1., "background": 0., "sld": 6., "sld_solvent": 1.,
      "radius": 120., "radius_pd": 0.2, "radius_pd_n":45},
     0.2, 0.228843],
    [{"radius": 120., "radius_pd": 0.2, "radius_pd_n":45}, "ER", 120.],
    [{"radius": 120., "radius_pd": 0.2, "radius_pd_n":45}, "VR", 1.],
]
```

tests=[[{parameters}, q, result], . . .] is a list of lists. Each list is one test and contains, in order:

- a dictionary of parameter values. This can be *{}* using the default parameters, or filled with some parameters that will be different from the default, such as *{"radius":10.0, "sld":4}*. Unlisted parameters will be given the default values.
- the input q value or tuple of (q_x, q_y) values.
- the output $I(q)$ or $I(q_x, q_y)$ expected of the model for the parameters and input value given.
- input and output values can themselves be lists if you have several q values to test for the same model parameters.
- for testing *ER* and *VR*, give the inputs as "ER" and "VR" respectively; the output for *VR* should be the sphere/shell ratio, not the individual sphere and shell values.

Test Your New Model

Minimal Testing

From SasView either open the Python shell (*Tools* > *Python Shell/Editor*) or the plugin editor (*Fitting* > *Plugin Model Operations* > *Advanced Plugin Editor*), load your model, and then select *Run* > *Check Model* from the menu bar. An *Info* box will appear with the results of the compilation and a check that the model runs.

If you are not using sasmodels from SasView, skip this step.

Recommended Testing

If the model compiles and runs, you can next run the unit tests that you have added using the **test =** values.

From SasView, switch to the *Shell* tab and type the following:

```
from sasmodels.model_test import run_one
run_one("~/.sasview/plugin_models/model.py")
```

This should print:

```
test_model_python (sasmodels.model_test.ModelTestCase) ... ok
```

To check whether single precision is good enough, type the following:

```
from sasmodels.compare import main as compare
compare("~/.sasview/plugin_models/model.py")
```

This will pop up a plot showing the difference between single precision and double precision on a range of q values.

```
demo = dict(scale=1, background=0,
            sld=6, sld_solvent=1,
            radius=120,
            radius_pd=.2, radius_pd_n=45)
```

demo={'par': value, ...} in the model file sets the default values for the comparison. You can include polydispersity parameters such as *radius_pd=0.2, radius_pd_n=45* which would otherwise be zero.

These commands can also be run directly in the python interpreter:

> $ python -m sasmodels.model_test -v ~/.sasview/plugin_models/model.py $ python -m sasmodels.compare ~/.sasview/plugin_models/model.py

The options to compare are quite extensive; type the following for help:

```
compare()
```

Options will need to be passed as separate strings. For example to run your model with a random set of parameters:

```
compare("-random", "-pars", "~/.sasview/plugin_models/model.py")
```

For the random models,

- *sld* will be in the range (-0.5,10.5),
- angles (*theta, phi, psi*) will be in the range (-180,180),
- angular dispersion will be in the range (0,45),
- polydispersity will be in the range (0,1)
- other values will be in the range (0, 2v), where v is the value of the parameter in demo.

Dispersion parameters n, *sigma* and *type* will be unchanged from demo so that run times are more predictable (polydispersity calculated across multiple parameters can be very slow).

If your model has 2D orientation calculation, then you should also test with:

```
compare("-2d", "~/.sasview/plugin_models/model.py")
```

Check The Docs

You can get a rough idea of how the documentation will look using the following:

```
compare("-help", "~/.sasview/plugin_models/model.py")
```

This does not use the same styling as the rest of the docs, but it will allow you to check that your ReStructuredText and LaTeX formatting. Here are some tools to help with the inevitable syntax errors:

- Sphinx cheat sheet
- Sphinx Documentation
- MathJax
- amsmath

There is also a neat online WYSIWYG ReStructuredText editor at http://rst.ninjs.org.

Clean Lint - (Developer Version Only)

NB: For now we are not providing pylint with the installer version of SasView; so unless you have a SasView build environment available, you can ignore this section!

Run the lint check with:

```
python -m pylint --rcfile=extra/pylint.rc ~/.sasview/plugin_models/model.py
```

We are not aiming for zero lint just yet, only keeping it to a minimum. For now, don't worry too much about *invalid-name*. If you really want a variable name *Rg* for example because R_g is the right name for the model parameter then ignore the lint errors. Also, ignore *missing-docstring* for standard model functions *Iq*, *Iqac*, etc.

We will have delinting sessions at the SasView Code Camps, where we can decide on standards for model files, parameter names, etc.

For now, you can tell pylint to ignore things. For example, to align your parameters in blocks:

```
# pylint: disable=bad-whitespace,line-too-long
#    ["name",                  "units", default, [lower, upper], "type",
↪"description"],
parameters = [
    ["contrast_factor",       "barns",    10.0,  [-inf, inf], "", "Contrast factor␣
↪of the polymer"],
    ["bjerrum_length",        "Ang",       7.1,  [0, inf],    "", "Bjerrum length
↪"],
    ["virial_param",          "1/Ang^2",  12.0,  [-inf, inf], "", "Virial parameter
↪"],
    ["monomer_length",        "Ang",      10.0,  [0, inf],    "", "Monomer length
↪"],
    ["salt_concentration",    "mol/L",     0.0,  [-inf, inf], "", "Concentration␣
↪of monovalent salt"],
    ["ionization_degree",     "",          0.05, [0, inf],    "", "Degree of␣
↪ionization"],
    ["polymer_concentration", "mol/L",     0.7,  [0, inf],    "", "Polymer molar␣
↪concentration"],
    ]
# pylint: enable=bad-whitespace,line-too-long
```

Don't put in too many pylint statements, though, since they make the code ugly.

Share Your Model!

Once compare and the unit test(s) pass properly and everything is done, consider adding your model to the Model Marketplace so that others may use it!

Document History

2016-10-25 Steve King
2017-05-07 Paul Kienzle - Moved from sasview to sasmodels docs

GPU Setup

SAS model evaluations can run on your graphics card (GPU) or they can run on the processor (CPU). In general, calculations performed on the GPU will run faster.

OpenCL Installation

Warning! GPU devices do not in general offer the same level of memory protection as CPU devices. If your code attempts to write outside allocated memory buffers unpredicatable behaviour may result (eg, your video display may freeze, or your system may crash, etc). Do not install OpenCL drivers without first checking for known issues (eg, some computer manufacturers install modified graphics drivers so replacing these may not be a good idea!). If in doubt, seek advice from an IT professional before proceeding further.

Check if you have OpenCL already installed

Windows

The following instructions are based on http://web.engr.oregonstate.edu/~mjb/cs475/DoIHaveOpenCL.pdf

- Go to: Start -> Control Panel -> System & Security -> Administrative Tools
- Double Click on Computer Managment
- Click on Device Manager
- Click open Display Adapters
- Right-click on available adapter and select Properties
- Click on Driver
- Go to Driver Details
- Scroll down and see if OpenCL is installed (look for OpenCL*.dll files)

Mac OSX

For OS X operating systems higher than 10.6 OpenCL is shipped along with the system.

However, OpenCL has had a rocky history on Macs. Apple provide a useful compatibility table at https://support.apple.com/en-us/HT202823

Installation

Windows

Depending on the graphic card in your system, drivers can be obtained from different sources:

- NVIDIA: https://developer.nvidia.com/opencl
- AMD: http://developer.amd.com/tools-and-sdks/opencl-zone/

Mac OSX

N/A

You cannot download OpenCL driver updates for your Mac. They are packaged with the normal quarterly OS X updates from Apple.

Note: Intel provides OpenCL drivers for Intel processors at https://software.intel.com/en-us/articles/opencl-drivers These can sometimes make use of special vector instructions across multiple processors, so it is worth installing if the GPU does not support double precision. You can install this driver alongside the GPU driver for NVIDIA or AMD.

GPU Selection

The logic for choosing the compute platform is a little bit complicated. If the model has the line *single=False* then it requires double precision. If the GPU is single precision only, then it will try running via OpenCL on the CPU. If the OpenCL driver is not available for the CPU then it will run as a normal program on the CPU.

For models with a large number of parameters or with a lot of code, the GPU may be too small to run the program effectively. In this case, you should try simplifying the model, maybe breaking it into several different models so that you don't need *IF* statements in your code. If it is still too big, you can set *opencl=False* in the model file and the model will only run as a normal program on the CPU. This will not usually be necessary.

Device Selection

If you have multiple GPU devices you can tell the program which device to use. By default, the program looks for one GPU and one CPU device from available OpenCL platforms. It prefers AMD or NVIDIA drivers for GPU, and prefers Intel or Apple drivers for CPU. Both GPU and CPU are included on the assumption that CPU is always available and supports double precision.

The device order is important: GPU is checked before CPU on the assumption that it will be faster. By examining ~/sasview.log you can see which device was used to run the model.

If you don't want to use OpenCL, you can set *SAS_OPENCL=None* in your environment settings, and it will only use normal programs.

If you want to use one of the other devices, you can run the following from the python console:

```
import pyopencl as cl
cl.create_some_context()
```

This will provide a menu of different OpenCL drivers available. When one is selected, it will say "set PYOPENCL_CTX=…" Use that value as the value of *SAS_OPENCL*.

Device Testing

Unfortunately, not all vendors provide working OpenCL implementations for their GPU devices. For example, the HD 6000 Intel GPUs with double precision support fail for some of the double precision models.

The SasView user interface provides a Fitting OpenCL Options dialog for selecting amongst and testing the available devices. After a few minutes of seeming to freeze, the application will return a list of model tests which have passed. The same tests can be run directly from the python console using:

```
from sasmodels.model_tests import main as model_tests
model_tests("-v", "opencl", "all")
```

Compiler Selection

For models run as normal programs, you may need to specify a compiler. This is done using the *SAS_COMPILER* environment variable, and the *SAS_OPENMP* environment variable if OpenMP support is available for the compiler.

On Windows, set *SAS_COMPILER=tinycc* for the tinycc compiler, *SAS_COMPILER=msvc* for the Microsoft Visual C compiler, or *SAS_COMPILER=mingw* for the MinGW compiler. If TinyCC is available on the python path (it is provided with SasView), that will be the default. If you want one of the other compilers, be sure to have it available in your *PATH* so we can find it!

On Mac OS/X and Linux, set *SAS_COMPILER=unix* for the compiler. This will use the unix cc command to compile the model, with gcc style command line options. For OS/X you will need to install the Xcode package to make the compiler available.

Document History

2017-09-27 Paul Kienzle

Scripting Interface

Need some basic details here of how to load models and data via script, evaluate them at given parameter values and run bumps fits.

The key functions are `sasmodels.core.load_model()` for loading the model definition and compiling the kernel and `sasmodels.data.load_data()` for calling sasview to load the data.

Preparing data

Usually you will load data via the sasview loader, with the `sasmodels.data.load_data()` function. For example:

```
from sasmodels.data import load_data
data = load_data("sasmodels/example/093191_201.dat")
```

You may want to apply a data mask, such a beam stop, and trim high q:

```
from sasmodels.data import set_beam_stop
set_beam_stop(data, qmin, qmax)
```

The `sasmodels.data.set_beam_stop()` method simply sets the *mask* attribute for the data.

The data defines the resolution function and the q values to evaluate, so even if you simulating experiments prior to making measurements, you still need a data object for reference. Use `sasmodels.data.empty_data1D()` or `sasmodels.data.empty_data2D()` to create a container with a given q and $\Delta q/q$. For example:

```
import numpy as np
from sasmodels.data import empty_data1D

# 120 points logarithmically spaced from 0.005 to 0.2, with dq/q = 5%
q = np.logspace(np.log10(5e-3), np.log10(2e-1), 120)
data = empty_data1D(q, resolution=0.05)
```

To use a more realistic model of resolution, or to load data from a file format not understood by SasView, you can use `sasmodels.data.Data1D` or `sasmodels.data.Data2D` directly. The 1D data uses *x*, *y*, *dx* and *dy* for $x = q$ and $y = I(q)$, and 2D data uses *x*, *y*, *z*, *dx*, *dy*, *dz* for $x, y = qx, qy$ and $z = I(qx, qy)$. [Note: internally, the Data2D object uses SasView conventions, *qx_data*, *qy_data*, *data*, *dqx_data*, *dqy_data*, and *err_data*.]

For USANS data, use 1D data, but set *dxl* and *dxw* attributes to indicate slit resolution:

```
data.dxl = 0.117
```

See `sasmodels.resolution.slit_resolution()` for details.

SESANS data is more complicated; if your SESANS format is not supported by SasView you need to define a number of attributes beyond *x*, *y*. For example:

```
SElength = np.linspace(0, 2400, 61) # [A]
data = np.ones_like(SElength)
err_data = np.ones_like(SElength)*0.03

class Source:
    wavelength = 6 # [A]
    wavelength_unit = "A"
```

(continues on next page)

(continued from previous page)

```
class Sample:
    zacceptance = 0.1 # [A^-1]
    thickness = 0.2 # [cm]

class SESANSData1D:
    #q_zmax = 0.23 # [A^-1]
    lam = 0.2 # [nm]
    x = SElength
    y = data
    dy = err_data
    sample = Sample()
data = SESANSData1D()

x, y = ... # create or load sesans
data = smd.Data
```

The *data* module defines various data plotters as well.

Using sasmodels directly

Once you have a computational kernel and a data object, you can evaluate the model for various parameters using `sasmodels.direct_model.DirectModel`. The resulting object f will be callable as *f(par=value, . . .)*, returning the $I(q)$ for the q values in the data. For example:

```
import numpy as np
from sasmodels.data import empty_data1D
from sasmodels.core import load_model
from sasmodels.direct_model import DirectModel

# 120 points logarithmically spaced from 0.005 to 0.2, with dq/q = 5%
q = np.logspace(np.log10(5e-3), np.log10(2e-1), 120)
data = empty_data1D(q, resolution=0.05)
kernel = load_model("ellipsoid)
f = DirectModel(data, kernel)
Iq = f(radius_polar=100)
```

Polydispersity information is set with special parameter names:

- *par_pd* for polydispersity width, $\Delta p/p$,
- *par_pd_n* for the number of points in the distribution,
- *par_pd_type* for the distribution type (as a string), and
- *par_pd_nsigmas* for the limits of the distribution.

Using sasmodels through the bumps optimizer

Like DirectModel, you can wrap data and a kernel in a *bumps* model with class:*sasmodels.bumps_model.Model* and create an class:*sasmodels.bumps_model.Experiment* that you can fit with the *bumps* interface. Here is an example from the *example* directory such as *example/model.py*:

```
import sys
from bumps.names import *
from sasmodels.core import load_model
from sasmodels.bumps_model import Model, Experiment
from sasmodels.data import load_data, set_beam_stop, set_top

""" IMPORT THE DATA USED """
```

(continues on next page)

(continued from previous page)

```
radial_data = load_data('DEC07267.DAT')
set_beam_stop(radial_data, 0.00669, outer=0.025)
set_top(radial_data, -.0185)

kernel = load_model("ellipsoid")

model = Model(kernel,
    scale=0.08,
    radius_polar=15, radius_equatorial=800,
    sld=.291, sld_solvent=7.105,
    background=0,
    theta=90, phi=0,
    theta_pd=15, theta_pd_n=40, theta_pd_nsigma=3,
    radius_polar_pd=0.222296, radius_polar_pd_n=1, radius_polar_pd_nsigma=0,
    radius_equatorial_pd=.000128, radius_equatorial_pd_n=1, radius_equatorial_pd_
↪nsigma=0,
    phi_pd=0, phi_pd_n=20, phi_pd_nsigma=3,
    )

# SET THE FITTING PARAMETERS
model.radius_polar.range(15, 1000)
model.radius_equatorial.range(15, 1000)
model.theta_pd.range(0, 360)
model.background.range(0,1000)
model.scale.range(0, 10)

#cutoff = 0     # no cutoff on polydisperisity loops
#cutoff = 1e-5  # default cutoff
cutoff = 1e-3  # low precision cutoff
M = Experiment(data=radial_data, model=model, cutoff=cutoff)
problem = FitProblem(M)
```

Assume that bumps has been installed and the bumps command is available. Maybe need to set the path to sasmodels/sasview using *PYTHONPATH=path/to/sasmodels:path/to/sasview/src*. To run the model use the *bumps* program:

```
$ bumps example/model.py --preview
```

Note that bumps and sasmodels are included as part of the SasView distribution. On windows, bumps can be called from the cmd prompt as follows:

```
SasViewCom bumps.cli example/model.py --preview
```

Calling the computation kernel

Getting a simple function that you can call on a set of q values and return a result is not so simple. Since the time critical use case (fitting) involves calling the function over and over with identical q values, we chose to optimize the call by only transfering the q values to the GPU once at the start of the fit. We do this by creating a `sasmodels.kernel.Kernel` object from the `sasmodels.kernel.KernelModel` returned from `sasmodels.core.load_model()` using the `sasmodels.kernel.KernelModel.make_kernel()` method. What it actual does depends on whether it is running as a DLL, as OpenCL or as a pure python kernel. Once the kernel is in hand, we can then marshal a set of parameters into a `sasmodels.details.CallDetails` object and ship it to the kernel using the `sansmodels.direct_model.call_kernel()` function. An example should help, *example/cylinder_eval.py*:

```
from numpy import logspace
from matplotlib import pyplot as plt
from sasmodels.core import load_model
```

(continues on next page)

(continued from previous page)

```
from sasmodels.direct_model import call_kernel

model = load_model('cylinder')
q = logspace(-3, -1, 200)
kernel = model.make_kernel([q])
Iq = call_kernel(kernel, dict(radius=200.))
plt.loglog(q, Iq)
plt.show()
```

On windows, this can be called from the cmd prompt using sasview as:

```
SasViewCom example/cylinder_eval.py
```

References

Small-Angle Scattering of X-Rays A Guinier and G Fournet John Wiley & Sons, New York (1955)

P Stckel, R May, I Strell, Z Cejka, W Hoppe, H Heumann, W Zillig and H Crespi *Eur. J. Biochem.*, 112, (1980), 411-417

G Porod in *Small Angle X-ray Scattering* (editors) O Glatter and O Kratky Academic Press (1982)

Structure Analysis by Small-Angle X-Ray and Neutron Scattering L.A Feigin and D I Svergun Plenum Press, New York (1987)

S Hansen *J. Appl. Cryst.* 23, (1990), 344-346

S J Henderson *Biophys. J.* 70, (1996), 1618-1627

B C McAlister and B P Grady *J. Appl. Cryst.* 31, (1998), 594-599

S R Kline *J Appl. Cryst.* 39(6), (2006), 895

Also see the references at the end of the each model function descriptions.

1.3.2 P(r) Calculation

Description

This tool calculates a real-space distance distribution function, *P(r)*, using the inversion approach (Moore, 1980).

P(r) is set to be equal to an expansion of base functions of the type

$$\Phi_{n(r)} = 2r sin(\frac{\pi n r}{D_{max}})$$

The coefficient of each base function in the expansion is found by performing a least square fit with the following fit function

$$\chi^2 = \frac{\sum_i (I_{meas}(Q_i) - I_{th}(Q_i))^2}{error^2} + Reg\_term$$

where $I_{meas}(Q_i)$ is the measured scattering intensity and $I_{th}(Q_i)$ is the prediction from the Fourier transform of the *P(r)* expansion.

The Reg_term term is a regularization term set to the second derivative $d^2P(r)/d^2r$ integrated over r. It is used to produce a smooth *P(r)* output.

Using P(r) inversion

The user must enter

- *Number of terms*: the number of base functions in the P(r) expansion.
- *Regularization constant*: a multiplicative constant to set the size of the regularization term.
- *Maximum distance*: the maximum distance between any two points in the system.

P(r) inversion requires that the background be perfectly subtracted. This is often difficult to do well and thus many data sets will include a background. For those cases, the user should check the “Estimate background level” option and the module will do its best to estimate it. If you know the background value for your data, select the “Input manual background level” option. Note that this value will be treated as having 0 error.

The P(r) module is constantly computing in the background what the optimum *number of terms* should be as well as the optimum *regularization constant*. These are constantly updated in the buttons next to the entry boxes on the GUI. These are almost always close and unless the user has a good reason to choose differently they should just click on the buttons to accept both. {D_max} must still be set by the user. However, besides looking at the output, the user can click the explore button which will bring up a graph of chi^2 vs Dmax over a range around the current Dmax. The user can change the range and the number of points to explore in that range. They can also choose to plot several other parameters as a function of Dmax including: I0, Rg, Oscillation parameter, background, positive fraction, and 1-sigma positive fraction.

Reference

P.B. Moore *J. Appl. Cryst.*, 13 (1980) 168-175

Note: This help document was last modified by Paul Butler, 05 September, 2016

1.3.3 Invariant Calculation

Description

The scattering, or Porod, invariant (Q^*) is a model-independent quantity that can be easily calculated from scattering data.

For two phase systems, the scattering invariant is defined as the integral of the square of the wavevector transfer (Q) multiplied by the scattering cross section over the full range of Q from zero to infinity, that is

$$Q^* = \int_0^\infty q^2 I(q)\, dq$$

in the case of pinhole geometry. For slit geometry the invariant is given by

$$Q^* = \Delta q_v \int_0^\infty q I(q)\, dq$$

where Δq_v is the slit height.

The worth of Q^* is that it can be used to determine the volume fraction and the specific area of a sample. Whilst these quantities are useful in their own right they can also be used in further analysis.

The difficulty with using Q^* arises from the fact that experimental data is never measured over the range $0 \leq Q \leq \infty$. At best, combining USAS and WAS data might cover the range $10^{-5} \leq Q \leq 10$ 1/Å . Thus it is usually necessary to extrapolate the experimental data to low and high Q. For this

High-Q region (>= *Qmax* in data)

- The power law function C/Q^4 is used where the constant $C = 2\pi \Delta\rho S_v$ is to be found by fitting part of data within the range Q_{N-m} to Q_N (where $m < N$).

Low-Q region (<= *Qmin* in data)

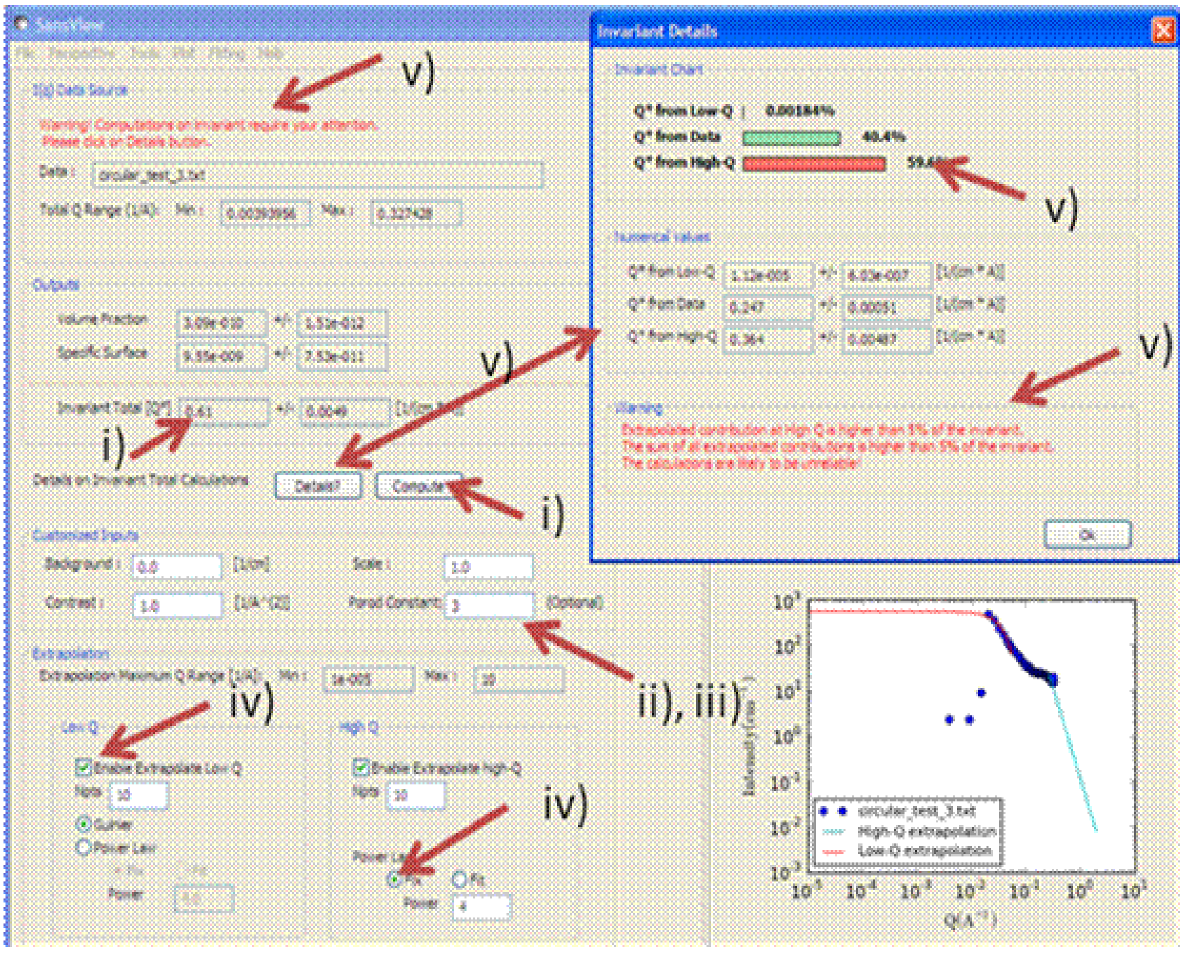
- The Guinier function $I_0 exp(-R_g^2 Q^2/3)$ where I_0 and R_g are obtained by fitting as for the high-Q region above. Alternatively a power law can be used.

Using invariant analysis

1. Select *Invariant* from the *Analysis* menu on the SasView toolbar.
2. Load some data with the *Data Explorer*.
3. Select a dataset and use the *Send To* button on the *Data Explorer* to load the dataset into the *Invariant* panel.
4. Use the *Customised Input* boxes on the *Invariant* panel to subtract any background, specify the contrast (i.e. difference in SLDs - this must be specified for the eventual value of Q^* to be on an absolute scale), or to rescale the data.
5. Adjust the extrapolation range as necessary. In most cases the default values will suffice.
6. Click the *Compute* button.
7. To include a lower and/or higher Q range, check the relevant *Enable Extrapolate* check boxes.

   If power law extrapolations are chosen, the exponent can be either held fixed or fitted. The number of points, Npts, to be used for the basis of the extrapolation can also be specified.
8. If the value of Q^* calculated with the extrapolated regions is invalid, a red warning will appear at the top of the *Invariant* panel.

   The details of the calculation are available by clicking the *Details* button in the middle of the panel.



Parameters

Volume Fraction

The volume fraction ϕ is related to Q^* by

$$\phi(1-\phi) = \frac{Q^*}{2\pi^2(\Delta\rho)^2} \equiv A$$

where $\Delta\rho$ is the SLD contrast.

$$\phi = \frac{1 \pm \sqrt{1-4A}}{2}$$

Specific Surface Area

The specific surface area S_v is related to Q^* by

$$S_v = \frac{2\pi\phi(1-\phi)C_p}{Q^*} = \frac{2\pi A C_p}{Q^*}$$

where C_p is the Porod constant.

Reference

O. Glatter and O. Kratky Chapter 2 in *Small Angle X-Ray Scattering* Academic Press, New York, 1982

http://web.archive.org/web/20110824105537/http://physchem.kfunigraz.ac.at/sm/Service/Glatter_Kratky_SAXS_1982.zip

Note: This help document was last changed by Steve King, 01May2015

1.3.4 Correlation Function Analysis

Description

This currently performs correlation function analysis on SAXS/SANS data, but in the the future is also planned to generate model-independent volume fraction profiles from the SANS from adsorbed polymer/surfactant layers. The two types of analyses differ in the mathematical transform that is applied to the data (Fourier vs Hilbert). However, both functions are returned in *real space*.

A correlation function may be interpreted in terms of an imaginary rod moving through the structure of the material. Γ(x) is the probability that a rod of length x has equal electron/neutron scattering length density at either end. Hence a frequently occurring spacing within a structure will manifest itself as a peak in Γ(x). *SasView* will return both the one-dimensional (Γ_1(x)) and three-dimensional (Γ_3(x)) correlation functions, the difference being that the former is only averaged in the plane of the scattering vector.

A volume fraction profile Φ(z) describes how the density of polymer segments/surfactant molecules varies with distance, z, normal to an (assumed locally flat) interface. The form of Φ(z) can provide information about the arrangement of polymer/surfactant molecules at the interface. The width of the profile provides measures of the layer thickness, and the area under the profile is related to the amount of material that is adsorbed.

Both analyses are performed in 3 stages:

- Extrapolation of the scattering curve to $q = 0$ and toward $q = \infty$
- Smoothed merging of the two extrapolations into the original data

- Fourier / Hilbert Transform of the smoothed data to give the correlation function or volume fraction profile, respectively
- (Optional) Interpretation of Γ_1(x) assuming the sample conforms to an ideal lamellar morphology

Extrapolation

To $q = 0$

The data are extrapolated to q = 0 by fitting a Guinier function to the data points in the low-q range.

The equation used is:

$$I(q) = Ae^{Bq^2}$$

Where the parameter B is related to the effective radius-of-gyration of a spherical object having the same small-angle scattering in this region.

Note that as q tends to zero this function tends to a limiting value and is therefore less appropriate for use in systems where the form factor does not do likewise. However, because of the transform, the correlation functions are most affected by the Guinier back-extrapolation at *large* values of x where the impact on any extrapolated parameters will be least significant.

To $q = \infty$

The data are extrapolated towards q = ∞ by fitting a Porod model to the data points in the high-q range and then computing the extrapolation to 100 times the maximum q value in the experimental dataset. This should be more than sufficient to ensure that on transformation any truncation artefacts introduced are at such small values of x that they can be safely ignored.

The equation used is:

$$I(q) = Kq^{-4}e^{-q^2\sigma^2} + Bg$$

Where Bg is the background, K is the Porod constant, and σ (which must be > 0) describes the width of the electron/neutron scattering length density profile at the interface between the crystalline and amorphous regions as shown below.

Smoothing

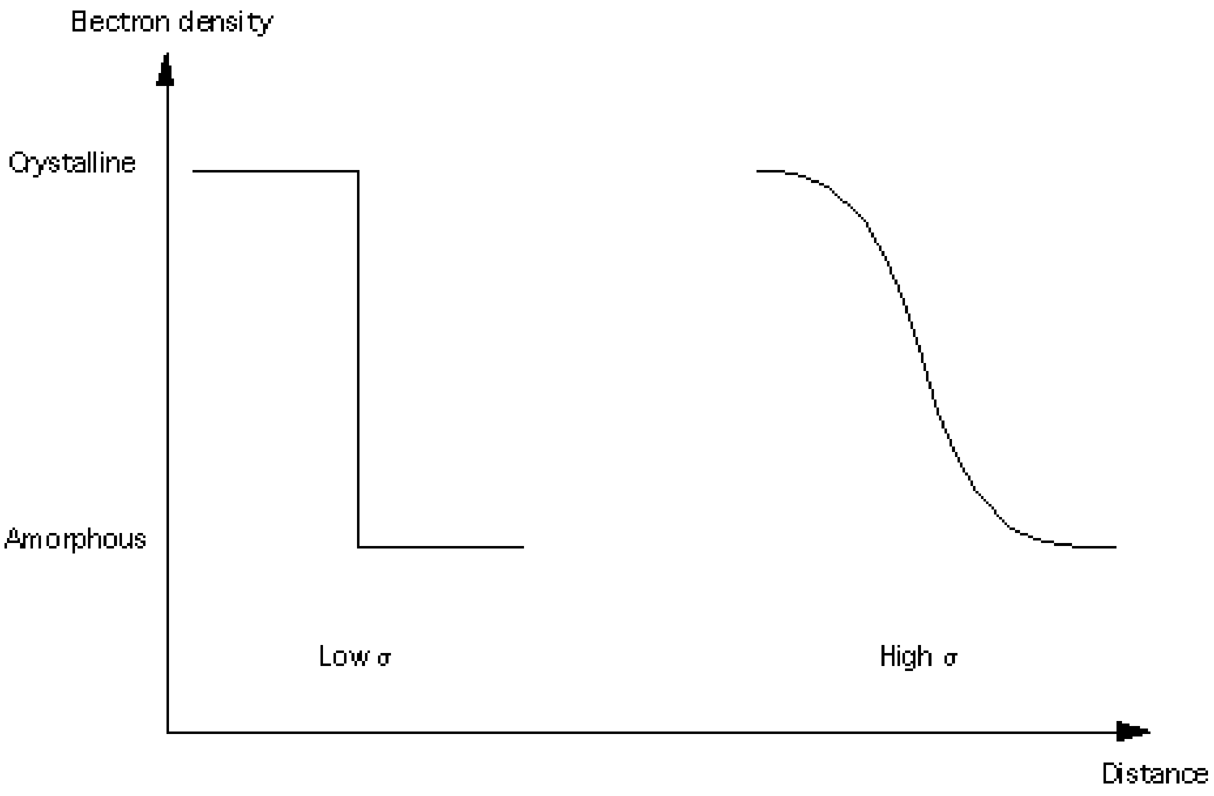
The extrapolated data set consists of the Guinier back-extrapolation from q ~ 0 up to the lowest q value in the original data, then the original scattering data, and then the Porod tail-fit beyond this. The joins between the original data and the Guinier/Porod extrapolations are smoothed using the algorithm below to try and avoid the formation of truncation ripples in the transformed data:

Functions $f(x_i)$ and $g(x_i)$ where $x_i \in \{x_1, x_2, ..., x_n\}$, are smoothed over the range $[a, b]$ to produce $y(x_i)$, by the following equations:

$$y(x_i) = h_i g(x_i) + (1 - h_i) f(x_i)$$

where:

$$h_i = \frac{1}{1 + \frac{(x_i-b)^2}{(x_i-a)^2}}$$



Transformation

Fourier

If "Fourier" is selected for the transform type, *SasView* will perform a discrete cosine transform on the extrapolated data in order to calculate the 1D correlation function as:

$$\Gamma_1(x) = \frac{1}{Q^*} \int_0^{\infty} I(q) q^2 cos(qx) dq$$

where Q^* is the Scattering (also called Porod) Invariant.

The following algorithm is applied:

$$\Gamma(x_k) = 2 \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right) k\right] \text{ for } k = 0, 1, \ldots, N-1, N$$

The 3D correlation function is calculated as:

$$\Gamma_3(x) = \frac{1}{Q^*} \int_0^{\infty} I(q) q^2 \frac{sin(qx)}{qx} dq$$

Note: It is always advisable to inspect Γ_1(x) and Γ_3(x) for artefacts arising from the extrapolation and transformation processes:

- do they tend to zero as x tends to ∞?
- do they smoothly curve onto the ordinate at x = 0? (if not check the value of σ is sensible)
- are there ripples at x values corresponding to (2 π over) the two q values at which the extrapolated and experimental data are merged?
- are there any artefacts at x values corresponding to 2 π / q_{max} in the experimental data?
- and lastly, do the significant features/peaks in the correlation functions actually correspond to anticpated spacings in the sample?!!!

Finally, the program calculates the interface distribution function (IDF) $g_1(x)$ as the discrete cosine transform of:

$$-q^4 I(q)$$

The IDF is proportional to the second derivative of $\Gamma_1(x)$ and represents a superposition of thickness distributions from all the contributing lamellae.

Hilbert

If "Hilbert" is selected for the transform type, the analysis will perform a Hilbert transform on the extrapolated data in order to calculate the Volume Fraction Profile.

Note: The Hilbert transform functionality is not yet implemented in SasView.

Interpretation

Correlation Function

Once the correlation functions have been calculated *SasView* can be asked to try and interpret $\Gamma_1(x)$ in terms of an ideal lamellar morphology as shown below.

The structural parameters extracted are:

- Long Period $= L_p$
- Average Hard Block Thickness $= L_c$
- Average Core Thickness $= D_0$
- Average Interface Thickness $= D_{tr}$
- Polydispersity $= \Gamma_{\text{min}}/\Gamma_{\text{max}}$
- Local Crystallinity $= L_c/L_p$

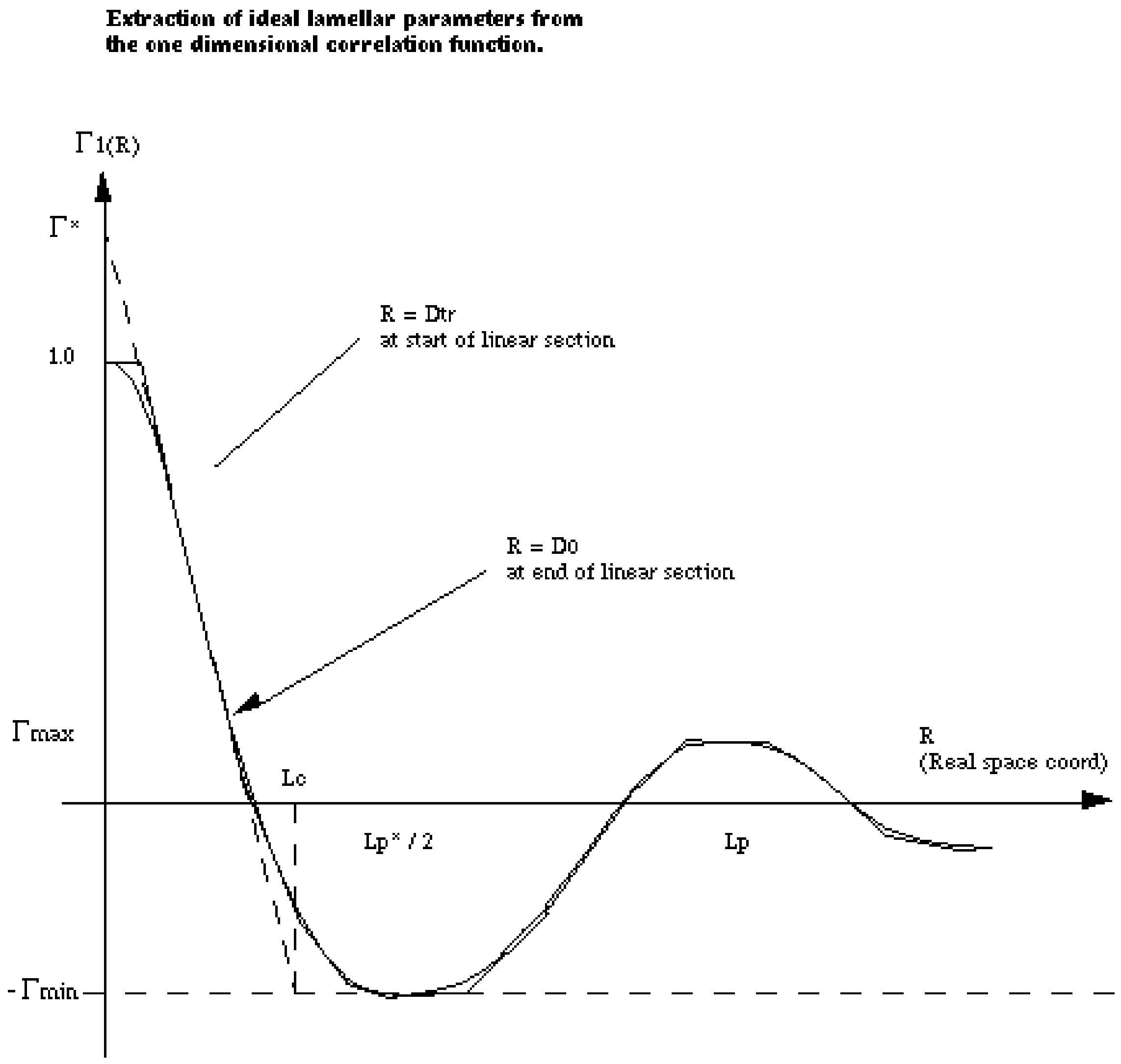
Warning: If the sample does not possess lamellar morphology then "Compute Parameters" will return garbage!

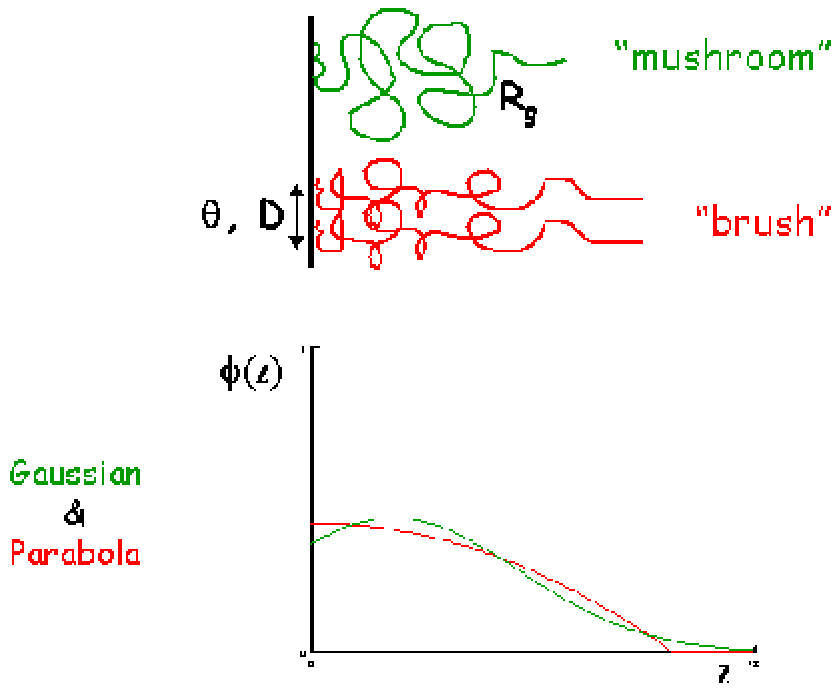
Volume Fraction Profile

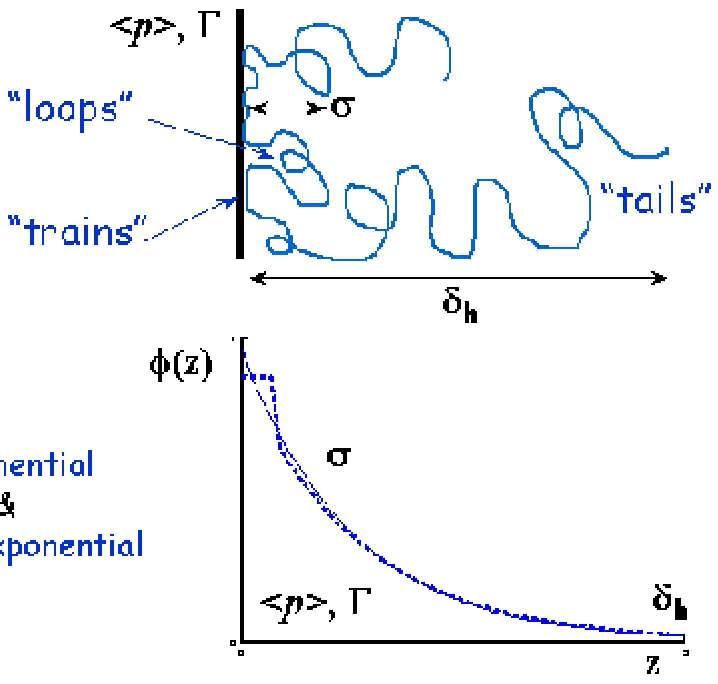
SasView does not provide any automatic interpretation of volume fraction profiles in the same way that it does for correlation functions. However, a number of structural parameters are obtainable by other means:

- Surface Coverage $= \theta$
- Anchor Separation $= D$
- Bound Fraction $= < p >$
- Second Moment $= \sigma$
- Maximum Extent $= \delta_{\text{h}}$
- Adsorbed Amount $= \Gamma$

The reader is directed to the references for information on these parameters.


Extraction of ideal lamellar parameters from
the one dimensional correlation function.
Γ1(R)
Γ*
1.0
R = Dtr
at start of linear section
R = D0
at end of linear section
Γmax
Lc
R
(Real space coord)
Lp* / 2
Lp
- Γmin


"mushroom"
Rg
θ, D
"brush"
φ(z)
Gaussian
&
Parabola
z


<p>, Γ
"loops"
σ
"trains"
"tails"
δh
φ(z)
Exponential
&
Block+Exponential
σ
<p>, Γ
δh
z

References

Correlation Function

Strobl, G. R.; Schneider, M. *J. Polym. Sci.* (1980), 18, 1343-1359

Koberstein, J.; Stein R. *J. Polym. Sci. Phys. Ed.* (1983), 21, 2181-2200

Baltá Calleja, F. J.; Vonk, C. G. *X-ray Scattering of Synthetic Poylmers*, Elsevier. Amsterdam (1989), 247-251

Baltá Calleja, F. J.; Vonk, C. G. *X-ray Scattering of Synthetic Poylmers*, Elsevier. Amsterdam (1989), 257-261

Baltá Calleja, F. J.; Vonk, C. G. *X-ray Scattering of Synthetic Poylmers*, Elsevier. Amsterdam (1989), 260-270

Göschel, U.; Urban, G. *Polymer* (1995), 36, 3633-3639

Stribeck, N. *X-Ray Scattering of Soft Matter*, Springer. Berlin (2007), 138-161

FDR (PDF format)

Volume Fraction Profile

Washington, C.; King, S. M. *J. Phys. Chem.*, (1996), 100, 7603-7609

Cosgrove, T.; King, S. M.; Griffiths, P. C. *Colloid-Polymer Interactions: From Fundamentals to Practice*, Wiley. New York (1999), 193-204

King, S. M.; Griffiths, P. C.; Cosgrove, T. *Applications of Neutron Scattering to Soft Condensed Matter*, Gordon & Breach. Amsterdam (2000), 77-105

King, S.; Griffiths, P.; Hone, J.; Cosgrove, T. *Macromol. Symp.* (2002), 190, 33-42

Usage

Upon sending data for correlation function analysis, it will be plotted (minus the background value), along with a *red* bar indicating the *upper end of the low-Q range* (used for Guinier back-extrapolation), and 2 *purple* bars indicating the range to be used for Porod forward-extrapolation. These bars may be moved by grabbing and dragging, or by entering appropriate values in the Q range input boxes.

Once the Q ranges have been set, click the "Calculate Bg" button to determine the background level. Alternatively, enter your own value into the box. If the box turns yellow this indicates that background subtraction has created some negative intensities. This may still be fine provided the peak intensity is very much greater than the background level. The important point is that the extrapolated dataset must approach zero at high-q.

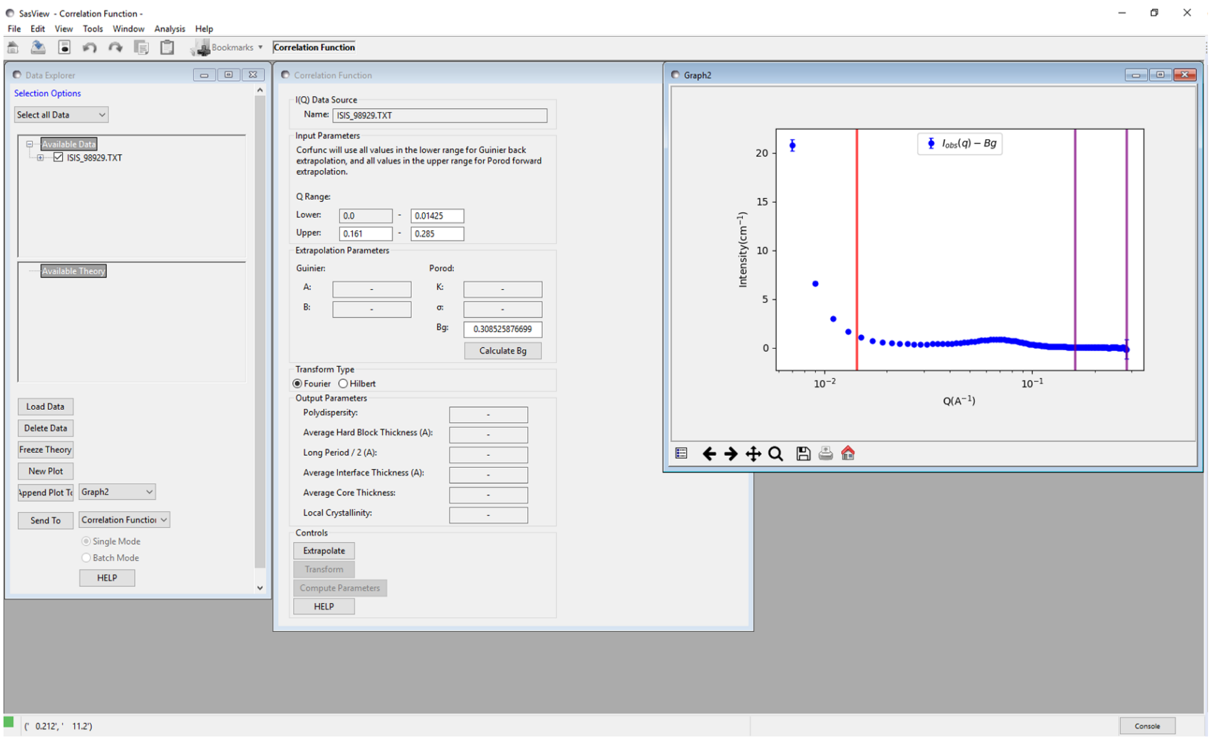
Now click the "Extrapolate" button to extrapolate the data. The graph window will update to show the extrapolated data, and the values of the parameters used for the Guinier and Porod extrapolations will appear in the "Extrapolation Parameters" section of the SasView GUI.

Now select which type of transform you would like to perform, using the radio buttons:

- **Fourier**: to perform a Fourier Transform to calculate the correlation functions
- **Hilbert**: to perform a Hilbert Transform to calculate the volume fraction profile

and click the "Transform" button to perform the selected transform and plot the results.

If a Fourier Transform was performed, the "Compute Parameters" button can now be clicked to interpret the correlation function as described earlier. The parameters will appear in the "Output Parameters" section of the SasView GUI.



Correlation Function

I(Q) Data Source

Name: ISIS_98929.TXT

Input Parameters

Corfunc will use all values in the lower range for Guinier back extrapolation, and all values in the upper range for Porod forward extrapolation.

Q Range:

Lower: 0.0 - 0.01425

Upper: 0.161 - 0.285

Extrapolation Parameters

Guinier: A: - B: -

Porod: K: - σ: - Bg: 0.308525876699

Calculate Bg

Transform Type: (●) Fourier ( ) Hilbert

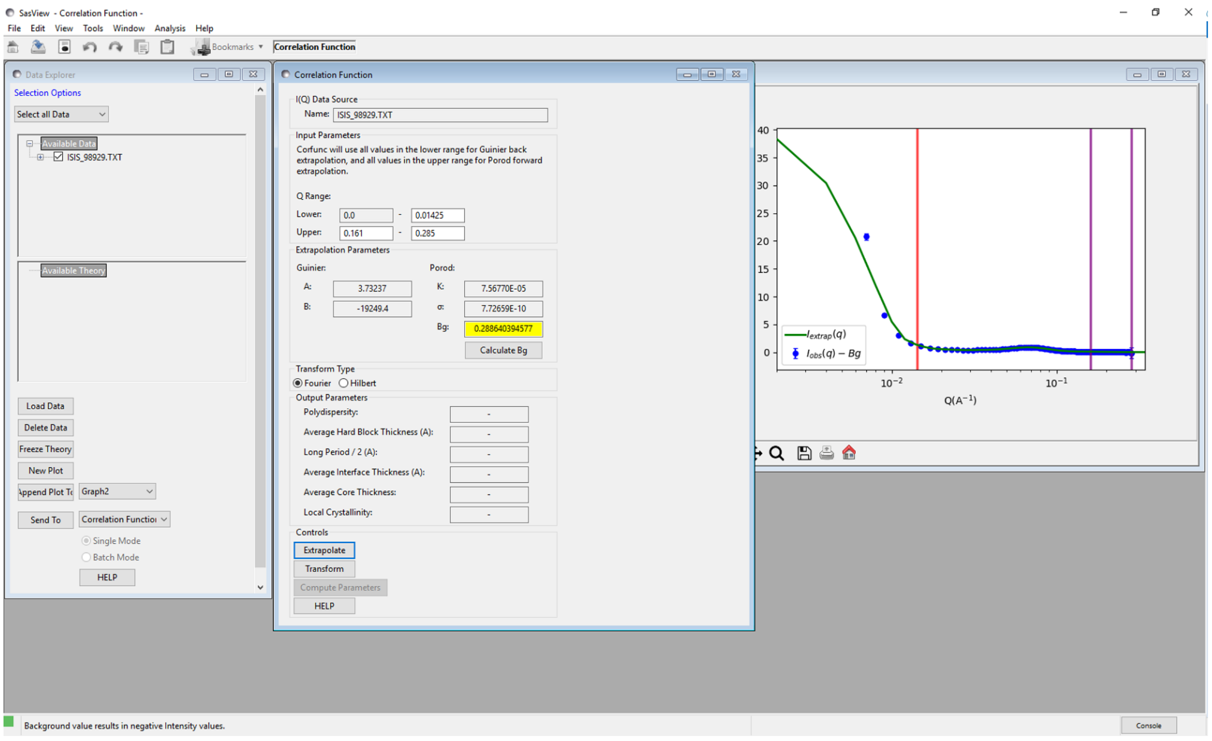
Output Parameters

Polydispersity:	-
Average Hard Block Thickness (A):	-
Long Period / 2 (A):	-
Average Interface Thickness (A):	-
Average Core Thickness:	-
Local Crystallinity:	-

Controls: Extrapolate, Transform, Compute Parameters, HELP

Graph2 send to: Graph2, Correlation Function

('0.212', '11.2')



Correlation Function

I(Q) Data Source

Name: ISIS_98929.TXT

Input Parameters

Corfunc will use all values in the lower range for Guinier back extrapolation, and all values in the upper range for Porod forward extrapolation.

Q Range:

Lower: 0.0 - 0.01425

Upper: 0.161 - 0.285

Extrapolation Parameters

Guinier: A: 3.73237 B: -19249.4

Porod: K: 7.56770E-05 σ: 7.72659E-10 Bg: 0.288640394577

Calculate Bg

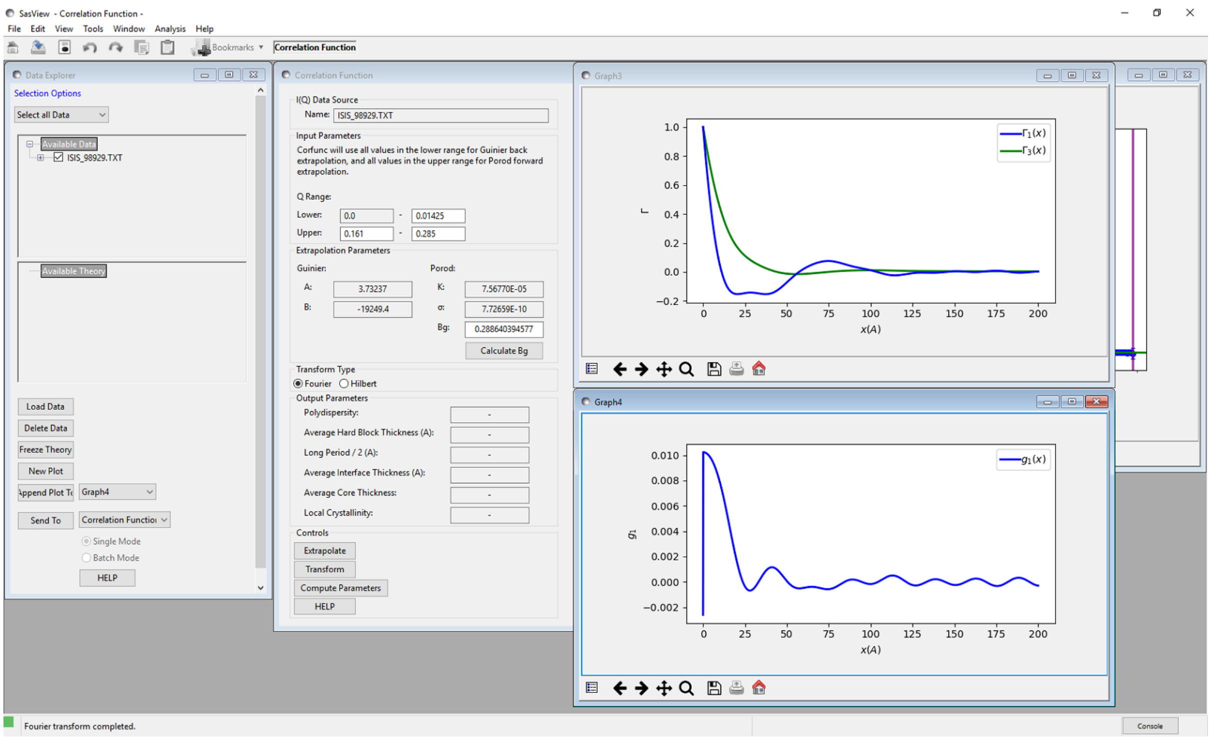
Transform Type: (●) Fourier ( ) Hilbert

Output Parameters

Polydispersity:	-
Average Hard Block Thickness (A):	-
Long Period / 2 (A):	-
Average Interface Thickness (A):	-
Average Core Thickness:	-
Local Crystallinity:	-

Controls: Extrapolate, Transform, Compute Parameters, HELP

Background value results in negative Intensity values.



Correlation Function

I(Q) Data Source

Name: ISIS_98929.TXT

Input Parameters

Corfunc will use all values in the lower range for Guinier back extrapolation, and all values in the upper range for Porod forward extrapolation.

Q Range:

Lower: 0.0 - 0.01425

Upper: 0.161 - 0.285

Extrapolation Parameters

Guinier:		Porod:	
A:	3.73237	K:	7.56770E-05
B:	-19249.4	σ:	7.72659E-10
		Bg:	0.288640394577

Transform Type

[✓ Fourier] Hilbert

Output Parameters

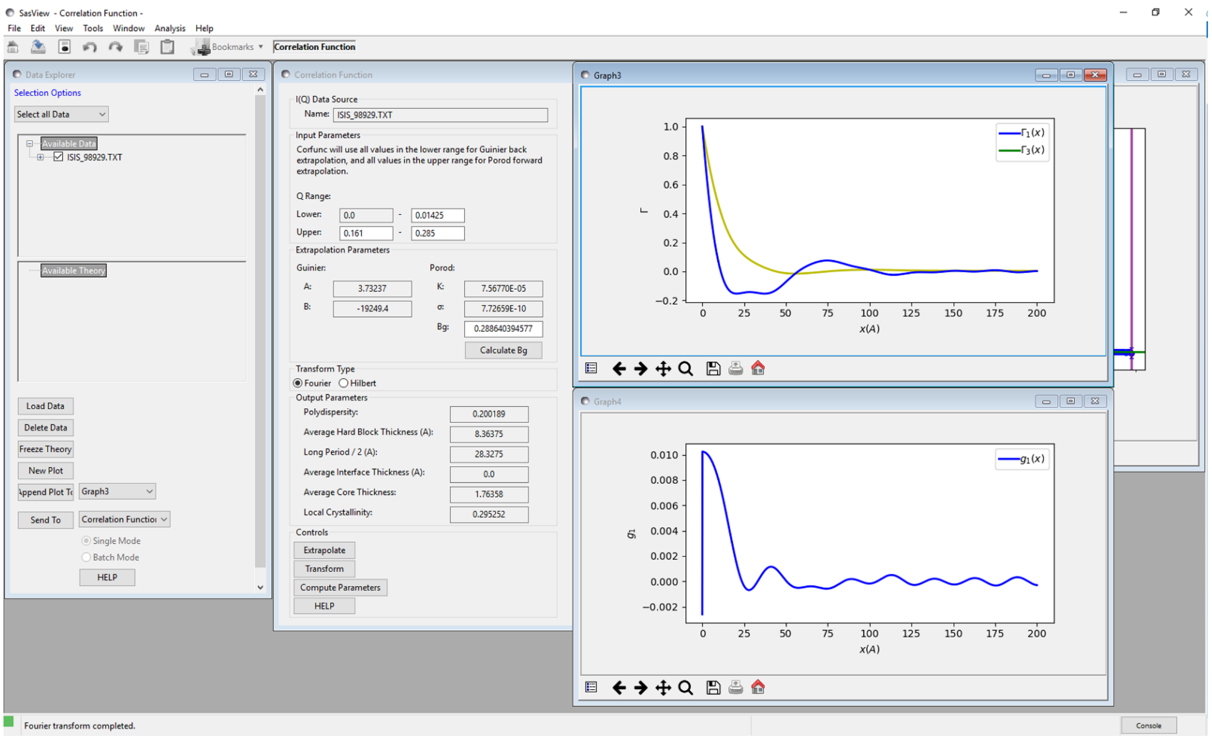
Polydispersity:	-
Average Hard Block Thickness (A):	-
Long Period / 2 (A):	-
Average Interface Thickness (A):	-
Average Core Thickness:	-
Local Crystallinity:	-

ISIS_98929.TXT

Append Plot To: Graph4

Send To: Correlation Function

Fourier transform completed.



Correlation Function

I(Q) Data Source

Name: ISIS_98929.TXT

Input Parameters

Corfunc will use all values in the lower range for Guinier back extrapolation, and all values in the upper range for Porod forward extrapolation.

Q Range:

Lower: 0.0 - 0.01425

Upper: 0.161 - 0.285

Extrapolation Parameters

Guinier:		Porod:	
A:	3.73237	K:	7.56770E-05
B:	-19249.4	σ:	7.72659E-10
		Bg:	0.288640394577

Transform Type

[✓ Fourier] Hilbert

Output Parameters

Polydispersity:	0.200189
Average Hard Block Thickness (A):	8.36375
Long Period / 2 (A):	28.3275
Average Interface Thickness (A):	0.0
Average Core Thickness:	1.76358
Local Crystallinity:	0.295252

ISIS_98929.TXT

Append Plot To: Graph3

Send To: Correlation Function

Fourier transform completed.

Note: This help document was last changed by Steve King, 28Sep2017

1.4 Tools & Utilities

Note: In Windows use [Alt]-[Cursor left] to return to the previous page

1.4.1 Data Operations Tool

Description

This tool permits arithmetic operations between two data sets. Alternatively, the last data set can be a number.

NOTE! When Data1 *and* Data2 *are both data, their Q (or Qx and Qy for 2D) value(s) must match with each other UNLESS using the 'append' operator.*

Using the tool

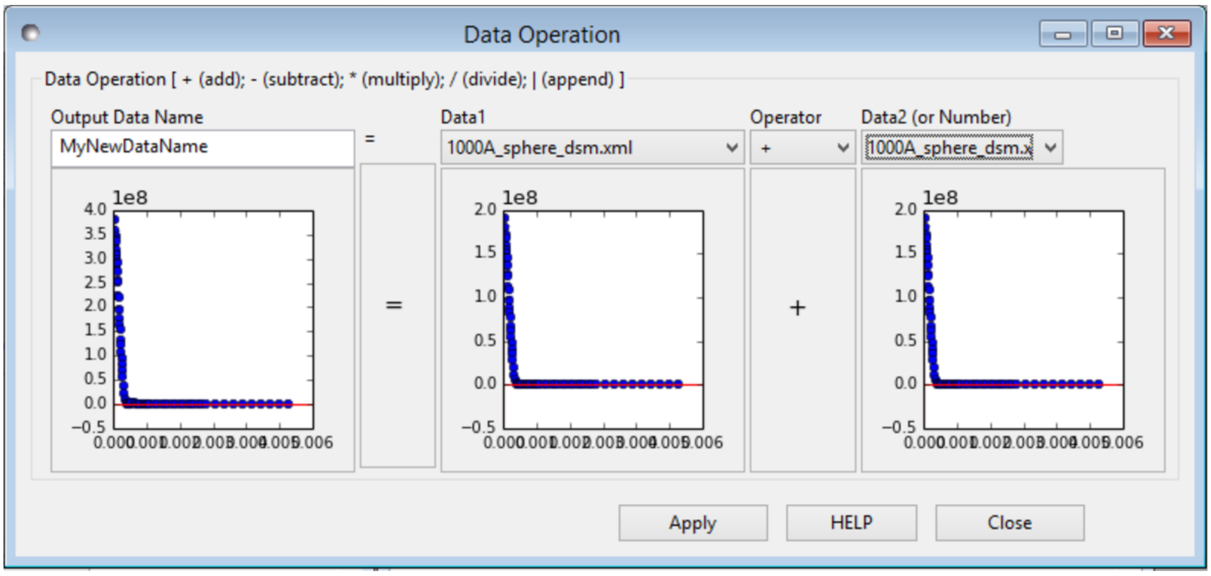
1. Ensure you have loaded data into the *Data Explorer* (see *Loading Data*).
2. Select *Data Operation* from the *Tool* menu on the SasView toolbar.
3. Select a dataset/theory in the drop-down menu *Data1*. A mini-plot of the data will appear underneath.
4. Select a dataset/theory in the drop-down menu *Data2* or select *Number* and enter a number in the box that appears alongside.
5. Select an arithmetic operator symbol from the *Operator* drop-down. The available operators are:

- + (for addition)
- - (for subtraction)
- * (for multiplication)
- / (for division)
- | (for combination of two data sets)

   If two data sets do not match, the operation will fail and the background color of the combo box items will turn to red (WIN only).

6. If the operation is successful, hit the Apply button to make the new dataset. The new dataset will appear in the *Data Explorer*.

NOTE! Any errors and warnings will be displayed at the bottom of the SasView window.



Note: This help document was last changed by Steve King, 01May2015

1.4.2 SLD Calculator Tool

Description

The neutron scattering length density (SLD, β_N) is defined as

$$\beta_N = (b_{c1} + b_{c2} + ... + b_{cn})/V_m$$

where b_{ci} is the bound coherent scattering length of ith of n atoms in a molecule with the molecular volume V_m.

Specifying the Compound Name

To calculate scattering length densities enter the empirical formula of a compound and its mass density and click "Calculate".

Entering a wavelength value is optional (a default value of 6.0 Å will be used).

TIPS!

- Formula strings consist of atoms and the number of them, such as "CaCO3+6H2O".
- Groups can be separated by '+' or *space*, so "CaCO3 6H2O" works as well.
- Groups can be defined using parentheses, such as "CaCO3(H2O)6".
- Parentheses can be nested, such as "(CaCO3(H2O)6)1".
- Isotopes are represented by their atomic number in *square brackets*, such as "CaCO[18]3+6H2O", H[1], or H[2].
- Numbers of atoms can be integer or decimal, such as "CaCO3+(3HO0.5)2".
- The SLD of mixtures can be calculated as well. For example, for a 70-30 mixture of H2O/D2O write "H14O7+D6O3" or more simply "H7D3O5" (i.e. this says 7 hydrogens, 3 deuteriums, and 5 oxygens) and enter a mass density calculated on the percentages of H2O and D2O.
- Type "C[13]6 H[2]12 O[18]6" for C(13)6H(2)12O(18)6 (6 Carbon-13 atoms, 12 deuterium atoms, and 6 Oxygen-18 atoms).

Note: This help document was last changed by Paul Kienzle, 05Apr2017

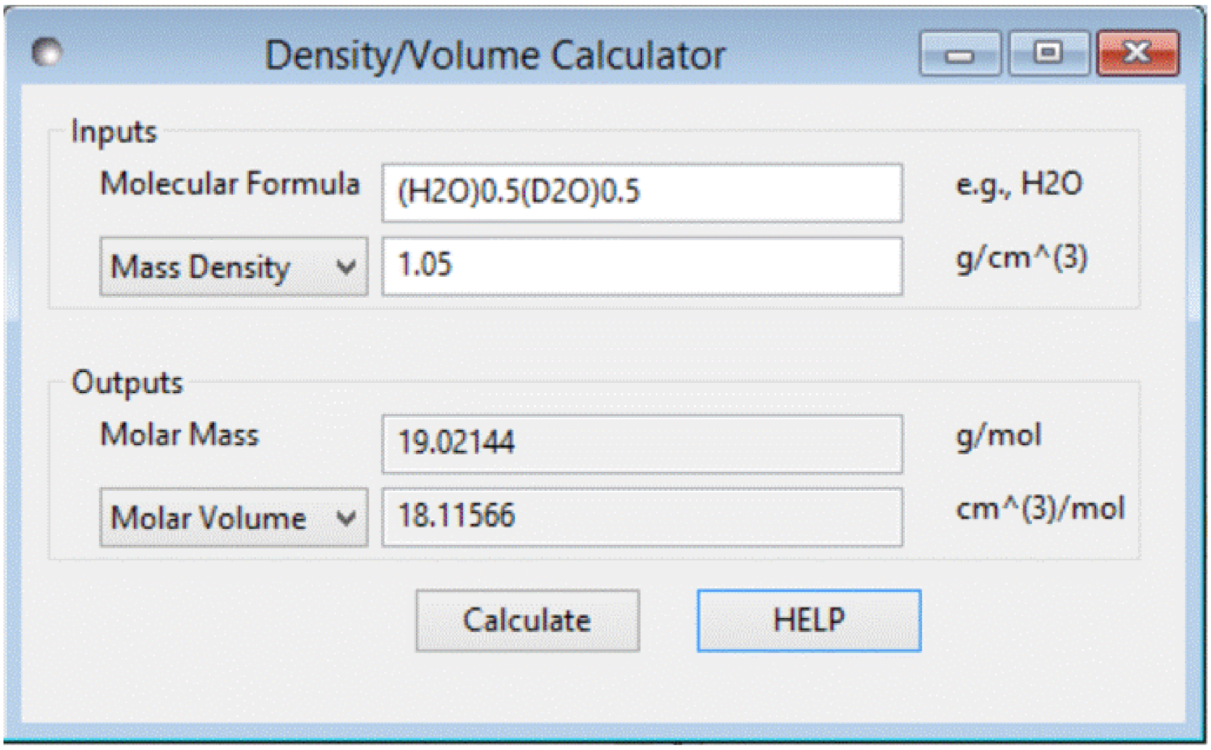
1.4.3 Density/Volume Calculator Tool

Description

This tool calculates the mass density from the molar volume or vice versa. To calculate the mass density, the chemical formula and molar volume should be provided.

Using the tool

1. Select *Density/Volume Calculator* from the *Tool* menu on the SasView toolbar.
2. Enter the empirical formula of a molecule. For mixtures, the ratio of each of the molecules should be used, for example, (H2O)0.5(D2O)0.5.
3. Use the input combo box to choose between molar volume or mass density and then type in an input value.
4. Click the 'Calculate' button to perform the calculation.



Molecular Formula (H2O)0.5(D2O)0.5 e.g., H2O

Mass Density 1.05 g/cm^(3)

Molar Mass 19.02144 g/mol

Molar Volume 18.11566 cm^(3)/mol

Note: This help document was last changed by Steve King, 01May2015

1.4.4 Slit Size Calculator Tool

Description

This tool enables X-ray users to calculate the slit size (FWHM/2) for resolution smearing purposes based on their half beam profile data (as Q vs Intensity; any other data fields are ignored).

Method

The tool works by sequentially summing 10 or more intensity values until a maximum value is attained. It then locates the Q values for the points just before, and just after, **half** of this maximum value and interpolates between them to get an accurate value for the Q value for the half maximum.

NOTE! Whilst it may have some more generic applicability, the calculator has only been tested with beam profile data from Anton-Paar SAXSess^{TM} software. The beam profile file does not carry any information about the units of the Q data. It is probably nm^{-1} but the resolution calculations assume the slit height/width has units of Å^{-1}. If the beam profile data is not in these units then it, or the result, must be manually converted.

Using the tool

1. Select *Slit Size Calculator* from the *Tool* menu on the SasView toolbar.
2. Load a beam profile file in the *Data* field using the *Browse* button.

   NOTE! To see an example of the beam profile file format, visit the file beam profile.DAT in your {installation_directory}/SasView/test_1d folder.
3. Once a data is loaded, the slit size is automatically computed and displayed in the tool window.

Note: This help document was last changed by Steve King, 09Sep2018

1.4.5 Kiessig Thickness Calculator Tool

Description

This tool estimates real space dimensions from the position or spacing of features in recipricol space. In particular a particle of size d will give rise to Bragg peaks with spacing Δq according to the relation

$$d = 2\pi/\Delta q$$

Similarly, the spacing between the peaks in Kiessig fringes in reflectometry data arise from layers of thickness d.

Using the tool

To get a rough thickness or particle size, simply type the fringe or peak position (in units of 1/Å) and click on the *Compute* button.

Note: This help document was last changed by Paul Kienzle, 05Apr2017

1.4.6 Q Resolution Estimator Tool

Description

This tool is approximately estimates the resolution of Q from SAS instrumental parameter values assuming that the detector is flat and normal to the incident beam.

Using the tool

1. Select *SAS Resolution Estimator* from the *Tool* menu on the SasView toolbar.
2. Select the source (Neutron or Photon) and source type (Monochromatic or TOF).

   NOTE! The computational difference between the sources is only the gravitational contribution due to the mass of the particles.
3. Change the default values of the instrumental parameters as required. Be careful to note that distances are specified in cm!
4. Enter values for the source wavelength(s), λ, and its spread (= FWHM/λ).

   For monochromatic sources, the inputs are just one value. For TOF sources, the minimum and maximum values should be separated by a '-' to specify a range.

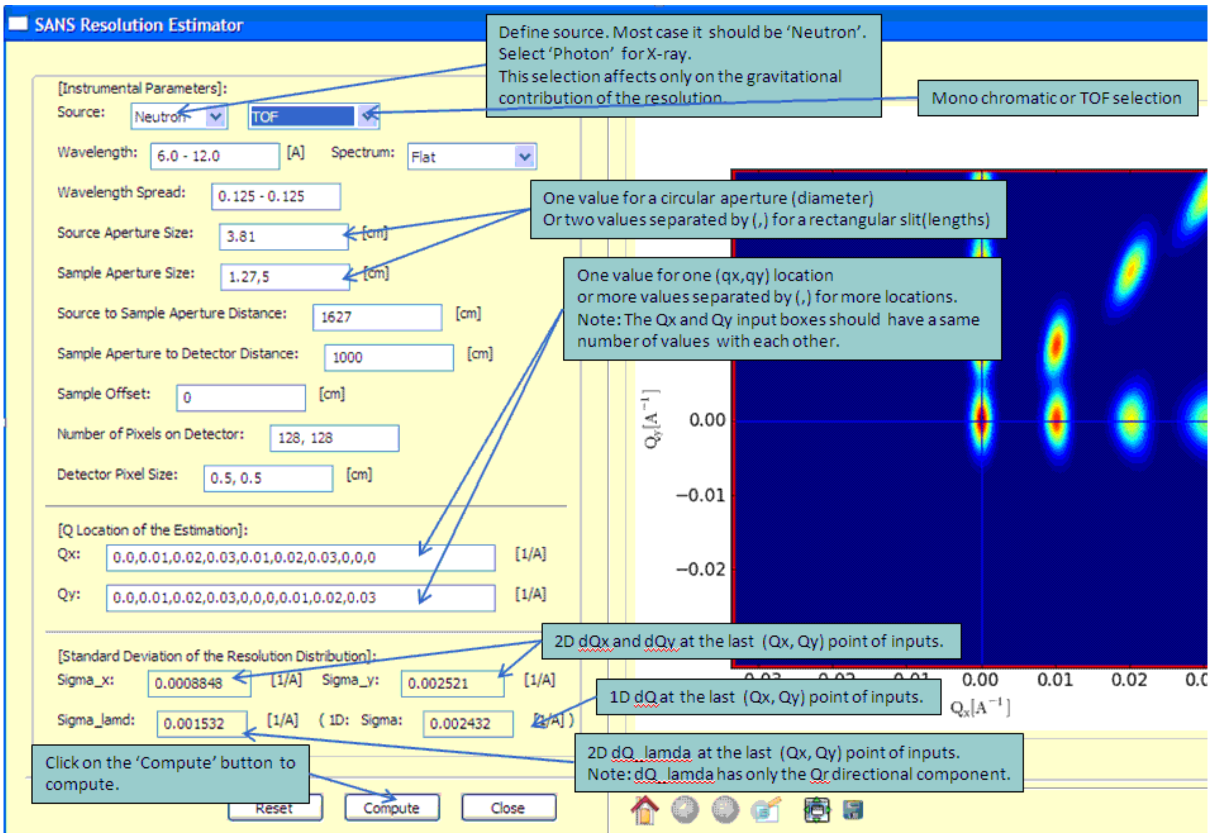
   Optionally, the wavelength (BUT NOT of the wavelength spread) can be extended by adding '; nn' where the 'nn' specifies the number of the bins for the numerical integration. The default value is nn = 10. The same number of bins will be used for the corresponding wavelength spread.
5. For TOF, the default wavelength spectrum is flat. A custom spectral distribution file (2-column text: wavelength (Å) vs Intensity) can also be loaded by selecting *Add new* in the combo box.
6. When ready, click the *Compute* button. Depending on the computation the calculation time will vary.
7. 1D and 2D dQ values will be displayed at the bottom of the panel, and a 2D resolution weight distribution (a 2D elliptical Gaussian function) will also be displayed in the plot panel even if the Q inputs are outside of the detector limit (the red lines indicate the limits of the detector).

   TOF only: green lines indicate the limits of the maximum Q range accessible for the longest wavelength due to the size of the detector.

   Note that the effect from the beam block/stop is ignored, so in the small Q region near the beam block/stop

   [i.e., $Q < (2\pi \cdot w)/(d_s \cdot \lambda_{\mathrm{min}})$, where w is the beam block diameter, d_s is the sample-to-detector distance, and λ_{min} is the minimum wavelength.]

   the variance is slightly under estimated.
8. A summary of the calculation is written to the SasView *Console* at the bottom of the main SasView window.



SANS Resolution Estimator

[Instrumental Parameters]:

Source:	Neutron	TOF		
Wavelength:	6.0 - 12.0	[A]	Spectrum:	Flat
Wavelength Spread:	0.125 - 0.125			
Source Aperture Size:	3.81	[cm]		
Sample Aperture Size:	1.27,5	[cm]		
Source to Sample Aperture Distance:	1627	[cm]		
Sample Aperture to Detector Distance:	1000	[cm]		
Sample Offset:	0	[cm]		
Number of Pixels on Detector:	128, 128			
Detector Pixel Size:	0.5, 0.5	[cm]		

[Q Location of the Estimation]:

Qx:	0.0,0.01,0.02,0.03,0.01,0.02,0.03,0,0,0	[1/A]
Qy:	0.0,0.01,0.02,0.03,0,0,0,0.01,0.02,0.03	[1/A]

[Standard Deviation of the Resolution Distribution]:

Sigma_x:	0.0008848	[1/A]	Sigma_y:	0.002521 [1/A]
Sigma_lamd:	0.001532	[1/A]	(1D: Sigma:	0.002432 [1/A])

Define source. Most case it should be 'Neutron'. Select 'Photon' for X-ray. This selection affects only on the gravitational contribution of the resolution.

Mono chromatic or TOF selection

One value for a circular aperture (diameter) Or two values separated by (,) for a rectangular slit(lengths)

One value for one (qx,qy) location or more values separated by (,) for more locations. Note: The Qx and Qy input boxes should have a same number of values with each other.

2D dQx and dQy at the last (Qx, Qy) point of inputs.

1D dQ at the last (Qx, Qy) point of inputs.

2D dQ_lamda at the last (Qx, Qy) point of inputs. Note: dQ_lamda has only the Qr directional component.

Click on the 'Compute' button to compute.

Theory

The scattering wave transfer vector is by definition

$\mathbf{q} = \mathbf{k}_2 - \mathbf{k}_1,$

$q = |\mathbf{q}| = 4\pi/\lambda \sin(\theta/2)$

Variance of q,

$\sigma_q^2 \sim q^2 [(\Delta\lambda)^2 / \lambda^2 + (\Delta\theta)^2/\theta^2]$ for a small angle.

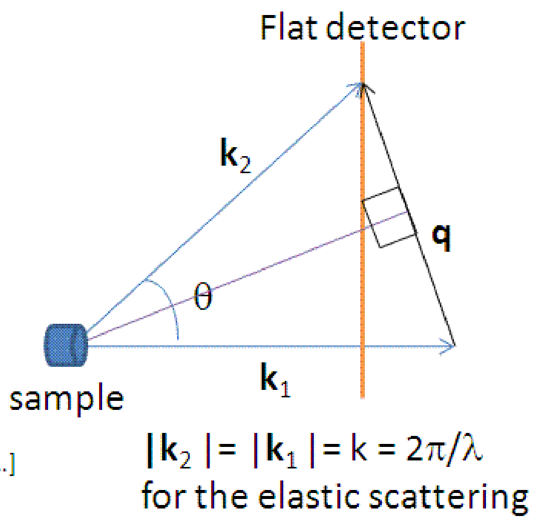
$= \sigma_\lambda^2 + \sigma_{geo}^2 (+ \sigma_{gr}^2)$

where $\sigma_{geo}^2 = \sigma_{src}^2 + \sigma_{sample}^2 + \sigma_{det}^2$

$= <\mathbf{q}^2> - <\mathbf{q}>^2$

$= k^2 [<T_1^2> - <T_1>^2] + ...,$

setting $\mathbf{q} = 2\pi/\lambda[T_0 + T_1(\mathbf{r}) + T_2(\mathbf{r}) + ...]$


Flat detector
$\mathbf{k}_2$
$\mathbf{q}$
θ
sample
$\mathbf{k}_1$
$|\mathbf{k}_2| = |\mathbf{k}_1| = k = 2\pi/\lambda$
for the elastic scattering

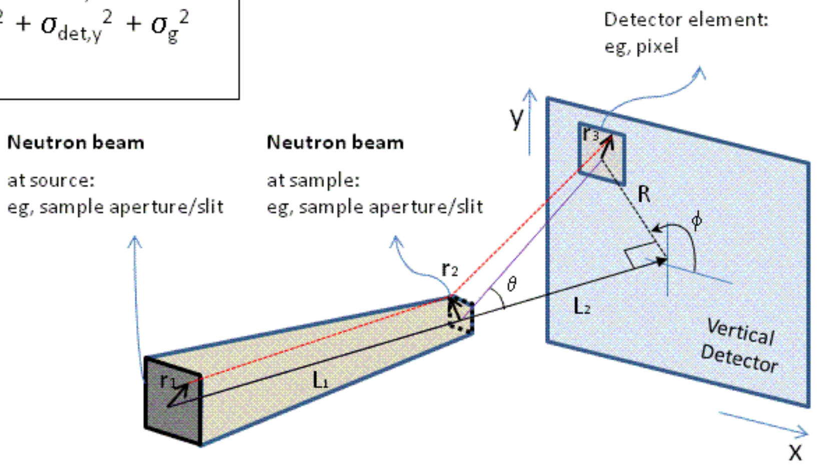
In the small-angle limit, the variance of Q is to a first-order approximation

$$\begin{aligned}\sigma_{q,x}^2 &= \sigma_{src,x}^2 + \sigma_{sample,x}^2 + \sigma_{det,x}^2\\ \sigma_{q,y}^2 &= \sigma_{src,y}^2 + \sigma_{sample,y}^2 + \sigma_{det,y}^2 + \sigma_g^2\\ \sigma_{q,r}^2 &= \sigma_\lambda^2\end{aligned}$$

$$\sigma_{src,r_1}^{\ 2} = k^2\left[\frac{\langle r_1^2\rangle}{L_1^2}\right]$$

$$\sigma_{sample,r_2}^{\ 2} = k^2\left[\frac{\langle r_2^2\rangle}{L'^2}\right]$$

$$\sigma_{det,r_3}^{\ 2} = k^2\frac{\langle r_3^2\rangle}{L_2^2}$$

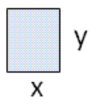
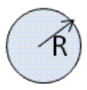
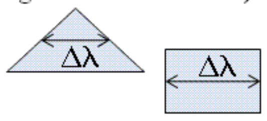


$$1/L' = 1/L_1 + 1/L_2$$

$$\sigma_\lambda^{\ 2} = \frac{k^2}{12}\left[\frac{b(2A\lambda_0^2\hat{y} - R\hat{r})^2}{L_2^2}\left(\frac{\Delta\lambda}{\lambda}\right)^2\right]$$

$$, A = L_2(L_1+L_2)gm^2/(2h^2) \text{ where } \Delta y_g = -A\lambda^2.$$

The geometric and gravitational contributions can then be summarised as

Geometric Contribution Case I: Rectangular shape w/ x by y : 	$\langle r^2\rangle_x = x^2/12, \quad \langle r^2\rangle_y = y^2/12$ $1D: \quad \langle r^2\rangle_{1D} = (\langle r^2\rangle_x + \langle r^2\rangle_y)/2$
Geometric Contribution Case II: Circular shape w/ R = radius of an aperture; 	$\langle r^2\rangle_x = R^2/4, \quad \langle r^2\rangle_y = R^2/4$ $1D: \quad \langle r^2\rangle_{1D} = (\langle r^2\rangle_x + \langle r^2\rangle_y)/2$
The **wavelength contribution** is always in **R direction (including the gravitational effect)** Triangular pulse: b = 2 Rectangular pulse: b = 1 	$\sigma_\lambda^{\ 2} = \frac{k^2}{12}\left[\frac{b(2A\lambda_0^2\hat{y} - R\hat{r})^2}{L_2^2}\left(\frac{\Delta\lambda}{\lambda}\right)^2\right]$ $1D: \quad (\sigma_\lambda^{\ 2})_{1D} = \sigma_{\lambda,r}^{\ 2} = \frac{k^2}{12}\left[\frac{b(R^2 + 2A\lambda_0^2)}{L_2^2}\left(\frac{\Delta\lambda}{\lambda}\right)^2\right]$
The **gravitational contribution** is always in **y (vertical) direction** and can not be separated from the wavelength contribution.	See above. The terms with A are due to the gravitation.

Finally, a Gaussian function is used to describe the 2D weighting distribution of the uncertainty in Q.

References

D.F.R. Mildner and J.M. Carpenter *J. Appl. Cryst.* 17 (1984) 249-256

D.F.R. Mildner, J.M. Carpenter and D.L. Worcester *J. Appl. Cryst.* 19 (1986) 311-319

Note: This help document was last changed by Steve King, 01May2015

1.4.7 Generic SANS Calculator Tool

Description

This tool attempts to simulate the SANS expected from a specified shape/structure or scattering length density profile. The tool can handle both nuclear and magnetic contributions to the scattering.

Theory

In general, a particle with a volume V can be described by an ensemble containing N 3-dimensional rectangular pixels where each pixel is much smaller than V.

Assuming that all the pixel sizes are the same, the elastic scattering intensity from the particle is

$$I(\vec{Q}) = \frac{1}{V}\left|\sum_j^N v_j\beta_j \exp(i\vec{Q}\cdot\vec{r}_j)\right|^2$$

Equation 1.

where β_j and r_j are the scattering length density and the position of the j^{th} pixel respectively.

The total volume V

$$V = \sum_j^N v_j$$

for $\beta_j \neq 0$ where v_j is the volume of the j^{th} pixel (or the j^{th} natural atomic volume (= atomic mass / (natural molar density * Avogadro number) for the atomic structures).

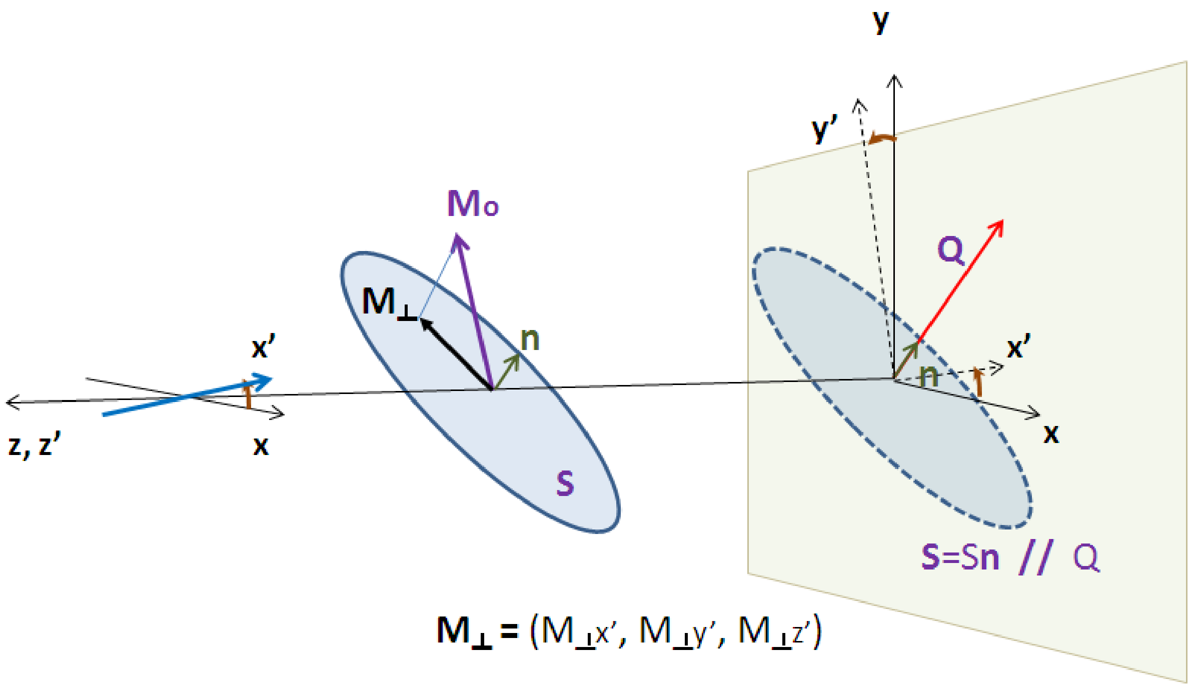
V can be corrected by users. This correction is useful especially for an atomic structure (such as taken from a PDB file) to get the right normalization.

NOTE! β_j displayed in the GUI may be incorrect but this will not affect the scattering computation if the correction of the total volume V is made.

The scattering length density (SLD) of each pixel, where the SLD is uniform, is a combination of the nuclear and magnetic SLDs and depends on the spin states of the neutrons as follows.

Magnetic Scattering

For magnetic scattering, only the magnetization component, $\mathbf{M}_\perp$, perpendicular to the scattering vector $\vec{Q}$ contributes to the magnetic scattering length.



The magnetic scattering length density is then

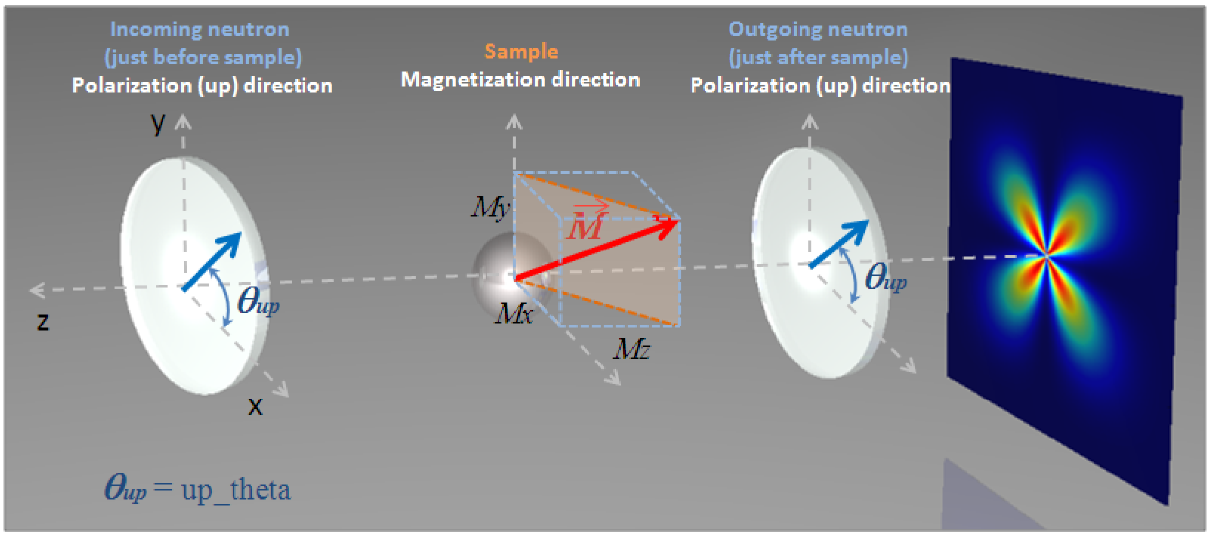
$$\beta_M = \frac{\gamma r_0}{2\mu_B}\sigma \cdot \mathbf{M}_\perp = D_M \sigma \cdot \mathbf{M}_\perp$$

where the gyromagnetic ratio is $\gamma = -1.913$, μ_B is the Bohr magneton, r_0 is the classical radius of electron, and σ is the Pauli spin.

For a polarized neutron, the magnetic scattering is depending on the spin states.

Let us consider that the incident neutrons are polarised both parallel (+) and anti-parallel (-) to the x' axis (see below). The possible states after scattering from the sample are then

- Non-spin flips: (+ +) and (- -)
- Spin flips: (+ -) and (- +)



Now let us assume that the angles of the $\vec{Q}$ vector and the spin-axis (x') to the x-axis are ϕ and $\theta_{\rm up}$ respectively (see above). Then, depending upon the polarization (spin) state of neutrons, the scattering length densities, including the nuclear scattering length density (β_N) are given as

- for non-spin-flips

$$\beta_{\pm\pm} = \beta_N \mp D_M M_{\perp x'}$$

- for spin-flips

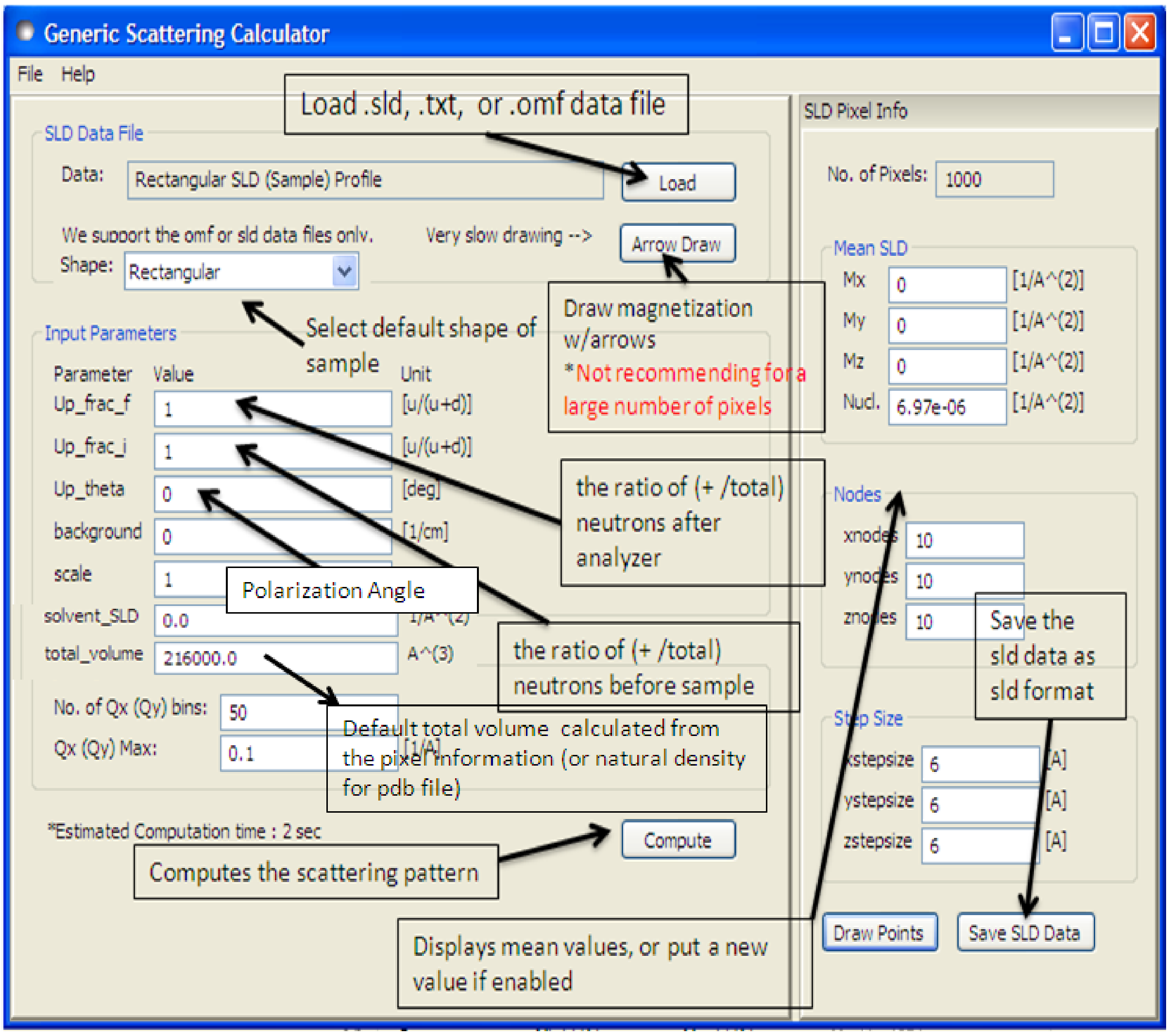
$$\beta_{\pm\mp} = -D_M (M_{\perp y'} \pm i M_{\perp z'})$$

where

$$\begin{aligned} M_{\perp x'} &= M_{0q_x} \cos\theta_{\text{up}} + M_{0q_y} \sin\theta_{\text{up}} \\ M_{\perp y'} &= M_{0q_y} \cos\theta_{\text{up}} - M_{0q_x} \sin\theta_{\text{up}} \\ M_{\perp z'} &= M_{0z} \\ M_{0q_x} &= (M_{0x} \cos\phi - M_{0y} \sin\phi) \cos\phi \\ M_{0q_y} &= (M_{0y} \sin\phi - M_{0y} \cos\phi) \sin\phi \end{aligned}$$

Here the M_{0x}, M_{0y} and M_{0z} are the x, y and z components of the magnetisation vector in the laboratory x-y-z frame.

Using the tool



Generic Scattering Calculator

File Help

Load .sld, .txt, or .omf data file

SLD Data File

Data: Rectangular SLD (Sample) Profile — Load

We support the omf or sld data files only.

Very slow drawing --> Arrow Draw

Shape: Rectangular

Select default shape of sample

Draw magnetization w/arrows

*Not recommending for a large number of pixels

Input Parameters

Parameter	Value	Unit
Up_frac_f	1	[u/(u+d)]
Up_frac_i	1	[u/(u+d)]
Up_theta	0	[deg]
background	0	[1/cm]
scale	1	
solvent_SLD	0.0	1/A^(2)
total_volume	216000.0	A^(3)

the ratio of (+ /total) neutrons after analyzer

Polarization Angle

the ratio of (+ /total) neutrons before sample

No. of Qx (Qy) bins: 50

Qx (Qy) Max: 0.1 [1/A]

Default total volume calculated from the pixel information (or natural density for pdb file)

*Estimated Computation time : 2 sec

Compute

Computes the scattering pattern

Displays mean values, or put a new value if enabled

SLD Pixel Info

No. of Pixels: 1000

Mean SLD

Mx	0	[1/A^(2)]
My	0	[1/A^(2)]
Mz	0	[1/A^(2)]
Nucl.	6.97e-06	[1/A^(2)]

Nodes

xnodes	10
ynodes	10
znodes	10

Save the sld data as sld format

Step Size

xstepsize	6	[A]
ystepsize	6	[A]
zstepsize	6	[A]

Draw Points   Save SLD Data

After computation the result will appear in the *Theory* box in the SasView *Data Explorer* panel.

Up_frac_in and *Up_frac_out* are the ratio

(spin up) / (spin up + spin down)

of neutrons before the sample and at the analyzer, respectively.

NOTE 1. The values of Up_frac_in *and* Up_frac_out *must be in the range 0.0 to 1.0. Both values are 0.5 for unpolarized neutrons.*

NOTE 2. This computation is totally based on the pixel (or atomic) data fixed in xyz coordinates. No angular orientational averaging is considered.

NOTE 3. For the nuclear scattering length density, only the real component is taken account.

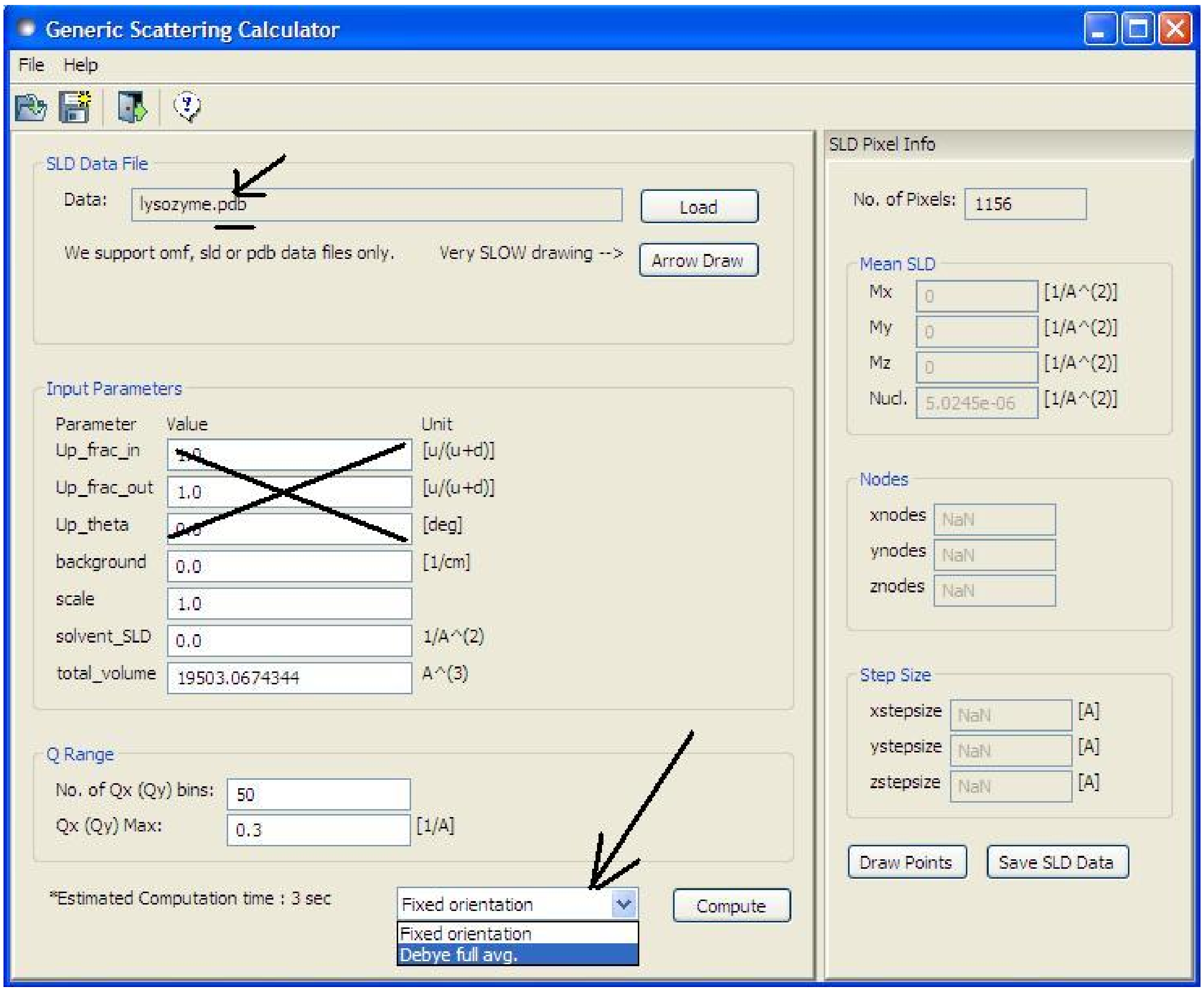
Using PDB/OMF or SLD files

The SANS Calculator tool can read some PDB, OMF or SLD files but ignores polarized/magnetic scattering when doing so, thus related parameters such as *Up_frac_in*, etc, will be ignored.

The calculation for fixed orientation uses Equation 1 above resulting in a 2D output, whereas the scattering calculation averaged over all the orientations uses the Debye equation below providing a 1D output

$$I(|\vec{Q}|) = \frac{1}{V}\sum_j^N v_j\beta_j \sum_k^N v_k\beta_k \frac{\sin(|\vec{Q}||\vec{r_j}-\vec{r_k}|)}{|\vec{Q}||\vec{r_j}-\vec{r_k}|}$$

where $v_j\beta_j \equiv b_j$ is the scattering length of the j^{th} atom. The calculation output is passed to the *Data Explorer* for further use.



Data: lysozyme.pdb

We support omf, sld or pdb data files only.

Very SLOW drawing -->

Input Parameters

Parameter	Value	Unit
Up_frac_in	~~1.0~~	[u/(u+d)]
Up_frac_out	~~1.0~~	[u/(u+d)]
Up_theta	~~0.0~~	[deg]
background	0.0	[1/cm]
scale	1.0	
solvent_SLD	0.0	1/A^(2)
total_volume	19503.0674344	A^(3)

Q Range

No. of Qx (Qy) bins: 50

Qx (Qy) Max: 0.3 [1/A]

*Estimated Computation time : 3 sec

Fixed orientation

Fixed orientation
Debye full avg.

SLD Pixel Info

No. of Pixels: 1156

Mean SLD

Mx	0	[1/A^(2)]
My	0	[1/A^(2)]
Mz	0	[1/A^(2)]
Nucl.	5.0245e-06	[1/A^(2)]

Nodes

xnodes	NaN
ynodes	NaN
znodes	NaN

Step Size

xstepsize	NaN	[A]
ystepsize	NaN	[A]
zstepsize	NaN	[A]

Note: This help document was last changed by Steve King, 01May2015

1.4.8 Python Shell-Editor Tool

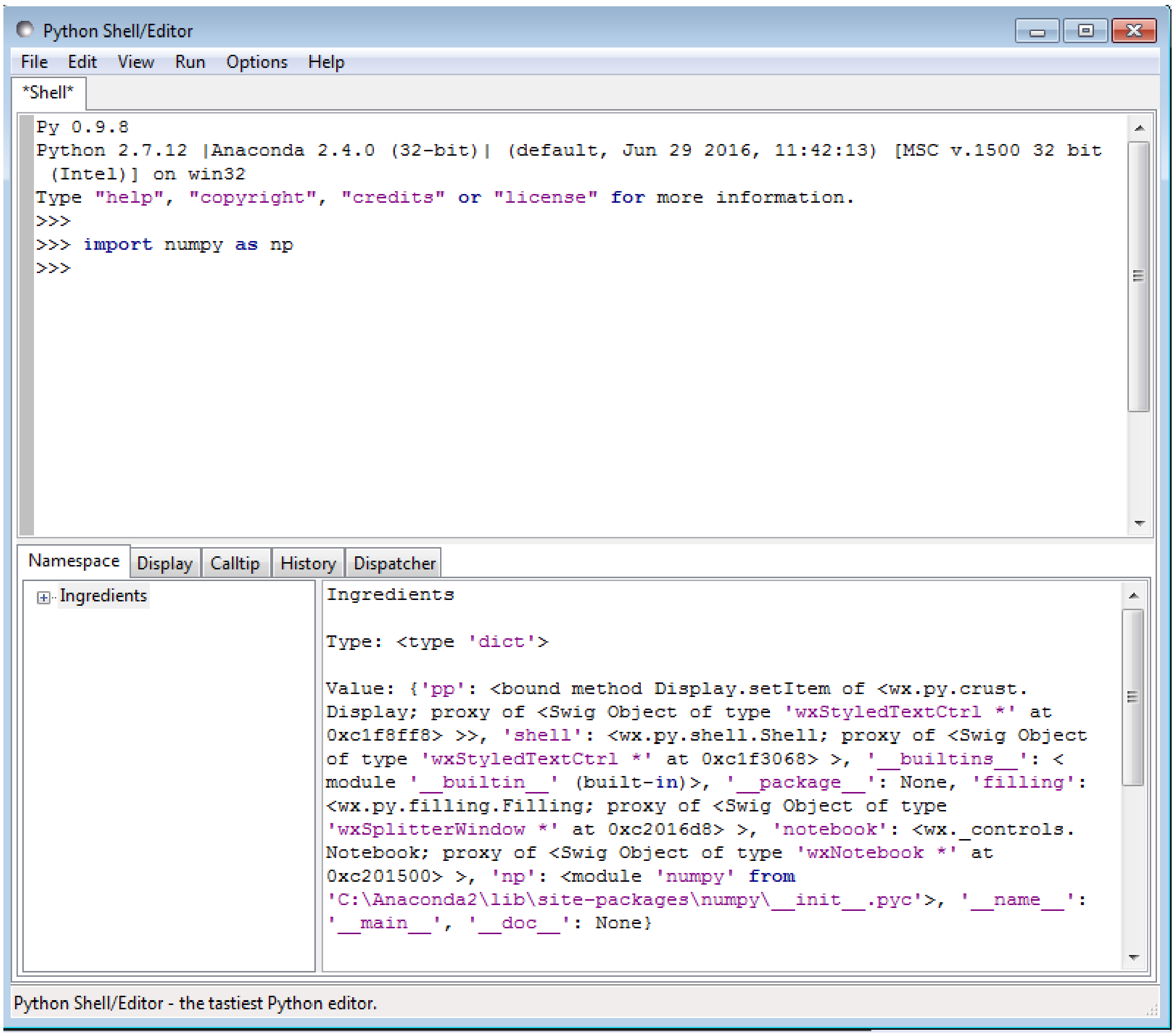
Description

This is a Python shell/editor provided with WxPython.

For the help about Python, visit the website http://docs.python.org/tutorial/

Note: This shell/editor has its own help, but the Help() and Credits() calls do not work on Macs.

The NumPy, SciPy, and Matplotlib, etc, libraries are shipped with SasView and so functions from these can be imported into the shell/editor, however, some functionality may not work.



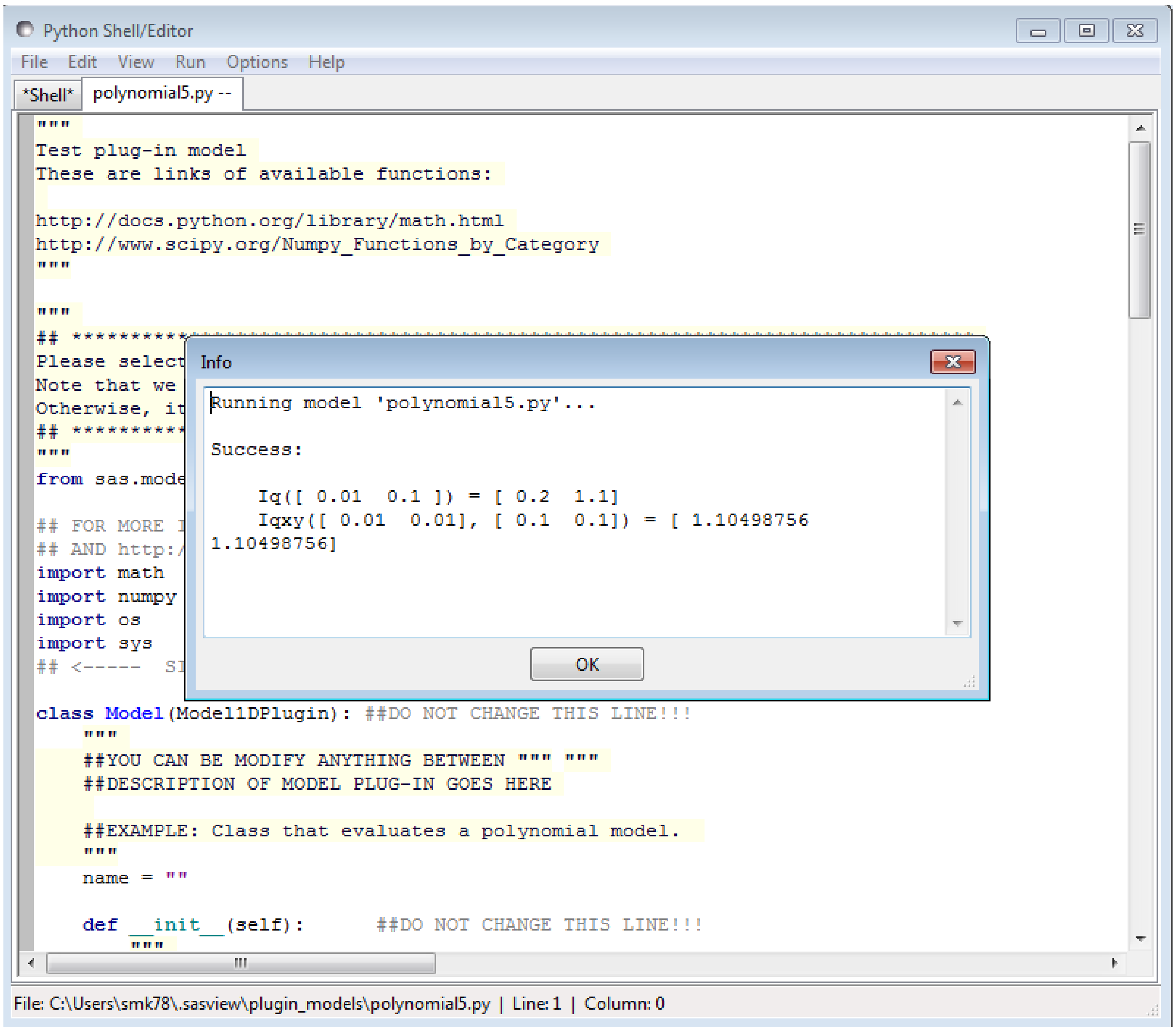
```
Py 0.9.8
Python 2.7.12 |Anaconda 2.4.0 (32-bit)| (default, Jun 29 2016, 11:42:13) [MSC v.1500 32 bit
 (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> import numpy as np
>>>
```

```
Ingredients

Type: <type 'dict'>

Value: {'pp': <bound method Display.setItem of <wx.py.crust.
Display; proxy of <Swig Object of type 'wxStyledTextCtrl *' at
0xc1f8ff8> >>, 'shell': <wx.py.shell.Shell; proxy of <Swig Object
of type 'wxStyledTextCtrl *' at 0xc1f3068> >, '__builtins__': <
module '__builtin__' (built-in)>, '__package__': None, 'filling':
<wx.py.filling.Filling; proxy of <Swig Object of type
'wxSplitterWindow *' at 0xc2016d8> >, 'notebook': <wx._controls.
Notebook; proxy of <Swig Object of type 'wxNotebook *' at
0xc201500> >, 'np': <module 'numpy' from
'C:\Anaconda2\lib\site-packages\numpy\__init__.pyc'>, '__name__':
'__main__', '__doc__': None}
```

When a Python file, for example a fitting model, is created or loaded with the *New* or *Open* options from the menu, a new tab opens with an editing notebook.



```
"""
Test plug-in model
These are links of available functions:

http://docs.python.org/library/math.html
http://www.scipy.org/Numpy_Functions_by_Category
"""

"""
## **********
Please select
Note that we
Otherwise, it
## **********
"""
from sas.mode

## FOR MORE I
## AND http:/
import math
import numpy
import os
import sys
## <-----   SI

class Model(Model1DPlugin): ##DO NOT CHANGE THIS LINE!!!
    """
    ##YOU CAN BE MODIFY ANYTHING BETWEEN """ """
    ##DESCRIPTION OF MODEL PLUG-IN GOES HERE

    ##EXAMPLE: Class that evaluates a polynomial model.
    """
    name = ""

    def __init__(self):      ##DO NOT CHANGE THIS LINE!!!
        """
```

```
Running model 'polynomial5.py'...

Success:

    Iq([ 0.01  0.1 ]) = [ 0.2  1.1]
    Iqxy([ 0.01  0.01], [ 0.1  0.1]) = [ 1.10498756
1.10498756]
```

File: C:\Users\smk78\.sasview\plugin_models\polynomial5.py | Line: 1 | Column: 0

If a Python (.py) model has a linked C (.c) subroutine *in the same folder* then the shell/editor will open both! However input focus is usually transferred to the tab with the .c file.

To compile a model, select *Run* > *Check Model* from the shell/editor menu. If the model contains a unit test (which it should!!!) then this will also run and a popup window will report the success/failure of the test.

Note: This help document was last changed by Steve King, 10Oct2015

1.4.9 Image Viewer Tool

Description

This tool loads image files and displays them as 2D (x-y coordinate against counts per pixel). The plot can then can be saved, printed, and copied. The plot can also be resized by dragging the corner of the panel.

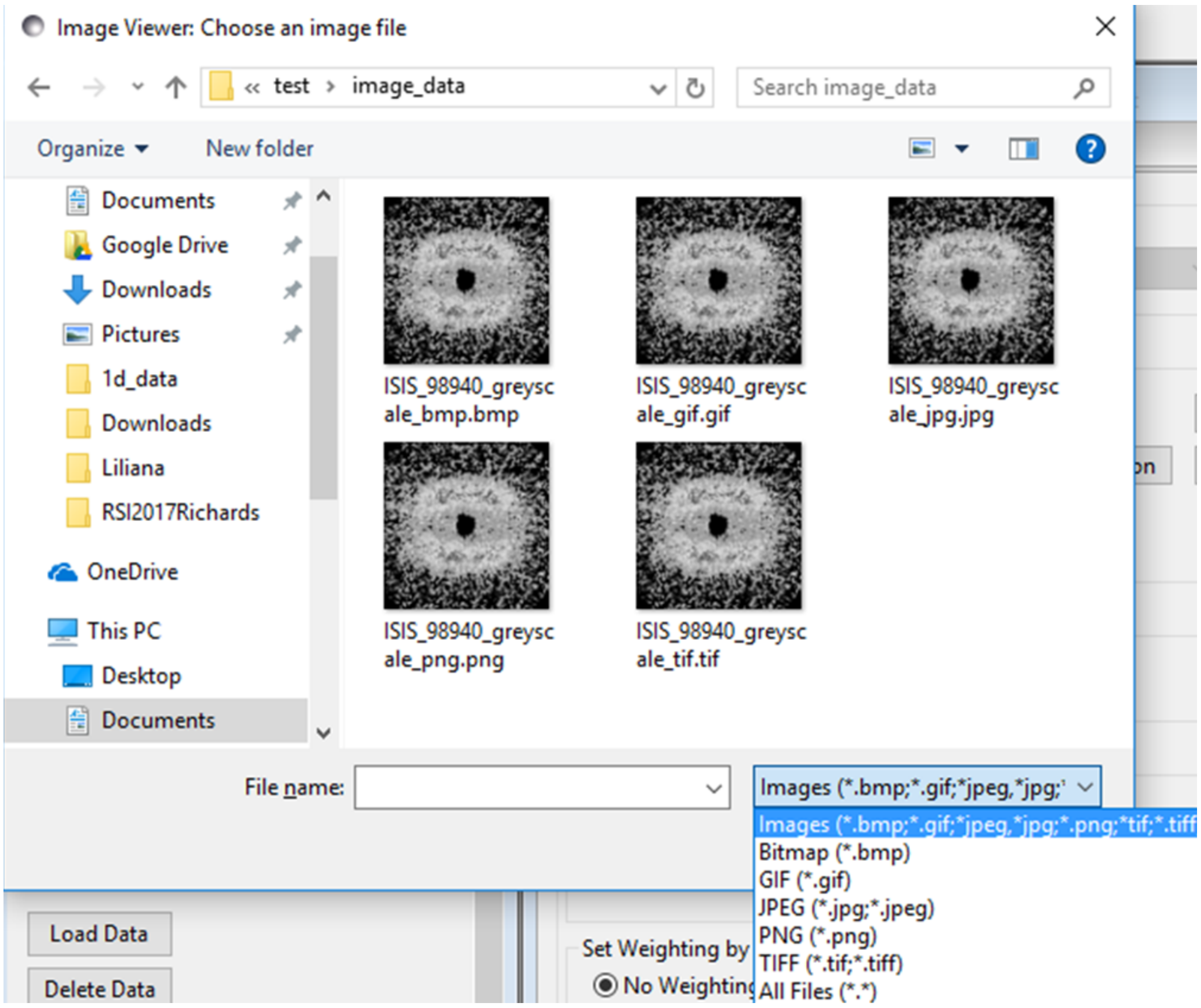
The supported input image formats are:

- BMP (bitmap format)
- GIF (graphical interchange format)
- JPG (joint photographic experts group format)
- PNG (portable network graphics format)

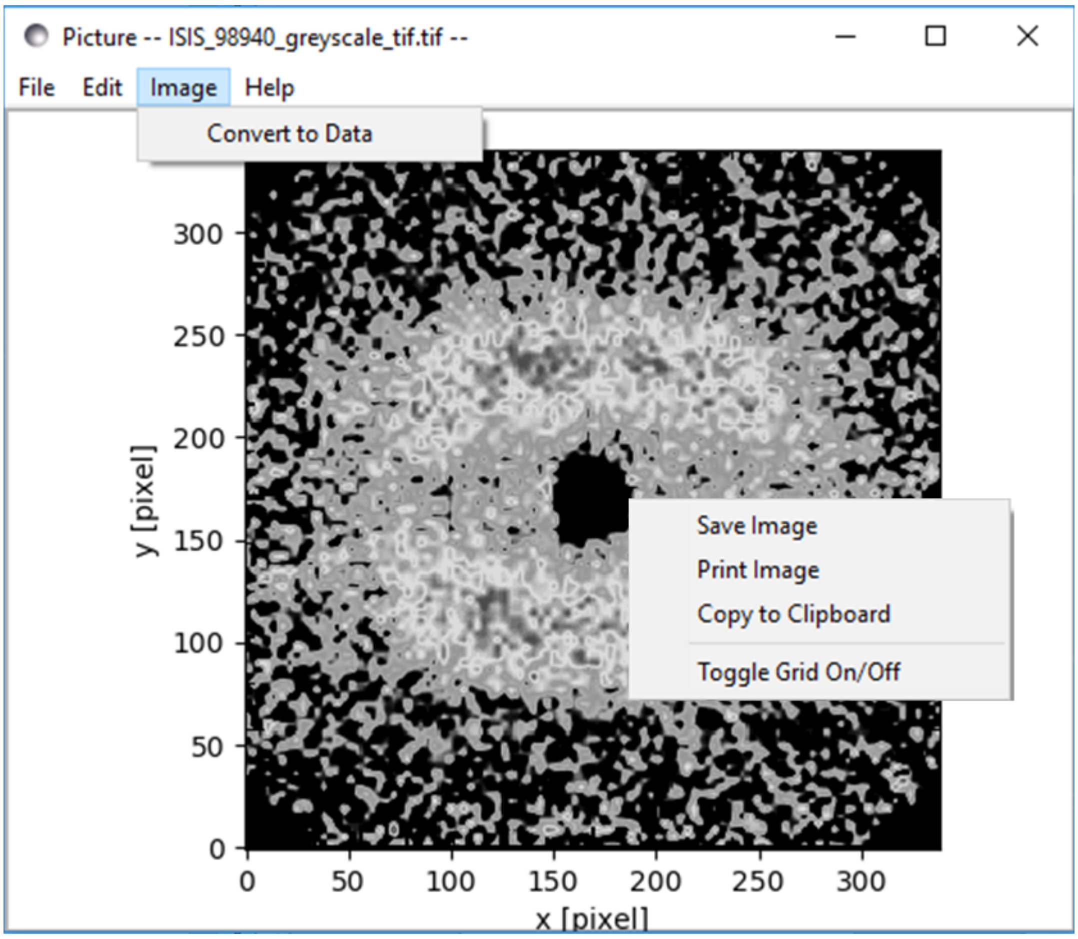
- TIF (tagged image format)

Using the tool

1. Select *Image Viewer* from the *Tool* menu on the SasView toolbar.
2. Select a file and then click *Open*. If the loading is successful the image will be displayed.



3. To save, print, or copy the image, or to apply a grid overlay, right-click anywhere in the plot.

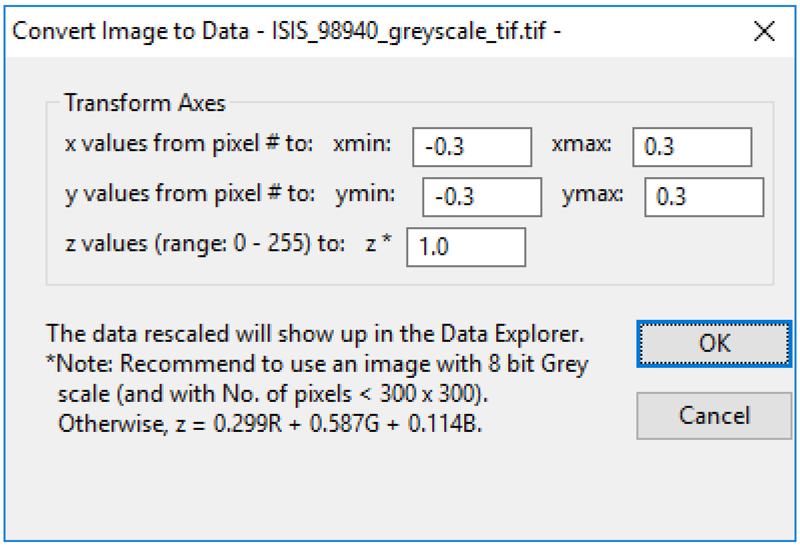


4. If the image is taken from a 2D detector, SasView can attempt to convert the colour/grey scale into pseudo-intensity 2D data using

   z = (0.299 x R) + (0.587 x G) + (0.114 x B)

   unless the image is formatted as 8-bit grey-scale TIF.

5. In the *Convert to Data* dialog, set the parameters relevant to the data and then click the OK.


Convert Image to Data - ISIS_98940_greyscale_tif.tif -

Transform Axes

x values from pixel # to: xmin: -0.3 xmax: 0.3

y values from pixel # to: ymin: -0.3 ymax: 0.3

z values (range: 0 - 255) to: z * 1.0

The data rescaled will show up in the Data Explorer.
*Note: Recommend to use an image with 8 bit Grey scale (and with No. of pixels < 300 x 300).
Otherwise, z = 0.299R + 0.587G + 0.114B.

Note: This help document was last changed by Steve King, 01May2015

1.4.10 File Converter Tool

Description

This tool converts file formats with the Q data and Intensity data stored in separate files, into a single CanSAS (XML) or NXcanSAS (HDF5) file.

It can also convert 2D BSL/OTOKO files into a NXcanSAS file.

Supported input file formats (examples may be found in the /test/convertible_files folder):

- Single-column ASCII data, with lines that end without any delimiter, or with a comma or semi-colon delimiter
- 2D ISIS ASCII formatted data
- 1D BSL/OTOKO format data
- 2D BSL/OTOKO format data

Supported output file formats:

- CanSAS
- NXcanSAS

Using the Tool

1. Select the files containing your Q-axis and Intensity-axis data
2. Choose whether the files are in ASCII 1D, ASCII 2D, 1D BSL/OTOKO or 2D BSL/OTOKO format
3. Choose where you would like to save the converted file
4. Optionally, input some metadata such as sample size, detector name, etc
5. Click *Convert* to save the converted file

Files With Multiple Frames

If a BSL/OTOKO file with multiple frames is selected for the Intensity-axis file, a dialog will appear asking which frames you would like converted. You may enter a start frame, end frame & increment, and all frames in that subset will be converted. For example, entering 0, 50 and 10 will convert frames 0, 10, 20, 30, 40 & 50.

To convert a single frame, enter the same value for first frame & last frame, and 1 as the increment.

CanSAS XML files can become quite large when exporting multiple frames to a single file, so there is an option in the *Select Frame* dialog to output each frame to its own file. The single file option will produce one file with multiple <*SASdata*> elements. The multiple file option will output a separate file with one <*SASdata*> element for each frame. The frame number will also be appended to the file name.

The multiple file option is not available when exporting to NXcanSAS because the HDF5 format is more efficient at handling large amounts of data.

Note: This help document was last changed by Steve King, 08Oct2016

1.5 Working with SasView

Note: In Windows use [Alt]-[Cursor left] to return to the previous page

1.5.1 Data Formats

SasView reads several different 1D SAS (*I(Q) vs Q*), 2D SAS(*I(Qx,Qy) vs (Qx,Qy)*) and 1D SESANS (*P(z) vs z*) data files. From SasView 4.1 onwards, a *File Converter Tool* allows some legacy formats to be converted into modern formats that SasView will read.

1D SAS Formats

SasView will read ASCII ('text') files with 2 to 4 columns of numbers in the following order:

Q, I(Q), (dI(Q), dQ(Q))

where *dQ(Q)* is the instrumental resolution in *Q* and assumed to have originated from pinhole geometry.

Numbers can be separated by spaces or commas.

SasView recognises the following file extensions which are not case-sensitive:

- .TXT
- .DAT
- .XML (in canSAS format v1.0 and 1.1)
- .H5 (as NeXus NXcanSAS only)
- .NXS (as NeXus NXcanSAS only)

Note: From SasView version 4.2 onwards files written in the NIST .ASC format are no longer read. This is because that format normally represents *raw* and not reduced data.

If using CSV output from, for example, a spreadsheet, ensure that it is not using commas as delimiters for thousands.

The SasView *File Converter Tool* available in SasView 4.1 onwards can be used to convert data sets with separated *I(Q)* and *Q* files (for example, BSL/OTOKO, and some output from FIT2D and other SAXS-oriented software) into either the canSAS SASXML (XML) format or the NeXus NXcanSAS (HDF5) format.

For a description of the CanSAS/SASXML format see: http://www.cansas.org/formats/canSAS1d/1.1/doc/

For a description of the ISIS 1D format see: https://www.isis.stfc.ac.uk/Pages/colette-ascii-file-format-descriptions.pdf

For a description of the NXcanSAS format see: http://cansas-org.github.io/NXcanSAS/classes/contributed_definitions/NXcanSAS.html

All the above formats are written by the Mantid Framework.

For a description of the NIST 1D format see: http://danse.chem.utk.edu/trac/wiki/NCNROutput1D_IQ

For a description of the BSL/OTOKO format see: http://www.diamond.ac.uk/Beamlines/Soft-Condensed-Matter/small-angle/SAXS-Software/CCP13/BSL.html

2D SAS Formats

SasView will read ASCII ('text') files in the NIST 2D format (with the extension .DAT) or files in the NeXus NXcanSAS (HDF5) format (with the extension .H5 or .NXS). File extensions are not case-sensitive. Both of these formats are written by the Mantid Framework.

Most of the header lines in the NIST 2D format can actually be removed except the last line, and only the first three columns (*Qx, Qy,* and *I(Qx,Qy)*) are actually required.

Note: From SasView version 4.2 onwards files written in the NIST .ASC format are no longer read. This is because that format normally represents *raw* and not reduced data.

Note: SasView does not read the standard NeXus format, only the NXcanSAS subset.

The SasView *File Converter Tool* available in SasView 4.1 onwards can be used to convert data sets in the 2D BSL/OTOKO format into the NeXus NXcanSAS (HDF5) format.

For a description of the NIST 2D format see: http://danse.chem.utk.edu/trac/wiki/NCNROutput1D_2DQxQy

For a description of the NXcanSAS format see: http://cansas-org.github.io/NXcanSAS/classes/contributed_definitions/NXcanSAS.html

For a description of the BSL/OTOKO format see: For a description of the BSL/OTOKO format see: http://www.diamond.ac.uk/Beamlines/Soft-Condensed-Matter/small-angle/SAXS-Software/CCP13/BSL.html

1D SESANS Format

SasView version 4.1 onwards will read ASCII ('text') files in a prototype SESANS standard format (with the extensions .SES or .SESANS). The file extensions are not case-sensitive.

The file format has a list of name-value pairs at the top of the file which detail the general experimental parameters necessary for fitting and analyzing data. This list should contain all the information necessary for the file to be 'portable' between users.

Following the header is a 8 (only the first 4 are really needed) column list of instrument experimental variables:

- Spin echo length (z, in Angstroms)
- depolarization $(log(P/P_0)/(lambda^2 * thickness)$, in Angstrom $^{-1}$ cm $^{-1}$)
- depolarization error in the same unit) (measurement error)
- Spin echo length error (Δz, in Angstroms) (experimental resolution)
- Neutron wavelength (λ, in Angstroms)
- Neutron wavelength error ($\Delta\lambda$, in Angstroms)
- Normalized polarization (P/P_0, unitless)
- Normalized polarization error ($\Delta(P/P_0)$, unitless) (measurement error)

Note: This help document was last changed by Wim Bouwman, 05Apr2017

1.5.2 Loading Data

The data explorer

Data Explorer is a panel that allows the user more interactions with data. Some functionalities provided by the *Data Explorer* are also available through the context menu of plot panels or other menus within the application.

Under *View* in the menu bar, *Data Explorer* can be toggled between Show and Hide by clicking *Show/Hide Data Explorer*.

NOTE! When Data Explorer *is hidden, all data loaded will be sent directly to the current active analysis, if possible. When* Data Explorer *is shown, data go first to the* Data Explorer.

Loading data

To load data, do one of the following:

Select File -> Load Data File(s), and navigate to your data;

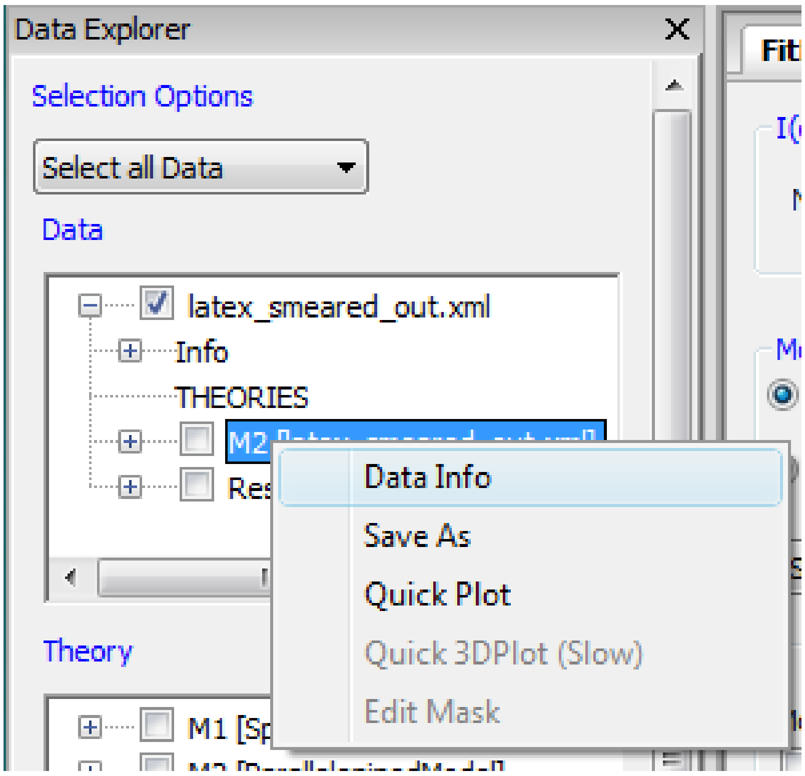
Select File -> Load Data Folder, which will attempt to load all the data in the specified folder;

Or, in the *Data Explorer* click the button *Load Data*, then select one or more (by holding down the Ctrl key) files to load into SasView.

The name of each loaded file will be listed in the *Data Explorer*. Clicking the + symbol alongside will display any available metadata read from the file.

The handy menu

Right-clicking on a loaded dataset (or model calculation, what SasView calls a 'theory') brings up a *Handy Menu* from which it is possible to access *Data Info*, *Save* the data/theory, or *Plot* the data/theory.



Activating data

To interact with data it must be activated. This is accomplished by checking the box next to the file name in the *Data Explorer*. A green tick will appear.

Unchecking/unticking a box deactivates that data set.

There is also a combo box labeled *Selection Options* from which you can activate or deactivate multiple data sets in one go.

Removing data

WARNING! Remove Data *will stop any data operations currently using the selected data sets.*

Remove Data removes all references to selected data from SasView.

Creating a new plot

Click on the *New Plot* button to create a new plot panel where the currently selected data will be plotted.

Appending plots to a graph

This operation can currently only be performed on 1D data and plot panels containing 1D data.

Click on the button *Append Plot To* to add selected data to a plot panel. Next to the button is a combo box containing the names of available plot panels. Selecting a name from this combo box will move that plot into focus.

If a plot panel is not available, the combo box and button will be disabled.

2D Data cannot be appended to any plot panels.

Freezing the theory

The *Freeze Theory* button generates data from the selected theory.

NOTE! This operation can only be performed when theory labels are selected in the Data panel.

Sending data to applications

Click on the *Send To* button to send the currently selected data to one of the available types of analysis (*Fitting*, *P(r) Inversion*, or *Invariant* calculation).

The *Single*/*Batch* mode radio buttons only apply to *Fitting*.

Batch mode provides serial (batch) fitting with one model function, that is, fitting one data set followed by another. If several data sets need to be fitted at the same time, use *Simultaneous* fitting under the *Fitting* option on the menu bar.

Note: This help document was last changed by Steve King, 01May2015

1.5.3 Plotting Data/Models

SasView generates three different types of graph window: one that displays *1D data* (i.e., $I(Q)$ vs Q), one that displays *1D residuals* (ie, the difference between the experimental data and the theory at the same Q values), and *2D color maps*.

Graph window options

Invoking the graph menu

To invoke the *Graph Menu* simply right-click on a data/theory plot, or click the *Graph Menu* (bullet list) icon in the toolbar at the bottom of the plot. Then select a menu item.

How to Hide-Show-Delete a graph

To expand a plot window, click the *Maximise* (square) icon in the top-right corner.

To shrink a plot window, click the *Restore down* (square-on-square) icon in the top-right corner.

To hide a plot, click the *Minimise* (-) icon in the top-right corner of the plot window.

To show a hidden plot, select the *Restore up* (square-on-square) icon on the minimised window.

To delete a plot, click the *Close* (x) icon in the top-right corner of the plot window.

Note: *If a residuals graph (when fitting data) is hidden, it will not show up after computation.*

Dragging a plot

Select the *Pan* (crossed arrows) icon in the toolbar at the bottom of the plot to activate this option. Move the mouse pointer to the plot. It will change to a hand. Then left-click and drag the plot around. The axis values will adjust accordingly.

To disable dragging mode, unselect the *crossed arrows* icon on the toolbar.

Zooming In-Out on a plot

Select the *Zoom* (magnifying glass) button in the toolbar at the bottom of the plot to activate this option. Move the mouse pointer to the plot. It will change to a cross-hair. Then left-click and drag the pointer around to generate a region of interest. Release the mouse button to generate the new view.

To disable zoom mode, unselect the *Zoom* button on the toolbar.

After zooming in on a a region, the *left arrow* or *right arrow* buttons on the toolbar will switch between recent views.

The axis range can also be specified manually. To do so go to the *Graph Menu* (see *Invoking_the_graph_menu* for further details), choose the *Set Graph Range* option and enter the limits in the pop box.

NOTE! If a wheel mouse is available scrolling the wheel will zoom in/out on the current plot (changing both axes). Alternatively, point at the numbers on one axis and scroll the wheel to zoom in/out on just that axis.

To return to the original view of the data, click the the *Reset* (home) icon in the toolbar at the bottom of the plot (see *Resetting_the_graph* for further details).

Saving a plot image

To save the current plot as an image file, right click on the plot to bring up the *Graph Menu* (see *Invoking_the_graph_menu*) and select *Save Image*. Alternatively, click on the *Save* (floppy disk) icon in the toolbar at the bottom of the plot.

A dialog window will open. Select a folder, enter a filename, choose an output image type, and click *Save*.

The currently supported image types are:

- EPS (encapsulated postscript)
- EMF (enhanced metafile)
- JPG/JPEG (joint photographics experts group)
- PDF (portable documant format)
- PNG (portable network graphics)
- PS (postscript)
- RAW/RGBA (bitmap, stored as 935x635 pixels of depth 8)
- SVG/SVGA (scalable vector graphics)
- TIF/TIFF (tagged iamge file)

Printing a plot

To send the current plot to a printer, click on the *Print* (printer) icon in the toolbar at the bottom of the plot.

Resetting the graph

To reset the axis range of a graph to its initial values select *Reset Graph Range* on the *Graph Menu* (see *Invoking_the_graph_menu*). Alternatively, use the *Reset* (home) icon in the toolbar at the bottom of the plot.

Modifying the graph

It is possible to make custom modifications to plots including:

- changing the plot window title
- changing the default legend location and toggling it on/off
- changing the axis label text
- changing the axis label units
- changing the axis label font & font colour
- adding/removing a text string
- adding a grid overlay

The legend and text strings can be drag and dropped around the plot

These options are accessed through the *Graph Menu* (see *Invoking_the_graph_menu*) and selecting *Modify Graph Appearance* (for axis labels, grid overlay and legend position) or *Add Text* to add textual annotations, selecting font, color, style and size. *Remove Text* will remove the last annotation added. To change the legend. *Window Title* allows a custom title to be entered instead of Graph x.

Changing scales

This menu option is only available with 1D data.

From the *Graph Menu* (see *Invoking_the_graph_menu*) select *Change Scale*. A dialog window will appear in which it is possible to choose different transformations of the x (usually Q) or y (usually I(Q)) axes, including:

- x, x^2, x^4, ln(x), log10(x), log10(x^4)
- y, 1/y, ln(y), y^2, y.(x^4), 1/sqrt(y),
- log10(y), ln(y.x), ln(y.x^2), ln(y.x^4), log10(y.x^4)

A *View* option includes short-cuts to common SAS transformations, such as:

- linear
- Guinier
- X-sectional Guinier
- Porod
- Kratky

For properly corrected and scaled data, these SAS transformations can be used to estimate, for example, Rg, rod diameter, or SANS incoherent background levels, via a linear fit (see *Making_a_linear_fit*).

Toggling scales

This menu option is only available with 2D data.

From the *Graph Menu* (see *Invoking_the_graph_menu*) select *Toggle Linear/Log Scale* to switch between a linear to log intensity scale. The type of scale selected is written alongside the colour scale.

2D color maps

This menu option is only available with 2D data.

From the *Graph Menu* (see *Invoking_the_graph_menu*) select *2D Color Map* to choose a different color scale for the image and/or change the maximum or minimum limits of the scale.

Getting data coordinates

Clicking anywhere in the plot window will cause the current coordinates to be displayed in the status bar at the very bottom-left of the SasView window.

Dataset menu options

Invoking the dataset menu

From the *Graph Menu* (see *Invoking_the_graph_menu*) highlight a plotted dataset.

Getting data info

In the *Dataset Menu* (see *Invoking_the_dataset_menu*), highlight a data set and select *DataInfo* to bring up a data information dialog panel for that data set.

Saving data

In the *Dataset Menu* (see *Invoking_the_dataset_menu*), select *Save Points as a File* (if 1D data) or *Save as a file(DAT)* (if 2D data). A save dialog will appear.

1D data can be saved in either ASCII text (.TXT) or CanSAS/SASXML (.XML) formats (see *Data Formats*).

2D data can only be saved in the NIST 2D format (.DAT) (see *Data Formats*).

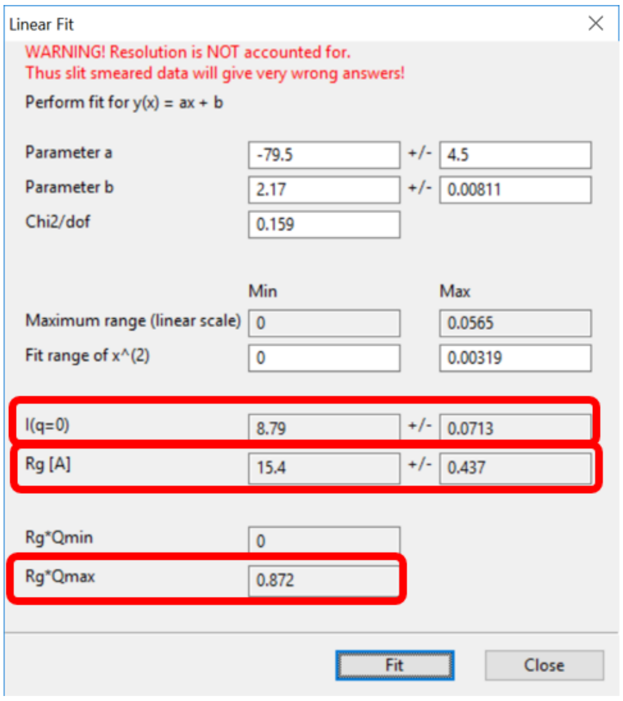
Making a linear fit

Linear fit performs a simple $y(x) = ax + b$ linear fit within the plot window.

In the *Dataset Menu* (see *Invoking_the_dataset_menu*), select *Linear Fit*. A fitting dialog will appear. Set some initial parameters and data limits and click *Fit*. The fitted parameter values are displayed and the resulting line calculated from them is added to the plot.

This option is most useful for performing simple Guinier, XS Guinier, and Porod type analyses, for example, to estimate R_g, a rod diameter, or incoherent background level, respectively.

The following figure shows an example of a Guinier analysis using this option


Linear Fit

WARNING! Resolution is NOT accounted for.
Thus slit smeared data will give very wrong answers!

Perform fit for y(x) = ax + b

Parameter a	-79.5	+/-	4.5
Parameter b	2.17	+/-	0.00811
Chi2/dof	0.159		
	Min		Max
Maximum range (linear scale)	0		0.0565
Fit range of x^(2)	0		0.00319
I(q=0)	8.79	+/-	0.0713
Rg [A]	15.4	+/-	0.437
Rg*Qmin	0		
Rg*Qmax	0.872		

Removing data from the plot

In the *Dataset Menu* (see *Invoking_the_dataset_menu*), select *Remove*. The selected data will be removed from the plot.

Note: The Remove data set action cannot be undone.

Show-Hide error bars

In the *Dataset Menu* (see *Invoking_the_dataset_menu*), select *Show Error Bar* or *Hide Error Bar* to switch between showing/hiding the errors associated with the chosen dataset.

Modify plot properties

In the *Dataset Menu* (see *Invoking_the_dataset_menu*), select *Modify Plot Property* to change the size, color, or shape of the displayed marker for the chosen dataset, or to change the dataset label that appears in the plot legend box.

2D data averaging

Purpose

This feature is only available with 2D data.

2D data averaging allows you to perform different types of averages on your data. The region to be averaged is displayed in the plot window and its limits can be modified by dragging the boundaries around.

How to average

In the *Dataset Menu* (see *Invoking_the_dataset_menu*), select one of the following averages

- Perform Circular Average
- Sector [Q view]
- Annulus [Phi view]
- Box sum
- Box averaging in Qx
- Box averaging on Qy

A 'slicer' will appear (except for *Perform Circular Average*) in the plot that you can drag by clicking on a slicer's handle. When the handle is highlighted in red, it means that the slicer can move/change size.

NOTE! The slicer size will reset if you try to select a region greater than the size of the data.

Alternatively, once a 'slicer' is active you can also select the region to average by bringing back the *Dataset Menu* and selecting *Edit Slicer Parameters and Batch Fitting*. A dialog window will appear in which you can enter values to define a region, select the number of points to plot (*nbins*), or apply the slicer to any or all other 2D data plots.

A separate plot window will also have appeared, displaying the requested average.

Note: The displayed average only updates when input focus is moved back to that window; ie, when the mouse pointer is moved onto that plot.

Selecting *Box Sum* automatically brings up the 'Slicer Parameters' dialog in order to display the average numerically, rather than graphically.

To remove a 'slicer', bring back the *Dataset menu* and select *Clear Slicer*.

Batch Slicing

A slicer can be applied to any or all existing 2D data plots using the 'Slicer Parameters' window. To open the window, select *Edit Slicer Parameters and Batch Fitting* in the *Dataset Menu* (see *Invoking_the_dataset_menu*). Batch slicing options are available at the bottom of the window.

Select the 2D plots you want to apply the slicer to. All 2D plots are selected by default. The resulting 1D data for all slicers can be saved as a text file and then sent to fitting by selecting the *Auto save generated 1D* check box. Sending data to the fitting perspective requires the data be saved.

Once the auto save check box is selected, you can select where the files are saved. The file name for the saved data is the slicer name plus the file name of the original data set, plus what is in the *Append to file name* field. The default value in the append to field includes the names and values for all of the slicer parameters.

The batch of slices can be sent to fitting if desired, with three options available. The first is to not fit the data, the second is to send the slices to individual fit pages, and the third is to send all sliced data to a single batch fit window.

Clicking *Apply Slicer to Selected Plots* will create a slicer for each selected plot with the parameters entered in the 'Slicer Parameters' window. Depending on the options selected the data may then be saved, loaded as separate data sets in the data manager panel, and finally sent to fitting.

Unmasked circular average

This operation will perform an average in constant Q rings around the (x,y) pixel location of the beam center.

Masked circular average

This operation is the same as 'Unmasked Circular Average' except that any masked region is excluded.

Sector average [Q View]

This operation averages in constant Q arcs.

The width of the sector is specified in degrees ($\pm\delta|\phi|$) each side of the central angle ϕ.

Annular average [ϕ]

This operation performs an average between two Q values centered on (0,0), and averaged over a specified number of pixels.

The data is returned as a function of angle ϕ in degrees with zero degrees at the 3 O'clock position.

Box sum

This operation performs a sum of counts in a 2D region of interest.

When editing the slicer parameters, the user can enter the length and the width the rectangular slicer and the coordinates of the center of the rectangle.

Box Averaging in Qx

This operation computes an average $I(Q_x)$ for the region of interest.

When editing the slicer parameters, the user can control the length and the width the rectangular slicer. The averaged output is calculated from constant bins with rectangular shape. The resultant Q values are nominal values, that is, the central value of each bin on the x-axis.

Box Averaging in Qy

This operation computes an average $I(Q_y)$ for the region of interest.

When editing the slicer parameters, the user can control the length and the width the rectangular slicer. The averaged output is calculated from constant bins with rectangular shape. The resultant Q values are nominal values, that is, the central value of each bin on the x-axis.

Note: This help document was last modified by Paul Butler, 05 September, 2016

1.5.4 Test Data

Test data sets are included as a convenience to our users. Look in the test sub-folder in your SasView installation folder.

The test data sets are organized based on their data structure:

- *1D data*
- *convertible 1D data files*
- *2D data*

- *coordinate data*
- *image data*
- *SESANS data*
- *save states*
- *upcoming formats*

1D Data

1D data sets EITHER have:

- at least two columns of data with I(Q) (assumed to be in absolute units) on the y-axis and Q on the x-axis. And additional columns of data may carry uncertainty data, resolution data, or other metadata.

OR:

- the I(Q) and Q data in separate files *with no other information*.

Data in the latter format need to be converted to a single file format with the *File Converter Tool* before they can be analysed in SasView. Test files are located in the /convertible_files folder.

1D Test Data

33837rear_1D_1.75_16.5

- Data from a magnetically-oriented surfactant liquid crystal output by the Mantid framework. The data was collected on the SANS2D instrument at ISIS.

10wtAOT_Reline_120_reduced / Anton-Paar / saxsess_example

- Data from Anton-Paar SAXSess instruments saved in Otto Glatter's PDH format.

AOT_Microemulsion

- Aerosol-OT surfactant stabilised oil-in-water microemulsion data at three contrasts: core (oil core), drop (oil core + surfactant layer), and shell (surfactant layer).
- Suitable for testing simultaneous fitting.

APS_DND-CAT

- ASCII data from the DND-CAT beamline at the APS.

hSDS_D2O

- h25-sodium dodecyl sulphate solutions at two concentrations: 0.5wt% (just above the cmc), 2wt% (well above the cmc), and 2wt% but with 0.2mM NaCl electrolyte.
- Suitable for testing charged S(Q) models.

ISIS_83404 / ISIS_98929

- Polyamide-6 fibres hydrated in D2O exhibiting a broad lamellar peak from the semi-crystalline nanostructure.
- This is the *same data* as that in the BSL/OTOKO Z8300* / Z9800* files but in an amalgamated ASCII format!
- Suitable for testing *Correlation Function Analysis* .

ISIS_Polymer_Blend_TK49

- Monodisperse (Mw/Mn~1.02) 49wt% d8-polystyrene : 51wt% h8-polystyrene polymer blend.
- Suitable for testing Poly_GaussCoil and RPA10 models.

P123_D2O

- Lyotropic liquid crystalline solutions of non-ionic ABA block copolymer Pluronic P123 in water at three concentrations: 10wt%, 30wt%, and 40wt%.
- Suitable for testing paracrystal models.

Convertible 1D Data

APS_X / APS_Y

- ASCII data output by a reduction software package at the APS.
- Suitable for testing the *File Converter Tool* .

FIT2D_I / FIT2D_Q

- ASCII data output by the FIT2D software package at the ESRF.
- Suitable for testing the *File Converter Tool* .

Z8300*.I1D / Z8300*.QAX / Z9800*.I1D / Z9800*.QAX

- BSL/OTOKO data from polyamide-6 fibres hydrated in D2O exhibiting a broad lamellar peak from the semi-crystalline nanostructure.
- This is the *same data* as that in ISIS_83404 / ISIS_98929 but in an older separated format!
- Suitable for testing the *File Converter Tool* .
- Suitable for testing *Correlation Function Analysis* .

2D Data

2D data sets are data sets that give the reduced intensity for each Qx-Qy bin. Depending on the file format, uncertainty data and metadata may also be available.

2D Test Data

33837rear_2D_1.75_16.5

- Data from a magnetically-oriented surfactant liquid crystal output by the Mantid framework. The data was collected on the SANS2D instrument at ISIS.

P123_D2O

- Lyotropic liquid crystalline solutions of non-ionic ABA block copolymer Pluronic P123 in water at three concentrations: 10wt%, 30wt%, and 40wt%.
- Suitable for testing paracrystal models.

Coordinate Data

Coordinate data sets, such as PDB or OMF files, and which describe a specific structure, are designed to be read and viewed in the *Generic SANS Calculator Tool* .

Coordinate Test Data

A_Raw_Example-1

- OMF format data file from a simulation of magnetic spheres.

diamond

- PDB format data file for diamond.

dna

- PDB format data file for DNA.

sld_file

- Example SLD format data file.

Image Data

Image data sets are designed to be read by the *Image Viewer Tool* . They can be converted into synthetic 2D data.

Image Test Data

ISIS_98940

- Polyamide-6 fibres hydrated in D2O exhibiting a broad lamellar peak from the semi-crystalline nanostructure.
- Data is presented in Windows Bitmap (BMP), GIF, JPEG (JPG), PNG, and TIFF (TIF) formats.

SESANS Data

SESANS (Spin-Echo SANS) data sets primarily contain the neutron polarisation as a function of the spin-echo length. Also see *SANS to SESANS conversion* .

SESANS Test Data

spheres2micron

- SESANS data from 2 micron polystyrene spheres in 53% H2O / 47% D2O.

Save States

Saved states are projects and analyses saved by the SasView program. A single analysis file contains the data and parameters for a single fit (.fit), p(r) inversion (.prv), or invariant calculation (.inv). A project file (.svs) contains the results for every active analysis in a SasView session.

Saved State Test Data

fitstate.fitv

- a saved fitting analysis.

test.inv

- a saved invariant analysis.

test002.inv

- a saved invariant analysis.

prstate.prv

- a saved P(r) analysis.

newone.svs

- a saved SasView project.

Upcoming Formats

Data in this folder are in formats that are not yet implemented in SasView but which might be in future versions of the program.

Other Test Data

phi_weights.txt

radius_dist.txt

THETA_weights.txt

Note: This help document was last changed by Steve King, 06Oct2016

1.5.5 Tutorials

Note: In Windows use [Alt]-[Cursor left] to return to the previous page

The following tutorial was written for Version 2.x but is still a useful overview of much of the analysis capability in SasView (but disregard Appendix XIII if using Version 4.2 or later)

`Old Tutorial`

The following tutorials have been written for Version 4.x

`Getting Started with Sasview`

`Basic 1D Fitting in Sasview`

`Simultaneous 1D Fitting in Sasview`

`Correlation Function Analysis in SasView`

1.5.6 Environment Variables

SasView creates and uses a number of environment variables:

- **SAS_MODELPATH=path**
  - sets the directory containing custom models
- **SAS_DLL_PATH=path**
  - sets the path to the compiled modules
- **SAS_WEIGHTS_PATH=~/.sasview/weights**
  - sets the path to custom distribution files (see *Polydispersity & Orientational Distributions*)
- **XDG_CACHE_HOME=~/.cache**
  - sets the pyopencl cache root (linux only)
  - defined in the appdirs package
- **SAS_COMPILER=tinycc|msvc|mingw|unix**
  - sets the DLL compiler
- **SAS_OPENCL=vendor:device|none**
  - sets the target OpenCL device for GPU computations

    - use *none* to disable

- **SAS_OPENMP=1|0**
    - turns on/off OpenMP multi-processing of the DLLs

Document History

2018-07-20 Steve King

1.5.7 Model Marketplace

The Model Marketplace allows members of the SAS Community to contribute plug-in fitting models for *SasView* for all to use.

Note: These plug-in models require SasView version 4.0 or later. However, not *all* models may work with *every* version of SasView because of changes to our API.

Contributed models should be written in Python (only version 2.7.x is currently supported) or, if computational speed is an issue, in a combination of Python and C. You only need to upload the .py/.c source code files to the Marketplace!

Please put a comment in your code to indicate which version of SasView you wrote the model for.

For guidance on how to write a plugin model see *Writing a Plugin Model* . It may also be helpful to examine the library models in the */sasmodels-data/models* sub-folder of your SasView installation directory.

The Marketplace also provides the option to upload a SasView text file of the scattering function data computed by your model. If you do this a graph of the scattering function will appear under the Marketplace entry for your model.

Note: The SasView Development Team regret to say that they do not have the resources to fix the bugs or polish the code in every model contributed to the Marketplace!

CHAPTER

TWO

DEVELOPER DOCUMENTATION

Note: In Windows use [Alt]-[Cursor left] to return to the previous page

2.1 Contents

2.1.1 SasView

sas package

Subpackages

sas.sascalc package

Subpackages

sas.sascalc.calculator package

Submodules

sas.sascalc.calculator.BaseComponent module

Provide base functionality for all model components

class `sas.sascalc.calculator.BaseComponent.`**`BaseComponent`**

Basic model component

Since version 0.5.0, basic operations are no longer supported.

`calculate_ER``()`

Calculate effective radius

`calculate_VR``()`

Calculate volume fraction ratio

`clone``()`

Returns a new object identical to the current object

`evalDistribution``(`*qdist*`)`

Evaluate a distribution of q-values.

- For 1D, a numpy array is expected as input:

```
evalDistribution(q)
```

where q is a numpy array.

- For 2D, a list of numpy arrays are expected: [qx_prime,qy_prime], where 1D arrays,

  ```
  qx_prime = [ qx[0], qx[1], qx[2], ....]
  ```

  and

  ```
  qy_prime = [ qy[0], qy[1], qy[2], ....]
  ```

Then get

```
q = np.sqrt(qx_prime^2+qy_prime^2)
```

that is a qr in 1D array;

```
q = [q[0], q[1], q[2], ....]
```

Note: Due to 2D speed issue, no anisotropic scattering is supported for python models, thus C-models should have their own evalDistribution methods.

The method is then called the following way:

```
evalDistribution(q)
```

where q is a numpy array.

Parameters **`qdist`** – ndarray of scalar q-values or list [qx,qy] where qx,qy are 1D ndarrays

`getDispParamList`()
Return a list of all available parameters for the model

`getParam`(*name*)
Set the value of a model parameter :param name: name of the parameter

`getParamList`()
Return a list of all available parameters for the model

`getParamListWithToken`(*token*, *member*)
get Param List With Token

`getParamWithToken`(*name*, *token*, *member*)
get Param With Token

`getProfile`()
Get SLD profile

: return: (z, beta) where z is a list of depth of the transition points beta is a list of the corresponding SLD values

`is_fittable`(*par_name*)
Check if a given parameter is fittable or not

Parameters **`par_name`** – the parameter name to check

`run`(*x*)
run 1d

`runXY`(*x*)
run 2d

`setParam`(*name*, *value*)
Set the value of a model parameter

Parameters

- **name** – name of the parameter
- **value** – value of the parameter

setParamWithToken(*name*, *value*, *token*, *member*)
set Param With Token

set_dispersion(*parameter*, *dispersion*)
model dispersions

sas.sascalc.calculator.instrument module

This module is a small tool to allow user to control instrumental parameters

class sas.sascalc.calculator.instrument.**Aperture**
Bases: `object`

An object class that defines the aperture variables

set_sample_distance(*distance=[]*)
Set the sample aperture distance

set_sample_size(*size=[]*)
Set the sample aperture size

set_source_size(*size=[]*)
Set the source aperture size

class sas.sascalc.calculator.instrument.**Detector**
Bases: `object`

An object class that defines the detector variables

set_distance(*distance=[]*)
Set the detector distance

set_pix_size(*size=[]*)
Set the detector pix_size

set_size(*size=[]*)
Set the detector size

class sas.sascalc.calculator.instrument.**Neutron**
Bases: `object`

An object that defines the wavelength variables

get_band()
To get the wavelength band

get_default_spectrum()
get default spectrum

get_intensity()
To get the value of intensity

get_mass()
To get the neutron mass

get_ramdom_value()
To get the value of wave length

get_spectrum()
To get the wavelength spectrum

get_wavelength()
To get the value of wavelength

get_wavelength_spread()
To get the value of wavelength spread

plot_spectrum()
To plot the wavelength spactrum : requirement: matplotlib.pyplot

set_band(*band=[]*)
To set the wavelength band

Parameters **band** – array of [min, max]

set_full_band()
set band to default value

set_intensity(*intensity=368428*)
Sets the intensity in counts/sec

set_mass(*mass=1.67492729e-24*)
Sets the wavelength

set_spectrum(*spectrum*)
Set spectrum

Parameters **spectrum** – numpy array

set_wavelength(*wavelength=6.0*)
Sets the wavelength

set_wavelength_spread(*spread=0.125*)
Sets the wavelength spread

setup_spectrum()
To set the wavelength spectrum, and intensity, assumes wavelength is already within the spectrum

class sas.sascalc.calculator.instrument.**Sample**
Bases: object

An object class that defines the sample variables

set_distance(*distance=[]*)
Set the sample distance

set_size(*size=[]*)
Set the sample size

set_thickness(*thickness=0.0*)
Set the sample thickness

class sas.sascalc.calculator.instrument.**TOF**
Bases: *sas.sascalc.calculator.instrument.Neutron*

TOF: make list of wavelength and wave length spreads

get_intensity_list()
get list of the intensity wrt wavelength_list

get_wave_list()
Get wavelength and wavelength_spread list

set_wave_list(*wavelength=[]*)
Set wavelength list

Parameters **wavelength** – list of wavelengths

set_wave_spread_list(*wavelength_spread=[]*)
Set wavelength_spread list

Parameters **wavelength_spread** – list of wavelength spreads

sas.sascalc.calculator.instrument.**validate**(*value=None*)
Check if the value is folat > 0.0

Return value True / False

sas.sascalc.calculator.kiessig_calculator module

This module is a small tool to allow user to quickly determine the size value in real space from the fringe width in q space.

class `sas.sascalc.calculator.kiessig_calculator.`**`KiessigThicknessCalculator`**

Bases: `object`

compute thickness from the fringe width of data

`compute_thickness``()`

Calculate thickness.

Returns the thickness.

`get_deltaq``()`

return deltaQ value in 1/A unit

`get_thickness_unit``()`

Returns the thickness unit.

`set_deltaq``(`*dq=None*`)`

Receive deltaQ value

Parameters **dq** – q fringe width in 1/A unit

sas.sascalc.calculator.resolution_calculator module

This object is a small tool to allow user to quickly determine the variance in q from the instrumental parameters.

class `sas.sascalc.calculator.resolution_calculator.`**`ResolutionCalculator`**

Bases: `object`

compute resolution in 2D

`compute``(`*wavelength*, *wavelength_spread*, *qx_value*, *qy_value*, *coord='cartesian'*, *tof=False*`)`

Compute the Q resoltuion in || and + direction of 2D : qx_value: x component of q : qy_value: y component of q

`compute_and_plot``(`*qx_value*, *qy_value*, *qx_min*, *qx_max*, *qy_min*, *qy_max*, *coord='cartesian'*`)`

Compute the resolution : qx_value: x component of q : qy_value: y component of q

`get_all_instrument_params``()`

Get all instrumental parameters

`get_default_spectrum``()`

Get default_spectrum

`get_detector_pix_size``()`

Get detector pixel size

`get_detector_qrange``()`

get max detector q ranges

: return: qx_min, qx_max, qy_min, qy_max tuple

`get_detector_size``()`

Get detector size

`get_image``(`*qx_value*, *qy_value*, *sigma_1*, *sigma_2*, *sigma_r*, *qx_min*, *qx_max*, *qy_min*, *qy_max*, *coord='cartesian'*, *full_cal=True*`)`

Get the resolution in polar coordinate ready to plot : qx_value: qx_value value : qy_value: qy_value

value : sigma_1: variance in r direction : sigma_2: variance in phi direction : coord: coordinate system of image, ‘polar’ or ‘cartesian’

get_intensity()
Get intensity

get_intensity_list()
Set wavelength spread

get_neutron_mass()
Get Neutron mass

get_sample2detector_distance()
Get detector sample2detector_distance

get_sample2sample_distance()
Get detector sampleslitsample_distance

get_sample_aperture_size()
Get sample aperture size

get_source2sample_distance()
Get detector source2sample_distance

get_source_aperture_size()
Get source aperture size

get_spectrum()
Get _spectrum

get_variance(*size=[]*, *distance=0*, *phi=0*, *comp='radial'*)
Get the variance when the slit/pinhole size is given : size: list that can be one(diameter for circular) or two components(lengths for rectangular) : distance: [z, x] where z along the incident beam, x // qx_value : comp: direction of the sigma; can be ‘phi’, ‘y’, ‘x’, and ‘radial’

: return variance: sigma^2

get_variance_gravity(*s_distance*, *d_distance*, *wavelength*, *spread*, *phi*, *comp='radial'*, *switch='on'*)
Get the variance from gravity when the wavelength spread is given

: s_distance: source to sample distance : d_distance: sample to detector distance : wavelength: wavelength : spread: wavelength spread (ratio) : comp: direction of the sigma; can be ‘phi’, ‘y’, ‘x’, and ‘radial’

: return variance: sigma^2

get_variance_wave(*A_value*, *radius*, *distance*, *spread*, *phi*, *comp='radial'*, *switch='on'*)
Get the variance when the wavelength spread is given

: radius: the radial distance from the beam center to the pix of q : distance: sample to detector distance : spread: wavelength spread (ratio) : comp: direction of the sigma; can be ‘phi’, ‘y’, ‘x’, and ‘radial’

: return variance: sigma^2 for 2d, sigma^2 for 1d [tuple]

get_wave_list()
Set wavelength spread

get_wavelength()
Get wavelength

get_wavelength_spread()
Get wavelength spread

plot_image(*image*)
Plot image using pyplot : image: 2d resolution image

: return plt: pylab object

reset_image()
Reset image to default (=[])

set_detector_pix_size(*size*)
Set detector pixel size

set_detector_size(*size*)
Set detector size in number of pixels : param size: [pixel_nums] or [x_pix_num, yx_pix_num]

set_intensity(*intensity*)
Set intensity

set_neutron_mass(*mass*)
Set Neutron mass

set_sample2detector_distance(*distance*)
Set detector sample2detector_distance

: param distance: [distance, x_offset]

set_sample2sample_distance(*distance*)
Set detector sample_slit2sample_distance

: param distance: [distance, x_offset]

set_sample_aperture_size(*size*)
Set sample aperture size

: param size: [dia_value] or [xheight_value, yheight_value]

set_source2sample_distance(*distance*)
Set detector source2sample_distance

: param distance: [distance, x_offset]

set_source_aperture_size(*size*)
Set source aperture size

: param size: [dia_value] or [x_value, y_value]

set_spectrum(*spectrum*)
Set spectrum

set_wave(*wavelength*)
Set wavelength list or wavelength

set_wave_list(*wavelength_list*, *wavelengthspread_list*)
Set wavelength and its spread list

set_wave_spread(*wavelength_spread*)
Set wavelength spread or wavelength spread

set_wavelength(*wavelength*)
Set wavelength

set_wavelength_spread(*wavelength_spread*)
Set wavelength spread

setup_tof(*wavelength*, *wavelength_spread*)
Setup all parameters in instrument

: param ind: index of lambda, etc

sas.sascalc.calculator.sas_gen module

SAS generic computation and sld file readers

class sas.sascalc.calculator.sas_gen.**GenSAS**

Bases: *sas.sascalc.calculator.BaseComponent.BaseComponent*

Generic SAS computation Model based on sld (n & m) arrays

evalDistribution (*qdist*)

Evaluate a distribution of q-values.

Parameters **qdist** – ndarray of scalar q-values (for 1D) or list [qx,qy] where qx,qy are 1D ndarrays (for 2D).

getProfile ()

Get SLD profile : return: sld_data

run (*x=0.0*)

Evaluate the model :param x: simple value :return: (I value)

runXY (*x=0.0*)

Evaluate the model :param x: simple value :return: I value :Use this runXY() for the computation

set_is_avg (*is_avg=False*)

Sets is_avg: [bool]

set_pixel_volumes (*volume*)

Set the volume of a pixel in (A^3) unit :Param volume: pixel volume [float]

set_sld_data (*sld_data=None*)

Sets sld_data

class sas.sascalc.calculator.sas_gen.**MagSLD** (*pos_x*, *pos_y*, *pos_z*, *sld_n=None*, *sld_mx=None*, *sld_my=None*, *sld_mz=None*, *vol_pix=None*)

Bases: object

Magnetic SLD.

get_sldn ()

Returns nuclear sld

pos_x = None

pos_y = None

pos_z = None

set_conect_lines (*line_x*, *line_y*, *line_z*)

Set bonding line data if taken from pdb

set_nodes ()

Set xnodes, ynodes, and znodes

set_pix_type (*pix_type*)

Set pixel type :Param pix_type: string, 'pixel' or 'atom'

set_pixel_symbols (*symbol='pixel'*)

Set pixel :Params pixel: str; pixel or atomic symbol, or array of strings

set_pixel_volumes (*vol*)

Set pixel volumes :Params pixel: str; pixel or atomic symbol, or array of strings

set_sldms (*sld_mx*, *sld_my*, *sld_mz*)

Sets mx, my, mz and abs(m).

set_sldn (*sld_n*)

Sets neutron SLD

set_stepsize ()

Set xtepsize, ystepsize, and zstepsize

sld_mx = None

sld_my = None

sld_mz = None

sld_n = None

class `sas.sascalc.calculator.sas_gen.`**OMF2SLD**

Bases: `object`

Convert OMFData to MAgData

get_magsld()
: return MagSLD

get_omfdata()
: Return all data

get_output()
: Return output

remove_null_points(*remove=False*, *recenter=False*)
: Removes any mx, my, and mz = 0 points

set_data(*omfdata*, *shape='rectangular'*)
: Set all data

class `sas.sascalc.calculator.sas_gen.`**OMFData**

Bases: `object`

OMF Data.

set_m(*mx*, *my*, *mz*)
: Set the Mx, My, Mz values

class `sas.sascalc.calculator.sas_gen.`**OMFReader**

Bases: `object`

Class to load omf/ascii files (3 columns w/header).

ext = ['.omf', '.OMF']

read(*path*)
: Load data file :param path: file path :return: x, y, z, sld_n, sld_mx, sld_my, sld_mz

type = ['OMF files (*.OMF, *.omf)|*.omf']

type_name = 'OMF ASCII'

class `sas.sascalc.calculator.sas_gen.`**PDBReader**

Bases: `object`

PDB reader class: limited for reading the lines starting with 'ATOM'

ext = ['.pdb', '.PDB']

read(*path*)
: Load data file

Parameters `path` – file path

Returns MagSLD

Raises `RuntimeError` – when the file can't be opened

type = ['pdb files (*.PDB, *.pdb)|*.pdb']

type_name = 'PDB'

write(*path*, *data*)
: Write

class sas.sascalc.calculator.sas_gen.**SLDReader**

Bases: object

Class to load ascii files (7 columns).

ext = ['.sld', '.SLD', '.txt', '.TXT', '.*']

read(*path*)

Load data file :param path: file path :return MagSLD: x, y, z, sld_n, sld_mx, sld_my, sld_mz :raise RuntimeError: when the file can't be opened :raise ValueError: when the length of the data vectors are inconsistent

type = ['sld files (*.SLD, *.sld)|*.sld', 'txt files (*.TXT, *.txt)|*.txt', 'all fi

type_name = 'SLD ASCII'

write(*path, data*)

Write sld file :Param path: file path :Param data: MagSLD data object

sas.sascalc.calculator.sas_gen.**decode**(*s*)

sas.sascalc.calculator.sas_gen.**mag2sld**(*mag, v_unit=None*)

Convert magnetization to magnatic SLD sldm = Dm * mag where Dm = gamma * classical elec. radius/(2*Bohr magneton) Dm ~ 2.853E-12 [A^(-2)] ==> Shouldn't be 2.90636E-12 [A^(-2)]???

sas.sascalc.calculator.sas_gen.**test**()

Test code

sas.sascalc.calculator.sas_gen.**test_load**()

Test code

sas.sascalc.calculator.sas_gen.**test_save**()

sas.sascalc.calculator.sas_gen.**transform_center**(*pos_x, pos_y, pos_z*)

re-center :return: posx, posy, posz [arrays]

sas.sascalc.calculator.slit_length_calculator module

This module is a small tool to allow user to quickly determine the slit length value of data.

class sas.sascalc.calculator.slit_length_calculator.**SlitlengthCalculator**

Bases: object

compute slit length from SAXSess beam profile (1st col. Q , 2nd col. I , and 3rd col. dI.: don't need the 3rd)

calculate_slit_length()

Calculate slit length.

Returns the slit length calculated value.

get_slit_length_unit()

Returns the slit length unit.

set_data(*x=None, y=None*)

Receive two vector x, y and prepare the slit calculator for computation.

Parameters

- **x** – array
- **y** – array

Module contents

sas.sascalc.corfunc package

Submodules

sas.sascalc.corfunc.corfunc_calculator module

This module implements corfunc

class sas.sascalc.corfunc.corfunc_calculator.**CorfuncCalculator**(*data=None, lowerq=None, upperq=None, scale=1*)

Bases: object

compute_background(*upperq=None*)
Compute the background level from the Porod region of the data

compute_extrapolation()
Extrapolate and interpolate scattering data

Returns The extrapolated data

compute_transform(*extrapolation, trans_type, background=None, completefn=None, updatefn=None*)
Transform an extrapolated scattering curve into a correlation function.

Parameters

- **extrapolation** – The extrapolated data
- **background** – The background value (if not provided, previously calculated value will be used)
- **extrap_fn** – A callable function representing the extraoplated data
- **completefn** – The function to call when the transform calculation is complete
- **updatefn** – The function to call to update the GUI with the status of the transform calculation

Returns The transformed data

extract_parameters(*transformed_data*)
Extract the interesting measurements from a correlation function

Parameters **transformed_data** – Fourier transformation of the extrapolated data

set_data(*data, scale=1*)
Prepares the data for analysis

Returns new_data = data * scale - background

stop_transform()

transform_isrunning()

sas.sascalc.corfunc.transform_thread module

class sas.sascalc.corfunc.transform_thread.**FourierThread**(*raw_data*, *extrapolated_data*, *bg*, *updatefn=None*, *completefn=None*)

Bases: *sas.sascalc.data_util.calcthread.CalcThread*

check_if_cancelled()

compute()

class sas.sascalc.corfunc.transform_thread.**HilbertThread**(*raw_data*, *extrapolated_data*, *bg*, *updatefn=None*, *completefn=None*)

Bases: *sas.sascalc.data_util.calcthread.CalcThread*

compute()

Module contents

sas.sascalc.data_util package

Submodules

sas.sascalc.data_util.calcthread module

class sas.sascalc.data_util.calcthread.**CalcCommandline**(*n=20000*)

Test method

complete(*total=0.0*)

update(*i=0*)

class sas.sascalc.data_util.calcthread.**CalcDemo**(*completefn=None*, *updatefn=None*, *yieldtime=0.01*, *worktime=0.01*, *exception_handler=None*)

Bases: *sas.sascalc.data_util.calcthread.CalcThread*

Example of a calculation thread.

compute(*n*)

class sas.sascalc.data_util.calcthread.**CalcThread**(*completefn=None*, *updatefn=None*, *yieldtime=0.01*, *worktime=0.01*, *exception_handler=None*)

Threaded calculation class. Inherit from here and specialize the compute() method to perform the appropriate operations for the class.

If you specialize the __init__ method be sure to call CalcThread.__init__, passing it the keyword arguments for yieldtime, worktime, update and complete.

When defining the compute() method you need to include code which allows the GUI to run. They are as follows:

```
self.isquit()            # call frequently to check for interrupts
self.update(kw=...)      # call when the GUI could be updated
self.complete(kw=...)    # call before exiting compute()
```

The update() and complete() calls accept field=value keyword arguments which are passed to the called function. complete() should be called before exiting the GUI function. A KeyboardInterrupt event is triggered if the GUI signals that the computation should be halted.

The following documentation should be included in the description of the derived class.

The user of this class will call the following:

```
thread = Work(...,kw=...)   # prepare the work thread.
thread.queue(...,kw=...)    # queue a work unit
thread.requeue(...,kw=...)  # replace work unit on the end of queue
thread.reset(...,kw=...)    # reset the queue to the given work unit
thread.stop()               # clear the queue and halt
thread.interrupt()          # halt the current work unit but continue
thread.ready(delay=0.)      # request an update signal after delay
thread.isrunning()          # returns true if compute() is running
```

Use queue() when all work must be done. Use requeue() when intermediate work items don't need to be done (e.g., in response to a mouse move event). Use reset() when the current item doesn't need to be completed before the new event (e.g., in response to a mouse release event). Use stop() to halt the current and pending computations (e.g., in response to a stop button).

The methods queue(), requeue() and reset() are proxies for the compute() method in the subclass. Look there for a description of the arguments. The compute() method can be called directly to run the computation in the main thread, but it should not be called if isrunning() returns true.

The constructor accepts additional keywords yieldtime=0.01 and worktime=0.01 which determine the cooperative multitasking behaviour. Yield time is the duration of the sleep period required to give other processes a chance to run. Work time is the duration between sleep periods.

Notifying the GUI thread of work in progress and work complete is done with updatefn=updatefn and completefn=completefn arguments to the constructor. Details of the parameters to the functions depend on the particular calculation class, but they will all be passed as keyword arguments. Details of how the functions should be implemented vary from framework to framework.

For wx, something like the following is needed:

```
import wx, wx.lib.newevent
(CalcCompleteEvent, EVT_CALC_COMPLETE) = wx.lib.newevent.NewEvent()

# methods in the main window class of your application
def __init__():
    ...
    # Prepare the calculation in the GUI thread.
    self.work = Work(completefn=self.CalcComplete)
    self.Bind(EVT_CALC_COMPLETE, self.OnCalcComplete)
    ...
    # Bind work queue to a menu event.
    self.Bind(wx.EVT_MENU, self.OnCalcStart, id=idCALCSTART)
    ...

def OnCalcStart(self,event):
    # Start the work thread from the GUI thread.
    self.work.queue(...work unit parameters...)

def CalcComplete(self,**kwargs):
    # Generate CalcComplete event in the calculation thread.
    # kwargs contains field1, field2, etc. as defined by
    # the Work thread class.
    event = CalcCompleteEvent(**kwargs)
    wx.PostEvent(self, event)

def OnCalcComplete(self,event):
    # Process CalcComplete event in GUI thread.
```

(continues on next page)

(continued from previous page)

```
    # Use values from event.field1, event.field2 etc. as
    # defined by the Work thread class to show the results.
    ...
```

complete(***kwargs*)
Update the GUI with the completed results from a work unit.

compute(**args*, ***kwargs*)
Perform a work unit. The subclass will provide details of the arguments.

exception()
An exception occurred during computation, so call the exception handler if there is one. If not, then log the exception and continue.

interrupt()
Stop the current work item. To clear the work queue as well call the stop() method.

isquit()
Check for interrupts. Should be called frequently to provide user responsiveness. Also yields to other running threads, which is required for good performance on OS X.

isrunning()

queue(**args*, ***kwargs*)
Add a work unit to the end of the queue. See the compute() method for details of the arguments to the work unit.

ready(*delay=0.0*)
Ready for another update after delay=t seconds. Call this for threads which can show intermediate results from long calculations.

requeue(**args*, ***kwargs*)
Replace the work unit on the end of the queue. See the compute() method for details of the arguments to the work unit.

reset(**args*, ***kwargs*)
Clear the queue and start a new work unit. See the compute() method for details of the arguments to the work unit.

stop()
Clear the queue and stop the thread. New items may be queued after stop. To stop just the current work item, and continue the rest of the queue call the interrupt method

update(***kwargs*)
Update GUI with the lastest results from the current work unit.

sas.sascalc.data_util.err1d module

Error propogation algorithms for simple arithmetic

Warning: like the underlying numpy library, the inplace operations may return values of the wrong type if some of the arguments are integers, so be sure to create them with floating point inputs.

sas.sascalc.data_util.err1d.**add**(*X*, *varX*, *Y*, *varY*)
Addition with error propagation

sas.sascalc.data_util.err1d.**add_inplace**(*X*, *varX*, *Y*, *varY*)
In-place addition with error propagation

sas.sascalc.data_util.err1d.**div**(*X*, *varX*, *Y*, *varY*)
Division with error propagation

sas.sascalc.data_util.err1d.**div_inplace**(*X*, *varX*, *Y*, *varY*)
In-place division with error propagation

`sas.sascalc.data_util.err1d.`**`exp`**(*X*, *varX*)
    Exponentiation with error propagation

`sas.sascalc.data_util.err1d.`**`log`**(*X*, *varX*)
    Logarithm with error propagation

`sas.sascalc.data_util.err1d.`**`mul`**(*X*, *varX*, *Y*, *varY*)
    Multiplication with error propagation

`sas.sascalc.data_util.err1d.`**`mul_inplace`**(*X*, *varX*, *Y*, *varY*)
    In-place multiplication with error propagation

`sas.sascalc.data_util.err1d.`**`pow`**(*X*, *varX*, *n*)
    X**n with error propagation

`sas.sascalc.data_util.err1d.`**`pow_inplace`**(*X*, *varX*, *n*)
    In-place X**n with error propagation

`sas.sascalc.data_util.err1d.`**`sub`**(*X*, *varX*, *Y*, *varY*)
    Subtraction with error propagation

`sas.sascalc.data_util.err1d.`**`sub_inplace`**(*X*, *varX*, *Y*, *varY*)
    In-place subtraction with error propagation

sas.sascalc.data_util.formatnum module

Format values and uncertainties nicely for printing.

`format_uncertainty_pm()` produces the expanded format v +/- err.

`format_uncertainty_compact()` produces the compact format v(##), where the number in parenthesis is the uncertainty in the last two digits of v.

`format_uncertainty()` uses the compact format by default, but this can be changed to use the expanded +/- format by setting format_uncertainty.compact to False.

The formatted string uses only the number of digits warranted by the uncertainty in the measurement.

If the uncertainty is 0 or not otherwise provided, the simple %g floating point format option is used.

Infinite and indefinite numbers are represented as inf and NaN.

Example:

```
>>> v,dv = 757.2356,0.01032
>>> print format_uncertainty_pm(v,dv)
757.236 +/- 0.010
>>> print format_uncertainty_compact(v,dv)
757.236(10)
>>> print format_uncertainty(v,dv)
757.236(10)
>>> format_uncertainty.compact = False
>>> print format_uncertainty(v,dv)
757.236 +/- 0.010
```

UncertaintyFormatter() returns a private formatter with its own formatter.compact flag.

`sas.sascalc.data_util.formatnum.`**`format_uncertainty_pm`**(*value*, *uncertainty*)
    Given *value* v and *uncertainty* dv, return a string v +/- dv.

`sas.sascalc.data_util.formatnum.`**`format_uncertainty_compact`**(*value*, *uncertainty*)
    Given *value* v and *uncertainty* dv, return the compact representation v(##), where ## are the first two digits of the uncertainty.

sas.sascalc.data_util.nxsunit module

Define unit conversion support for NeXus style units.

The unit format is somewhat complicated. There are variant spellings and incorrect capitalization to worry about, as well as forms such as "mili*metre" and "1e-7 seconds".

This is a minimal implementation of units including only what I happen to need now. It does not support the complete dimensional analysis provided by the package udunits on which NeXus is based, or even the units used in the NeXus definition files.

Unlike other units packages, this package does not carry the units along with the value but merely provides a conversion function for transforming values.

Usage example:

```
import nxsunit
u = nxsunit.Converter('mili*metre')  # Units stored in mm
v = u(3000,'m')  # Convert the value 3000 mm into meters
```

NeXus example:

```
# Load sample orientation in radians regardless of how it is stored.
# 1. Open the path
file.openpath('/entry1/sample/sample_orientation')
# 2. scan the attributes, retrieving 'units'
units = [for attr,value in file.attrs() if attr == 'units']
# 3. set up the converter (assumes that units actually exists)
u = nxsunit.Converter(units[0])
# 4. read the data and convert to the correct units
v = u(file.read(),'radians')
```

This is a standalone module, not relying on either DANSE or NeXus, and can be used for other unit conversion tasks.

Note: minutes are used for angle and seconds are used for time. We cannot tell what the correct interpretation is without knowing something about the fields themselves. If this becomes an issue, we will need to allow the application to set the dimension for the unit rather than inferring the dimension from an example unit.

class `sas.sascalc.data_util.nxsunit.`**`Converter`**`(`*name*`)`

Bases: `object`

Unit converter for NeXus style units.

`dims = [{'kilometre': 1000.0, 'centi*Metre': 0.01, 'mili*meter': 0.001, 'gigamete`

All normal dictionary methods are available. Update and comparison is restricted to other OrderedDict objects.

Various sequence methods are available, including the ability to explicitly mutate the key ordering.

__contains__ tests:

```
>>> d = OrderedDict(((1, 3),))
>>> 1 in d
1
>>> 4 in d
0
```

__getitem__ tests:

```
>>> OrderedDict(((1, 3), (3, 2), (2, 1)))[2]
1
>>> OrderedDict(((1, 3), (3, 2), (2, 1)))[4]
Traceback (most recent call last):
KeyError: 4
```

__len__ tests:

```
>>> len(OrderedDict())
0
>>> len(OrderedDict(((1, 3), (3, 2), (2, 1))))
3
```

get tests:

```
>>> d = OrderedDict(((1, 3), (3, 2), (2, 1)))
>>> d.get(1)
3
>>> d.get(4) is None
1
>>> d.get(4, 5)
5
>>> d
OrderedDict([(1, 3), (3, 2), (2, 1)])
```

has_key tests:

```
>>> d = OrderedDict(((1, 3), (3, 2), (2, 1)))
>>> d.has_key(1)
1
>>> d.has_key(4)
0
```

clear()

```
>>> d = OrderedDict(((1, 3), (3, 2), (2, 1)))
>>> d.clear()
>>> d
OrderedDict([])
```

copy()

```
>>> OrderedDict(((1, 3), (3, 2), (2, 1))).copy()
OrderedDict([(1, 3), (3, 2), (2, 1)])
```

index(*key*)
Return the position of the specified key in the OrderedDict.

```
>>> d = OrderedDict(((1, 3), (3, 2), (2, 1)))
>>> d.index(3)
1
>>> d.index(4)
Traceback (most recent call last):
ValueError: list.index(x): x not in list
```

insert (*index*, *key*, *value*)

Takes `index`, `key`, and `value` as arguments.

Sets `key` to `value`, so that `key` is at position `index` in the OrderedDict.

```
>>> d = OrderedDict(((1, 3), (3, 2), (2, 1)))
>>> d.insert(0, 4, 0)
>>> d
OrderedDict([(4, 0), (1, 3), (3, 2), (2, 1)])
>>> d.insert(0, 2, 1)
>>> d
OrderedDict([(2, 1), (4, 0), (1, 3), (3, 2)])
>>> d.insert(8, 8, 1)
>>> d
OrderedDict([(2, 1), (4, 0), (1, 3), (3, 2), (8, 1)])
```

items ()

`items` returns a list of tuples representing all the `(key, value)` pairs in the dictionary.

```
>>> d = OrderedDict(((1, 3), (3, 2), (2, 1)))
>>> d.items()
[(1, 3), (3, 2), (2, 1)]
>>> d.clear()
>>> d.items()
[]
```

iteritems ()

```
>>> ii = OrderedDict(((1, 3), (3, 2), (2, 1))).iteritems()
>>> ii.next()
(1, 3)
>>> ii.next()
(3, 2)
>>> ii.next()
(2, 1)
>>> ii.next()
Traceback (most recent call last):
StopIteration
```

iterkeys ()

```
>>> ii = OrderedDict(((1, 3), (3, 2), (2, 1))).iterkeys()
>>> ii.next()
1
>>> ii.next()
3
>>> ii.next()
2
>>> ii.next()
Traceback (most recent call last):
StopIteration
```

itervalues ()

```
>>> iv = OrderedDict(((1, 3), (3, 2), (2, 1))).itervalues()
>>> iv.next()
3
>>> iv.next()
2
>>> iv.next()
1
>>> iv.next()
Traceback (most recent call last):
StopIteration
```

keys()

Return a list of keys in the `OrderedDict`.

```
>>> d = OrderedDict(((1, 3), (3, 2), (2, 1)))
>>> d.keys()
[1, 3, 2]
```

pop(*key*, **args*)

No dict.pop in Python 2.2, gotta reimplement it

```
>>> d = OrderedDict(((1, 3), (3, 2), (2, 1)))
>>> d.pop(3)
2
>>> d
OrderedDict([(1, 3), (2, 1)])
>>> d.pop(4)
Traceback (most recent call last):
KeyError: 4
>>> d.pop(4, 0)
0
>>> d.pop(4, 0, 1)
Traceback (most recent call last):
TypeError: pop expected at most 2 arguments, got 3
```

popitem(*i=-1*)

Delete and return an item specified by index, not a random one as in dict. The index is -1 by default (the last item).

```
>>> d = OrderedDict(((1, 3), (3, 2), (2, 1)))
>>> d.popitem()
(2, 1)
>>> d
OrderedDict([(1, 3), (3, 2)])
>>> d.popitem(0)
(1, 3)
>>> OrderedDict().popitem()
Traceback (most recent call last):
KeyError: 'popitem(): dictionary is empty'
>>> d.popitem(2)
Traceback (most recent call last):
IndexError: popitem(): index 2 not valid
```

rename(*old_key*, *new_key*)

Rename the key for a given value, without modifying sequence order.

For the case where new_key already exists this raise an exception, since if new_key exists, it is ambiguous as to what happens to the associated values, and the position of new_key in the sequence.

```
>>> od = OrderedDict()
>>> od['a'] = 1
>>> od['b'] = 2
```

(continues on next page)

(continued from previous page)

```
>>> od.items()
[('a', 1), ('b', 2)]
>>> od.rename('b', 'c')
>>> od.items()
[('a', 1), ('c', 2)]
>>> od.rename('c', 'a')
Traceback (most recent call last):
ValueError: New key already exists: 'a'
>>> od.rename('d', 'b')
Traceback (most recent call last):
KeyError: 'd'
```

reverse()

Reverse the order of the OrderedDict.

```
>>> d = OrderedDict(((1, 3), (3, 2), (2, 1)))
>>> d.reverse()
>>> d
OrderedDict([(2, 1), (3, 2), (1, 3)])
```

setdefault(*key*, *defval=None*)

```
>>> d = OrderedDict(((1, 3), (3, 2), (2, 1)))
>>> d.setdefault(1)
3
>>> d.setdefault(4) is None
True
>>> d
OrderedDict([(1, 3), (3, 2), (2, 1), (4, None)])
>>> d.setdefault(5, 0)
0
>>> d
OrderedDict([(1, 3), (3, 2), (2, 1), (4, None), (5, 0)])
```

setitems(*items*)

This method allows you to set the items in the dict.

It takes a list of tuples - of the same sort returned by the `items` method.

```
>>> d = OrderedDict()
>>> d.setitems(((3, 1), (2, 3), (1, 2)))
>>> d
OrderedDict([(3, 1), (2, 3), (1, 2)])
```

setkeys(*keys*)

`setkeys` all ows you to pass in a new list of keys which will replace the current set. This must contain the same set of keys, but need not be in the same order.

If you pass in new keys that don't match, a `KeyError` will be raised.

```
>>> d = OrderedDict(((1, 3), (3, 2), (2, 1)))
>>> d.keys()
[1, 3, 2]
>>> d.setkeys((1, 2, 3))
>>> d
OrderedDict([(1, 3), (2, 1), (3, 2)])
>>> d.setkeys(['a', 'b', 'c'])
Traceback (most recent call last):
KeyError: 'Keylist is not the same as current keylist.'
```

setvalues(*values*)

You can pass in a list of values, which will replace the current list. The value list must be the same len as the OrderedDict.

(Or a `ValueError` is raised.)

```
>>> d = OrderedDict(((1, 3), (3, 2), (2, 1)))
>>> d.setvalues((1, 2, 3))
>>> d
OrderedDict([(1, 1), (3, 2), (2, 3)])
>>> d.setvalues([6])
Traceback (most recent call last):
ValueError: Value list is not the same length as the OrderedDict.
```

`sort` (**args*, ***kwargs*)
Sort the key order in the OrderedDict.

This method takes the same arguments as the `list.sort` method on your version of Python.

```
>>> d = OrderedDict(((4, 1), (2, 2), (3, 3), (1, 4)))
>>> d.sort()
>>> d
OrderedDict([(1, 4), (2, 2), (3, 3), (4, 1)])
```

`update` (*from_od*)
Update from another OrderedDict or sequence of (key, value) pairs

```
>>> d = OrderedDict(((1, 0), (0, 1)))
>>> d.update(OrderedDict(((1, 3), (3, 2), (2, 1))))
>>> d
OrderedDict([(1, 3), (0, 1), (3, 2), (2, 1)])
>>> d.update({4: 4})
Traceback (most recent call last):
TypeError: undefined order, cannot get items from dict
>>> d.update((4, 4))
Traceback (most recent call last):
TypeError: cannot convert dictionary update sequence element "4" to a 2-
↪item sequence
```

`values` (*values=None*)
Return a list of all the values in the OrderedDict.

Optionally you can pass in a list of values, which will replace the current list. The value list must be the same len as the OrderedDict.

```
>>> d = OrderedDict(((1, 3), (3, 2), (2, 1)))
>>> d.values()
[3, 2, 1]
```

class `sas.sascalc.data_util.odict.`**`SequenceOrderedDict`** (*init_val=()*, *strict=True*)
Bases: *`sas.sascalc.data_util.odict.OrderedDict`*

Experimental version of OrderedDict that has a custom object for `keys`, `values`, and `items`.

These are callable sequence objects that work as methods, or can be manipulated directly as sequences.

Test for `keys`, `items` and `values`.

```
>>> d = SequenceOrderedDict(((1, 2), (2, 3), (3, 4)))
>>> d
SequenceOrderedDict([(1, 2), (2, 3), (3, 4)])
>>> d.keys
[1, 2, 3]
>>> d.keys()
[1, 2, 3]
```

(continues on next page)

(continued from previous page)

```
>>> d.setkeys((3, 2, 1))
>>> d
SequenceOrderedDict([(3, 4), (2, 3), (1, 2)])
>>> d.setkeys((1, 2, 3))
>>> d.keys[0]
1
>>> d.keys[:]
[1, 2, 3]
>>> d.keys[-1]
3
>>> d.keys[-2]
2
>>> d.keys[0:2] = [2, 1]
>>> d
SequenceOrderedDict([(2, 3), (1, 2), (3, 4)])
>>> d.keys.reverse()
>>> d.keys
[3, 1, 2]
>>> d.keys = [1, 2, 3]
>>> d
SequenceOrderedDict([(1, 2), (2, 3), (3, 4)])
>>> d.keys = [3, 1, 2]
>>> d
SequenceOrderedDict([(3, 4), (1, 2), (2, 3)])
>>> a = SequenceOrderedDict()
>>> b = SequenceOrderedDict()
>>> a.keys == b.keys
1
>>> a['a'] = 3
>>> a.keys == b.keys
0
>>> b['a'] = 3
>>> a.keys == b.keys
1
>>> b['b'] = 3
>>> a.keys == b.keys
0
>>> a.keys > b.keys
0
>>> a.keys < b.keys
1
>>> 'a' in a.keys
1
>>> len(b.keys)
2
>>> 'c' in d.keys
0
>>> 1 in d.keys
1
>>> [v for v in d.keys]
[3, 1, 2]
>>> d.keys.sort()
>>> d.keys
[1, 2, 3]
>>> d = SequenceOrderedDict(((1, 2), (2, 3), (3, 4)), strict=True)
>>> d.keys[::-1] = [1, 2, 3]
>>> d
SequenceOrderedDict([(3, 4), (2, 3), (1, 2)])
>>> d.keys[:2]
[3, 2]
>>> d.keys[:2] = [1, 3]
```

(continues on next page)

(continued from previous page)

```
Traceback (most recent call last):
KeyError: 'Keylist is not the same as current keylist.'
```

```
>>> d = SequenceOrderedDict(((1, 2), (2, 3), (3, 4)))
>>> d
SequenceOrderedDict([(1, 2), (2, 3), (3, 4)])
>>> d.values
[2, 3, 4]
>>> d.values()
[2, 3, 4]
>>> d.setvalues((4, 3, 2))
>>> d
SequenceOrderedDict([(1, 4), (2, 3), (3, 2)])
>>> d.values[::-1]
[2, 3, 4]
>>> d.values[0]
4
>>> d.values[-2]
3
>>> del d.values[0]
Traceback (most recent call last):
TypeError: Can't delete items from values
>>> d.values[::2] = [2, 4]
>>> d
SequenceOrderedDict([(1, 2), (2, 3), (3, 4)])
>>> 7 in d.values
0
>>> len(d.values)
3
>>> [val for val in d.values]
[2, 3, 4]
>>> d.values[-1] = 2
>>> d.values.count(2)
2
>>> d.values.index(2)
0
>>> d.values[-1] = 7
>>> d.values
[2, 3, 7]
>>> d.values.reverse()
>>> d.values
[7, 3, 2]
>>> d.values.sort()
>>> d.values
[2, 3, 7]
>>> d.values.append('anything')
Traceback (most recent call last):
TypeError: Can't append items to values
>>> d.values = (1, 2, 3)
>>> d
SequenceOrderedDict([(1, 1), (2, 2), (3, 3)])
```

```
>>> d = SequenceOrderedDict(((1, 2), (2, 3), (3, 4)))
>>> d
SequenceOrderedDict([(1, 2), (2, 3), (3, 4)])
>>> d.items()
[(1, 2), (2, 3), (3, 4)]
>>> d.setitems([(3, 4), (2 ,3), (1, 2)])
>>> d
SequenceOrderedDict([(3, 4), (2, 3), (1, 2)])
```

(continues on next page)

(continued from previous page)

```
>>> d.items[0]
(3, 4)
>>> d.items[:-1]
[(3, 4), (2, 3)]
>>> d.items[1] = (6, 3)
>>> d.items
[(3, 4), (6, 3), (1, 2)]
>>> d.items[1:2] = [(9, 9)]
>>> d
SequenceOrderedDict([(3, 4), (9, 9), (1, 2)])
>>> del d.items[1:2]
>>> d
SequenceOrderedDict([(3, 4), (1, 2)])
>>> (3, 4) in d.items
1
>>> (4, 3) in d.items
0
>>> len(d.items)
2
>>> [v for v in d.items]
[(3, 4), (1, 2)]
>>> d.items.count((3, 4))
1
>>> d.items.index((1, 2))
1
>>> d.items.index((2, 1))
Traceback (most recent call last):
ValueError: list.index(x): x not in list
>>> d.items.reverse()
>>> d.items
[(1, 2), (3, 4)]
>>> d.items.reverse()
>>> d.items.sort()
>>> d.items
[(1, 2), (3, 4)]
>>> d.items.append((5, 6))
>>> d.items
[(1, 2), (3, 4), (5, 6)]
>>> d.items.insert(0, (0, 0))
>>> d.items
[(0, 0), (1, 2), (3, 4), (5, 6)]
>>> d.items.insert(-1, (7, 8))
>>> d.items
[(0, 0), (1, 2), (3, 4), (7, 8), (5, 6)]
>>> d.items.pop()
(5, 6)
>>> d.items
[(0, 0), (1, 2), (3, 4), (7, 8)]
>>> d.items.remove((1, 2))
>>> d.items
[(0, 0), (3, 4), (7, 8)]
>>> d.items.extend([(1, 2), (5, 6)])
>>> d.items
[(0, 0), (3, 4), (7, 8), (1, 2), (5, 6)]
```

sas.sascalc.data_util.ordereddict module

Backport from python2.7 to python <= 2.6.

class `sas.sascalc.data_util.ordereddict.`**`OrderedDict`**(**args*, ***kwds*)

Bases: `dict`

clear() → None. Remove all items from D.

copy() → a shallow copy of D

classmethod fromkeys(*S*[, *v*]) → New dict with keys from S and values equal to v.
v defaults to None.

items() → list of D's (key, value) pairs, as 2-tuples

keys() → list of D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem() → (k, v), remove and return some (key, value) pair as a
2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → list of D's values

sas.sascalc.data_util.ordereddicttest module

class `sas.sascalc.data_util.ordereddicttest.`**TestOrderedDict**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

test_clear()

test_copying()

test_delitem()

test_equality()

test_init()

test_iterators()

test_pop()

test_popitem()

test_reinsert()

test_repr()

test_setdefault()

test_setitem()

test_update()

sas.sascalc.data_util.pathutils module

Utilities for path manipulation. Not to be confused with the pathutils module from the pythonutils package (http://groups.google.com/group/pythonutils).

`sas.sascalc.data_util.pathutils.`**relpath**(*p1*, *p2*)
Compute the relative path of p1 with respect to p2.

sas.sascalc.data_util.registry module

File extension registry.

This provides routines for opening files based on extension, and registers the built-in file extensions.

class sas.sascalc.data_util.registry.**ExtensionRegistry**(***kw*)

Bases: object

Associate a file loader with an extension.

Note that there may be multiple loaders for the same extension.

Example:

```
registry = ExtensionRegistry()

# Add an association by setting an element
registry['.zip'] = unzip

# Multiple extensions for one loader
registry['.tgz'] = untar
registry['.tar.gz'] = untar

# Generic extensions to use after trying more specific extensions;
# these will be checked after the more specific extensions fail.
registry['.gz'] = gunzip

# Multiple loaders for one extension
registry['.cx'] = cx1
registry['.cx'] = cx2
registry['.cx'] = cx3

# Show registered extensions
print registry.extensions()

# Can also register a format name for explicit control from caller
registry['cx3'] = cx3
print registry.formats()

# Retrieve loaders for a file name
registry.lookup('hello.cx') -> [cx3,cx2,cx1]

# Run loader on a filename
registry.load('hello.cx') ->
    try:
        return cx3('hello.cx')
    except:
        try:
            return cx2('hello.cx')
        except:
            return cx1('hello.cx')

# Load in a specific format ignoring extension
registry.load('hello.cx',format='cx3') ->
    return cx3('hello.cx')
```

extensions()

Return a sorted list of registered extensions.

formats()

Return a sorted list of the registered formats.

load(*path*, *format=None*)

Call the loader for the file type of path.

Raises

- **`ValueError`** – if no loader is available.
- **`KeyError`** – if format is not available.

May raise a loader-defined exception if loader fails.

`lookup`(*path*)
Return the loader associated with the file type of path.

Parameters **`path`** – Data file path

Raises **`ValueError`** – When no loaders are found for the file.

Returns List of available readers for the file extension

sas.sascalc.data_util.uncertainty module

Uncertainty propagation class for arithmetic, log and exp.

Based on scalars or numpy vectors, this class allows you to store and manipulate values+uncertainties, with propagation of gaussian error for addition, subtraction, multiplication, division, power, exp and log.

Storage properties are determined by the numbers used to set the value and uncertainty. Be sure to use floating point uncertainty vectors for inplace operations since numpy does not do automatic type conversion. Normal operations can use mixed integer and floating point. In place operations such as *a *= b* create at most one extra copy for each operation. By contrast, *c = a*b* uses four intermediate vectors, so shouldn't be used for huge arrays.

`class` `sas.sascalc.data_util.uncertainty.`**`Uncertainty`**(*x*, *variance=None*)
Bases: `object`

`dx`
standard deviation

`exp`()

`log`()

Module contents

sas.sascalc.dataloader package

Subpackages

sas.sascalc.dataloader.readers package

Submodules

sas.sascalc.dataloader.readers.abs_reader module

IGOR 1D data reader

`class` `sas.sascalc.dataloader.readers.abs_reader.`**`Reader`**
Bases: *`sas.sascalc.dataloader.file_reader_base_class.FileReader`*

Class to load IGOR reduced .ABS files

`ext = ['.abs', '.cor']`

`get_file_contents`()
Get the contents of the file

Raises

- **RuntimeError** – when the file can't be opened
- **ValueError** – when the length of the data vectors are inconsistent

type = ['IGOR 1D files (*.abs)|*.abs', 'IGOR 1D USANS files (*.cor)|*.cor']

type_name = 'IGOR 1D'

sas.sascalc.dataloader.readers.anton_paar_saxs_reader module

CanSAS 2D data reader for reading HDF5 formatted CanSAS files.

class sas.sascalc.dataloader.readers.anton_paar_saxs_reader.**Reader**(*xml=None*, *schema=None*)

Bases: *sas.sascalc.dataloader.readers.xml_reader.XMLreader*

A class for reading in Anton Paar .pdh files

allow_all = False

errors = None

ext = ['.pdh', '.PDH']

get_file_contents()

This is the general read method that all SasView data_loaders must have.

Parameters **filename** – A path for an XML formatted Anton Paar SAXSess data file.

Returns List of Data1D objects or a list of errors.

logging = None

parent_list = None

raw_data = None

read_data()

reset_state()

Resets the class state to a base case when loading a new data file so previous data files do not appear a second time

type = ['Anton Paar SAXSess Files (*.pdh)|*.pdh']

type_name = 'Anton Paar SAXSess'

sas.sascalc.dataloader.readers.ascii_reader module

Generic multi-column ASCII data reader

class sas.sascalc.dataloader.readers.ascii_reader.**Reader**

Bases: *sas.sascalc.dataloader.file_reader_base_class.FileReader*

Class to load ascii files (2, 3 or 4 columns).

allow_all = True

ext = ['.txt', '.dat', '.abs', '.csv']

get_file_contents()

Get the contents of the file

min_data_pts = 5

type = ['ASCII files (*.txt)|*.txt', 'ASCII files (*.dat)|*.dat', 'ASCII files (*.a

type_name = 'ASCII'

sas.sascalc.dataloader.readers.associations module

Module to associate default readers to file extensions. The module reads an xml file to get the readers for each file extension. The readers are tried in order they appear when reading a file.

`sas.sascalc.dataloader.readers.associations.`**`read_associations`**`(`*loader, settings={'.abs': 'abs_reader', '.cor': 'abs_reader', '.dat': 'red2d_reader', '.h5': 'cansas_reader_HDF5', '.nxs': 'cansas_reader_HDF5', '.pdh': 'anton_paar_saxs_reader', '.sans': 'danse_reader', '.ses': 'sesans_reader', '.txt': 'ascii_reader', '.xml': 'cansas_reader'}*`)`

Read the specified settings file to associate default readers to file extension.

Parameters

- **`loader`** – Loader object
- **`settings`** – path to the json settings file [string]

sas.sascalc.dataloader.readers.cansas_constants module

Information relating to the CanSAS data format. These constants are used in the cansas_reader.py file to read in any version of the cansas format.

`class` `sas.sascalc.dataloader.readers.cansas_constants.`**`CansasConstants`**

Bases: `object`

The base class to define where all of the data is to be saved by cansas_reader.py.

`ANY = {'storeas': 'content'}`

`CANSAS_FORMAT = {'SASentry': {'attributes': {'name': {}}, 'units_optional': True`

`CANSAS_NS = {'1.0': {'ns': 'cansas1d/1.0', 'schema': 'cansas1d_v1_0.xsd'}, '1.1'`

`RUN = {'attributes': {'name': {}}}`

`SASDATA = {'attributes': {'name': {}}, 'children': {'zacceptance': {'storeas':`

`SASDATA_IDATA = {'attributes': {'timestamp': {'storeas': 'timestamp'}, 'name':`

`SASDATA_IDATA_DQL = {'attributes': {'unit': {'storeas': 'content'}}, 'storeas':`

`SASDATA_IDATA_DQW = {'attributes': {'unit': {'storeas': 'content'}}, 'storeas':`

`SASDATA_IDATA_I = {'attributes': {'unit': {'storeas': 'content'}}, 'storeas':`

`SASDATA_IDATA_IDEV = {'attributes': {'unit': {'storeas': 'content'}}, 'storeas':`

`SASDATA_IDATA_Q = {'attributes': {'unit': {'storeas': 'content'}}, 'storeas':`

```
SASDATA_IDATA_QDEV = {'attributes': {'unit': {'storeas': 'content'}}, 'storeas':
SASDATA_IDATA_QMEAN = {'attributes': {'unit': {}}, 'unit': 'x_unit'}
SASDATA_IDATA_SHADOWFACTOR = {}
SASINSTR = {'children': {'SAScollimation': {'attributes': {'name': {}}, 'childre
SASINSTR_COLL = {'attributes': {'name': {}}, 'children': {'aperture': {'attribut
SASINSTR_COLL_APER = {'attributes': {'type': {}, 'name': {}}, 'children': {'dist
SASINSTR_COLL_APER_ATTR = {'unit': {}}
SASINSTR_COLL_APER_DIST = {'attributes': {'unit': {}}, 'storeas': 'float', 'unit
SASINSTR_COLL_APER_SIZE = {'attributes': {'unit': {}}, 'children': {'y': {'attri
SASINSTR_COLL_APER_X = {'attributes': {'unit': {}}, 'storeas': 'float', 'unit':
SASINSTR_COLL_APER_Y = {'attributes': {'unit': {}}, 'storeas': 'float', 'unit':
SASINSTR_COLL_APER_Z = {'attributes': {'unit': {}}, 'storeas': 'float', 'unit':
SASINSTR_DET = {'attributes': {'name': {'storeas': 'content'}}, 'children': {'or
SASINSTR_DET_BC = {'children': {'y': {'attributes': {'storeas': 'content'}, 'sto
SASINSTR_DET_BC_X = {'attributes': {'storeas': 'content'}, 'storeas': 'float', 'u
SASINSTR_DET_BC_Y = {'attributes': {'storeas': 'content'}, 'storeas': 'float', 'u
SASINSTR_DET_BC_Z = {'attributes': {'storeas': 'content'}, 'storeas': 'float', 'u
SASINSTR_DET_OFF = {'children': {'y': {'attributes': {'unit': {'storeas': 'cont
SASINSTR_DET_OFF_ATTR = {'unit': {'storeas': 'content'}}
SASINSTR_DET_OFF_X = {'attributes': {'unit': {'storeas': 'content'}}, 'storeas':
SASINSTR_DET_OFF_Y = {'attributes': {'unit': {'storeas': 'content'}}, 'storeas':
SASINSTR_DET_OFF_Z = {'attributes': {'unit': {'storeas': 'content'}}, 'storeas':
SASINSTR_DET_OR = {'children': {'yaw': {'attributes': {}, 'storeas': 'float', 'u
SASINSTR_DET_OR_ATTR = {}
SASINSTR_DET_OR_PITCH = {'attributes': {}, 'storeas': 'float', 'unit': 'orientati
SASINSTR_DET_OR_ROLL = {'attributes': {}, 'storeas': 'float', 'unit': 'orientatic
SASINSTR_DET_OR_YAW = {'attributes': {}, 'storeas': 'float', 'unit': 'orientation
SASINSTR_DET_PIXEL = {'children': {'y': {'attributes': {'storeas': 'content'},
SASINSTR_DET_PIXEL_X = {'attributes': {'storeas': 'content'}, 'storeas': 'float',
SASINSTR_DET_PIXEL_Y = {'attributes': {'storeas': 'content'}, 'storeas': 'float',
SASINSTR_DET_PIXEL_Z = {'attributes': {'storeas': 'content'}, 'storeas': 'float',
SASINSTR_DET_SDD = {'attributes': {'unit': {}}, 'storeas': 'float', 'unit': 'dis
SASINSTR_DET_SLIT = {'attributes': {'unit': {}}, 'storeas': 'float', 'unit': 'sl
SASINSTR_SRC = {'attributes': {'name': {}}, 'children': {'wavelength_max': {'att
SASINSTR_SRC_BEAMSIZE = {'attributes': {'name': {}}, 'children': {'y': {'attribu
SASINSTR_SRC_BEAMSIZE_ATTR = {'unit': ''}
SASINSTR_SRC_BEAMSIZE_X = {'attributes': {'unit': ''}, 'storeas': 'float', 'unit
SASINSTR_SRC_BEAMSIZE_Y = {'attributes': {'unit': ''}, 'storeas': 'float', 'unit
SASINSTR_SRC_BEAMSIZE_Z = {'attributes': {'unit': ''}, 'storeas': 'float', 'unit
```

SASINSTR_SRC_WL = {'attributes': {'unit': {}}, 'storeas': 'float', 'unit': 'wave

SASINSTR_SRC_WL_MAX = {'attributes': {'unit': {'storeas': 'content'}}, 'storeas'

SASINSTR_SRC_WL_MIN = {'attributes': {'unit': {'storeas': 'content'}}, 'storeas'

SASINSTR_SRC_WL_SPR = {'attributes': {'unit': {'storeas': 'content'}}, 'storeas'

SASNOTE = {}

SASPROCESS = {'children': {'term': {'attributes': {'name': {}, 'unit': {}}}, '

SASPROCESS_SASPROCESSNOTE = {'children': {'<any>': {'storeas': 'content'}}}

SASPROCESS_TERM = {'attributes': {'name': {}, 'unit': {}}}

SASSAMPLE = {'attributes': {'name': {}}, 'children': {'<any>': {'storeas': 'cor

SASSAMPLE_ORIENT = {'children': {'yaw': {'attributes': {'unit': {}}, 'storeas':

SASSAMPLE_ORIENT_ATTR = {'unit': {}}

SASSAMPLE_ORIENT_PITCH = {'attributes': {'unit': {}}, 'storeas': 'float', 'unit'

SASSAMPLE_ORIENT_ROLL = {'attributes': {'unit': {}}, 'storeas': 'float', 'unit':

SASSAMPLE_ORIENT_YAW = {'attributes': {'unit': {}}, 'storeas': 'float', 'unit':

SASSAMPLE_POS = {'children': {'y': {'attributes': {'unit': {}}, 'storeas': 'flo

SASSAMPLE_POS_ATTR = {'unit': {}}

SASSAMPLE_POS_X = {'attributes': {'unit': {}}, 'storeas': 'float', 'unit': 'pos

SASSAMPLE_POS_Y = {'attributes': {'unit': {}}, 'storeas': 'float', 'unit': 'pos

SASSAMPLE_POS_Z = {'attributes': {'unit': {}}, 'storeas': 'float', 'unit': 'pos

SASSAMPLE_TEMP = {'attributes': {'unit': {}}, 'storeas': 'float', 'unit': 'tempe

SASSAMPLE_THICK = {'attributes': {'unit': {}}, 'storeas': 'float', 'unit': 'thic

SASSAMPLE_TRANS = {'storeas': 'float'}

SASTRANSSPEC = {'attributes': {'timestamp': {}, 'name': {}}, 'children': {'Tdata

SASTRANSSPEC_TDATA = {'children': {'Tdev': {'attributes': {'unit': {'storeas':

SASTRANSSPEC_TDATA_LAMDBA = {'attributes': {'unit': {'storeas': 'content'}}, 'sto

SASTRANSSPEC_TDATA_T = {'attributes': {'unit': {'storeas': 'content'}}, 'storeas

SASTRANSSPEC_TDATA_TDEV = {'attributes': {'unit': {'storeas': 'content'}}, 'store

TITLE = {}

format = ''

get_namespace_map()
Helper method to get the names namespace list

iterate_namespace(*namespace*)
Method to iterate through a cansas constants tree based on a list of names

Parameters namespace – A list of names that match the tree structure of cansas_constants

names = ''

class sas.sascalc.dataloader.readers.cansas_constants.**CurrentLevel**
Bases: object

A helper class to hold information on where you are in the constants tree

current_level = ''

get_current_level()
Helper method to get the current_level map

get_data_type()
Helper method to get the ns_datatype label

get_variable()
Helper method to get the ns_variable label

ns_datatype = ''

ns_optional = True

sas.sascalc.dataloader.readers.cansas_reader module

class sas.sascalc.dataloader.readers.cansas_reader.**Reader**(*xml=None*, *schema=None*)
Bases: *sas.sascalc.dataloader.readers.xml_reader.XMLreader*

allow_all = True

base_ns = '{cansas1d/1.0}'

cansas_defaults = None

cansas_version = '1.0'

current_data1d = None

data = None

errors = set([])

ext = ['.xml', '.XML', '.svs', '.SVS']

frm = ''

get_file_contents(*xml_file=None*, *schema_path=''*, *invalid=True*)
Reader specific class to access the contents of the file All reader classes that inherit from FileReader must implement

invalid = True

is_cansas(*ext='xml'*)
Checks to see if the XML file is a CanSAS file

Parameters `ext` – The file extension of the data file

Raises *FileContentsException* – Raised if XML file isn't valid CanSAS

load_file_and_schema(*xml_file*, *schema_path=''*)

logging = None

names = None

ns_list = None

read(*xml_file*, *schema_path=''*, *invalid=True*)
Basic file reader

Parameters `filepath` – The full or relative path to a file to be loaded

reset_state()
Resets the class state to a base case when loading a new data file so previous data files do not appear a second time

type = ['XML files (*.xml)|*.xml', 'SasView Save Files (*.svs)|*.svs']

type_name = 'canSAS'

write (*filename*, *datainfo*)
Write the content of a Data1D as a CanSAS XML file

Parameters

- **filename** – name of the file to write
- **datainfo** – Data1D object

write_node (*parent*, *name*, *value*, *attr=None*)

Parameters

- **doc** – document DOM
- **parent** – parent node
- **name** – tag of the element
- **value** – value of the child text node
- **attr** – attribute dictionary

Returns True if something was appended, otherwise False

sas.sascalc.dataloader.readers.cansas_reader.**get_content** (*location*, *node*)
Get the first instance of the content of a xpath location.

Parameters

- **location** – xpath location
- **node** – node to start at

Returns Element, or None

sas.sascalc.dataloader.readers.cansas_reader.**getattrchain** (*obj*, *chain*, *default=None*)
Like getattr, but the attr may contain multiple parts separated by '.'

sas.sascalc.dataloader.readers.cansas_reader.**setattrchain** (*obj*, *chain*, *value*)
Like setattr, but the attr may contain multiple parts separated by '.'

sas.sascalc.dataloader.readers.cansas_reader.**write_node** (*doc*, *parent*, *name*, *value*, *attr=None*)

Parameters

- **doc** – document DOM
- **parent** – parent node
- **name** – tag of the element
- **value** – value of the child text node
- **attr** – attribute dictionary

Returns True if something was appended, otherwise False

sas.sascalc.dataloader.readers.cansas_reader_HDF5 module

NXcanSAS data reader for reading HDF5 formatted CanSAS files.

class sas.sascalc.dataloader.readers.cansas_reader_HDF5.**Reader**
Bases: *sas.sascalc.dataloader.file_reader_base_class.FileReader*

A class for reading in NXcanSAS data files. The current implementation has been tested to load data generated by multiple facilities, all of which are known to produce NXcanSAS standards compliant data. Any number of data sets may be present within the file and any dimensionality of data may be used. Currently 1D and 2D SAS data sets are supported, but should be immediately extensible to SESANS data.

Any number of SASdata groups may be present in a SASentry and the data within each SASdata group can be a single 1D I(Q), multi-framed 1D I(Q), 2D I(Qx, Qy) or multi-framed 2D I(Qx, Qy).

> **Dependencies** The NXcanSAS HDF5 reader requires h5py => v2.5.0 or later.

`add_data_set`(*key=''*)
: Adds the current_dataset to the list of outputs after preforming final processing on the data and then calls a private method to generate a new data set.

  Parameters **`key`** – NeXus group name for current tree level

`add_intermediate`()
: This method stores any intermediate objects within the final data set after fully reading the set.

  Parameters **`parent`** – The NXclass name for the h5py Group object that just finished being processed

`allow_all = True`

`static as_list_or_array`(*iterable*)
: Return value as a list if not already a list or array. :param iterable: :return:

`cansas_version = 2.0`

`ext = ['.h5', '.H5']`

`final_data_cleanup`()
: Does some final cleanup and formatting on self.current_datainfo and all data1D and data2D objects and then combines the data and info into Data1D and Data2D objects

`get_file_contents`()
: This is the general read method that all SasView data_loaders must have.

  Parameters **`filename`** – A path for an HDF5 formatted CanSAS 2D data file.

  Returns List of Data1D/2D objects and/or a list of errors.

`process_1d_data_object`(*data_set*, *key*, *unit*)
: SASdata processor method for 1d data items :param data_set: data from HDF5 file :param key: canSAS_class attribute :param unit: unit attribute

`process_2d_data_object`(*data_set*, *key*, *unit*)

`process_aperture`(*data_point*, *key*)
: SASaperture processor :param data_point: Single point from an HDF5 data file :param key: class name data_point was taken from

`process_collimation`(*data_point*, *key*, *unit*)
: SAScollimation processor :param data_point: Single point from an HDF5 data file :param key: class name data_point was taken from :param unit: unit attribute from data set

`process_detector`(*data_point*, *key*, *unit*)
: SASdetector processor :param data_point: Single point from an HDF5 data file :param key: class name data_point was taken from :param unit: unit attribute from data set

`process_process`(*data_point*, *key*)
: SASprocess processor :param data_point: Single point from an HDF5 data file :param key: class name data_point was taken from

`process_sample`(*data_point*, *key*)
: SASsample processor :param data_point: Single point from an HDF5 data file :param key: class name data_point was taken from

`process_source`(*data_point*, *key*, *unit*)
: SASsource processor :param data_point: Single point from an HDF5 data file :param key: class name data_point was taken from :param unit: unit attribute from data set

process_trans_spectrum(*data_set*, *key*)
SAStransmission_spectrum processor :param data_set: data from HDF5 file :param key: canSAS_class attribute

read_children(*data*, *parent_list*)
A recursive method for stepping through the hierarchical data file.

Parameters

- **data** – h5py Group object of any kind
- **parent** – h5py Group parent name

reset_state()
Create the reader object and define initial states for class variables

type = ['NXcanSAS HDF5 Files (*.h5)|*.h5|']

type_name = 'NXcanSAS'

sas.sascalc.dataloader.readers.cansas_reader_HDF5.**h5attr**(*node*, *key*, *default=None*)

sas.sascalc.dataloader.readers.danse_reader module

DANSE/SANS file reader

class sas.sascalc.dataloader.readers.danse_reader.**Reader**
Bases: *sas.sascalc.dataloader.file_reader_base_class.FileReader*

Example data manipulation

ext = ['.sans', '.SANS']

get_file_contents()
Reader specific class to access the contents of the file All reader classes that inherit from FileReader must implement

type = ['DANSE files (*.sans)|*.sans']

type_name = 'DANSE'

sas.sascalc.dataloader.readers.red2d_reader module

TXT/IGOR 2D Q Map file reader

class sas.sascalc.dataloader.readers.red2d_reader.**Reader**
Bases: *sas.sascalc.dataloader.file_reader_base_class.FileReader*

Simple data reader for Igor data files

ext = ['.DAT', '.dat']

get_file_contents()
Reader specific class to access the contents of the file All reader classes that inherit from FileReader must implement

type = ['IGOR/DAT 2D file in Q_map (*.dat)|*.DAT']

type_name = 'IGOR/DAT 2D Q_map'

write(*filename*, *data*)
Write to .dat

Parameters

- **filename** – file name to write

- **data** – data2D

sas.sascalc.dataloader.readers.red2d_reader.**check_point**(*x_point*)
check point validity

sas.sascalc.dataloader.readers.sesans_reader module

SESANS reader (based on ASCII reader)

Reader for .ses or .sesans file format

Jurrian Bakker

class sas.sascalc.dataloader.readers.sesans_reader.**Reader**
Bases: *sas.sascalc.dataloader.file_reader_base_class.FileReader*

Class to load sesans files (6 columns).

allow_all = True

ext = ['.ses', '.SES', '.sesans', '.SESANS']

get_file_contents()
Reader specific class to access the contents of the file All reader classes that inherit from FileReader must implement

type = ['SESANS files (*.ses)|*.ses', 'SESANS files (*..sesans)|*.sesans']

type_name = 'SESANS'

sas.sascalc.dataloader.readers.tiff_reader module

Image reader. Untested.

class sas.sascalc.dataloader.readers.tiff_reader.**Reader**
Example data manipulation

ext = ['.tif', '.tiff']

read(*filename=None*)
Open and read the data in a file

Parameters file – path of the file

type = ['TIF files (*.tif)|*.tif', 'TIFF files (*.tiff)|*.tiff']

type_name = 'TIF'

sas.sascalc.dataloader.readers.xml_reader module

Generic XML read and write utility

Usage: Either extend xml_reader or add as a class variable.

class sas.sascalc.dataloader.readers.xml_reader.**XMLreader**(*xml=None*, *schema=None*)
Bases: *sas.sascalc.dataloader.file_reader_base_class.FileReader*

Generic XML read and write class. Mostly helper functions. Makes reading/writing XML a bit easier than calling lxml libraries directly.

Dependencies This class requires lxml 2.3 or higher.

append(*element*, *tree*)
Append an etree Element to an ElementTree.

Parameters

- **element** – etree Element to append
- **tree** – ElementTree object to append to

break_processing_instructions (*string*, *dic*)
Method to break a processing instruction string apart and add to a dict

Parameters

- **string** – A processing instruction as a string
- **dic** – The dictionary to save the PIs to

create_element (*name*, *attrib=None*, *nsmap=None*)
Create an XML element for writing to file

Parameters **name** – The name of the element to be created

create_element_from_string (*xml_string*)
Create an element from an XML string

Parameters **xml_string** – A string of xml

create_tree (*root*)
Create an element tree for processing from an etree element

Parameters **root** – etree Element(s)

ebuilder (*parent*, *elementname*, *text=None*, *attrib=None*)
Use lxml E builder class with arbitrary inputs.

Parameters

- **parnet** – The parent element to append a child to
- **elementname** – The name of the child in string form
- **text** – The element text
- **attrib** – A dictionary of attribute names to attribute values

encoding = None

find_invalid_xml ()
Finds the first offending element that should not be present in XML file

parse_schema_and_doc ()
Creates a dictionary of the parsed schema and xml files.

processing_instructions = None

reader ()
Read in an XML file into memory and return an lxml dictionary

return_processing_instructions ()
Get all processing instructions saved when loading the document

Parameters **tree** – etree.ElementTree object to write PIs to

schema = None

schemadoc = None

set_encoding (*attr_str*)
Find the encoding in the xml declaration and save it as a string

Parameters **attr_str** – All attributes as a string e.g. “foo1=”bar1” foo2=”bar2” foo3=”bar3” … foo_n=”bar_n”“

set_processing_instructions()
Take out all processing instructions and create a dictionary from them If there is a default encoding, the value is also saved

set_schema(*schema*)
Set the schema file and parse

set_xml_file(*xml*)
Set the XML file and parse

set_xml_string(*tag_soup*)
Set an XML string as the working XML.

Parameters **tag_soup** – XML formatted string

to_string(*elem*, *pretty_print=False*, *encoding=None*)
Converts an etree element into a string

validate_xml()
Checks to see if the XML file meets the schema

write_attribute(*elem*, *attr_name*, *attr_value*)
Write attributes to an Element

Parameters

- **elem** – etree.Element object
- **attr_name** – attribute name to write
- **attr_value** – attribute value to set

write_text(*elem*, *text*)
Write text to an etree Element

Parameters

- **elem** – etree.Element object
- **text** – text to write to the element

xml = None

xmldoc = None

xmlroot = None

Module contents

Submodules

sas.sascalc.dataloader.data_info module

Module that contains classes to hold information read from reduced data files.

A good description of the data members can be found in the CanSAS 1D XML data format:

http://www.smallangles.net/wgwiki/index.php/cansas1d_documentation

class sas.sascalc.dataloader.data_info.**Aperture**
Bases: object

distance = None

distance_unit = 'mm'

name = None

size = None

size_name = None

size_unit = 'mm'

type = None

class sas.sascalc.dataloader.data_info.**Collimation**

Bases: `object`

Class to hold collimation information

aperture = None

length = None

length_unit = 'mm'

name = None

class sas.sascalc.dataloader.data_info.**Data1D**(*x=None*, *y=None*, *dx=None*, *dy=None*, *lam=None*, *dlam=None*, *isSesans=None*)

Bases: `sas.sascalc.dataloader.data_info.plottable_1D`, `sas.sascalc.dataloader.data_info.DataInfo`

1D data class

clone_without_data(*length=0*, *clone=None*)

Clone the current object, without copying the data (which will be filled out by a subsequent operation). The data arrays will be initialized to zero.

Parameters

- **length** – length of the data array to be initialized
- **clone** – if provided, the data will be copied to clone

is_slit_smeared()

Check whether the data has slit smearing information :return: True is slit smearing info is present, False otherwise

class sas.sascalc.dataloader.data_info.**Data2D**(*data=None*, *err_data=None*, *qx_data=None*, *qy_data=None*, *q_data=None*, *mask=None*, *dqx_data=None*, *dqy_data=None*)

Bases: `sas.sascalc.dataloader.data_info.plottable_2D`, `sas.sascalc.dataloader.data_info.DataInfo`

2D data class

I_unit = '1/cm'

Q_unit = '1/A'

clone_without_data(*length=0*, *clone=None*)

Clone the current object, without copying the data (which will be filled out by a subsequent operation). The data arrays will be initialized to zero.

Parameters

- **length** – length of the data array to be initialized
- **clone** – if provided, the data will be copied to clone

isSesans = False

x_bins = None

y_bins = None

class sas.sascalc.dataloader.data_info.**DataInfo**

Bases: `object`

Class to hold the data read from a file. It includes four blocks of data for the instrument description, the sample description, the data itself and any other meta data.

add_notes(*message=''*)

Add notes to datainfo

append_empty_process()

collimation = None

detector = None

errors = None

filename = ''

instrument = ''

isSesans = None

meta_data = None

notes = None

process = None

run = None

run_name = None

sample = None

source = None

title = ''

trans_spectrum = None

class sas.sascalc.dataloader.data_info.**Detector**

Bases: `object`

Class to hold detector information

beam_center = None

beam_center_unit = 'mm'

distance = None

distance_unit = 'mm'

name = None

offset = None

offset_unit = 'm'

orientation = None

orientation_unit = 'degree'

pixel_size = None

pixel_size_unit = 'mm'

slit_length = None

slit_length_unit = 'mm'

class sas.sascalc.dataloader.data_info.**Process**

Bases: `object`

Class that holds information about the processes performed on the data.

date = ''

description = ''

is_empty()
Return True if the object is empty

name = ''

notes = None

single_line_desc()
Return a single line string representing the process

term = None

class sas.sascalc.dataloader.data_info.**Sample**
Bases: `object`

Class to hold the sample description

ID = ''

details = None

name = ''

orientation = None

orientation_unit = 'degree'

position = None

position_unit = 'mm'

temperature = None

temperature_unit = None

thickness = None

thickness_unit = 'mm'

transmission = None

yacceptance = (0, '')

zacceptance = (0, '')

class sas.sascalc.dataloader.data_info.**Source**
Bases: `object`

Class to hold source information

beam_shape = None

beam_size = None

beam_size_name = None

beam_size_unit = 'mm'

name = None

radiation = None

wavelength = None

wavelength_max = None

wavelength_max_unit = 'nm'

wavelength_min = None

wavelength_min_unit = 'nm'

wavelength_spread = None

wavelength_spread_unit = 'percent'

wavelength_unit = 'A'

class sas.sascalc.dataloader.data_info.**TransmissionSpectrum**

Bases: `object`

Class that holds information about transmission spectrum for white beams and spallation sources.

name = ''

timestamp = ''

transmission = None

transmission_deviation = None

transmission_deviation_unit = ''

transmission_unit = ''

wavelength = None

wavelength_unit = 'A'

class sas.sascalc.dataloader.data_info.**Vector**(*x=None*, *y=None*, *z=None*)

Bases: `object`

Vector class to hold multi-dimensional objects

x = None

y = None

z = None

sas.sascalc.dataloader.data_info.**combine_data_info_with_plottable**(*data*, *datainfo*)

A function that combines the DataInfo data in self.current_datainto with a plottable_1D or 2D data object.

Parameters **data** – A plottable_1D or plottable_2D data object

Returns A fully specified Data1D or Data2D object

class sas.sascalc.dataloader.data_info.**plottable_1D**(*x*, *y*, *dx=None*, *dy=None*, *dxl=None*, *dxw=None*, *lam=None*, *dlam=None*)

Bases: `object`

Data1D is a place holder for 1D plottables.

dlam = None

dx = None

dxl = None

dxw = None

dy = None

lam = None

x = None

xaxis(*label*, *unit*)

set the x axis label and unit

y = None

yaxis(*label*, *unit*)

set the y axis label and unit

class sas.sascalc.dataloader.data_info.**plottable_2D**(*data=None*, *err_data=None*, *qx_data=None*, *qy_data=None*, *q_data=None*, *mask=None*, *dqx_data=None*, *dqy_data=None*)

Bases: `object`

Data2D is a place holder for 2D plottables.

data = None

dqx_data = None

dqy_data = None

err_data = None

mask = None

q_data = None

qx_data = None

qy_data = None

xaxis(*label*, *unit*)
set the x axis label and unit

xmax = None

xmin = None

yaxis(*label*, *unit*)
set the y axis label and unit

ymax = None

ymin = None

zaxis(*label*, *unit*)
set the z axis label and unit

sas.sascalc.dataloader.file_reader_base_class module

This is the base file reader class most file readers should inherit from. All generic functionality required for a file loader/reader is built into this class

class sas.sascalc.dataloader.file_reader_base_class.**FileReader**
Bases: `object`

allow_all = False

convert_data_units(*default_q_unit='1/A'*)
Converts al; data to the sasview default of units of A^{-1} for Q and cm^{-1} for I. :param default_q_unit: The default Q unit used by Sasview

data_cleanup()
Clean up the data sets and refresh everything :return: None

deprecated_extensions = ['.asc']

ext = ['.txt']

format_unit(*unit=None*)
Format units a common way :param unit: :return:

get_file_contents()
Reader specific class to access the contents of the file All reader classes that inherit from FileReader must implement

handle_error_message(*msg*)
Generic error handler to add an error to the current datainfo to propagate the error up the error chain. :param msg: Error message

has_converter = True

nextline()
Returns the next line in the file as a string.

nextlines()
Returns the next line in the file as a string.

read(*filepath*)
Basic file reader

Parameters **filepath** – The full or relative path to a file to be loaded

readall()
Returns the entire file as a string.

remove_empty_q_values()
Remove any point where Q == 0

reset_data_list(*no_lines=0*)
Reset the plottable_1D object

reset_state()
Resets the class state to a base case when loading a new data file so previous data files do not appear a second time

send_to_output()
Helper that automatically combines the info and set and then appends it to output

set_all_to_none()
Set all mutable values to None for error handling purposes

static set_default_1d_units(*data*)
Set the x and y axes to the default 1D units :param data: 1D data set :return:

static set_default_2d_units(*data*)
Set the x and y axes to the default 2D units :param data: 2D data set :return:

sort_data()
Sort 1D data along the X axis for consistency

static splitline(*line*)
Splits a line into pieces based on common delimiters :param line: A single line of text :return: list of values

type = ['Text files (*.txt|*.TXT)']

type_name = 'ASCII'

sas.sascalc.dataloader.file_reader_base_class.**decode**(*s*)

sas.sascalc.dataloader.loader module

File handler to support different file extensions. Uses reflectometer registry utility.

The default readers are found in the 'readers' sub-module and registered by default at initialization time.

To add a new default reader, one must register it in the register_readers method found in readers/__init__.py.

A utility method (find_plugins) is available to inspect a directory (for instance, a user plug-in directory) and look for new readers/writers.

class `sas.sascalc.dataloader.loader.`**`Loader`**

Bases: `object`

Utility class to use the Registry as a singleton.

`associate_file_reader`(*ext*, *loader*)

Append a reader object to readers

Parameters

- **`ext`** – file extension [string]
- **`module`** – reader object

`associate_file_type`(*ext*, *module*)

Look into a module to find whether it contains a Reader class. If so, append it to readers and (potentially) to the list of writers for the given extension

Parameters

- **`ext`** – file extension [string]
- **`module`** – module object

`find_plugins`(*directory*)

Find plugins in a given directory

Parameters **`dir`** – directory to look into to find new readers/writers

`get_wildcards`()

Return the list of wildcards

`load`(*file*, *format=None*)

Load a file

Parameters

- **`file`** – file name (path)
- **`format`** – specified format to use (optional)

Returns DataInfo object

`save`(*file*, *data*, *format*)

Save a DataInfo object to file :param file: file name (path) :param data: DataInfo object :param format: format to write the data in

class `sas.sascalc.dataloader.loader.`**`Registry`**

Bases: *`sas.sascalc.data_util.registry.ExtensionRegistry`*

Registry class for file format extensions. Readers and writers are supported.

`associate_file_reader`(*ext*, *loader*)

Append a reader object to readers

Parameters

- **`ext`** – file extension [string]
- **`module`** – reader object

`associate_file_type`(*ext*, *module*)

Look into a module to find whether it contains a Reader class. If so, APPEND it to readers and (potentially) to the list of writers for the given extension

Parameters

- **`ext`** – file extension [string]
- **`module`** – module object

find_plugins(*dir*)
Find readers in a given directory. This method can be used to inspect user plug-in directories to find new readers/writers.

Parameters **dir** – directory to search into

Returns number of readers found

load(*path*, *format=None*)
Call the loader for the file type of path.

Parameters

- **path** – file path
- **format** – explicit extension, to force the use of a particular reader

Defaults to the ascii (multi-column), cansas XML, and cansas NeXuS readers if no reader was registered for the file's extension.

lookup_writers(*path*)

Returns the loader associated with the file type of path.

Raises ValueError if file type is not known.

save(*path*, *data*, *format=None*)
Call the writer for the file type of path.

Raises ValueError if no writer is available. Raises KeyError if format is not available. May raise a writer-defined exception if writer fails.

sas.sascalc.dataloader.loader_exceptions module

Exceptions specific to loading data.

exception `sas.sascalc.dataloader.loader_exceptions.`**DataReaderException**(*e=None*)
Bases: `exceptions.Exception`

Exception for files that were able to mostly load, but had minor issues along the way. Any exceptions of this type should be put into the datainfo.errors

exception `sas.sascalc.dataloader.loader_exceptions.`**DefaultReaderException**(*e=None*)
Bases: `exceptions.Exception`

Exception for files with no associated reader. This should be thrown by default readers only to tell Loader to try the next reader.

exception `sas.sascalc.dataloader.loader_exceptions.`**FileContentsException**(*e=None*)
Bases: `exceptions.Exception`

Exception for files with an associated reader, but with no loadable data. This is useful for catching loader or file format issues.

exception `sas.sascalc.dataloader.loader_exceptions.`**NoKnownLoaderException**(*e=None*)
Bases: `exceptions.Exception`

Exception for files with no associated reader based on the file extension of the loaded file. This exception should only be thrown by loader.py.

sas.sascalc.dataloader.manipulations module

class `sas.sascalc.dataloader.manipulations.`**Binning**(*min_value*, *max_value*, *n_bins*, *base=None*)
Bases: `object`

This class just creates a binning object either linear or log

get_bin_index(*value*)
The general formula logarithm binning is: bin = floor(N * (log(x) - log(min)) / (log(max) - log(min)))

class sas.sascalc.dataloader.manipulations.**Boxavg**(*x_min=0.0*, *x_max=0.0*, *y_min=0.0*, *y_max=0.0*)
Bases: `sas.sascalc.dataloader.manipulations.Boxsum`

Perform the average of counts in a 2D region of interest.

class sas.sascalc.dataloader.manipulations.**Boxcut**(*x_min=0.0*, *x_max=0.0*, *y_min=0.0*, *y_max=0.0*)
Bases: `object`

Find a rectangular 2D region of interest.

class sas.sascalc.dataloader.manipulations.**Boxsum**(*x_min=0.0*, *x_max=0.0*, *y_min=0.0*, *y_max=0.0*)
Bases: `object`

Perform the sum of counts in a 2D region of interest.

class sas.sascalc.dataloader.manipulations.**CircularAverage**(*r_min=0.0*, *r_max=0.0*, *bin_width=0.0005*)
Bases: `object`

Perform circular averaging on 2D data

The data returned is the distribution of counts as a function of Q

class sas.sascalc.dataloader.manipulations.**Ring**(*r_min=0*, *r_max=0*, *center_x=0*, *center_y=0*, *nbins=36*)
Bases: `object`

Defines a ring on a 2D data set. The ring is defined by r_min, r_max, and the position of the center of the ring.

The data returned is the distribution of counts around the ring as a function of phi.

Phi_min and phi_max should be defined between 0 and 2*pi in anti-clockwise starting from the x- axis on the left-hand side

class sas.sascalc.dataloader.manipulations.**Ringcut**(*r_min=0*, *r_max=0*, *center_x=0*, *center_y=0*)
Bases: `object`

Defines a ring on a 2D data set. The ring is defined by r_min, r_max, and the position of the center of the ring.

The data returned is the region inside the ring

Phi_min and phi_max should be defined between 0 and 2*pi in anti-clockwise starting from the x- axis on the left-hand side

class sas.sascalc.dataloader.manipulations.**SectorPhi**(*r_min*, *r_max*, *phi_min=0*, *phi_max=6.283185307179586*, *nbins=20*, *base=None*)
Bases: `sas.sascalc.dataloader.manipulations._Sector`

Sector average as a function of phi. I(phi) is return and the data is averaged over Q.

A sector is defined by r_min, r_max, phi_min, phi_max. The number of bin in phi also has to be defined.

class sas.sascalc.dataloader.manipulations.**SectorQ**(*r_min*, *r_max*, *phi_min=0*, *phi_max=6.283185307179586*, *nbins=20*, *base=None*)
Bases: `sas.sascalc.dataloader.manipulations._Sector`

Sector average as a function of Q for both symatric wings. I(Q) is return and the data is averaged over phi.

A sector is defined by r_min, r_max, phi_min, phi_max. r_min, r_max, phi_min, phi_max >0. The number of bin in Q also has to be defined.

class sas.sascalc.dataloader.manipulations.**Sectorcut**(*phi_min=0*, *phi_max=3.141592653589793*)

Bases: `object`

Defines a sector (major + minor) region on a 2D data set. The sector is defined by phi_min, phi_max, where phi_min and phi_max are defined by the right and left lines wrt central line.

Phi_min and phi_max are given in units of radian and (phi_max-phi_min) should not be larger than pi

class sas.sascalc.dataloader.manipulations.**SlabX**(*x_min=0.0*, *x_max=0.0*, *y_min=0.0*, *y_max=0.0*, *bin_width=0.001*)

Bases: `sas.sascalc.dataloader.manipulations._Slab`

Compute average I(Qx) for a region of interest

class sas.sascalc.dataloader.manipulations.**SlabY**(*x_min=0.0*, *x_max=0.0*, *y_min=0.0*, *y_max=0.0*, *bin_width=0.001*)

Bases: `sas.sascalc.dataloader.manipulations._Slab`

Compute average I(Qy) for a region of interest

sas.sascalc.dataloader.manipulations.**flip_phi**(*phi*)

Correct phi to within the 0 <= to <= 2pi range

Returns phi in >=0 and <=2Pi

sas.sascalc.dataloader.manipulations.**get_dq_data**(*data2D*)

Get the dq for resolution averaging The pinholes and det. pix contribution present in both direction of the 2D which must be subtracted when converting to 1D: dq_overlap should calculated ideally at q = 0. Note This method works on only pinhole geometry. Extrapolate dqx(r) and dqy(phi) at q = 0, and take an average.

sas.sascalc.dataloader.manipulations.**get_intercept**(*q*, *q_0*, *q_1*)

Returns the fraction of the side at which the q-value intercept the pixel, None otherwise. The values returned is the fraction ON THE SIDE OF THE LOWEST Q.

```
    A           B
+-----------+--------+    <--- pixel size
0                    1
Q_0 -------- Q ----- Q_1   <--- equivalent Q range
if Q_1 > Q_0, A is returned
if Q_1 < Q_0, B is returned
if Q is outside the range of [Q_0, Q_1], None is returned
```

sas.sascalc.dataloader.manipulations.**get_pixel_fraction**(*qmax*, *q_00*, *q_01*, *q_10*, *q_11*)

Returns the fraction of the pixel defined by the four corners (q_00, q_01, q_10, q_11) that has q < qmax.:

```
        q_01                q_11
y=1         +--------------+
            |              |
            |              |
            |              |
y=0         +--------------+
        q_00                q_10

            x=0            x=1
```

sas.sascalc.dataloader.manipulations.**get_pixel_fraction_square**(*x*, *xmin*, *xmax*)

Return the fraction of the length from xmin to x.:

```
    A             B
+-----------+---------+
xmin        x         xmax
```

Parameters

- **x** – x-value
- **xmin** – minimum x for the length considered
- **xmax** – minimum x for the length considered

Returns (x-xmin)/(xmax-xmin) when xmin < x < xmax

`sas.sascalc.dataloader.manipulations.`**`get_q`**(*dx*, *dy*, *det_dist*, *wavelength*)

Parameters

- **dx** – x-distance from beam center [mm]
- **dy** – y-distance from beam center [mm]

Returns q-value at the given position

`sas.sascalc.dataloader.manipulations.`**`get_q_compo`**(*dx*, *dy*, *det_dist*, *wavelength*, *compo=None*)

This reduces tiny error at very large q. Implementation of this func is not started yet.<–ToDo

`sas.sascalc.dataloader.manipulations.`**`reader2D_converter`**(*data2d=None*)

convert old 2d format opened by IhorReader or danse_reader to new Data2D format This is mainly used by the Readers

Parameters **data2d** – 2d array of Data2D object

Returns 1d arrays of Data2D object

Module contents

sas.sascalc.file_converter package

Submodules

sas.sascalc.file_converter.ascii2d_loader module

ASCII 2D Loader

class `sas.sascalc.file_converter.ascii2d_loader.`**`ASCII2DLoader`**(*data_path*)

Bases: `object`

`load`()

Load the data from the file into a Data2D object

Returns A Data2D instance containing data from the file

Raises **ValueError** – Raises a ValueError if the file is incorrectly formatted

sas.sascalc.file_converter.bsl_loader module

class `sas.sascalc.file_converter.bsl_loader.`**`BSLLoader`**(*filename*)

Bases: `CLoader`

Loads 2D SAS data from a BSL file. CLoader is a C extension (found in c_ext/bsl_loader.c)

See http://www.diamond.ac.uk/Beamlines/Soft-Condensed-Matter/small-angle/SAXS-Software/CCP13/BSL.html for more info on the BSL file format.

load_frames (*frames*)

exception sas.sascalc.file_converter.bsl_loader.**BSLParsingError**
Bases: exceptions.Exception

sas.sascalc.file_converter.cansas_writer module

class sas.sascalc.file_converter.cansas_writer.**CansasWriter** (*xml=None*, *schema=None*)
Bases: *sas.sascalc.dataloader.readers.cansas_reader.Reader*

write (*filename*, *frame_data*, *sasentry_attrs=None*)
Write the content of a Data1D as a CanSAS XML file

Parameters

- **filename** – name of the file to write
- **datainfo** – Data1D object

sas.sascalc.file_converter.nxcansas_writer module

NXcanSAS 1/2D data reader for writing HDF5 formatted NXcanSAS files.

class sas.sascalc.file_converter.nxcansas_writer.**NXcanSASWriter**
Bases: *sas.sascalc.dataloader.readers.cansas_reader_HDF5.Reader*

A class for writing in NXcanSAS data files. Any number of data sets may be written to the file. Currently 1D and 2D SAS data sets are supported

NXcanSAS spec: http://download.nexusformat.org/sphinx/classes/contributed_definitions/NXcanSAS.html

Dependencies The NXcanSAS writer requires h5py => v2.5.0 or later.

write (*dataset*, *filename*)
Write an array of Data1d or Data2D objects to an NXcanSAS file, as one SASEntry with multiple SASData elements. The metadata of the first elememt in the array will be written as the SASentry metadata (detector, instrument, sample, etc).

Parameters

- **dataset** – A list of Data1D or Data2D objects to write
- **filename** – Where to write the NXcanSAS file

sas.sascalc.file_converter.otoko_loader module

Here we handle loading of "OTOKO" data (for more info about this format see the comment in load_otoko_data). Given the paths of header and data files, we aim to load the data into numpy arrays for use later.

class sas.sascalc.file_converter.otoko_loader.**CStyleStruct** (***kwds*)
A nice and easy way to get "C-style struct" functionality.

class sas.sascalc.file_converter.otoko_loader.**OTOKOData** (*q_axis*, *data_axis*)

class sas.sascalc.file_converter.otoko_loader.**OTOKOLoader** (*qaxis_path*, *data_path*)
Bases: object

load_otoko_data()

Loads "OTOKO" data, which is a format that stores each axis separately. An axis is represented by a "header" file, which in turn will give details of one or more binary files where the actual data is stored.

Given the paths of two header files, this function will load each axis in turn. If loading is successful then an instance of the OTOKOData class will be returned, else an exception will be raised.

For more information on the OTOKO file format, please see: http://www.diamond.ac.uk/Home/Beamlines/small-angle/SAXS-Software/CCP13/ XOTOKO.html

exception `sas.sascalc.file_converter.otoko_loader.`**OTOKOParsingError**

Bases: `exceptions.Exception`

sas.sascalc.file_converter.red2d_writer module

class `sas.sascalc.file_converter.red2d_writer.`**Red2DWriter**

Bases: *`sas.sascalc.dataloader.readers.red2d_reader.Reader`*

write(*filename*, *data*, *thread*)

Write to .dat

Parameters

- **filename** – file name to write
- **data** – data2D

Module contents

sas.sascalc.fit package

Submodules

sas.sascalc.fit.AbstractFitEngine module

class `sas.sascalc.fit.AbstractFitEngine.`**FResult**(*model=None*, *param_list=None*, *data=None*)

Bases: `object`

Storing fit result

print_summary()

set_fitness(*fitness*)

set_model(*model*)

exception `sas.sascalc.fit.AbstractFitEngine.`**FitAbort**

Bases: `exceptions.Exception`

Exception raise to stop the fit

class `sas.sascalc.fit.AbstractFitEngine.`**FitArrange**

add_data(*data*)

add_data fill a self.data_list with data to fit

Parameters **data** – Data to add in the list

get_data()

Returns list of data data_list

get_model()

Returns saved model

get_to_fit()
return self.selected value

remove_data(*data*)
Remove one element from the list

Parameters **data** – Data to remove from data_list

set_model(*model*)
set_model save a copy of the model

Parameters **model** – the model being set

set_to_fit(*value=0*)
set self.selected to 0 or 1 for other values raise an exception

Parameters **value** – integer between 0 or 1

class sas.sascalc.fit.AbstractFitEngine.**FitData1D**(*x*, *y*, *dx=None*, *dy=None*, *smearer=None*, *data=None*, *lam=None*, *dlam=None*)
Bases: *sas.sascalc.dataloader.data_info.Data1D*

Wrapper class for SAS data FitData1D inherits from DataLoader.data_info.Data1D. Implements a way to get residuals from data.

get_fit_range()
Return the range of data.x to fit

residuals(*fn*)
Compute residuals.

If self.smearer has been set, use if to smear the data before computing chi squared.

Parameters **fn** – function that return model value

Returns residuals

residuals_deriv(*model*, *pars=[]*)

Returns residuals derivatives .

Note in this case just return empty array

set_fit_range(*qmin=None*, *qmax=None*)
to set the fit range

size()
Number of measurement points in data set after masking, etc.

class sas.sascalc.fit.AbstractFitEngine.**FitData2D**(*sas_data2d*, *data=None*, *err_data=None*)
Bases: *sas.sascalc.dataloader.data_info.Data2D*

Wrapper class for SAS data

get_fit_range()
return the range of data.x to fit

residuals(*fn*)
return the residuals

residuals_deriv(*model*, *pars=[]*)

Returns residuals derivatives .

Note in this case just return empty array

set_data(*sas_data2d*, *qmin=None*, *qmax=None*)
Determine the correct qx_data and qy_data within range to fit

set_fit_range(*qmin=None*, *qmax=None*)
To set the fit range

set_smearer(*smearer*)
Set smearer

size()
Number of measurement points in data set after masking, etc.

class sas.sascalc.fit.AbstractFitEngine.**FitEngine**

get_model(*id*)

Parameters **id** – id is key in the dictionary containing the model to return

Returns a model at this id or None if no FitArrange element was created with this id

get_problem_to_fit(*id*)
return the self.selected value of the fit problem of id

Parameters **id** – the id of the problem

remove_fit_problem(*id*)
remove fitarrange in id

select_problem_for_fit(*id*, *value*)
select a couple of model and data at the id position in dictionary and set in self.selected value to value

Parameters **value** – the value to allow fitting. can only have the value one or zero

set_data(*data*, *id*, *smearer=None*, *qmin=None*, *qmax=None*)
Receives plottable, creates a list of data to fit,set data in a FitArrange object and adds that object in a dictionary with key id.

Parameters

- **data** – data added
- **id** – unique key corresponding to a fitArrange object with data

set_model(*model*, *id*, *pars=[]*, *constraints=[]*, *data=None*)
set a model on a given in the fit engine.

Parameters

- **model** – sas.models type
- **id** – is the key of the fitArrange dictionary where model is saved as a value
- **pars** – the list of parameters to fit
- **constraints** – list of tuple (name of parameter, value of parameters) the value of parameter must be a string to constraint 2 different parameters. Example: we want to fit 2 model M1 and M2 both have parameters A and B. constraints can be `constraints = [(M1.A, M2.B+2), (M1.B= M2.A *5),...,]`

Note pars must contains only name of existing model's parameters

class sas.sascalc.fit.AbstractFitEngine.**FitHandler**
Bases: `object`

Abstract interface for fit thread handler.

The methods in this class are called by the optimizer as the fit progresses.

Note that it is up to the optimizer to call the fit handler correctly, reporting all status changes and maintaining the 'done' flag.

abort()
Fit was aborted.

done = False
True when the fit job is complete

error(*msg*)
Model had an error; print traceback

finalize()
Fit is complete; best results are reported

improvement()
Called when a result is observed which is better than previous results from the fit.

result is a FitResult object, with parameters, #calls and fitness.

progress(*current*, *expected*)
Called each cycle of the fit, reporting the current and the expected amount of work. The meaning of these values is optimizer dependent, but they can be converted into a percent complete using (100*current)//expected.

Progress is updated each iteration of the fit, whatever that means for the particular optimization algorithm. It is called after any calls to improvement for the iteration so that the update handler can control I/O bandwidth by suppressing intermediate improvements until the fit is complete.

result = None
The current best result of the fit

set_result(*result=None*)

update_fit(*last=False*)

class sas.sascalc.fit.AbstractFitEngine.**Model**(*sas_model*, *sas_data=None*, ***kw*)
Fit wrapper for SAS models.

eval(*x*)
Override eval method of model.

Parameters **x** – the x value used to compute a function

eval_derivs(*x*, *pars=[]*)
Evaluate the model and derivatives wrt pars at x.

pars is a list of the names of the parameters for which derivatives are desired.

This method needs to be specialized in the model to evaluate the model function. Alternatively, the model can implement is own version of residuals which calculates the residuals directly instead of calling eval.

get_params(*fitparams*)
return a list of value of parameter to fit

Parameters **fitparams** – list of parameters name to fit

set(***kw*)

set_params(*paramlist*, *params*)
Set value for parameters to fit

Parameters **params** – list of value for parameters to fit

sas.sascalc.fit.BumpsFitting module

BumpsFitting module runs the bumps optimizer.

class sas.sascalc.fit.BumpsFitting.**BumpsFit**
Bases: *sas.sascalc.fit.AbstractFitEngine.FitEngine*

Fit a model using bumps.

fit(*msg_q=None*, *q=None*, *handler=None*, *curr_thread=None*, *ftol=1.49012e-08*, *reset_flag=False*)

class `sas.sascalc.fit.BumpsFitting.`**BumpsMonitor**(*handler*, *max_step*, *pars*, *dof*)
Bases: `object`

config_history(*history*)

class `sas.sascalc.fit.BumpsFitting.`**ConvergenceMonitor**
Bases: `object`

ConvergenceMonitor contains population summary statistics to show progress of the fit. This is a list [(best, 0%, 25%, 50%, 75%, 100%)] or just a list [(best,)] if population size is 1.

config_history(*history*)

class `sas.sascalc.fit.BumpsFitting.`**ParameterExpressions**(*models*)
Bases: `object`

class `sas.sascalc.fit.BumpsFitting.`**Progress**(*history*, *max_step*, *pars*, *dof*)
Bases: `object`

class `sas.sascalc.fit.BumpsFitting.`**SasFitness**(*model*, *data*, *fitted=[]*, *constraints={}*, *initial_values=None*, ***kw*)
Bases: `object`

Wrap SAS model as a bumps fitness object

nllf()

numpoints()

parameters()

residuals()

set_fitted(*param_list*)
Flag a set of parameters as fitted parameters.

theory()

update()

`sas.sascalc.fit.BumpsFitting.`**get_fitter**()

`sas.sascalc.fit.BumpsFitting.`**run_bumps**(*problem*, *handler*, *curr_thread*)

sas.sascalc.fit.Loader module

class `sas.sascalc.fit.Loader.`**Load**(*x=None*, *y=None*, *dx=None*, *dy=None*)
This class is loading values from given file or value giving by the user

get_filename()
return the file's path

get_values()
Return x, y, dx, dy

load_data(*data*)
Return plottable

set_filename(*path=None*)
Store path into a variable.If the user doesn't give a path as a parameter a pop-up window appears to select the file.

Parameters **path** – the path given by the user

set_values()
Store the values loaded from file in local variables

sas.sascalc.fit.MultiplicationModel module

class sas.sascalc.fit.MultiplicationModel.**MultiplicationModel**(*p_model*, *s_model*)

Bases: *sas.sascalc.calculator.BaseComponent.BaseComponent*

Use for P(Q)*S(Q); function call must be in the order of P(Q) and then S(Q): The model parameters are combined from both models, P(Q) and S(Q), except 1) 'radius_effective' of S(Q) which will be calculated from P(Q) via calculate_ER(), and 2) 'scale' in P model which is synchronized w/ volfraction in S then P*S is multiplied by a new parameter, 'scale_factor'. The polydispersion is applicable only to P(Q), not to S(Q).

Note: P(Q) refers to 'form factor' model while S(Q) does to 'structure factor'.

evalDistribution(*x=[]*)

Evaluate the model in cartesian coordinates

Parameters **x** – input q[], or [qx[], qy[]]

Returns scattering function P(q[])

fill_description(*p_model*, *s_model*)

Fill the description for P(Q)*S(Q)

getProfile()

Get SLD profile of p_model if exists

Returns (r, beta) where r is a list of radius of the transition points beta is a list of the corresponding SLD values

Note: This works only for func_shell num = 2 (exp function).

run(*x=0.0*)

Evaluate the model

Parameters **x** – input q-value (float or [float, float] as [r, theta])

Returns (scattering function value)

runXY(*x=0.0*)

Evaluate the model

Parameters **x** – input q-value (float or [float, float] as [qx, qy])

Returns scattering function value

setParam(*name*, *value*)

Set the value of a model parameter

Parameters

- **name** – name of the parameter
- **value** – value of the parameter

set_dispersion(*parameter*, *dispersion*)

Set the dispersion object for a model parameter

Parameters **parameter** – name of the parameter [string]

Dispersion dispersion object of type DispersionModel

sas.sascalc.fit.expression module

Parameter expression evaluator.

For systems in which constraints are expressed as string expressions rather than python code, `compile_constraints()` can construct an expression evaluator that substitutes the computed values of the expressions into the parameters.

The compiler requires a symbol table, an expression set and a context. The symbol table maps strings containing fully qualified names such as 'M1.c[3].full_width' to parameter objects with a 'value' property that can be queried and set. The expression set maps symbol names from the symbol table to string expressions. The context provides additional symbols for the expressions in addition to the usual mathematical functions and constants.

The expressions are compiled and interpreted by python, with only minimal effort to make sure that they don't contain bad code. The resulting constraints function returns 0 so it can be used directly in a fit problem definition.

Extracting the symbol table from the model depends on the structure of the model. If fitness.parameters() is set correctly, then this should simply be a matter of walking the parameter data, remembering the path to each parameter in the symbol table. For compactness, dictionary elements should be referenced by .name rather than ["name"]. Model name can be used as the top level.

Getting the parameter expressions applied correctly is challenging. The following monkey patch works by overriding model_update in FitProblem so that after setp(p) is called and, the constraints expression can be applied before telling the underlying fitness function that the model is out of date:

```
# Override model update so that parameter constraints are applied
problem._model_update = problem.model_update
def model_update():
    constraints()
    problem._model_update()
problem.model_update = model_update
```

Ideally, this interface will change

`sas.sascalc.fit.expression.`**`compile_constraints`**(*symtab*, *exprs*, *context={}*)

Build and return a function to evaluate all parameter expressions in the proper order.

Input:

symtab is the symbol table for the model: { 'name': parameter }

exprs is the set of computed symbols: { 'name': 'expression' }

context is any additional context needed to evaluate the expression

Return:

updater function which sets parameter.value for each expression

Raises:

AssertionError - model, parameter or function is missing

SyntaxError - improper expression syntax

ValueError - expressions have circular dependencies

This function is not terribly sophisticated, and it would be easy to trick. However it handles the common cases cleanly and generates reasonable messages for the common errors.

This code has not been fully audited for security. While we have removed the builtins and the ability to import modules, there may be other vectors for users to perform more than simple function evaluations. Unauthenticated users should not be running this code.

Parameter names are assumed to contain only _.a-zA-Z0-9#[]

Both names are provided for inverse functions, e.g., acos and arccos.

Should try running the function to identify syntax errors before running it in a fit.

Use help(fn) to see the code generated for the returned function fn. dis.dis(fn) will show the corresponding python vm instructions.

`sas.sascalc.fit.expression.`**`no_constraints`**`()`
This parameter set has no constraints between the parameters.

`sas.sascalc.fit.expression.`**`order_dependencies`**`(`*pairs*`)`
Order elements from pairs so that b comes before a in the ordered list for all pairs (a,b).

`sas.sascalc.fit.expression.`**`test_deps`**`()`

`sas.sascalc.fit.expression.`**`test_expr`**`()`

sas.sascalc.fit.models module

Utilities to manage models

`class` `sas.sascalc.fit.models.`**`ModelManager`**
Bases: `object`

manage the list of available models

`base = None`

`cat_model_list``()`

`composable_models``()`

`get_model_dictionary``()`

`get_model_list``()`

`plugins_reset``()`

`update``()`

`class` `sas.sascalc.fit.models.`**`ModelManagerBase`**
Bases: `object`

Base class for the model manager

`composable_models``()`
return list of standard models that can be used in sum/multiply

`get_model_list``()`
return dictionary of classified models

Structure Factors are the structure factor models *Multi-Functions* are the multiplicity models *Plugin Models* are the plugin models

Note that a model can be both a plugin and a structure factor or multiplicity model.

`last_time_dir_modified = 0`
timestamp on the plugin directory at the last plugin update

`model_dictionary = None`
mutable dictionary of models, continually updated to reflect the current set of plugins

`plugin_models = None`
list of plugin models reset each time the plugin directory is queried

`plugins_reset``()`
return a dictionary of model

`plugins_update``()`
return a dictionary of model if new models were added else return empty dictionary

`standard_models = None`
constant list of standard models

class sas.sascalc.fit.models.**ReportProblem**
Bases: object

Class to check for problems with specific values

sas.sascalc.fit.models.**compile_file**(*dir*)
Compile a py file

sas.sascalc.fit.models.**find_plugin_models**()
Find custom models

sas.sascalc.fit.models.**find_plugins_dir**()
Find path of the plugins directory. The plugin directory is located in the user's home directory.

sas.sascalc.fit.models.**initialize_plugins_dir**(*path*)

sas.sascalc.fit.models.**plugin_log**(*message*)
Log a message in a file located in the user's home directory

sas.sascalc.fit.pagestate module

Class that holds a fit page state

class sas.sascalc.fit.pagestate.**PageState**(*model=None*, *data=None*)
Bases: object

Contains information to reconstruct a page of the fitpanel.

clone()
Create a new copy of the current object

from_xml(*file=None*, *node=None*)
Load fitting state from a file

Parameters

- **file** – .fitv file
- **node** – node of a XML document to read from

static param_remap_from_sasmodels_convert(*params*)
Converts {name : value} map back to [] param list :param params: parameter map returned from sasmodels :return: None

static param_remap_to_sasmodels_convert(*params*, *is_string=False*)
Remaps the parameters for sasmodels conversion

Parameters **params** – list of parameters (likely self.parameters)

Returns remapped dictionary of parameters

report(*fig_urls*)
Invoke report dialog panel

: param figs: list of pylab figures [list]

to_xml(*file='fitting_state.fitv'*, *doc=None*, *entry_node=None*, *batch_fit_state=None*)
Writes the state of the fit panel to file, as XML.

Compatible with standalone writing, or appending to an already existing XML document. In that case, the XML document is required. An optional entry node in the XML document may also be given.

Parameters

- **file** – file to write to
- **doc** – XML document object [optional]
- **entry_node** – XML node within the XML document at which we will append the data [optional]

- **batch_fit_state** – simultaneous fit state

class sas.sascalc.fit.pagestate.**Reader**(*call_back=None*, *cansas=True*)
Bases: *sas.sascalc.dataloader.readers.cansas_reader.Reader*

Class to load a .fitv fitting file

ext = ['.fitv', '.FITV', '.svs', 'SVS']

get_state()

read(*path*)
Load a new P(r) inversion state from file

Parameters path – file path

type = ['Fitting files (*.fitv)|*.fitvSASView file (*.svs)|*.svs']

type_name = 'Fitting'

write(*filename*, *datainfo=None*, *fitstate=None*)
Write the content of a Data1D as a CanSAS XML file only for standalone

Parameters

- **filename** – name of the file to write
- **datainfo** – Data1D object
- **fitstate** – PageState object

write_toXML(*datainfo=None*, *state=None*, *batchfit=None*)
Write toXML, a helper for write(), could be used by guimanager._on_save()

: return: xml doc

class sas.sascalc.fit.pagestate.**SimFitPageState**
Bases: object

State of the simultaneous fit page for saving purposes

sas.sascalc.fit.pagestate.**parse_entry_helper**(*node*, *item*)
Create a numpy list from value extrated from the node

Parameters

- **node** – node from each the value is stored
- **item** – list name of three strings.the two first are name of data attribute and the third one is the type of the value of that attribute. type can be string, float, bool, etc.

: return: numpy array

sas.sascalc.fit.pluginmodel module

class sas.sascalc.fit.pluginmodel.**Model1DPlugin**(*name='Plugin Model'*)
Bases: *sas.sascalc.calculator.BaseComponent.BaseComponent*

function(*x*)
Function to be implemented by the plug-in writer

is_multiplicity_model = False

run(*x=0.0*)
Evaluate the model

Parameters x – input x, or [x, phi] [radian]

Returns function value

runXY (*x=0.0*)
Evaluate the model

Parameters **x** – input x, or [x, y]

Returns function value

set_details ()
Set default details

sas.sascalc.fit.qsmearing module

Handle Q smearing

class sas.sascalc.fit.qsmearing.**PySmear** (*resolution*, *model*, *offset=None*)
Bases: object

Wrapper for pure python sasmodels resolution functions.

apply (*iq_in*, *first_bin=0*, *last_bin=None*)
Apply the resolution function to the data. Note that this is called with iq_in matching data.x, but with iq_in[first_bin:last_bin] set to theory values for these bins, and the remainder left undefined. The first_bin, last_bin values should be those returned from get_bin_range. The returned value is of the same length as iq_in, with the range first_bin:last_bin set to the resolution smeared values.

get_bin_range (*q_min=None*, *q_max=None*)
For a given q_min, q_max, find the corresponding indices in the data. Returns first, last. Note that these are indexes into q from the data, not the q_calc needed by the resolution function. Note also that these are the indices, not the range limits. That is, the complete range will be q[first:last+1].

class sas.sascalc.fit.qsmearing.**PySmear2D** (*data=None*, *model=None*)
Bases: object

Q smearing class for SAS 2d pinhole data

get_value ()
Over sampling of r_nbins times phi_nbins, calculate Gaussian weights, then find smeared intensity

set_accuracy (*accuracy='Low'*)
Set accuracy.

Parameters **accuracy** – string

set_data (*data=None*)
Set data.

Parameters **data** – DataLoader.Data_info type

set_index (*index=None*)
Set index.

Parameters **index** – 1d arrays

set_model (*model=None*)
Set model.

Parameters **model** – sas.models instance

set_smearer (*smearer=True*)
Set whether or not smearer will be used

Parameters **smearer** – smear object

sas.sascalc.fit.qsmearing.**pinhole_smear** (*data*, *model=None*)

sas.sascalc.fit.qsmearing.**slit_smear** (*data*, *model=None*) `sas.sascalc.fit.qsmearing.`**`smear_selection`**(*data*, *model=None*)

Creates the right type of smearer according to the data. The canSAS format has a rule that either slit smearing data OR resolution smearing data is available.

For the present purpose, we choose the one that has none-zero data. If both slit and resolution smearing arrays are filled with good data (which should not happen), then we choose the resolution smearing data.

Parameters

- **`data`** – Data1D object
- **`model`** – sas.model instance

Module contents

sas.sascalc.invariant package

Submodules

sas.sascalc.invariant.invariant module

This module implements invariant and its related computations.

author Gervaise B. Alina/UTK

author Mathieu Doucet/UTK

author Jae Cho/UTK

`class` `sas.sascalc.invariant.invariant.`**`Extrapolator`**(*data*, *model=None*)

Bases: `object`

Extrapolate I(q) distribution using a given model

`fit`(*power=None*, *qmin=None*, *qmax=None*)

Fit data for y = ax + b return a and b

Parameters

- **`power`** – a fixed, otherwise None
- **`qmin`** – Minimum Q-value
- **`qmax`** – Maximum Q-value

`class` `sas.sascalc.invariant.invariant.`**`Guinier`**(*scale=1*, *radius=60*)

Bases: *`sas.sascalc.invariant.invariant.Transform`*

class of type Transform that performs operations related to guinier function

`evaluate_model`(*x*)

return F(x)= scale* e-((radius*x)**2/3)

`evaluate_model_errors`(*x*)

Returns the error on I(q) for the given array of q-values

Parameters **`x`** – array of q-values

`extract_model_parameters`(*constant*, *slope*, *dconstant=0*, *dslope=0*)

assign new value to the scale and the radius

`linearize_q_value`(*value*)

Transform the input q-value for linearization

Parameters **`value`** – q-value

Returns q*q

class sas.sascalc.invariant.invariant.**InvariantCalculator**(*data*, *background=0*, *scale=1*)

Bases: `object`

Compute invariant if data is given. Can provide volume fraction and surface area if the user provides Porod constant and contrast values.

Precondition the user must send a data of type DataLoader.Data1D the user provide background and scale values.

Note Some computations depends on each others.

get_data()

Returns self._data

get_extra_data_high(*npts_in=None*, *q_end=10*, *npts=20*)

Returns the extrapolated data used for the high-Q invariant calculation. By default, the distribution will cover the data points used for the extrapolation. The number of overlap points is a parameter (npts_in). By default, the maximum q-value of the distribution will be Q_MAXIMUM, the maximum q-value used when extrapolating for the purpose of the invariant calculation.

Parameters

- **npts_in** – number of data points for which the extrapolated data overlap
- **q_end** – is the maximum value to uses for extrapolated data
- **npts** – the number of points in the extrapolated distribution

get_extra_data_low(*npts_in=None*, *q_start=None*, *npts=20*)

Returns the extrapolated data used for the loew-Q invariant calculation. By default, the distribution will cover the data points used for the extrapolation. The number of overlap points is a parameter (npts_in). By default, the maximum q-value of the distribution will be the minimum q-value used when extrapolating for the purpose of the invariant calculation.

Parameters

- **npts_in** – number of data points for which the extrapolated data overlap
- **q_start** – is the minimum value to uses for extrapolated data
- **npts** – the number of points in the extrapolated distribution

get_extrapolation_power(*range='high'*)

Returns the fitted power for power law function for a given extrapolation range

get_qstar(*extrapolation=None*)

Compute the invariant of the local copy of data.

Parameters **extrapolation** – string to apply optional extrapolation

Return q_star invariant of the data within data's q range

Warning When using setting data to Data1D , the user is responsible of checking that the scale and the background are properly apply to the data

get_qstar_high()

Compute the invariant for extrapolated data at high q range.

Implementation: data = self._get_extra_data_high() return self._get_qstar()

Return q_star the invariant for data extrapolated at high q.

get_qstar_low()

Compute the invariant for extrapolated data at low q range.

Implementation: data = self._get_extra_data_low() return self._get_qstar()

Return q_star the invariant for data extrapolated at low q.

get_qstar_with_error (*extrapolation=None*)

Compute the invariant uncertainty. This uncertainty computation depends on whether or not the data is smeared.

Parameters **extrapolation** – string to apply optional extrapolation

Returns invariant, the invariant uncertainty

get_surface (*contrast*, *porod_const*, *extrapolation=None*)

Compute the specific surface from the data.

Implementation:

```
V =  self.get_volume_fraction(contrast, extrapolation)

Compute the surface given by:
  surface = (2*pi *V(1- V)*porod_const)/ q_star
```

Parameters

- **contrast** – contrast value to compute the volume
- **porod_const** – Porod constant to compute the surface
- **extrapolation** – string to apply optional extrapolation

Returns specific surface

get_surface_with_error (*contrast*, *porod_const*, *extrapolation=None*)

Compute uncertainty of the surface value as well as the surface value. The uncertainty is given as follow:

```
dS = porod_const *2*pi[( dV -2*V*dV)/q_star
     + dq_star(v-v**2)

q_star: the invariant value
dq_star: the invariant uncertainty
V: the volume fraction value
dV: the volume uncertainty
```

Parameters

- **contrast** – contrast value
- **porod_const** – porod constant value
- **extrapolation** – string to apply optional extrapolation

Return S, dS the surface, with its uncertainty

get_volume_fraction (*contrast*, *extrapolation=None*)

Compute volume fraction is deduced as follow:

```
q_star = 2*(pi*contrast)**2* volume( 1- volume)
for k = 10^(-8)*q_star/(2*(pi*|contrast|)**2)
we get 2 values of volume:
     with   1 - 4 * k >= 0
     volume1 = (1- sqrt(1- 4*k))/2
     volume2 = (1+ sqrt(1- 4*k))/2

q_star: the invariant value included extrapolation is applied
             unit  1/A^(3)*1/cm
        q_star = self.get_qstar()
```

(continues on next page)

(continued from previous page)

```
the result returned will be 0 <= volume <= 1
```

Parameters

- **contrast** – contrast value provides by the user of type float. contrast unit is 1/A^(2)= 10^(16)cm^(2)
- **extrapolation** – string to apply optional extrapolation

Returns volume fraction

Note volume fraction must have no unit

get_volume_fraction_with_error(*contrast*, *extrapolation=None*)

Compute uncertainty on volume value as well as the volume fraction This uncertainty is given by the following equation:

```
dV = 0.5 * (4*k* dq_star) /(2* math.sqrt(1-k* q_star))

for k = 10^(-8)*q_star/(2*(pi*|contrast|)**2)

q_star: the invariant value including extrapolated value if existing
dq_star: the invariant uncertainty
dV: the volume uncertainty
```

The uncertainty will be set to -1 if it can't be computed.

Parameters

- **contrast** – contrast value
- **extrapolation** – string to apply optional extrapolation

Returns V, dV = volume fraction, error on volume fraction

set_extrapolation(*range*, *npts=4*, *function=None*, *power=None*)

Set the extrapolation parameters for the high or low Q-range. Note that this does not turn extrapolation on or off.

Parameters

- **range** – a keyword set the type of extrapolation . type string
- **npts** – the numbers of q points of data to consider for extrapolation
- **function** – a keyword to select the function to use for extrapolation. of type string.
- **power** – an power to apply power_low function

class sas.sascalc.invariant.invariant.**PowerLaw**(*scale=1*, *power=4*)

Bases: *sas.sascalc.invariant.invariant.Transform*

class of type transform that perform operation related to power_law function

evaluate_model(*x*)

given a scale and a radius transform x, y using a power_law function

evaluate_model_errors(*x*)

Returns the error on I(q) for the given array of q-values :param x: array of q-values

extract_model_parameters(*constant*, *slope*, *dconstant=0*, *dslope=0*)

Assign new value to the scale and the power

linearize_q_value(*value*)

Transform the input q-value for linearization

Parameters **value** – q-value

Returns log(q)

class `sas.sascalc.invariant.invariant.`**`Transform`**

Bases: `object`

Define interface that need to compute a function or an inverse function given some x, y

`evaluate_model`(*x*)
Returns an array f(x) values where f is the Transform function.

`evaluate_model_errors`(*x*)
Returns an array of I(q) errors

`extract_model_parameters`(*constant*, *slope*, *dconstant=0*, *dslope=0*)
set private member

`get_allowed_bins`(*data*)
Goes through the data points and returns a list of boolean values to indicate whether each points is allowed by the model or not.

Parameters **`data`** – Data1D object

`linearize_data`(*data*)
Linearize data so that a linear fit can be performed. Filter out the data that can't be transformed.

Parameters **`data`** – LoadData1D instance

`linearize_q_value`(*value*)
Transform the input q-value for linearization

sas.sascalc.invariant.invariant_mapper module

This module is a wrapper to a map function. It allows to loop through different invariant objects to call the same function

`sas.sascalc.invariant.invariant_mapper.`**`get_qstar`**(*inv*, *extrapolation=None*)
Get invariant value (Q*)

`sas.sascalc.invariant.invariant_mapper.`**`get_qstar_with_error`**(*inv*, *extrapolation=None*)
Get invariant value with uncertainty

`sas.sascalc.invariant.invariant_mapper.`**`get_surface`**(*inv*, *contrast*, *porod_const*, *extrapolation=None*)
Get surface with uncertainty

`sas.sascalc.invariant.invariant_mapper.`**`get_surface_with_error`**(*inv*, *contrast*, *porod_const*, *extrapolation=None*)
Get surface with uncertainty

`sas.sascalc.invariant.invariant_mapper.`**`get_volume_fraction`**(*inv*, *contrast*, *extrapolation=None*)
Get volume fraction

`sas.sascalc.invariant.invariant_mapper.`**`get_volume_fraction_with_error`**(*inv*, *contrast*, *extrapolation=None*)
Get volume fraction with uncertainty

Module contents

sas.sascalc.pr package

Submodules

sas.sascalc.pr.distance_explorer module

Module to explore the P(r) inversion results for a range of D_max value. User picks a number of points and a range of distances, then get a series of outputs as a function of D_max over that range.

class sas.sascalc.pr.distance_explorer.**DistExplorer**(*pr_state*)

Bases: `object`

The explorer class

class sas.sascalc.pr.distance_explorer.**Results**

Bases: `object`

Class to hold the inversion output parameters as a function of D_max

sas.sascalc.pr.invertor module

Module to perform P(r) inversion. The module contains the Invertor class.

FIXME: The way the Invertor interacts with its C component should be cleaned up

class sas.sascalc.pr.invertor.**Invertor**

Bases: `Cinvertor`

Invertor class to perform P(r) inversion

The problem is solved by posing the problem as Ax = b, where x is the set of coefficients we are looking for.

Npts is the number of points.

In the following i refers to the ith base function coefficient. The matrix has its entries j in its first Npts rows set to

```
A[j][i] = (Fourier transformed base function for point j)
```

We then choose a number of r-points, n_r, to evaluate the second derivative of P(r) at. This is used as our regularization term. For a vector r of length n_r, the following n_r rows are set to

```
A[j+Npts][i] = (2nd derivative of P(r), d**2(P(r))/d(r)**2,
evaluated at r[j])
```

The vector b has its first Npts entries set to

```
b[j] = (I(q) observed for point j)
```

The following n_r entries are set to zero.

The result is found by using scipy.linalg.basic.lstsq to invert the matrix and find the coefficients x.

Methods inherited from Cinvertor:

- `get_peaks(pars)`: returns the number of P(r) peaks
- `oscillations(pars)`: returns the oscillation parameters for the output P(r)
- `get_positive(pars)`: returns the fraction of P(r) that is above zero
- `get_pos_err(pars)`: returns the fraction of P(r) that is 1-sigma above zero

background = 0

chi2 = 0

clone()
Return a clone of this instance

cov = None

elapsed = 0

estimate_alpha(*nfunc*)
Returns a reasonable guess for the regularization constant alpha

Parameters **nfunc** – number of terms to use in the expansion.

Returns alpha, message, elapsed

where alpha is the estimate for alpha, message is a message for the user, elapsed is the computation time

estimate_numterms(*isquit_func=None*)
Returns a reasonable guess for the number of terms

Parameters **isquit_func** – reference to thread function to call to check whether the computation needs to be stopped.

Returns number of terms, alpha, message

from_file(*path*)
Load the state of the Invertor from a file, to be able to generate P(r) from a set of parameters.

Parameters **path** – path of the file to load

info = {}

invert(*nfunc=10, nr=20*)
Perform inversion to P(r)

The problem is solved by posing the problem as Ax = b, where x is the set of coefficients we are looking for.

Npts is the number of points.

In the following i refers to the ith base function coefficient. The matrix has its entries j in its first Npts rows set to

```
A[i][j] = (Fourier transformed base function for point j)
```

We then choose a number of r-points, n_r, to evaluate the second derivative of P(r) at. This is used as our regularization term. For a vector r of length n_r, the following n_r rows are set to

```
A[i+Npts][j] = (2nd derivative of P(r), d**2(P(r))/d(r)**2, evaluated at␣
↪r[j])
```

The vector b has its first Npts entries set to

```
b[j] = (I(q) observed for point j)
```

The following n_r entries are set to zero.

The result is found by using scipy.linalg.basic.lstsq to invert the matrix and find the coefficients x.

Parameters

- **nfunc** – number of base functions to use.
- **nr** – number of r points to evaluate the 2nd derivative at for the reg. term.

Returns c_out, c_cov - the coefficients with covariance matrix

invert_optimize(*nfunc=10*, *nr=20*)

Slower version of the P(r) inversion that uses scipy.optimize.leastsq.

This probably produce more reliable results, but is much slower. The minimization function is set to sum_i[(I_obs(q_i) - I_theo(q_i))/err**2] + alpha * reg_term, where the reg_term is given by Svergun: it is the integral of the square of the first derivative of P(r), d(P(r))/dr, integrated over the full range of r.

Parameters

- **nfunc** – number of base functions to use.
- **nr** – number of r points to evaluate the 2nd derivative at for the reg. term.

Returns c_out, c_cov - the coefficients with covariance matrix

iq(*out*, *q*)

Function to call to evaluate the scattering intensity

Parameters **args** – c-parameters, and q

Returns I(q)

lstsq(*nfunc=5*, *nr=20*)

The problem is solved by posing the problem as Ax = b, where x is the set of coefficients we are looking for.

Npts is the number of points.

In the following i refers to the ith base function coefficient. The matrix has its entries j in its first Npts rows set to

```
A[i][j] = (Fourier transformed base function for point j)
```

We then choose a number of r-points, n_r, to evaluate the second derivative of P(r) at. This is used as our regularization term. For a vector r of length n_r, the following n_r rows are set to

```
A[i+Npts][j] = (2nd derivative of P(r), d**2(P(r))/d(r)**2,
evaluated at r[j])
```

The vector b has its first Npts entries set to

```
b[j] = (I(q) observed for point j)
```

The following n_r entries are set to zero.

The result is found by using scipy.linalg.basic.lstsq to invert the matrix and find the coefficients x.

Parameters

- **nfunc** – number of base functions to use.
- **nr** – number of r points to evaluate the 2nd derivative at for the reg. term.

If the result does not allow us to compute the covariance matrix, a matrix filled with zeros will be returned.

nfunc = 10

out = None

pr_err(*c*, *c_cov*, *r*)

Returns the value of P(r) for a given r, and base function coefficients, with error.

Parameters

- **c** – base function coefficients
- **c_cov** – covariance matrice of the base function coefficients
- **r** – r-value to evaluate P(r) at

Returns P(r)

`pr_fit` (*nfunc=5*)
This is a direct fit to a given P(r). It assumes that the y data is set to some P(r) distribution that we are trying to reproduce with a set of base functions.

This method is provided as a test.

`suggested_alpha = 0`

`to_file` (*path*, *npts=100*)
Save the state to a file that will be readable by SliceView.

Parameters

- **`path`** – path of the file to write
- **`npts`** – number of P(r) points to be written

`sas.sascalc.pr.invertor.`**`help`**()
Provide general online help text Future work: extend this function to allow topic selection

sas.sascalc.pr.num_term module

`class` `sas.sascalc.pr.num_term.`**`NTermEstimator`**(*invertor*)
Bases: `object`

`compare_err`()

`get0_out`()

`is_odd`(*n*)

`ls_osc`()

`median_osc`()

`num_terms`(*isquit_func=None*)

`sort_osc`()

`sas.sascalc.pr.num_term.`**`load`**(*path*)

Module contents

P(r) inversion for SAS

Module contents

sas.sasgui package

Subpackages

sas.sasgui.guiframe package

Subpackages

sas.sasgui.guiframe.local_perspectives package

Subpackages

sas.sasgui.guiframe.local_perspectives.data_loader package

Submodules

sas.sasgui.guiframe.local_perspectives.data_loader.data_loader module

plugin DataLoader responsible of loading data

class `sas.sasgui.guiframe.local_perspectives.data_loader.data_loader.`**`Plugin`**

Bases: *`sas.sasgui.guiframe.plugin_base.PluginBase`*

`can_load_data`()
: if return True, then call handler to load data

`get_data`(*path*, *format=None*)

`get_file_path`(*path*)
: Receive a list containing folder then return a list of file

`load_complete`(*output*, *message=''*, *info='warning'*)
: post message to status bar and return list of data

`load_data`(*event*)
: Load data

`load_error`(*error=None*)
: Pop up an error message.

: **Parameters** **`error`** – details error message to be displayed

`load_update`(*message=''*, *info='warning'*)
: print update on the status bar

`populate_file_menu`()
: get a menu item and append it under file menu of the application add load file menu item and load folder item

sas.sasgui.guiframe.local_perspectives.data_loader.load_thread module

Loading thread

class sas.sasgui.guiframe.local_perspectives.data_loader.load_thread.**DataReader**(*path, loader, flag=True, transform_data= completefn=No updatefn=Nor yieldtime=0.01, worktime=0.01*)

Bases: *sas.sascalc.data_util.calcthread.CalcThread*

Load a data given a filename

compute()
: read some data

isquit()
: **Raises KeyboardInterrupt** – when the thread is interrupted

Module contents

sas.sasgui.guiframe.local_perspectives.plotting package

Submodules

sas.sasgui.guiframe.local_perspectives.plotting.AnnulusSlicer module

class sas.sasgui.guiframe.local_perspectives.plotting.AnnulusSlicer.**AnnulusInteractor**(*ba ax co zo*

Bases: sas.sasgui.guiframe.local_perspectives.plotting.BaseInteractor._BaseInteractor

Select an annulus through a 2D plot. This interactor is used to average 2D data with the region defined by 2 radius. this class is defined by 2 Ringinterators.

clear()
: Clear the slicer and all connected events related to this slicer

draw()

freeze_axes()

get_params()
: Store a copy of values of parameters of the slicer into a dictionary.

  Return params the dictionary created

move(*x, y, ev*)
: Process move to a new position, making sure that the move is allowed.

moveend(*ev*)
: Called when any dragging motion ends. Post an event (type =SlicerParameterEvent) to plotter 2D with a copy slicer parameters Call _post_data method

restore()
: Restore the roughness for this layer.

save (*ev*)
Remember the roughness for this layer and the next so that we can restore on Esc.

set_cursor (*x*, *y*)

set_layer (*n*)
Allow adding plot to the same panel

Parameters **n** – the number of layer

set_params (*params*)
Receive a dictionary and reset the slicer with values contained in the values of the dictionary.

Parameters **params** – a dictionary containing name of slicer parameters and values the user assigned to the slicer.

thaw_axes ()

update ()
Respond to changes in the model by recalculating the profiles and resetting the widgets.

class `sas.sasgui.guiframe.local_perspectives.plotting.AnnulusSlicer.`**CircularMask** (*base*, *axes*, *color='gr*, *zorder=3*, *side=Non*

Bases: `sas.sasgui.guiframe.local_perspectives.plotting.BaseInteractor._BaseInteractor`

Draw a ring Given a radius

clear ()
Clear the slicer and all connected events related to this slicer

draw ()

freeze_axes ()

get_params ()
Store a copy of values of parameters of the slicer into a dictionary.

Return params the dictionary created

move (*x*, *y*, *ev*)
Process move to a new position, making sure that the move is allowed.

moveend (*ev*)
Called when any dragging motion ends. Post an event (type =SlicerParameterEvent) to plotter 2D with a copy slicer parameters Call _post_data method

restore ()
Restore the roughness for this layer.

save (*ev*)
Remember the roughness for this layer and the next so that we can restore on Esc.

set_cursor (*x*, *y*)

set_layer (*n*)
Allow adding plot to the same panel

Parameters **n** – the number of layer

set_params (*params*)
Receive a dictionary and reset the slicer with values contained in the values of the dictionary.

Parameters **params** – a dictionary containing name of slicer parameters and values the user assigned to the slicer.

thaw_axes ()

update()
Respond to changes in the model by recalculating the profiles and resetting the widgets.

class sas.sasgui.guiframe.local_perspectives.plotting.AnnulusSlicer.**RingInteractor**(*base, axes, color=*, *zorder*, *r=1.0, sign=1*

Bases: sas.sasgui.guiframe.local_perspectives.plotting.BaseInteractor._BaseInteractor

Draw a ring Given a radius

clear()
Clear the slicer and all connected events related to this slicer

get_params()
Store a copy of values of parameters of the slicer into a dictionary.

Return params the dictionary created

get_radius()

Return self._inner_mouse_x the current radius of the ring

move(*x, y, ev*)
Process move to a new position, making sure that the move is allowed.

moveend(*ev*)
Called after a dragging motion

restore()
Restore the roughness for this layer.

save(*ev*)
Remember the roughness for this layer and the next so that we can restore on Esc.

set_cursor(*x, y*)
draw the ring given x, y value

set_layer(*n*)
Allow adding plot to the same panel

Parameters n – the number of layer

set_params(*params*)
Receive a dictionary and reset the slicer with values contained in the values of the dictionary.

Parameters params – a dictionary containing name of slicer parameters and values the user assigned to the slicer.

update()
Draw the new roughness on the graph.

sas.sasgui.guiframe.local_perspectives.plotting.Arc module

Arc slicer for 2D data

class sas.sasgui.guiframe.local_perspectives.plotting.Arc.**ArcInteractor**(*base, axes, color='black', zorder=5, r=1.0, theta1=0.39269908169*, *theta2=0.78539816339*

Bases: `sas.sasgui.guiframe.local_perspectives.plotting.BaseInteractor._BaseInteractor`

Select an annulus through a 2D plot

clear()
Clear this slicer and its markers

get_params()

get_radius()
Return arc radius

move(*x*, *y*, *ev*)
Process move to a new position, making sure that the move is allowed.

moveend(*ev*)
After a dragging motion reset the flag self.has_move to False :param ev: event

restore()
Restore the roughness for this layer.

save(*ev*)
Remember the roughness for this layer and the next so that we can restore on Esc.

set_cursor(*radius*, *phi_min*, *phi_max*, *nbins*)

set_layer(*n*)
Allow adding plot to the same panel :param n: the number of layer

set_params(*params*)

update(*theta1=None*, *theta2=None*, *nbins=None*, *r=None*)
Update the plotted arc :param theta1: starting angle of the arc :param theta2: ending angle of the arc :param nbins: number of points along the arc :param r: radius of the arc

sas.sasgui.guiframe.local_perspectives.plotting.AzimutSlicer module

class `sas.sasgui.guiframe.local_perspectives.plotting.AzimutSlicer.`**SectorInteractor**(*base*, *axes*, *color* *zorde*

Bases: `sas.sasgui.guiframe.local_perspectives.plotting.BaseInteractor._BaseInteractor`

Select an annulus through a 2D plot

clear()

draw()

freeze_axes()

get_params()

move(*x*, *y*, *ev*)
Process move to a new position, making sure that the move is allowed.

moveend(*ev*)

post_data(*new_sector*)
post data averaging in Q

restore()
Restore the roughness for this layer.

save(*ev*)
Remember the roughness for this layer and the next so that we can restore on Esc.

set_cursor(*x*, *y*)

set_layer(*n*)

set_params(*params*)

thaw_axes()

update()

Respond to changes in the model by recalculating the profiles and resetting the widgets.

class sas.sasgui.guiframe.local_perspectives.plotting.AzimutSlicer.**SectorInteractorPhi**(*b*
a
c
z

Bases: *sas.sasgui.guiframe.local_perspectives.plotting.AzimutSlicer.SectorInteractor*

class sas.sasgui.guiframe.local_perspectives.plotting.AzimutSlicer.**SectorInteractorQ**(*base*
axes
colo
zord

Bases: *sas.sasgui.guiframe.local_perspectives.plotting.AzimutSlicer.SectorInteractor*

sas.sasgui.guiframe.local_perspectives.plotting.BaseInteractor module

sas.sasgui.guiframe.local_perspectives.plotting.Edge module

class sas.sasgui.guiframe.local_perspectives.plotting.Edge.**RadiusInteractor**(*base*, *axes*, *color='black'*, *zorder=5*, *arc1=None*, *arc2=None*, *theta=0.3926990*

Bases: sas.sasgui.guiframe.local_perspectives.plotting.BaseInteractor._BaseInteractor

Select an annulus through a 2D plot

clear()

get_angle()

get_params()

move(*x*, *y*, *ev*)

Process move to a new position, making sure that the move is allowed.

moveend(*ev*)

restore(*ev*)

Restore the roughness for this layer.

save(*ev*)

Remember the roughness for this layer and the next so that we can restore on Esc.

set_cursor(*r_min*, *r_max*, *theta*)

set_layer(*n*)

set_params(*params*)

update(*r1=None*, *r2=None*, *theta=None*)

Draw the new roughness on the graph.

sas.sasgui.guiframe.local_perspectives.plotting.Plotter1D module

class sas.sasgui.guiframe.local_perspectives.plotting.Plotter1D.**ModelPanel1D**(*parent*, *id=-1*, *color=None*, *dpi=None*, *style=0*, ***kwargs*)

Bases: `sas.sasgui.plottools.PlotPanel.PlotPanel`, `sas.sasgui.guiframe.panel_base.PanelBase`

Plot panel for use with the GUI manager

ALWAYS_ON = True

createAppDialog(*event*)
: Create the custom dialog for fit appearance modification

cusor_line(*event*)
: Move the cursor line to write Q range

draw_plot()
: Draw plot

get_color_label()
: Associates label to a specific color

get_data_xy_vals(*xval*)
: Get x, y data values near x = x_val

get_symbol_label()
: Associates label to symbol

group_id = None

modifyGraphAppearance(*event*)
: On Modify Graph Appearance

onContextMenu(*event*)
: 1D plot context menu

Parameters **event** – wx context event

onFreeze(*event*)
: on Freeze data

onLeftDown(*event*)
: left button down and ready to drag Display the position of the mouse on the statusbar

onSetRange(*event*)

on_AppDialog_close(*event*)
: on_Modify Plot Property_close

on_close(*event*)
: On Close Event

on_graphApp_close(*event*)
: Gets values from graph appearance dialog and sends them off to modify the plot

on_plot_qrange(*event=None*)
: On Qmin Qmax vertical line event

plot_data(*data*)
: Data is ready to be displayed

Parameters **event** – data event

remove_data_by_id(*id*)
Remove data from plot

schedule_full_draw(*func='append'*)
Put self in schedule to full redraw list

set_data(*list=None*)

set_resizing(*resizing=False*)
Set the resizing (True/False)

window_caption = 'Graph'

window_name = 'plotpanel'

sas.sasgui.guiframe.local_perspectives.plotting.Plotter1D.**find_key**(*dic*, *val*)
return the key of dictionary dic given the value

sas.sasgui.guiframe.local_perspectives.plotting.Plotter2D module

class sas.sasgui.guiframe.local_perspectives.plotting.Plotter2D.**ModelPanel2D**(*parent*, *id=-1*, *data2d=None*, *color=None*, *dpi=None*, *style=0*, ***kwargs*)

Bases: *sas.sasgui.guiframe.local_perspectives.plotting.Plotter1D.ModelPanel1D*

Plot panel for use with the GUI manager

ALWAYS_ON = True

add_toolbar()
add toolbar

freeze_axes()

group_id = None

modifyGraphAppearance(*e*)
On Modify Graph Appearance

onBoxSum(*event*)

onBoxavgX(*event*)
Perform 2D data averaging on Qx Create a new slicer .

Parameters **event** – wx.menu event

onBoxavgY(*event*)
Perform 2D data averaging on Qy Create a new slicer .

Parameters **event** – wx.menu event

onCircular(*event*, *ismask=False*)
perform circular averaging on Data2D

Parameters **event** – wx.menu event

onClearSlicer(*event*)
Clear the slicer on the plot

onContextMenu(*event*)
2D plot context menu

Parameters **event** – wx context event

onEditLabels(*event*)
Edit legend label

onLeftDown(*event*)
left button down and ready to drag

onMaskedCircular(*event*)
perform circular averaging on Data2D with mask if it exists

Parameters **event** – wx.menu event

onMouseMotion(*event*)

onSectorPhi(*event*)
Perform sector averaging on Phi and draw annulus slicer

onSectorQ(*event*)
Perform sector averaging on Q and draw sector slicer

onWheel(*event*)

on_graphApp_close(*e*)
Gets values from graph appearance dialog and sends them off to modify the plot

on_plot_qrange(*event=None*)
On Qmin Qmax vertical line event

plot_data(*data*)
Data is ready to be displayed

TODO this name should be changed to something more appropriate Don't forget that changing this name will mean changing code in plotting.py

Parameters **event** – data event

thaw_axes()

update(*draw=True*)
Respond to changes in the model by recalculating the profiles and resetting the widgets.

window_caption = 'Plot Panel'

window_name = 'plotpanel'

class sas.sasgui.guiframe.local_perspectives.plotting.Plotter2D.**NavigationToolBar2D**(*canva par- ent=*

Bases: *sas.sasgui.plottools.toolbar.NavigationToolBar*

add_option()
add item to the toolbar

delete_option()
remove default toolbar item

sas.sasgui.guiframe.local_perspectives.plotting.Plotter2D.**find_key**(*dic, val*)
return the key of dictionary dic given the value

sas.sasgui.guiframe.local_perspectives.plotting.SectorSlicer module

Sector interactor

class sas.sasgui.guiframe.local_perspectives.plotting.SectorSlicer.**LineInteractor**(*base, axes, color='l zorder=. r=1.0, theta=0.*

Bases: sas.sasgui.guiframe.local_perspectives.plotting.BaseInteractor._BaseInteractor

Select an annulus through a 2D plot

clear()

get_params()

move(*x, y, ev*)
Process move to a new position, making sure that the move is allowed.

moveend(*ev*)

restore()
Restore the roughness for this layer.

save(*ev*)
Remember the roughness for this layer and the next so that we can restore on Esc.

set_cursor(*x, y*)

set_layer(*n*)

set_params(*params*)

update(*theta=None*)
Draw the new roughness on the graph.

class sas.sasgui.guiframe.local_perspectives.plotting.SectorSlicer.**SectorInteractor**(*base, axes, color zorde*

Bases: sas.sasgui.guiframe.local_perspectives.plotting.BaseInteractor._BaseInteractor

Draw a sector slicer.Allow to performQ averaging on data 2D

clear()
Clear the slicer and all connected events related to this slicer

draw()

freeze_axes()

get_params()
Store a copy of values of parameters of the slicer into a dictionary.

Return params the dictionary created

move(*x, y, ev*)
Process move to a new position, making sure that the move is allowed.

moveend(*ev*)
Called a dragging motion ends.Get slicer event

restore()
Restore the roughness for this layer.

save(*ev*)
Remember the roughness for this layer and the next so that we can restore on Esc.

set_cursor(*x, y*)

set_layer(*n*)

Allow adding plot to the same panel

Parameters **n** – the number of layer

set_params (*params*)
Receive a dictionary and reset the slicer with values contained in the values of the dictionary.

Parameters **params** – a dictionary containing name of slicer parameters and values the user assigned to the slicer.

thaw_axes ()

update ()
Respond to changes in the model by recalculating the profiles and resetting the widgets.

class sas.sasgui.guiframe.local_perspectives.plotting.SectorSlicer.**SideInteractor** (*base*, *axes*, *color='b*, *zorder=*, *r=1.0*, *phi=0.7*, *theta2=*

Bases: sas.sasgui.guiframe.local_perspectives.plotting.BaseInteractor._BaseInteractor

Draw an oblique line

Parameters

- **phi** – the phase between the middle line and one side line
- **theta2** – the angle between the middle line and x- axis

clear ()
Clear the slicer and all connected events related to this slicer

get_params ()

move (*x*, *y*, *ev*)
Process move to a new position, making sure that the move is allowed.

moveend (*ev*)

restore ()
Restore the roughness for this layer.

save (*ev*)
Remember the roughness for this layer and the next so that we can restore on Esc.

set_cursor (*x*, *y*)

set_layer (*n*)
Allow adding plot to the same panel

Parameters **n** – the number of layer

set_params (*params*)

update (*phi=None*, *delta=None*, *mline=None*, *side=False*, *left=False*, *right=False*)
Draw oblique line

Parameters

- **phi** – the phase between the middle line and the current line
- **delta** – phi/2 applied only when the mline was moved

sas.sasgui.guiframe.local_perspectives.plotting.SimplePlot module

Simple Plot Frame : supporting only copy, print, scale

class sas.sasgui.guiframe.local_perspectives.plotting.SimplePlot.**PlotFrame**(*parent, id, title, scale='log_{10}', size=wx.Size(550, 470), show_menu_icons=*

Bases: wx._windows.Frame

Frame for simple plot

add_plot(*plot*)
Add Image

disable_app_menu(*panel*)

get_current_context_menu(*plotpanel*)

im_show(*img*)
Show background image :Param img: [imread(path) from matplotlib.pyplot]

on_close(*event*)
On Close

on_copy_image(*event*)
Save image

on_print_image(*event*)
Save image

on_print_preview(*event*)
Save image

on_save_file(*event*)
Save image

set_plot_unfocus()
un focusing

set_schedule(*schedule=False*)

set_schedule_full_draw(*panel, func*)

class sas.sasgui.guiframe.local_perspectives.plotting.SimplePlot.**SimplePlotPanel**(*parent, id=-1, color=No dpi=None style=0, **kwargs*

Bases: sas.sasgui.guiframe.local_perspectives.plotting.Plotter2D.ModelPanel2D

PlotPanel for 1d and 2d

add_toolbar()

draw()

onContextMenu(*event*)
2D plot context menu

Parameters event – wx context event

onLeftDown (*event*)
: left button down and ready to drag

on_grid_onoff (*event*)
: On grid on/off

on_kill_focus (*event*)
: Reset the panel color

on_set_focus (*event*)
: By pass default boundary blue color drawing

show_plot (*plot*)
: Show the plot

sas.sasgui.guiframe.local_perspectives.plotting.appearanceDialog module

Dialog for appearance of plot symbols, color, size etc.

This software was developed by Institut Laue-Langevin as part of Distributed Data Analysis of Neutron Scattering Experiments (DANSE).

Copyright 2012 Institut Laue-Langevin

class `sas.sasgui.guiframe.local_perspectives.plotting.appearanceDialog.`**appearanceDialog**

Bases: `wx._windows.Frame`

Appearance dialog

close_dlg (*event*)
: On Close Dlg

combo_click (*event*)
: Combox on click

custom_size (*event*)
: On custom size

static find_key (*dic, val*)
: Find key

get_current_values ()
: Get Current Values :returns : (size, color, symbol, dataname)

init_ui ()
: Create spacing needed

on_ok (*event*)
: On OK button clicked

populate_color ()
: Populate Colors

populate_size ()
: Populate Size

populate_symbol ()
: Populate Symbols

set_defaults (*size, color, symbol, label*)
: Set Defaults

sas.sasgui.guiframe.local_perspectives.plotting.binder module

Extension to MPL to support the binding of artists to key/mouse events.

class `sas.sasgui.guiframe.local_perspectives.plotting.binder.`**`BindArtist`**(*figure*)

Bases: `object`

Track keyboard modifiers for events. TODO: Move keyboard modifier support into the backend. We cannot TODO: properly support it from outside the windowing system since there TODO: is no way to recognized whether shift is held down when the mouse TODO: first clicks on the the application window.

`alt = False`

`clear`(*h1*, *h2*, *...*)

Remove connections for artists h1, h2, . . .

Use clearall() to reset all connections.

`clearall`()

Clear connections to all artists.

Use clear(h1,h2,. . .) to reset specific artists.

`control = False`

`dclick_threshhold = 0.25`

`disconnect`()

In case we need to disconnect from the canvas. . .

`events = ['enter', 'leave', 'motion', 'click', 'dclick', 'drag', 'release', 'scroll`

`meta = False`

`shift = False`

`trigger`(*actor*, *action*, *ev*)

Trigger a particular event for the artist. Fallback to axes, to figure, and to 'all' if the event is not processed.

class `sas.sasgui.guiframe.local_perspectives.plotting.binder.`**`Selection`**(*artist=None*, *prop={}*)

Bases: `object`

Store and compare selections.

`artist = None`

`prop = {}`

sas.sasgui.guiframe.local_perspectives.plotting.boxMask module

class `sas.sasgui.guiframe.local_perspectives.plotting.boxMask.`**`BoxMask`**(*base*, *axes*, *color='black'*, *zorder=3*, *side=None*, *x_min=0.008*, *x_max=0.008*, *y_min=0.0025*, *y_max=0.0025*)

Bases: `sas.sasgui.guiframe.local_perspectives.plotting.BaseInteractor._BaseInteractor`

BoxMask Class: determine 2 rectangular area to find the pixel of a Data inside of box.

Uses PointerInteractor , VerticalDoubleLine,HorizontalDoubleLine.

Parameters

- **`zorder`** – Artists with lower zorder values are drawn first.
- **`x_min`** – the minimum value of the x coordinate
- **`x_max`** – the maximum value of the x coordinate
- **`y_min`** – the minimum value of the y coordinate
- **`y_max`** – the maximum value of the y coordinate

`clear`()
Clear the slicer and all connected events related to this slicer

`draw`()

`freeze_axes`()

`get_mask`()
return mask as a result of boxcut

`get_params`()
Store a copy of values of parameters of the slicer into a dictionary.

Return params the dictionary created

`move`(*x*, *y*, *ev*)
Process move to a new position, making sure that the move is allowed.

`moveend`(*ev*)
After a dragging motion this function is called to compute the error and the sum of pixel of a given data 2D

`restore`()
Restore the roughness for this layer.

`save`(*ev*)
Remember the roughness for this layer and the next so that we can restore on Esc.

`set_cursor`(*x*, *y*)

`set_params`(*params*)
Receive a dictionary and reset the slicer with values contained in the values of the dictionary.

Parameters `params` – a dictionary containing name of slicer parameters and values the user assigned to the slicer.

`thaw_axes`()

`update`()
Respond to changes in the model by recalculating the profiles and resetting the widgets.

class `sas.sasgui.guiframe.local_perspectives.plotting.boxMask.`**`inner_BoxMask`**(*base*, *axes*, *color='black'*, *zorder=3*, *side=None*, *x_min=0.008*, *x_max=0.008*, *y_min=0.0025*, *y_max=0.0025*)

Bases: *`sas.sasgui.guiframe.local_perspectives.plotting.boxMask.BoxMask`*

sas.sasgui.guiframe.local_perspectives.plotting.boxSlicer module

class sas.sasgui.guiframe.local_perspectives.plotting.boxSlicer.**BoxInteractor**(*base*, *axes*, *color='black'*, *zorder=3*)

Bases: `sas.sasgui.guiframe.local_perspectives.plotting.BaseInteractor._BaseInteractor`

BoxInteractor define a rectangle that return data1D average of Data2D in a rectangle area defined by -x, x ,y, -y

clear()
Clear the slicer and all connected events related to this slicer

draw()

freeze_axes()

get_params()
Store a copy of values of parameters of the slicer into a dictionary.

Return params the dictionary created

move(*x*, *y*, *ev*)
Process move to a new position, making sure that the move is allowed.

moveend(*ev*)
Called after a dragging event. Post the slicer new parameters and creates a new Data1D corresponding to the new average

post_data(*new_slab=None*, *nbins=None*, *direction=None*)
post data averaging in Qx or Qy given new_slab type

Parameters

- **new_slab** – slicer that determine with direction to average
- **nbins** – the number of points plotted when averaging
- **direction** – the direction of averaging

restore()
Restore the roughness for this layer.

save(*ev*)
Remember the roughness for this layer and the next so that we can restore on Esc.

set_cursor(*x*, *y*)

set_layer(*n*)
Allow adding plot to the same panel

Parameters n – the number of layer

set_params(*params*)
Receive a dictionary and reset the slicer with values contained in the values of the dictionary.

Parameters params – a dictionary containing name of slicer parameters and values the user assigned to the slicer.

thaw_axes()

update()
Respond to changes in the model by recalculating the profiles and resetting the widgets.

update_and_post()
Update the slicer and plot the resulting data

class sas.sasgui.guiframe.local_perspectives.plotting.boxSlicer.**BoxInteractorX**(*base*, *axes*, *color='black*, *zorder=3*)

Bases: `sas.sasgui.guiframe.local_perspectives.plotting.boxSlicer.BoxInteractor`

Average in Qx direction

class sas.sasgui.guiframe.local_perspectives.plotting.boxSlicer.**BoxInteractorY**(*base*, *axes*, *color='black*, *zorder=3*)

Bases: `sas.sasgui.guiframe.local_perspectives.plotting.boxSlicer.BoxInteractor`

Average in Qy direction

class sas.sasgui.guiframe.local_perspectives.plotting.boxSlicer.**HorizontalLines**(*base*, *axes*, *color='bla*, *zorder=5*, *x=0.5*, *y=0.5*)

Bases: `sas.sasgui.guiframe.local_perspectives.plotting.BaseInteractor._BaseInteractor`

Draw 2 Horizontal lines centered on (0,0) that can move on the x- direction and in opposite direction

clear()
Clear this slicer and its markers

move(*x*, *y*, *ev*)
Process move to a new position, making sure that the move is allowed.

moveend(*ev*)
Called after a dragging this edge and set self.has_move to False to specify the end of dragging motion

restore()
Restore the roughness for this layer.

save(*ev*)
Remember the roughness for this layer and the next so that we can restore on Esc.

set_layer(*n*)
Allow adding plot to the same panel

Parameters n – the number of layer

update(*x=None*, *y=None*)
Draw the new roughness on the graph.

Parameters

- **x** – x-coordinates to reset current class x
- **y** – y-coordinates to reset current class y

class sas.sasgui.guiframe.local_perspectives.plotting.boxSlicer.**VerticalLines**(*base*, *axes*, *color='black'*, *zorder=5*, *x=0.5*, *y=0.5*)

Bases: `sas.sasgui.guiframe.local_perspectives.plotting.BaseInteractor._BaseInteractor`

Select an annulus through a 2D plot

clear()
Clear this slicer and its markers

move(*x*, *y*, *ev*)
Process move to a new position, making sure that the move is allowed.

moveend(*ev*)
Called after a dragging this edge and set self.has_move to False to specify the end of dragging motion

restore()
Restore the roughness for this layer.

save(*ev*)
Remember the roughness for this layer and the next so that we can restore on Esc.

set_layer(*n*)
Allow adding plot to the same panel

Parameters **n** – the number of layer

update(*x=None*, *y=None*)
Draw the new roughness on the graph.

Parameters

- **x** – x-coordinates to reset current class x
- **y** – y-coordinates to reset current class y

sas.sasgui.guiframe.local_perspectives.plotting.boxSum module

Boxsum Class: determine 2 rectangular area to compute the sum of pixel of a Data.

class sas.sasgui.guiframe.local_perspectives.plotting.boxSum.**BoxSum**(*base*, *axes*, *color='black'*, *zorder=3*, *x_min=0.008*, *x_max=0.008*, *y_min=0.0025*, *y_max=0.0025*)

Bases: `sas.sasgui.guiframe.local_perspectives.plotting.BaseInteractor._BaseInteractor`

Boxsum Class: determine 2 rectangular area to compute the sum of pixel of a Data. Uses PointerInteractor , VerticalDoubleLine,HorizontalDoubleLine. @param zorder: Artists with lower zorder values are drawn first. @param x_min: the minimum value of the x coordinate @param x_max: the maximum value of the x coordinate @param y_min: the minimum value of the y coordinate @param y_max: the maximum value of the y coordinate

clear()
Clear the slicer and all connected events related to this slicer

draw()

freeze_axes()

get_params()
Store a copy of values of parameters of the slicer into a dictionary. :return params: the dictionary created

get_result()
return the result of box summation

move(*x*, *y*, *ev*)
Process move to a new position, making sure that the move is allowed.

moveend(*ev*)
After a dragging motion this function is called to compute the error and the sum of pixel of a given data 2D

restore()
Restore the roughness for this layer.

save(*ev*)
Remember the roughness for this layer and the next so that we can restore on Esc.

set_cursor(*x*, *y*)

set_layer(*n*)
Allow adding plot to the same panel :param n: the number of layer

set_panel_name(*name*)
Store the name of the panel associated to this slicer @param name: the name of this panel

set_params(*params*)
Receive a dictionary and reset the slicer with values contained in the values of the dictionary. :param params: a dictionary containing name of slicer parameters and values the user assigned to the slicer.

thaw_axes()

update()
Respond to changes in the model by recalculating the profiles and resetting the widgets.

class sas.sasgui.guiframe.local_perspectives.plotting.boxSum.**HorizontalDoubleLine**(*base*, *axes*, *color='l*, *zorder=*, *x=0.5*, *y=0.5*, *cen-ter_x=0*, *cen-ter_y=0*

Bases: `sas.sasgui.guiframe.local_perspectives.plotting.BaseInteractor._BaseInteractor`

Select an annulus through a 2D plot

clear()
Clear this figure and its markers

move(*x*, *y*, *ev*)
Process move to a new position, making sure that the move is allowed.

moveend(*ev*)
After a dragging motion reset the flag self.has_move to False

restore()
Restore the roughness for this layer.

save(*ev*)
Remember the roughness for this layer and the next so that we can restore on Esc.

set_cursor(*x*, *y*)
Update the figure given x and y

set_layer(*n*)
Allow adding plot to the same panel @param n: the number of layer

update(*x1=None*, *x2=None*, *y1=None*, *y2=None*, *width=None*, *height=None*, *center=None*)
Draw the new roughness on the graph. :param x1: new maximum value of x coordinates :param x2: new minimum value of x coordinates :param y1: new maximum value of y coordinates :param y2:

new minimum value of y coordinates :param width: is the width of the new rectangle :param height: is the height of the new rectangle :param center: provided x, y coordinates of the center point

class sas.sasgui.guiframe.local_perspectives.plotting.boxSum.**PointInteractor**(*base, axes, color='black', zorder=5, center_x=0.0, center_y=0.0*)

Bases: sas.sasgui.guiframe.local_perspectives.plotting.BaseInteractor._BaseInteractor

Draw a point that can be dragged with the marker. this class controls the motion the center of the BoxSum

clear()
Clear this figure and its markers

move(*x, y, ev*)
Process move to a new position, making sure that the move is allowed.

moveend(*ev*)

restore()
Restore the roughness for this layer.

save(*ev*)
Remember the roughness for this layer and the next so that we can restore on Esc.

set_cursor(*x, y*)

set_layer(*n*)
Allow adding plot to the same panel @param n: the number of layer

update(*center_x=None, center_y=None*)
Draw the new roughness on the graph.

class sas.sasgui.guiframe.local_perspectives.plotting.boxSum.**VerticalDoubleLine**(*base, axes, color='bla zorder=5, x=0.5, y=0.5, center_x=0.0, center_y=0.0*)

Bases: sas.sasgui.guiframe.local_perspectives.plotting.BaseInteractor._BaseInteractor

Draw 2 vertical lines moving in opposite direction and centered on a point (PointInteractor)

clear()
Clear this slicer and its markers

move(*x, y, ev*)
Process move to a new position, making sure that the move is allowed.

moveend(*ev*)
After a dragging motion reset the flag self.has_move to False

restore()
Restore the roughness for this layer.

save(*ev*)
Remember the roughness for this layer and the next so that we can restore on Esc.

set_cursor (*x*, *y*)
Update the figure given x and y

set_layer (*n*)
Allow adding plot to the same panel :param n: the number of layer

update (*x1=None*, *x2=None*, *y1=None*, *y2=None*, *width=None*, *height=None*, *center=None*)
Draw the new roughness on the graph. :param x1: new maximum value of x coordinates :param x2: new minimum value of x coordinates :param y1: new maximum value of y coordinates :param y2: new minimum value of y coordinates :param width: is the width of the new rectangle :param height: is the height of the new rectangle :param center: provided x, y coordinates of the center point

sas.sasgui.guiframe.local_perspectives.plotting.detector_dialog module

Widget to display a 2D map of the detector

class `sas.sasgui.guiframe.local_perspectives.plotting.detector_dialog.`**DetectorDialog** (*pare id= bas dpi= cma ob- ject re- set_ re- set_ *arg **k*

Bases: `wx._windows.Dialog`

Dialog box to let the user edit detector settings

class Event
Bases: `object`

beam = 0

cmap = None

qpax = 0

sym4 = False

xnpts = 0

ynpts = 0

zmax = 0

zmin = 0

checkValues (*event*)
Check the valitidity of zmin and zmax value zmax should be a float and zmin less than zmax

getContent ()
return event containing value to reset the detector of a given data

onSetFocus (*event*)
Highlight the txtcrtl

resetValues (*event*)
reset detector info

setContent (*xnpts*, *ynpts*, *qmax*, *beam*, *zmin=None*, *zmax=None*, *sym=False*)
received value and displayed them

Parameters

- **xnpts** – the number of point of the x_bins of data
- **ynpts** – the number of point of the y_bins of data
- **qmax** – the maxmimum value of data pixel
- **beam** – the radius of the beam
- **zmin** – the value to get the minimum color
- **zmax** – the value to get the maximum color
- **sym** –

sas.sasgui.guiframe.local_perspectives.plotting.graphAppearance module

Dialog for general graph appearance

This software was developed by Institut Laue-Langevin as part of Distributed Data Analysis of Neutron Scattering Experiments (DANSE).

Copyright 2012 Institut Laue-Langevin

class sas.sasgui.guiframe.local_perspectives.plotting.graphAppearance.**graphAppearance**(*pa ti- tle leg en*

Bases: wx._windows.Frame

InitUI()

fillLegendLocs()

get_legend_loc()

get_loc_label()
Associates label to a specific legend location

get_togglegrid()

get_togglelegend()

get_xcolor()

get_xfont()

get_xlab()

get_xtick_check()

get_xunit()

get_ycolor()

get_yfont()

get_ylab()

get_ytick_check()

get_yunit()

on_cancel(*e*)

on_ok(*e*)

on_x_font(*e*)

on_y_font(*e*)

setDefaults(*grid*, *legend*, *xlab*, *ylab*, *xunit*, *yunit*, *xaxis_font*, *yaxis_font*, *legend_loc*, *xcolor*, *ycolor*, *is_xtick*, *is_ytick*)

xfill_colors()

yfill_colors()

sas.sasgui.guiframe.local_perspectives.plotting.masking module

Mask editor

class sas.sasgui.guiframe.local_perspectives.plotting.masking.**CalcPlot**(*id=-1*, *panel=None*, *image=None*, *completefn=None*, *updatefn=None*, *elapsed=0*, *yieldtime=0.01*, *worktime=0.01*)

Bases: *sas.sascalc.data_util.calcthread.CalcThread*

Compute Resolution

compute()
executing computation

class sas.sasgui.guiframe.local_perspectives.plotting.masking.**FloatPanel**(*parent=None*, *base=None*, *data=None*, *dimension=1*, *id=139*, **args*, ***kwds*)

Bases: `wx._windows.Dialog`

Provides the Mask Editor GUI.

CENTER_PANE = False

ID = 139

OnClose(*event*)

complete(*panel*, *image*, *elapsed=None*)
Plot image

Parameters **image** – newplot [plotpanel]

freeze_axes()
freeze axes

get_plot()
Get Plot panel

set_plot_unfocus()
Not implemented

thaw_axes()
thaw axes

window_caption = 'Plot'

window_name = 'Plot'

class sas.sasgui.guiframe.local_perspectives.plotting.masking.**MaskPanel**(*parent=None*, *base=None*, *data=None*, *id=-1*, **args*, ***kwds*)

Bases: wx._windows.Dialog

Provides the Mask Editor GUI.

CENTER_PANE = True

OnClose(*event*)
Processing close event

ShowMessage(*msg=''*)
Show error message when mask covers whole data area

freeze_axes()
freeze axes

onMouseMotion(*event*)
onMotion event

onWheel(*event*)
on wheel event

set_plot_unfocus()
Not implemented

thaw_axes()
thaw axes

update(*draw=True*)
Respond to changes in the model by recalculating the profiles and resetting the widgets.

window_caption = 'Mask Editor'

window_name = 'Mask Editor'

class sas.sasgui.guiframe.local_perspectives.plotting.masking.**Maskplotpanel**(*parent*, *id=-1*, *dimension=2*, *color=None*, *dpi=None*, ***kwargs*)

Bases: *sas.sasgui.plottools.PlotPanel.PlotPanel*

PlotPanel for Quick plot and masking plot

add_image(*plot*)
Add Image

add_toolbar()
Add toolbar

draw()
Draw

onContextMenu(*event*)
Default context menu for a plot panel

onLeftDown(*event*)
Disables LeftDown

onMouseMotion(*event*)
Disable dragging 2D image

onPick(*event*)
Disables OnPick

onWheel(*event*)

on_set_focus(*event*)
send to the parenet the current panel on focus

class sas.sasgui.guiframe.local_perspectives.plotting.masking.**ViewApp**(*redirect=False*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*)

Bases: `wx._core.App`

OnInit()

class sas.sasgui.guiframe.local_perspectives.plotting.masking.**ViewerFrame**(*parent*, *id*, *title*)

Bases: `wx._windows.Frame`

Add comment

sas.sasgui.guiframe.local_perspectives.plotting.parameters_panel_boxsum module

class sas.sasgui.guiframe.local_perspectives.plotting.parameters_panel_boxsum.**SlicerPane**

Bases: `wx._windows.Panel`, *sas.sasgui.guiframe.panel_base.PanelBase*

Panel class to show the slicer parameters

CENTER_PANE = False

on_close(*event*)
On Close Event

on_set_focus(*evt*)
Highlight the txtcrtl

on_text_enter(*evt*)
Parameters have changed

set_slicer(*type*, *params*)
Rebuild the panel

window_caption = 'Slicer Panel'

window_name = 'Slicer panel'

sas.sasgui.guiframe.local_perspectives.plotting.parameters_panel_slicer module

class sas.sasgui.guiframe.local_perspectives.plotting.parameters_panel_slicer.**SlicerPara**

Bases: wx._windows.Dialog

Panel for dynamically changing slicer parameters and apply the same slicer to multiple 2D plot panels

apply_params_list_and_process(*evt=None*)
Event based parameter setting.

Parameters evt – Event triggered to apply parameters to a list of plots evt should have attrs plot_list and params

check_item_and_children(*data_ctrl*, *check_value=True*)

on_auto_save_checked(*evt=None*)
Enable/Disable auto append when checkbox is checked :param evt: Event

on_batch_slicer(*evt=None*)
Event triggered when batch slicing button is pressed :param evt: Event triggering the batch slicing

on_change_slicer(*evt*)
Event driven slicer change when self.type_select changes :param evt: Event triggering this change

on_check_box_list(*evt=None*)
Prevent a checkbox item from being unchecked :param evt: Event triggered when a checkbox list item is checked

on_evt_slicer(*event*)
Process EVT_SLICER events When the slicer changes, update the panel

Parameters event – EVT_SLICER event

on_help(*event=None*)
Opens a help window for the slicer parameters/batch slicing window :param event: :return:

on_param_change(*evt*)
receive an event end reset value text fields inside self.parameters

on_text_enter(*evt*)
Parameters have changed

process_list()
Populate the check list from the currently plotted 2D data

save_files(*evt=None*)
Automatically save the sliced data to file. :param evt: Event that triggered the call to the method

send_to_fitting(*fit='No fitting'*, *file_list=None*)
Send a list of data to the fitting perspective :param fit: fit type desired :param file_list: list of loaded file names to send to fit

set_slicer(*type*, *params*)
Rebuild the panel

update_file_append(*params=None*)
Update default_value when any parameters are changed :param params: dictionary of parameters

sas.sasgui.guiframe.local_perspectives.plotting.plotting module

class sas.sasgui.guiframe.local_perspectives.plotting.plotting.**Plugin**

Bases: *sas.sasgui.guiframe.plugin_base.PluginBase*

Plug-in class to be instantiated by the GUI manager

clear_panel()

Clear and Hide all plot panels, and remove them from menu

clear_panel_by_id(*group_id*)

clear the graph

create_1d_panel(*data*, *group_id*)

create_2d_panel(*data*, *group_id*)

create_panel_helper(*new_panel*, *data*, *group_id*, *title=None*)

delete_panel(*group_id*)

get_panels(*parent*)

Create and return a list of panel objects

hide_panel(*group_id*)

hide panel with group ID = group_id

is_always_active()

return True is this plugin is always active even if the user is switching between perspectives

populate_menu(*parent*)

Create a 'Plot' menu to list the panels available for displaying

Parameters

- **id** – next available unique ID for wx events
- **parent** – parent window

remove_plot(*group_id*, *id*)

remove plot of ID = id from a panel of group ID =group_id

set_panel_on_focus(*panel*)

update_panel(*data*, *panel*)

update the graph of a given panel

sas.sasgui.guiframe.local_perspectives.plotting.profile_dialog module

SLD Profile Dialog for multifunctional models

class sas.sasgui.guiframe.local_perspectives.plotting.profile_dialog.**SLDPanel**(*parent=None, base=None, data=None, axes=['Radiu id=- 1, *args, **kwds*)

Bases: wx._windows.Dialog

Provides the SLD profile plot panel.

CENTER_PANE = True

disable_app_menu(*panel*)

Disable menu bar

get_current_context_menu(*graph=None*)
When the context menu of a plot is rendered, the get_context_menu method will be called to give you a chance to add a menu item to the context menu. :param graph: the Graph object to which we attach the context menu

Returns a list of menu items with call-back function

on_change_caption(*name*, *old_caption*, *new_caption*)

set_plot_unfocus()
Set_plot unfocus

set_schedule(*schedule=False*)
Set schedule for redraw

set_schedule_full_draw(*panel=None*, *func=None*)
Set_schedule for full draw

show_data1d(*data*, *name*)
Show data dialog

window_caption = 'Scattering Length Density Profile'

window_name = 'Scattering Length Density Profile'

class sas.sasgui.guiframe.local_perspectives.plotting.profile_dialog.**SLDplotpanel**(*parent*, *axes=[]*, *id=-1*, *color=N*, *dpi=Nor*, ***kwarg*

Bases: *sas.sasgui.guiframe.local_perspectives.plotting.Plotter1D.ModelPanel1D*

add_image(*plot*)
Add image(Theory1D)

onChangeCaption(*event*)
Not implemented

on_kill_focus(*event*)
reset the panel color

on_set_focus(*event*)
send to the parenet the current panel on focus

class sas.sasgui.guiframe.local_perspectives.plotting.profile_dialog.**ViewApp**(*redirect=False*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*)

Bases: wx._core.App

OnInit()

class sas.sasgui.guiframe.local_perspectives.plotting.profile_dialog.**ViewerFrame**(*parent*, *id*, *title*)

Bases: wx._windows.Frame

Add comment

sas.sasgui.guiframe.local_perspectives.plotting.sector_mask module

Sector mask interactor

class `sas.sasgui.guiframe.local_perspectives.plotting.sector_mask.`**`SectorMask`**(*base*, *axes*, *color='gray'*, *zorder=3*, *side=False*)

Bases: `sas.sasgui.guiframe.local_perspectives.plotting.BaseInteractor._BaseInteractor`

Draw a sector slicer.Allow to find the data 2D inside of the sector lines

`clear`()
Clear the slicer and all connected events related to this slicer

`draw`()

`freeze_axes`()

`get_params`()
Store a copy of values of parameters of the slicer into a dictionary.

Return params the dictionary created

`move`(*x*, *y*, *ev*)
Process move to a new position, making sure that the move is allowed.

`moveend`(*ev*)
Called a dragging motion ends.Get slicer event

`restore`()
Restore the roughness for this layer.

`save`(*ev*)
Remember the roughness for this layer and the next so that we can restore on Esc.

`set_cursor`(*x*, *y*)

`set_params`(*params*)
Receive a dictionary and reset the slicer with values contained in the values of the dictionary.

Parameters **`params`** – a dictionary containing name of slicer parameters and values the user assigned to the slicer.

`thaw_axes`()

`update`()
Respond to changes in the model by recalculating the profiles and resetting the widgets.

Module contents

Module contents

Submodules

sas.sasgui.guiframe.CategoryInstaller module

Class for making sure all category stuff is installed and works fine.

Copyright (c) Institut Laue-Langevin 2012

@author kieranrcampbell@gmail.com @modified by NIST/MD sasview team

class `sas.sasgui.guiframe.CategoryInstaller.`**`CategoryInstaller`**

Bases: `object`

Class for making sure all category stuff is installed

Note - class is entirely static!

static check_install(*homedir=None*, *model_list=None*)

Makes sure categories.json exists and if not compile it and install.

This is the main method of this class.

Parameters

- **homefile** – Override the default home directory
- **model_list** – List of model names except those in Plugin Models which are user supplied.

static get_default_file()

static get_user_file()

returns the user data file, eg .sasview/categories.json.json

sas.sasgui.guiframe.CategoryManager module

This software was developed by Institut Laue-Langevin as part of Distributed Data Analysis of Neutron Scattering Experiments (DANSE).

Copyright 2012 Institut Laue-Langevin

class `sas.sasgui.guiframe.CategoryManager.`**`CategoryManager`**(*parent*, *win_id*, *title*)

Bases: `wx._windows.Frame`

A class for managing categories

dial_ok(*dialog=None*, *model=None*)

modify_dialog onclose

class `sas.sasgui.guiframe.CategoryManager.`**`ChangeCat`**(*parent*, *title*, *cat_list*, *current_cats*)

Bases: `wx._windows.Dialog`

dialog for changing the categories of a model

get_category()

Returns a list of categories applying to this model

on_add(*event*)

Callback for new category added

on_existing(*event*)

Callback for existing category selected

on_newcat(*event*)

Callback for new category added

on_ok_mac(*event*)

On OK pressed (MAC only)

on_remove(*event*)

Callback for a category removed

class `sas.sasgui.guiframe.CategoryManager.`**`CheckListCtrl`**(*parent*, *callback_func*)

Bases: `wx._controls.ListCtrl`, `wx.lib.mixins.listctrl.CheckListCtrlMixin`, `wx.lib.mixins.listctrl.ListCtrlAutoWidthMixin`

Taken from http://zetcode.com/wxpython/advanced/

OnCheckItem(*index*, *flag*)
When the user checks the item we need to save that state

sas.sasgui.guiframe.CategoryManager.**logger = <logging.Logger object>**
Notes The category manager mechanism works from 3 data structures used: - self.master_category_dict: keys are the names of categories, the values are lists of tuples, the first being the model names (the models belonging to that category), the second a boolean of whether or not the model is enabled - self.by_model_dict: keys are model names, values are a list of categories belonging to that model - self.model_enabled_dict: keys are model names, values are bools of whether the model is enabled use self._regenerate_model_dict() to create the latter two structures from the former use self._regenerate_master_dict() to create the first structure from the latter two

The need for so many data structures comes from the fact sometimes we need fast access to all the models in a category (eg user selection from the gui) and sometimes we need access to all the categories corresponding to a model (eg user modification of model categories)

sas.sasgui.guiframe.aboutbox module

class sas.sasgui.guiframe.aboutbox.**DialogAbout**(**args*, ***kwds*)
Bases: `wx._windows.Dialog`

"About" Dialog

Shows product name, current version, authors, and link to the product page. Current version is taken from version.py

onAnstoLogo(*event*)

onBamLogo(*event*)

onDanseLogo(*event*)

onDlsLogo(*event*)

onEssLogo(*event*)

onIllLogo(*event*)

onIsisLogo(*event*)

onNistLogo(*event*)

onNsfLogo(*event*)

onOrnlLogo(*event*)

onSnsLogo(*event*)

onTudelftLogo(*event*)

onUTLogo(*event*)

onUmdLogo(*event*)

class sas.sasgui.guiframe.aboutbox.**MyApp**(*redirect=False*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*)
Bases: `wx._core.App`

OnInit()

sas.sasgui.guiframe.aboutbox.**launchBrowser**(*url*)
Launches browser and opens specified url

In some cases may require BROWSER environment variable to be set up.

Parameters **url** – URL to open

sas.sasgui.guiframe.acknowledgebox module

Created on Feb 18, 2015

@author: jkrzywon

class sas.sasgui.guiframe.acknowledgebox.**DialogAcknowledge**(**args*, ***kwds*)

Bases: wx._windows.Dialog

"Acknowledgement" Dialog Box

Shows the current method for acknowledging SasView in scholarly publications.

class sas.sasgui.guiframe.acknowledgebox.**MyApp**(*redirect=False*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*)

Bases: wx._core.App

Class for running module as stand alone for testing

OnInit()

Defines an init when running as standalone

sas.sasgui.guiframe.config module

Application settings

sas.sasgui.guiframe.config.**printEVT**(*message*)

sas.sasgui.guiframe.custom_pstats module

class sas.sasgui.guiframe.custom_pstats.**CustomPstats**(**args*, ***kwds*)

Bases: pstats.Stats

write_stats(**amount*)

sas.sasgui.guiframe.custom_pstats.**f8**(*x*)

sas.sasgui.guiframe.custom_pstats.**func_std_string**(*func_name*)

sas.sasgui.guiframe.custom_pstats.**profile**(*fn*, *name='profile.txt'*, **args*, ***kw*)

sas.sasgui.guiframe.dataFitting module

Adapters for fitting module

class sas.sasgui.guiframe.dataFitting.**Data1D**(*x=None*, *y=None*, *dx=None*, *dy=None*, *lam=None*, *dlam=None*, *isSesans=False*)

Bases: sas.sasgui.plottools.plottables.Data1D, sas.sascalc.dataloader.data_info.Data1D

copy_from_datainfo(*data1d*)

copy values of Data1D of type DataLaoder.Data_info

class sas.sasgui.guiframe.dataFitting.**Data2D**(*image=None*, *err_image=None*, *qx_data=None*, *qy_data=None*, *q_data=None*, *mask=None*, *dqx_data=None*, *dqy_data=None*, *xmin=None*, *xmax=None*, *ymin=None*, *ymax=None*, *zmin=None*, *zmax=None*)

Bases: *sas.sasgui.plottools.plottables.Data2D*, *sas.sascalc.dataloader.data_info.Data2D*

copy_from_datainfo(*data2d*)
copy value of Data2D of type DataLoader.data_info

class sas.sasgui.guiframe.dataFitting.**Theory1D**(*x=None*, *y=None*, *dy=None*)
Bases: *sas.sasgui.plottools.plottables.Theory1D*, *sas.sascalc.dataloader.data_info.Data1D*

copy_from_datainfo(*data1d*)
copy values of Data1D of type DataLaoder.Data_info

sas.sasgui.guiframe.dataFitting.**check_data_validity**(*data*)
Return True is data is valid enough to compute chisqr, else False

sas.sasgui.guiframe.data_manager module

This module manages all data loaded into the application. Data_manager makes available all data loaded for the current perspective.

All modules "creating Data" posts their data to data_manager . Data_manager make these new data available for all other perspectives.

class sas.sasgui.guiframe.data_manager.**DataManager**
Bases: object

Manage a list of data

add_data(*data_list*)
receive a list of

create_gui_data(*data*, *path=None*)
Receive data from loader and create a data to use for guiframe

delete_by_id(*id_list=None*)
save data and path

delete_by_name(*name_list=None*)
save data and path

delete_data(*data_id*, *theory_id=None*, *delete_all=False*)

delete_theory(*data_id*, *theory_id*)

freeze(*theory_id*)

freeze_theory(*data_id*, *theory_id*)

get_all_data()
return list of all available data

get_by_id(*id_list=None*)

get_by_name(*name_list=None*)
return a list of data given a list of data names

get_data_state(*data_id*)
Send list of selected data

get_message()
return message

rename(*name*)
rename data

update_data(*prev_data*, *new_data*)

update_theory(*theory*, *data_id=None*, *state=None*)

sas.sasgui.guiframe.data_panel module

This module provides Graphic interface for the data_manager module.

class sas.sasgui.guiframe.data_panel.**DataDialog**(*data_list*, *parent=None*, *text=''*, **args*, ***kwds*)

Bases: `wx._windows.Dialog`

Allow file selection at loading time

get_data()
return the selected data

class sas.sasgui.guiframe.data_panel.**DataFrame**(*parent=None*, *owner=None*, *manager=None*, *size=(300, 800)*, *list_of_perspective=[]*, *list=[]*, **args*, ***kwds*)

Bases: `wx._windows.Frame`

Data Frame

ALWAYS_ON = True

load_data_list(*list=[]*)
Fill the list inside its panel

window_caption = 'Data Panel'

window_name = 'Data Panel'

class sas.sasgui.guiframe.data_panel.**DataPanel**(*parent*, *list=None*, *size=(255, 750)*, *id=-1*, *list_of_perspective=None*, *manager=None*, **args*, ***kwds*)

Bases: `wx.lib.scrolledpanel.ScrolledPanel`, *sas.sasgui.guiframe.panel_base.PanelBase*

This panel displays data available in the application and widgets to interact with data.

append_theory(*state_id*, *theory_list*)
append theory object under data from a state of id = state_id replace that theory if already displayed

append_theory_helper(*tree*, *root*, *state_id*, *theory_list*)
Append theory helper

check_theory_to_freeze()
Check_theory_to_freeze

define_panel_structure()
Define the skeleton of the panel

disable_app_combo(*enable*)
Disable app combo box

do_layout()
Create the panel layout

enable_append()
enable or disable append button

enable_freeze()
enable or disable the freeze button

enable_import()
enable or disable send button

enable_plot()
enable or disable plot button

enable_remove()
: enable or disable remove button

enable_remove_plot()
: enable remove plot button if there is a plot panel on focus

enable_selection()
: enable or disable combobo box selection

fill_cbox_analysis(*plugin*)
: fill the combobox with analysis name

get_frame()

layout_batch()
: Set up batch mode options

layout_button()
: Layout widgets related to buttons

layout_data_list()
: Add a listcrtl in the panel

layout_selection()
: Create selection option combo box

load_data_list(*list*)
: add need data with its theory under the tree

load_error(*error=None*)
: Pop up an error message.

  Parameters `error` – details error message to be displayed

onContextMenu(*event*)
: Retrieve the state selected state

on_append_plot(*event=None*)
: append plot to plot panel on focus

on_batch_mode(*event*)
: Change to batch mode :param event: UI event

on_check_item(*event*)
: On check item

on_close(*event*)
: On close event

on_close_page(*event=None*)
: On close

on_close_plot(*event*)
: clseo the panel on focus

on_data_info(*event*)
: Data Info panel

on_edit_data(*event*)
: Pop Up Data Editor

on_freeze(*event*)
: On freeze to make a theory to a data set

on_help(*event*)

  Bring up the data manager Documentation whenever the HELP button is clicked.

  Calls DocumentationWindow with the path of the location within the documentation tree (after /doc/ ....". Note that when using old versions of Wx (before 2.9) and thus not the release version of installers, the help comes up at the top level of the file as webbrowser does not pass anything past the # to the browser when it is running "file:///...."

Parameters **`event`** – Triggers on clicking the help button

`on_import`(*event=None*)
Get all select data and set them to the current active perspetive

`on_plot`(*event=None*)
Send a list of data names to plot

`on_plot_3d`(*event*)
Frozen image of 3D

`on_quick_plot`(*event*)
Frozen plot

`on_remove`(*event*, *prompt=True*)
Get a list of item checked and remove them from the treectrl Ask the parent to remove reference to this item

`on_right_click_data`(*event*)
Allow Editing Data

`on_right_click_theory`(*event*)
On click theory data

`on_save_as`(*event*)
Save data as a file

`on_single_mode`(*event*)
Change to single mode :param event: UI event

`remove_by_id`(*id*)
Remove_dat by id

`set_active_perspective`(*name*)
set the active perspective

`set_data_helper`()
Set data helper

`set_frame`(*frame*)

`set_panel_on_focus`(*name=None*)
set the plot panel on focus

`set_plot_unfocus`()
Unfocus plot

`set_schedule_full_draw`(*panel=None*, *func='del'*)
Send full draw to guimanager

`show_data_button`()
show load data and remove data button if dataloader on else hide them

`window_caption = 'Data Explorer'`

`window_name = 'Data Panel'`

`window_type = 'Data Panel'`

class `sas.sasgui.guiframe.data_panel.`**`DataTreeCtrl`**(*parent*, *root*, **args*, ***kwds*)
Bases: `wx.lib.agw.customtreectrl.CustomTreeCtrl`

Check list control to be used for Data Panel

`OnCompareItems`(*item1*, *item2*)
Overrides OnCompareItems in wx.TreeCtrl. Used by the SortChildren method.

class `sas.sasgui.guiframe.data_panel.`**`State`**

DataPanel State

`sas.sasgui.guiframe.data_panel.`**`set_data_state`**(*data=None*, *path=None*, *theory=None*, *state=None*)

Set data state

sas.sasgui.guiframe.data_processor module

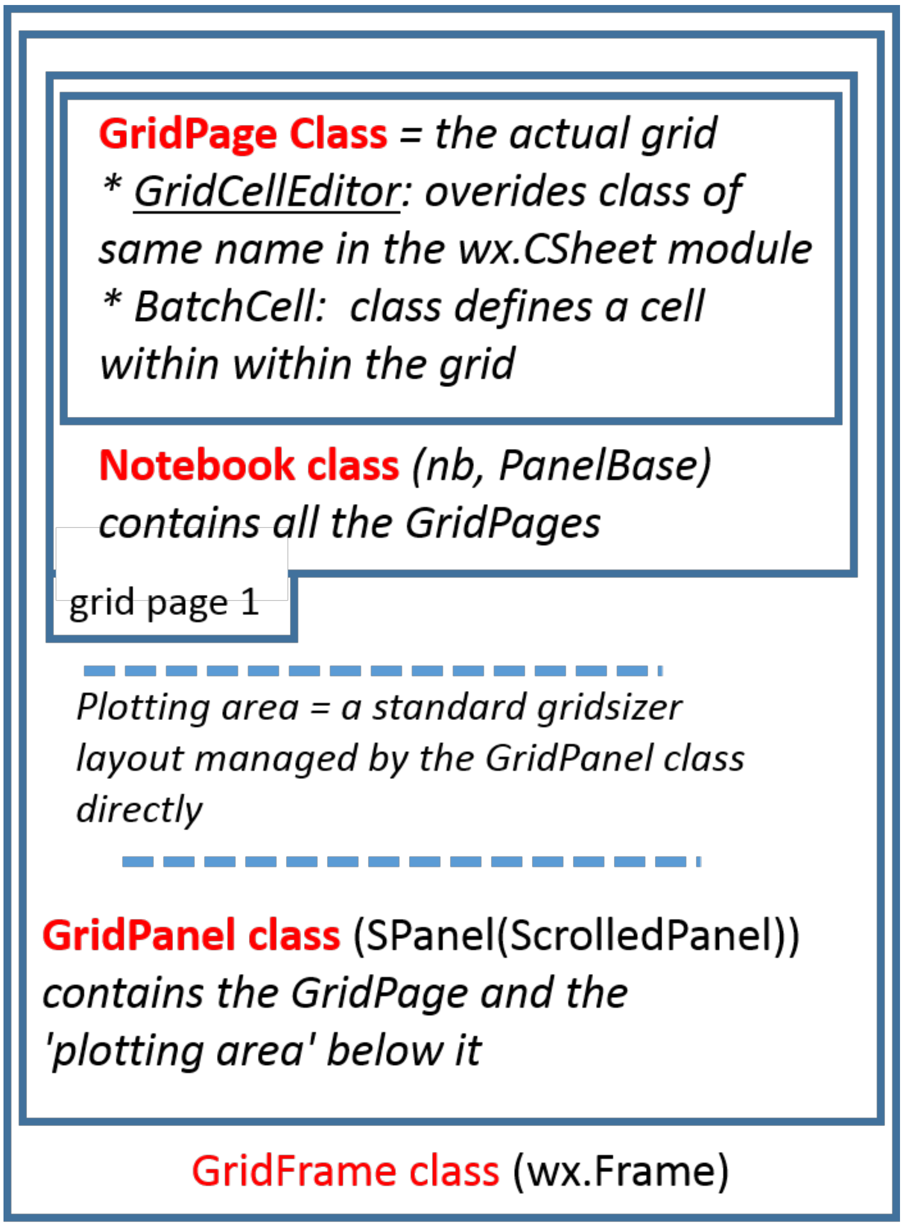
Implement grid used to store results of a batch fit.

This is in Guiframe rather than fitting which is probably where it should be. Actually could be a generic framework implemented in fit gui module. At this point however there this grid behaves independently of the fitting panel and only knows about information sent to it but not about the fits or fit panel and thus cannot feed back to the fitting panel. This could change in the future.

The organization of the classes goes as:

Note: Path to this is: /sasview/src/sas/sasgui/guiframe/data_processor.py

Note: Path to image is: /sasview/src/sas/sasgui/guiframe/media/BatchGridClassLayout.png


GridPage Class = the actual grid
* GridCellEditor: overides class of same name in the wx.CSheet module
* BatchCell: class defines a cell within within the grid
Notebook class (nb, PanelBase) contains all the GridPages
grid page 1
Plotting area = a standard gridsizer layout managed by the GridPanel class directly
GridPanel class (SPanel(ScrolledPanel)) contains the GridPage and the 'plotting area' below it
GridFrame class (wx.Frame)

class `sas.sasgui.guiframe.data_processor.`**`BatchCell`**

Bases: `object`

Object describing a cell in the grid.

class `sas.sasgui.guiframe.data_processor.`**`BatchOutputFrame`**(*parent, data_inputs, data_outputs, file_name='', details='', *args, **kwds*)

Bases: `wx._windows.Frame`

Allow to select where the result of batch will be displayed or stored

`on_apply`(*event*)

Get the user selection and display output to the selected application

`on_close`(*event*)

close the Window

`onselect`(*event=None*)

Receive event and display data into third party application or save data to file.

class sas.sasgui.guiframe.data_processor.**GridCellEditor**(*grid*)

Bases: `wx.lib.sheet.CCellEditor`

Custom cell editor

This subclasses the sheet.CCellEditor (itself a subclass of grid.GridCellEditor) in order to override two of its methods: PaintBackrgound and EndEdit.

This is necessary as the sheet module is broken in wx 3.0.2 and improperly subclasses grid.GridCellEditor

EndEdit(*row*, *col*, *grid*, *previous*)

Commit editing the current cell. Returns True if the value has changed.

Parameters **previous** – previous value in the cell

PaintBackground(*dc*, *rect*, *attr*)

Overrides wx.sheet.CCellEditor.PaintBackground which incorrectly calls the base class method.

In wx3.0 all paint objects must explicitly have a wxPaintDC (Device Context) object. Thus the paint event which generates a call to this method provides such a DC object and the base class in grid expects to receive that object. sheet was apparently not updated to reflect this and hence fails. This could thus become obsolete in a future bug fix of wxPython.

Apart from adding a dc variable in the list of arguments in the def and in the call to the base class the rest of this method is copied as is from sheet.CCellEditor.PaintBackground

From original GridCellEditor docs:

Draws the part of the cell not occupied by the edit control. The base class version just fills it with background colour from the attribute.

Note: There is no need to override this if you don't need to do something out of the ordinary.

Parameters **dc** – the wxDC object for the paint

class sas.sasgui.guiframe.data_processor.**GridFrame**(*parent=None*, *data_inputs=None*, *data_outputs=None*, *id=-1*, *title='Batch Fitting Results Panel'*, *size=(800, 500)*)

Bases: `wx._windows.Frame`

The main wx.Frame for the batch results grid

GetLabelText(*id*)

Get Label Text

add_edit_menu(*menubar*)

populates the edit menu on the menubar. Not activated as of SasView 3.1.0

add_table(*event*)

Add a new table

on_append_column(*event*)

Append a new column to the grid

on_clear(*event*)

On Clear from the Edit menu item on the menubar

on_close(*event*)

on_copy(*event*)

On Copy from the Edit menu item on the menubar

on_menu_open (*event*)
On menu open

on_open (*event*)
Open file containing batch result

on_paste (*event*)
On Paste from the Edit menu item on the menubar

on_remove_column (*event*)
On remove column from the Edit menu Item on the menubar

on_save_page (*event*)
Saves data in grid to a csv file.

At this time only the columns displayed get saved. Thus any error bars not inserted before saving will not be saved in the file

open_with_excel (*event*)
open excel and display batch result in Excel

set_data (*data_inputs*, *data_outputs*, *details=''*, *file_name=None*)
Set data

class sas.sasgui.guiframe.data_processor.**GridPage** (*parent*, *panel=None*)
Bases: wx.lib.sheet.CSheet

Class that receives the results of a batch fit.

GridPage displays the received results in a wx.grid using sheet. This is then used by GridPanel and GridFrame to present the full GUI.

OnCellChange (*event*)
Overrides sheet.CSheet.OnCellChange.

Processes when a cell has been edited by a cell editor. Checks for the edited row being outside the max row to use attribute and if so updates the last row. Then calls the base handler using skip.

OnLeftClick (*event*)
Overrides sheet.CSheet.OnLefClick.

Processes when a cell is selected by left clicking on that cell. First process the base Sheet method then the current class specific method

get_grid_view ()
Return value contained in the grid

get_nofrows ()
Return number of total rows

insert_after_col_menu (*menu*, *label*, *window*)
Method called to populate the 'insert column after current column' submenu

insert_col_menu (*menu*, *label*, *window*)
method called to populate the 'insert column before current column' submenu.

insert_column (*col*, *col_name*)
Insert column at position col with data[col_name] into the current grid.

onContextMenu (*event*)
Method to handle cell right click context menu.

THIS METHOD IS NOT CURRENTLY USED. It is designed to provide a cell pop up context by right clicking on a cell and gives the option to cut, paste, and clear. This will probably be removed in future versions and is being superseded by more traditional cut and paste options.

on_clear (*event*)
Called when clear cell is chosen from cell right click context menu

THIS METHOD IS NOT CURRENTLY USED. it is part of right click cell context menu which is being removed. This will probably be removed in future versions and is being superseded by more traditional cut and paste options

on_copy(*event*)
Called when copy is chosen from cell right click context menu

THIS METHOD IS NOT CURRENTLY USED. it is part of right click cell context menu which is being removed. This will probably be removed in future versions and is being superseded by more traditional cut and paste options

on_insert_after_column(*event*)
Called when user chooses insert 'column after' submenu of the column context menu obtained when right clicking on a given column header.

Sets up to insert column into the current grid after the current highlighted column location and sets up what to populate that column with. Then calls insert_column method to actually do the insertion.

on_insert_column(*event*)
Called when user chooses insert 'column before' submenu of the column context menu obtained when right clicking on a given column header.

Sets up to insert column into the current grid before the current highlighted column location and sets up what to populate that column with. Then calls insert_column method to actually do the insertion.

on_left_click(*event*)
Is triggered when the left mouse button is clicked while the mouse is hovering over the column 'label.'

This processes the information on the selected column: the column name (in row 0 of column) and the range of cells with a valid value to be used by the GridPanel set_axis methods.

on_paste(*event*)
Called when paste is chosen from cell right click context menu

THIS METHOD IS NOT CURRENTLY USED. it is part of right click cell context menu which is being removed. This will probably be removed in future versions and is being superseded by more traditional cut and paste options

on_remove_column(*event=None*)
Called when user chooses remove from the column right click menu Checks the columnn exists then calls the remove_column method

on_right_click(*event*)
Is triggered when the right mouse button is clicked while the mouse is hovering over the column 'label.'

This brings up a context menu that allows the deletion of the column, or the insertion of a new column either to the right or left of the current column. If inserting a new column can insert a blank column or choose a number of hidden columns. By default all the error parameters are in hidden columns so as to save space on the grid. Also any other intrinsic variables stored with the data such as Temperature, pressure, time etc can be used to populate this menu.

on_selected_cell(*event*)
Handler catching cell selection.

Called after calling base 'on left click' method.

on_set_x_axis(*event*)
Just calls the panel version of the method

on_set_y_axis(*event*)
Just calls the panel version of the method

remove_column(*col*, *numCols=1*)
Remove the col column from the current grid

set_data(*data_inputs*, *data_outputs*, *details*, *file_name*)
Add data to the grid

Parameters

- **`data_inputs`** – data to use from the context menu of the grid
- **`data_ouputs`** – default columns displayed

`set_grid_values`()
Set the values in grids

class `sas.sasgui.guiframe.data_processor.`**`GridPanel`**(*parent*, *data_inputs=None*, *data_outputs=None*, **args*, ***kwds*)
Bases: *`sas.sasgui.guiframe.data_processor.SPanel`*

A ScrolledPanel class that contains the grid sheet as well as a number of widgets to create interesting plots and buttons for help etc.

`add_column`()

`create_axis_label`(*cell_list*)
Receive a list of cells and create a string presenting the selected cells.

Parameters **`cell_list`** – list of tuple

`edit_axis_helper`(*tcrtl_label*, *tcrtl_title*, *label*, *title*)
get controls to modify

`get_plot_axis`(*col*, *list*)

`get_sentence`(*dict*, *sentence*, *column_names*)
Get sentence from dict

`layout_grid`()
Draw the area related to the grid by adding it as the first element in the panel's grid_sizer

`layout_plotting_area`()
Add the area containing all the plot options, buttons etc to a plotting area sizer to later be added to the top level grid_sizer

`on_edit_axis`(*event*)
Get the selected column on the visible grid and set values for axis

`on_help`(*event*)
Bring up the Batch Grid Panel Usage Documentation whenever the HELP button is clicked.

Calls DocumentationWindow with the path of the location within the documentation tree (after /doc/ ....". Note that when using old versions of Wx (before 2.9) and thus not the release version of installers, the help comes up at the top level of the file as webbrowser does not pass anything past the # to the browser when it is running "file:///...."

Parameters **`evt`** – Triggers on clicking the help button

`on_plot`(*event*)
Evaluate the contains of textcrtl and plot result

`on_remove_column`()

`on_view`(*event*)
Get object represented by the given cells and plot them. Basically plot the column in y vs the column in x.

`set_dyaxis`(*label=''*, *dy=None*)

`set_xaxis`(*label=''*, *x=None*)

`set_yaxis`(*label=''*, *y=None*)

class `sas.sasgui.guiframe.data_processor.`**`Notebook`**(*parent*, *manager=None*, *data=None*, **args*, ***kwargs*)
Bases: `wx.aui.AuiNotebook`, *`sas.sasgui.guiframe.panel_base.PanelBase`*

## Internal name for the AUI manager window_name = "Fit panel" ## Title to appear on top of the window

add_column()
Append a new column to the grid

add_empty_page()

create_axis_label(*cell_list*)
Receive a list of cells and create a string presenting the selected cells that can be used as data for one axis of a plot.

Parameters **cell_list** – list of tuple

enable_close_button()
display the close button on the tab if more than 1 tab exits. Otherwise remove the close button

get_column_labels()
return dictionary of columns labels on the current page

get_highlighted_row(*is_number=True*)
Add highlight rows

get_odered_results(*inputs*, *outputs=None*)
Order a list of 'inputs.' Used to sort rows and columns to present in batch results grid.

on_close_page(*event*)
close the page

on_edit_axis()
Return the select cell range from a given selected column. Checks that all cells are from the same column

on_remove_column()
Remove the selected column from the grid

set_data(*data_inputs*, *data_outputs*, *details="*, *file_name=None*)

window_caption = 'Notebook '

class sas.sasgui.guiframe.data_processor.**SPanel**(*parent*, **args*, ***kwds*)
Bases: wx.lib.scrolledpanel.ScrolledPanel

ensure proper scrolling of GridPanel

Adds a SetupScrolling call to the normal ScrolledPanel init. GridPanel then subclasses this class

sas.sasgui.guiframe.data_processor.**parse_string**(*sentence*, *list*)
Return a dictionary of column label and index or row selected

Parameters

- **sentence** – String to parse
- **list** – list of columns label

Returns col_dict

sas.sasgui.guiframe.data_state module

class sas.sasgui.guiframe.data_state.**DataState**(*data=None*, *parent=None*)
Bases: object

Store information about data

clone()

get_data()

get_message()
return message

get_name()

get_path()
return the path of the loaded data

get_theory()

set_data(*data*)

set_name(*name*)

set_path(*path*)
Set the path of the loaded data

set_theory(*theory_data*, *theory_state=None*)

sas.sasgui.guiframe.documentation_window module

documentation module provides a simple means to add help throughout the application. It checks for the existence of html2 package needed to support fully html panel which supports css. The class defined here takes a title for the particular help panel, a pointer to the html documentation file of interest within the documentation tree along with a 'command' string such as a page anchor or a query string etc. The path to the doc directory is retrieved automatically by the class itself. Thus with these three pieces of information the class generates a panel with the appropriate title bar and help file formatted according the style sheets called in the html file. Finally, if an old version of Python is running and the html2 package is not available the class brings up the default browser and passes the file:/// string to it. In this case however the instruction portion is usually not passed for security reasons.

class sas.sasgui.guiframe.documentation_window.**DocumentationWindow**(*parent*, *dummy_id*, *path*, *url_instruction*, *title*, *size=(850, 540)*)

Bases: `wx._windows.Frame`

DocumentationWindow inherits from wx.Frame and provides a centralized coherent framework for all help documentation. Help files must be html files stored in an properly organized tree below the top 'doc' folder. In order to display the appropriate help file from anywhere in the gui, the code simply needs to know the location below the top level where the help file resides along with the name of the help file. called (self, parent, dummy_id, path, url_instruction, title, size=(850, 540))

Parameters

- **path** – path to html file beginning AFTER /doc/ and ending in the file.html.
- **url_instructions** – anchor string or other query e.g. '#MyAnchor'
- **title** – text to place in the title bar of the help panel

OnError(*evt*)

sas.sasgui.guiframe.documentation_window.**main**()
main loop function if running alone for testing.

sas.sasgui.guiframe.documentation_window.**start_documentation_server**(*doc_root*, *port*)

sas.sasgui.guiframe.dummyapp module

Dummy application. Allows the user to set an external data manager

class sas.sasgui.guiframe.dummyapp.**DummyView**(*redirect=False*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*)

Bases: *sas.sasgui.guiframe.gui_manager.SasViewApp*

class sas.sasgui.guiframe.dummyapp.**SasView**

class sas.sasgui.guiframe.dummyapp.**TestPlugin**(*name='Test_plugin'*)

Bases: *sas.sasgui.guiframe.plugin_base.PluginBase*

get_context_menu(*graph=None*)

This method is optional.

When the context menu of a plot is rendered, the get_context_menu method will be called to give you a chance to add a menu item to the context menu.

A ref to a Graph object is passed so that you can investigate the plot content and decide whether you need to add items to the context menu.

This method returns a list of menu items. Each item is itself a list defining the text to appear in the menu, a tool-tip help text, and a call-back method.

Parameters **graph** – the Graph object to which we attach the context menu

Returns a list of menu items with call-back function

get_panels(*parent*)

Create and return the list of wx.Panels for your plug-in. Define the plug-in perspective.

Panels should inherit from DefaultPanel defined below, or should present the same interface. They must define “window_caption” and “window_name”.

Parameters **parent** – parent window

Returns list of panels

get_tools()

Returns a set of menu entries for tools

populate_menu(*parent*)

Create and return the list of application menu items for the plug-in. :param parent: parent window

Returns plug-in menu

sas.sasgui.guiframe.events module

sas.sasgui.guiframe.gui_manager module

Gui manager: manages the widgets making up an application

class sas.sasgui.guiframe.gui_manager.**DefaultPanel**(*parent*, **args*, ***kwds*)

Bases: wx._windows.Panel, *sas.sasgui.guiframe.panel_base.PanelBase*

Defines the API for a panels to work with the GUI manager

CENTER_PANE = True

window_caption = 'Welcome panel'

window_name = 'default'

class sas.sasgui.guiframe.gui_manager.**MDIFrame**(*parent*, *panel*, *title='Untitled'*, *size=(300, 200)*)

Bases: `wx._windows.Frame`

Frame for panels

OnClose(*event*)
On Close event

set_panel(*panel*)

set_panel_focus(*event*)

show_data_panel(*action*)
Turns on the data panel

The the data panel is optional. Most of its functions can be performed from the menu bar and from the plots.

class sas.sasgui.guiframe.gui_manager.**SasViewApp**(*redirect=False*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*)

Bases: `wx._core.App`

SasView application

OnInit()
When initialised

add_perspective(*perspective*)
Manually add a perspective to the application GUI

build_gui()
Build the GUI

clean_plugin_models(*path*)
Delete plugin models in app folder

Parameters **path** – path of the plugin_models folder in app

display_splash_screen(*parent*, *path='/home/sasview/Jenkins/workspace/SasView_Ubuntu14.10/sasview/build/lib.lin*
x86_64-2.7/sas/sasview/images/SVwelcome_mini.png')
Displays the splash screen. It will exactly cover the main frame.

maximize_win()
Maximize the window after the frame shown

on_close_splash_screen(*event*)
When the splash screen is closed.

open_file()
open a state file at the start of the application

set_manager(*manager*)
Sets a reference to the application manager of the GUI manager (Frame)

set_welcome_panel(*panel_class*)
Set the welcome panel

Parameters **panel_class** – class of the welcome panel to be instantiated

window_placement(*size*)
Determines the position and size of the application frame such that it fits on the user's screen without obstructing (or being obstructed by) the Windows task bar. The maximum initial size in pixels is bounded by WIDTH x HEIGHT. For most monitors, the application will be centered on the screen; for very large monitors it will be placed on the left side of the screen.

class sas.sasgui.guiframe.gui_manager.**ViewerFrame**(*parent*, *title*, *size=(-1, -1)*, *gui_style=381*, *style=541072960*, *pos=wx.Point(-1, -1)*)

Bases: wx._windows.Frame

Main application frame

Close(*event=None*)
Quit the application

PopStatusText(**args*, ***kwds*)

PushStatusText(**args*, ***kwds*)

SetStatusText(**args*, ***kwds*)

WindowClose(*event=None*)
Quit the application from x icon

add_data(*data_list*)
receive a dictionary of data from loader store them its data manager if possible send to data the current active perspective if the data panel is not active. :param data_list: dictionary of data's ID and value Data

add_data_helper(*data_list*)

add_icon()
get list of child and attempt to add the default icon

add_perspective(*plugin*)
Add a perspective if it doesn't already exist.

append_bookmark(*event=None*)
Bookmark available information of the panel on focus

build_gui()
Build the GUI by setting up the toolbar, menu and layout.

check_multimode(*perspective=None*)
Check the perspective have batch mode capablitity

create_gui_data(*data*, *path=None*)

delete_data(*data*)
Delete the data.

delete_panel(*uid*)
delete panel given uid

disable_app_menu(*p_panel=None*)
Disables all menus in the menubar

enable_add_data(*new_plot*)
Enable append data on a plot panel

enable_bookmark()
Bookmark

enable_copy()
enable copy related control

enable_drag(*event=None*)
drag

enable_edit_menu()
enable menu item under edit menu depending on the panel on focus

enable_paste()
enable paste

enable_preview()
preview

enable_print()
print

enable_redo()
enable redo

enable_reset()
reset the current panel

enable_save()
save

enable_undo()
enable undo related control

enable_zoom()
zoom

enable_zoom_in()
zoom in

enable_zoom_out()
zoom out

freeze(*data_id*, *theory_id*)
Saves theory/model and passes to data loader.

..warning:: This seems to be the exact same code as the next function called simply freeze. This probably needs fixing

full_draw()
Draw the panels with axes in the schedule to full dwar list

get_client_size()
return client size tuple

get_context_menu(*plotpanel=None*)
Get the context menu items made available by the different plug-ins. This function is used by the plotting module

get_current_context_menu(*plotpanel=None*)
Get the context menu items made available by the current plug-in. This function is used by the plotting module

get_current_perspective()
return the current perspective

get_data(*path*)

get_data_manager()
return the data manager.

get_paneinfo(*name*)
Get pane Caption from window_name

Parameters **name** – window_name in AuiPaneInfo

Returns AuiPaneInfo of the name

get_save_location()
return the _default_save_location

get_schedule()
Get schedule

get_style()
Return the gui style

get_toolbar()
return the toolbar.

get_toolbar_height()

get_window_size()
Get window size

Returns size

Return type tuple

load_data(*path*)
load data from command line

load_folder(*path*)
Load entire folder

load_from_cmd(*path*)
load data from cmd or application

load_state(*path*, *is_project=False*)
load data from command line or application

on_batch_selection(*event=None*)

Parameters **event** – contains parameter enable. When enable is set to True the application is in Batch mode otherwise the application is in Single mode.

on_bookmark_panel(*event=None*)
bookmark panel

on_category_panel(*event*)
On cat panel

on_change_caption(*name*, *old_caption*, *new_caption*)
Change the panel caption

Parameters

- **name** – window_name of the pane
- **old_caption** – current caption [string]
- **new_caption** – new caption [string]

on_change_categories(*evt*)

on_close_welcome_panel()
Close the welcome panel

on_color_selection(*event*)

Parameters **event** – contains parameters for id and color

on_copy_panel(*event=None*)
copy the last panel on focus if possible

on_drag_panel(*event=None*)
drag apply to the panel on focus

on_load_data(*event*)
received an event to trigger load from data plugin

on_panel_close(*event*)
Gets called when the close event for a panel runs. This will check which panel has been closed and delete it.

on_paste_panel(*event=None*)
paste clipboard to the last panel on focus

on_preview_panel(*event=None*)
preview information on the panel on focus

on_print_panel(*event=None*)
print available information on the last panel on focus

on_read_batch_tofile(*base*)
Open a file dialog , extract the file to read and display values into a grid

on_redo_panel(*event=None*)
redo the last cancel action done on the last panel on focus

on_reset_panel(*event=None*)
reset the current panel

on_save_helper(*doc*, *reader*, *panel*, *path*)
Save state into a file

on_save_panel(*event=None*)
save possible information on the current panel

on_set_batch_result(*data_outputs*, *data_inputs=None*, *plugin_name="*)
Display data into a grid in batch mode and show the grid

on_set_plot_focus(*panel*)
Set focus on a plot panel

on_undo_panel(*event=None*)
undo previous action of the last panel on focus if possible

on_view(*evt*)
A panel was selected to be shown. If it's not already shown, display it.

Parameters **evt** – menu event

on_zoom_in_panel(*event=None*)
zoom in of the panel on focus

on_zoom_out_panel(*event=None*)
zoom out on the panel on focus

on_zoom_panel(*event=None*)
zoom on the current panel if possible

onfreeze(*theory_id*)
Saves theory/model and passes to data loader.

..warning:: This seems to be the exact same code as the next function called simply freeze. This probably needs fixing

open_with_externalapp(*data*, *file_name*, *details="*)
Display data in the another application , by default Excel

open_with_localapp(*data_inputs=None*, *details="*, *file_name=None*, *data_outputs=None*)
Display value of data into the application grid :param data_inputs: dictionary of string and list of items :param details: descriptive string :param file_name: file name :param data_outputs: Data outputs

plot_data(*state_id*, *data_id=None*, *theory_id=None*, *append=False*)
send a list of data to plot

popup_panel(*p*)
Add a panel object to the AUI manager

Parameters **p** – panel object to add to the AUI manager

Returns ID of the event associated with the new panel [int]

post_init()
This initialization method is called after the GUI has been created and all plug-ins loaded. It calls the post_init() method of each plug-in (if it exists) so that final initialization can be done.

put_icon (*frame*)
Put icon on the tap of a panel

quit_guiframe ()
Pop up message to make sure the user wants to quit the application

read_batch_tofile (*file_name*)
Extract value from file name and Display them into a grid

remove_data (*data_id*, *theory_id=None*)
Delete data state if data_id is provide delete theory created with data of id data_id if theory_id is provide if delete all true: delete the all state else delete theory

reset_bookmark_menu (*panel*)
Reset Bookmark menu list

: param panel: a control panel or tap where the bookmark is

save_data1d (*data*, *fname*)
Save data dialog

save_data2d (*data*, *fname*)
Save data2d dialog

send_focus_to_datapanel (*name*)
Send focusing on ID to data explorer

set_current_perspective (*perspective*)
set the current active perspective

set_custom_default_perspective ()
Set default starting perspective

set_data (*data_id*, *theory_id=None*)
set data to current perspective

set_input_file (*input_file*)

Parameters **input_file** – file to read

set_manager (*manager*)
Sets the application manager for this frame

Parameters **manager** – frame manager

set_panel_on_focus (*event*)
Store reference to the last panel on focus update the toolbar if available update edit menu if available

set_panel_on_focus_helper ()
Helper for panel on focus with data_panel

set_perspective (*panels*)
Sets the perspective of the GUI. Opens all the panels in the list, and closes all the others.

Parameters **panels** – list of panels

set_plot_unfocus ()
Un focus all plot panels

set_schedule (*schedule=False*)
Set schedule

set_schedule_full_draw (*panel=None*, *func='del'*)
Add/subtract the schedule full draw list with the panel given

Parameters

- **panel** – plot panel
- **func** – append or del [string]

set_theory(*state_id*, *theory_id=None*)

setup_custom_conf()
Set up custom configuration if exists

show_batch_frame(*event=None*)
show the grid of result

show_data1d(*data*, *name*)
Show data dialog

show_data2d(*data*, *name*)
Show data dialog

show_data_panel(*event=None*, *action=True*)
show the data panel

show_welcome_panel(*event*)
Display the welcome panel

update_data(*prev_data*, *new_data*)
Update the data.

update_theory(*data_id*, *theory*, *state=None*)
Update the theory

write_batch_tofile(*data*, *file_name*, *details="*)
Helper to write result from batch into cvs file

sas.sasgui.guiframe.gui_manager.**custom_value**(*name*, *default=None*)
Fetch a config value from custom_config. Fallback to config, and then to default if it doesn't exist in config.

sas.sasgui.guiframe.gui_statusbar module

Defines and draws the status bar that should appear along the bottom of the main SasView window.

class sas.sasgui.guiframe.gui_statusbar.**Console**(*parent=None*, *status="*, **args*, ***kwds*)
Bases: `wx._windows.Frame`

The main class defining the Console window.

Close(*event*)
Calling close on the panel will hide the panel.

Parameters event – A wx event.

set_message(*status*, *event=None*)
Exposing the base ConsolePanel set_message

Parameters

- **status** – A status message to be sent to the console log.
- **event** – A wx event.

set_multiple_messages(*messages=[]*)
Method to send an arbitrary number of messages to the console log

Parameters messages – A list of strings to be sent to the console log.

class sas.sasgui.guiframe.gui_statusbar.**ConsolePanel**(*parent*, **args*, ***kwargs*)
Bases: `wx._windows.Panel`

Interaction class for adding messages to the Console log.

set_message(*status="*, *event=None*)
Adds a message to the console log as well as the main sasview.log

Parameters

- **status** – A status message to be sent to the console log.
- **event** – A wx event.

class sas.sasgui.guiframe.gui_statusbar.**SPageStatusbar**(*parent*, *timeout=None*, **args*, ***kwds*)

Bases: wx._windows.StatusBar

class sas.sasgui.guiframe.gui_statusbar.**StatusBar**(*parent*, *id*)

Bases: wx._windows.StatusBar

Application status bar

PopStatusText(**args*, ***kwds*)
Override status bar

PushStatusText(**args*, ***kwds*)

SetStatusText(*text=''*, *number=1*, *event=None*)
Set the text that will be displayed in the status bar.

clear_gauge(*msg=''*)
Hide the gauge

enable_clear_gauge()
clear the progress bar

get_msg_position()
Get the last known message that was displayed on the console window.

on_idle(*event*)
When the window is idle, check if the window has been resized

on_size(*evt*)
If the window is resized, redraw the window.

reposition()
Place the various fields in their proper position

set_dialog(*event*)
Display dialogbox

set_gauge(*event*)
change the state of the gauge according the state of the current job

set_icon(*event*)
Display icons related to the type of message sent to the statusbar when available. No icon is displayed if the message is empty

set_message(*event*)
display received message on the statusbar

set_status(*event*)
Update the status bar .

Parameters

- **type** – type of message send. type must be in ["start","progress","update","stop"]
- **msg** – the message itself as string
- **thread** – if updatting using a thread status

sas.sasgui.guiframe.gui_style module

Provide the style for guiframe

class sas.sasgui.guiframe.gui_style.**GUIFRAME**

> **CALCULATOR_ON = 256**
>
> **DATALOADER_ON = 16**
>
> **DEFAULT_STYLE = 92**
>
> **FIXED_PANEL = 4**
>
> **FLOATING_PANEL = 2**
>
> **MANAGER_ON = 1**
>
> **MULTIPLE_APPLICATIONS = 92**
>
> **PLOTTING_ON = 8**
>
> **SINGLE_APPLICATION = 64**
>
> **TOOLBAR_ON = 32**
>
> **WELCOME_PANEL_ON = 128**

class sas.sasgui.guiframe.gui_style.**GUIFRAME_ICON**

> **BOOKMARK_ICON = <wx._core.Image; proxy of <Swig Object of type 'wxImage *' at 0x2f1**
>
> **BOOKMARK_ICON_PATH = '/home/sasview/Jenkins/workspace/SasView_Ubuntu14.10/sasview/b**
>
> **COPY_ICON = <wx._core.Image; proxy of <Swig Object of type 'wxImage *' at 0x2fcc300**
>
> **COPY_ICON_PATH = '/home/sasview/Jenkins/workspace/SasView_Ubuntu14.10/sasview/build**
>
> **DRAG_ICON = <wx._core.Image; proxy of <Swig Object of type 'wxImage *' at 0x26ffd20**
>
> **DRAG_ID_PATH = '/home/sasview/Jenkins/workspace/SasView_Ubuntu14.10/sasview/build/l**
>
> **FRAME_ICON_PATH = '/home/sasview/Jenkins/workspace/SasView_Ubuntu14.10/sasview/buil**
>
> **HIDE_ICON = <wx._core.Image; proxy of <Swig Object of type 'wxImage *' at 0x30aaf10**
>
> **HIDE_ID_PATH = '/home/sasview/Jenkins/workspace/SasView_Ubuntu14.10/sasview/build/l**
>
> **PASTE_ICON = <wx._core.Image; proxy of <Swig Object of type 'wxImage *' at 0x32d0a2**
>
> **PASTE_ICON_PATH = '/home/sasview/Jenkins/workspace/SasView_Ubuntu14.10/sasview/buil**
>
> **PATH = '/home/sasview/Jenkins/workspace/SasView_Ubuntu14.10/sasview/build/lib.linux-**
>
> **PREVIEW_ICON = <wx._core.Image; proxy of <Swig Object of type 'wxImage *' at 0x34b5**
>
> **PREVIEW_ID_PATH = '/home/sasview/Jenkins/workspace/SasView_Ubuntu14.10/sasview/buil**
>
> **PRINT_ICON = <wx._core.Image; proxy of <Swig Object of type 'wxImage *' at 0x30e7bd**
>
> **PRINT_ID_PATH = '/home/sasview/Jenkins/workspace/SasView_Ubuntu14.10/sasview/build/**
>
> **REDO_ICON = <wx._core.Image; proxy of <Swig Object of type 'wxImage *' at 0x26047d0**
>
> **REDO_ICON_PATH = '/home/sasview/Jenkins/workspace/SasView_Ubuntu14.10/sasview/build**
>
> **REPORT_ICON = <wx._core.Image; proxy of <Swig Object of type 'wxImage *' at 0x2e964**
>
> **RESET_ICON = <wx._core.Image; proxy of <Swig Object of type 'wxImage *' at 0x3543cd**
>
> **RESET_ID_PATH = '/home/sasview/Jenkins/workspace/SasView_Ubuntu14.10/sasview/build/**
>
> **SAVE_ICON = <wx._core.Image; proxy of <Swig Object of type 'wxImage *' at 0x2e99d60**
>
> **SAVE_ICON_PATH = '/home/sasview/Jenkins/workspace/SasView_Ubuntu14.10/sasview/build**
>
> **UNDO_ICON = <wx._core.Image; proxy of <Swig Object of type 'wxImage *' at 0x268f6b0**
>
> **UNDO_ICON_PATH = '/home/sasview/Jenkins/workspace/SasView_Ubuntu14.10/sasview/build**

ZOOM_ICON = <wx._core.Image; proxy of <Swig Object of type 'wxImage *' at 0x3531970>

ZOOM_ID_PATH = '/home/sasview/Jenkins/workspace/SasView_Ubuntu14.10/sasview/build/l

ZOOM_IN_ICON = <wx._core.Image; proxy of <Swig Object of type 'wxImage *' at 0x2f30a

ZOOM_IN_ID_PATH = '/home/sasview/Jenkins/workspace/SasView_Ubuntu14.10/sasview/build

ZOOM_OUT_ICON = <wx._core.Image; proxy of <Swig Object of type 'wxImage *' at 0x2e2

ZOOM_OUT_ID_PATH = '/home/sasview/Jenkins/workspace/SasView_Ubuntu14.10/sasview/bui

class `sas.sasgui.guiframe.gui_style.`**GUIFRAME_ID**

BOOKMARK_ID = 104

COPYAS_ID = 116

COPYEX_ID = 114

COPYLAT_ID = 115

COPY_ID = 102

CURRENT_APPLICATION = 113

CURVE_SYMBOL_NUM = 13

DRAG_ID = 109

PASTE_ID = 103

PREVIEW_ID = 111

PRINT_ID = 112

REDO_ID = 101

RESET_ID = 110

SAVE_ID = 105

UNDO_ID = 100

ZOOM_ID = 108

ZOOM_IN_ID = 106

ZOOM_OUT_ID = 107

sas.sasgui.guiframe.gui_toolbar module

class `sas.sasgui.guiframe.gui_toolbar.`**GUIToolBar**(*parent*, **args*, ***kwds*)

Bases: `wx._controls.ToolBar`

Implement toolbar for guiframe

ID_BOOKMARK = 138

add_bookmark_default()

Add default items in bookmark menu

append_bookmark(*event*)

receive item to append on the toolbar button bookmark

append_bookmark_item(*id*, *label*)

Append a item in bookmark

do_layout()

enable_bookmark(*panel*)

enable_copy (*panel*)

enable_paste (*panel*)

enable_preview (*panel*)

enable_print (*panel*)

enable_redo (*panel*)

enable_reset (*panel*)

enable_save (*panel*)

enable_undo (*panel*)

enable_zoom (*panel*)

enable_zoom_in (*panel*)

enable_zoom_out (*panel*)

get_bookmark_items ()
Get bookmark menu items

on_bind_button ()
Bind the buttons

on_bookmark (*event*)
add book mark

remove_bookmark_item (*item*)
Remove a bookmark item

update_button (*application_name=", panel_name="*)

update_toolbar (*panel=None*)

sas.sasgui.guiframe.gui_toolbar.**clear_image** (*image*)

sas.sasgui.guiframe.panel_base module

class sas.sasgui.guiframe.panel_base.**PanelBase** (*parent=None*)
Defines the API for a panels to work with the ViewerFrame toolbar and menu bar

get_bookmark_flag ()
Get the bookmark flag to update appropriately the tool bar

get_copy_flag ()
Get the copy flag to update appropriately the tool bar

get_data ()
return list of current data

get_drag_flag ()
Get the drag flag to update appropriately the tool bar

get_frame ()

get_manager ()

get_paste_flag ()
Get the copy flag to update appropriately the tool bar

get_preview_flag ()
Get the preview flag to update appropriately the tool bar

get_print_flag ()
Get the print flag to update appropriately the tool bar

get_redo_flag()
Get the redo flag to update appropriately the tool bar

get_reset_flag()
Get the reset flag to update appropriately the tool bar

get_save_flag()
Get the save flag to update appropriately the tool bar

get_state()
return the current state

get_undo_flag()
Get the undo flag to update appropriately the tool bar

get_zoom_flag()
Get the zoom flag to update appropriately the tool bar

get_zoom_in_flag()
Get the zoom in flag to update appropriately the tool bar

get_zoom_out_flag()
Get the zoom out flag to update appropriately the tool bar

group_id = None

has_changed()

on_batch_selection(*event*)

Parameters **event** – contains parameter enable. When enable is set to True the application is in Batch mode otherwise the application is in Single mode.

on_bookmark(*event*)
The derivative class is on bookmark mode if implemented

on_close(*event*)
Close event. Hide the whole window.

on_copy(*event*)
The copy action if possible

on_drag(*event*)
The derivative class allows dragging motion if implemented

on_kill_focus(*event=None*)
The derivative class is on unfocus if implemented

on_paste(*event*)
The paste action if possible

on_preview(*event*)
Display a printable version of the class derivative

on_redo(*event*)
The previous action is restored if possible

on_reset(*event*)
The derivative class state is restored

on_save(*event*)
The state of the derivative class is restored

on_set_focus(*event=None*)
The derivative class is on focus if implemented

on_tap_focus()
Update menu on clicking the panel tap

on_undo (*event*)
The current action is canceled

on_zoom (*event*)
The derivative class is on zoom mode (using pane) if zoom mode is implemented

on_zoom_in (*event*)
The derivative class is on zoom in mode if implemented

on_zoom_out (*event*)
The derivative class is on zoom out mode if implemented

save_project (*doc=None*)
return an xml node containing state of the panel that guiframe can write to file

set_manager (*manager*)

uid = None

sas.sasgui.guiframe.pdfview module

class `sas.sasgui.guiframe.pdfview.`**PDFFrame** (*parent*, *id*, *title*, *path*)
Bases: `wx._windows.Frame`

Frame for PDF panel

class `sas.sasgui.guiframe.pdfview.`**PDFPanel** (*parent*, *path=None*)
Bases: `wx._windows.Panel`

Panel that contains the pdf reader

OnClose (*event*)
Close panel

OnLoad (*event=None*, *path=None*)
Load a pdf file

: Param path: full path to the file

OnNextPageButton (*event*)
Goes to Next page

OnOpenButton (*event*)
Open file button

OnPrevPageButton (*event*)
Goes to Previous page

class `sas.sasgui.guiframe.pdfview.`**TextFrame** (*parent*, *id*, *title*, *text*)
Bases: `wx._windows.Frame`

Frame for PDF panel

class `sas.sasgui.guiframe.pdfview.`**TextPanel** (*parent*, *text=None*)
Bases: `wx.lib.scrolledpanel.ScrolledPanel`

Panel that contains the text

OnClose (*event*)
Close panel

class `sas.sasgui.guiframe.pdfview.`**ViewApp** (*redirect=False*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*)
Bases: `wx._core.App`

OnInit ()

sas.sasgui.guiframe.plugin_base module

Defines the interface for a Plugin class that can be used by the gui_manager.

class sas.sasgui.guiframe.plugin_base.**PluginBase**(*name='Test_plugin'*)

Bases: object

This class defines the interface for a Plugin class that can be used by the gui_manager.

Plug-ins should be placed in a sub-directory called "perspectives". For example, a plug-in called Foo should be place in "perspectives/Foo". That directory contains at least two files:

1. perspectives/Foo/__init__.py contains two lines:

```
PLUGIN_ID = "Foo plug-in 1.0"
from Foo import *
```

2. perspectives/Foo/Foo.py contains the definition of the Plugin class for the Foo plug-in. The interface of that Plugin class should follow the interface of the class you are looking at.

See dummyapp.py for a plugin example.

add_color(*color, id*)
Adds color to a plugin

can_load_data()
if return True, then call handler to load data

clear_panel()
clear all related panels

delete_data(*data_id*)
Delete all references of data which id are in data_list.

get_batch_capable()
Check if the plugin has a batch capability

get_context_menu(*plotpanel=None*)
This method is optional.

When the context menu of a plot is rendered, the get_context_menu method will be called to give you a chance to add a menu item to the context menu.

A ref to a plotpanel object is passed so that you can investigate the plot content and decide whether you need to add items to the context menu.

This method returns a list of menu items. Each item is itself a list defining the text to appear in the menu, a tool-tip help text, and a call-back method.

Parameters **graph** – the Graph object to which we attach the context menu

Returns a list of menu items with call-back function

get_extensions()
return state reader and its extensions

get_frame()
Returns MDIChildFrame

get_panels(*parent*)
Create and return the list of wx.Panels for your plug-in. Define the plug-in perspective.

Panels should inherit from DefaultPanel defined below, or should present the same interface. They must define "window_caption" and "window_name".

Parameters **parent** – parent window

Returns list of panels

get_perspective()
Get the list of panel names for this perspective

get_tools()
Returns a set of menu entries for tools

is_always_active()
return True is this plugin is always active and it is local to guiframe even if the user is switching between perspectives

is_in_use(*data_id*)

get a data id a list of data name if data data is currently used by the plugin and the name of the plugin

data_name = 'None' in_use = False example [(data_name, self.sub_menu)]

load_data(*event*)
Load data

load_folder(*event*)
Load entire folder

on_batch_selection(*flag*)
need to be overwritten by the derivated class

on_perspective(*event=None*)
Call back function for the perspective menu item. We notify the parent window that the perspective has changed.

Parameters **event** – menu event

on_set_state_helper(*event*)
update state

populate_file_menu()
Append menu item under file menu item of the frame

populate_menu(*parent*)
Create and return the list of application menu items for the plug-in.

Parameters **parent** – parent window

Returns plug-in menu

post_init()
Post initialization call back to close the loose ends

set_batch_selection(*flag*)
the plugin to its batch state if flag is True

set_data(*data_list=None*)
receive a list of data and use it in the current perspective

set_is_active(*active=False*)
Set if the perspective is always active

set_state(*state=None*, *datainfo=None*)
update state

set_theory(*theory_list=None*)

Parameters **theory_list** – list of information related to available theory state

use_data()
return True if these plugin use data

sas.sasgui.guiframe.proxy module

class sas.sasgui.guiframe.proxy.**Connection**(*url*, *timeout*)
Bases: object

connect()
Performs the request and gets a response from self.url @return: response object from urllib2.urlopen

sas.sasgui.guiframe.proxy.**logger = <logging.Logger object>**
HTTP Proxy parser and Connection

connect() function:

- auto detects proxy in windows, osx
- in ux systems, the http_proxy environment variable must be set
- if it fails, try to find the proxy.pac address. - parses the file, and looks up for all possible proxies

sas.sasgui.guiframe.report_dialog module

Base class for reports. Child classes will need to implement the onSave() method.

class sas.sasgui.guiframe.report_dialog.**BaseReportDialog**(*report_list*, *imgRAM*, *fig_urls*, **args*, ***kwds*)
Bases: wx._windows.Dialog

HTML2PDF(*data*, *filename*)
Create a PDF file from html source string. Returns True is the file creation was successful. : data: html string : filename: name of file to be saved

onClose(*event=None*)
Close the Dialog : event: Close button event

onPreview(*event=None*)
Preview : event: Preview button event

onPrint(*event=None*)
Print : event: Print button event

sas.sasgui.guiframe.report_image_handler module

class sas.sasgui.guiframe.report_image_handler.**ReportImageHandler**

static check_for_empty_instance()

instance = None

static remove_figure()

static set_figs(*bitmaps*, *perspective*)

sas.sasgui.guiframe.startup_configuration module

sas.sasgui.guiframe.startup_configuration.**PANEL_HEIGHT = 215**
Dialog to set Appication startup configuration

class sas.sasgui.guiframe.startup_configuration.**StartupConfiguration**(*parent, gui, id=-1, title='Startup Setting'*)

Bases: wx._windows.Dialog

Dialog for Startup Configuration

OnCurrent(*event=None*)
Set to curent setup

OnDefault(*event=None*)
Set to default

write_custom_config()
Write custom configuration

sas.sasgui.guiframe.utils module

Contains common classes and functions

class sas.sasgui.guiframe.utils.**IdList**
Create a list of wx ids that can be reused.

Ids for items need to be unique within their context. In a dynamic application where the number of ids needed different each time the form is created, depending for example, on the number of items that need to be shown in the context menu, you cannot preallocate the ids that you are going to use for the form. Instead, you can use an IdList, which will reuse ids from context to context, adding new ones if the new context requires more than a previous context.

IdList is set up as an iterator, which returns new ids forever or until it runs out. This makes it pretty useful for defining menus:

```
class Form(wx.Dialog):
    _form_id_pool = IdList()
    def __init__(self):
        ...
        menu = wx.Menu()
        for item, wx_id in zip(menu_items, self._form_id_pool):
            name, description, callback = item
            menu.Append(wx_id, name, description)
            wx.EVT_MENU(self, wx_id, callback)
        ...
```

It is a little unusual to use an iterator outside of a loop, but it is supported. For example, when defining a form, your class definition might look something like:

```
class Form(wx.Dialog):
    _form_id_pool = IdList()
    def __init__(self, pairs, ...):
        ids = iter(_form_id_pool)
        ...
        wx.StaticText(self, ids.next(), "Some key-value pairs")
        for name, value in pairs:
            label = wx.StaticText(self, ids.next(), name)
            input = wx.TextCtrl(self, ids.next(), value=str(value))
            ...
        ...
```

If the dialog is really dynamic, and not defined all in one place, then save the id list iterator as *self._ids = iter(_form_id_pool)* in the constructor.

The wx documentation is not clear on whether ids need to be unique. Clearly different dialogs can use the same ids, as this is done for the standard button ids such as wx.ID_HELP. Presumably each widget on the form needs its own id, but whether these ids can match the ids of menu items is not indicated, or whether different submenus need their own ids. Using different id lists for menu items and widgets is safest, but probably not necessary. And what about notebook tabs. Do the ids need to be unique across all tabs?

class `sas.sasgui.guiframe.utils.`**`PanelMenu`**(**args*, ***kwargs*)
Bases: `wx._core.Menu`

`graph = None`

`plots = None`

`set_graph`(*graph*)

`set_plots`(*plots*)

`sas.sasgui.guiframe.utils.`**`check_float`**(*item*)

Parameters **`item`** – txtcrtl containing a value

`sas.sasgui.guiframe.utils.`**`check_int`**(*item*)

Parameters **`item`** – txtcrtl containing a value

`sas.sasgui.guiframe.utils.`**`format_number`**(*value*, *high=False*)
Return a float in a standardized, human-readable formatted string

`sas.sasgui.guiframe.utils.`**`look_for_tag`**(*string1*, *begin*, *end=None*)
this method remove the begin and end tags given by the user from the string .

Parameters

- **`begin`** – the initial tag
- **`end`** – the final tag
- **`string`** – the string to check

Returns begin_flag==True if begin was found, end_flag==if end was found else return false, false

`sas.sasgui.guiframe.utils.`**`parse_name`**(*name*, *expression*)
remove "_" in front of a name

`sas.sasgui.guiframe.utils.`**`split_list`**(*separator*, *mylist*, *n=0*)
returns a list of string without white space of separator

Parameters **`separator`** – the string to remove

`sas.sasgui.guiframe.utils.`**`split_text`**(*separator*, *string1*, *n=0*)
return a list of string without white space of separator

Parameters **`separator`** – the string to remove

Module contents

`sas.sasgui.guiframe.`**`data_files`**()
Return the data files associated with guiframe images .

The format is a list of (directory, [files...]) pairs which can be used directly in setup(...,data_files=...) for setup.py.

`sas.sasgui.guiframe.`**`get_data_path`**(*media*)

`sas.sasgui.guiframe.`**`get_media_path`**(*media*)

sas.sasgui.perspectives package

Subpackages

sas.sasgui.perspectives.calculator package

Submodules

sas.sasgui.perspectives.calculator.aperture_editor module

class `sas.sasgui.perspectives.calculator.aperture_editor.`**`ApertureDialog`**(*parent=None, manager=None, aperture=None, *args, **kwds*)

Bases: `wx._windows.Dialog`

`get_aperture`()
: return the current aperture

`get_notes`()
: return notes

`on_change_distance`()
: Change distance of the aperture

`on_change_name`()
: Change name

`on_change_size`()
: Change aperture size

`on_change_size_name`()
: Change the size’s name

`on_change_type`()
: Change aperture type

`on_click_apply`(*event*)
: Apply user values to the aperture

`on_click_cancel`(*event*)
: reset the current aperture to its initial values

`reset_aperture`()
: put the default value of the detector back to the current aperture

`set_manager`(*manager*)
: Set manager of this window

`set_values`()
: take the aperture values of the current data and display them through the panel

sas.sasgui.perspectives.calculator.calculator module

Calculator Module

class `sas.sasgui.perspectives.calculator.calculator.`**`Plugin`**

Bases: *`sas.sasgui.guiframe.plugin_base.PluginBase`*

This class defines the interface for a Plugin class for calculator perspective

get_python_panel (*filename=None*)
Get the python shell panel

Parameters filename – file name to open in editor

get_tools ()
Returns a set of menu entries for tools

on_calculate_dv (*event*)
Compute the mass density or molar voulme

on_calculate_kiessig (*event*)
Compute the Kiessig thickness

on_calculate_resoltuion (*event*)
Estimate the instrumental resolution

on_calculate_sld (*event*)
Compute the scattering length density of molecula

on_calculate_slit_size (*event*)
Compute the slit size a given data

on_data_operation (*event*)
Data operation

on_edit_data (*event*)
Edit meta data

on_gen_model (*event*)
On Generic model menu event

on_image_viewer (*event*)
Get choose an image file dialog

Parameters event – menu event

on_python_console (*event*)
Open Python Console

Parameters event – menu event

on_show_orientation (*event*)
Make sasmodels orientation & jitter viewer available

put_icon (*frame*)
Put icon in the frame title bar

sas.sasgui.perspectives.calculator.calculator_widgets module

This software was developed by the University of Tennessee as part of the Distributed Data Analysis of Neutron Scattering Experiments (DANSE) project funded by the US National Science Foundation.

See the license text in license.txt

copyright 2009, University of Tennessee

class sas.sasgui.perspectives.calculator.calculator_widgets.**InputTextCtrl** (*parent=None*, **args*, ***kwds*)

Bases: wx._controls.TextCtrl

Text control for model and fit parameters. Binds the appropriate events for user interactions.

class sas.sasgui.perspectives.calculator.calculator_widgets.**InterActiveOutputTextCtrl** (**a*
****

Bases: wx._controls.TextCtrl

Text control used to display outputs. No editing allowed. The background is grayed out. User can't select text.

class sas.sasgui.perspectives.calculator.calculator_widgets.**OutputTextCtrl**(**args*, ***kwds*)

Bases: *sas.sasgui.perspectives.calculator.calculator_widgets.InterActiveOutputTextCtrl*

Text control used to display outputs. No editing allowed. The background is grayed out. User can't select text.

sas.sasgui.perspectives.calculator.collimation_editor module

class sas.sasgui.perspectives.calculator.collimation_editor.**CollimationDialog**(*parent=None*, *manager=None*, *collimation=[]*, **args*, ***kwds*)

Bases: wx._windows.Dialog

add_aperture(*event*)
: Append empty aperture to data's list of aperture

add_collimation(*event*)
: Append empty collimation to data's list of collimation

edit_aperture(*event*)
: Edit the selected aperture

enable_aperture()
: Enable /disable widgets crelated to aperture

enable_collimation()
: Enable /disable widgets related to collimation

fill_aperture_combox()
: fill the current combobox with the available aperture

fill_collimation_combox()
: fill the current combobox with the available collimation

get_collimation()
: return the current collimation

get_current_collimation()

get_notes()
: return notes

on_change_length()
: Change the length

on_change_name()
: Change name

on_click_apply(*event*)
: Apply user values to the collimation

on_click_cancel(*event*)
: leave the collimation as it is and close

on_select_collimation(*event*)
fill the control on the panel according to the current selected collimation

remove_aperture(*event*)
Remove aperture to data's list of aperture

remove_collimation(*event*)
Remove collimation to data's list of collimation

reset_aperture_combobox(*edited_aperture*)
take all edited editor and reset clientdata of aperture combo box

reset_collimation_combobox(*edited_collimation*)
take all edited editor and reset clientdata of collimation combo box

set_aperture(*aperture*)
set aperture for data

set_manager(*manager*)
Set manager of this window

set_values()
take the collimation values of the current data and display them through the panel

sas.sasgui.perspectives.calculator.console module

Console Module display message of a dialog

class sas.sasgui.perspectives.calculator.console.**ConsoleDialog**(*parent=None, manager=None, data=None, title='Data Summary', size=(530, 560)*)

Bases: `wx._windows.Dialog`

Data summary dialog

set_manager(*manager*)
Set the manager of this window

set_message(*msg=''*)
Display the message received

sas.sasgui.perspectives.calculator.data_editor module

class sas.sasgui.perspectives.calculator.data_editor.**DataEditorPanel**(*parent, data=[], *args, **kwds*)

Bases: `wx._windows.ScrolledWindow`

Parameters **data** – when not empty the class can same information into a dat object and post event containing the changed data object to some other frame

choose_data_file(*location=None*)
Open a file dialog to allow loading a file

complete_loading(*data=None, filename=''*)
Complete the loading and compute the slit size

edit_collimation()
Edit the selected collimation

edit_detector()
Edit the selected detector

edit_sample()
Open the dialog to edit the sample of the current data

edit_source()
Open the dialog to edit the saource of the current data

enable_data_cbox()

fill_data_combox()
fill the current combobox with the available data

get_current_data()

get_data()
return the current data

get_notes()
return notes

on_change_run(*event=None*)
Change run

on_change_title(*event=None*)
Change title

on_click_apply(*event*)
changes are saved in data object imported to edit

on_click_browse(*event*)
Open a file dialog to allow the user to select a given file. Display the loaded data if available.

on_click_reset(*event*)

on_click_save(*event*)
Save change into a file

on_click_view(*event*)
Display data info

on_close(*event*)
leave data as it is and close

on_edit(*event*)

on_select_data(*event=None*)

reset_panel()

reset_radiobox()

set_collimation(*collimation*, *notes=None*)
set collimation for data

set_detector(*detector*, *notes=None*)
set detector for data

set_sample(*sample*, *notes=None*)
set sample for data

set_source(*source*, *notes=None*)
set source for data

set_values()
take the aperture values of the current data and display them through the panel

class sas.sasgui.perspectives.calculator.data_editor.**DataEditorWindow**(*parent*, *manager*, *data=None*, **args*, ***kwds*)

Bases: `wx._windows.Frame`

get_data()
return the current data

sas.sasgui.perspectives.calculator.data_editor.**load_error**(*error=None*)
Pop up an error message.

@param error: details error message to be displayed

sas.sasgui.perspectives.calculator.data_operator module

GUI for the data operations panel (sum and multiply)

class sas.sasgui.perspectives.calculator.data_operator.**DataOperPanel**(*parent*, **args*, ***kwds*)

Bases: `wx._windows.ScrolledWindow`

check_data_inputs()
Check data1 and data2 whether or not they are ready for operation

disconnect_panels()

draw_output(*output*)
Draw output data(temp)

fill_data_combox()
fill the current combobox with the available data

fill_oprator_combox()
fill the current combobox with the operator

get_datalist()

make_data_out(*data1*, *data2*)
Make a temp. data output set

on_click_apply(*event*)
changes are saved in data object imported to edit

on_close(*event*)
leave data as it is and close

on_help(*event*)

Bring up the Data Operations Panel Documentation whenever the HELP button is clicked.

Calls DocumentationWindow with the path of the location within the documentation tree (after /doc/ ....". Note that when using old versions of Wx (before 2.9) and thus not the release version of installers, the help comes up at the top level of the file as webbrowser does not pass anything past the # to the browser when it is running "file:///...."

Parameters **evt** – Triggers on clicking the help button

on_name(*event=None*)
On data name typing

on_number(*event=None*, *control=None*)
On selecting Number for Data2

on_select_data1 (*event=None*)
On select data1

on_select_data2 (*event=None*)
On Selecting Data2

on_select_operator (*event=None*)
On Select an Operator

put_text_pic (*pic=None, content="*)
Put text to the pic

send_warnings (*msg=", info='info'*)
Send warning to status bar

set_panel_on_focus (*event*)
On Focus at this window

set_plot_unfocus ()
Unfocus on right click

class sas.sasgui.perspectives.calculator.data_operator.**DataOperatorWindow** (*parent, manager, *args, **kwds*)

Bases: wx._windows.Frame

OnClose (*event=None*)
On close event

class sas.sasgui.perspectives.calculator.data_operator.**SmallPanel** (*parent, id=-1, is_number=False, content='?', **kwargs*)

Bases: *sas.sasgui.plottools.PlotPanel.PlotPanel*

PlotPanel for Quick plot and masking plot

add_image (*plot*)
Add Image

add_text ()
Text in the plot

add_toolbar ()
Add toolbar

draw ()
Draw

erase_legend ()
Remove Legend

onContextMenu (*event*)
Default context menu for a plot panel

onLeftDown (*event*)
Disables LeftDown

onMouseMotion (*event*)
Disable dragging 2D image

onPick (*event*)
Remove Legend

onWheel (*event*)

on_set_focus (*event*)
send to the parenet the current panel on focus

ontogglescale (*event*)
On toggle 2d scale

set_content (*content=''*)
Set text content

sas.sasgui.perspectives.calculator.density_panel module

This module provide GUI for the mass density calculator

class sas.sasgui.perspectives.calculator.density_panel.**DensityPanel** (*parent*, *base=None*, **args*, ***kwds*)

Bases: wx.lib.scrolledpanel.ScrolledPanel, *sas.sasgui.guiframe.panel_base.PanelBase*

Provides the mass density calculator GUI.

CENTER_PANE = True

calculate (*event*)
Calculate the mass Density/molar Volume of the molecules

check_inputs ()
Check validity user inputs

clear_outputs ()
Clear the outputs textctrl

get_input ()
Return the current input and output combobox values

on_close (*event*)
close the window containing this panel

on_help (*event*)

Bring up the density/volume calculator Documentation whenever the HELP button is clicked.

Calls DocumentationWindow with the path of the location within the documentation tree (after /doc/ ….". Note that when using old versions of Wx (before 2.9) and thus not the release version of installers, the help comes up at the top level of the file as webbrowser does not pass anything past the # to the browser when it is running "file:///…."

Parameters **evt** – Triggers on clicking the help button

on_select_input (*event*)
On selection of input combobox, update units and output combobox

on_select_output (*event*)
On selection of output combobox, update units and input combobox

set_values ()
Sets units and combobox values

window_caption = 'Mass Density Calculator'

window_name = 'Mass Density Calculator'

class sas.sasgui.perspectives.calculator.density_panel.**DensityWindow**(*parent=None, title='Density/Volume Calculator', base=None, manager=None, size=(483.0, 296.7741935483871), *args, **kwds*)

Bases: wx._windows.Frame

on_close(*event*)
On close event

class sas.sasgui.perspectives.calculator.density_panel.**ViewApp**(*redirect=False, filename=None, useBestVisual=False, clearSigInt=True*)

Bases: wx._core.App

OnInit()

sas.sasgui.perspectives.calculator.detector_editor module

class sas.sasgui.perspectives.calculator.detector_editor.**DetectorDialog**(*parent=None, manager=None, detector=None, title='Detector Editor', size=(550, 480)*)

Bases: wx._windows.Dialog

add_detector(*event*)
Append empty detector to data's list of detector

enable_detector()
Enable /disable widgets crelated to detector

fill_detector_combox()
fill the current combobox with the available detector

get_current_detector()

get_detector()
return the current detector

get_notes()
return notes

on_change_beam_center()
Change the detector beam center

on_change_distance()
Change distance of the sample to the detector

on_change_instrument()
Change instrument

on_change_offset()
Change the detector offset

on_change_orientation()
Change the detector orientation

on_change_pixel_size()
Change the detector pixel size

on_change_slit_length()
Change slit length of the detector

on_click_apply(*event*)
Apply user values to the detector

on_click_cancel(*event*)
reset the current detector to its initial values

remove_detector(*event*)
Remove detector to data's list of detector

reset_detector()
put the default value of the detector back to the current detector

reset_detector_combobox(*edited_detector*)
take all edited editor and reset clientdata of detector combo box

set_detector(*detector*)
set detector for data

set_manager(*manager*)
Set manager of this window

set_values()
take the detector values of the current data and display them through the panel

sas.sasgui.perspectives.calculator.gen_scatter_panel module

Generic Scattering panel. This module relies on guiframe manager.

class sas.sasgui.perspectives.calculator.gen_scatter_panel.**CalcGen**(*id=-1*, *input=None*, *completefn=None*, *updatefn=None*, *yieldtime=0.01*, *worktime=0.01*)

Bases: `sas.sascalc.data_util.calcthread.CalcThread`

Computation

compute()
executing computation

class sas.sasgui.perspectives.calculator.gen_scatter_panel.**OmfPanel**(*parent*, **args*, ***kwds*)
Bases: wx.lib.scrolledpanel.ScrolledPanel, *sas.sasgui.guiframe.panel_base.PanelBase*

Provides the sas gen calculator GUI.

check_inputs()
check if the inputs are valid

display_npts(*nop*)
Displays Npts ctrl

get_pix_volumes()
Get the pixel volume

get_sld_val()
Set sld_n of slddata on sld input

on_save(*event*)
Close the window containing this panel

on_sld_draw(*event*)
Draw sld profile as scattered plot

set_npts_from_slddata()
Set total n. of points form the sld data

set_sld_ctr(*sld_data*)
Set sld textctrls

set_slddata(*slddata*)
Set sld data related items

window_caption = 'SLD Pixel Info '

window_name = 'SLD Pixel Info'

class sas.sasgui.perspectives.calculator.gen_scatter_panel.**SasGenPanel**(*parent*, **args*, ***kwds*)
Bases: wx.lib.scrolledpanel.ScrolledPanel, *sas.sasgui.guiframe.panel_base.PanelBase*

Provides the sas gen calculator GUI.

choose_data_file(*location=None*)
Choosing a dtata file

complete(*input*, *update=None*)
Gen compute complete function :Param input: input list [qx_data, qy_data, i_out]

complete_loading(*data=None*, *filename=''*)
Complete the loading

estimate_ctime()
Calculation time estimation

load_update()
print update on the status bar

on_compute(*event*)
Compute I(qx, qy)

on_help(*event*)

Bring up the General scattering Calculator Documentation whenever the HELP button is clicked.

Calls DocumentationWindow with the path of the location within the documentation tree (after /doc/ ….". Note that when using old versions of Wx (before 2.9) and thus not the release version of installers, the help comes up at the top level of the file as webbrowser does not pass anything past the # to the browser when it is running "file:///…."

Parameters **`evt`** – Triggers on clicking the help button

`on_load_data`(*event*)
Open a file dialog to allow the user to select a given file. The user is only allow to load file with extension .omf, .txt, .sld. Display the slit size corresponding to the loaded data.

`on_panel_close`(*event*)
close the window containing this panel

`set_est_time`()
Set text for est. computation time

`set_input_params`()
Set model parameters

`set_scale2d`(*scale*)
Set SLD plot scale

`set_volume_ctl_val`(*val*)
Set volume txtctrl value

`sld_draw`(*event=None*, *has_arrow=True*)
Draw 3D sld profile

`window_caption = 'Generic SAS '`

`window_name = 'Generic SAS Calculator'`

class `sas.sasgui.perspectives.calculator.gen_scatter_panel.`**`SasGenWindow`**(*parent=None*, *manager=None*, *title='Generic Scattering Calculator'*, *size=(868.0, 610.5)*, **args*, ***kwds*)

Bases: `wx._windows.Frame`

GEN SAS main window

`build_panels`()

`check_omfpanel_inputs`()
Check OMF panel inputs

`draw_graph`(*plot*, *title=''*)

`get_npix`()
Get no. of pixels from omf panel

get_path()
File location

get_pix_volumes()
Get a pixel volume

get_sld_data()
Return slddata

get_sld_data_from_omf()

get_sld_from_omf()

on_close(*event*)
Close

on_open_file(*event*)
On Open

on_panel_close(*event*)

on_save_file(*event*)
On Close

set_etime()
Sets est. computation time on panel

set_file_location(*path*)
File location

set_main_panel_sld_data(*sld_data*)

set_omfpanel_default_shap(*shape*)
Set default_shape in omfpanel

set_omfpanel_npts()
Set Npts in omf panel

set_scale2d(*scale*)

set_schedule_full_draw(*panel=None*, *func='del'*)
Send full draw to gui frame

set_sld_data(*data*)
Set omfdata

set_sld_n(*sld*)

set_volume_ctr_val(*val*)
Set volume txtctl value

sld_draw()
sld draw

sas.sasgui.perspectives.calculator.gen_scatter_panel.**add_icon**(*parent*, *frame*)
Add icon in the frame

sas.sasgui.perspectives.calculator.image_viewer module

class sas.sasgui.perspectives.calculator.image_viewer.**ImageFrame**(*parent*, *id*, *title*, *image=None*, *scale='log_{10}'*, *size=wx.Size(550, 470)*)
Bases: *sas.sasgui.guiframe.local_perspectives.plotting.SimplePlot.*

`PlotFrame`

Frame for simple plot

on_help(*event*)
Bring up Image Viewer Documentation from the image viewer window whenever the help menu item "how to" is clicked. Calls DocumentationWindow with the path of the location within the documentation tree (after /doc/ ....".

Parameters evt – Triggers on clicking "how to" in help menu

on_set_data(*event*)
Rescale the x y range, make 2D data and send it to data explore

class sas.sasgui.perspectives.calculator.image_viewer.**ImageView**(*parent=None*)
Open a file dialog to allow the user to select a given file. Display the loaded data if available.

choose_data_file(*location=None*)
Open a file dialog to allow loading a file

load()
load image files

class sas.sasgui.perspectives.calculator.image_viewer.**SetDialog**(*parent, id=-1, title='Convert to Data', image=None, size=(480, 270)*)

Bases: `wx._windows.Dialog`

Dialog for Data Set

OnClose(*event*)
Close event

convert_image(*rgb, xmin, xmax, ymin, ymax, zscale*)
Convert image to data2D

on_set(*event*)
Set image as data

rgb2gray(*rgb*)
RGB to Grey

sas.sasgui.perspectives.calculator.kiessig_calculator_panel module

This software was developed by the University of Tennessee as part of the Distributed Data Analysis of Neutron Scattering Experiments (DANSE) project funded by the US National Science Foundation.

See the license text in license.txt

copyright 2008, 2009, University of Tennessee

class sas.sasgui.perspectives.calculator.kiessig_calculator_panel.**KiessigThicknessCalcul**

Bases: `wx._windows.Panel`, `sas.sasgui.guiframe.panel_base.PanelBase`

Provides the Kiessig thickness calculator GUI.

CENTER_PANE = True

format_number(*value=None*)
Return a float in a standardized, human-readable formatted string

on_close (*event*)
close the window containing this panel

on_compute (*event*)
Execute the computation of thickness

on_help (*event*)

Bring up the Kiessig fringe calculator Documentation whenever the HELP button is clicked. Calls DocumentationWindow with the path of the location within the documentation tree (after /doc/ ….". Note that when using old versions of Wx (before 2.9) and thus not the release version of installers, the help comes up at the top level of the file as webbrowser does not pass anything past the # to the browser when it is running "file:///…."

Parameters **evt** – Triggers on clicking the help button

window_caption = 'Kiessig Thickness Calculator'

window_name = 'Kiessig Thickness Calculator'

class sas.sasgui.perspectives.calculator.kiessig_calculator_panel.**KiessigWindow** (*parent=No manager=None title='Kiessi Thickness Calculator', size=(560, 230), *args, **kwds*)

Bases: `wx._windows.Frame`

on_close (*event*)
Close event

sas.sasgui.perspectives.calculator.load_thread module

Thread handler used to load data

class sas.sasgui.perspectives.calculator.load_thread.**DataReader** (*path, completefn=None, updatefn=None, yieldtime=0.01, worktime=0.01*)

Bases: *sas.sascalc.data_util.calcthread.CalcThread*

Load a data given a filename

compute ()
read some data

isquit ()
@raise KeyboardInterrupt: when the thread is interrupted

class sas.sasgui.perspectives.calculator.load_thread.**GenReader**(*path*, *loader*, *completefn=None*, *updatefn=None*, *yieldtime=0.01*, *worktime=0.01*)

Bases: *sas.sascalc.data_util.calcthread.CalcThread*

Load a sld data given a filename

compute()
read some data

isquit()
@raise KeyboardInterrupt: when the thread is interrupted

sas.sasgui.perspectives.calculator.model_editor module

This module provides three model editor classes: the composite model editor, the easy editor which provides a simple interface with tooltip help to enter the parameters of the model and their default value and a panel to input a function of y (usually the intensity). It also provides a drop down of standard available math functions. Finally a full python editor panel for complete customization is provided.

:TODO the writing of the file and name checking (and maybe some other functions?) should be moved to a computational module which could be called from a python script. Basically one just needs to pass the name, description text and function text (or in the case of the composite editor the names of the first and second model and the operator to be used).

class sas.sasgui.perspectives.calculator.model_editor.**EditorPanel**(*parent*, *base*, *path*, *title*, **args*, ***kwds*)

Bases: wx._windows.ScrolledWindow

Simple Plugin Model function editor

check_name()
Check name if exist already

get_notes()
return notes

get_param_helper(*param_str*)
yield a sequence of name, value pairs for the parameters in param_str

Parameters can be defined by one per line by name=value, or multiple on the same line by separating the pairs by semicolon or comma. The value is optional and defaults to "1.0".

get_warning()
Get the warning msg

on_change_name(*event=None*)
Change name

on_click_apply(*event*)
Changes are saved in data object imported to edit.

checks firs for valid name, then if it already exists then checks that a function was entered and finally that if entered it contains at least a return statement. If all passes writes file then tries to compile. If compile fails or import module fails or run method fails tries to remove any .py and pyc files that may have been created and sets error message.

:todo this code still could do with a careful going over to clean up and simplify. the non GUI methods such as this one should be removed to computational code of SasView. Most of those computational methods would be the same for both the simple editors.

on_close(*event*)
leave data as it is and close

on_help(*event*)
Bring up the New Plugin Model Editor Documentation whenever the HELP button is clicked.

Calls DocumentationWindow with the path of the location within the documentation tree (after /doc/ ….". Note that when using old versions of Wx (before 2.9) and thus not the release version of installers, the help comes up at the top level of the file as webbrowser does not pass anything past the # to the browser when it is running "file:///…."

Parameters **evt** – Triggers on clicking the help button

on_over_cb(*event*)
Set overwrite name flag on cb event

set_function_helper(*line*)
Get string in line to define the local params

Parameters **line** – one line of string got from the param_str

write_file(*fname*, *name*, *desc_str*, *param_str*, *func_str*)
Write content in file

Parameters

- **fname** – full file path
- **desc_str** – content of the description strings
- **param_str** – content of params; Strings
- **func_str** – content of func; Strings

class sas.sasgui.perspectives.calculator.model_editor.**EditorWindow**(*parent*, *base*, *path*, *title*, *size=(800, 735)*, **args*, ***kwds*)

Bases: `wx._windows.Frame`

Editor Window

on_close(*event*)
On close event

class sas.sasgui.perspectives.calculator.model_editor.**TextDialog**(*parent=None*, *base=None*, *id=None*, *title="*, *model_list=[]*, *plugin_dir=None*)

Bases: `wx._windows.Dialog`

Dialog for easy custom composite models. Provides a wx.Dialog panel to choose two existing models (including pre-existing Plugin Models which may themselves be composite models) as well as an operation on those models (add or multiply) the resulting model will add a scale parameter for summed models and a background parameter for a multiplied model.

The user also gives a brief help for the model in a description box and must provide a unique name which is verified as unique before the new model is saved.

This Dialog pops up for the user when they press 'Sum|Multi(p1,p2)' under 'Plugin Model Operations' under 'Fitting' menu. This is currently called as a Modal Dialog.

:TODO the built in compiler currently balks at when it tries to import a model whose name contains spaces or symbols (such as + … underscore should be fine). Have fixed so the editor cannot save such a file name but if a file is dropped in the plugin directory from outside this class will create a file that cannot be compiled. Should add the check to the write method or to the on_modelx method.

- PDB:April 5, 2015

`check_name`(*event=None*)
Check that proposed new model name is a valid Python module name and that it does not already exist. If not show error message and pink background in text box else call on_apply

:TODO this should be separated out from the GUI code. For that we need to pass it the name (or if we want to keep the default name option also need to pass the self._operator attribute) We just need the function to return an error code that the name is good or if not why (not a valid name, name exists already). The rest of the error handling should be done in this module. so on_apply would then start by checking the name and then either raise errors or do the deed.

`compile_file`(*path*)
Compile the file in the path

`delete_file`(*path*)
Delete file in the path

`fill_explanation_helpstring`(*operator*)
Choose the equation to use depending on whether we now have a sum or multiply model then create the appropriate string

for the sum model the result will be: scale_factor * (scale1 * model_1 + scale2 * model_2) + background while for the multiply model it will just be: scale_factor * (model_1* model_2) + background

`fill_operator_combox`()
fill the current combobox with the operator

`get_textnames`()
Returns model name string as list

`on_apply`(*path*)
This method is a misnomer - it is not bound to the apply button event. Instead the apply button event goes to check_name which then calls this method if the name of the new file is acceptable.

:TODO this should be bound to the apply button. The first line should call the check_name method which itself should be in another module separated from the the GUI modules.

`on_change_name`(*event=None*)
Change name

`on_help`(*event*)

Bring up the Composite Model Editor Documentation whenever the HELP button is clicked.

Calls DocumentationWindow with the path of the location within the documentation tree (after /doc/ ….". Note that when using old versions of Wx (before 2.9) and thus not the release version of installers, the help comes up at the top level of the file as webbrowser does not pass anything past the # to the browser when it is running "file:///…."

Parameters **`evt`** – Triggers on clicking the help button

on_model1 (*event*)
Set model1

on_model2 (*event*)
Set model2

on_select_operator (*event=None*)
On Select an Operator

update_cm_list ()
Update custom model list

write_string (*fname*, *model1_name*, *model2_name*)
Write and Save file

sas.sasgui.perspectives.calculator.pyconsole module

Console Module display Python console

class sas.sasgui.perspectives.calculator.pyconsole.**PyConsole** (*parent=None*, *base=None*, *manager=None*, *panel=None*, *title='Python Shell/Editor'*, *filename=None*, *size=(830, 730)*)

Bases: `wx.py.editor.EditorNotebookFrame`

CENTER_PANE = False

OnAbout (*event*)
On About

OnCheckModel (*event*)
Compile

OnHelp (*event*)
Show a help dialog.

OnNewFile (*event*)
OnFileOpen

OnOpenFile (*event*)
OnFileOpen

OnRun (*event*)
Run

OnSaveAsFile (*event*)
OnFileSaveAs overwrite

OnSaveFile (*event*)
OnFileSave overwrite

OnUpdateCompileMenu (*event*)
Update Compile menu items based on current tap.

bufferOpen ()
Open file in buffer, bypassing editor bufferOpen

bufferSaveAs ()
Save buffer to a new filename: Bypassing editor bufferSaveAs

on_close(*event*)
Close event

set_manager(*manager*)
Set the manager of this window

window_caption = 'Plugin Model Editor'

window_name = 'Custom Model Editor'

class sas.sasgui.perspectives.calculator.pyconsole.**ResizableScrolledMessageDialog**(*parent, msg, caption, pos=wx.*
1, - 1), size=(50 300), style=53

Bases: `wx._windows.Dialog`

Custom version of wx ScrolledMessageDialog, allowing border resize

sas.sasgui.perspectives.calculator.pyconsole.**check_model**(*path*)
Check that the model on the path can run.

sas.sasgui.perspectives.calculator.pyconsole.**show_model_output**(*parent, fname*)

sas.sasgui.perspectives.calculator.resolcal_thread module

Thread for Resolution computation

class sas.sasgui.perspectives.calculator.resolcal_thread.**CalcRes**(*id=-1, func=None, qx=None, qy=None, qx_min=None, qx_max=None, qy_min=None, qy_max=None, image=None, completefn=None, updatefn=None, elapsed=0, yieldtime=0.01, worktime=0.01*)

Bases: *sas.sascalc.data_util.calcthread.CalcThread*

Compute Resolution

compute()
executing computation

sas.sasgui.perspectives.calculator.resolution_calculator_panel module

This software was developed by the University of Tennessee as part of the Distributed Data Analysis of Neutron Scattering Experiments (DANSE) project funded by the US National Science Foundation.

See the license text in license.txt

copyright 2008, 2009, 2010 University of Tennessee

class `sas.sasgui.perspectives.calculator.resolution_calculator_panel.`**`ResolutionCalculato`**

Bases: `wx.lib.scrolledpanel.ScrolledPanel`

Provides the Resolution calculator GUI.

`CENTER_PANE = True`

`complete`(*image*, *elapsed=None*)
Callafter complete: wx call after needed for stable output

`complete_cal`(*image*, *elapsed=None*)
Complete computation

`format_number`(*value=None*)
Return a float in a standardized, human-readable formatted string

`on_close`(*event*)
close the window containing this panel

`on_compute`(*event=None*)
Execute the computation of resolution

`on_compute_call`(*event=None*)
Execute the computation of resolution

`on_help`(*event*)

Bring up the Resolution calculator Documentation whenever the HELP button is clicked.

Calls DocumentationWindow with the path of the location within the documentation tree (after /doc/ ….". Note that when using old versions of Wx (before 2.9) and thus not the release version of installers, the help comes up at the top level of the file as webbrowser does not pass anything past the # to the browser when it is running "file:///…."

Parameters **`evt`** – Triggers on clicking the help button

`on_reset`(*event*)
Execute the reset

`window_caption = ''`

`window_name = 'Q Resolution Estimator'`

class sas.sasgui.perspectives.calculator.resolution_calculator_panel.**ResolutionWindow**(*pa
ma
ag
ti-
tle
Re
o-
lu-
tio
Es
ti-
ma
tor
siz
66
*a
***

Bases: `wx._windows.Frame`

Resolution Window

OnClose(*event*)
: On close event

sas.sasgui.perspectives.calculator.sample_editor module

class sas.sasgui.perspectives.calculator.sample_editor.**SampleDialog**(*parent=None, manager=None, sample=None, size=(550, 430), title='Sample Editor'*)

Bases: `wx._windows.Dialog`

get_notes()
: return notes

get_sample()
: return the current sample

on_change_details()
: Change details

on_change_id()
: Change id of the sample

on_change_name()
: Change name

on_change_orientation()
: Change orientation

on_change_position()
: Change position

on_change_temperature()
: Change temperature

on_change_thickness()
Change thickness

on_change_transmission()
Change transmission

on_click_apply(*event*)
Apply user values to the sample

on_click_cancel(*event*)
leave the sample as it is and close

reset_sample()
Put initial values of the sample back to the current sample

set_details(*sample*)
print details on the current sample

set_manager(*manager*)
Set manager of this window

set_values()
take the sample values of the current data and display them through the panel

sas.sasgui.perspectives.calculator.sld_panel module

This module provide GUI for the neutron scattering length density calculator

class sas.sasgui.perspectives.calculator.sld_panel.**SldPanel**(*parent*, *base=None*, **args*, ***kwds*)
Bases: wx._windows.Panel, *sas.sasgui.guiframe.panel_base.PanelBase*

Provides the SLD calculator GUI.

CENTER_PANE = True

calculateSld(*event*)
Calculate the neutron scattering density length of a molecule

calculate_sld_helper(*element*, *density*, *molecule_formula*)
Get an element and compute the corresponding SLD for a given formula

Parameters **element** – elements a string of existing atom

calculate_xray_sld(*element*)
Get an element and compute the corresponding SLD for a given formula

Parameters **element** – elements a string of existing atom

check_inputs()
Check validity user inputs

clear_outputs()
Clear the outputs textctrl

fill_xray_cbox()
fill the x-ray combobox with the sources

on_close(*event*)
close the window containing this panel

on_help(*event*)

Bring up the SLD Documentation whenever the HELP button is clicked.

Calls DocumentationWindow with the path of the location within the documentation tree (after /doc/ ....". Note that when using old versions of Wx (before 2.9) and thus not the release version of installers, the help comes up at the top level of the file as webbrowser does not pass anything past the # to the browser when it is running "file:///...."

Parameters **evt** – Triggers on clicking the help button

on_select_xray(*event=None*)
On Selecting a source

window_caption = 'SLD Calculator'

window_name = 'SLD Calculator'

class sas.sasgui.perspectives.calculator.sld_panel.**SldWindow**(*parent=None, title='SLD Calculator', base=None, manager=None, size=(410, 410), *args, **kwds*)

Bases: wx._windows.Frame

on_close(*event*)
On close event

class sas.sasgui.perspectives.calculator.sld_panel.**ViewApp**(*redirect=False, filename=None, useBestVisual=False, clearSigInt=True*)

Bases: wx._core.App

OnInit()

sas.sasgui.perspectives.calculator.slit_length_calculator_panel module

This software was developed by the University of Tennessee as part of the Distributed Data Analysis of Neutron Scattering Experiments (DANSE) project funded by the US National Science Foundation.

See the license text in license.txt

copyright 2008, 2009, University of Tennessee

class sas.sasgui.perspectives.calculator.slit_length_calculator_panel.**SlitLengthCalculat**

Bases: wx._windows.Panel, *sas.sasgui.guiframe.panel_base.PanelBase*

Provides the slit length calculator GUI.

CENTER_PANE = True

choose_data_file(*location=None*)

complete_loading(*data=None, filename=''*)
Complete the loading and compute the slit size

load_update()
print update on the status bar

on_close(*event*)
close the window containing this panel

on_help(*event*)

Bring up the slit length calculator Documentation whenever the HELP button is clicked.

Calls DocumentationWindow with the path of the location within the documentation tree (after /doc/ ....". Note that when using old versions of Wx (before 2.9) and thus not the release version of installers, the help comes up at the top level of the file as webbrowser does not pass anything past the # to the browser when it is running "file:///...."

Parameters **`evt`** – Triggers on clicking the help button

`on_load_data`(*event*)
Open a file dialog to allow the user to select a given file. The user is only allow to load file with extension .DAT or .dat. Display the slit size corresponding to the loaded data.

`window_caption = 'Slit Size Calculator'`

`window_name = 'Slit Size Calculator'`

`class` `sas.sasgui.perspectives.calculator.slit_length_calculator_panel.`**`SlitLengthCalculat`**

Bases: `wx._windows.Frame`

`on_close`(*event*)
Close event

sas.sasgui.perspectives.calculator.source_editor module

`class` `sas.sasgui.perspectives.calculator.source_editor.`**`SourceDialog`**(*parent=None, manager=None, source=None, *args, **kwds*)

Bases: `wx._windows.Dialog`

`get_notes`()
return notes

`get_source`()
return the current source

`on_change_beam_shape`()
Change beams shape

`on_change_beam_size`()
Change beam size

`on_change_beam_size_name`()
Change beam size name

`on_change_name`()
Change name

on_change_radiation()
Change radiation of the sample

on_change_wavelength()
Change the wavelength

on_change_wavelength_max()
Change the wavelength maximum

on_change_wavelength_min()
Change the wavelength minimum

on_change_wavelength_spread()
Change the wavelength spread

on_click_apply(*event*)
Apply user values to the source

on_click_cancel(*event*)
reset the current source

reset_source()
put back initial values of the source

set_manager(*manager*)
Set manager of this window

set_values()
take the source values of the current data and display them through the panel

Module contents

sas.sasgui.perspectives.calculator.**data_files**()
Return the data files associated with media calculator.

The format is a list of (directory, [files…]) pairs which can be used directly in setup(…,data_files=…) for setup.py.

sas.sasgui.perspectives.calculator.**get_data_path**(*media*)

sas.sasgui.perspectives.corfunc package

Submodules

sas.sasgui.perspectives.corfunc.corfunc module

Corfunc perspective

class sas.sasgui.perspectives.corfunc.corfunc.**Plugin**
Bases: *sas.sasgui.guiframe.plugin_base.PluginBase*

This class defines the interface for a plugin class for a correlation function perspective

clear_data()

delete_data(*data*)
Delete the data from the perspective

get_context_menu(*plotpanel=None*)
Get the context menu items available for Corfunc.

Parameters **plotpanel** – A Plotter1D panel

Returns a list of menu items with call-back function

Note if Data1D was generated from Theory1D the fitting option is not allowed

`get_panels`(*parent*)
Define the GUI panels

`set_data`(*data_list=None*)
Load the data that's been selected

Parameters **`data_list`** – The data to load in

`set_state`(*state=None*, *datainfo=None*)
Callback for CorfuncState reader. Called when a .crf file is loaded

`show_data`(*data*, *label*, *reset=False*, *active_ctrl=None*)
Show data read from a file

Parameters

- **`data`** – The data to plot (Data1D)
- **`label`** – What to label the plot. Also used as the plot ID
- **`reset`** – If True, all other plottables will be cleared

sas.sasgui.perspectives.corfunc.corfunc_panel module

class `sas.sasgui.perspectives.corfunc.corfunc_panel.`**`CorfuncPanel`**(*parent*, *data=None*, *manager=None*, **args*, ***kwds*)

Bases: `wx.lib.scrolledpanel.ScrolledPanel`, *`sas.sasgui.guiframe.panel_base.PanelBase`*

`CENTER_PANE = True`

`compute_extrapolation`(*event=None*)
Compute and plot the extrapolated data. Called when Extrapolate button is pressed.

`compute_transform`(*event=None*)
Compute and plot the transformed data. Called when Transform button is pressed.

`extract_parameters`(*event=None*)
Called when "Extract Parameters" is clicked

`get_data`()
return list of current data

`get_save_flag`()
Get the save flag to update appropriately the tool bar

`get_state`()
Return the state of the panel

`onSetFocus`(*evt*)

`on_help`(*event=None*)
Show the corfunc documentation

`on_save`(*event=None*)
Save corfunc state into a file

`on_set_focus`(*event=None*)
The derivative class is on focus if implemented

`plot_qrange`(*active=None*, *leftdown=False*)

radio_changed(*event=None*)
Called when the "Transform type" radio button are changed

save_project(*doc=None*)
Return an XML node containing the state of the panel

Parameters **doc** – Am xml node to attach the project state to (optional)

set_background(*bg*)

set_data(*data=None*, *set_qrange=True*)
Update the GUI to reflect new data that has been loaded in

Parameters **data** – The data that has been loaded

set_extracted_params(*params=None*, *reset=False*)
Displays the values of the parameters extracted from the Fourier transform

set_extrapolation_params(*params=None*)
Displays the value of the parameters calculated in the extrapolation

set_qmax(*qmax*)

set_qmin(*qmin*)

set_state(*state=None*, *data=None*)
Set the state of the panel. If no state is provided, the panel will be set to the default state.

Parameters

- **state** – A CorfuncState object
- **data** – A Data1D object

transform_complete(*transforms=None*)
Called from FourierThread when calculation has completed

transform_update(*msg="*)
Called from FourierThread to update on status of calculation

window_caption = 'Correlation Function'

window_name = 'Correlation Function'

sas.sasgui.perspectives.corfunc.corfunc_state module

class sas.sasgui.perspectives.corfunc.corfunc_state.**CorfuncState**
Bases: `object`

Stores information about the state of CorfuncPanel

fromXML(*node*)
Load corfunc states from a file

Parameters **node** – node of an XML document to read from (optional)

set_saved_state(*name*, *value*)
Set a value in the current state.

Parameters

- **name** – The name of the parameter to set
- **value** – The value to set the parameter to

toXML(*filename='corfunc_state.crf'*, *doc=None*, *entry_node=None*)
Writes the state of the CorfuncPanel panel to file, as XML.

Compatible with standalone writing, or appending to an already existing XML document. In that case, the XML document is required. An optional entry node in the XML document may also be given.

Parameters

- **`file`** – file to write to
- **`doc`** – XML document object [optional]
- **`entry_node`** – XML node within the XML document at which we will append the data [optional]

Returns None if no doc is provided, modified XML document if doc!=None

class `sas.sasgui.perspectives.corfunc.corfunc_state.`**`Reader`**(*callback*)

Bases: *`sas.sascalc.dataloader.readers.cansas_reader.Reader`*

Reads a CanSAS file containing the state of a CorfuncPanel

`ext = ['.crf', '.CRF', '.svs', '.SVS']`

`get_state`()

`read`(*path*)

Load data and corfunc information frmo a CanSAS file.

Parameters **`path`** – The file path to read from

Returns Data1D object, a list of Data1D objects, or None

Raises

- **`IOError`** – When the file can't be found
- **`IOError`** – When the file is an invalid file type
- **`ValueError`** – When the length of the data vectors are inconsistent

`type = ['Corfunc file (*.crf)|*.crf', 'SASView file (*.svs)|*.svs']`

`type_name = 'Corfunc'`

`write`(*filename*, *datainfo=None*, *state=None*)

Write the content of a Data1D as a CanSAS file.

: param filename: Name of the file to write : param datainfo: Data1D object : param state: Corfunc-State object

sas.sasgui.perspectives.corfunc.plot_labels module

Module contents

sas.sasgui.perspectives.file_converter package

Submodules

sas.sasgui.perspectives.file_converter.converter_panel module

This module provides a GUI for the file converter

class `sas.sasgui.perspectives.file_converter.converter_panel.`**`ConverterPanel`**(*parent*, *base=None*, **args*, ***kwargs*)

Bases: `wx.lib.scrolledpanel.ScrolledPanel`, *`sas.sasgui.guiframe.panel_base.PanelBase`*

This class provides the File Converter GUI

ask_frame_range(*n_frames*)
Display a dialog asking the user to input the range of frames they would like to export

Parameters **n_frames** – How many frames the loaded data file has

Returns A dictionary containing the parameters input by the user

convert_1d_data(*qdata*, *iqdata*)
Formats a 1D array of q_axis data and a 2D array of I axis data (where each row of iqdata is a separate row), into an array of Data1D objects

convert_2d_data(*dataset*)

convert_to_cansas(*frame_data*, *filepath*, *single_file*)
Saves an array of Data1D objects to a single CanSAS file with multiple <SasData> elements, or to multiple CanSAS files, each with one <SasData> element.

Parameters

- **frame_data** – If single_file is true, an array of Data1D objects. If single_file is false, a dictionary of the form *{frame_number: Data1D}*.
- **filepath** – Where to save the CanSAS file
- **single_file** – If true, array is saved as a single file, if false, each item in the array is saved to it's own file

datatype_changed(*event*)
Update the UI and self.data_type when a data type radio button is pressed

extract_ascii_data(*filename*)
Extracts data from a single-column ASCII file

Parameters **filename** – The file to load data from

Returns A numpy array containing the extracted data

extract_bsl_data(*filename*)
Extracts data from a 2D BSL file

Parameters **filename** – The header file to extract the data from

Return x_data A 1D array containing all the x coordinates of the data

Return y_data A 1D array containing all the y coordinates of the data

Return frame_data A dictionary of the form *{frame_number: data}*, where data is a 2D numpy array containing the intensity data

extract_otoko_data(*filename*)
Extracts data from a 1D OTOKO file

Parameters **filename** – The OTOKO file to load the data from

Returns A numpy array containing the extracted data

get_metadata()

metadata_changed(*event*)

on_collapsible_pane(*event*)
Resize the scrollable area to fit the metadata pane when it's collapsed or expanded

on_convert(*event*)
Called when the Convert button is clicked

on_help(*event*)
Show the File Converter documentation

radiationtype_changed(*event*)

show_detector_window(*event*)
Show the window for inputting Detector metadata

show_sample_window(*event*)
Show the window for inputting Sample metadata

show_source_window(*event*)
Show the window for inputting Source metadata

validate_inputs()

class sas.sasgui.perspectives.file_converter.converter_panel.**ConverterWindow**(*parent=None, title='File Converter', base=None, manager=None, size=(518.4, 486.0), *args, **kwargs*)

Bases: `wx._windows.Frame`

Displays ConverterPanel

on_close(*event*)

sas.sasgui.perspectives.file_converter.converter_widgets module

This module provides some custom wx widgets for the file converter perspective

class sas.sasgui.perspectives.file_converter.converter_widgets.**FileInput**(*parent, wildcard=''*)

Bases: `object`

GetCtrl()

GetPath()

SetWildcard(*wildcard*)

class sas.sasgui.perspectives.file_converter.converter_widgets.**VectorInput**(*parent, control_name, callback=None, labels=['x: ', 'y: ', 'z: '], z_enabled=False*)

Bases: `object`

An input field for inputting 2 (or 3) components of a vector.

GetName()

GetSizer()
Get the control's sizer

Return sizer a wx.BoxSizer object

GetValue()
Get the value of the vector input

Return v A Vector object

SetValue(*vector*)
Set the value of the vector input

Parameters **vector** – A Vector object

Validate()
Validate the contents of the inputs

Return all_valid Whether or not the inputs are valid

Return invalid_ctrl A control that is not valid (or None if all are valid)

sas.sasgui.perspectives.file_converter.file_converter module

File Converter Plugin

class sas.sasgui.perspectives.file_converter.file_converter.**Plugin**
Bases: *sas.sasgui.guiframe.plugin_base.PluginBase*

This class defines the interface for a Plugin class for File Converter perspective

get_tools()
Returns a set of menu entries

on_file_converter(*event*)

put_icon(*frame*)
Put icon in the frame title bar

sas.sasgui.perspectives.file_converter.frame_select_dialog module

class sas.sasgui.perspectives.file_converter.frame_select_dialog.**FrameSelectDialog**(*n_fram*
is_bsl=
Bases: wx._windows.Dialog

This class provides a wx.Dialog subclass for selecting which frames of a multi-frame file to export

sas.sasgui.perspectives.file_converter.meta_panels module

class sas.sasgui.perspectives.file_converter.meta_panels.**DetectorPanel**(*parent*, *detector*, *base=None*, **args*, ***kwargs*)
Bases: *sas.sasgui.perspectives.file_converter.meta_panels.MetadataPanel*

on_close(*event=None*)
Close event. Hide the whole window.

class sas.sasgui.perspectives.file_converter.meta_panels.**MetadataPanel**(*parent*, *metadata*, *base=None*, **args*, ***kwargs*)

Bases: `wx.lib.scrolledpanel.ScrolledPanel`, `sas.sasgui.guiframe.panel_base.PanelBase`

A common base class to be extended by panels that deal with metadata input. Handles input validation and passing inputted data back to ConverterPanel.

get_property_string(*name*, *is_float=False*)

on_change(*event*)

on_close(*event=None*)
Close event. Hide the whole window.

class sas.sasgui.perspectives.file_converter.meta_panels.**MetadataWindow**(*PanelClass*, *parent=None*, *title=''*, *base=None*, *manager=None*, *size=(470, 376.0)*, *metadata=None*, **args*, ***kwargs*)

Bases: `wx._windows.Frame`

on_close(*event*)

class sas.sasgui.perspectives.file_converter.meta_panels.**SamplePanel**(*parent*, *sample*, *base=None*, **args*, ***kwargs*)

Bases: `sas.sasgui.perspectives.file_converter.meta_panels.MetadataPanel`

on_close(*event=None*)
Close event. Hide the whole window.

class sas.sasgui.perspectives.file_converter.meta_panels.**SourcePanel**(*parent*, *source*, *base=None*, **args*, ***kwargs*)

Bases: `sas.sasgui.perspectives.file_converter.meta_panels.MetadataPanel`

on_close(*event=None*)
Close event. Hide the whole window.

Module contents

sas.sasgui.perspectives.fitting package

Subpackages

sas.sasgui.perspectives.fitting.plugin_models package

Module contents

Example plugin models to be added to the SasView plugin directory on startup if no plugins are present.

This is currently empty.

Submodules

sas.sasgui.perspectives.fitting.basepage module

Base Page for fitting

class `sas.sasgui.perspectives.fitting.basepage.`**`BasicPage`**(*parent*, *color='blue'*, ***kwargs*)

Bases: `wx.lib.scrolledpanel.ScrolledPanel`, *`sas.sasgui.guiframe.panel_base.PanelBase`*

This class provide general structure of the fitpanel page

`ID_BOOKMARK = 177`

`ID_DISPERSER_HELP = 178`

`check_invalid_panel`()
check if the user can already perform some action with this panel

`createMemento`()
return the current state of the page

`create_default_data`()
Given the user selection, creates a 1D or 2D data Only when the page is on theory mode.

`define_page_structure`()
Create empty sizer for a panel

`formfactor_combo_init`()
First time calls _show_combox_helper

`get_all_checked_params`()
Found all parameters current check and add them to list of parameters to fit if implemented

`get_cat_combo_box_pos`(*state*)
Iterate through the categories to find the structurefactor :return: combo_box_position

`get_clipboard`()
Get strings in the clipboard

`get_copy`()
Get copy params to clipboard

`get_copy_excel`()
Get copy params to clipboard

`get_copy_latex`()
Get copy params to clipboard

get_copy_params()
Get the string copies of the param names and values in the tap

get_copy_params_excel()
Get the string copies of the param names and values in the tap

get_copy_params_latex()
Get the string copies of the param names and values in the tap

get_data()
return the current data

get_data_list()
return the current data

get_images()
Get the images of the plots corresponding this panel for report

: return graphs: list of figures : Need Move to guiframe

get_paste()
Paste params from the clipboard

get_paste_params(*text=''*)
Get the string copies of the param names and values in the tap

get_state()
return the current page state

get_weight_flag()
Get flag corresponding to a given weighting dI data if implemented

initialize_combox()
put default value in the combo box

onContextMenu(*event*)
Retrieve the state selected state

onPinholeSmear(*event*)
Create a custom pinhole smear object if implemented

onRedo(*event*)
Restore the previous action cancelled

onResetModel(*event*)
Reset model state

onSetFocus(*evt*)
highlight the current textcrtl and hide the error text control shown after fitting

onSlitSmear(*event*)
Create a custom slit smear object if implemented

onSmear(*event*)
Create a smear object if implemented

onUndo(*event*)
Cancel the previous action

on_bookmark(*event*)
save history of the data and model

on_copy(*event*)
Copy Parameter values to the clipboad

on_function_help_clicked(*event*)
Function called when 'Help' button is pressed next to model of interest. This calls DocumentationWindow from documentation_window.py. It will load the top level of the html model help documenation sphinx generated if either a plugin model (which normally does not have an html help help file) is

selected or if no model is selected. Otherwise, if a regula model is selected, the documention for that model will be sent to a browser window.

:todo the quick fix for no documentation in plugins is the if statment. However, the right way to do this would be to check whether the hmtl file exists and load the model docs if it does and the general docs if it doesn't - this will become important if we ever figure out how to build docs for plugins on the fly. Sep 9, 2018 -PDB

Parameters `event` – on Help Button pressed event

`on_left_down`(*event*)
Get key stroke event

`on_model_help_clicked`(*event*)
Function called when 'Description' button is pressed next to model of interest. This calls the Description embedded in the model. This should work with either Wx2.8 and lower or higher. If no model is selected it will give the message that a model must be chosen first in the box that would normally contain the description. If a badly behaved model is encountered which has no description then it will give the message that none is available.

Parameters `event` – on Description Button pressed event

`on_paste`(*event*)
Paste Parameter values to the panel if possible

`on_pd_help_clicked`(*event*)
Bring up Polydispersity Documentation whenever the ? button is clicked. Calls DocumentationWindow with the path of the location within the documentation tree (after /doc/ ….". Note that when using old versions of Wx (before 2.9) and thus not the release version of istallers, the help comes up at the top level of the file as webbrowser does not pass anything past the # to the browser when it is running "file:///…."

Parameters `event` – Triggers on clicking ? in polydispersity box

`on_preview`(*event*)
Report the current fit results

`on_reset_clicked`(*event*)
On 'Reset' button for Q range clicked

`on_save`(*event*)
Save the current state into file

`on_set_focus`(*event*)
On Set Focus, update guimanger and menu

`on_smear_helper`(*update=False*)
Help for onSmear if implemented

Parameters `update` – force or not to update

`on_tap_focus`()
Update menu1 on cliking the page tap

`populate_box`(*model_list_box*)
Store list of model

Parameters `model_list_box` – dictionary containing categorized models

`read_file`(*path*)
Read two columns file

Parameters `path` – the path to the file to read

`reset_page`(*state*, *first=False*)
reset the state if implemented

`reset_page_helper`(*state*)
Use page_state and change the state of existing page

Precondition the page is already drawn or created

Postcondition the state of the underlying data changes as well as the state of the graphic interface

`save_current_state`()
Store current state

`save_current_state_fit`()
Store current state for fit_page

`select_log`(*event*)
Log checked to generate log spaced points for theory model

`select_param`(*event*)
Select TextCtrl checked if implemented

`set_clipboard`(*content=None*)
Put the string to the clipboard

`set_data`(*data=None*)
Sets data if implemented

`set_dispers_sizer`()
fill sizer containing dispersity info

`set_index_model`(*index*)
Index related to this page

`set_layout`()
layout

`set_manager`(*manager*)
set panel manager

Parameters **`manager`** – instance of plugin fitting

`set_model_dictionary`(*model_dictionary*)
Store a dictionary linking model name -> model object

Parameters **`model_dictionary`** – dictionary containing all models

`set_model_state`(*state*)
reset page given a model state

`set_owner`(*owner*)
set owner of fitpage

Parameters **`owner`** – the class responsible of plotting

`show_npts2fit`()
setValue Npts for fitting if implemented

`update_pinhole_smear`()
Method to be called by sub-classes Moveit; This method doesn't belong here

`update_slit_smear`()
called by kill_focus on pinhole TextCntrl to update the changes if implemented

`window_caption = 'Fit Page '`

`window_name = 'Fit Page'`

class sas.sasgui.perspectives.fitting.basepage.**ModelTextCtrl**(*parent, id=-1, value=u", pos=wx.Point(-1, -1), size=wx.Size(-1, -1), style=0, validator=<wx._core.Validator; proxy of <Swig Object of type 'wxValidator *'> >, name=u'text', kill_focus_callback=None, set_focus_callback=None, mouse_up_callback=None, text_enter_callback=None*)

Bases: `wx._controls.TextCtrl`

Text control for model and fit parameters. Binds the appropriate events for user interactions. Default callback methods can be overwritten on initialization

Parameters

- **kill_focus_callback** – callback method for EVT_KILL_FOCUS event
- **set_focus_callback** – callback method for EVT_SET_FOCUS event
- **mouse_up_callback** – callback method for EVT_LEFT_UP event
- **text_enter_callback** – callback method for EVT_TEXT_ENTER event

full_selection = False

sas.sasgui.perspectives.fitting.batchfitpage module

Batch panel

class sas.sasgui.perspectives.fitting.batchfitpage.**BGTextCtrl**(**args, **kwds*)

Bases: `wx._controls.TextCtrl`

Text control used to display outputs. No editing allowed. The background is grayed out. User can't select text.

class sas.sasgui.perspectives.fitting.batchfitpage.**BatchFitPage**(*parent, color=None*)

Bases: *sas.sasgui.perspectives.fitting.fitpage.FitPage*

Batch Page

window_caption = 'BatchFit'

window_name = 'BatchFit'

sas.sasgui.perspectives.fitting.console module

class sas.sasgui.perspectives.fitting.console.**ConsoleUpdate**(*parent, manager=None, quiet=False, progress_delta=60, improvement_delta=5*)

Bases: *sas.sascalc.fit.AbstractFitEngine.FitHandler*

Print progress to the console.

abort()

error(*msg*)
Model had an error; print traceback

finalize()

get_result()

improvement()
Called when a result is observed which is better than previous results from the fit.

improvement_delta = 5
Number of seconds between improvement updates

isbetter = False
Record whether results improved since last update

print_result()
Print result object

progress(*k*, *n*)
Report on progress.

progress_delta = 60
Number of seconds between progress updates

set_result(*result*)

starting_fit()

stop(*msg*)
Post event msg and stop

update_fit(*last=False*)

sas.sasgui.perspectives.fitting.fit_thread module

class sas.sasgui.perspectives.fitting.fit_thread.**FitThread**(*fn*, *page_id*, *handler*, *batch_outputs*, *batch_inputs=None*, *pars=None*, *completefn=None*, *updatefn=None*, *yieldtime=0.03*, *worktime=0.03*, *reset_flag=False*)

Bases: *sas.sascalc.data_util.calcthread.CalcThread*

Thread performing the fit

compute()
Perform a fit

isquit()

Raises KeyboardInterrupt – when the thread is interrupted

sas.sasgui.perspectives.fitting.fit_thread.**map_apply**(*arguments*)

sas.sasgui.perspectives.fitting.fit_thread.**map_getattr**(*classInstance*, *classFunc*, **args*)
Take an instance of a class and a function name as a string. Execute class.function and return result

sas.sasgui.perspectives.fitting.fitpage module

FitPanel class contains fields allowing to display results when fitting a model and one data

class `sas.sasgui.perspectives.fitting.fitpage.`**`BGTextCtrl`**(**args*, ***kwds*)

Bases: `wx._controls.TextCtrl`

Text control used to display outputs. No editing allowed. The background is grayed out. User can't select text.

class `sas.sasgui.perspectives.fitting.fitpage.`**`FitPage`**(*parent*, *color=None*)

Bases: *`sas.sasgui.perspectives.fitting.basepage.BasicPage`*

FitPanel class contains fields allowing to display results when fitting a model and one data

Note For Fit to be performed the user should check at least one parameter on fit Panel window.

`compute_data_range`(*data*)

compute the minimum and the maximum range of the data return the npts contains in data :param data:

`compute_data_set_range`(*data_list*)

find the range that include all data in the set return the minimum and the maximum values

`enable_datasource`()

Enable or disable data source control depending on existing data

`enable_fit_button`()

Enable fit button if data is valid and model is valid

`fill_data_combobox`(*data_list*)

Get a list of data and fill the corresponding combobox

`get_all_checked_params`()

Found all parameters current check and add them to list of parameters to fit

`get_chi2`()

return the current chi2

`get_npts2fit`()

return numbers of data points within qrange

Note This is to normalize chisq by Npts of fit

`get_range`()

return the fitting range

`get_view_mode`()

return True if the panel allow 2D or False if 1D

`get_weight_flag`()

Get flag corresponding to a given weighting dI data.

`onPinholeSmear`(*event*)

Create a custom pinhole smear object that will change the way residuals are compute when fitting

Note accuracy is given by strings'High','Med', 'Low' FOR 2d, None for 1D

`onSlitSmear`(*event*)

Create a custom slit smear object that will change the way residuals are compute when fitting

`onSmear`(*event*)

Create a smear object that will change the way residuals are computed when fitting

`onWeighting`(*event*)

On Weighting radio button event, sets the weightbt_string

`on_complete_chisqr`(*event*)

Display result chisqr on the panel :event: activated by fitting/ complete after draw

on_key(*event*)
On Key down

on_qrange_text(*event*)
#On q range value updated. DO not combine with qrange_click().

on_right_down(*event*)
Get key stroke event

on_select_data(*event=None*)
On_select_data

on_set_focus(*event*)
Override the basepage focus method to ensure the save flag is set properly when focusing on the fit page.

on_smear_helper(*update=False*)
Help for onSmear

Parameters **update** – force or not to update

onsetValues(*chisqr*, *p_name*, *out*, *cov*)
Build the panel from the fit result

Parameters

- **chisqr** – Value of the goodness of fit metric
- **p_name** – the name of parameters
- **out** – list of parameter with the best value found during fitting
- **cov** – Covariance matrix

qrang_set_focus(*event=None*)
ON Qrange focus

qrange_click(*event*)
On Qrange textctrl click, make the qrange lines in the plot

rename_model()
find a short name for model

reset_page(*state*, *first=False*)
reset the state

select_param(*event=None*)
Select TextCtrl checked for fitting purpose and stores them in self.param_toFit=[] list

set_data(*data*)
reset the current data

set_fitbutton()
Set fit button label depending on the fit_started[bool]

set_model_param_sizer(*model*)
Build the panel from the model content

Parameters **model** – the model selected in combo box for fitting purpose

show_npts2fit()
setValue Npts for fitting

update_pinhole_smear()
called by kill_focus on pinhole TextCntrl to update the changes

Returns False when wrong value was entered

update_slit_smear()
called by kill_focus on pinhole TextCntrl to update the changes

Returns False when wrong value was entered

sas.sasgui.perspectives.fitting.fitpanel module

FitPanel class contains fields allowing to fit models and data

note For Fit to be performed the user should check at least one parameter on fit Panel window.

`class sas.sasgui.perspectives.fitting.fitpanel.`**`FitPanel`**(*parent*, *manager=None*, **args*, ***kwargs*)

Bases: `wx.aui.AuiNotebook`, *`sas.sasgui.guiframe.panel_base.PanelBase`*

FitPanel class contains fields allowing to fit models and data

Note For Fit to be performed the user should check at least one parameter on fit Panel window.

`CENTER_PANE = True`

`add_empty_page`()
add an empty page

`add_sim_page`(*caption='Const & Simul Fit'*)
Add the simultaneous fit page

`clear_panel`()
Clear and close all panels, used by guimanager

`close_all`()
remove all pages, used when a svs file is opened

`close_page_with_data`(*deleted_data*)
close a fit page when its data is completely remove from the graph

`delete_data`(*data*)
Delete the given data

`enable_close_button`()
display the close button on tab for more than 1 tabs else remove the close button

`get_current_page`()

Returns the current page selected

`get_data`()
get the data in the current page

`get_page_by_id`(*uid*)

`get_state`()
return the state of the current selected page

`helper_on_page_change`()

`on_close_page`(*event=None*)
close page and remove all references to the closed page

`on_closed`(*event*)

`on_page_changing`(*event*)
calls the function when the current event handler has exited. avoiding to call panel on focus on a panel that is currently deleted

`on_set_focus`(*event*)

`reset_pmodel_list`()

`save_project`(*doc=None*)
return an xml node containing state of the panel that guiframe can write to file

set_data(*data_list*)
Add a fitting page on the notebook contained by fitpanel

Parameters **data_list** – data to fit

:return panel : page just added for further used. is used by fitting module

set_data_on_batch_mode(*data_list*)
Add all data to a single tab when the application is on Batch mode. However all data in the set of data must be either 1D or 2D type. This method presents option to select the data type before creating a tab.

set_manager(*manager*)
set panel manager

Parameters **manager** – instance of plugin fitting

set_model_dictionary(*model_dictionary*)
copy a dictionary of model name -> model object

Parameters **model_dictionary** – dictionary linking model name -> model object

set_model_list(*dict*)
copy a dictionary of model into its own dictionary

Parameters **dict** – dictionnary made of model name as key and model class as value

set_model_state(*state*)
receive a state to reset the model in the current page

set_state(*state*)
Restore state of the panel

update_model_list()

window_caption = 'Fit Panel '

window_name = 'Fit panel'

sas.sasgui.perspectives.fitting.fitproblem module

Inferface containing information to store data, model, range of data, etc… and retrieve this information. This is an inferface for a fitProblem i.e relationship between data and model.

class `sas.sasgui.perspectives.fitting.fitproblem.`**FitProblem**
Bases: `object`

Define the relationship between data and model, including range, weights, etc.

clear_model_param()
clear constraint info

enable_smearing(*flag=False*)

Parameters **flag** – bool.When flag is 1 get the computer smear value. When flag is 0 ignore smear value.

get_fit_data()

Returns data associate with this class

get_fit_tab_caption()

get_graph_id()
Get graph_id

get_model()

Returns saved model

get_model_param()
return list of couple of parameter name and value

get_name()

get_origin_data()

get_param2fit()
return the list param names to fit

get_range()

Returns fitting range

get_residuals()

Returns residuals

get_result()
get result

get_scheduled()
return true or false if a problem as being schedule for fitting

get_smearer()
return smear object

get_theory_data()

Returns theory generated with the current model and data of this class

get_weight()
returns weight array

save_model_name(*name*)

schedule_tofit(*schedule=0*)
set schedule to true to decide if this fit must be performed

set_fit_data(*data*)
Store data associated with this class :param data: list of data selected

set_fit_tab_caption(*caption*)

set_graph_id(*id*)
Set graph id (from data_group_id at the time the graph produced)

set_model(*model*)
associates each model with its new created name :param model: model selected :param name: name created for model

set_model_param(*name*, *value=None*)
Store the name and value of a parameter of this fitproblem's model :param name: name of the given parameter :param value: value of that parameter

set_param2fit(*list*)
Store param names to fit (checked) :param list: list of the param names

set_range(*qmin=None*, *qmax=None*)
set fitting range :param qmin: minimum value to consider for the fit range :param qmax: maximum value to consider for the fit range

set_residuals(*residuals*)
save a copy of residual :param data: data selected

set_result(*result*)

set_smearer(*smearer*)
save reference of smear object on fitdata

Parameters **smear** – smear object from DataLoader

set_theory_data(*data*)
save a copy of the data select to fit

Parameters **data** – data selected

set_weight(*is2d*, *flag=None*)
Received flag and compute error on data. :param flag: flag to transform error of data. :param is2d: flag to distinguish 1D to 2D Data

class sas.sasgui.perspectives.fitting.fitproblem.**FitProblemDictionary**
Bases: dict

This module implements a dictionary of fitproblem objects

add_data(*data*)
Add data to the current dictionary of fitproblem. if data id does not exist create a new fit problem. :note: only data changes in the fit problem

clear_model_param(*fid=None*)
clear constraint info

enable_smearing(*flag=False*, *fid=None*)

Parameters **flag** – bool.When flag is 1 get the computer smear value. When flag is 0 ignore smear value.

get_batch_result()
get result

get_fit_data(*fid*)
return data for the given fitproblem id :param fid: key representing a fitproblem, usually extract from data id

get_fit_problem()
return fitproblem contained in this dictionary

get_fit_tab_caption()
Return the caption of the page associated with object

get_graph_id()
Get graph_id

get_model(*fid*)

Returns saved model

get_model_param(*fid*)
return list of couple of parameter name and value

get_name(*fid=None*)

get_param2fit()
return the list param names to fit

get_range(*fid*)

Returns fitting range

get_residuals(*fid*)

Returns residuals

get_result(*fid*)
get result

get_scheduled()
return true or false if a problem as being schedule for fitting

get_smearer(*fid=None*)
return smear object

get_theory_data (*fid*)

Returns list of data dList

get_weight (*fid=None*)
return fit weight

save_model_name (*name*, *fid=None*)

schedule_tofit (*schedule=0*)
set schedule to true to decide if this fit must be performed

set_batch_result (*batch_inputs*, *batch_outputs*)
set a list of result

set_fit_data (*data*)
save a copy of the data select to fit :param data: data selected

set_fit_tab_caption (*caption*)
store the caption of the page associated with object

set_graph_id (*id*)
Set graph id (from data_group_id at the time the graph produced)

set_model (*model*, *fid=None*)
associates each model with its new created name :param model: model selected :param name: name created for model

set_model_param (*name*, *value=None*, *fid=None*)
Store the name and value of a parameter of this fitproblem's model :param name: name of the given parameter :param value: value of that parameter

set_param2fit (*list*)
Store param names to fit (checked) :param list: list of the param names

set_range (*qmin=None*, *qmax=None*, *fid=None*)
set fitting range

set_residuals (*residuals*, *fid*)
save a copy of residual :param data: data selected

set_result (*result*, *fid*)

set_smearer (*smearer*, *fid=None*)
save reference of smear object on fitdata :param smear: smear object from DataLoader

set_theory_data (*fid*, *data=None*)
save a copy of the data select to fit :param data: data selected

set_weight (*is2d*, *flag=None*, *fid=None*)
fit weight

sas.sasgui.perspectives.fitting.fitting module

Fitting perspective

class sas.sasgui.perspectives.fitting.fitting.**Plugin**
Bases: *sas.sasgui.guiframe.plugin_base.PluginBase*

Fitting plugin is used to perform fit

add_color (*color*, *id*)
adds a color as a key with a plot id as its value to a dictionary

add_fit_page (*data*)
given a data, ask to the fitting panel to create a new fitting page, get this page and store it into the page_finder of this plug-in :param data: is a list of data

clear_panel()

create_fit_problem(*page_id*)
Given an ID create a fitproblem container

create_theory_1D(*x*, *y*, *page_id*, *model*, *data*, *state*, *data_description*, *data_id*, *dy=None*)
Create a theory object associate with an existing Data1D and add it to the data manager. @param x: x-values of the data @param y: y_values of the data @param page_id: fit page ID @param model: model used for fitting @param data: Data1D object to create the theory for @param state: model state @param data_description: title to use in the data manager @param data_id: unique data ID

delete_custom_model(*event*)
Delete custom model file

delete_data(*data*)
delete the given data from panel

delete_fit_problem(*page_id*)
Given an ID create a fitproblem container

draw_model(*model*, *page_id*, *data=None*, *smearer=None*, *enable1D=True*, *enable2D=False*, *state=None*, *fid=None*, *toggle_mode_on=False*, *qmin=None*, *qmax=None*, *update_chisqr=True*, *weight=None*, *source='model'*)
Draw model.

Parameters

- **model** – the model to draw
- **name** – the name of the model to draw
- **data** – the data on which the model is based to be drawn
- **description** – model's description
- **enable1D** – if true enable drawing model 1D
- **enable2D** – if true enable drawing model 2D
- **qmin** – Range's minimum value to draw model
- **qmax** – Range's maximum value to draw model
- **qstep** – number of step to divide the x and y-axis
- **update_chisqr** – update chisqr [bool]

edit_custom_model(*event*)
Get the python editor panel

get_batch_capable()
Check if the plugin has a batch capability

get_context_menu(*plotpanel=None*)
Get the context menu items available for P(r).them allow fitting option for Data2D and Data1D only.

Parameters **graph** – the Graph object to which we attach the context menu

Returns a list of menu items with call-back function

Note if Data1D was generated from Theory1D the fitting option is not allowed

get_graph_id(*uid*)
Set graph_id for fitprobelm

get_page_finder()
return self.page_finder used also by simfitpage.py

get_panels(*parent*)
Create and return a list of panel objects

load_plugin_models(*event*)
Update of models in plugin_models folder

make_new_model(*event*)
Make new model

make_sum_model(*event*)
Edit summodel template and make one

onFit(*uid*)
Get series of data, model, associates parameters and range and send then to series of fitters. Fit data and model, display result to corresponding panels. :param uid: id related to the panel currently calling this fit function.

on_add_new_page(*event=None*)
ask fit panel to create a new empty page

on_add_sim_page(*event*)
Create a page to access simultaneous fit option

on_batch_selection(*flag*)
switch the the notebook of batch mode or not

on_bumps_options(*event=None*)
Open the bumps options panel.

on_fit_results(*event=None*)
Make the Fit Results panel visible.

on_fitter_changed(*event*)

on_gpu_options(*event=None*)
Make the Fit Results panel visible.

on_help(*algorithm_id*)

on_reset_batch_flag(*event*)
Set batch_reset_flag

on_set_batch_result(*page_id*, *fid*, *batch_outputs*, *batch_inputs*)

on_set_state_helper(*event=None*)
Set_state_helper. This actually sets state after plotting data from state file.

: event: FitStateUpdateEvent called by dataloader.plot_data from guiframe

populate_menu(*owner*)
Create a menu for the Fitting plug-in

Parameters

- `id` – id to create a menu
- `owner` – owner of menu

Returns list of information to populate the main menu

put_icon(*frame*)
Put icon in the frame title bar

remove_plot(*uid*, *fid=None*, *theory=False*)
remove model plot when a fit page is closed :param uid: the id related to the fitpage to close :param fid: the id of the fitproblem(data, model, range,etc)

save_fit_state(*filepath*, *fitstate*)
save fit page state into file

schedule_for_fit(*value=0*, *uid=None*)
Set the fit problem field to 0 or 1 to schedule that problem to fit. Schedule the specified fitproblem or

get the fit problem related to the current page and set value. :param value: integer 0 or 1 :param uid: the id related to a page containing fitting information

select_data (*panel*)

set_data (*data_list=None*)
receive a list of data to fit

set_edit_menu (*owner*)
Set list of the edit model menu labels

set_edit_menu_helper (*owner=None*, *menu=None*)
help for setting list of the edit model menu labels

set_fit_range (*uid*, *qmin*, *qmax*, *fid=None*)
Set the fitting range of a given page for all its data by default. If fid is provide then set the range only for the data with fid as id :param uid: id corresponding to a fit page :param fid: id corresponding to a fit problem (data, model) :param qmin: minimum value of the fit range :param qmax: maximum value of the fit range

set_fit_weight (*uid*, *flag*, *is2d=False*, *fid=None*)
Set the fit weights of a given page for all its data by default. If fid is provide then set the range only for the data with fid as id :param uid: id corresponding to a fit page :param fid: id corresponding to a fit problem (data, model) :param weight: current dy data

set_graph_id (*uid*, *graph_id*)
Set graph_id for fitprobelm

set_page_finder (*modelname*, *names*, *values*)
Used by simfitpage.py to reset a parameter given the string constrainst.

Parameters

- **modelname** – the name of the model for with the parameter has to reset
- **value** – can be a string in this case.
- **names** – the parameter name

set_param2fit (*uid*, *param2fit*)
Set the list of param names to fit for fitprobelm

set_smearer (*uid*, *smearer*, *fid*, *qmin=None*, *qmax=None*, *draw=True*, *enable_smearer=False*)
Get a smear object and store it to a fit problem of fid as id. If proper flag is enable , will plot the theory with smearing information.

Parameters

- **smearer** – smear object to allow smearing data of id fid
- **enable_smearer** – Define whether or not all (data, model) contained in the structure of id uid will be smeared before fitting.
- **qmin** – the maximum value of the theory plotting range
- **qmax** – the maximum value of the theory plotting range
- **draw** – Determine if the theory needs to be plot

set_state (*state=None*, *datainfo=None*, *format=None*)
Call-back method for the fit page state reader. This method is called when a .fitv/.svs file is loaded.

: param state: PageState object : param datainfo: data

set_theory (*theory_list=None*)

split_string (*item*)
receive a word containing dot and split it. used to split parameterset name into model name and parameter name example:

```
parameterset (item) = M1.A
Will return model_name = M1 , parameter name = A
```

stop_fit (*uid*)
Stop the fit

store_data (*uid*, *data_list=None*, *caption=None*)
Receive a list of data and store them ans well as a caption of the fit page where they come from. :param uid: if related to a fit page :param data_list: list of data to fit :param caption: caption of the window related to these data

update_custom_combo ()
Update custom model list in the fitpage combo box

update_fit (*result=None*, *msg=''*)

sas.sasgui.perspectives.fitting.fitting_widgets module

class sas.sasgui.perspectives.fitting.fitting_widgets.**BatchDataDialog** (*parent=None*, **args*, ***kwds*)

Bases: `wx._windows.Dialog`

The current design of Batch fit allows only of type of data in the data set. This allows the user to make a quick selection of the type of data to use in fit tab.

get_data ()
return 1 if user requested Data1D , 2 if user requested Data2D

class sas.sasgui.perspectives.fitting.fitting_widgets.**DataDialog** (*data_list*, *parent=None*, *text=''*, *nb_data=4*, **args*, ***kwds*)

Bases: `wx._windows.Dialog`

Allow file selection at loading time

get_data ()
return the selected data

class sas.sasgui.perspectives.fitting.fitting_widgets.**DialogPanel** (**args*, ***kwds*)

Bases: `wx.lib.scrolledpanel.ScrolledPanel`

sas.sasgui.perspectives.fitting.gpu_options module

Module provides dialog for setting SAS_OPENCL variable, which defines device choice for OpenCL calculation

Created on Nov 29, 2016

@author: wpotrzebowski

class sas.sasgui.perspectives.fitting.gpu_options.**CustomMessageBox** (*parent*, *msg*, *title*)

Bases: `wx._windows.Dialog`

Custom message box for OpenCL results

class sas.sasgui.perspectives.fitting.gpu_options.**GpuOptions**(**args*, ***kwds*)

Bases: wx._windows.Dialog

"OpenCL options" Dialog Box

Provides dialog for setting SAS_OPENCL variable, which defines device choice for OpenCL calculation

on_OK(*event*)
Close window on accpetance

on_check(*event*)
Action triggered when box is selected :param event: :return:

on_help(*event*)
Provide help on opencl options.

on_reset(*event*)
Resets selected values

on_test(*event*)
Run sasmodels check from here and report results from

sas.sasgui.perspectives.fitting.hint_fitpage module

This class provide general structure of fitpanel page

class sas.sasgui.perspectives.fitting.hint_fitpage.**HelpWindow**(*parent*, *id*, *title*)

Bases: wx._windows.Frame

class sas.sasgui.perspectives.fitting.hint_fitpage.**HintFitPage**(*parent*)

Bases: wx._windows.ScrolledWindow, *sas.sasgui.guiframe.panel_base.PanelBase*

This class provide general structure of fitpanel page

createMemento()

do_layout()
Draw the page

window_caption = 'Hint page '

window_name = 'Hint Page'

sas.sasgui.perspectives.fitting.model_thread module

Calculation thread for modeling

class sas.sasgui.perspectives.fitting.model_thread.**Calc1D**(*model*, *page_id*, *data*, *fid=None*, *qmin=None*, *qmax=None*, *weight=None*, *smearer=None*, *toggle_mode_on=False*, *state=None*, *completefn=None*, *update_chisqr=True*, *source='model'*, *updatefn=None*, *yieldtime=0.01*, *worktime=0.01*, *exception_handler=None*)

Bases: `sas.sascalc.data_util.calcthread.CalcThread`

Compute 1D data

compute()
Compute model 1d value given qmin , qmax , x value

results()
Send resuts of the computation

class sas.sasgui.perspectives.fitting.model_thread.**Calc2D**(*data*, *model*, *smearer*, *qmin*, *qmax*, *page_id*, *state=None*, *weight=None*, *fid=None*, *toggle_mode_on=False*, *completefn=None*, *updatefn=None*, *update_chisqr=True*, *source='model'*, *yieldtime=0.04*, *worktime=0.04*, *exception_handler=None*)

Bases: `sas.sascalc.data_util.calcthread.CalcThread`

Compute 2D model This calculation assumes a 2-fold symmetry of the model where points are computed for one half of the detector and I(qx, qy) = I(-qx, -qy) is assumed.

compute()
Compute the data given a model function

sas.sasgui.perspectives.fitting.report_dialog module

Dialog report panel to show and summarize the results of the fitting calculation.

class sas.sasgui.perspectives.fitting.report_dialog.**ReportDialog**(*report_list*, **args*, ***kwds*)

Bases: `sas.sasgui.guiframe.report_dialog.BaseReportDialog`

The report dialog box.

onSave (*event=None*)
Save

sas.sasgui.perspectives.fitting.resultpanel module

FitPanel class contains fields allowing to fit models and data

note For Fit to be performed the user should check at least one parameter on fit Panel window.

class sas.sasgui.perspectives.fitting.resultpanel.**ResultPanel** (*parent*, *manager=None*, **args*, ***kwargs*)

Bases: `wx.aui.AuiNotebook`, *sas.sasgui.guiframe.panel_base.PanelBase*

FitPanel class contains fields allowing to fit models and data

Note For Fit to be performed the user should check at least one parameter on fit Panel window.

CENTER_PANE = True

get_frame ()

on_close (*event*)
Close event. Hide the whole window.

on_plot_results (*event*)

window_caption = 'Result Panel'

window_name = 'Result panel'

sas.sasgui.perspectives.fitting.simfitpage module

Simultaneous or Batch fit page

class sas.sasgui.perspectives.fitting.simfitpage.**ConstraintLine** (*model_cbox*, *param_cbox*, *egal_txt*, *constraint*, *btRemove*, *sizer*)

Bases: `tuple`

btRemove
Alias for field number 4

constraint
Alias for field number 3

egal_txt
Alias for field number 2

model_cbox
Alias for field number 0

param_cbox
Alias for field number 1

sizer
Alias for field number 5

class `sas.sasgui.perspectives.fitting.simfitpage.`**`SimultaneousFitPage`**(*parent*, *page_finder={}*, *id=-1*, *batch_on=False*, **args*, ***kwargs*)

Bases: `wx.lib.scrolledpanel.ScrolledPanel`, *`sas.sasgui.guiframe.panel_base.PanelBase`*

Simultaneous fitting panel All that needs to be defined are the two data members window_name and window_caption

`ID_ADD = 182`

`ID_DOC = 179`

`ID_FIT = 181`

`ID_SET_ALL = 180`

`check_all_model_name`(*event=None*)
check all models names

`check_model_name`(*event*)
Save information related to checkbox and their states

`define_page_structure`()
Create empty sizers, their hierarchy and set the sizer for the panel

`draw_page`()
Construct the Simultaneous/Constrained fit page. fills the first region (sizer1) with the list of available fit page pairs of data and models. Then fills sizer2 with the checkbox for adding constraints, and finally fills sizer3 with the fit button and instructions.

`get_state`()
Return the state of the current page :return: self.state

`load_from_save_state`(*sim_state*)
Load in a simultaneous/constrained fit from a save state :param fit: Fitpanel object :return: None

`on_fit`(*event*)
signal for fitting

`on_remove`(*event*)
Remove constraint fields

`on_set_focus`(*event=None*)
The derivative class is on focus if implemented

`set_fitbutton`()
Set fit button label depending on the fit_started

`set_manager`(*manager*)
set panel manager

Parameters **`manager`** – instance of plugin fitting

`set_state`()
Define a set of state parameters for saving simultaneous fits.

`window_caption = 'Simultaneous Fit Page'`

`window_name = 'Simultaneous Fit Page'`

`sas.sasgui.perspectives.fitting.simfitpage.`**`get_fittableParam`**(*model*)
return list of fittable parameters from a model

Parameters **`model`** – the model used `sas.sasgui.perspectives.fitting.simfitpage.`**`setComboBoxItems`**`(`*cbox*, *items*`)`

sas.sasgui.perspectives.fitting.utils module

Module contains functions frequently used in this package

`sas.sasgui.perspectives.fitting.utils.`**`get_weight`**`(`*data*, *is2d*, *flag=None*`)`
Received flag and compute error on data. :param flag: flag to transform error of data. :param is2d: flag to distinguish 1D to 2D Data

Module contents

`sas.sasgui.perspectives.fitting.`**`data_files`**`()`
Return the data files associated with media.

The format is a list of (directory, [files…]) pairs which can be used directly in setup(…,data_files=…) for setup.py.

`sas.sasgui.perspectives.fitting.`**`get_data_path`**`(`*media*`)`

sas.sasgui.perspectives.invariant package

Submodules

sas.sasgui.perspectives.invariant.invariant module

`class` `sas.sasgui.perspectives.invariant.invariant.`**`Plugin`**
Bases: *`sas.sasgui.guiframe.plugin_base.PluginBase`*

This class defines the interface for invariant Plugin class that can be used by the gui_manager.

`clear_panel``()`

`compute_helper``(`*data*`)`

`delete_data``(`*data_id*`)`

`get_context_menu``(`*plotpanel=None*`)`
This method is optional.

When the context menu of a plot is rendered, the get_context_menu method will be called to give you a chance to add a menu item to the context menu.

A ref to a Graph object is passed so that you can investigate the plot content and decide whether you need to add items to the context menu.

This method returns a list of menu items. Each item is itself a list defining the text to appear in the menu, a tool-tip help text, and a call-back method.

Parameters **`graph`** – the Graph object to which we attach the context menu

Returns a list of menu items with call-back function

`get_data``()`

`get_panels``(`*parent*`)`
Create and return the list of wx.Panels for your plug-in. Define the plug-in perspective.

Panels should inherit from DefaultPanel defined below, or should present the same interface. They must define “window_caption” and “window_name”.

Parameters **`parent`** – parent window

Returns list of panels

on_set_state_helper (*event=None*)
Set the state when called by EVT_STATE_UPDATE event from guiframe after a .inv/.svs file is loaded

plot_data (*scale*, *background*)
replot the current data if the user enters a new scale or background

plot_theory (*data=None*, *name=None*)
Receive a data set and post a NewPlotEvent to parent.

Parameters

- **data** – extrapolated data to be plotted
- **name** – Data's name to use for the legend

save_file (*filepath*, *state=None*)
Save data in provided state object.

Parameters

- **filepath** – path of file to write to
- **state** – invariant state

set_data (*data_list=None*)
receive a list of data and compute invariant

set_state (*state=None*, *datainfo=None*)
Call-back method for the state reader. This method is called when a .inv/.svs file is loaded.

Parameters **state** – State object

sas.sasgui.perspectives.invariant.invariant_details module

Invariant panel

class sas.sasgui.perspectives.invariant.invariant_details.**InvariantContainer**
Bases: wx._core.Object

This class stores some values resulting resulting from invariant calculations. Given the value of total invariant, this class can also determine the percentage of invariants resulting from extrapolation.

check_values ()
check the validity if invariant

compute_percentage ()
Compute percentage of each invariant

class sas.sasgui.perspectives.invariant.invariant_details.**InvariantDetailsPanel** (*parent=No id=- 1, qs- tar_contai ti- tle='Invari De- tails', size=(530, 430)*)
Bases: wx._windows.Dialog

This panel describes proportion of invariants

get_scale (*percentage*, *scale_name='scale'*)
Check scale receive in this panel.

on_close(*event*)
Close the current window

on_paint(*event*)
Draw the chart

set_color_bar()
Change the color for low and high bar when necessary

set_values()
Set value of txtcrtl

sas.sasgui.perspectives.invariant.invariant_panel module

This module provides the GUI for the invariant perspective panel

class sas.sasgui.perspectives.invariant.invariant_panel.**InvariantDialog**(*parent=None, id=1, graph=None, data=None, title='Invariant', base=None*)

Bases: wx._windows.Dialog

Invariant Dialog

class sas.sasgui.perspectives.invariant.invariant_panel.**InvariantPanel**(*parent, data=None, manager=None, *args, **kwds*)

Bases: wx.lib.scrolledpanel.ScrolledPanel, *sas.sasgui.guiframe.panel_base.PanelBase*

Main class defining the sizers (wx "panels") used to draw the Invariant GUI.

CENTER_PANE = True

clear_panel()
Clear panel to defaults, used by set_state of manager

compute_invariant(*event=None*)
compute invariant

display_details(*event*)
open another panel for more details on invariant calculation

get_background()
return the background textcrtl value as a float

get_bookmark_by_num(*num=None*)
Get the bookmark state given by number

: param num: the given bookmark number

get_contrast()
return the contrast textcrtl value as a float

get_data()

get_extrapolation_type(*low_q, high_q*)
get extrapolation type

get_high_qstar(*inv, high_q=False*)
get high qstar

`get_low_qstar`(*inv*, *npts_low*, *low_q=False*)
get low qstar

`get_porod_const`()
return the porod constant textcrtl value as a float

`get_qstar`(*inv*)
get qstar

`get_scale`()
return the scale textcrtl value as a float

`get_state`()

`get_state_by_num`(*state_num=None*)
Get the state given by number

: param state_num: the given state number

`get_surface`(*inv*, *contrast*, *porod_const*, *extrapolation*)
get surface value

`get_total_qstar`(*inv*, *extrapolation*)
get total qstar

`get_volume`(*inv*, *contrast*, *extrapolation*)
get volume fraction

`on_bookmark`(*event*)
Save the panel state in memory and add the list on the popup menu on bookmark context menu event

`on_help`(*event*)

Bring up the Invariant Documentation whenever the HELP button is clicked.

Calls DocumentationWindow with the path of the location within the documentation tree (after /doc/ ….". Note that when using old versions of Wx (before 2.9) and thus not the release version of installers, the help comes up at the top level of the file as webbrowser does not pass anything past the # to the browser when it is running "file:///…."

Parameters **`evt`** – Triggers on clicking the help button

`on_preview`(*event=None*)
Invoke report dialog panel

: param event: report button event

`on_redo`(*event=None*)
Go forward to the previous state

: param event: redo button event

`on_save`(*evt=None*)
Save invariant state into a file

`on_undo`(*event=None*)
Go back to the previous state

: param event: undo button event

`reset_panel`()
set the panel at its initial state.

`save_project`(*doc=None*)

return an xml node containing state of the panel that guiframe can write to file

`set_data`(*data*)
Set the data

set_extrapolation_high(*inv*, *high_q=False*)
return float value necessary to compute invariant a high q

set_extrapolation_low(*inv*, *low_q=False*)
return float value necessary to compute invariant a low q

set_manager(*manager*)
set value for the manager

set_message()
Display warning message if available

set_state(*state=None*, *data=None*)
set state when loading it from a .inv/.svs file

window_caption = 'Invariant'

window_name = 'Invariant'

class sas.sasgui.perspectives.invariant.invariant_panel.**InvariantWindow**(*parent=None*, *id=1*, *graph=None*, *data=None*, *title='Invariant'*, *base=None*)

Bases: `wx._windows.Frame`

Invariant Window

class sas.sasgui.perspectives.invariant.invariant_panel.**MyApp**(*redirect=False*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*)

Bases: `wx._core.App`

Test App

OnInit()
Init

sas.sasgui.perspectives.invariant.invariant_state module

State class for the invariant UI

class sas.sasgui.perspectives.invariant.invariant_state.**InvariantState**
Bases: `object`

Class to hold the state information of the InversionControl panel.

clone_state()
deepcopy the state

fromXML(*file=None*, *node=None*)
Load invariant states from a file

: param file: .inv file : param node: node of a XML document to read from

set_plot_state(*extra_high=False*, *extra_low=False*)
Build image state that wx.html understand by plotting, putting it into wx.FileSystem image object

: extrap_high,extra_low: low/high extrapolations are possible extra-plots

set_report_string()
Get the values (strings) from __str__ for report

set_saved_state(*name*, *value*)
Set the state list

: param name: name of the state component : param value: value of the state component

toXML(*file='inv_state.inv'*, *doc=None*, *entry_node=None*)
Writes the state of the InversionControl panel to file, as XML.

Compatible with standalone writing, or appending to an already existing XML document. In that case, the XML document is required. An optional entry node in the XML document may also be given.

: param file: file to write to : param doc: XML document object [optional] : param entry_node: XML node within the document at which we will append the data [optional]

class sas.sasgui.perspectives.invariant.invariant_state.**Reader**(*call_back*, *cansas=True*)
Bases: `sas.sascalc.dataloader.readers.cansas_reader.Reader`

Class to load a .inv invariant file

ext = ['.inv', '.INV', '.svs', 'SVS']

get_state()

read(*path*)
Load a new invariant state from file

: param path: file path : return: None

type = ['Invariant file (*.inv)|*.inv', 'SASView file (*.svs)|*.svs']

type_name = 'Invariant'

write(*filename*, *datainfo=None*, *invstate=None*)
Write the content of a Data1D as a CanSAS XML file

: param filename: name of the file to write : param datainfo: Data1D object : param invstate: InvariantState object

write_toXML(*datainfo=None*, *state=None*)
Write toXML, a helper for write()

: return: xml doc

sas.sasgui.perspectives.invariant.invariant_widgets module

class sas.sasgui.perspectives.invariant.invariant_widgets.**DataDialog**(*data_list*, *parent=None*, *text=''*, **args*, ***kwds*)
Bases: `wx._windows.Dialog`

Allow file selection at loading time

get_data()
return the selected data

class sas.sasgui.perspectives.invariant.invariant_widgets.**DialogPanel**(**args*, ***kwds*)
Bases: `wx.lib.scrolledpanel.ScrolledPanel`

class sas.sasgui.perspectives.invariant.invariant_widgets.**InvTextCtrl**(**args*, ***kwds*)
Bases: `wx._controls.TextCtrl`

Text control for model and fit parameters. Binds the appropriate events for user interactions.

class sas.sasgui.perspectives.invariant.invariant_widgets.**OutputTextCtrl**(**args*, ***kwds*)

Bases: `wx._controls.TextCtrl`

Text control used to display outputs. No editing allowed. The background is grayed out. User can't select text.

sas.sasgui.perspectives.invariant.report_dialog module

Dialog report panel to show and summarize the results of the invariant calculation.

class sas.sasgui.perspectives.invariant.report_dialog.**ReportDialog**(*report_list*, **args*, ***kwds*)

Bases: *sas.sasgui.guiframe.report_dialog.BaseReportDialog*

The report dialog box.

onSave(*event=None*)
Save

Module contents

sas.sasgui.perspectives.invariant.**data_files**()
Return the data files associated with media invariant.

The format is a list of (directory, [files…]) pairs which can be used directly in setup(…,data_files=…) for setup.py.

sas.sasgui.perspectives.invariant.**get_data_path**(*media*)

sas.sasgui.perspectives.pr package

Submodules

sas.sasgui.perspectives.pr.explore_dialog module

Dialog panel to explore the P(r) inversion results for a range of D_max value. User picks a number of points and a range of distances, then can toggle between inversion outputs and see their distribution as a function of D_max.

class sas.sasgui.perspectives.pr.explore_dialog.**ExploreDialog**(*pr_state*, *nfunc*, **args*, ***kwds*)

Bases: `wx._windows.Dialog`

The explorer dialog box. This dialog is meant to be invoked by the InversionControl class.

class Event

Bases: `object`

Class that holds the content of the form

dmax = 0

dmin = 0

npts = 0

send_focus_to_datapanel(*name*)
The GUI manager sometimes calls this method TODO: refactor this

set_plot_unfocus()
Not implemented

class sas.sasgui.perspectives.pr.explore_dialog.**OutputPlot**(*d_min*, *d_max*, *parent*, *id=-1*, *color=None*, *dpi=None*, *style=0*, ***kwargs*)
Bases: *sas.sasgui.plottools.PlotPanel.PlotPanel*

Plot panel used to show the selected results as a function of D_max

onContextMenu(*event*)
Default context menu for the plot panel

TODO Would be nice to add printing and log/linear scales. The current version of plottools no longer plays well with plots outside of guiframe. Guiframe team needs to fix this.

window_caption = 'D Explorer'

class sas.sasgui.perspectives.pr.explore_dialog.**Results**
Bases: object

Class to hold the inversion output parameters as a function of D_max

sas.sasgui.perspectives.pr.inversion_panel module

class sas.sasgui.perspectives.pr.inversion_panel.**InversionControl**(*parent*, *id=-1*, *plots=None*, ***kwargs*)
Bases: wx.lib.scrolledpanel.ScrolledPanel, *sas.sasgui.guiframe.panel_base.PanelBase*

CENTER_PANE = True

clear_panel()

get_data()

get_state()
Get the current state

: return: state object

on_help(*event*)

Bring up the P(r) Documentation whenever the HELP button is clicked.

Calls DocumentationWindow with the path of the location within the documentation tree (after /doc/ ....". Note that when using old versions of Wx (before 2.9) and thus not the release version of installers, the help comes up at the top level of the file as webbrowser does not pass anything past the # to the browser when it is running "file:///...."

Parameters evt – Triggers on clicking the help button

on_reset(*event=None*)
Resets inversion parameters

on_save(*evt=None*)
Method used to create a memento of the current state

Returns state object

oscillation_max = 1.5

save_project (*doc=None*)

return an xml node containing state of the panel that guiframe can write to file

set_manager (*manager*)

set_state (*state*)

Set the state of the panel and inversion problem to the state passed as a parameter. Execute the inversion immediately after filling the controls.

Parameters **state** – InversionState object

window_caption = 'P(r) control panel'

window_name = 'pr_control'

class sas.sasgui.perspectives.pr.inversion_panel.**PrDistDialog** (*parent*, *id*)

Bases: `wx._windows.Dialog`

Property dialog to let the user change the number of points on the P(r) plot.

get_content ()

Return the content of the dialog. At this point the values have already been checked.

set_content (*npts*)

Initialize the content of the dialog.

sas.sasgui.perspectives.pr.inversion_state module

Handling of P(r) inversion states

class sas.sasgui.perspectives.pr.inversion_state.**InversionState**

Bases: `object`

Class to hold the state information of the InversionControl panel.

fromXML (*file=None*, *node=None*)

Load a P(r) inversion state from a file

Parameters

- **file** – .prv file
- **node** – node of a XML document to read from

toXML (*file='pr_state.prv'*, *doc=None*, *entry_node=None*)

Writes the state of the InversionControl panel to file, as XML.

Compatible with standalone writing, or appending to an already existing XML document. In that case, the XML document is required. An optional entry node in the XML document may also be given.

Parameters

- **file** – file to write to
- **doc** – XML document object [optional]
- **entry_node** – XML node within the XML document at which we will append the data [optional]

class sas.sasgui.perspectives.pr.inversion_state.**Reader** (*call_back*, *cansas=True*)

Bases: `sas.sascalc.dataloader.readers.cansas_reader.Reader`

Class to load a .prv P(r) inversion file

ext = ['.prv', '.PRV', '.svs', '.SVS']

read (*path*)

Load a new P(r) inversion state from file

Parameters **`path`** – file path

Returns None

`type = ['P(r) files (*.prv)|*.prv', 'SASView files (*.svs)|*.svs']`

`type_name = 'P(r)'`

`write` (*filename*, *datainfo=None*, *prstate=None*)
Write the content of a Data1D as a CanSAS XML file

Parameters

- **`filename`** – name of the file to write
- **`datainfo`** – Data1D object
- **`prstate`** – InversionState object

`write_toXML` (*datainfo=None*, *state=None*)
Write toXML, a helper for write()

: return: xml doc

sas.sasgui.perspectives.pr.pr module

P(r) perspective for SasView

class `sas.sasgui.perspectives.pr.pr.`**`Plugin`**
Bases: *`sas.sasgui.guiframe.plugin_base.PluginBase`*

P(r) inversion perspective

`DEFAULT_ALPHA = 0.0001`

`DEFAULT_DMAX = 140.0`

`DEFAULT_NFUNC = 10`

`delete_data` (*data_id*)
delete the data association with prview

`estimate_file_inversion` (*alpha*, *nfunc*, *d_max*, *data*, *path=None*, *q_min=None*, *q_max=None*, *bck=False*, *height=0*, *width=0*)
Estimate parameters for inversion

`estimate_plot_inversion` (*alpha*, *nfunc*, *d_max*, *q_min=None*, *q_max=None*, *est_bck=False*, *bck_val=0*, *height=0*, *width=0*)
Estimate parameters from plotted data

`get_context_menu` (*plotpanel=None*)
Get the context menu items available for P(r)

Parameters **`graph`** – the Graph object to which we attach the context menu

Returns a list of menu items with call-back function

`get_data` ()
Returns the current data

`get_npts` ()
Returns the number of points in the I(q) data

`get_panels` (*parent*)
Create and return a list of panel objects

`help` (*evt*)
Show a general help dialog.

TODO replace the text with a nice image

load(*data*)
Load data. This will eventually be replaced by our standard DataLoader class.

load_abs(*path*)
Load an IGOR .ABS reduced file

Parameters **path** – file path

Returns x, y, err vectors

load_columns(*path='sphere_60_q0_2.txt'*)
Load 2- or 3- column ascii

perform_estimate()
Perform parameter estimation

perform_estimateNT()
Perform parameter estimation

perform_inversion()
Perform inversion

post_init()
Post initialization call back to close the loose ends [Somehow openGL needs this call]

pr_theory(*r*, *R*)
Return P(r) of a sphere for a given R For test purposes

save_data(*filepath*, *prstate=None*)
Save data in provided state object.

TODO move the state code away from inversion_panel and move it here. Then remove the "prstate" input and make this method private.

Parameters

- **filepath** – path of file to write to
- **prstate** – P(r) inversion state

set_data(*data_list=None*)
receive a list of data to compute pr

set_state(*state=None*, *datainfo=None*)
Call-back method for the inversion state reader. This method is called when a .prv file is loaded.

Parameters

- **state** – InversionState object
- **datainfo** – Data1D object [optional]

setup_file_inversion(*alpha*, *nfunc*, *d_max*, *data*, *path=None*, *q_min=None*, *q_max=None*, *bck=False*, *height=0*, *width=0*)
Set up inversion

setup_plot_inversion(*alpha*, *nfunc*, *d_max*, *q_min=None*, *q_max=None*, *est_bck=False*, *bck_val=0*, *height=0*, *width=0*)
Set up inversion from plotted data

show_data(*path=None*, *data=None*, *reset=False*)
Show data read from a file

Parameters

- **path** – file path
- **reset** – if True all other plottables will be cleared

show_iq(*out*, *pr*, *q=None*)
Display computed I(q)

show_pr (*out*, *pr*, *cov=None*)

show_shpere (*x*, *radius=70.0*, *x_range=70.0*)

start_thread ()
Start a calculation thread

sas.sasgui.perspectives.pr.pr_thread module

class sas.sasgui.perspectives.pr.pr_thread.**CalcPr** (*pr*, *nfunc=5*, *error_func=None*, *completefn=None*, *updatefn=None*, *yieldtime=0.01*, *worktime=0.01*)

Bases: `sas.sascalc.data_util.calcthread.CalcThread`

Compute P(r)

compute ()
Perform P(r) inversion

class sas.sasgui.perspectives.pr.pr_thread.**EstimateNT** (*pr*, *nfunc=5*, *error_func=None*, *completefn=None*, *updatefn=None*, *yieldtime=0.01*, *worktime=0.01*)

Bases: `sas.sascalc.data_util.calcthread.CalcThread`

compute ()
Calculates the estimate

isquit ()

class sas.sasgui.perspectives.pr.pr_thread.**EstimatePr** (*pr*, *nfunc=5*, *error_func=None*, *completefn=None*, *updatefn=None*, *yieldtime=0.01*, *worktime=0.01*)

Bases: `sas.sascalc.data_util.calcthread.CalcThread`

Estimate P(r)

compute ()
Calculates the estimate

sas.sasgui.perspectives.pr.pr_widgets module

Text controls for input/output of the main PrView panel

class sas.sasgui.perspectives.pr.pr_widgets.**DataDialog** (*data_list*, *parent=None*, *text=''*, **args*, ***kwds*)

Bases: `wx._windows.Dialog`

Allow file selection at loading time

get_data ()
return the selected data

class sas.sasgui.perspectives.pr.pr_widgets.**DataFileTextCtrl** (**args*, ***kwds*)

Bases: `sas.sasgui.perspectives.pr.pr_widgets.OutputTextCtrl`

Text control used to display only the file name given a full path.

TODO now that we no longer choose the data file from the panel, it's no longer necessary to pass around the file path. That code should be refactored away and simplified.

GetValue()
Return the full path

SetValue(*value*)
Sets the file name given a path

class sas.sasgui.perspectives.pr.pr_widgets.**DialogPanel**(**args*, ***kwds*)
Bases: wx.lib.scrolledpanel.ScrolledPanel

class sas.sasgui.perspectives.pr.pr_widgets.**OutputTextCtrl**(**args*, ***kwds*)
Bases: wx._controls.TextCtrl

Text control used to display outputs. No editing allowed. The background is grayed out. User can't select text.

class sas.sasgui.perspectives.pr.pr_widgets.**PrTextCtrl**(**args*, ***kwds*)
Bases: wx._controls.TextCtrl

Text control for model and fit parameters. Binds the appropriate events for user interactions.

sas.sasgui.perspectives.pr.pr_widgets.**load_error**(*error=None*)
Pop up an error message.

Parameters **error** – details error message to be displayed

Module contents

Module contents

sas.sasgui.plottools package

Submodules

sas.sasgui.plottools.BaseInteractor module

sas.sasgui.plottools.LabelDialog module

class sas.sasgui.plottools.LabelDialog.**LabelDialog**(*parent*, *id*, *title*, *label*)
Bases: wx._windows.Dialog

getText()

sas.sasgui.plottools.LineModel module

Provide Line function (y= Ax + B). Until July 10, 2016 this function provided (y= A + Bx). This however was contrary to all the other code using it which assumed (y= mx+b) or in this nomenclature (y=Ax + B). This lead to some contortions in the code and worse incorrect calculations until now for at least some of the functions. This seemed the easiest to fix particularly since this function should disappear in a future iteration (see notes in fitDialog)

PDB July 10, 2016

class sas.sasgui.plottools.LineModel.**LineModel**
Bases: object

Class that evaluates a linear model.

f(x) = Ax + B

List of default parameters: A = 1.0 B = 1.0

getParam(*name*)
Return parameter value

run(*x=0.0*)
Evaluate the model

Parameters **x** – simple value

Returns (Line value)

Note: This is the function called by fitDialog to calculate the the y(xmin) and y(xmax), but the only difference between this and runXY is when the if statement is true. I however cannot see what that function is for. It needs to be documented here or removed. PDB 7/10/16

runXY(*x=0.0*)
Evaluate the model.

Parameters **x** – simple value

Returns Line value

Note: This is to be what is called by fitDialog for the actual fit the only difference between this and run is when the if statement is true. I however cannot see what that function is for. It needs to be documented here or removed. PDB 7/10/16

setParam(*name*, *value*)
Set parameter value

sas.sasgui.plottools.PlotPanel module

Plot panel.

class sas.sasgui.plottools.PlotPanel.**NoRepaintCanvas**(**args*, ***kwargs*)
Bases: `matplotlib.backends.backend_wxagg.FigureCanvasWxAgg`

We subclass FigureCanvasWxAgg, overriding the _onPaint method, so that the draw method is only called for the first two paint events. After that, the canvas will only be redrawn when it is resized.

class sas.sasgui.plottools.PlotPanel.**PlotPanel**(*parent*, *id=-1*, *xtransform=None*, *ytransform=None*, *scale='log_{10}'*, *color=None*, *dpi=None*, ***kwargs*)
Bases: `wx._windows.Panel`

The PlotPanel has a Figure and a Canvas. OnSize events simply set a flag, and the actually redrawing of the figure is triggered by an Idle event.

ChangeLegendLoc(*label*)
Changes legend loc based on user input

OnCopyFigureMenu(*evt*)
Copy the current figure to clipboard

On_Paint(*event*)

SetColor(*rgbtuple*)
Set figure and canvas colours to be the same

add_toolbar()
add toolbar

clear()
Reset the plot

curve(*x*, *y*, *dy=None*, *color=0*, *symbol=0*, *label=None*)
Draw a line on a graph, possibly with confidence intervals.

cusor_line(*event*)

draw()
Where the actual drawing happens

get_loc_label()
Associates label to a specific legend location

get_xscale()

Returns x-axis scale

get_yscale()

Returns Y-axis scale

image(*data*, *qx_data*, *qy_data*, *xmin*, *xmax*, *ymin*, *ymax*, *zmin*, *zmax*, *color=0*, *symbol=0*, *markersize=0*, *label='data2D'*, *cmap=<matplotlib.colors.LinearSegmentedColormap object>*)
Render the current data

interactive_curve(*x*, *y*, *dy=None*, *name="*, *color=0*, *symbol=0*, *zorder=1*, *id=None*, *label=None*)
Draw markers with error bars

interactive_points(*x*, *y*, *dx=None*, *dy=None*, *name="*, *color=0*, *symbol=0*, *markersize=5*, *zorder=1*, *id=None*, *label=None*, *hide_error=False*)
Draw markers with error bars

is_zoomed

legend_picker(*legend*, *event*)
Pick up the legend patch

linear_plottable_fit(*plot*)
when clicking on linear Fit on context menu, display Fitting Dialog

Parameters **plot** – PlotPanel owning the graph

onChangeCaption(*event*)

onChangeLegendLoc(*event*)
Changes legend loc based on user input

onContextMenu(*event*)
Default context menu for a plot panel

onFitDisplay(*tempx*, *tempy*, *xminView*, *xmaxView*, *xmin*, *xmax*, *func*)
Add a new plottable into the graph .In this case this plottable will be used to fit some data

Parameters

- **tempx** – The x data of fit line
- **tempy** – The y data of fit line
- **xminView** – the lower bound of fitting range
- **xminView** – the upper bound of fitting range
- **xmin** – the lowest value of data to fit to the line
- **xmax** – the highest value of data to fit to the line

onFitting(*event*)
when clicking on linear Fit on context menu , display Fitting Dialog

onGridOnOff(*gridon_off*)
Allows ON/OFF Grid

onLeftDown(*event*)
left button down and ready to drag

onLeftUp(*event*)
Dragging is done

onLegend(*legOnOff*)
Toggles whether legend is visible/not visible

onMouseMotion(*event*)
check if the left button is press and the mouse in moving. computer delta for x and y coordinates and then calls draghelper to perform the drag

onPick(*event*)
On pick legend

onPrint(*event=None*)

onPrinterPreview(*event=None*)
Matplotlib camvas can no longer print itself. Thus need to do everything ourselves: need to create a printpreview frame to to see the preview but needs a parent frame object. Also needs a printout object (just as any printing task).

onPrinterSetup(*event=None*)

onResetGraph(*event*)
Reset the graph by plotting the full range of data

onSaveImage(*evt*)
Implement save image

onToolContextMenu(*event*)
ContextMenu from toolbar

Parameters `event` – toolbar event

onWheel(*event*)
Process mouse wheel as zoom events

Parameters `event` – Wheel event

on_kill_focus(*event*)
Reset the panel color

on_set_focus(*event*)
Send to the parenet the current panel on focus

plottable_selected(*id*)
Called to register a plottable as selected

points(*x*, *y*, *dx=None*, *dy=None*, *color=0*, *symbol=0*, *marker_size=5*, *label=None*, *id=None*, *hide_error=False*)
Draw markers with error bars

properties(*prop*)
Set some properties of the graph. The set of properties is not yet determined.

remove_legend(*ax=None*)
Remove legend for ax or the current axes.

render()
Commit the plot after all objects are drawn

resetFitView()
For fit Dialog initial display

returnTrans ()
Return values and labels used by Fit Dialog

schedule_full_draw (*func='append'*)
Put self in schedule to full redraw list

setTrans (*xtrans*, *ytrans*)

Parameters

- **xtrans** – set x transformation on Property dialog
- **ytrans** – set y transformation on Property dialog

set_legend_alpha (*alpha=1*)
Set legend alpha

set_resizing (*resizing=False*)
Set the resizing (True/False)

set_selected_from_menu (*menu*, *id*)
Set selected_plottable from context menu selection

Parameters

- **menu** – context menu item
- **id** – menu item id

set_xscale (*scale='linear'*)
Set the scale on x-axis

Parameters **scale** – the scale of x-axis

set_yscale (*scale='linear'*)
Set the scale on Y-axis

Parameters **scale** – the scale of y-axis

xaxis (*label*, *units*, *font=None*, *color='black'*, *t_font=None*)
xaxis label and units.

Axis labels know about units.

We need to do this so that we can detect when axes are not commesurate. Currently this is ignored other than for formatting purposes.

yaxis (*label*, *units*, *font=None*, *color='black'*, *t_font=None*)
yaxis label and units.

sas.sasgui.plottools.PlotPanel.**show_tree** (*obj*, *d=0*)
Handy function for displaying a tree of graph objects

sas.sasgui.plottools.PropertyDialog module

class sas.sasgui.plottools.PropertyDialog.**Properties** (*parent*, *id=-1*, *title='Select the scale of the graph'*)
Bases: wx._windows.Dialog

getValues ()

setValues (*x*, *y*, *view*)

viewChanged (*event*)

sas.sasgui.plottools.RangeDialog module

class `sas.sasgui.plottools.RangeDialog.`**`RangeDialog`**(*parent*, *id*, *title='Set Graph Range'*)

Bases: `wx._windows.Dialog`

`GetXRange`()

`GetYRange`()

`SetXRange`(*x_range*)

`SetYRange`(*y_range*)

sas.sasgui.plottools.SimpleFont module

This software was developed by Institut Laue-Langevin as part of Distributed Data Analysis of Neutron Scattering Experiments (DANSE).

Copyright 2012 Institut Laue-Langevin

class `sas.sasgui.plottools.SimpleFont.`**`SimpleFont`**(*parent*, *id*, *title*)

Bases: `wx._windows.Dialog`

`InitUI`()

`get_font`()

`get_ticklabel_check`()

Get tick label check value

`set_default_font`(*font*)

`set_ticklabel_check`(*check=False*)

Set tick label check value

sas.sasgui.plottools.SizeDialog module

class `sas.sasgui.plottools.SizeDialog.`**`SizeDialog`**(*parent*, *id*, *title*)

Bases: `wx._windows.Dialog`

`getText`()

Get text typed

sas.sasgui.plottools.TextDialog module

class `sas.sasgui.plottools.TextDialog.`**`TextDialog`**(*parent*, *id*, *title*, *label=''*, *unit=None*)

Bases: `wx._windows.Dialog`

`getColor`()

Returns font size for the text box

`getFamily`()

Returns font family for the text box

`getSize`()

Returns font size for the text box

`getStyle`()

Returns font tyle for the text box

`getText`()

Returns text string as input by user.

getTickLabel()
Bool for use on tick label

getUnit()
Returns unit string as input by user.

getWeight()
Returns font weight for the text box

on_color(*event*)
Set the color

on_family(*event*)
Set the family

on_size(*event*)
Set the size

on_style(*event*)
Set the style

on_tick_label(*event*)
Set the font for tick label

on_weight(*event*)
Set the weight

sas.sasgui.plottools.arrow3d module

Module that draws multiple arrows in 3D coordinates

class sas.sasgui.plottools.arrow3d.**Arrow3D**(*base*, *xs*, *ys*, *zs*, *colors*, **args*, ***kwargs*)
Bases: `matplotlib.patches.FancyArrowPatch`

Draw 3D arrow

draw(*renderer*, *rasterized=True*)
Drawing actually happens here

on_left_down(*event*)
Mouse left-down event

on_left_up(*event*)
Mouse left up event

sas.sasgui.plottools.binder module

Extension to MPL to support the binding of artists to key/mouse events.

class sas.sasgui.plottools.binder.**BindArtist**(*figure*)
Bases: `object`

alt = False

clear(*h1*, *h2*, *...*)
Remove connections for artists h1, h2, . . .

Use clearall() to reset all connections.

clearall()
Clear connections to all artists.

Use clear(h1,h2,. . .) to reset specific artists.

control = False

dclick_threshhold = 0.25

disconnect()
In case we need to disconnect from the canvas…

events = ['enter', 'leave', 'motion', 'click', 'dclick', 'drag', 'release', 'scroll

meta = False

shift = False

trigger(*actor*, *action*, *ev*)
Trigger a particular event for the artist. Fallback to axes, to figure, and to 'all' if the event is not processed.

class `sas.sasgui.plottools.binder.`**Selection**(*artist=None*, *prop={}*)
Bases: `object`

Store and compare selections.

artist = None

prop = {}

sas.sasgui.plottools.canvas module

This module implements a faster canvas for plotting. it ovewrites some matplolib methods to allow printing on sys.platform=='win32'

class `sas.sasgui.plottools.canvas.`**FigureCanvas**(**args*, ***kw*)
Bases: `matplotlib.backends.backend_wxagg.FigureCanvasWxAgg`

Add features to the wx agg canvas for better support of AUI and faster plotting.

draw(*drawDC=None*)
Render the figure using agg.

draw_idle(**args*, ***kwargs*)
Render after a delay if no other render requests have been made.

scroll_event(*x*, *y*, *step=1*, *guiEvent=None*)
Backend derived classes should call this function on any scroll wheel event. x,y are the canvas coords: 0,0 is lower, left. button and key are as defined in MouseEvent

set_panel(*panel*)
Set axes

set_resizing(*resizing=False*)
Setting the resizing

`sas.sasgui.plottools.canvas.`**OnPrintPage**(*self*, *page*)
override printPage of matplotlib

`sas.sasgui.plottools.canvas.`**draw_image**(*self*, *x*, *y*, *im*, *bbox*, *clippath=None*, *clippath_trans=None*)
Draw the image instance into the current axes;

Parameters

- **x** – is the distance in pixels from the left hand side of the canvas.
- **y** – the distance from the origin. That is, if origin is upper, y is the distance from top. If origin is lower, y is the distance from bottom
- **im** – the class'matplotlib._image.Image' instance
- **bbox** – a class *matplotlib.transforms.Bbox* instance for clipping, or None

`sas.sasgui.plottools.canvas.`**select**(*self*)

`sas.sasgui.plottools.canvas.`**unselect**(*self*)

sas.sasgui.plottools.config module

sas.sasgui.plottools.convert_units module

Convert units to strings that can be displayed This is a cleaned up version of unitConverter.py

`sas.sasgui.plottools.convert_units.`**`convert_unit`**(*power*, *unit*)
Convert units to strings that can be displayed

sas.sasgui.plottools.fitDialog module

class `sas.sasgui.plottools.fitDialog.`**`LinearFit`**(*parent*, *plottable*, *push_data*, *transform*, *title*)
Bases: `wx._windows.Dialog`

`checkFitValues`(*item*)
Check the validity of input values

`floatForwardTransform`(*x*)
transform a float.

`floatInvTransform`(*x*)
transform a float.It is used to determine the x.View min and x.View max for values not in x. Also used to properly calculate RgQmin, RgQmax and to update qmin and qmax in the linear range boxes on the panel.

`floatTransform`(*x*)
transform a float.It is use to determine the x. View min and x.View max for values not in x

`layout`()
Sets up the panel layout for the linear fit including all the labels, text entry boxes, and buttons.

`register_close`(*owner*)
Method to register the close event to a parent window that needs notification when the dialog is closed

Parameters **`owner`** – parent window

`setFitRange`(*xmin*, *xmax*, *xminTrans*, *xmaxTrans*)
Set fit parameters

`set_fit_region`(*xmin*, *xmax*)
Set the fit region :param xmin: minimum x-value to be included in fit :param xmax: maximum x-value to be included in fit

class `sas.sasgui.plottools.fitDialog.`**`MyApp`**(*redirect=False*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*)
Bases: `wx._core.App`

Test application

`OnInit`()
Test application initialization

`onFitDisplay`(*tempx*, *tempy*, *xminView*, *xmaxView*, *xmin*, *xmax*, *func*)
Test application dummy method

`returnTrans`()
Test application dummy method

`sas.sasgui.plottools.fitDialog.`**`format_number`**(*value*, *high=False*)
Return a float in a standardized, human-readable formatted string. This is used to output readable (e.g. x.xxxe-y) values to the panel.

sas.sasgui.plottools.fittings module

This module is used to fit a set of x,y data to a model passed to it. It is used to calculate the slope and intercepts for the linearized fits. Two things should be noted:

First, this fitting module uses the NLLSQ module of SciPy rather than a linear fit. This along with a few other modules could probably be removed if we move to a linear regression approach.

Second, this infrastructure does not allow for resolution smearing of the the models. Hence the results are not that accurate even for pinhole collimation of SANS but may be good for SAXS. It is completely wrong for slit smeared data.

class `sas.sasgui.plottools.fittings.`**`Parameter`**(*model*, *name*, *value=None*)

Bases: `object`

Class to handle model parameters - sets the parameters and their initial value from the model based to it.

`set`(*value*)

Set the value of the parameter

`sas.sasgui.plottools.fittings.`**`calcCommandline`**(*event*)

`sas.sasgui.plottools.fittings.`**`sasfit`**(*model*, *pars*, *x*, *y*, *err_y*, *qmin=None*, *qmax=None*)

Fit function

Parameters

- **`model`** – sas model object
- **`pars`** – list of parameters
- **`x`** – vector of x data
- **`y`** – vector of y data
- **`err_y`** – vector of y errors

sas.sasgui.plottools.plottable_interactor module

This module allows more interaction with the plot

class `sas.sasgui.plottools.plottable_interactor.`**`PointInteractor`**(*base*, *axes*, *color='black'*, *zorder=3*, *id="*)

Bases: `sas.sasgui.plottools.BaseInteractor._BaseInteractor`

`clear`()

`connect_markers`(*markers*)

Connect markers to callbacks

`curve`(*x*, *y*, *dy=None*, *color=0*, *symbol=0*, *zorder=10*, *label=None*, *width=2.0*)

`points`(*x*, *y*, *dx=None*, *dy=None*, *color=0*, *symbol=0*, *zorder=1*, *markersize=5*, *label=None*, *hide_error=False*)

`step`(*x*, *y*, *dy=None*, *color=0*, *symbol=0*, *zorder=1*, *label=None*, *width=2.0*)

`update`()

Update

`vline`(*x*, *y*, *dy=None*, *color=0*, *symbol=0*, *zorder=1*, *label=None*, *width=2.0*)

sas.sasgui.plottools.plottables module

Prototype plottable object support.

The main point of this prototype is to provide a clean separation between the style (plotter details: color, grids, widgets, etc.) and substance (application details: which information to plot). Programmers should not be dictating line colours and plotting symbols.

Unlike the problem of style in CSS or Word, where most paragraphs look the same, each line on a graph has to be distinguishable from its neighbours. Our solution is to provide parametric styles, in which a number of different classes of object (e.g., reflectometry data, reflectometry theory) representing multiple graph primitives cycle through a colour palette provided by the underlying plotter.

A full treatment would provide perceptual dimensions of prominence and distinctiveness rather than a simple colour number.

class `sas.sasgui.plottools.plottables.`**`Chisq`**(*chisq=None*)

Bases: `sas.sasgui.plottools.plottables.Plottable`

Chisq plottable plots the chisq

`render`(*plot*, ***kw*)

`setChisq`(*chisq*)

Set the chisq value.

class `sas.sasgui.plottools.plottables.`**`Data1D`**(*x*, *y*, *dx=None*, *dy=None*, *lam=None*, *dlam=None*)

Bases: `sas.sasgui.plottools.plottables.Plottable`

Data plottable: scatter plot of x,y with errors in x and y.

`changed`()

classmethod **`labels`**(*collection*)

Build a label mostly unique within a collection

`render`(*plot*, ***kw*)

Renders the plottable on the graph

class `sas.sasgui.plottools.plottables.`**`Data2D`**(*image=None*, *qx_data=None*, *qy_data=None*, *err_image=None*, *xmin=None*, *xmax=None*, *ymin=None*, *ymax=None*, *zmin=None*, *zmax=None*)

Bases: `sas.sasgui.plottools.plottables.Plottable`

2D data class for image plotting

`changed`()

classmethod **`labels`**(*collection*)

Build a label mostly unique within a collection

`render`(*plot*, ***kw*)

Renders the plottable on the graph

`setValues`(*datainfo=None*)

Use datainfo object to initialize data2D

Parameters **`datainfo`** – object

`set_zrange`(*zmin=None*, *zmax=None*)

`xaxis`(*label*, *unit*)

set x-axis

Parameters

- **`label`** – x-axis label

- **unit** – x-axis unit

yaxis (*label*, *unit*)
set y-axis

Parameters

- **label** – y-axis label
- **unit** – y-axis unit

zaxis (*label*, *unit*)
set z-axis

Parameters

- **label** – z-axis label
- **unit** – z-axis unit

class sas.sasgui.plottools.plottables.**Fit1D** (*data=None*, *theory=None*)
Bases: *sas.sasgui.plottools.plottables.Plottable*

Fit plottable: composed of a data line plus a theory line. This is treated like a single object from the perspective of the graph, except that it will have two legend entries, one for the data and one for the theory.

The color of the data and theory will be shared.

changed ()

render (*plot*, ***kw*)

class sas.sasgui.plottools.plottables.**Graph** (***kw*)
Bases: object

Generic plottables graph structure.

Plot styles are based on color/symbol lists. The user gets to select the list of colors/symbols/sizes to choose from, not the application developer. The programmer only gets to add/remove lines from the plot and move to the next symbol/color.

Another dimension is prominence, which refers to line sizes/point sizes.

Axis transformations allow the user to select the coordinate view which provides clarity to the data. There is no way we can provide every possible transformation for every application generically, so the plottable objects themselves will need to provide the transformations. Here are some examples from reflectometry:

```
independent: x -> f(x)
   monitor scaling: y -> M*y
   log:  y -> log(y if y > min else min)
   cos:  y -> cos(y*pi/180)
dependent:   x -> f(x,y)
   Q4:      y -> y*x^4
   fresnel: y -> y*fresnel(x)
coordinated: x,y = f(x,y)
   Q:    x -> 2*pi/L (cos(x*pi/180) - cos(y*pi/180))
         y -> 2*pi/L (sin(x*pi/180) + sin(y*pi/180))
reducing: x,y = f(x1,x2,y1,y2)
   spin asymmetry: x -> x1, y -> (y1 - y2)/(y1 + y2)
   vector net: x -> x1, y -> y1*cos(y2*pi/180)
```

Multiple transformations are possible, such as Q4 spin asymmetry

Axes have further complications in that the units of what are being plotted should correspond to the units on the axes. Plotting multiple types on the same graph should be handled gracefully, e.g., by creating a separate tab for each available axis type, breaking into subplots, showing multiple axes on the same plot, or generating inset plots. Ultimately the decision should be left to the user.

Graph properties such as grids/crosshairs should be under user control, as should the sizes of items such as axis fonts, etc. No direct access will be provided to the application.

Axis limits are mostly under user control. If the user has zoomed or panned then those limits are preserved even if new data is plotted. The exception is when, e.g., scanning through a set of related lines in which the user may want to fix the limits so that user can compare the values directly. Another exception is when creating multiple graphs sharing the same limits, though this case may be important enough that it is handled by the graph widget itself. Axis limits will of course have to understand the effects of axis transformations.

High level plottable objects may be composed of low level primitives. Operations such as legend/hide/show copy/paste, etc. need to operate on these primitives as a group. E.g., allowing the user to have a working canvas where they can drag lines they want to save and annotate them.

Graphs need to be printable. A page layout program for entire plots would be nice.

add(*plottable*, *color=None*)
Add a new plottable to the graph

changed()
Detect if any graphed plottables have changed

delete(*plottable*)
Remove an existing plottable from the graph

get(*key*)
Get the graph properties

get_plottable(*name*)
Return the plottable with the given name if it exists. Otherwise return None

get_range()
Return the range of all displayed plottables

isPlotted(*plottable*)
Return True is the plottable is already on the graph

render(*plot*)
Redraw the graph

replace(*plottable*)
Replace an existing plottable from the graph

reset()
Reset the graph.

reset_scale()
Resets the scale transformation data to the underlying data

returnPlottable()
This method returns a dictionary of plottables contained in graph It is just by Plotpanel to interact with the complete list of plottables inside the graph.

set(***kw*)
Set the graph properties

title(*name*)
Graph title

xaxis(*name*, *units*)
Properties of the x axis.

yaxis(*name*, *units*)
Properties of the y axis.

class sas.sasgui.plottools.plottables.**Plottable**
Bases: `object`

check_data_PlottableX()
Since no transformation is made for log10(x), check that no negative values is plot in log scale

check_data_PlottableY()
Since no transformation is made for log10(y), check that no negative values is plot in log scale

colors()
Return the number of colors need to render the object

custom_color = None

dx = None

dy = None

get_xaxis()
Return the units and name of x-axis

get_yaxis()
Return the units and name of y- axis

hidden = False

interactive = True

is_empty()
Returns True if there is no data stored in the plottable

classmethod labels(*collection*)
Construct a set of unique labels for a collection of plottables of the same type.

Returns a map from plottable to name.

markersize = 5

name = None

onFitRange(*xmin=None*, *xmax=None*)
It limits View data range to plot from min to max

Parameters

- **xmin** – the minimum value of x to plot.
- **xmax** – the maximum value of x to plot

onReset()
Reset x, y, dx, dy view with its parameters

render(*plot*)
The base class makes sure the correct units are being used for subsequent plottable.

For now it is assumed that the graphs are commensurate, and if you put a Qx object on a Temperature graph then you had better hope that it makes sense.

reset_view()
Reload view with new value to plot

returnValuesOfView()
Return View parameters and it is used by Fit Dialog

setLabel(*labelx*, *labely*)
It takes a label of the x and y transformation and set View parameters

Parameters

- **transx** – The label of x transformation is sent by Properties Dialog
- **transy** – The label of y transformation is sent Properties Dialog

set_View(*x*, *y*)
Load View

set_data(*x*, *y*, *dx=None*, *dy=None*)

short_name = None

transformView()
It transforms x, y before displaying

transformX(*transx*, *transdx*)
Receive pointers to function that transform x and dx and set corresponding View pointers

Parameters

- **transx** – pointer to function that transforms x
- **transdx** – pointer to function that transforms dx

transformY(*transy*, *transdy*)
Receive pointers to function that transform y and dy and set corresponding View pointers

Parameters

- **transy** – pointer to function that transforms y
- **transdy** – pointer to function that transforms dy

x = None

xaxis(*name*, *units*)
Set the name and unit of x_axis

Parameters

- **name** – the name of x-axis
- **units** – the units of x_axis

y = None

yaxis(*name*, *units*)
Set the name and unit of y_axis

Parameters

- **name** – the name of y-axis
- **units** – the units of y_axis

class sas.sasgui.plottools.plottables.**Text**(*text=None*, *xpos=0.5*, *ypos=0.9*, *name='text'*)
Bases: *sas.sasgui.plottools.plottables.Plottable*

getText(*text*)
Get the text string.

render(*plot*, ***kw*)

setText(*text*)
Set the text string.

set_x(*x*)
Set the x position of the text ACCEPTS: float

set_y(*y*)
Set the y position of the text ACCEPTS: float

class sas.sasgui.plottools.plottables.**Theory1D**(*x*, *y*, *dy=None*)
Bases: *sas.sasgui.plottools.plottables.Plottable*

Theory plottable: line plot of x,y with confidence interval y.

class sas.sasgui.plottools.plottables.**Transform**
Bases: object

Define a transform plugin to the plottable architecture.

Transforms operate on axes. The plottable defines the set of transforms available for it, and the axes on which they operate. These transforms can operate on the x axis only, the y axis only or on the x and y axes together.

This infrastructure is not able to support transformations such as log and polar plots as these require full control over the drawing of axes and grids.

A transform has a number of attributes.

name user visible name for the transform. This will appear in the context menu for the axis and the transform menu for the graph.

type operational axis. This determines whether the transform should appear on x,y or z axis context menus, or if it should appear in the context menu for the graph.

inventory (not implemented) a dictionary of user settable parameter names and their associated types. These should appear as keyword arguments to the transform call. For example, Fresnel reflectivity requires the substrate density: `{ 'rho':  type.Value(10e-6/units.angstrom**2) }` Supply reasonable defaults in the callback so that limited plotting clients work even though they cannot set the inventory.

class `sas.sasgui.plottools.plottables.`**`View`**(*x=None*, *y=None*, *dx=None*, *dy=None*)

Bases: `object`

Representation of the data that might include a transformation

`check_data_logX`()

Remove negative value in x vector to avoid plotting negative value of Log10

`check_data_logY`()

Remove negative value in y vector to avoid plotting negative value of Log10

`dx = None`

`dy = None`

`onFitRangeView`(*xmin=None*, *xmax=None*)

It limits View data range to plot from min to max

Parameters

- **`xmin`** – the minimum value of x to plot.
- **`xmax`** – the maximum value of x to plot

`onResetView`()

Reset x,y,dx and y in their full range and in the initial scale in case their previous range has changed

`returnXview`()

Return View x,y,dx,dy

`setTransformX`(*funcx*, *funcdx*)

Receive pointers to function that transform x and dx and set corresponding View pointers

Parameters

- **`transx`** – pointer to function that transforms x
- **`transdx`** – pointer to function that transforms dx

`setTransformY`(*funcy*, *funcdy*)

Receive pointers to function that transform y and dy and set corresponding View pointers

Parameters

- **`transx`** – pointer to function that transforms y
- **`transdx`** – pointer to function that transforms dy

`transform`(*x=None*, *y=None*, *dx=None*, *dy=None*)

Transforms the x,y,dx and dy vectors and stores the output in View parameters

Parameters

- **x** – array of x values
- **y** – array of y values
- **dx** – array of errors values on x
- **dy** – array of error values on y

x = None

y = None

sas.sasgui.plottools.plottables.**all**(*L*)

sas.sasgui.plottools.plottables.**any**(*L*)

sas.sasgui.plottools.plottables.**demo_plotter**(*graph*)

sas.sasgui.plottools.plottables.**sample_graph**()

sas.sasgui.plottools.toolbar module

This module overwrites matplotlib toolbar

class sas.sasgui.plottools.toolbar.**NavigationToolBar**(*canvas*, *parent=None*)
Bases: `matplotlib.backends.backend_wxagg.NavigationToolbar2WxAgg`

context_menu(*event*)
Default context menu for a plot panel

copy_figure(*event*)

on_menu(*event*)

print_figure(*event*)

class sas.sasgui.plottools.toolbar.**PlotPrintout**(*canvas*)
Bases: `wx._windows.Printout`

Create the wx.Printout object for matplotlib figure from the PlotPanel. Provides the required OnPrintPage and HasPage overrides. Other methods may be added/overriden in the future. :TODO: this needs LOTS of TLC .. but fixes immediate problem

GetPageInfo()
just sets the page to 1 - no flexibility for now

OnPrintPage(*page*)
Most rudimentry OnPrintPage override. instatiates a dc object, gets its size, gets the size of the figure object, scales it to the dc canvas size keeping the aspect ratio intact, then prints as bitmap

sas.sasgui.plottools.toolbar.**bind**(*actor*, *event*, *action*, ***kw*)

sas.sasgui.plottools.toolbar.**copy_image_to_clipboard**(*canvas*)

sas.sasgui.plottools.transform module

sas.sasgui.plottools.transform.**errFromX2**(*x*, *y=None*, *dx=None*, *dy=None*)
calculate error of sqrt(x)

Parameters

- **x** – float value
- **dx** – float value

`sas.sasgui.plottools.transform.`**`errFromX4`** (*x*, *y=None*, *dx=None*, *dy=None*)
calculate error of x^1/4

Parameters

- **`x`** – float value
- **`dx`** – float value

`sas.sasgui.plottools.transform.`**`errOneOverSqrtX`** (*x*, *y=None*, *dx=None*, *dy=None*)
Calculate error on 1/sqrt(x)

`sas.sasgui.plottools.transform.`**`errOneOverX`** (*x*, *y=None*, *dx=None*, *dy=None*)
calculate error on 1/x

`sas.sasgui.plottools.transform.`**`errToLog10X`** (*x*, *y=None*, *dx=None*, *dy=None*)
calculate error of Log(x)

Parameters

- **`x`** – float value
- **`dx`** – float value

`sas.sasgui.plottools.transform.`**`errToLogX`** (*x*, *y=None*, *dx=None*, *dy=None*)
calculate error of Log(x)

Parameters

- **`x`** – float value
- **`dx`** – float value

`sas.sasgui.plottools.transform.`**`errToLogXY`** (*x*, *y*, *dx=None*, *dy=None*)
calculate error of Log(xy)

`sas.sasgui.plottools.transform.`**`errToLogYX2`** (*y*, *x*, *dy=None*, *dx=None*)
calculate error of Log(yx**2)

`sas.sasgui.plottools.transform.`**`errToLogYX4`** (*y*, *x*, *dy=None*, *dx=None*)
error for ln(y*x^(4))

Parameters **`x`** – float value

`sas.sasgui.plottools.transform.`**`errToX`** (*x*, *y=None*, *dx=None*, *dy=None*)
calculate error of x**2

Parameters

- **`x`** – float value
- **`dx`** – float value

`sas.sasgui.plottools.transform.`**`errToX2`** (*x*, *y=None*, *dx=None*, *dy=None*)
calculate error of x**2

Parameters

- **`x`** – float value
- **`dx`** – float value

`sas.sasgui.plottools.transform.`**`errToX4`** (*x*, *y=None*, *dx=None*, *dy=None*)
calculate error of x**4

Parameters

- **`x`** – float value
- **`dx`** – float value

`sas.sasgui.plottools.transform.`**`errToX_pos`** (*x*, *y=None*, *dx=None*, *dy=None*)
calculate error of x**2

Parameters

- **x** – float value
- **dx** – float value

`sas.sasgui.plottools.transform.`**errToYX2**(*y*, *x*, *dy=None*, *dx=None*)

`sas.sasgui.plottools.transform.`**errToYX4**(*y*, *x*, *dy=None*, *dx=None*)
error for (y*x^(4))

Parameters **x** – float value

`sas.sasgui.plottools.transform.`**fromX2**(*x*, *y=None*)
This function is used to load value on Plottable.View Calculate square root of x

Parameters **x** – float value

`sas.sasgui.plottools.transform.`**fromX4**(*x*, *y=None*)
This function is used to load value on Plottable.View Calculate square root of x

Parameters **x** – float value

`sas.sasgui.plottools.transform.`**toLogX**(*x*, *y=None*)
This function is used to load value on Plottable.View calculate log x

Parameters **x** – float value

`sas.sasgui.plottools.transform.`**toLogXY**(*y*, *x*)
This function is used to load value on Plottable.View calculate log x

Parameters **x** – float value

`sas.sasgui.plottools.transform.`**toLogYX2**(*y*, *x*)

`sas.sasgui.plottools.transform.`**toLogYX4**(*y*, *x*)

`sas.sasgui.plottools.transform.`**toOneOverSqrtX**(*y*, *x=None*)

`sas.sasgui.plottools.transform.`**toOneOverX**(*x*, *y=None*)

`sas.sasgui.plottools.transform.`**toX**(*x*, *y=None*)
This function is used to load value on Plottable.View

Parameters **x** – Float value

Returns x

`sas.sasgui.plottools.transform.`**toX2**(*x*, *y=None*)
This function is used to load value on Plottable.View

Calculate x^(2)

Parameters **x** – float value

`sas.sasgui.plottools.transform.`**toX4**(*x*, *y=None*)
This function is used to load value on Plottable.View

Calculate x^(4)

Parameters **x** – float value

`sas.sasgui.plottools.transform.`**toX_pos**(*x*, *y=None*)
This function is used to load value on Plottable.View

Parameters **x** – Float value

Returns x

`sas.sasgui.plottools.transform.`**toYX2**(*y*, *x*)

`sas.sasgui.plottools.transform.`**toYX4**(*y*, *x*)

Module contents

Module contents

sas.sasview package

Submodules

sas.sasview.custom_config module

Application appearance custom configuration

sas.sasview.local_config module

Application settings

`sas.sasview.local_config.`**`printEVT`**`(`*message*`)`

sas.sasview.sasview module

Base module for loading and running the main SasView application.

`class` `sas.sasview.sasview.`**`SasView`**

Bases: `object`

Main class for running the SasView application

`check_sasmodels_compiler``()`

Checking c compiler for sasmodels and raises xcode command line tools for installation

`sas.sasview.sasview.`**`run_cli`**`()`

`sas.sasview.sasview.`**`run_gui`**`()`

__main__ method for loading and running SasView

`sas.sasview.sasview.`**`setup_logging`**`()`

`sas.sasview.sasview.`**`setup_mpl`**`(`*backend=None*`)`

`sas.sasview.sasview.`**`setup_sasmodels`**`()`

Prepare sasmodels for running within sasview.

`sas.sasview.sasview.`**`setup_wx`**`()`

sas.sasview.welcome_panel module

Welcome page

`class` `sas.sasview.welcome_panel.`**`ViewApp`**`(`*redirect=False*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*`)`

Bases: `wx._core.App`

Test application

`OnInit``()`

`class` `sas.sasview.welcome_panel.`**`WelcomeFrame`**`(`*parent*, *id*, *title*`)`

Bases: `wx._windows.Frame`

Test frame

class sas.sasview.welcome_panel.**WelcomePage**(*parent*, **args*, ***kwds*)
Bases: wx.lib.scrolledpanel.ScrolledPanel

Panel created like about box as a welcome page Shows product name, current version, authors, and link to the product page.

CENTER_PANE = True

set_data(*data=None*)

window_caption = 'Welcome panel'

window_name = 'default'

class sas.sasview.welcome_panel.**WelcomePanel**(*parent*, **args*, ***kwds*)
Bases: wx.aui.AuiNotebook, *sas.sasgui.guiframe.panel_base.PanelBase*

Panel created like about box as a welcome page Shows product name, current version, authors, and link to the product page.

CENTER_PANE = True

get_frame()

on_close_page(*event*)
Called when the welcome panel is closed

set_data(*data=None*)

set_frame(*frame*)

set_manager(*manager*)
the manager of the panel in this case the application itself

window_caption = 'Welcome panel'

window_name = 'default'

sas.sasview.wxcruft module

class sas.sasview.wxcruft.**CallLater**(*millis*, *callableObj*, **args*, ***kwargs*)
A convenience class for *wx.Timer*, that calls the given callable object once after the given amount of milliseconds, passing any positional or keyword args. The return value of the callable is availbale after it has been run with the *GetResult* method.

If you don't need to get the return value or restart the timer then there is no need to hold a reference to this object.

See *wx.CallAfter*

GetInterval()

GetResult()

HasRun()

Interval

IsRunning()

Notify()
The timer has expired so call the callable.

Restart(*millis=None*, **args*, ***kwargs*)
(Re)start the timer

Result

SetArgs(**args*, ***kwargs*)
(Re)set the args passed to the callable object. This is useful in conjunction with Restart if you want to schedule a new call to the same callable object but with different parameters.

Start(*millis=None*, **args*, ***kwargs*)
(Re)start the timer

Stop()
Stop and destroy the timer.

class `sas.sasview.wxcruft.`**FutureCall**(*millis*, *callableObj*, **args*, ***kwargs*)
Bases: `sas.sasview.wxcruft.CallLater`

A compatibility alias for *CallLater*.

`sas.sasview.wxcruft.`**NewId**()

class `sas.sasview.wxcruft.`**PyTimer**(*notify*, **args*, ***kw*)
Bases: `wx._misc.Timer`

Notify(*self*)

`sas.sasview.wxcruft.`**call_later_fix**()

`sas.sasview.wxcruft.`**trace_new_id**()

Module contents

Submodules

sas.logger_config module

class `sas.logger_config.`**SetupLogger**(*logger_name*)
Bases: `object`

Called at the beginning of run.py or sasview.py

config_development()

config_production()

Module contents

`sas.`**get_app_dir**()
The directory where the sasview application is found.

Returns the path to sasview if running in place or installed with setup. If the application is frozen, returns the parent directory of the application resources such as test files and images.

`sas.`**get_user_dir**()
The directory where the per-user configuration is stored.

Returns ~/.sasview, creating it if it does not already exist.

`sas.`**get_local_config**()
Loads the local config file.

`sas.`**get_custom_config**()
Setup the custom config dir and cat file

2.1.2 Sasmodels Developers Guide

Code Overview

Computational kernels

- `core`
- `modelinfo`
- `kernel`
- `product`
- `mixture`

At the heart of *sasmodels* are the individual computational kernels. These functions take a particular q value and a set of parameter values and return the expected scattering for that q. The instructions for writing a kernel are documented in *Writing a Plugin Model*. The source code for the kernels is stored in `models`.

The primary interface to the models is through `core`, which provides functions for listing available models, loading the model definition and compiling the model. Use `core.load_model()` to load in a model definition and compile it. This makes use of `core.load_model_info()` to load the model definition and `core.build_model()` to turn it into a computational kernel model `kernel.KernelModel`.

The `modelinfo.ModelInfo` class defines the properties of the model including the available model parameters `modelinfo.ParameterTable` with individual parameter attributes such as units and hard limits defined in `modelinfo.Parameter`.

The `product.ProductModel` and `mixture.MixtureModel` classes are derived models, created automatically for models with names like "hardsphere*sphere" and "cylinder+sphere".

Data loaders

- `data`

In order to test models a minimal set of data management routines is provided in `data`. In particular, it provides mock `data.Data1D` and `data.Data2D` classes which mimic those classes in *SasView*. The functions `data.empty_data1D()` and `data.empty_data2D()` are handy for creating containers with a particular set of q, Δq points which can later be evaluated, and `data.plot_theory()` to show the result. If *SasView* is available on the path then `data.load_data()` can be used to load any data type defined in *SasView*. The function `data.plot_data()` can plot that data alone without the theory value.

Kernel execution

- `resolution`
- `resolution2d`
- `sesans`
- `weights`
- `details`
- `direct_model`
- `bumps_model`
- `sasview_model`

To execute a computational kernel at a particular set of q values, the use `kernel.KernelModel.make_kernel()`, which returns a callable `kernel.Kernel` for that q vector (or a pair of q_x, q_y for 2-D datasets).

The calculated q values should include the measured data points as well as additional q values required to properly compute the q resolution function. The *Resolution* subclasses in `resolution` define the *q_calc* attribute for this purpose. These are `resolution.Perfect1D` for perfect resolution, `resolution.Pinhole1D` for the usual SANS pinhole aperture, `resolution.Slit1D` for the usual USANS slit aperture and `resolution2d.Pinhole2D` for 2-D pinhole data. In addition, `resolution2d.Slit2D` defines 1-D slit smeared data for oriented samples, which require calculation at particular q_x and q_y values instead of $|q|$ as is the case for orientationally averaged USANS. The `sesans.SesansTransform` class acts like a 1-D resolution, having a *q_calc* attribute that defines the calculated q values for the SANS models that get converted to spin-echo values by the `sesnas.SesansTransform.apply()` method.

Polydispersity is defined by `weights.Dispersion` classes, `weights.RectangleDispersion`, `weights.ArrayDispersion`, `weights.LogNormalDispersion`, `weights.GaussianDispersion`, `weights.SchulzDispersion`. The `weights.get_weights()` function creates a dispersion object of the class matching `weights.Dispersion.type`, and calls it with the current value of the parameter. This returns a vector of values and weights for a weighted average polydispersity.

In order to call the `kernel.Kernel`, the values and weights for all parameters must be composed into a `details.CallDetails` object. This is a compact vector representation of the entire polydispersity loop that can be passed easily to the kernel. Additionally, the magnetic parameters must be converted from polar to cartesian coordinates. This work is done by the `details.make_kernel_args()` function, which returns values that can be sent directly to the kernel. It uses `details.make_details()` to set the details object and `details.convert_magnetism()` for the coordinate transform.

In the end, making a simple theory function evaluation requires a lot of setup. To make calling them a little easier, the *DirectModel* and *BumpsModel* interfaces are provided. See *Scripting Interface* for an example.

The `direct_model.DirectModel` interface accepts a data object and a kernel model. Within the class, the `direct_model.DataMixin._interpret_data()` method is called to query the data and set the resolution. The `direct_model.DataMixin._calc_theory()` takes a set of parameter values, builds the kernel arguments, calls the kernel, and applies the resolution function, returning the predicted value for the data q values. The `bumps_model.Experiment` class is like the DirectModel class, but it defines a Fitness class that can be handed directly to the bumps optimization and uncertainty analysis program.

The `sasview_model.SasviewModel` class defines a SasView 4.x compatible interface to the sasmodels definitions, allowing sasmodels to be used directly from SasView. Over time the SasView shim should disappear as SasView access the `modelinfo.ModelInfo` and computational kernels directly.

Kernel execution

- `kernelcl`
- `kerneldll`
- `kernelpy`
- `generate`

The kernel functions for the most part do not define polydispersity, resolution or magnetism directly. Instead sasmodels automatically applies these, calling the computation kernel as needed.

The outermost loop is the resolution calculation. For the 1-D case this computes a single vector of $I(q)$ values and applies the convolution to the resulting set. Since the same $I(q)$ vector is used to compute the convolution at each point, it can be precomputed before the convolution, and so the convolution is reasonably efficient. The 2-D case is not that efficient, and instead recomputes the entire shifted/scaled set of q_x, q_y values many times, or very many times depending on the accuracy requested.

Polydispersity is handled as a mesh over the polydisperse parameters. This is the next level of the loop. For C kernels run in a DLL or using OpenCL, the polydisperisty loop is generated separately for each model as C code. Inside the polydispersity loop there is a loop over the magnetic cross sections for magnetic models, updating the SLD parameters with the effective magnetic SLD for that particular q value. For OpenCL, each q value loops over the polydispersity mesh on a separate processor. For DLL, the outer loop cycles through polydispersity, and the inner loop distributes q values amongst the processors. Like the DLL, the Python kernel execution cycles over the polydisperse parameters and the magnetic cross sections, calling the computation kernel with a vector of q values. Assuming the kernel code accepts vectors, this can be fast enough (though it is painfully slow if not vectorized).

Further details are provided in the next section, *Calculator Interface*

Orientation and Numerical Integration

For 2d data from oriented anisotropic particles, the mean particle orientation is defined by angles θ, ϕ and Ψ, which are not in general the same as similarly named angles in many form factors. The wikipedia page on Euler angles (https://en.wikipedia.org/wiki/Euler_angles) lists the different conventions available. To quote: "Different authors may use different sets of rotation axes to define Euler angles, or different names for the same angles. Therefore, any discussion employing Euler angles should always be preceded by their definition."

We are using the z-y-z convention with extrinsic rotations Ψ-θ-ϕ for the particle orientation and x-y-z convention with extrinsic rotations Ψ-θ-ϕ for jitter, with jitter applied before particle orientation.

When computing the orientation dispersity integral, the weights for the individual points depends on the map projection used to translate jitter angles into latitude/longitude. The choice of projection is set by *sasmodels.generate.PROJECTION*, with the default *PROJECTION=1* for equirectangular and *PROJECTION=2* for sinusoidal. The more complicated Guyou and Postel projections are not implemented. See jitter.draw_mesh for details.

For numerical integration within form factors etc. sasmodels is mostly using Gaussian quadrature with 20, 76 or 150 points depending on the model. It also makes use of symmetries such as calculating only over one quadrant rather than the whole sphere. There is often a U-substitution replacing θ with $cos(\theta)$ which changes the limits of integration from 0 to $\pi/2$ to 0 to 1 and also conveniently absorbs the $sin(\theta)$ scale factor in the integration. This can cause confusion if checking equations to include in a paper or thesis! Most models use the same core kernel code expressed in terms of the rotated view (q_a, q_b, q_c) for both the 1D and the 2D models, but there are also historical quirks such as the parallelepiped model, which has a useless transformation representing $j_0(aq_a)$ as $j_0(bq_a a/b)$.

Useful testing routines include:

The *sascomp* utility is used to view and compare models with different parameters and calculation engines. The usual case is to simply plot a model that you are developing:

```
python sascomp path/to/model.py
```

Once the obvious problems are addressed, check the numerical precision across a variety of randomly generated inputs:

```
python sascomp -engine=single,double path/to/model.py -sets=10
```

You can compare different parameter values for the same or different models. For example when looking along the long axis of a cylinder ($\theta = 0$), dispersity in θ should be equivalent to dispersity in ϕ when $\phi = 90$:

```
python sascomp -2d cylinder theta=0 phi=0,90 theta_pd_type=rectangle \\
phi_pd_type=rectangle phi_pd=10,1 theta_pd=1,10 length=500 radius=10
```

It turns out that they are not because the equirectangular map projection weights the points by $\cos(\theta)$ so $\Delta\theta$ is not identical to $\Delta\phi$. Setting *PROJECTION=2* in `sasmodels.generate` helps somewhat. Postel would help even more in this case, though leading to distortions elsewhere. See `sasmodels.compare` for many more details.

sascomp -ngauss=n allows you to set the number of quadrature points used for the 1D integration for any model. For example, a carbon nanotube with length 10 μm and radius 1 nm is not computed correctly at high q:

```
python sascomp cylinder length=100000 radius=10 -ngauss=76,500 -double -highq
```

Note: ticket 702 gives some forms for long rods and thin disks that may be used in place of the integration, depending on q, radius and length; if the cylinder model is updated with these corrections then above call will show no difference.

explore/check1d.py uses sasmodels 1D integration and compares that with a rectangle distribution in θ and ϕ, with θ limits set to $\pm 90/\sqrt(3)$ and ϕ limits set to $\pm 180/\sqrt(3)$ [The rectangle weight function uses the fact that the distribution width column is labelled sigma to decide that the 1-σ width of a rectangular distribution needs to be multiplied by $\sqrt(3)$ to get the corresponding gaussian equivalent, or similar reasoning.] This should rotate the sample through the entire θ-ϕ surface according to the pattern that you see in jitter.py when you use 'rectangle' rather than 'gaussian' for its distribution without changing the viewing angle. In order to match the 1-D pattern for an arbitrary viewing angle on triaxial shapes, we need to integrate over θ, ϕ and Ψ.

sascomp -sphere=n uses the same rectangular distribution as check1d to compute the pattern of the q_x-q_y grid. This ought to produce a spherically symmetric pattern.

explore/precision.py investigates the accuracy of individual functions on the different execution platforms. This includes the basic special functions as well as more complex expressions used in scattering. In many cases the OpenCL function in sasmodels will use a polynomial approximation over part of the range to improve accuracy, as shown in:

```
python explore/precision.py 3j1/x
```

explore/realspace.py allows you to set up a Monte Carlo simulation of your model by sampling random points within and computing the 1D and 2D scattering patterns. This was used to check the core shell parallelepiped example. This is similar to the general sas calculator in sasview, though it uses different code.

sasmodels/jitter.py is for exploring different options for handling orientation and orientation dispersity. It uses *sasmodels/guyou.py* to generate the Guyou projection.

explore/asymint.py is a direct implementation of the 1D integration for a number of different models. It calculates the integral for a particular q using several different integration schemes, including mpmath with 100 bits of precision (33 digits), so you can use it to check the target values for the 1D model tests. This is not a general purpose tool; you will need to edit the file to change the model parameters. It does not currently apply the $u = cos(\theta)$ substitution that many models are using internally so the 76-point gaussian quadrature may not match the value produced by the eqivalent model from sasmodels.

explore/symint.py is for rotationally symmetric models (just cylinder for now), so it can compute an entire curve rather than a single q point. It includes code to compare the long cylinder approximation to cylinder.

explore/rpa.py is for checking different implementations of the RPA model against calculations for specific blends. This is a work in (suspended) progress.

There are a few modules left over in *explore* that can probably be removed. *angular_pd.py* was an early version of *jitter.py*. *sc.py* and *sc.c* was used to explore different calculations for paracrystal models; it has been absorbed into *asymint.py*. *transform_angles.py* translates orientation parameters from the SasView 3.x definition to sasmodels.

explore/angles.py generates code for the view and jitter transformations. Keep this around since it may be needed if we add new projections.

Testing

- `model_test`
- `compare`
- `compare_many`
- `rst2html`
- `list_pars`

Individual models should all have test values to make sure that the evaluation is correct. This is particularly important in the context of OpenCL since sasmodels doesn't control the compiler or the hardware, and since GPUs are notorious for preferring speed over precision. The tests can be run as a group using `model_test` as main:

```
$ python -m sasmodels.model_test all
```

Individual models can be listed instead of *all*, which is convenient when adding new models.

The `compare` module, usually invoked using *./sascomp* provides a rich interface for exploring model accuracy, execution speed and parameter ranges. It also allows different models to be compared. The `compare_many` module does batch comparisons, keeping a list of the particular random seeds which lead to large differences in output between different computing platforms.

The `rst2html` module provides tools for converting model docs to html and viewing the html. This is used by `compare` to display the model description, such as:

```
$ ./sascomp -html sphere
```

This makes debugging the latex much faster, though this may require Qt in order for mathjax to work correctly.

When run as main, it can display arbitrary ReStructuredText files. E.g.,

```
$ python -m sasmodels.rst2html doc/developer/overview.rst
```

This is handy for sorting out rst and latex syntax. With some work the results could be improved so that it recognizes sphinx roles such as *mod*, *class* and *func*, and so that it uses the style sheets from the sasmodels docs.

The `list_pars` module lists all instances of parameters across all models. This helps to make sure that similar parameters have similar names across the different models. With the verbose flag, the particular models which use each named parameter are listed.

Model conversion

- `convert`
- `conversion_table`

Model definitions are not static. As needs change or problems are found, models may be updated with new model names or may be reparameterized with new parameter definitions. For example, in translating the Teubner-Strey model from SasView 3.x to sasmodels, the definition in terms of *drho*, *k*, *c1*, *c2*, *a2* and prefactor was replaced by the defintion in terms of *volfraction_a*, *xi*, *d*, *sld_a* and *sld_b*. Within `convert`, the *_hand_convert_3_1_2_to_4_1* function must be called when loading a 3.x model definition to update it to 4.1, and then the model should be further updated to 4.2, 5.0, and so on. The `convert.convert_model()` function does this, using the conversion table in `conversion_table` (which handled the major renaming from SasView 3.x to sasmodels), and using the internal *_hand_convert* function for the more complicated cases.

Other

- `exception`
- `alignment`

The `exception.annotate_exception()` function annotates the current exception with a context string, such as “while opening myfile.dat” without adjusting the traceback.

The `alignment` module is unused.

Calculator Interface

This document describes the layer between the form factor kernels and the model calculator which implements the dispersity and magnetic SLD calculations. There are three separate implementations of this layer, `kernelcl` for OpenCL, which operates on a single Q value at a time, `kerneldll` for the DLL, which loops over a vector of Q values, and `kernelpy` for python models which operates on vector Q values.

Each implementation provides three different calls *Iq*, *Iqxy* and *Imagnetic* for 1-D, 2-D and 2-D magnetic kernels respectively. The C code is defined in *kernel_iq.c*, with the minor differences between OpenCL and DLL handled by #ifdef statements.

The kernel call looks as follows:

```
kernel void KERNEL_NAME(
    int nq,                  // Number of q values in the q vector
    int pd_start,            // Starting position in the dispersity loop
    int pd_stop,             // Ending position in the dispersity loop
    ProblemDetails *details, // dispersity info
    double *values,          // Value and weights vector
    double *q,               // q or (qx,qy) vector
    double *result,          // returned I(q), with result[nq] = pd_weight
    double cutoff)           // dispersity weight cutoff
```

The details for OpenCL and the python loop are slightly different, but these data structures are common.

nq indicates the number of q values that will be calculated.

The *pd_start* and *pd_stop* parameters set the range of the dispersity loop to compute for the current kernel call. Give a dispersity calculation with 30 weights for length and 30 weights for radius for example, there are a total of 900 calls to the form factor required to compute the kernel. These might be done in chunks of 100, so the first call would start at zero and stop after 100 iterations. The second call would then have to set the length index to 3 and the radius index to 10 for a position of 3*30+10=100, and could then proceed to position 200. This allows us to interrupt the calculation in the middle of a long dispersity loop without having to do special tricks with the C code. More importantly, it stops the OpenCL kernel in a reasonable time; because the GPU is used by the operating system to show its windows, if a GPU kernel runs too long then it will be automatically killed and no results will be returned to the caller.

The *ProblemDetails* structure is a direct map of the `details.CallDetails` buffer. This indicates which parameters have dispersity, and where in the values vector the values and weights can be found. For each parameter with dispersity there is a parameter id, the length of the dispersity loop for that parameter, the offset of the parameter values in the pd value and pd weight vectors and the 'stride' from one index to the next, which is used to translate between the position in the dispersity loop and the particular parameter indices. The *num_eval* field is the total size of the dispersity loop. *num_weights* is the number of elements in the pd value and pd weight vectors. *num_active* is the number of non-trivial pd loops (parameters with dispersity should be ordered by decreasing pd vector length, with a length of 1 meaning no dispersity). Oriented objects in 2-D need a cos(theta) spherical correction on the angular variation in order to preserve the 'surface area' of the weight distribution. *theta_par* is the id of the polar coordinate parameter if there is one.

The *values* vector consists of the fixed values for the model plus pd value and pd weight vectors. There are *NUM_VALUES* fixed values for the model, which includes the initial two slots for scale and background, the NUM_PARS values for the kernel parameters, three slots for the applied magnetism, and three slots for each of the SLD parameters for the sample magnetiziation *(Mx, My, Mz)*. Sample magnetization is translated from *(M, theta, phi)* to *(Mx, My, Mz)* before the kernel is called. After the fixed values comes the pd value vector, with the dispersity values for each parameter stacked one after the other. The order isn't important since the location for each parameter is stored in the *pd_offset* field of the *ProblemDetails* structure, but the values do need to be contiguous. After *num_weights* values, the pd weight vector is stored, with the same configuration as the pd value vector. Note that the pd vectors can contain values that are not in the dispersity loop; this is used by `mixture.MixtureKernel` to make it easier to call the various component kernels.

The *q* vector contains one value for each q for *Iq* kernels, or a pair of *(qx,qy)* values for each q for *Iqxy* and *Imagnetic* kernels. OpenCL pads all vectors to 32 value boundaries just to be safe, including the *values* vector and the the *results* vector.

The *results* vector contains one slot for each of the *nq* values, plus one extra slot at the end for the weight normalization accumulated across all points in the dispersity mesh. This is required when the dispersity loop is broken across several kernel calls.

cutoff is a importance cutoff so that points which contribute negligibly to the total scattering can be skipped without calculating them.

`generate.make_source()` defines the following C macros:

- USE_OPENCL is defined if running in opencl
- MAX_PD is the maximum depth of the dispersity loop [model specific]

- NUM_PARS is the number of parameter values in the kernel. This may be more than the number of parameters if some of the parameters are vector values.
- NUM_VALUES is the number of fixed values, which defines the offset in the value list to the dispersity value and weight vectors.
- NUM_MAGNETIC is the number of magnetic SLDs
- MAGNETIC_PARS is a comma separated list of the magnetic SLDs, indicating their locations in the values vector.
- KERNEL_NAME is the name of the function being declared
- PARAMETER_TABLE is the declaration of the parameters to the kernel:

  Cylinder:

  ```
  #define PARAMETER_TABLE \
  double length; \
  double radius; \
  double sld; \
  double sld_solvent;
  ```

  Note: scale and background are never included

  Multi-shell cylinder (10 shell max):

  ```
  #define PARAMETER_TABLE \
  double num_shells; \
  double length; \
  double radius[10]; \
  double sld[10]; \
  double sld_solvent;
  ```

- CALL_IQ(q, i, var) is the declaration of a call to the kernel:

  Cylinder:

  ```
  #define CALL_IQ(q, i, var) Iq(q[i], \
  var.length, \
  var.radius, \
  var.sld, \
  var.sld_solvent)
  ```

  Multi-shell cylinder:

  ```
  #define CALL_IQ(q, i, var) Iq(q[i], \
  var.num_shells, \
  var.length, \
  var.radius, \
  var.sld, \
  var.sld_solvent)
  ```

  Cylinder2D:

  ```
  #define CALL_IQ(q, i, var) Iqxy(qa, qc, \
  var.length, \
  var.radius, \
  var.sld, \
  var.sld_solvent)
  ```

- CALL_VOLUME(var) is similar, but for calling the form volume:

```
#define CALL_VOLUME(var) \
form_volume(var.length, var.radius)
```

There is an additional macro that can be defined within the model.c file:

- INVALID(var) is a test for model parameters in the correct range:

  Cylinder:

  ```
  #define INVALID(var) 0
  ```

  BarBell:

  ```
  #define INVALID(var) (var.bell_radius < var.radius)
  ```

  Model with complicated constraints:

  ```
  inline bool constrained(p1, p2, p3) { return expression; }
  #define INVALID(var) constrained(var.p1, var.p2, var.p3)
  ```

Our design supports a limited number of dispersity loops, wherein we need to cycle through the values of the dispersity, calculate the I(q, p) for each combination of parameters, and perform a normalized weighted sum across all the weights. Parameters may be passed to the underlying calculation engine as scalars or vectors, but the dispersity calculator treats the parameter set as one long vector.

Let's assume we have 8 parameters in the model, two of which allow dispersity. Since this is a 1-D model the orientation parameters won't be used:

```
0: scale        {scl = constant}
1: background   {bkg = constant}
2: radius       {r = vector of 10pts}
3: length       {l = vector of 30pts}
4: sld          {s1 = constant/(radius**2*length)}
5: sld_solvent  {s2 = constant}
6: theta        {not used}
7: phi          {not used}
```

This generates the following call to the kernel. Note that parameters 4 and 5 are treated as having dispersity even though they don't — this is because it is harmless to do so and simplifies the looping code:

```
MAX_PD = 4
NUM_PARS = 6          // kernel parameters only
NUM_VALUES = 17       // all values, including scale and background
NUM_MAGNETIC = 2      // two parameters might be magnetic
MAGNETIC_PARS = 4, 5  // they are sld and sld_solvent

details {
    pd_par = {3, 2, 4, 5}         // parameters *radius* and *length* vary
    pd_length = {30, 10, 1, 1}    // *length* has more, so it is first
    pd_offset = {10, 0, 31, 32}   // *length* starts at index 10 in weights
    pd_stride = {1, 30, 300, 300} // cumulative product of pd length
    num_eval = 300   // 300 values in the dispersity loop
    num_weights = 42 // 42 values in the pd vector
    num_active = 2   // only the first two pd are active
    theta_var =  6   // spherical correction
}

values = { scl, bkg,                                  // universal
           r, l, s1, s2, theta, phi,                  // kernel pars
           in spin, out spin, spin angle,             // applied magnetism
           mx s1, my s1, mz s1, mx s2, my s2, mz s2,  // magnetic slds
           r0, .., r9, l0, .., l29, s, s2,            // pd values
           r0, .., r9, l0, .., l29, s, s2}            // pd weights

nq = 130
q = { q0, q1, ..., q130, x, x }  # pad to 8 element boundary
result = {r1, ..., r130, pd_norm, x }
```

The dispersity parameters are stored in as an array of parameter indices, one for each parameter, stored in pd_par[n]. Parameters which do not support dispersity do not appear in this array. Each dispersity parameter has a weight vector whose length is stored in pd_length[n]. The weights are stored in a contiguous vector of weights for all parameters, with the starting position for the each parameter stored in pd_offset[n]. The values corresponding to the weights are stored together in a separate weights[] vector, with offset stored in par_offset[pd_par[n]]. Dispersity parameters should be stored in decreasing order of length for highest efficiency.

We limit the number of dispersity dimensions to MAX_PD (currently 4), though some models may have fewer if they have fewer dispersity parameters. The main reason for the limit is to reduce code size. Each additional dispersity parameter requires a separate dispersity loop. If more than 4 levels of dispersity are needed, then we need to switch to a monte carlo importance sampling algorithm with better performance for high-dimensional integrals.

Constraints between parameters are not supported. Instead users will have to define a new model with the constraints built in by making a copy of the existing model. Mac provides OpenCL and we are supplying the tinycc compiler for Windows so this won't be a complete limitation, but it is rather inconvenient. The process could perhaps be automated so that there is no code copying required, just an alternate CALL_IQ macro that implements the constraints. Think carefully about constraining theta since the polar coordinates normalization is tied to this parameter.

If there is no dispersity we pretend that we have a disperisty mesh over a single parameter with a single point in the distribution, giving pd_start=0 and pd_stop=1.

The problem details structure could be allocated and sent in as an integer array using the read-only flag. This would allow us to copy it once per fit along with the weights vector, since features such as the number of disperity points per pd parameter won't change between function evaluations. A new parameter vector must be sent for each I(q) evaluation. This is not currently implemented, and would require some resturcturing of the `sasview_model.SasviewModel` interface.

The results array will be initialized to zero for dispersity loop entry zero, and preserved between calls to [start, stop] so that the results accumulate by the time the loop has completed. Background and scale will be applied when the loop reaches the end. This does require that the results array be allocated read-write, which is less efficient for the GPU, but it makes the calling sequence much more manageable.

For accuracy we may want to introduce Kahan summation into the integration:

```
double accumulated_error = 0.0;
...
#if USE_KAHAN_SUMMATION
    const double y = next - accumulated_error;
    const double t = ret + y;
    accumulated_error = (t - ret) - y;
    ret = t;
#else
    ret += next;
#endif
```

This will require accumulated error for each I(q) value to be preserved between kernel calls to implement this fully. The *kernel_iq.c* code, which loops over q for each parameter set in the dispersity loop, will also need the accumulation vector.

2.2 Indices and Search

- genindex
- modindex
- search

CHAPTER

THREE

RELEASE NOTES

Note: In Windows use [Alt]-[Cursor left] to return to the previous page

CHAPTER

FOUR

FEATURES

4.1 New in Version 4.2.1

The major changes for this point release were to fix several problems with using the built in editor to create new models and to bring the NXcanSAS reader into compliance with the final published specification. The orignal reader was based on a draft version of the specification. As an early adopter, interpretation and implementation of the spec was iterated with all producers of NXcanSAS reduced data known to the SasView team in order to ensure compatibility and verify the implementation. A few other enhancements and bug fixed were also introduced such as cleaning up the resolution section of the fitting page GUI, increasing the max size range allowed in the corfunc analysis, and adding the incomplete gamma function to the python library.

4.1.1 Resolved Issues

- Fixes # 976: CanSas HDF reader will not read all valid CanSas HDF (NXcanSAS) files
- Fixes # 1074: Add incomplete gamma function to sasmodels
- Fixes # 1111: Convert all input Q units to 1/A
- Fixes # 1129: NXcanSAS writer not writing all meta data
- Fixes # 1142: Plugin framework is broken
- Fixes # 1183: Test from creating new model reset all parameters to default in all open FitPages
- Fixes # 1188: Colons removed from magnetic parameter names to address Python variable issue - done in 4.2. but documented in 4.2.1
- Fixes # 1205: 4.2 set weighting choice seems to be ignored.
- Fixes # 1206: Incorrect (and confusing) presentation of dQ from data in instrumental smearing section
- Fixes # 1212: Bug in Iqxqy plotting non rectangular / square matrices?
- Fixes # 1221: ABS reader does not read in USANS data properly GitHub
- Fixes # 1222: smearing options incorrect on show2D and show1D in fitpage14: Loading a saved project is really really slow
- Fixes # 1223: Expand permitted range of transformed data in Corfunc implementation

4.2 New in Version 4.2.0

This release heralds many improvements and a host of bug fixes, along with some significant changes from previous versions. Further, as promised, it marks the end of support for 32 bit operating systems and is only available for 64 bit operating systems.

With this version the change to the new model API and plugins infrastructure begun with 4.0 is essentially complete (though extensions are in the works, and more are likely, they should remain backwardly compatible with previous versions of SasView).

Warning: Old-style plugin models, including old sum|multiply models, continue to be supported (i.e. SasView will run them) in 4.x, although our automatic on-the-fly translation may not cope in all use cases (see Known Issues below). However, this backward compatibility will be removed in 5.0 and users are therefore strongly encouraged to convert their custom models to the new API.

Finally, the changes to orientation angles and orientational distribution definitions are now also complete.

4.2.1 Changes

- The infrastucture for calculating 2D patterns from 3D orientated objects has been totally re-factored. It is now more accurate and consistent across models.
- The way that SasView defines the orientation of anisometric and aligned objects has been completely overhauled. It now differs from previous versions.
- Plugin models, including sum|multiply models, have completely migrated to the new infrastructure. NOTE that 3.x type models as well as early, intermediate 4.x type models, including those generated by sum|multiply will continue to be supported in 4.x but will likely no longer be supported after the move to 5.0. Users are strongly encouraged to migrate any custom models.
- The NeXus loader has been removed as it is superseded by the NXcanSAS standard loader and SasView does not support the treatment of raw data.

4.2.2 Improvements

- The accuracy/speed of some numerical integrations have been improved.
- An orientation viewer tool has been introduced to assist in understanding the new orientation framework.
- Problems with the computation of magnetic scattering from some objects have been rectified. Some questions remain however.
- The known issue with the core_shell_parallelepiped model is now fixed.
- An error in the be_polyelectrolyte model was identified and rectified, but the fix is yet to be validated.
- (Added post-release) An error with the reporting of the scale parameter from the spinodal model was rectified.
- A number of issues and inconsistencies with the creation of sum|multiply models have been rectified.
- A Boltzmann distribution has been added for polydispersity/orientational distributions.
- Some batch slicing options have been introduced.
- Correlation function analysis now computes both the 1D and 3D functions.
- There are several data loading improvements.
- There are several improvements to Save/Load Project.
- The SasView version number now appears in Reports.
- The Release Notes are now available from the program Help menu.
- There have been numerous other bug fixes.

4.2.3 Documentation

Several sections of the help documentation have undergone significant checking and updating, particularly those relating to orientation, magnetic scattering, and polydispersity distributions.

Detailed advanced instructions for plugin writing and some scripting instructions have also been added.

Concerns about the intended versus implemented meaning of some parameters in the bcc_paracrystal, fcc_paracrystal, and sc_paracrystal models have been brought to our attention. These have yet to be resolved and so a Warning has been placed on each of these models. Anyone who feels they may have the requisite expertise to investigate these concerns is strongly encouraged to contact the Developers!

4.2.4 Other Work

- A Third-Party initiative has recently succeeded in getting SasView to run on Debian. More details at http://trac.sasview.org/wiki/DevNotes/Projects/Debian
- With this release we have started to prepare for the inevitable move to Python 3, which will occur with the release of 5.0
- SasView 5.0 is currently in development. The two most significant features of this version will be (i) a move away from the present WxPython GUIs to new, completely rewritten, Qt5 GUIs, and (ii)implementation of the Beta-approximation for S(Q). Subject to resources, some limited access to the latter functionality may be available in a future SasView 4.x release.

4.2.5 Bug Fixes

- Fixes # 14: Loading a saved project is really really slow
- Fixes # 260: Box integration does not update when entering values in dialog
- Fixes # 446: Saving plot as PGF (not PDF!) format throws error
- Fixes # 467: Extend batch functionality to slicer
- Fixes # 489: ABS reader (NIST 1D) does not handle negative dx properly (USANS slit smearing)
- Fixes # 499: create sin(x)/x, 2*J1(x)/x and 3*j1(x)/x functions
- Fixes # 510: Build PDF documentation along with HTML
- Fixes # 525: Add GUI category defaults to models in sasmodels
- Fixes # 579: clean up sasview directory
- Fixes # 597: Need to document Combine Batch Fit
- Fixes # 645: GUI logic problem in Batch vs single fit mode
- Fixes # 648: Need to allow user input background value in Pr perspective
- Fixes # 685: Fix data upload to marketplace
- Fixes # 695: linear slope in onion model
- Fixes # 735: Review new Corfunc documentation
- Fixes # 741: Recalculate P(Q) and S(Q) components on model update.
- Fixes # 767: Sum/Product Models don’t do what they should
- Fixes # 776: angular dispersity
- Fixes # 784: Add 3D integral to Correlation Function analysis
- Fixes # 786: core_shell_parallelepiped 1-D model is incorrect
- Fixes # 818: “report button” followed by “save” makes an empty pdf file???

- Fixes # 830: Check compliance of loader against NXcanSAS-1.0 release
- Fixes # 838: Fix model download from marketplace
- Fixes # 848: can't save analysis when only one fit page
- Fixes # 849: Load Folder should ignore files starting with .
- Fixes # 852: More unit tests, especially for oriented or 2d models
- Fixes # 854: remove unnecessary sleep() in fitting perspective
- Fixes # 856: Reading SAS_OPENCL from custom_config sometimes raises an ERROR
- Fixes # 861: cannot defined a structure factor plugin
- Fixes # 864: New Model Editor (simple plugin editor) error parsing parameter line
- Fixes # 865: Plugin live discovery issues
- Fixes # 866: inform user when NaN is returned from compute
- Fixes # 869: fit page computation thread cleanup
- Fixes # 875: Possible weirdness with 1D NXcanSAS data
- Fixes # 876: Add check for HDF5 format in dataloader
- Fixes # 887: reorganize tree, separating the installed source from the build source
- Fixes # 889: Refactor dataloader error handling infrastructure
- Fixes # 890: use new orientation definition for asymmetric shapes
- Fixes # 891: update docs for oriented shapes with new orientation definition
- Fixes # 896: equations in core shell parallelepiped docs do not match code
- Fixes # 898: Image Viewer Tool file selector issue
- Fixes # 899: Igor Reader q calculation
- Fixes # 902: IgorReader Q calculation needs fixing/improving
- Fixes # 903: sasview - all non-gui tests should be converted to run in Python 3
- Fixes # 906: polydispersity not showing up in tabulated results
- Fixes # 912: About box points to misleading contributors page on Github
- Fixes # 913: Need to add Diamond developer and logo in relevant places
- Fixes # 915: load project issues
- Fixes # 916: Proper Logging
- Fixes # 920: Logarithmic binning option in the slice viewer
- Fixes # 921: Improve developer communication methods
- Fixes # 922: Remove support for all data formats that are not in q space
- Fixes # 923: Add CI and trac integrations to Slack
- Fixes # 930: fitting help says chisq is normalized to number of points
- Fixes # 931: Allow admins to edit all models and upload data etc on marketplace
- Fixes # 932: Need to fix upload of data files to marketplace
- Fixes # 934: Slurp tutorial repo for tutorials
- Fixes # 935: Build new tutorials as PDF
- Fixes # 943: Deep copy error on setting model after data is selected
- Fixes # 950: Most of the readers don't close files properly.

- Fixes # 954: cross check dll/opencl/python polydispersity and orientation results
- Fixes # 956: Possible problem with new doc build process
- Fixes # 961: sasmodels tests should fail if the parameter name does not exist
- Fixes # 962: star polymer typo in docs
- Fixes # 966: Inconsistent chi2 reporting
- Fixes # 967: no uncertainties errors on fitting parameters
- Fixes # 969: About Box not picking up dls_logo.png
- Fixes # 970: ASCII loader doesn't handle ISIS 2D ASCII
- Fixes # 974: blacklist Intel HD 620/630 for double precision
- Fixes # 978: load project fails for pages which have not been defined
- Fixes # 983: Remove Nexus Loader
- Fixes # 984: PDF reports are not being properly generated on Windows
- Fixes # 985: Saving Project Fails
- Fixes # 986: Send to fitting overwrites theory page even if blank FitPage has focus
- Fixes # 990: utest_sasview.py giving different results than run_one.py
- Fixes # 993: Windows x64 versions not installing to correct folder
- Fixes # 994: Error changing fit engine
- Fixes # 995: OpenCL required on Linux even if turned off in GUI
- Fixes #1006: multiplicity models don't work with SQ
- Fixes #1007: spherical_sld model freezes SasView
- Fixes #1008: plugin model scaling not working?
- Fixes #1010: Win64 build script not creating working executable
- Fixes #1011: sld_test failing on ubuntu
- Fixes #1013: FileReaderBaseClass output[] not reset - same file loaded multiple times
- Fixes #1018: add Boltzmann distribution
- Fixes #1021: add PDF documentation to website and document in wiki release process
- Fixes #1024: Update version numbers in master
- Fixes #1025: Sum/multiply editor hangs
- Fixes #1030: volume normalization for hollow shapes is different from solvent-filled shapes
- Fixes #1032: convert C++ modules to C
- Fixes #1035: Order of combining P(Q) and S(Q) in Plugins seems to matter
- Fixes #1037: data loader crop not working? & all fits crashing
- Fixes #1043: problem compiling marketplace models
- Fixes #1044: Unable to upload c file to marketplace
- Fixes #1046: convert non builtin models in the marketplace to new API
- Fixes #1050: fix appveyor test for sasmodels win 64 python 3
- Fixes #1052: Can't use a user-created plugin model in a plugin model
- Fixes #1054: Check plugin & orientation descriptions in full docs once SasModels PR #57 is merged
- Fixes #1057: phi rotation issue for elliptical cylinder

- Fixes #1060: incorrect default for rectangle dispersion
- Fixes #1062: win32 build not installing correctly
- Fixes #1064: "Fitting did not converge!!!" error with a Sum|Multi plugin model
- Fixes #1068: 2d data (from NG7) not loadiing - strange format?
- Fixes #1069: GUI problem when using polydispersity/orientation distributions
- Fixes #1070: Parameter error boxes should not be editable
- Fixes #1072: Orientation distributions seem to depend on initial angle
- Fixes #1079: Remove save button in report dialog on Mac
- Fixes #1081: GUI problem with new orientation distribution
- Fixes #1083: Magnetic models not being computed
- Fixes #1099: Erratic behaviour of Sum|Multi model in 4.1.2
- Fixes #1101: Batch results page not displaying polydispersity values
- Fixes #1128: AutoPlot generation for model documentation does not include background
- Fixes #1131: OpencCl dialog does not open
- Fixes #1132: Slit Size Calculator Tool not working
- Fixes #1139: Missing Docs and Help for new Batch Slicing
- Fixes #1141: Intro to scripting.rst needs improvement
- Fixes #1142: Plugin framework is broken
- Fixes #1145: Update models in model marketplace to 4.2 when 4.2 is released.
- Fixes #1155: BE Polyelectrolyte errors
- Fixes #1160: fix VR for core_shell_cylinder, fractal_core_shell and hollow_cylinder
- Fixes #1163: Fix help note in sum of sum|multiply interface
- Fixes #1164: Sphinx doc build does not support superscript or substitution
- Fixes #1166: No longer able to report from multiple fit pages
- Fixes #1167: Clarify the documentation for the Spinodal Model
- Fixes #1173: more problems with math in plugins
- Fixes #1176: Make Release Notes/Known Issues available from Help in Menu Bar
- Fixes #1179: PDF Report should contain SasView Version Number
- Fixes #1183: Test from creating new model reset all parameters to default in all open FitPages
- Fixes #1188: fitpage hangs if change model while magnetism is on
- Fixes #1191: Correct erroneous Scale reported by Spinodal model

It is recommended that all users upgrade to this version, but your attention is drawn to the Changes section above.

4.3 New in Version 4.2.0-Beta

This is a beta pre-release version of 4.2.0. A number of fixes and changes have been made in the year since the previous release. Full release notes will be compiled prior to the full release 4.2.0.

Highlights are:

- Infrastucture for calculating 2D patterns from 3D orientated objects has now been totally refactored

- Plugins have completely migrated to the new infrastructure now, including sum/multiply models
- Some batch slicing options have been introduced
- The known issue with the core_shell_parallelepiped is now fixed
- Several data loading improvements
- Several save Project improvements (though there are more to come)
- Numerous bug fixes
- Lots of documentation enhancement

In the meantime please report any bugs or issues found while using this beta

4.4 New in Version 4.1.2

This point release is a bug-fix release addressing:

- Fixes #984: PDF Reports Generate Empty PDFs
- Fixes a path typo
- 64 bit and 32 bit Windows executables now available

It is recommended that all users upgrade to this version

4.5 New in Version 4.1.1

This point release is a bug-fix release addressing:

- Fixes #948: Mathjax CDN is going away
- Fixes #938: Cannot read canSAS1D file output by SasView
- Fixes #960: Save project throws error if empty fit page
- Fixes #929: Problem deleting data in first fit page
- Fixes #918: Test folders not bundled with release
- Fixes an issue with the live discovery of plugin models
- Fixes an issue with the NXcanSAS data loader
- Updated tutorials for SasView 4.x.y

4.6 New in Version 4.1.0

This incremental release brings a series of new features and improvements, and a host of bug fixes. Of particular note are:

4.6.1 Correlation Function Analysis (Corfunc)

This performs a correlation function analysis of one-dimensional SAXS/SANS data, or generates a model-independent volume fraction profile from the SANS from an adsorbed polymer/surfactant layer.

A correlation function may be interpreted in terms of an imaginary rod moving through the structure of the material. G1(R) is the probability that a rod of length R moving through the material has equal electron/neutron scattering length density at either end. Hence a frequently occurring spacing within a structure manifests itself as a peak.

A volume fraction profile Phi(z) describes how the density of polymer segments/surfactant molecules varies with distance from an (assumed locally flat) interface. *This is not yet implemented.*

4.6.2 Fitting of SESANS Data

Data from Spin-Echo SANS measurements can now be loaded and fitted. The data will be plotted against the correct axes and models will automatically perform a Hankel transform in order to calculate SESANS from a SANS model.

4.6.3 Documentation

The documentation has undergone significant checking and updating.

4.6.4 Improvements

- Correlation function (corfunc) analysis of 1D SAS data added from CCP13
- File converter tool for multi-file single column data sets
- SESANS data loading and direct fitting using the Hankel transformation
- Saving and loading of simultaneous and constrained fits now supported
- Save states from SasView v3.x.y now loaded using sasmodel model names
- Saving and loading of projects with 2D fits now supported
- Loading a project removes all existing data, fits, and plots
- Structure factor and form factor can be plotted independently
- OpenCL is disabled by default and can be enabled through a fit menu
- Data and theory fields are now independently expandable

4.6.5 Bug Fixes

- Fixes #667: Models computed multiple times on parameters changes
- Fixes #673: Custom models override built in models of same name
- Fixes #678: Hard crash when running complex models on GPU
- Fixes $774: Old style plugin models unloadable
- Fixes #789: stacked disk scale doesn't match cylinder model
- Fixes #792: core_shell_fractal uses wrong effective radius
- Fixes #800: Plot range reset on plot redraws
- Fixes #811 and #825: 2D smearing broken
- Fixes #815: Integer model parameter handling
- Fixes #824: Cannot apply sector averaging when no detector data present
- Fixes #830: Cansas HDF5 reader fully compliant with NXCanSAS v1.0 format
- Fixes #835: Fractal model breaks with negative Q values
- Fixes #843: Multilayer vesicle does not define effective radius
- Fixes #858: Hayter MSA S(Q) returns errors
- Numerous grammatical and contexual errors in documention

4.7 New in Version 4.0.1

This release fixes the critical bug #750 in P(Q)xS(Q). Most damaging it appears that the background term was being added to S(Q) prior to multiplication by P(Q).

4.8 New in Version 4.0

This release fixes the various bugs found during the alpha and beta testing

4.8.1 Improvements

- Support for reading data files from Anton Paar Saxess instruments
- Adds documentation on how to write custom models in the new framework

4.8.2 Bug Fixes

- Fixes #604: Pringle model questions
- Fixes #472: Reparameterize Teubner-Strey
- Fixes #530: Numerical instabilities in Teubner Strey model
- Fixes #658: ASCII reader very broken

4.9 New in Version 4.0 beta 1

This beta adds support for the magnetic and multilevel models of 3.1.2 and along with a host of bug fixes found in the alpha.

4.9.1 Model package changes and improvements

- All 3.1.2 models now available in new interface
- Old custom models should now still work
- Custom model editor now creates new style models
- Custom model editor supports better error checking

Note: Old custom models should be converted to the new model format which is now the same as the built-in models and offers much better support. The old custom model format will be deprecated in a future version.

4.9.2 Documentation improvements

- Continued general cleanup

4.9.3 Other improvements/additions

- Support for new canSAS 2D data files added
- Plot axes range can now be set manually as well as by zooming
- Plot annotations can now be moved around after being placed on plot.
- The active optimizer is now listed on the top of the fit panel.
- Linear fits now update qmin and max when the x scale limits are changed. Also the plot range no longer resets after a fit.

4.9.4 Bug fixes

- Fixes #511: Errors in linearized fits and clean up of interface including Kratky representation
- Fixes #186: Data operation Tool now executes when something is entered in the text box and does not wait for the user to hit enter
- Fixes #459: plot context menu bug
- Fixes #559: copy to clipboard in graph menu broken
- Fixes #466: cannot remove a linear fit from graph
- Numerous bugs introduced in the alpha

4.10 New in Version 4.0.0-alpha

This alpha release brings a major overhaul of the model system. The new model package allows rapid integration of custom models and access to polydispersity without requiring a compiler.

4.10.1 Model package changes and improvements

- Separation of GUI and calculations for future GUI enhancements
- Model interface moved to independent sasmodels package.
- Most models converted to new interface.
- Allows rapid integration of user-written models.
- OpenCL GPU utilization for faster fitting.
- Improved numerical integration of Bessel functions.

4.10.2 SESANS integration and implementation

- Scripting interface added for analysis of SESANS data.
- Hankel transformation now accepts finite acceptance angles.
- 2D cosine transformation added for TOF SESANS analysis.

4.10.3 Documentation improvements

- The documentation tree was restructured for a better end user experience.
- The documentation for each model was revamped and verified by at least two people following the conversion of the model.
- Theoretical 1D (and 2D if applicable) scattering curves are auto-generated and added to the model documentation for each model.

4.10.4 Bug fixes

- Fixes #411: No stop button on simultaneous fit page
- Fixes #410: Error with raspberry model
- Fixes #364: Possible inconsistency in Poly_GausCoil model
- Fixes #439: Hayter Penfold MSA code needs checking
- Fixes #484: lammellerPC is precision limited
- Fixes #498: $HOME/.matplotlib conflicts
- Fixes #348: Control order in which fit parameters appear in the gui
- Fixes #456: Provide DREAM Results Panel with something to identify data and age of results
- Fixes #556: Build script improvements for developers

4.11 New in Version 3.1.2

This release is a major stability improvement, having fixed a serious bug that came to light since release 3.1.1. All users should upgrade.

- Fixes #468: broken remove constraint buttons in simultaneous/constrained fitting panel
- Fixes #474: resulting from changes in 3.1.1 that had introduced an error in the high-Q of slit-smeared models.
- Fixes #478: which would cause wx to run out of IDs and result in SasView crashing even if left alone.
- Fixes #479: missing help button on simultaneous/constrained fit page
- Fixes #480: GUI resizing issues on simultaneous fit page
- Fixes #486: broken Report Results
- Fixes #488: redraw issues in fit page

4.12 New in Version 3.1.1

Fixes #457 that prevented SasView from starting if the user was not connected to the internet, or was behind a proxy server.

4.13 New in Version 3.1.0

The documentation/help has had a complete overhaul including:

- A completely new presentation interface (Sphinx).
- Proof reading!

- Updating for latest features.
- A Help (or sometimes ?) button has been added to every panel, and some sub panels if appropriate, linking to the appropriate section in the documentation.
- The model help has been split so that the Details button now brings up a very short pop-up giving the equation being used while HELP goes to the section in the full documentation describing the model.
- Extensive help has also been added for the new optimizer engine (see below) including rules of thumb on how and when to choose a given optimizer and what the parameters do.

The optimizer engine has been completely replaced. The new optimizer still defaults to the standard Levenberg-Marquardt algorithm. However 4 other optimizers are now also available. Each starts with a set of default parameters which can be tuned. The DREAM optimizer takes the longest but is the most powerful and yields rich information including full parameter correlation and uncertainty plots. A results panel has been added to accommodate this. The five new optimizers are:

- A Levenberg-Marquardt optimizer
- A Quasi-Newton BFGS optimizer
- A Nelder-Mead Simplex optimizer
- A Differential Evolution optimizer
- A Monte Carlo optimizer (DREAM)

New models were added:

- MicelleSphCoreModel (currently residing in the Uncategorized category)

Existing models were updated:

- LamellarPS (bug in polydispersity integration fixed)
- RectangularPrismModel
- RectangularHollowPrismModel
- RectangularHollowPrismInfThinWallsModel

Other work:

- Infrastructure to allow SESANS data to be fit with models was added. This will become available in a future release but can currently be used from the command line with some caveats.
- A number of bugs were fixed including a thread crashing issue and an incorrect slit smearing resolution calculation.
- Implemented much more robust error logging to enable much easier debugging in general but particularly the debugging of issues reported by SasView users.
- A number of infrastructure tasks under the hood to enhance maintainability
- Upgrade from Wx 2.8 to Wx 3.0.2 which allows several new features but required significant additional rework as well.
- Fully implemented Sphinx to the build process to produce both better user documentation and developer documentation.
- Restructuring of the code base to more unified nomenclature and structure so that the source installation tree more closely matches the installer version tree.
- Code cleanup (an ongoing task).
- Migration of the repository to github simplifying contributions from non-project personnel through pull requests.

4.14 New in Version 3.0.0

- The GUI look and feel has been refactored to be more familiar for Windows users by using MDI frames. Graph windows are also now free- floating.
- Five new models have been added: PringlesModel, CoreShellEllipsoidXTModel, RectangularPrismModel, RectangularHollowPrismModel and RectangularHollowPrismInfThinWallsModel.
- The data loader now supports ILL DAT data files and reads the full meta information from canSAS file formats.
- Redefined convention for specifying angular parameters for anisotropic models.
- A number of minor features have been added such as permitting a log distribution of points when using a model to simulate data, and the addition of a Kratky plot option to the linear plots.
- A number of bugs have also been fixed.
- Save Project and Save Analysis now work more reliably.
- BETA: Magnetic contrast supporting full polarization analysis has been implemented for some spherical and cylindrical models.
- BETA: Two new tools have been added:
  - A generic scattering calculator which takes an atomic, magnetic or SLD distribution in space and generates the appropriate 2D scattering pattern. In some cases the orientationally averaged (powder) 1D scattering can also be computed. Supported formats include: SLD or text, PDB, and OMF magnetic moment distribution file.
  - An image viewer/converter for data in image format; this reads in an image file and will attempt to convert the image pixels to data. Supported formats include: TIFF, TIF, PNG, BMP, JPG.

4.15 New in Version 2.2.1

- Minor patch to support CanSAS XML v1.1 file format
- Added DataInfo for data in the DataExplorer and plots
- Added Maximize/Restore button in the title bar of the graphs
- Added a hide button in the toolbar of the graph panel
- The ‘x’ button now deletes a graph
- Edit SUM Model from the menubar can now generate and save more than one sum model
- Reports can now be saved in pdf format on WIN and MAC
- Made significant improvements to the batch/grid panel and fixed several bugs
- Fixed a number of other minor bugs

4.16 New in Version 2.2.0

- Application name changed to SasView
- New fully customizable Category Manager added for better management of increasing number of models
- Improved the Grid Window functionality in the batch fitting mode
- Added a simpler Graph/Plot modification interface
- Added a new ‘Data Operation’ tool for addition, subtraction, multiplication, division, of two data sets.
- The ‘Sum Model’ editor was extended and renamed ‘Summation and Multiplication’ editor

- Added more plot symbols options for 1d plots
- Added improved trapping of compiling errors to the ‘New model editor’
- Added some intelligent outputs (e.g., Rg, background, or rod diameter depending on the choice of axis scale of the plot) to the linear fits
- Added more models

4.17 Feature set from previous versions

4.17.1 Perspectives Available

- Invariant calculator: Calculates the invariant, volume fraction, and specific surface area.
- P(r) inversion calculator: Indirect Fourier transformation method.
- Fitting: the tool used for modeling and fitting 1D and 2D data to analytical model functions
- Tools: provides a number of useful supplementary tools such as SLD calculation

4.17.2 Fitting

- Includes a large number of model functions, both form factors and structure factors.
- Support P(Q)*S(Q) for form factors that flag they can be so multiplied.
- Supports Gaussian, lognormal, Shulz, rectangular and custom distribution functions for models that need to include polydispersity or for orientational distributions if appropriate.
- Anisotropic shapes and magnetic moment modeling in 2D allow for a non-uniform distribution of orientations of a given axis leading to modeling and fitting capabilities of non azimuthaly symmetric data.
- User can choose to weight fits or not. If using weights, the user can choose the error bar on each point if provided in the file, the square root of the intensity or the intensity itself.
- Instrumental resolution smearing of model or fits is provided with several options: read the resolution/point fromt he file. Input a pinhole resolution or a slit resolution.
- Users can define the Qrange (Qmin and Qmax) for both 1D and 2D data for fitting and modeling, but not graphically. The range can be reset to the defaults (limits of q in data set for a fit) with the reset button.
- A mask can be applied to 2D calculation and fitting.
- Normalized residual plots are provided with every fit.
- Model function help available through detail button or from the fitting panel.
- Simultaneous/(advanced)constrained fitting allows for fitting a single data set or several different sets simultaneously with the application of advanced constraints relating fit parameters to functions of other parameters (including from a different set). For example thickness of shell = sin(30) times the length.
- Models that are the sum of two other models can be easily generated through the SUM Model menubar item.
- New Python models can be added on the fly by creating an appropriate Python file in the model plugin directory. Two tools are provided to help: An easy to use custom model editor allows the quick generation of new Python models by supplying only the parameters and their default value (box 1) and the mathematical function of the model (box 2) and generating the necessary .py file. A separate advanced model editor provides a full Python file editor. Either way once saved the model becomes immediately available to the application.
- A batch fitting capability allows for the analysis of a series of data sets to a single model and provides the results in a tabular form suitable for saving or plotting the evolution of the fit parameters with error bars (from within the application).

4.17.3 Tools

- A scattering length density calculator,including some X-ray information is provided.
- A density to vol. fraction converter is provided
- In application access to a Python shell/editor (PyCrust) is provided
- An instrument resolution calculator, including possible gravitational and TOF effects is provided
- A slit size calculator optimized for Anton Paar Saxess is provided.
- A kiessig fringe thickness calculator is provided

4.17.4 Plots and plot management

- A 3D graphing option (for 2d data/results) is provided with the view controlled by the mouse
- 2D plots are shown with an intensity color bar. 2D Color map can be user adjusted.
- Supports output of plot to a variety of graphic formats. Supported formats include: png, eps, emf, jpg/jpeg, pdf, ps, tif/tiff, rawRGBbitmap(raw, rgba), and scalable vector graphic (svg/svgz)
- Supports ouput of data in plot (1 or 2D) to limited data formats
- Multiple data sets can be loaded into a single graph for viewing (but a fit plot can currently only have a single plot).
- Extensive context sensitive plot/fitting/manipulation options are available through a right mouse click pop-up menu on plots.

4.17.5 Data management

- Supports 2 + column 1D ASCII data, NIST 1D and 2D data, and canSAS data via plug-in mechanism which can easily allow other readers as appropriate.
- 2D data is expected in Q space but for historical reasons accepts the NIST 2D raw pixel format and will do conversion internally.
- The full data and metadata available to SasView is viewable in ASCII via right clicking on a data set and choosing Data Info in the DataExplorer or on the plots
- Supports loading a single file, multiple files, or a whole folder
- An optional Data Explorer is provided (default) which simplifies managing, plotting, deleting, or setup for computation. Most functions however do not require access to the explorer/manager and can be accessed through right click menus and the toolbar. The data explorer can be re-started from the menu bar.

4.17.6 Data manipulation

- Support various 2D averaging methods : Circular, sectors, annular, boxsum, boxQx and boxQy.
- A 2D data maks editor is provided
- 2D mask can be applied to the circular averaging.

4.17.7 Miscellaneous features

- Limited reports can be generated in pdf format
- Provides multiprocessor support(Windows only)
- Limited startup customization currently includes default startup data folder and choice of default starting with data manager

- Limited support for saving(opening) a SasView project or a SasView analysis (subproject) is provided.
- SasView can be launched and loaded with a file of interesty by double-clicking on that file (recognized extension)
- A data file or data folder can be passed to SasView when launched from the command line.
- Limited bookmarking capability to later recall the results of a fit calculation is provided.
- Extensive help is provided through context sensitive mouse roll-over, information bar (at the bottom of the panel), the console menu, and access to the help files in several different ways.

CHAPTER

FIVE

DOWNLOADING AND INSTALLING

Note: If you have a SasView installer (.EXE or .MSI), you do not need to worry about any of the following. However, it is highly recommended that any previous versions of SasView are uninstalled prior to installing the new version UNLESS you are installing SasView to versioned folders.

Note: The easiest approach to setting up the proper environment to build from source is to use Conda. Instructions for setting up and using Conda can be found at http://trac.sasview.org/wiki/DevNotes/CondaDevSetup

Note: Much more information is available at www.sasview.org under links/downloads. In particular, look in the 'For Developers' section. Also have a look at http://trac.sasview.org/

5.1 System Requirements

- Python version >= 2.5 and < 3.0 should be running on the system
- We currently use Python 2.7

5.2 Package Dependencies

- Ensure the required dependencies are installed
- For the latest list of dependencies see the appropriate yml file in the SasView repo at sasview/build_tools/conda/ymls

5.3 Installing from Source

- Get the source code
- Create a folder to contain the source code; if working with multiple versions you might want to use versioned folder names like 'sasview-x.x.x'
- Open a command line window in the source code folder
- To get the CURRENT DEVELOPMENT VERSION from source control use git clone https://github.com/SasView/sasview.git sasview git clone https://github.com/Sasview/sasmodels.git sasmodels git clone https://github.com/bumps/bumps.git bumps
- To get a SPECIFIC RELEASE VERSION from source control go to https://github.com/SasView/sasview/releases and download the required zip or tar.gz file. Unzip/untar it to the source code folder.

5.4 Building and Installing

- To build the code use ‘python setup.py build’
- To build the documentation use’python setup.py docs’

5.5 Running SasView

- use ‘python run.py’; this runs from the source directories, so you don’t have to rebuild every time you make a change, unless you are changing the C model files.
- if using Conda the above command will also build SasView, but you must issue ‘activate sasview’ first.

CHAPTER

SIX

KNOWN ISSUES

A full list of known bugs and feature requests by release version that users may wish to be aware of can be viewed at http://trac.sasview.org/report/3

6.1 4.2.0 - All systems

The refactoring of the plugin model architecture means that some issues may be encountered if Save Project/Analysis files using plugin models created in earlier versions of SasView are loaded in version 4.2.0.

For example:

- on loading an old project file an error window appears with the error *This model state has missing or outdated information* or *dictionary changed size during iteration*.
  - if this occurs, try restarting SasView and reloading the project.
- on loading an old project file all the FitPages and Graphs appear, but only the SasView default model parameters appear in the FitPages.
  - this has happened because plugin model parameter names have changed. There are two possible workarounds:
  - Install the version of SasView that the project was created in, recreate the plugin in that version, then run 4.2.0 and re-load the project. All being well, 4.2.0 will still compile the old plugin.
  - If 4.2.0 cannot compile the old plugin, the more tedious solution is to use a text editor to do global search & replace operations to change all the parameter names in the project file by hand. The quickest way to see the *existing* parameter names is simply to scroll to the bottom of the project file. To see what the *new* parameter names should be, simply create the equivalent plugin in SasView 4.2.0. In most instances, what was *p1_parameter* will become *A_parameter*, *p2_parameter* will become *B_parameter*, and so on.

6.2 4.1.x- All systems

The conversion to sasmodels infrastructure is ongoing and should be completed in the next release. In the meantime this leads to a few known issues:

- The way that orientation is defined is being refactored to address long standing issues and comments. In release 4.1 however only models with symmetry (e.g. a=b) have been converted to the new definitions. The rest (a <> b <> c - e.g. parellelepiped) maintain the same definition as before and will be converted in 4.2. Note that orientational distribution also makes much more sense in the new framework. The documentation should indicate which definition is being used for a given model.
- The infrastructure currently handles internal conversion of old style models so that user created models in previous versions should continue to work for now. At some point in the future such support will go away. Everyone is encouraged to convert to the new structure which should be relatively straight forward and provides a number of benefits.

- In that vein, the distributed models and those generated by the new plugin model editor are in the new format, however those generated by sum|multiply models are the old style sum|multiply models. This should also disappear in the near future
- The on the fly discovery of plugin models and changes thereto behave inconsistently. If a change to a plugin model does not seem to register, the Load Plugin Models (under fitting -> Plugin Model Operations) can be used. However, after calling Load Plugin Models, the active plugin will no longer be loaded (even though the GUI looks like it is) unless it is a sum|multiply model which works properly. All others will need to be recalled from the model dropdown menu to reload the model into the calculation engine. While it might be annoying it does not appear to prevent SasView from working..
- The model code and documentation review is ongoing. At this time the core shell parellelepiped is known to have the C shell effectively fixed at 0 (noted in documentation) while the triaxial ellipsoid does not seem to reproduce the limit of the oblate or prolate ellipsoid. If errors are found and corrected, corrected versions will be uploaded to the marketplace.
- (Added after Release 4.2.0) The scale parameter reported from the spinodal model is the square root of the true value.

6.3 3.1- All systems

- The documentation window may take a few seconds to load the first time it is called. Also, an internet connection is required before equations will render properly. Until then they will show in their original TeX format.
- If the documentation window remains stubbornly blank, try installing a different browser and set that as your default browser. Issues have been noted with Internet Explorer 11.
- Check for Updates may fail (with the status bar message ' Cannot connect to the application server') if your internet connection uses a proxy server. Tested resolutions for this are described on the website FAQ.
- The copy and paste functions (^C, ^V) in the batch mode results grid require two clicks: one to select the cell and a second to select the contents of the cell.
- The tutorial has not yet been updated and is somewhat out of date
- Very old computers may struggle to run the 3.x and later releases
- Polydispersity on multiple parameters included in a simultaneous/ constrained fit will likely not be correct
- Constrained/simultaneous fit page does not have a stop button
- Constrained/simultaneous fit do not accept min/max limits
- Save project does not store the state of all the windows
- Loading projects can be very slow
- Save Project only works once a data set has been associated with a model. Error is reported on status bar.
- There is a numerical precision problem with the multishell model when the inner radius gets large enough (ticket #288)
- The angular distribution angles are not clearly defined and may in some cases lead to incorrect calculations(ticket #332)

6.4 3.1 - Windows

- If installed to same directory as old version without first removing the old version, the old desktop icon will remain but point to the new exe version. Likewise all the start menu folders and items will have the old name even though pointing to the new version. Usually safest to uninstall old version prior to installing new version anyway.

6.5 3.1 - MAC

- Application normally starts up hidden. Click icon in Dock to view/use application.
- Multiprocessing does not currently work on MAC OS

6.6 3.1 - Linux

- Not well tested

CHAPTER

SEVEN

SASVIEW WEBSITE

http://www.sasview.org

This main project site is the gateway to all information about the sasview project. It includes information about the project, a FAQ page and links to all developer and user information, tools and resources.

CHAPTER

EIGHT

FREQUENTLY ASKED QUESTIONS

http://www.sasview.org/faq.html

CHAPTER

NINE

INSTALLER DOWNLOAD WEBSITE

Latest release Version https://github.com/SasView/sasview/releases

Latest developer builds https://jenkins.esss.dk/sasview/view/Master-Builds/

BIBLIOGRAPHY

[Levenberg1944] Levenberg, K. *Quarterly Journal of Applied Mathmatics* 1944, II (2), 164–168.

[Marquardt1963] Marquardt, D. W. *Journal of the Society for Industrial and Applied Mathematics* 1963, 11 (2), 431–441. DOI: 10.1137/0111030

[Nelder1965] Nelder, J. A.; Mead, R. *The Computer Journal* 1965, 7 (4), 308–313. DOI: 10.1093/comjnl/7.4.308

[Press1992] Press, W. H.; Flannery, B. P.; Teukolsky, S. A.; Vetterling, W. T. In *Numerical Recipes in C: The Art of Scientific Computing, Second Edition*; Cambridge University Press: Cambridge; New York, 1992; pp 408–412.

[Dennis1987] Dennis, J. E.; Schnabel, R. B. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*; Society for Industrial and Applied Mathematics: Philadelphia, 1987.

[Storn1997] Storn, R.; Price, K. *Journal of Global Optimization* 1997, 11 (4), 341–359. DOI: 10.1023/A:1008202821328

[Vrugt2009] Vrugt, J. A.; Ter Braak, C. J. F.; Diks, C. G. H.; Robinson, B. A.; Hyman, J. M.; Higdon, D. *International Journal of Nonlinear Sciences and Numerical Simulation* 2009, 10 (3), 273–290. DOI: 10.1515/IJNSNS.2009.10.3.273

[Kramer2010] Kramer, A.; Hasenauer, J.; Allgower, F.; Radde, N. *In 2010 IEEE International Conference on Control Applications (CCA)* 2010; pp 493–498. DOI: 10.1109/CCA.2010.5611198

[JCGM2008] JCGM. *Evaluation of measurement data — Supplement 1 to the "Guide to the expression of uncertainty in measurement" — Propagation of distributions using a Monte Carlo method*; Joint Committee for Guides in Metrology, JCGM 101:2008; Geneva, Switzerland, 2008; p 90. http://www.bipm.org/utils/common/documents/jcgm/JCGM_101_2008_E.pdf

[Kennedy1995] Kennedy, J.; Eberhart, R. Particle Swarm Optimization *Proceedings of IEEE International Conference on Neural Networks. IV.* 1995; pp 1942–1948. DOI: 10.1109/ICNN.1995.48896

[Sahin2013] Sahin, I. *An International Journal of Optimization and Control: Theories & Applications (IJOCTA)* 2013, 3 (2), 111–119.

[Swendsen1986] Swendsen, R. H.; Wang J. S. Replica Monte Carlo simulation of spin glasses *Physical Review Letters* 1986, 57, 2607-2609

PYTHON MODULE INDEX

INDEX